



TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE ACAPULCO

TESIS DE MAESTRÍA EN SISTEMAS COMPUTACIONALES

**Tekax - Aplicación móvil para el control de cultivos
hidropónicos utilizando IoT**

Presentada por

Uziel Trujillo Colón

Ingeniero en Sistemas Computacionales por el Instituto Tecnológico de
Acapulco

Como requisito para obtener el grado en:
Maestría en Sistemas Computacionales

Director de tesis:

M.C. José Francisco Gazga Portillo

Co-Director de tesis:

Dr. Mario Hernández Hernández

Tutor:

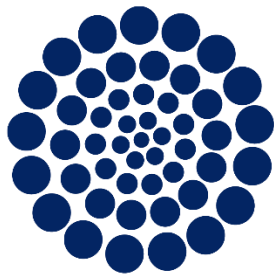
M.T.I. Rafael Hernández Reyna

Acapulco, Guerrero; noviembre 2020

El presente trabajo de tesis fue desarrollado en la División de Estudios de Posgrado e Investigación del Instituto Tecnológico de Acapulco, en el programa de Maestría en Sistemas Computacionales registrado en el Programa Nacional de Posgrado de Calidad (PNPC – CONACyT).

Con domicilio para recibir y oír notificaciones en Av. Instituto Tecnológico de Acapulco S/N, Crucero de Cayaco, Acapulco de Juárez, Guerrero, México. C. P. 39905.

Becario Uziel Trujillo Colón
CVU 421452
No. Apoyo 712851
Grado Maestría



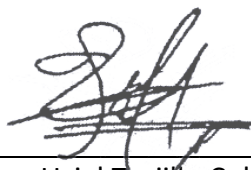
CONACYT
Consejo Nacional de Ciencia y Tecnología

Descargo de responsabilidad Institucional

El que suscribe, declara que el presente documento titulado “Tekax – Aplicación móvil para el control de cultivos hidropónicos utilizando IoT” es un trabajo propio y original, el cual, no ha sido utilizado anteriormente en institución alguna para propósitos de evaluación, publicación y/u obtención de algún grado académico.

Además, se han recogido todas las fuentes de información utilizadas, las cuales han sido citadas en la sección de referencias bibliográficas de este trabajo.

Acapulco de Juárez, Guerrero; a 12 de diciembre de 2020.



Ing. Uziel Trujillo Colón

Carta de cesión de derechos de autor

El que suscribe: Uziel Trujillo Colón, autor del trabajo escrito de evaluación profesional en la opción de Tesis Profesional de Maestría con el título “Tekax – Aplicación móvil para el control de cultivos hidropónicos utilizando IoT”, por medio de la presente con fundamento en lo dispuesto en los artículos 5, 18, 24, 25, 27, 30, 32 y 148 de la Ley Federal de Derechos de Autor, así como los numerales 2.15.5 de los lineamientos para la Operación de los Estudios de Posgrado; manifiesto mi autoría y originalidad de la obra mencionada que se presentó en la División de Estudios de Posgrado e Investigación, para ser evaluada con el fin de obtener el Título Profesional de Maestro en Sistemas Computacionales.

Así mismo expreso mi conformidad de ceder los derechos de reproducción, difusión y circulación de esta obra, en forma NO EXCLUSIVA, al Tecnológico Nacional de México campus Acapulco; se podrá realizar a nivel nacional e internacional, de manera parcial o total a través de cualquier medio de información que sea susceptible para ello, en una o varias ocasiones, así como en cualquier soporte documental, todo ello siempre y cuando sus fines sean académicos, humanísticos, tecnológicos, históricos, artísticos, sociales, científicos u otra manifestación de la cultura.

Entendiendo que dicha cesión no genera obligación alguna para el Tecnológico Nacional de México campus Acapulco y que podrá o no ejercer los derechos cedidos. Por lo que el autor da su consentimiento para la publicación de su trabajo escrito de evaluación profesional.

Se firma presente en la ciudad de Acapulco de Juárez, estado de Guerrero a los 12 días del mes de diciembre de 2020.



Ing. Uziel Trujillo Colón

Agradecimientos

A mi esposa Janet Virginia Carrillo Castro por haberme acompañado en todo este proceso, por todas las noches de desvelo, las veces que me apoyó incondicionalmente con tiempo, espacio y toda la paciencia brindada, por creer en mí y respetar cada una de las decisiones que fui tomando en el transcurso de este viaje.

A mi familia y amigos que siempre se interesaron en lo que hacía, proporcionando palabras de aliento y haciendo críticas constructivas respecto al proyecto.

Al Consejo Nacional de Ciencia y Tecnología por haberme apoyado económicamente durante la duración del proyecto, permitiendo avanzar en desarrollo de mis habilidades y mejorar mis conocimientos.

A mi director de tesis el M.C. José Francisco Gazga Portillo que siempre estuvo ahí para darme palabras de apoyo y los mejores consejos, procurando siempre que cumpliera mis metas establecidas.

A todos mis profesores que me brindaron sus conocimientos y me dieron la pauta para mejorar cada día más: Dra. Miriam Martínez Arroyo, M.I.D.S. Alma Delia de Jesús Islao, Dra. Jazmín Ávila Carbajal, M.C. Jorge Carranza Gómez, Dr. José Antonio Montero Valverde, Dr. Mario Hernández Hernández, M.T.I Rafael Hernández Reyna.

Al Ing. Edgar Moguel Beltrán y Raziel Tapia Bibiano, por el tiempo, conocimiento y apoyo ofrecido en la orientación y construcción del circuito electrónico.

Al departamento de Posgrado de Investigación del Instituto Tecnológico de Acapulco por brindarme el espacio y las instalaciones necesarias para desarrollar el proyecto de forma satisfactoria.

A mis compañeros de aula, que compartimos 2 años de experiencias juntos, preocupaciones, tareas, viajes y frustraciones, pero sobre todo risas y un excelente ambiente de compañerismo.

Dedicatoria

A mi esposa Janet Virginia Carrillo Castro, por ser el motor que me impulsa a seguir adelante, siempre a mi lado, en las buenas y en las malas, mi mejor amiga, mi confidente, la persona con la que comparto mis éxitos, fracasos, locuras y frustraciones, la que siempre está ahí para mí, a ella le dedico este trabajo tan especial, que fue realizado con mucho cariño.

A mi hija Maia Trujillo Carrillo, mi inspiración, mi amor pequeñito, que me ayudó de forma activa en todos los procesos del proyecto, desde la germinación, la limpieza, la siembra, la revisión y la cosecha, espero que en algún momento leas esta tesis, gracias por todo.

A mi padre Pedro Trujillo Valverio y a mi madre Cruz Emilia Colón García, por darme los estudios y las herramientas que me ayudaron a llegar a esta etapa de la vida, pese a todas las carencias que se pudieran presentar.

A mi abuela Carlota Valverio, por estar siempre ahí, apoyando cada una de las locuras, siempre orgullosa de mí, espero nunca decepcionarla.

A mi abuelo Hipólito Trujillo que me cuida desde el cielo, por darme el mejor consejo de vida y que hasta el día de hoy sigo: "Haz siempre lo mejor que puedas, ya que, si fallas, sabrás que lo diste todo".

A mis hermanas y hermanos: Zeire, Samara, Balam, Yolotzin, Estrella, esperando ser un buen ejemplo de vida como hermano mayor que soy.

Resumen

El conocimiento y el manejo de información es una de las mayores prioridades que las personas buscan al momento de brindar un servicio de calidad, por tal motivo, gran parte del desarrollo tecnológico en la actualidad, está orientado a sistemas de captación de información. Con la llegada de internet de las cosas (IoT), es posible utilizar tecnología y conectar diferentes componentes electrónicos, permitiendo conocer y determinar información de un área específica, llevando a cabo cualquier análisis deseado.

En el presente proyecto se describe la construcción de una arquitectura conformada por 4 componentes que permitirán llevar a cabo, la automatización, administración y control de cultivos hidropónicos. Investigando, analizando e implementando cada uno de estos componentes por separado.

El primer componente es el prototipo hidropónico, en el que se realiza un análisis de las diferentes técnicas existentes y se detalla el proceso de construcción de uno.

El segundo componente es el dispositivo electrónico, utilizando sensores, actuadores, módulos y un microcontrolador, se crea un sistema que permite automatizar y monitorizar el entorno del prototipo hidropónico.

El tercer componente es la base de datos, permite almacenar la información que ha recolectado el dispositivo electrónico de forma estructurada, posibilitando su acceso de forma remota.

El cuarto componente es la aplicación móvil, se encarga de acceder a la base de datos y consultar toda la información generada por el dispositivo electrónico de forma remota, permitiendo controlar y modificar ciertos elementos del dispositivo electrónico.

Abstract

Knowledge and information management is one of the highest priorities that people seek when providing a quality service, for this reason, today, a large part of technological development is oriented to information capture systems. With the arrival of the Internet of Things (IoT), it is possible to use technology and connect different electronic components, allowing to know and determine information from a specific area, carrying out any desired analysis.

This project describes the construction of an architecture made up of 4 components that will allow the automation, administration and control of hydroponic crops to be carried out. Investigating, analyzing and implementing each of these components separately.

The first component is the hydroponic prototype, an analysis of the different existing techniques is carried out and the construction process of one is detailed.

The second component is the electronic device, using sensors, actuators, modules and a microcontroller, a system is created that allows automating and monitoring the environment of the hydroponic prototype.

The third component is the database, it allows storing the information that the electronic device has collected in a structured form, making it possible to access it remotely.

The fourth component is the mobile application, it is responsible for accessing the database and consulting all the information generated by the electronic device remotely, allowing control and modification of certain elements of the electronic device.

Tabla de contenido

Descargo de responsabilidad Institucional	ii
Carta de cesión de derechos de autor.....	iii
Agradecimientos.....	iv
Dedicatoria.....	v
Resumen.....	vi
Abstract.....	vii
Capítulo 1 Introducción	1
1.1 Estado del arte.....	1
1.1.1 Investigación de Instituciones.....	1
1.1.2 Componentes de Hardware	2
1.1.3 Aplicaciones de Software.....	5
1.1.4 Componentes de Hardware junto con aplicaciones de Software.....	6
1.1.5 Aplicaciones Comerciales.....	8
1.2 Planteamiento del problema	10
1.3 Objetivos	13
1.3.1 Objetivo General	13
1.3.2 Objetivos Específicos	13
1.4 Hipótesis.....	14
1.5 Justificación	14
1.6 Alcances y limitaciones.....	15
1.7 Tabla comparativa de trabajos relacionados	16
1.8 Organización de la tesis.....	17
Capítulo 2 Marco Teórico	19
2.1 Prototipo hidropónico (Componente 1)	19
2.1.1 Técnicas Hidropónicas	19
2.1.2 Localización e instalación de una huerta hidropónica.	23
2.1.3 Recipientes y Contenedores	24
2.1.4 Sustratos o medios de cultivo	25
2.1.5 Semillas	25
2.1.6 Aireación	26
2.1.7 Ciclo de vida del proceso hidropónico.....	26
2.1.8 Otras consideraciones	26
2.2 Automatización hidropónica (Componente 2)	27
2.2.1 Microcontroladores	27
2.2.2 Módulos	30
2.2.3 Sensores	34
2.2.4 Actuadores	36

2.3 Base de datos (Componente 3)	38
2.3.1 Motor (Back-End)	38
2.3.2 Servidor web	38
2.3.3 Base de datos	39
2.4 Aplicación móvil (Componente 4)	40
2.4.1 Aplicaciones de software	41
2.4.2 Paradigmas en el desarrollo de aplicaciones	44
2.4.3 Patrones de diseño	47
2.4.4 Capa de presentación (Front-End)	49
2.4.5 Experiencia de Usuario (UX).....	49
2.4.6 Interface de Usuario (UI)	50
2.4.7 Aplicaciones Móviles	51
2.4.8 Marco de desarrollo Flutter	52
Capítulo 3 Análisis y diseño	56
3.1 Prototipo hidropónico (Componente 1)	57
3.1.1 Estructura de soporte hidropónico	57
3.1.2 Cama de cultivo	58
3.1.3 Contenedor y sistema de drenaje del prototipo hidropónico	59
3.1.4 Prototipo hidropónico propuesto	60
3.2 Automatización hidropónica (Componente 2)	60
3.2.1 NodeMCU y el sensor AM2302	61
3.2.2 NodeMCU y el módulo RTC AT24C32.....	62
3.2.3 NodeMCU y el relevador.....	63
3.2.4 NodeMCU y el sensor HC-SR04	64
3.2.5 NodeMCU y el módulo pH-4502C	65
3.2.6 NodeMCU y Firebase	68
3.2.7 Diagrama de flujo del programa dentro del módulo NodeMCU	69
3.3 Base de datos (Componente 3)	71
3.4 Aplicación móvil (Componente 4)	72
3.4.1 Diseño de interfaz y experiencia de usuario	73
3.4.2 Estructura superior de widgets	75
3.4.3 Nuevo usuario	76
3.4.4 Autenticación	77
3.4.5 Patrón de diseño BloC.....	80
3.4.6 Emparejar dispositivo	81
3.4.7 Navegación.....	82
3.4.8 Pantalla de inicio	83
3.4.9 Nuevo Cultivo	85
3.4.10 Información de los sensores	86
3.4.11 Configuraciones	87
3.4.12 Históricos.....	89
Capítulo 4 Implementación	92
4.1 Prototipo hidropónico (Componente 1)	92
4.2 Automatización hidropónica (Componente 2)	93
4.2.1 Configuración Arduino IDE.....	93
4.2.2 Configuración NodeMCU	93
4.2.3 Implementando la lectura de la humedad y temperatura	94

4.2.4 Implementando el temporizador	95
4.2.5 Implementando el relevador	95
4.2.6 Implementando el sensor ultrasónico	96
4.2.7 Implementando el medidor de pH	97
4.2.8 Lectura y escritura de datos en Firebase	98
4.3 Base de datos.....	100
4.3.1 Dispositivos de hardware disponibles	100
4.3.2 Usuario vinculado	101
4.3.3 Cultivo actual.....	101
4.3.4 Listado de cultivos por dispositivo	102
4.3.5 Valor actual de los sensores.....	102
4.3.6 Estado de la bomba de agua	103
4.3.7 Histórico de cultivos	104
4.3.8 Usuario autenticado	105
4.4 Aplicación móvil.....	106
4.4.1 Instalación y configuración de Flutter	106
4.4.2 Estructura superior de widgets	108
4.4.3 Nuevo usuario	109
4.4.4 Autenticación	112
4.4.5 Patrón de diseño	118
4.4.6 Emparejar dispositivo	118
4.4.7 Navegación.....	124
4.4.8 Pantalla de inicio	125
4.4.9 Nuevo cultivo	127
4.4.10 Información de los sensores	132
4.4.11 Configuraciones	137
4.4.12 Históricos.....	141
Capítulo 5 Resultados	148
5.1 Automatización hidropónica.....	148
5.2 Aplicación móvil.....	149
5.2.1 Autenticación	150
5.2.2 Nuevo usuario	151
5.2.3 Emparejar dispositivo	151
5.2.4 Pantalla de inicio	152
5.2.5 Nuevo cultivo	153
5.2.6 Información de sensores.....	153
5.2.7 Configuraciones	154
5.2.8 Históricos	155
5.3 Prototipo hidropónico.....	156
5.3.1 Lechuga orejona (Experimento 1)	156
5.3.2 Cilantro y maracuyá (Experimento 2).....	157
Capítulo 6 Conclusiones	160
6.1 Trabajo futuro.....	160
6.1.1 Prototipo hidropónico (Componente 1)	161
6.1.2 Automatización hidropónica (Componente 2)	161
6.1.3 Base de datos (Componente 3)	161
6.1.4 Aplicación móvil (Componente 4)	161

6.2 Participaciones y aportaciones	162
<i>Fuentes de Información</i>	<i>163</i>

Índice de figuras

Figura 1-1 Estimación de población mundial por las naciones unidas.....	10
Figura 2-1 Sistema de cultura de agua profunda.	20
Figura 2-2 Sistema de flujo y reflujo.....	20
Figura 2-3 Sistema Aeropónico.....	21
Figura 2-4 Sistema Acuapónico.	21
Figura 2-5 Sistema por goteo.	22
Figura 2-6 Técnica de película de nutrientes.....	23
Figura 2-7 Riego hidropónico manual.	23
Figura 2-8 Proceso de cosecha en cultivos hidropónicos.....	26
Figura 2-9 Pines del microcontrolador ESP8266.	29
Figura 2-10. Diagrama de pines del módulo NodeMCU.....	31
Figura 2-11. Diagrama de conexión entre el módulo AT24C32 y el MCU.....	32
Figura 2-12. Diagrama de conexión entre el módulo pH-4502C y el MCU.	33
Figura 2-13. Diagrama de conexión entre el sensor AM2302 y el MCU	35
Figura 2-14. Diagrama de conexión entre el sensor HC-SR04 y el MCU.	36
Figura 2-15. Diagrama de conexión entre el relevador y el MCU.	37
Figura 2-16. Arquitectura BLoC.	48
Figura 2-17. Arquitectura de Flutter.....	52
Figura 2-18. Clasificación de Widgets en Flutter.	53
Figura 2-19. Uso de flujos con el patrón de diseño BLoC.....	54
Figura 3-1. Arquitectura de componentes del proyecto Tekax.....	56
Figura 3-2. Estructura de soporte hidropónico.	57
Figura 3-3. Cama de cultivo	58
Figura 3-4. Contenedor y sistema de drenaje del prototipo hidropónico.	59
Figura 3-5. Prototipo hidropónico propuesto.	60
Figura 3-6. Diagrama de conexión entre módulos, sensores y microcontrolador del componente 2.....	61
Figura 3-7. Calibración del módulo ph-4502C.	66
Figura 3-8. Función lineal para el cálculo de pH.....	67
Figura 3-9. Diagrama de flujo del programa en el módulo NodeMCU.	70
Figura 3-10. Esquema de la base de datos.	71
Figura 3-11. Diagrama de casos de uso de la aplicación móvil.	73
Figura 3-12. Paleta de colores para el diseño de la aplicación.....	74
Figura 3-13. Logotipo del proyecto Tekax.	74
Figura 3-14. Diseño de interfaz y experiencia de usuario.	75
Figura 3-15. Conjunto de widgets que integran la estructura superior.....	76
Figura 3-16. Diagrama de secuencia que permite verificar el estado del usuario en la aplicación móvil.	76
Figura 3-17 Estructura de widgets para la creación de nuevo usuario.	77
Figura 3-18 Diagrama de secuencia para la creación de nuevo usuario.....	77
Figura 3-19. Estructura de widgets de la autenticación.	78
Figura 3-20 Diagrama de secuencia de la autenticación.....	79

Figura 3-21. Estructura de carpetas utilizada para implementar el patrón de diseño BLoC.	80
Figura 3-22. Estructura de widgets que se utiliza para emparejar el componente 2.	81
Figura 3-23. Diagrama de secuencia para emparejar el componente 2 con la aplicación móvil.	81
Figura 3-24. Estructura de widgets para implementar la navegación en la aplicación.	83
Figura 3-25. Estructura de widgets de la pantalla de inicio.	84
Figura 3-26. Diagrama de secuencia de la pantalla de inicio.	84
Figura 3-27. Estructura de widgets para la creación de nuevo cultivo.	85
Figura 3-28 Diagrama de secuencia para la creación de nuevos cultivos	85
Figura 3-29. Estructura de widgets para mostrar la información de los sensores.....	86
Figura 3-30 Diagrama de secuencia para mostrar la información de los sensores	87
Figura 3-31. Arquitectura de widgets para las configuraciones.....	88
Figura 3-32 Diagrama de secuencia para las configuraciones.	88
Figura 3-33. Estructura de widgets para la construcción de los históricos.....	90
Figura 3-34. Diagrama de secuencia para mostrar los históricos.	90
Figura 4-1 Implementación del prototipo hidropónico.....	92
Figura 4-2. Dispositivos de hardware (componente 2) disponibles.	101
Figura 4-3. Vinculación entre usuario y dispositivo en la base de datos.	101
Figura 4-4. Almacenamiento del cultivo actual en la base de datos.....	102
Figura 4-5. Listado de cultivos del usuario.	102
Figura 4-6. Valor almacenado en la BD de cada sensor del componente 2.	103
Figura 4-7. Obtener el estado de la bomba.....	104
Figura 4-8. Histórico de cultivos en la base de datos.	105
Figura 4-9. Almacenamiento de usuarios en la base de datos.....	106
Figura 4-10. Verificando la instalación de Flutter.	108
Figura 4-11 Estructura de carpetas del proyecto Tekax.....	118
Figura 4-12 Asignación de archivos dentro de las carpetas del patrón BLoC.	118
Figura 5-1 Circuito electrónico desarrollado.	148
Figura 5-2 Elementos que pueden conectarse y desconectarse del componente 2.	149
Figura 5-3 Caja MDF creada para proteger el circuito electrónico.	149
Figura 5-4 Autenticación en la aplicación móvil.....	150
Figura 5-5 Creación de nuevo usuario en la aplicación móvil.	151
Figura 5-6 Emparejamiento de usuario activo con el componente 2 desde la aplicación móvil.	152
Figura 5-7 Pantalla de inicio de la aplicación móvil.....	152
Figura 5-8 Creación de nuevo cultivo desde aplicación móvil.	153
Figura 5-9 Información del estado actual de los sensores desde la aplicación móvil.....	154
Figura 5-10 Configuraciones del prototipo hidropónico desde la aplicación móvil.....	154
Figura 5-11 Históricos en la aplicación móvil.	155
Figura 5-12 Resultados de lechuga orejona en el prototipo hidropónico.	157
Figura 5-13 Disposición de cilantro y maracuyá en el prototipo hidropónico.	158
Figura 5-14 Resultados de cosechar cilantro en el prototipo hidropónico.....	158
Figura 5-15 Resultados de la cosecha de maracuyá en el prototipo hidropónico.....	159

Índice de Tablas

Tabla 1-1. Comparativa de los trabajos relacionados	16
Tabla 2-1. Pines de conexión del ESP8266.	30
Tabla 2-2. Pines de conexión del módulo AT24C32.	33
Tabla 2-3. Pines de conexión del módulo analógico pH-4502C	34
Tabla 2-4. Pines de conexión del sensor AM2302.....	35
Tabla 2-5. Pines de conexión del sensor HC-SR04.....	36
Tabla 2-6. Pines de conexión del relevador.....	37
Tabla 3-1. Componentes a utilizar para automatización y control del sistema hidropónico.	61
Tabla 4-1. Requerimientos para la instalación de Flutter.	107
Tabla 5-1 Evolución lechuga orejona utilizando el prototipo hidropónico y la aplicación móvil.	156
Tabla 5-2 Evolución del cilantro y maracuyá en el prototipo hidropónico, utilizando la aplicación móvil.	157

Índice de Algoritmos

Algoritmo 3-1 Módulo NodeMCU y el sensor AM2302.	62
Algoritmo 3-2. Interacción entre NodeMCU y el módulo AT24C32.	63
Algoritmo 3-3. Automatización de bomba de agua utilizando NodeMCU y el relevador....	63
Algoritmo 3-4. Obtener distancia entre el sensor HC-SR04 y la solución nutritiva, utilizando NodeMCU.	65
Algoritmo 3-5. NodeMCU y el módulo ph-4502C.	68
Algoritmo 3-6. Integración de NodeMCU y Firebase.	69
Algoritmo 4-1. Almacenar información de históricos.	104

CAPÍTULO 1

Introducción

Capítulo 1 Introducción

1.1 Estado del arte

Actualmente, la tecnología pretende brindar soluciones a problemas de sostenibilidad, abarcando diferentes áreas. Con el crecimiento acelerado de la población, la agricultura tradicional puede no ser suficiente para las necesidades alimentarias, el desarrollo de sistemas inteligentes y técnicas que coadyuven a gestionar mejor los recursos naturales, son alternativas que se deben buscar para disminuir los problemas que se avecinan.

En el presente capítulo se analizan los trabajos relacionados con agricultura sostenible y tecnología, utilizando una clasificación por área de uso: instituciones, aplicaciones de hardware, aplicaciones de software, aplicaciones de hardware y software, finalizando con aplicaciones comerciales. Es posible analizar las fortalezas, debilidades, el mercado al que van dirigidos los proyectos y su facilidad de implementación, cada uno de estos elementos, es tomado como referencia para la construcción de un nuevo proyecto, que será desarrollado en el presente trabajo de tesis.

1.1.1 Investigación de Instituciones

Las instituciones forman un grupo importante en la investigación científica y tecnológica del país, realizando aportes a diversas áreas de interés, una de ellas, es el sector agrícola, dentro del cual, nos enfocaremos específicamente en los proyectos que tienen relación con los cultivos hidropónicos.

Laboratorio Ecológico de Hidroponía

Laboratorio perteneciente al Instituto Politécnico Nacional (IPN, 2018) dedicado a realizar diferentes tipos de investigaciones como son: agrobiotecnología complementada con hidroponía; investigación de técnicas de cultivo de alta eficiencia y acciones de cuidado al medio ambiente, uso de productos orgánicos para la prevención y corrección de problemas fitosanitarios, nutrición de las plantas. Dentro de las actividades realizadas en este laboratorio se encuentran:

- Germinación de semillas.
- Preparación de soluciones nutritivas.
- Control fitosanitario de productos.
- Formulación de soluciones nutritivas.
- Cultivo hidropónico de especies de interés económico.
- Deshidratación solar de productos vegetales.

Aplicaciones de aprendizaje automático para datos agrícolas que mejoren la granja inteligente

Trabajo enfocado en administrar una gran cantidad de información y datos heterogéneos provenientes de repositorios que recopilan valores físicos, biológicos y sensoriales de las plantas (Balducci et al., 2018).

Con un volumen importante de información, obtenido de diferentes repositorios, es posible utilizar técnicas de aprendizaje automático (Redes neuronales, modelos de regresión lineal y polinomial, KNN¹), permitiendo sugerir, en qué dirección emplear esfuerzos, mejorando la calidad de las cosechas.

El estudio contempla 3 diferentes fuentes de información, que son consideradas por múltiples herramientas para el aprendizaje automático, las cuales son: NIS (National Institute of Statistics), Istat (Istituto Nazionale di Statistica) y sensores IoT².

Detección y diagnóstico de fallas en hidroponía utilizando herramientas computacionales inteligentes de profundidad

Las herramientas computacionales inteligentes como redes neuronales y algoritmos genéticos, son utilizadas para desarrollar un sistema de detección y diagnóstico en tiempo real de fallas mecánicas en sensores (Ferentinos & Albright, 2003).

Se crea una nueva técnica para el diseño de redes neuronales y la parametrización del entrenamiento, basada en el método de optimización heurística de los algoritmos genéticos. Las fallas de un sensor o un actuador son detectadas realizando un diagnóstico, previendo con tiempo suficiente su detección y funcionando como un supervisor confiable automatizado.

El algoritmo genético desarrollado puede aplicarse con éxito a un problema combinatorio, como decidir la mejor arquitectura de red neuronal, funciones de activación y algoritmo de entrenamiento para un modelo específico.

1.1.2 Componentes de Hardware

El desarrollo de un componente electrónico es un elemento que debe ser implementado, constituye una solución de forma física para la problemática que se desea resolver (robots, tubos, circuitos, sensores, entre otros), dentro de esta sección se mostrarán los componentes de hardware³ encontrados que han sido utilizados en

¹ Por sus siglas en inglés K-Nearest Neighbor, es un algoritmo basado en instancia de tipo supervisado utilizando en las máquinas de aprendizaje.

² Por sus siglas en inglés Internet of Things, es un concepto que se refiere a una interconexión digital de objetos cotidianos con internet.

³ Conjunto de elementos físicos o materiales que constituyen una computadora o un sistema informático

investigaciones previas a la propuesta del presente trabajo de tesis, para el mantenimiento y automatización de cultivos hidropónicos.

Medidor de pH, EC y temperatura

Un grupo de estudiantes del I.T.C Instituto Tecnológico de Celaya, proponen un instrumento que permite monitorizar los niveles de pH⁴, EC⁵ en conjunto con un sensor de temperatura (*DS18B20*), dicho artefacto, sirve como base para medir los valores de la solución nutritiva en cultivos hidropónicos (Guadarrama et al., 2016).

La monitorización es realizada de forma inalámbrica utilizando un microcontrolador *ATMega328p* conectado a los sensores de: pH, EC y temperatura. Posteriormente se envía la información recolectada por los sensores utilizando el protocolo de comunicación Bluetooth⁶.

Para el pH se utiliza un sensor con un electrodo de vidrio y un electrodo de AgCl⁷. El primer electrodo se utiliza como indicador, mientras que el segundo electrodo se emplea como referencia, la medición del pH se realiza por medio de la fuerza electromotriz generada entre ambos electrodos.

Para obtener la información de la conductividad eléctrica (EC), se utiliza un oscilador que genera un tren de pulsos como señal de excitación a la sonda EC, de manera que, con los cambios de su impedancia, permite que la amplitud de la señal de salida en la sonda varíe, esta variación reflejará los valores de la conductividad.

Módulo sensor de alta precisión y bajo costo para sistema de cultivo hidropónico

Trabajo que desarrolla un sensor de alta precisión para cultivos hidropónicos, enfocado en medir: nivel de agua, temperatura del agua y conductividad eléctrica (EC) en la solución nutritiva (Nishimura et al., 2016). Utiliza un solo dispositivo electrónico equipado con un cable plano y electrodos de bajo costo.

El módulo cuenta con dos osciladores, el nivel de agua y el valor de concentración se convierten en frecuencias de las señales del oscilador. Los cambios de capacitancia y resistencia son detectadas y enviadas a través de un cable plano hacia él, finalmente, las

⁴ Es una medida de acidez o alcalinidad de una disolución, indica la concentración de iones de hidrógeno presentes en determinadas disoluciones.

⁵ Es la medida de la capacidad de un material o sustancia para dejar pasar la corriente eléctrica a través de él.

⁶ Es una especificación industrial para Redes Inalámbricas de Área Personal (WPAN) que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda ISM de los 2.4 GHz.

⁷ El cloruro de plata es un compuesto químico de fórmula AgCl. Este sólido cristalino es bien conocido por su baja solubilidad en agua.

frecuencias son convertidas en valores que miden el nivel del agua y el valor de la conductividad eléctrica (EC).

Sistema de hidropónico automatizado basado en Internet de las cosas.

Consiste en la creación de un entorno automatizado utilizando como base, una placa *ARDUINO Mega 2560*, conectando diferentes sensores: temperatura y humedad (*DHT11*), Temporizador (*DS3234*), pH, EC, luces LED, permite monitorizar ciertos aspectos que necesitan controlarse en los cultivos hidropónicos (Francis et al., 2018). La información recolectada por el sistema, es enviada a través de internet utilizando el módulo *ESP8266*, que almacena la información de cada sensor en una base de datos conectada a un servidor remoto.

Los sensores son controlados mediante un menú dentro de una pantalla *LCD*⁸ la cual se conecta a la placa *ARDUINO Mega 2560*, permitiendo cambiar los valores de los parámetros básicos en los sensores.

Cultivo hidropónico de espinaca mediante técnica NFT e Invernadero para el control de variables ambientales

Proyecto desarrollado por estudiantes de la Universidad Ricardo Palma, Perú (Chiara et al., 2016), consiste en la elaboración de un sistema hidropónico dentro de un invernadero utilizando la técnica *NFT*⁹. El enfoque principal es brindar el control de siguientes elementos: 1) La temperatura (*LM35*). 2) La Humedad (*DHT11*). 3) El pH.

La automatización del entorno se lleva a cabo mediante un temporizador digital, una bomba de agua es conectada a este dispositivo, la interacción entre ambos componentes permite al sistema, encender o apagar la bomba de agua en intervalos de tiempo específicos, como consecuencia, la circulación constante de los nutrientes fluye a través del sistema hidropónico.

Hommons: Sistema hidropónico de administración y monitoreo para una granja NTF basada en Internet de las cosas utilizando tecnología web.

Trabajo realizado por (Crisnapati et al., 2017), Se construye un sistema para monitorizar y recolectar de información de cultivos hidropónicos haciendo uso de la técnica *NTF*. Sistemáticamente se evalúan y analizan los parámetros generados por este sistema: Temperatura del agua, nivel de agua, pH y EC. El sistema previamente mencionado debe ser operado de forma presencial.

⁸ Es una pantalla delgada y plana formada por un número de píxeles en color o monocromos colocados delante de una fuente de luz o reflectora.

⁹ Nutrient Film Technique, es una técnica hidropónica donde una corriente de agua de poca profundidad que contiene todos los nutrientes disueltos para el crecimiento vegetal, se recircula más allá de las raíces desnudas de las plantas en un caudal hermético.

El prototipo utiliza celdas solares para cargar y obtener el voltaje necesario que alimentará una batería, un inversor de corriente suministra el voltaje a los aparatos eléctricos del sistema hidropónico: Bomba de agua, solución nutritiva, LED, microcontrolador *ARDUINO UNO*, microcomputador *Raspberry Pi*¹⁰ modelo B y los sensores.

Es posible acceder a la administración del cultivo hidropónico que ha sido conectado a la red a través de una página web utilizando el navegador e ingresando la dirección IP¹¹ del servidor, sin embargo, su funcionamiento solo es local. Los sensores y actuadores envían la información hacia el microcontrolador, que a su vez se conecta a un microcomputador mediante el protocolo 802.11, mejor conocido como *Wi-Fi*¹², este último recibe la información que envían los sensores y los inserta en una base de datos que se encuentra en un servidor remoto.

1.1.3 Aplicaciones de Software

Sistema para identificar la edad de la planta de cultivos hidropónicos en exteriores basado en el procesamiento de imágenes digitales

Investigadores del Departamento de Ingeniería Eléctrica en Indonesia, crean un sistema que permite identificar el crecimiento de las plantas utilizando técnicas de Inteligencia Artificial, procesando imágenes y haciendo uso de visión computacional es posible, realizar segmentación basada en la similitud de color con el modelo *HSV*¹³ (Nursyahid et al., 2018).

El método desarrollado selecciona un área de la planta, la convierte a blanco y negro, posteriormente calcula el número de píxeles y tomando esta cantidad, identifica la edad de la misma basándose en categorías establecidas con antelación.

Este trabajo fue realizado utilizando el microcomputador *Raspberry Pi 3 modelo B*, junto con un módulo de cámara que funciona con este último, el módulo toma una fotografía y la envía al microcomputador, posteriormente se utiliza un algoritmo que procesa la imagen, finalmente haciendo uso de una conexión de red *Wi-Fi* envía la información procesada a una base de datos remota, permitiendo que la información pueda ser consultada utilizando una página web.

¹⁰ Es un ordenador de placa reducida de bajo costo, que cuenta con los componentes básicos de una computadora.

¹¹ Es un número que identifica, de manera lógica y jerárquica, a una interfaz en red.

¹² Es una tecnología que permite la interconexión inalámbrica de dispositivos electrónicos. Los dispositivos habilitados con WiFi pueden conectarse entre sí o a internet a través de un punto de acceso de red inalámbrica.

¹³ Por sus siglas en inglés Hue, Saturation, Value, define un modelo de color en términos de sus componentes.

Autogrow

La empresa *Autogrow* ubicada en Estados Unidos, brinda soluciones que utilizan agro tecnología, crean una *API*¹⁴ de código abierto que posibilita la conexión entre hardware propietario y el consumo de su *API* para obtener la información de los sensores desde un software desarrollado por terceros, pudiendo ser una aplicación de escritorio, móvil o web (AUTOGROW-LAB, 2018).

La *API* es de tipo *RESTful*¹⁵ y se utiliza para interactuar con los dispositivos *Autogrow* (hardware propietario). Los componentes *Autogrow*, envían la información de los sensores hacia la *API* utilizando el software propietario *Intelligrow*, a medida que los datos comienzan a ser enviados, se puede utilizar la *API* para recuperar la información, utilizando el software que la empresa provee o desarrollando software de terceros.

El software propietario *my.autogrow*, es una solución para la visualización de datos en los controladores *MultiGrow* y *Aphaea* que utiliza la *API* de esta empresa. A través de los tableros que están integrados a los controladores previamente mencionados se puede visualizar el estado del cultivo y son capaces de mostrar: el uso de agua, escurrimiento, nutrientes, clima y humedad.

El software *Intelligrow*, es una aplicación que cuenta con un entorno de escritorio y móvil, utiliza la *API* desarrollada por la empresa y mediante una conexión a internet posibilita la administración de los siguientes elementos: temperatura, dióxido de carbono, humedad, iluminación, nutrientes y el pH del cultivo. El software funciona con diferentes sistemas operativos de escritorio (OS, Windows y Linux) y sistemas operativos móviles (iOS, Android). La aplicación permite acceder a los datos generados por los sensores, envía alertas informativas y realiza acciones para proteger y nutrir los cultivos.

1.1.4 Componentes de Hardware junto con aplicaciones de Software

Sistema hidropónico completamente automatizado para el crecimiento de plantas en interiores

Trabajo realizado por estudiantes del colegio de Ingeniería y ciencias computacionales en la Universidad del estado de California (Palande et al., 2018). El propósito del proyecto, es mejorar la utilización de los cultivos hidropónicos creando un sistema ambientalmente independiente que permita el crecimiento de plantas en interiores.

¹⁴ Application Programming Interface, es un conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

¹⁵ Es un estilo de arquitectura de software que define un conjunto de restricciones que se utilizarán para crear servicios web. Los servicios web que se ajustan al estilo arquitectónico de REST, denominados servicios web RESTful, proporcionan interoperabilidad entre sistemas informáticos en Internet.

Los componentes de hardware que se utilizan son: 1) Dos microcontroladores *ARDUINO UNO* para controlar y analizar los datos dentro de la aplicación. 2) Un microcomputador *Raspberry Pi modelo B* que ejecuta el software de automatización *Domoticz* de código abierto. Cuando el microcomputador recibe los datos de entrada, procesa la información y actualiza la base de datos del servidor. El software puede ser instalado en dispositivos móviles (iOS, Android). Debido a que la estructura del sistema hidropónico es pequeña, puede ser instalada en un espacio de oficina o casa.

Aplicando internet de las cosas para un ecosistema de cultivo hidropónico inteligente (HFE)

Investigadores de Tailandia, desarrollan un prototipo que permite controlar un sistema hidropónico mediante una aplicación móvil. Haciendo uso de internet de las cosas pueden monitorizar de forma automatizada los sensores de: humedad, la temperatura de la solución nutritiva, la temperatura del aire, la conductividad eléctrica (EC) y el pH de los cultivos procesados (Ruengittinun et al., 2017).

El microcontrolador *ARDUINO UNO* toma la información del sensor de pH y compara el valor actual contra el umbral previamente establecido desde la aplicación móvil. Si el valor no está dentro del rango permitido, una válvula solenoide será activada para liberar una sustancia, reduciendo la basicidad/alcalinidad en la solución nutritiva del agua y viceversa. La válvula solenoide ajustará el valor de pH automáticamente hasta que alcance el umbral establecido.

El control del valor EC es similar al de pH, con la diferencia de que la obtención de sus valores es realizada por otro sensor, y la sustancia liberada para regular la conductividad también es diferente.

La humedad, la temperatura de la solución nutritiva y la temperatura del ambiente, es información obtenida mediante el uso de diferentes sensores, los datos obtenidos en un tiempo determinado son enviados utilizando un microcontrolador hacia una base de datos remota.

Sistema inteligente para el control de bicarbonato en riego para agricultura de precisión hidropónica

Un equipo de investigadores de la Universidad Politécnica de Valencia (Cabra et al., 2018), presentan un sensor de pH auto calibrado capaz de detectar y ajustar los desequilibrios en los niveles de pH de la solución nutritiva utilizada en cultivos hidropónicos. El sensor está compuesto por una sonda que mide el pH y un conjunto de micro bombas que vierten secuencialmente las diferentes soluciones líquidas para mantener la calibración del sensor y las muestras de agua de los canales que contienen la solución nutritiva.

El componente desarrollado, utiliza un sensor de pH calibrado automáticamente y conectado a un nodo inalámbrico. La interconexión entre diferentes nodos da lugar a una Red de Sensores Inalámbricos (RSI) que permiten el control de un invernadero. Los sensores miden periódicamente el nivel de pH de cada soporte hidropónico en forma independiente y envían la información a una base de datos que almacena y analiza los datos para advertir a los agricultores sobre el estado de las diferentes cosechas. Posteriormente se puede acceder a la información mediante una interfaz web, ingresando una dirección IP.

Cultivo inteligente hidropónico que utiliza el Sistema Social Cibernético (CPSS) con Telegram Messenger

Investigadores del Institut Teknologi Bandung (Sisyanto et al., 2018), Indonesia, crean el Sistema Social Cibernético (CPSS), consiste en el desarrollo de un sistema de cultivo inteligente hidropónico, que puede ser monitorizado en línea a través de la plataforma *Telegram Messenger*. El software controla los siguientes elementos dentro del sistema hidropónico: la intensidad de la luz utilizando una célula fotoeléctrica *LDR*, la temperatura ambiente y la humedad (*DHT11*), EC y el pH. El prototipo hace uso de un microcontrolador utilizando *ARDUINO UNO* para la obtención de la información procesada por los sensores y la envían hacia un microcomputador *Raspberry Pi 3* que funciona como servidor web.

Finalmente crean un *BOT*¹⁶ de *Telegram* que permite monitorear sensores en línea a través de esta plataforma. La integración del sistema físico (*ARDUINO UNO*, sensores y *Raspberry Pi*) junto con el sistema social *Telegram Messenger*, proveen una comunicación entre la información que procesan los sensores y el usuario.

1.1.5 Aplicaciones Comerciales

Robots de Agricultura

La empresa estadounidense *Postscapes*, alberga una línea especializada en el desarrollo de agro tecnología para la medición, recuperación y vigilancia de cultivos, cada parte puede ser operada de forma independiente (POSTSCAPES, 2019), posee los siguientes productos:

- Drones e imágenes aéreas.
- Construcción de rociadores.
- Construcción de robots para cortar cosechas.
- Robot con autoaprendizaje (Selección de los mejores frutos).
- Navegación autónoma (Tractores con GPS).
- Robots para agricultura en interiores.
- Manejo de materiales.

¹⁶ Es un programa informático que efectúa automáticamente tareas repetitivas a través de Internet, cuya realización por parte de una persona sería imposible o muy tediosa.

Visco hydroponics

La Empresa Inglesa *Fullbloom Hydroponics*, trabaja con cultivos hidropónicos automatizados, poseen una serie de productos y contenedores que permiten construir y automatizar dichos cultivos de forma rápida y eficiente (FULLBLOOM HYDROPONICS, 2019), ofrecen los siguientes servicios:

- Tazas de cultivo.
- Bandejas de cultivo.
- Cajas crecientes.
- Contenedores de plantas para aguas profundas.
- Sistemas automatizados de trasplante.
- Sistema automatizado de plantación.
- Mecanismo de empuje automatizado.
- Sistemas automatizados de recolección.
- Lavado con flotador.

Farm.One

La empresa Farm One ubicada en Nueva York, se dedica a los cultivos verticales hidropónicos, tienen la capacidad de producir cerca de 500 hortalizas diferentes, se especializan en la venta de hierbas finas a restaurantes y en la construcción de prototipos hidropónicos a medida (FARM.ONE, 2018), sus principales fortalezas son:

- Análisis, diseño y construcción de prototipos hidropónicos.
- Recipientes de cultivo, luces, bombas, sensores, entre otras cosas, que funcionan únicamente con su estructura hidropónica.
- Software hidropónico automatizado propio que permite la visualización del estado de las cosechas, los pedidos, las entregas realizadas, las estadísticas, y permite la personalización de los umbrales en cada cultivo, todo esto en un solo punto de acceso.

Niwa-One

Trabajo realizado por la empresa Niwa One ubicada en Estados Unidos, crean un sistema hidropónico doméstico muy pequeño y autoadministrable que puede ser ubicado casi en cualquier parte de una vivienda tradicional, contando con las siguientes medidas *91cm x 49cm x 21cm* (NIWA-ONE, 2018). Dentro de este sistema se agregan los elementos necesarios para automatizar un cultivo (luces, sensores, contenedores, bombas y extractores de aire), integrando todo en un solo componente. El sistema únicamente necesita ser conectado a la corriente eléctrica, ser sincronizado con la aplicación móvil y comenzar a utilizarlo, las características principales de esta implementación son las siguientes:

- Cultivo automatizado total.
- Interfaz minimalista.
- Poco espacio necesario.
- Aplicación móvil.
- Diferentes cultivos.

1.2 Planteamiento del problema

“Los desafíos asociados con la prevención, la gestión y la resolución de conflictos inducidos por los recursos naturales bien podrían llegar a definir la paz y la seguridad global en el siglo XXI. Las tendencias globales, como los cambios demográficos, el aumento del consumo, la degradación del medio ambiente y el cambio climático, están creando presiones significativas y potencialmente insostenibles sobre la disponibilidad y la usabilidad de recursos naturales como la tierra, el agua y los ecosistemas” (NACIONES UNIDAS, 2012).

Un estudio realizado por (NACIONES UNIDAS, 2015), revela que para el año 2030 la población mundial habrá crecido un 15% respecto a la población actual, ver Figura 1-1, este incremento poblacional desmedido atrae una serie de problemáticas que deben de ser atendidas a la brevedad, entre ellas la alimentación. La gente dedicada a la agricultura tiene problemas en responder las necesidades actuales alimentarias, es por ello que mucha investigación científica está enfocada en resolver problemas de sostenibilidad.

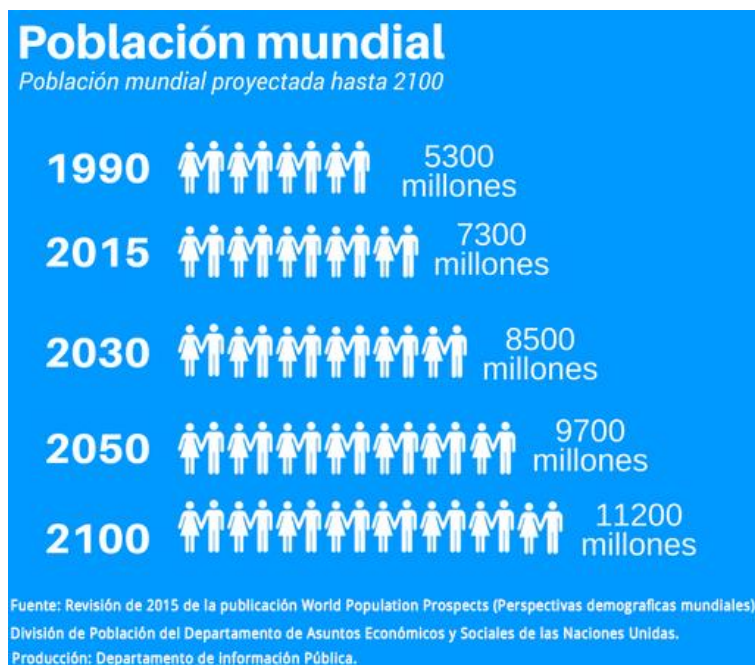


Figura 1-1 Estimación de población mundial por las naciones unidas.

El 79% de la agricultura que se practica dentro la República Mexicana es de temporal (INEGI, 2017), el otro 21% es agricultura de riego, esto hace que la producción agrícola dependa de la temporada de lluvia. Se deben buscar alternativas que permitan sembrar con la menor cantidad de agua una mayor producción de alimentos.

El clima con el que cuenta en mayor medida la región Sureste de México, es cálido subhúmedo (INEGI, 2019), su temperatura promedio se encuentra entre los 22°C a 28°C y permanecen constantes la mayor parte del año, estas condiciones, propician que las plantas cuenten con excelentes características que se reflejan en su crecimiento.

En la actualidad existen muchas empresas que fabrican productos agro-tecnológicos, dando paso a la agricultura sostenible que tiene como su principal objetivo, encontrar soluciones a las problemáticas alimentarias, conservando y preservando los recursos naturales, algunos de estos productos agro-tecnológicos son: 1) Tractores GPS que permiten arar la tierra de forma milimétrica y con el menor desperdicio de semillas mejorando así el abonado, la siembra y los tratamientos fitosanitarios. 2) Sistemas de riego guiados por telemetría para un mejor uso del agua. 3) Sistemas de recolección para la selección de los mejores productos. 4) Drones para la detección de plagas.

Lamentablemente la tecnología mencionada previamente, no es económica, por otra parte, el país tampoco manufactura este equipo, por lo que, si se desea conseguir es necesario importarlo, teniendo estos factores como limitantes, no muchos agricultores pueden pagarla. Según INEGI un estudio realizado revela que la clase social agrícola, son en su mayoría asalariados y pequeños propietarios, donde el ingreso laboral que perciben está muy por debajo de la media (Solís & Benza, 2013).

En la actualidad un agricultor promedio que vive en la región Sureste de México cosecha de la siguiente forma:

1. Espera todo el año la temporada de lluvias.
2. Prepara la tierra arando el campo manualmente o con el uso de un tractor.
3. Abona el campo aplicando fertilizantes en caso de ser necesario.
4. Siembra las semillas.
5. Si la temporada de lluvia no es buena tiene que regar la cosecha de alguna forma utilizando una bomba de agua o realizando este procedimiento manualmente.
6. Revisa el cultivo para ver si no hay algún tipo de infección en las plantas.
 - a. Si existe infección, aplicar un correctivo.
7. Recolecta la cosecha.
8. Transporta la cosecha.
9. Procesa o vende la cosecha.

Todo el proceso de siembra anteriormente mencionado en la región Sureste del país, es realizado de forma manual durante la mayor parte del año, llevarlo a cabo de esta manera provoca una serie de inconsistencias:

- No se optimiza correctamente el arado del campo.
- Al agregar en mayor o menor cantidad fertilizante, la tierra con el paso del tiempo absorbe estos químicos y pierde fertilidad.
- Al contaminar la tierra con fertilizantes las cosechas futuras son de menor calidad y existe el riesgo de tener productos contaminados por fitotoxicidad¹⁷.
- La cantidad de agua desperdiciada es mayor.
- El esfuerzo humano realizado es mayor.

Dentro de la agricultura, existe una técnica que permite realizar cultivos sin necesidad de tener suelo firme, la base para el desarrollo de esta técnica es tomada de los aztecas, que utilizaban las chinampas y los babilonios que utilizaban una técnica de distribución de agua en los Jardines Colgantes de Babilonia, con el paso del tiempo se refino este concepto y se creó una nueva técnica que recibió el nombre de hidroponía (del Griego [hidro] = “agua”, [ponos] = “labor, trabajo”), sin embargo, a pesar de que no es un descubrimiento reciente, llevar a cabo esta técnica de forma adecuada necesita una serie de procedimientos meticulosos y mediciones constantes que son controlados dentro de cada cultivo.

“El cultivo en hidroponía, es una modalidad en el manejo de plantas, que permite su cultivo sin suelo. Mediante esta técnica se producen plantas principalmente de tipo herbáceo, aprovechando sitios o áreas no convencionales, sin perder de vista las necesidades de las plantas, como luz, temperatura, agua y nutrientes. En el sistema hidropónico los elementos minerales esenciales son aportados por la solución nutritiva. El rendimiento de los cultivos hidropónicos puede duplicar o más los de los cultivos en suelo. La disponibilidad de agua y nutrientes, los niveles de radiación y temperatura del ambiente, la densidad de siembra o disposición de las plantas en el sistema hidropónico, la acción de patógenos o plagas, entre otros, incidirán fuertemente en el rendimiento del cultivo.

Hoy la hidroponía se vislumbra como una solución a la creciente disminución de las zonas agrícolas, producto de la contaminación, la desertización, el cambio climático y el crecimiento desproporcionado de las ciudades.” (Beltrano & Gimenez, 2015).

En el año 2013, la actual Secretaría de Agricultura y Desarrollo Rural (SADER) anteriormente SAGARPA, lanzó una serie de apoyos para que agricultores de diferentes estados, desarrollen proyectos basados en agricultura protegida, proveyendo material didáctico, capacitación, equipamiento y ayuda económica para iniciar proyectos hidropónicos, entre otros.

¹⁷ Es un término que se emplea para describir el grado de efecto tóxico producido por un compuesto sobre el crecimiento de las plantas.

La comunidad de “San Pedro las Playas” ubicada en el municipio de Acapulco de Juárez, Guerrero. Cuenta con agricultores que fueron beneficiados con el programa rural antes mencionado, sin embargo, al realizar una visita a las instalaciones donde está ubicada la estructura hidropónica construida con los recursos recibidos, se observó, que la instalación fue abandonada, al preguntar al propietario el motivo por el cual estaba en ese estado, contestó que: “Era muy complicado realizar mediciones, ajustes y controlar los tiempos de manera constante dentro de los cultivos hidropónicos, argumentando que debían dividir su tiempo entre alimentar al ganado, sembrar plantas de forma tradicional y supervisar el cultivo hidropónico, esto último les quitaba demasiado tiempo”.

El problema a resolver con este trabajo de tesis, es tomar de referencia los problemas de administración y control de cultivos hidropónicos que ocurrieron en la comunidad de “San Pedro las Playas” y proponer una solución tecnológica que permita la administración de los mismos.

1.3 Objetivos

1.3.1 Objetivo General

Implementar un prototipo constituido por hardware y software, manipulado desde una aplicación móvil, que permita controlar y administrar de forma remota, el estado de cosechas que hagan uso de una técnica hidropónica.

1.3.2 Objetivos Específicos

- Implementar una arquitectura de Hardware (microcontroladores, sensores y módulos) a bajo costo, que permita realizar lecturas constantes de los elementos que son monitorizados dentro de un cultivo hidropónico, como son: verificación del nivel de agua de la solución nutritiva, pH, temperatura y humedad, con el fin de automatizar la circulación de la solución nutritiva por todo el sistema.
- Crear un programa que sea almacenado dentro de un microcontrolador y sirva para comunicarse con cada sensor agregado en la arquitectura de hardware propuesta, deberá ser capaz de obtener su información y almacenarla en una base de datos remota.
- Desarrollar una estructura de base de datos que permita almacenar la información necesaria para llevar el control y la administración del cultivo hidropónico.
- Crear una aplicación para dispositivos móviles con sistema operativo Android que permita controlar y monitorizar los elementos electrónicos que fueron integrados al prototipo hidropónico.

1.4 Hipótesis

Al utilizar las herramientas propuestas de gestión, control y automatización, la persona encargada de administrar el cultivo hidropónico, cuenta con una herramienta que le permite conocer el estado de su cosecha sin necesidad de estar físicamente presente, permitiendo tomar acciones y/o ajustes en caso de ser necesario, si el usuario así lo requiere.

1.5 Justificación

A pesar de que la técnica hidropónica es antigua, cuenta con algunas variantes, y la información de cada técnica está documentada, realizar el proceso desde que la planta germinada es trasplantada en la cama de cultivo, hasta su cosecha, requiere emplear mucho tiempo en la revisión de diversos factores (pH, temperatura, humedad, nivel de agua, tiempo de circulación de solución nutritiva). Si no se tiene la supervisión y el cuidado adecuado, el producto puede contener una serie de problemáticas como: Problemas de fitotoxicidad, desperdicio de agua, desperdicio de alimento, calidad de las plantas.

Por este motivo, el presente trabajo de tesis, propone brindar una solución tecnológica que ayude a la reducción de procesos que son realizados de forma monótona en los cultivos hidropónicos, brindando las herramientas para llevar a cabo un control y automatización de los mismos, tratando de prevenir en mayor medida los posibles errores humanos.

Impacto Social

La automatización de un sistema hidropónico utilizando la herramienta propuesta permitirá que las personas cosechen y administren sus propias hortalizas, en espacios reducidos, utilizando la menor cantidad de recursos naturales.

El prototipo está diseñado para uso personal, sin embargo, puede crecer de forma flexible para producción en masa, solución que serviría a los agricultores del municipio de Acapulco y puedan ver en él, una alternativa más, para la cosecha de hortalizas.

Impacto Económico

La hidroponía permite cultivar una gran variedad de hortalizas algunos ejemplos son: acelgas, ajos, alcachofas, calabazas, cebollas, ejotes, jitomates, pepinos, zanahorias, chiles, todas las variedades de lechugas, etc. También permite cultivar frutos, como son: arándanos, fresas, frambuesas, zarzamoras, granada, maracuyá, melón, papaya, piña, plátano, sandía, entre otros.

Debido a la amplia variedad de frutos, hortalizas y plantas aromatizantes que se pueden cultivar utilizando esta técnica, se puede suplir con facilidad la compra de estos

productos en el mercado, consumiendo los productos cosechados en el hogar utilizando el prototipo propuesto, se puede tener un ahorro en la economía familiar.

Otra forma de dar uso a las cosechas realizadas, es la venta de hortalizas. Al tratarse de plantas constantemente controladas y al no contener fertilizantes ni plaguicidas, la calidad de las mismas puede ser considerada como producto orgánico, existe un nicho importante de personas que sólo consumen este tipo de productos y pagan una cantidad superior a lo que se pagaría por el mismo producto en el mercado.

Impacto Tecnológico

Utilizando la aplicación móvil propuesta, la administración y control de los cultivos hidropónicos será más sencilla, ya que brindará información de: históricos, estado actual del cultivo, información de usuario activo, circulación manual o automática de la solución nutritiva, y una serie de factores que permitirá conocer en todo momento el estado actual de la cosecha sin estar físicamente presente.

1.6 Alcances y limitaciones

- El prototipo de la estructura hidropónica, que será utilizado para probar la herramienta propuesta en el presente trabajo de investigación, utiliza la técnica de Flujo y Reflujo.
- El sistema automatizará la circulación del agua por toda la estructura hidropónica, y las lecturas de cada uno de los sensores integrados al componente electrónico.
- La temperatura y la humedad sólo servirán como indicadores, almacenando esta información en la base de datos.
- Las hortalizas del cultivo hidropónico que se pretendan cosechar deberán ser germinadas previamente de forma manual y serán trasplantadas hacia el prototipo de igual manera.
- La recolección de la cosecha, la limpieza del prototipo hidropónico, el vaciado de la solución nutritiva dentro del contenedor, también será realizado de forma manual.
- Las pruebas del presente trabajo serán realizadas con las siguientes hortalizas: Lechuga orejona, cilantro y maracuyá.

1.7 Tabla comparativa de trabajos relacionados

Proyecto	Inv.	Técnica Hidropónica	IoT	Com.	Web	Móvil	IA
<i>Detección y diagnóstico de fallas en hidroponía utilizando herramientas computacionales inteligentes de profundidad</i>	Si		Ventilación, evaporación, luz, sombra				Algoritmo genético
<i>Medidor pH, EC y temperatura</i>	No	Cultura de agua profunda	Arduino, pH, EC, temperatura		Bluetooth	Android	
<i>Aplicaciones de aprendizaje automático en conjunto de datos agrícolas para mejorar la granja inteligente</i>							Redes neuronales, modelos de regresión lineal
<i>Módulo sensor de alta precisión y bajo costo para sistema de cultivo hidropónico</i>	No	Sistema por goteo	Nivel agua, temp. Del agua, EC		Wi-Fi		
<i>Sistema hidropónico automatizado basado en internet de las cosas</i>	Si		Arduino, temp., hum., pH, EC, temp. Del aire, luz		LCD		
<i>Cultivo hidropónico de espinaca mediante la técnica NFT e Invernadero para el control de variables ambientales</i>	Si	NFT	Arduino, temp, hum, pH		LCD		
<i>Hommons: Sistema hidropónico de administración y monitoreo para una granja NFT basada en internet de las cosas que usa tecnología web.</i>	No	NFT	Arduino, Raspberry Pi, celdas solares, temp. De agua, nivel de agua, pH, EC		Wi-Fi	PHP	
<i>Sistema de identificación de la edad de la planta de cultivo hidropónico en exteriores basado en el procesamiento de imágenes digitales</i>	No	NFT	Raspberry Pi, módulo cámara		Wi-Fi	PHP	Técnicas de visión
<i>Sistema inteligente para el control de bicarbonato en riego para agricultura de precisión hidropónica</i>	Si	Sistema por goteo	pH, ventilación, irrigación, EC		Wi-Fi	Vue	Android
<i>Tekax - Aplicación móvil para el control de cultivos hidropónicos utilizando IoT</i>	No	Flujo y reflujos	NodeMCU, pH, temp., hum., nivel agua		Wi-Fi		Android

Tabla 1-1. Comparativa de los trabajos relacionados

1.8 Organización de la tesis

Capítulo 1 - Introducción

Esta descrito un análisis de los trabajos relacionados con procesos hidropónicos, analizando diferentes enfoques y dividiendo los trabajos por áreas (instituciones, productos de software, productos de hardware, productos de hardware y software). Finalmente se toma en consideración cada uno de los trabajos analizados y se establece una problemática que se pretende solucionar, estableciendo los objetivos del proyecto, hipótesis, justificación, los alcances y limitaciones del mismo.

Capítulo 2 – Marco Teórico

Se realiza un análisis de la arquitectura que conforman los componentes propuestos, cada componente da solución a una problemática en específico dentro del proyecto Tekax. Se abordan las herramientas y procesos necesarios para la construcción de un producto de calidad, revisando ¿Cuáles son las técnicas hidropónicas? y ¿Qué factores son necesarios para un correcto crecimiento de las plantas?, posteriormente se contempla ¿Qué circuitos electrónicos pueden ser necesarios para recolectar la información del entorno hidropónico?, también se analizan las diferencias entre los diferentes gestores de base de datos SQL y NoSQL, finalmente se determina ¿Cuáles son las características que debe tener una aplicación móvil de calidad?.

Capítulo 3 – Análisis y diseño

Se determinan las herramientas necesarias, realizando un diseño conceptual de cada uno de los 4 componentes que conforman la arquitectura del proyecto Tekax. Para el prototipo hidropónico se detalla el material, las medidas y un esquema de construcción del mismo. Para la automatización hidropónica se realiza la selección de componentes a utilizar y se crean los algoritmos que detallan su comunicación de forma generalizada. Respecto a la base de datos, se especifican los documentos, campos y la estructura que será utilizada. Finalmente, para el diseño de la aplicación móvil, se determina el diagrama de casos de uso y se realizan diagramas de secuencia para cada uno de los procesos realizados dentro de la misma.

Capítulo 4 - Implementación

La implementación esta segmentada por componentes. El prototipo hidropónico detalla la forma en la que, partiendo del concepto se procede a su construcción, integración y prueba. La automatización hidropónica construye el circuito electrónico, probando cada sensor y módulo por separado, integrando todo en un solo componente. La base de datos es construida acorde al análisis previo, realizando pruebas de lectura y escritura de información. Finalmente, la aplicación móvil es desarrollada partiendo con el diagrama de

casos de uso, los diagramas de secuencia y la interface de usuario previamente diseñada, implementando cada una de las secciones y realizando las pruebas necesarias.

Capítulo 5 – Resultados

Se muestran los resultados de la integración de los componentes. El resultado de la automatización hidropónica es una placa Fenólica¹⁸ que integra cada uno de los sensores y módulos necesarios para llevar a cabo dicho proceso. La aplicación móvil muestra la interface implementada, la descripción y el funcionamiento de cada uno de los módulos dentro de la misma. El prototipo hidropónico detalla los resultados de las pruebas realizadas sobre diferentes tipos de hortalizas.

Capítulo 6 - Conclusiones

Muestra la retrospectiva del proceso de desarrollo, las conclusiones a las que se llega después de haber analizado, implementado y probado cada componente. Por último, se propone trabajo futuro que es posible realizar en cada elemento del proyecto Tekax, que permitirá mejorar la experiencia del usuario.

¹⁸ Es una tablilla hecha generalmente de cobre, ideal para armar prototipos con soldadura de circuitos integrados.

CAPÍTULO 2

Marco Teórico

Capítulo 2 Marco Teórico

El presente trabajo de tesis, se divide en 4 componentes principales, debido a que cada uno de ellos presenta características propias, son tratados y evaluados de forma independiente, la conjunción de todos ellos da forma al proyecto Tekax.

2.1 Prototipo hidropónico (Componente 1)

Cuando se construye un prototipo hidropónico es necesario tomar en cuenta una serie de factores como: la técnica, ubicación, recipientes, contenedores, sustratos, medios de cultivo, preparación, siembra, manejo de plantas, control de plagas, entre otros (Beltrano & Gimenez, 2015).

A continuación, se analizan los elementos necesarios que deben ser tomados en cuenta para la realización de cultivos hidropónicos.

2.1.1 Técnicas Hidropónicas

La Hidroponía, es un conjunto de técnicas que permite el cultivo de plantas en un medio libre de suelo. Puede ser implementada en estructuras simples o complejas, permitiendo producir plantas principalmente de tipo herbáceo aprovechando sitios o áreas como azoteas, suelos infértiles, terrenos escabrosos, invernaderos climatizados o no, entre otros. A partir de este concepto se desarrollaron técnicas que se apoyan en sustratos (medios que sostienen a la planta), o en sistemas con aportes de soluciones de nutrientes estáticos o circulantes, sin perder de vistas las necesidades de la planta como la temperatura, humedad, agua y nutrientes. La palabra hidroponía deriva del griego *Hidro* (agua) y *Ponos* (labor o trabajo) lo cual significa literalmente trabajo en agua. Sin embargo, en la actualidad se utiliza para referirse al cultivo sin suelo (Beltrano & Gimenez, 2015).

Dentro de la hidroponía existen diferentes tipos de técnicas que permiten el desarrollo de estos cultivos, a continuación, se analizarán las ventajas y desventajas de cada una de ellas.

Sistema de cultura de agua profunda

Es una técnica hidropónica que suministra los nutrientes directamente en las raíces de la planta de forma continua a través del agua. Esta técnica asegurará que las raíces de la planta siempre se sumerjan en agua y oxígeno, mismo que, es suministrado utilizando una piedra de aireación, ver Figura 2-1. La ventaja del sistema de cultura de agua profunda es que tiene un alto nivel de oxígeno, menos fertilizante y bajo costo de mantenimiento y monitoreo (Saaid et al., 2013).

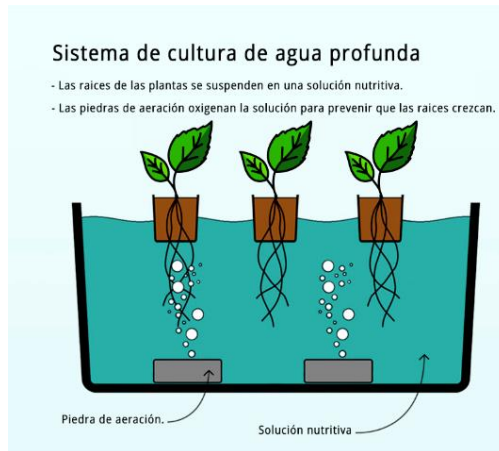


Figura 2-1 Sistema de cultura de agua profunda.

Sistema de Flujo y reflujo

Funciona al inundar temporalmente la bandeja de cultivo con una solución nutritiva y posteriormente drenarla dentro de un contenedor, ver Figura 2-2. Esta acción se realiza normalmente con una bomba sumergida que está conectada a un temporizador. El temporizador se programa para ser encendido varias veces al día, dependiendo del tamaño y tipo de las plantas, la temperatura, la humedad y el tipo de medio de cultivo utilizado (Dunn, 2015).



Figura 2-2 Sistema de flujo y reflujo.

Aeroponía

Es simple de operar y no involucra maquinaria compleja. El sistema impulsor que administra la solución nutritiva a las plantas está compuesto por tres partes: La bomba de agua, el eje y el aspersor.

La bomba de agua envía la solución nutritiva a través del eje y el aspersor rocía la solución directamente en las raíces de las plantas dentro del sistema hidropónico, ver Figura 2-3 (Zobel et al., 2008).

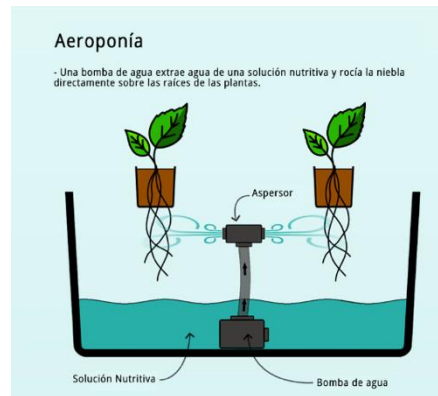


Figura 2-3 Sistema Aeropónico.

Acuaponía

La acuaponía es un sistema de producción de alimentos que incluye la incorporación de dos o más componentes: peces y vegetales o plantas, en un diseño basado en la recirculación de agua.

El principio básico radica en el aprovechamiento de la energía del sistema para utilizar diferentes formas por los componentes comerciales que desean producirse. Sólo una fracción del alimento para los peces 20% - 30%, se metaboliza e incorpora como tejido, mientras que el resto (excreción, alimento no consumido y diluido), se utiliza como nutriente para el crecimiento de las plantas; éstas pueden ser vegetales, frutas o flores, ver Figura 2-4 (García Ulloa et al., 2005).

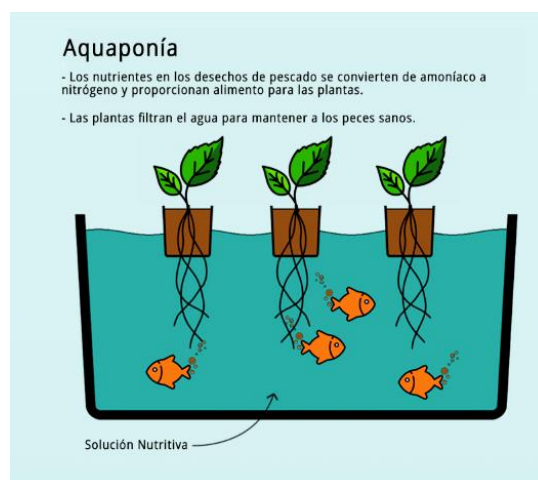


Figura 2-4 Sistema Acuapónico.

Un componente extra que debe encontrarse dentro del sistema, está constituido por las colonias de bacterias nitrificantes¹⁹ en el sustrato de las plantas, permitiendo realizar dos funciones: degradar los compuestos nitrogenados en su forma peligrosa para los peces (Amoníaco y Nitritos), y proveer de nutrientes a las plantas.

Sistema por goteo

Consiste en administrar la solución nutritiva a las plantas utilizando un sistema de goteo, algunas hortalizas necesitan menos agua que otras y los sistemas hidropónicos de goteo pueden configurarse para controlar fácilmente la cantidad de agua que obtienen sus plantas.

Una bomba de agua circula la solución nutritiva dentro del sistema hidropónico, el sistema de goteo permite que esta solución sea administrada directamente a las plantas, el agua que es drenada de las plantas es enviada nuevamente a la solución nutritiva para ser reutilizada o simplemente se desecha, ver Figura 2-5.

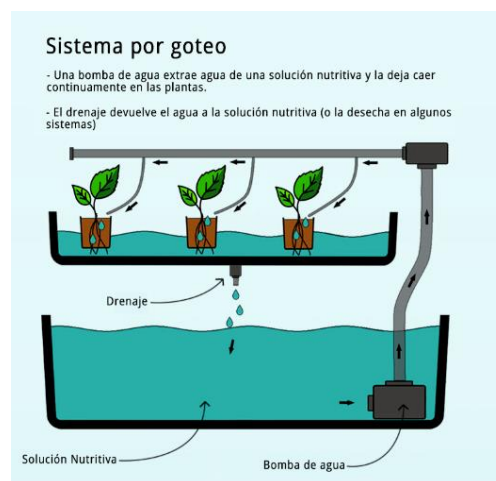


Figura 2-5 Sistema por goteo.

Técnica de película de nutrientes

La solución nutritiva puede circular de forma continua o intermitente. Se utiliza en los sistemas de canales profundos o semiprofundos, el aporte de oxígeno no es necesario, ya que la solución se encuentra en movimiento (Beltrano & Gimenez, 2015).

La técnica de película de nutrientes se basa en el uso de un canal con una pendiente adecuada, un caudal correcto, y una longitud correcta del canal. La principal ventaja de este sistema sobre otras formas de hidroponía es que las raíces de las plantas están expuestas a un suministro adecuado de agua, oxígeno y nutrientes, ver Figura 2-6.

¹⁹ Son organismos quimiolitotróficos que incluyen especies de los géneros Nitrosomonas, Nitrosococcus, Nitrobacter y Nitrococcus. Estas bacterias consiguen su energía por la oxidación de los compuestos inorgánicos del nitrógeno.

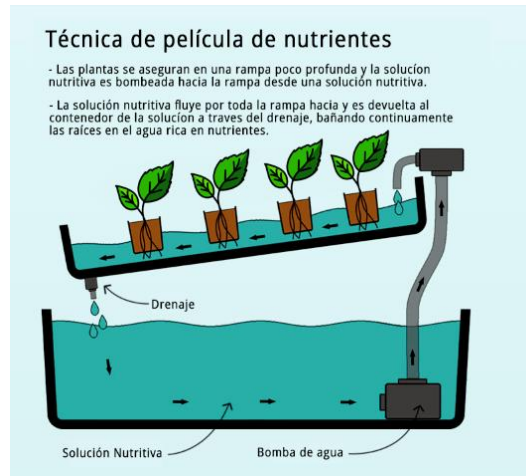


Figura 2-6 Técnica de película de nutrientes.

Riego de forma manual

Es la técnica más simple de todas, sólo es necesario un contenedor para depositar la solución nutritiva, misma que, es vertida de forma manual directamente sobre las plantas, ver Figura 2-7.



Figura 2-7 Riego hidropónico manual.

2.1.2 Localización e instalación de una huerta hidropónica.

Estas huertas pueden ser ubicadas en distintos lugares (paredes, techos, patios, ventanas, terrazas), sin embargo existen algunos criterios a tomar en consideración para obtener mayor eficiencia, mejores resultados y éxito en el producto final (Izquierdo & Marulanda, 2003).

La idea de que los cultivos sin tierra sólo se pueden obtener en condiciones de invernaderos plásticos no es completamente cierta. Algunas experiencias conducidas en distintos países de América Latina y el Caribe con cultivos de apio, acelgas, lechugas, nabos, pepinos, perejil, rabanitos, tomates y otras hortalizas, sin utilizar cobertura plástica, indican

que es posible obtener buenos productos y plantas a la libre exposición, cuando ellas están adaptadas a las condiciones ambientales del lugar donde se cultivan.

Factores que hay que tomar en cuenta para la instalación de un huerto hidropónico:

- Disponer de un mínimo de seis (6) horas de luz solar al día.
- El huerto debe estar próximo a una fuente de suministro de agua.
- No estar expuesto a vientos fuertes.
- Estar próximo al lugar donde se preparan y guardan los nutrientes hidropónicos.
- No estar excesivamente sombreado por árboles o construcciones.
- Ser protegido o cercado para evitar el acceso a animales domésticos
- De ser posible proteger contra condiciones extremas del clima (heladas, granizo, alta radiación solar)
- Lejos de focos de contaminación como aguas residuales o desechos industriales.

2.1.3 Recipientes y Contenedores

Los tipos de recipientes y contenedores que pueden ser utilizados o contruidos, deben estar relacionados con el espacio disponible, las posibilidades técnicas, económicas, las necesidades y aspiraciones de progreso y desarrollo de la persona que implementará la técnica (Izquierdo & Marulanda, 2003).

Las dimensiones (largo y ancho) de los contenedores pueden ser muy variables, pero su profundidad en cambio no debe ser mayor de 10-12 cm, dado que en el sistema hidropónico no es necesario un espacio mayor para el desarrollo de las raíces de las plantas.

Elementos que pueden ser utilizados como contenedores dentro de los cultivos hidropónicos:

- Cajas de madera forradas por dentro con plástico.
- Canales de plástico o bambú.
- Tubos de PVC o plástico.
- Llantas viejas de vehículo.
- Galones de aceite desocupados y abiertos por la mitad.

Para el desarrollo de la cama de cultivo en el prototipo hidropónico se recomienda lo siguiente:

- Debe existir un mínimo de 17 cm entre plantas, para su correcto crecimiento y expansión de raíces.
- Los agujeros que almacenarán las plantas en el sistema hidropónico, deben ser máximo de 2.5 cm (una pulgada) de diámetro.
- La solución nutritiva en los tubos debe permanecer entre 5.5 y 6.5 de pH.

- Evitar que la luz tenga contacto con el líquido de la solución nutritiva, esto para evitar el crecimiento de algas y una mayor evaporación de agua dentro del contenedor.
- Se recomienda una esponja plástica que debe tener 2.5cm x 2.5cm de largo y ancho de espesor. A cada cubo se hace un corte vertical atravesando de arriba hacia abajo la esponja. Este corte es donde se trasplantará la planta. Los cubos deben ser humedecidos con solución nutritiva antes del trasplante.
- Al momento del trasplante se sacan las plantas y se lava la raíz para que no quede sustrato (sin tocarla ni maltratarla), y se coloca dentro del corte que se hizo a la esponja dejando el cuello de la planta exactamente un centímetro por debajo de la superficie del cubo de esponja, a esta parte se le conoce como post-almácigo²⁰.

2.1.4 Sustratos o medios de cultivo

Los sustratos deben tener gran resistencia al desgaste o a la meteorización y es preferible que no tengan sustancias minerales solubles para no alterar el balance químico de la solución nutritiva. El material no debería ser portador de ninguna forma viva de macro o micro organismo para disminuir el riesgo de propagar enfermedades o causar daño a las plantas, a las personas o a los animales que las van a consumir (Izquierdo & Marulanda, 2003).

Lo más recomendable para un buen sustrato es:

- Que las partículas que lo componen tengan un tamaño no inferior a 0.5 milímetros y no superior a 0.7 milímetros.
- Que retengan una buena cantidad de humedad, pero que además faciliten la salida de los excesos de agua que pudieran caer con el riego o con la lluvia.
- Que no retengan mucha humedad en la superficie.
- Que no se descompongan o degraden con facilidad.
- Que tengan preferentemente coloración oscura.
- Que no contengan elementos nutritivos.
- Que no contengan micro organismos perjudiciales a la salud de los seres humanos o de las plantas.
- Que no contengan residuos industriales o humanos.
- Que sean abundantes y fáciles de conseguir, transportar y manejar.

2.1.5 Semillas

Las semillas que se utilizan son las mismas que en la horticultura tradicional. Debe tratarse de sembrar semillas producidas y distribuidas por casas comerciales semilleras.

²⁰ Es un sitio donde se siembran los vegetales o un lugar donde se guardan las semillas.

de reconocida trayectoria de preferencia, también se pueden reutilizar las semillas de una cosecha hidropónica.

2.1.6 Aireación

En el sistema de raíz flotante es indispensable batir con las manos al menos dos veces al día la solución nutritiva, con el fin de redistribuir los elementos nutritivos por todo el líquido y oxigenar la solución. Sin ello, las raíces empiezan a oscurecerse y a limitar la absorción de alimentos y agua. Si dentro del sistema hidropónico se cuenta con una bomba de agua que circula la solución nutritiva, no es necesario realizar este proceso, ya que la bomba al circular el agua realiza el proceso de oxigenación.

2.1.7 Ciclo de vida del proceso hidropónico.

El proceso dentro de los cultivos hidropónicos se puede dividir en 3 etapas principales, ver Figura 2-8.

1. Almacigo. Contempla el proceso de siembra, utilizando sustrato, agua y semillas, mismas que son regadas durante su germinación.
2. Post-Almacigo. Una vez que las semillas han sido germinadas y la planta cuenta con un tamaño considerable de unos 5 cm, pueden ser trasplantadas hacia el prototipo hidropónico, dando inicio a la etapa de crecimiento donde se desarrollarán utilizando la solución nutritiva, en la etapa final, se recomienda utilizar una mayor concentración de nutrientes, esto dependerá del tipo de planta germinada.
3. Cosecha. Retirar las plantas del prototipo hidropónico y limpiar el prototipo para la siguiente cosecha.

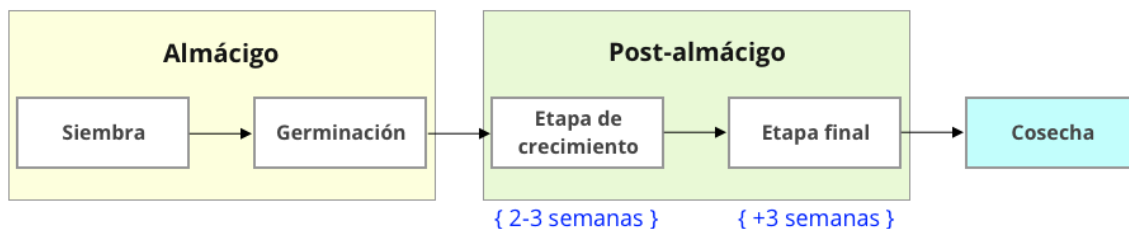


Figura 2-8 Proceso de cosecha en cultivos hidropónicos.

Es preciso conocer los tiempos necesarios entre siembra y germinación, germinación y trasplante, trasplante y cosecha de cada cultivo, ya que son variables.

2.1.8 Otras consideraciones

- Es importante en todo momento tener cuidado constante con la presencia de plagas que puedan afectar la cantidad y la calidad de las cosechas.

- También se debe evitar que los cultivos reciban exceso de sol, se puede utilizar una malla oscura para reducir la radiación solar.
- Evitar los excesos de frío, se recomienda cubrir los cultivos más susceptibles a este fenómeno con plástico transparente, preferentemente de uso agrícola, durante los días u horas que haya riesgo de que ocurran bajas temperaturas.
- Conocer las distancias de siembra o de trasplante recomendadas para las diferentes especies.

2.2 Automatización hidropónica (Componente 2)

La automatización en un sistema, es un conjunto de técnicas donde se transfieren las tareas de producción, realizadas habitualmente por operadores humanos a un conjunto de elementos tecnológicos.

Un sistema automatizado consta de 2 partes:

1. *Parte de Mando:* Es la parte que actúa directamente sobre la máquina. Son los elementos que hacen que la máquina se mueva y realice la operación deseada. Los elementos que forman la parte operativa son los accionadores de las máquinas como: motores, cilindros, actuadores, entre otros, y los captadores como: sensores, fotodiodos, entre otros.
2. *Parte Operativa:* Suele ser un autómata²¹ programable, este debe ser capaz de comunicarse con todos los componentes del sistema automatizado.

La hidroponía cuenta con varias operaciones que son constantemente repetidas y pueden ser automatizadas y controladas, existen muchos componentes electrónicos que permiten automatizar una actividad, este trabajo de investigación se enfocará solo en estos elementos.

2.2.1 Microcontroladores

Un microcontrolador realmente es una pequeña computadora en un chip. Tiene un procesador, uno o dos kilobytes de memoria de acceso aleatorio (*RAM*²²) para almacenar datos, unos pocos kilobytes de memoria de sólo lectura programable y borrable (*EPROM*²³) o memoria Flash para guardar sus programas y tiene pines de entrada y salida. Estos pines

²¹ Un autómata es un modelo matemático para una máquina de estados finitos, dada una entrada de símbolos, “salta” a través de una serie de estados de acuerdo a una función de transición.

²² La memoria de acceso aleatorio (Random Access Memory, RAM) se utiliza como memoria de trabajo de computadoras y otros dispositivos para el sistema operativo, los programas y la mayor parte del software.

²³ EPROM son las siglas de Erasable Programmable Read-Only Memory (ROM programable borrable). Es un tipo de chip de memoria ROM no volátil que permite ser borrado para ser reprogramado posteriormente.

de Entrada/Salida (E/S) conectan el microcontrolador al resto de sus componentes electrónicos (Monk, 2012).

Las entradas pueden leer información digital (¿el interruptor está encendido o apagado?), o analógica (¿cuál es el voltaje en un pin?). Esto abre la oportunidad de conectar diferentes tipos de sensores: luz, temperatura, sonido, entre otros.

Las salidas también pueden ser analógicas o digitales. Por lo tanto, configurar un pin para que esté encendido o apagado (0 voltios o 5 voltios) es relativamente sencillo, otra forma de hacer uso de estas salidas, es utilizando el control con dispositivos de mayor potencia, como motores, bombas, aires, luces, entre otras cosas. Básicamente es posible conectar cualquier componente electrónico al microcontrolador y operarlo de forma remota o autónoma utilizando estas entradas y salidas (Monk, 2012).

ESP8266

Es un microcontrolador reprogramable desarrollado por la empresa *Espressif*, el cual, permite la conectividad *Wi-Fi* por un precio económico, una de sus principales características es que puede ser utilizado de forma independiente, o funcionar como un transceptor²⁴ con otros microcontroladores, a través del puerto Serie mediante los terminales *TX* y *RX*.

El cache integrado de alta velocidad ayuda a aumentar el rendimiento del sistema y a optimizar la memoria del mismo. El microcontrolador contiene interruptores de antena, amplificador de potencia, amplificador de recepción de bajo ruido, filtros y módulos de gestión de potencia. Su diseño compacto minimiza el tamaño de la *PCB*²⁵ y requiere circuitos externos mínimos. Cuenta con un procesador de 32 bits de la serie *L106 Diamond de Tensilica* y *SRAM*²⁶ en chip. Finalmente puede conectarse con sensores externos y otros dispositivos a través de los *GPIO*²⁷ (Systems, 2020).

En la Figura 2-9, se muestra un diagrama general de los pines que integran el microcontrolador.

²⁴ Es un dispositivo que cuenta con un transmisor y un receptor, comparten parte de la circuitería o se encuentran dentro de la misma caja.

²⁵ Placa de circuito impreso, es una superficie constituida por caminos, pistas o buses de material conductor laminadas sobre una base no conductora.

²⁶ Memoria estática de acceso aleatorio, capaz de mantener los datos, mientras siga alimentada, sin necesidad de circuito de refresco.

²⁷ Entrada / Salida de propósito general, es un pin genérico en un chip, cuyo comportamiento se puede programar en tiempo de ejecución.

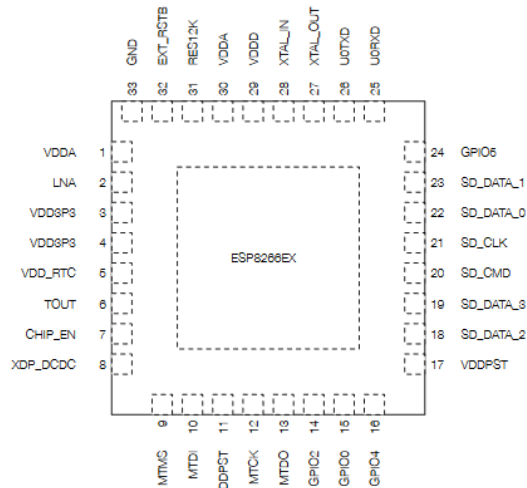


Figura 2-9 Pines del microcontrolador ESP8266.

Secuencia numérica de pines que integran el microcontrolador *ESP8266* (Systems, 2020), ver Tabla 2-1

Pin	Nombre	Tipo	Función
1	VDDA	P	Potencia analógica 2.5V – 3.6V
2	LNA	E/S	Interfaz de antena RF
3	VDD3P3	P	Potencia del amplificador 2.5V – 3.6V
4	VDD3P3	P	Potencia del amplificador 2.5V – 3.6V
5	VDD_RTC	P	NC (1.1 V)
6	TOUT	E	Se puede utilizar para probar el voltaje de la fuente de alimentación de VDD3P3 y el voltaje de alimentación de entrada de TOUT.
7	CHIP_EN	E	Habilitar chip. Alto (encendido). Bajo (apagado)
8	XPD_DCDC	E/S	Despertar, GPIO16
9	MTMS	E/S	GPIO 14; HSPI_CLK
10	MTDI	E/S	GPIO 12; HSPI_MISO
11	VDDPST	P	Fuente alimentación E/S 1.8V – 3.6V
12	MTCK	E/S	GPIO 13; HSPI_MOSI; UART0_CTS
13	MTDO	E/S	GPIO 15; HSPI_CS; UART0_RTS
14	GPIO2	E/S	UART TX durante la programación flash; GPIO2
15	GPIO0	E/S	GPIO0; SPI_CS2
16	GPIO4	E/S	GPIO4
17	VDDPST	P	Fuente alimentación E/S 1.8V – 3.6V
18	SDIO_DATA_2	E/S	Conectar aSD_D2; SPIHD; HSPIHD; GPIO9
19	SDIO_DATA_3	E/S	Conectar a SD_D3; SPIWP; HSPIWP; GPIO10
20	SDIO_CMD	E/S	Conectar a SD_CMD; SPI_CS0; GPIO11

Pin	Nombre	Tipo	Función
21	SDIO_CLK	E/S	Conectar a SD_CLK; SPI_CLK; GPIO6
22	SDIO_DATA_0	E/S	Conectar a SD_D0; SPI_MISO; GPIO7
23	SDIO_DATA_1	E/S	Conectar a SD_D1; SPI_MOSI; GPIO8
24	SDIO_DATA_1	E/S	GPIO5
25	U0RXD	E/S	UART RX durante la programación flash; GPIO3
26	U0TXD	E/S	UART TX durante la programación flash; GPIO1
27	XTAL_OUT	E/S	Conectarse a la salida del oscilador de cristal
28	XTAL_IN	E/S	Conectarse a la entrada del oscilador de cristal
29	VDDD	P	Potencia análoga 2.5V – 3.6V
30	VDDA	P	Potencia análoga 2.5 – 3.6V
31	RES12K	E	Conexión serial con un resistor de 12 kΩ y conexión a tierra.
32	EXT_RSTB	E	Reseteo externo

Tabla 2-1. Pines de conexión del ESP8266.

Características principales:

- Utiliza el protocolo 802.11 b/g/n.
- Procesador *Tensilica* a 32 bits.
- Interface periférica mediante *UART/SDIO/SPI/I2C/I2S/Control remoto por IR*.
- Voltaje de operación, 2.5V – 3.6V.
- Corriente de operación, 80 MA.
- Tamaño (5mm x 5m).
- Protocolos de red soportados *IPv4, TCP/UDP/HTTP*.
- Modos de *Wi-Fi Estación/ModoAP/ModoAP+Estación*.
- Seguridad *WPA/WPA2*.

2.2.2 Módulos

Un módulo es un circuito que contiene componentes electrónicos propios integrados entre sí, con el objetivo de realizar una actividad específica. Para poder utilizarlo generalmente se conecta y configura de acuerdo a las especificaciones de su creador, es necesario leer la documentación que define su funcionamiento y limitaciones.

La comunidad de desarrolladores de aplicaciones electrónicas han difundido una gran variedad de módulos que permiten realizar proyectos de una manera mucho más ágil (Monk, 2017).

Ejemplos de estos módulos son:

- Módulos para detectar movimiento.
- Módulos para conexión a internet.

Módulo reloj en tiempo real AT24C32 (I2C³²)

Una de las acciones que son necesarias dentro el proceso de automatización del presente proyecto, es la circulación de la solución nutritiva a través del prototipo hidropónico, durante lapsos específicos de tiempo, para llevar a cabo esta tarea, será necesario utilizar un temporizador.

El microcontrolador *ESP8266*, posee la capacidad de brindar información de tiempo y hora, sin embargo, un problema que presenta, es que, al desconectarlo de la toma de corriente, la fecha y la hora son restauradas. Debido a las características que posee el módulo *AT24C32* es posible configurar una fecha y hora para que la información se mantenga, independientemente si el microcontrolador es encendido o apagado, ya que cuenta con una batería que provee la energía suficiente para su funcionamiento.

El módulo *AT24C32* que utiliza el protocolo de comunicación *I2C*, es extremadamente preciso y de bajo costo, cuenta con un oscilador³³ de cristal que mide la temperatura. Sus principales características son: mantener fechas y horas exactas, poseer la capacidad de establecer 2 horas programables como alarma, funciona con una batería de 3.6V y finalmente contar con una memoria *EEPROM* que le permite almacenar 4 Bytes de forma permanente.

Diagrama de conexión del módulo *AT24C32* y el microcontrolador, ver Figura 2-11.

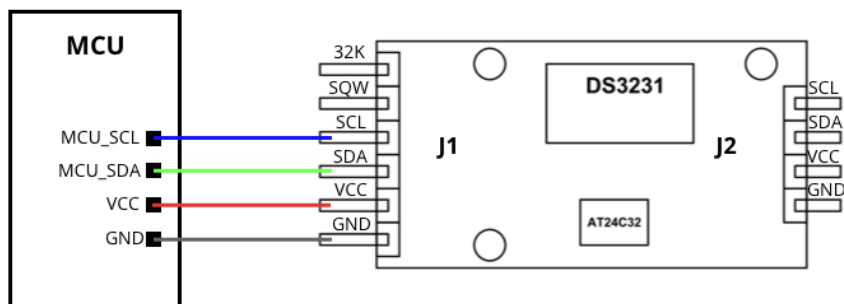


Figura 2-11. Diagrama de conexión entre el módulo *AT24C32* y el MCU.

Secuencia de pines para el módulo *AT24C32* (Atmel, 2003), ver Tabla 2-2.

No.	Pin	Función
J1 - 1	32K	Salida directa del oscilador
J1 - 2	SQW	Salida de interrupción o señal de reloj
J1 - 3	SCL	Señal de reloj de bus I2C
J1 - 4	SDA	Señal de datos de bus I2C

³² Es un bus de serie de datos. Se utiliza principalmente para la comunicación entre diferentes partes de un circuito.

³³ Es un circuito electrónico que produce una señal oscilante y periódica, a menudo una onda senoidal o una onda cuadrada.

No.	Pin	Función
J1 - 5	VCC	Alimentación de 5V del módulo
J1 - 6	GND	Conexión a tierra
J2 - 1	SCL	Señal de reloj de bus I2C
J2 - 2	SDA	Señal de datos de bus I2C
J2 - 3	VCC	Alimentación de 5V del módulo
J2 - 4	GND	Conexión a tierra

Tabla 2-2. Pines de conexión del módulo AT24C32.

Módulo analógico pH-4502C

El pH, es una variable que debe ser constantemente controlada y monitorizada dentro de los cultivos hidropónicos, en el presente trabajo de tesis se utiliza el módulo *pH-4502C*, permitiendo medir el nivel de pH de una solución acuosa, cuenta con un *LED* que funciona como indicador de encendido y una interface de conexión *BNC*³⁴ que ensambla a su vez, una sonda que mide la diferencia entre dos electrodos: un electrodo de referencia (plata / cloruro de plata) y un electrodo de vidrio que es sensible al ion de hidrogeno.

En la Figura 2-12, se muestra el diagrama de conexión entre el microcontrolador y el módulo *pH-4502C*.

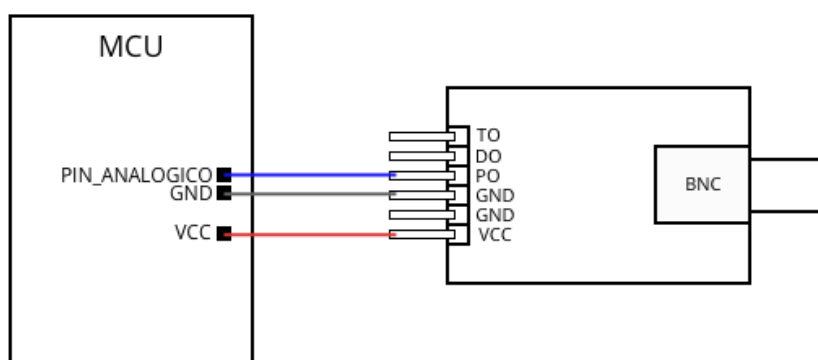


Figura 2-12. Diagrama de conexión entre el módulo *pH-4502C* y el MCU.

Secuencia numérica de pines del sensor pH análogo (DIY, 2017), ver Tabla 2-3.

No.	Pin	Función
1	TO	Pin de salida de temperatura
2	DO	Pin de salida 3.3V
3	PO	pH salida análoga
4	GND	Conexión a tierra
5	GND	Conexión a tierra
6	VCC	Corriente directa 5V

³⁴ Bayonet Neill- Concelman. Es un tipo de conector, de rápida conexión/desconexión, utilizado para cable coaxial.

Consideraciones para el correcto uso del módulo pH-4502C:

- Evitar la contaminación cruzada, si el sensor es introducido en diferentes soluciones acuosas con diferente pH, es necesario descontaminarlo utilizando agua destilada antes de introducirlo a una segunda solución.
- La conexión BNC debe permanecer limpia y seca en todo momento.
- Las lecturas con el sensor cambian con el tiempo, es necesario calibrarlo con regularidad para asegurarse que las mediciones son las correctas.
- Se recomienda utilizar soluciones madre para calibrar la sonda pH, en un rango de 4.01 a 6.86 de pH, dependiendo del uso deseado.

2.2.3 Sensores

Un sensor es todo aquello que tiene una propiedad sensible a una magnitud del medio, y al variar esta magnitud también varía con cierta intensidad la propiedad, es decir, manifiesta la presencia de dicha magnitud, y también su medida.

Un sensor en la industria es un objeto capaz de variar una propiedad ante magnitudes físicas o químicas, llamadas variables de instrumentación, y transformarlas con un transductor³⁵ en variables eléctricas.

Las variables de instrumentación pueden ser, por ejemplo: Intensidad lumínica, temperatura, distancia, aceleración, inclinación, presión, desplazamiento, fuerza, torsión, humedad, movimiento, pH, EC, entre otras.

Los sensores a menudo se usan para realizar mediciones físicas y todos permiten digitalizarse en una forma para que puedan hacer cosas como: mostrar los datos o realizar ciertas tareas dependiendo de las lecturas del sensor (Monk, 2017).

A continuación, se muestran los sensores que han sido seleccionados para integrar la capa de hardware del proyecto.

Sensor de temperatura y humedad AM2302

Dentro del proyecto, se realizarán lecturas de temperatura y humedad de forma constante, el sensor *AM2302* fue seleccionado para llevar a cabo esta tarea, utiliza un termistor³⁶ y un circuito integrado, permitiendo realizar las lecturas de forma eficiente. El

³⁵ Es un dispositivo capaz de transformar o convertir una determinada manifestación de energía de entrada, en otra diferente a la salida, pero de valores muy pequeños en términos relativos con respecto a un generador.

³⁶ Son resistencias de coeficiente de temperatura negativo, constituidas por un cuerpo semiconductor cuyo coeficiente de temperatura sea elevado, es decir, su conductividad crece muy rápidamente con la temperatura.

termistor contiene dos electrodos y un sustrato que retiene la humedad entre ellos. Entonces, a medida que varía la humedad, cambia la conductividad del sustrato o la resistencia entre ellos. Las variaciones en la resistencia son medidas y procesadas por el circuito integrado, mientras que preparan la información para ser enviadas hacia el microcontrolador (*MCU*).

A continuación, se muestra el diagrama de conexión entre el sensor *AM2302* y el microcontrolador, ver Figura 2-13.

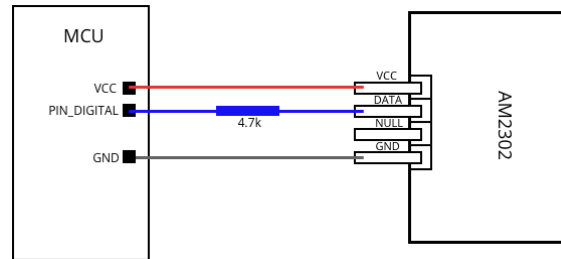


Figura 2-13. Diagrama de conexión entre el sensor *AM2302* y el *MCU*

La secuencia numérica de pines utilizado por el sensor *AM2302* (Liu, 2016) es la siguiente, ver Tabla 2-4.

No.	Pin	Función
1	VCC	Corriente directa 5V
2	DATA	Señal
3	NULL	NULL
4	GND	Conexión a tierra

Tabla 2-4. Pines de conexión del sensor *AM2302*.

Características:

- Rango de temperatura -40°C - 80°C.
- Rango de humedad 0 - 100%
- Voltaje necesario 3.3V - 6V
- Señal de salida, una sola vía a través del bus.
- Periodo de sensibilidad, cada 2 segundos.

Sensor ultrasónico *HC-SR04*

Conocer el nivel de agua dentro del contenedor de la solución nutritiva, es un parámetro que debe ser monitorizado en el presente proyecto, para solucionar este problema, se ha optado por utilizar el sensor *HC-SR04*, es un módulo que incorpora un par

de transductores³⁷ de ultrasonido que se utilizan de manera conjunta para determinar la distancia del sensor con un objeto colocado frente al mismo. Un transductor emite una ráfaga de sonido y el otro capta el rebote de dicha onda. El tiempo que tarda la onda sonora en ir y regresar a un objeto puede utilizarse para conocer la distancia que existe entre el origen del sonido y el objeto.

En la Figura 2-14, se muestra el diagrama de conexión entre el sensor y el microcontrolador.

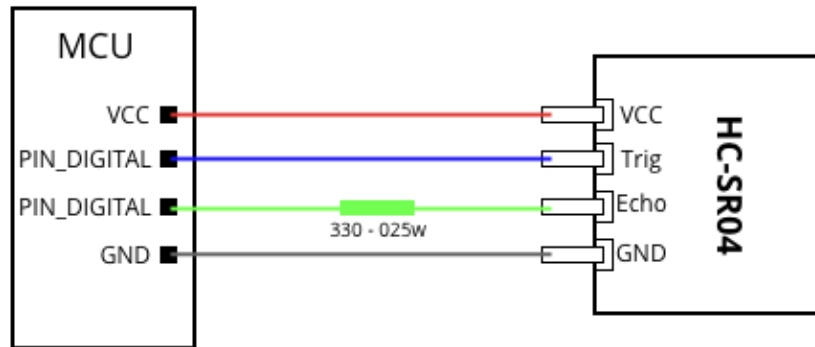


Figura 2-14. Diagrama de conexión entre el sensor HC-SR04 y el MCU.

Secuencia numérica de pines utilizada por el sensor HC-SR04 (Morgan, 2014), ver Tabla 5.

No.	Pin	Función
1	VCC	Corriente directa 5V
2	Trig	Pin de entrada
3	Echo	Pin de salida
4	GND	Conexión a tierra

Tabla 2-5. Pines de conexión del sensor HC-SR04.

2.2.4 Actuadores

Son dispositivos que brindan la posibilidad de transformar diferentes tipos de energía, permitiendo generar algún funcionamiento dentro de un sistema automatizado determinado. Usualmente, los actuadores generan una fuerza mecánica a partir de distintos tipos de energía, como puede ser eléctrica, neumática, o hidráulica.

Los actuadores electrónicos son accionados por medio de corrientes eléctricas. Existen actuadores electrónicos que consumen una considerable cantidad de energía, para este tipo de casos se utilizan controladores. Los actuadores eléctricos son utilizados en

³⁷ Dispositivo que tiene la misión de recibir energía de una naturaleza eléctrica, mecánica, acústica, etc., y suministrar otra energía de diferente naturaleza, pero de características dependientes de la que recibió.

diferentes aparatos mecatrónicos, como robots. Algunos tipos de actuadores electrónicos son:

- Motores de corriente directa.
- Motores de pulsos (paso a paso).
- Electro válvulas.
- Aleaciones con memoria de forma (como el Nitinol³⁸).
- Relevadores

Relevador 5v

Con el reloj en tiempo real analizado en el tema 2.2.2, es posible obtener la fecha y hora exacta, permitiendo conocer cuándo encender o apagar un dispositivo electrónico, en el caso del presente proyecto, la bomba de agua, sin embargo, para realizar la acción es necesario utilizar un relevador, funcionando como un interruptor que permite ser accionado eléctricamente a través de una bobina y un electroimán, se utiliza un pulso electromagnético para excitar la bobina permitiendo así, realizar un cambio de estado, lo cual se traduce en la apertura o cierre del circuito.

En la Figura 2-15, se muestra el diagrama de conexión entre el relevador y el microcontrolador.

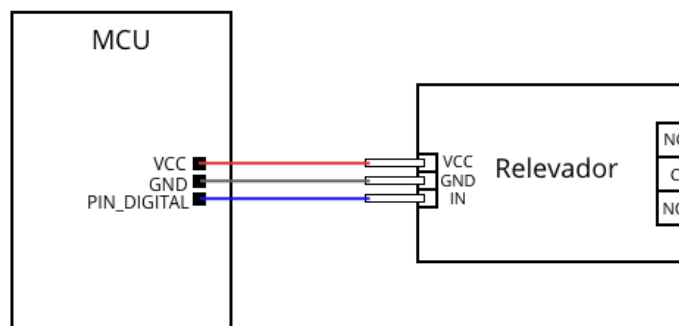


Figura 2-15. Diagrama de conexión entre el relevador y el MCU.

Secuencia numérica de pines del relevador, ver Tabla 2-6.

No.	Pin	Función
1	VCC	Corriente directa 5V
2	GND	Conexión a tierra
3	IN	Terminal de activación de señal

Tabla 2-6. Pines de conexión del relevador.

Características principales:

- 1 relevador.

³⁸ Es una aleación de níquel y titanio, y es el ejemplo más conocido de las llamadas aleaciones con memoria de forma

- 1 canal protegido.
- Led indicador.
- Voltaje necesario 5V
- Corriente 70 – 90 mA
- Voltaje de salida 250VCA o 30 VCD

2.3 Base de datos (Componente 3)

Debido a la necesidad del proyecto de comunicar el componente electrónico con una aplicación móvil, donde ambos elementos obtengan la misma información, se plantea el uso de una base de datos remota, permitiendo que esta pueda ser accedida desde cualquier parte del mundo, para la construcción de la misma, son necesarios 3 módulos independientes.

2.3.1 Motor (Back-End)

Es la capa del desarrollo de aplicaciones que siempre es ejecutada del lado del servidor, la cual recibe peticiones de los clientes (la capa de presentación) y contiene la lógica para devolver los datos de forma adecuada. El motor también incluye la base de datos que almacenará de forma persistente la información deseada.

En otras palabras el motor es toda la tecnología requerida para procesar la solicitud entrante, generar y enviar la respuesta al cliente, su arquitectura principal está conformada de 3 partes: El servidor, la aplicación y la base de datos (Abdullah & Zeki, 2014)

2.3.2 Servidor web

Un servidor es un computador u otro tipo de equipo informático encargado de suministrar información a una serie de clientes, que pueden ser tanto personas como otros dispositivos conectados a él. La información que puede transmitir es múltiple y variada: desde archivos de texto, imagen o vídeo, hasta programas informáticos, bases de datos, entre otras cosas. En otras palabras, es una computadora con mejores características que las de uso comercial, ya que están pensadas para trabajar en condiciones extremas.

Un servidor web, hablando en términos de software, es una computadora que utiliza el protocolo *HTTP*³⁹ para dar respuesta a peticiones que se realizan desde el exterior (un cliente).

Un servidor y un servidor web no son la misma cosa, un servidor web es creado dentro de un servidor. El servidor web ejecuta una aplicación que contiene la lógica sobre cómo responder a varias solicitudes basadas en el protocolo *HTTP* y el identificador

³⁹ Por sus siglas en inglés Hipertext Transfer Protocol, es el protocolo de comunicación que permite las transferencias de información en la red mundial de información.

uniforme de recursos (*URL*). A la conjunción entre el protocolo y la *URL* se le conoce como ruta y la combinación de ellos en función de una solicitud es denominada enrutamiento.

Las funciones que son procesadas en el enrutamiento pasarán a través de tuberías (middleware⁴⁰). Estas pueden modificar el objeto de la solicitud, consultar la base de datos, o de lo contrario procesar la solicitud entrante. Las funciones de tubería terminan pasando el control a la siguiente función, donde finalmente se envía la respuesta.

Finalmente, se llamará a una función de tubería que finaliza el ciclo de solicitud-respuesta enviando una respuesta *HTTP* hacia el cliente.

2.3.3 Base de datos

Un sistema gestor de bases de datos (*SGBD*) consiste en una colección de datos interrelacionados y un conjunto de programas para acceder a la información. La colección de datos, normalmente denominada base de datos, contiene información relevante para una empresa. El objetivo principal de un *SGBD* es proporcionar una forma de almacenar y recuperar su información, de tal forma que sea práctica y eficiente (Abraham Silberschatz, Henry F. Korth, 2006).

Las bases de datos pueden dividirse en 2 categorías: Bases de datos relacionales y bases de datos no relacionales.

Bases de datos relacionales (SQL). Consiste en un conjunto de tablas, a cada una de las cuales se le asigna un nombre exclusivo. Cada fila de la tabla representa una relación entre un conjunto de valores. De manera informal, cada tabla es un conjunto de entidades, y cada fila es una entidad. Dado que cada tabla es un conjunto de tales relaciones, hay una fuerte correspondencia entre el concepto de tabla y el concepto matemático de relación, del que toma su nombre el modelo de datos relacional (Abraham Silberschatz, Henry F. Korth, 2006). La mayor parte de los sistemas de bases de datos relacionales comerciales emplean el lenguaje *SQL*

Algunos ejemplos de gestores de base de datos relacionales son: *SQL Server*, *MySQL*, *PostgreSQL*, *Oracle*.

Bases de datos no relacionales (NoSQL). Es aquella que no usa el esquema tabular de filas y columnas que se encuentra en la mayoría de los sistemas de base de datos tradicionales. En su lugar, las bases de datos no relacionales usan un modelo de almacenamiento, que está optimizado para los requisitos específicos del tipo de datos que

⁴⁰ Es software que se sitúa entre un sistema operativo y las aplicaciones que se ejecutan en él. Funciona como una capa de traducción oculta para permitir la comunicación y la administración de datos en aplicaciones distribuidas

almacenan. Por ejemplo, los datos se pueden almacenar como pares clave/valor simple, documentos *JSON*⁴¹ o como un grafo que consta de bordes y vértices.

Un almacén de datos en este tipo de estructura, es denominado *documento*, administra un conjunto de campos de cadena con nombre y valores de datos. Normalmente, estos almacenes guardan la información en forma de documentos *JSON*. Cada valor del campo puede ser un elemento escalar, número, elemento compuesto, lista o una colección de elementos primarios y secundarios. Los datos de los campos de un documento se pueden codificar de varias formas, entre las que se incluyen *XML*⁴², *YAML*⁴³, *JSON*, *BSON*⁴⁴, o incluso se pueden almacenar como texto sin formato. Los campos de los documentos se exponen en el sistema de administración de almacenamiento, lo que permite que una aplicación consulte y filtre la información mediante el uso de los valores de estos campos.

Algunos ejemplos de gestores de base de datos no relacionales son: *MongoDb*, *Firebase*, *Google BigTable*, *Amazon SimpleDB*.

Una de las desventajas de bases de datos relacionales frente a las no relacionales es que poseen una estructura fija, por lo cual son mejores para almacenar información donde su estructura no cambie. Sin embargo, los requisitos del usuario y las características del software han evolucionado hasta incluir almacenes de datos, gestión de texto y procesamiento de flujo, este tipo de proceso tiene requisitos muy diferentes que el procesamiento de datos comerciales tradicional (Obay et al., 2014).

Para el presente trabajo de investigación, se utilizará una base de datos no relacional construida con el *SGBD Firebase*, principalmente porque la empresa Google, posibilita el acceso a un servidor web de forma gratuita, permitiendo el almacenamiento de la información y varias herramientas de integración con la misma.

2.4 Aplicación móvil (Componente 4)

Para la construcción de la aplicación móvil se analizan una serie de características que, trabajando en conjunto, brindarán una construcción más robusta del producto deseado.

⁴¹ De las siglas en inglés Javascript Object Notation, es un formato sencillo de texto para el intercambio de datos.

⁴² Por sus siglas eXtensible Markup Language, es un metalenguaje que permite definir lenguaje de marcas utilizado para almacenar datos en forma legible.

⁴³ Es un formato de serialización de datos legible por humanos, su característica principal es que no es un lenguaje de marcado.

⁴⁴ Es un lenguaje de intercambio de datos usado principalmente para su almacenamiento y transferencia en la base de datos *MongoDb*.

2.4.1 Aplicaciones de software

Software es un programa de computadora junto con su documentación asociada, generalmente se desarrollan para un cliente en particular o para el mercado en general (Sommerville, 2011).

Una aplicación de software es un programa o un conjunto de programas informáticos generalmente instalados en el sistema por el usuario y diseñadas para llevar a cabo un objetivo determinado y concreto, pueden ser de tipo lúdico, instrumental, comunicativo, informativo, entre otros.

Existen 2 tipos de aplicaciones de Software:

1. *Productos genéricos*. Consisten en sistemas independientes que se producen por una organización de desarrollo y se venden en el mercado abierto a cualquier cliente que desee comprarlos. Ejemplos de este tipo de productos incluyen software para PC, como bases de datos, procesadores de texto, paquetes de dibujo y herramientas de administración de proyectos, entre otros.
2. *Productos personalizados (o a la medida)*. Son sistemas que están destinados para un cliente en particular. Ejemplos de este tipo de software incluyen los sistemas de control para dispositivos electrónicos, sistemas escritos para apoyar cierto proceso empresarial y los sistemas de control de tráfico aéreo, entre otros.

Además del tipo de aplicaciones de software, existe una categorización para los productos a desarrollar:

1. *Aplicaciones independientes*. Se trata de sistemas de aplicación que corren en una computadora local, como una PC, e incluyen toda la funcionalidad necesaria y no requieren conectarse a una red. Ejemplos de tales aplicaciones son las de oficina en una PC, programas CAD, software de manipulación de fotografías, entre otros.
2. *Aplicaciones interactivas basadas en transacción*. Consisten en aplicaciones que se ejecutan en una computadora remota y a las que los usuarios acceden desde sus propias PC o terminales. Esta clase de aplicación también incluye sistemas empresariales, donde una organización brinda acceso a sus sistemas a través de un navegador Web o un programa de cliente de propósito específico y servicios basados en nube, como correo electrónico y compartición de fotografías.
3. *Sistemas de control embebido*. Se trata de sistemas de control de software que regulan y gestionan dispositivos de hardware. Algunos ejemplos de sistemas embebidos incluyen el software en un teléfono móvil (celular), el software en un horno de microondas para controlar el proceso de cocinado, entre otros.

4. *Sistemas de procesamiento en lotes.* Sistemas empresariales que se diseñan para procesar datos en grandes lotes. Los ejemplos de sistemas por lotes incluyen sistemas de facturación periódica, como los sistemas de facturación telefónica y los sistemas de pago de salario, entre otros
5. *Sistemas de entretenimiento.* Sistemas para uso sobre todo personal, que tienen la intención de entretener al usuario. La calidad de interacción ofrecida al usuario es la característica más importante de estos sistemas.
6. *Sistemas para modelado y simulación.* Sistemas que desarrollan científicos e ingenieros para modelar procesos o situaciones físicas, que incluyen muchos objetos separados interactuantes. Dichos sistemas a menudo son computacionalmente intensivos y para su ejecución requieren sistemas paralelos de alto desempeño.
7. *Sistemas de adquisición de datos.* Sistemas que desde su entorno recopilan datos usando un conjunto de sensores, y envían dichos datos para su procesamiento a otros sistemas. El software tiene que interactuar con los sensores y se instala regularmente en un ambiente hostil, como en el interior de un motor o en una ubicación remota.
8. *Sistemas de sistemas.* Son sistemas compuestos de un cierto número de sistemas de software. Algunos de ellos son producto del software genérico, como un programa de hoja de cálculo. Otros sistemas en el ensamble pueden estar especialmente escritos para ese entorno.

Es importante mencionar que estos conceptos son aplicados a cualquier tipo de Software: Aplicaciones de escritorio, aplicaciones móviles, aplicaciones web o aplicaciones embebidas.

El producto final que se propone desarrollar en el presente trabajo de tesis, pertenece a la categoría de *Producto personalizado*, mientras que la programación del microcontrolador pertenece a la sección de *Sistemas embebidos* y *Sistemas de adquisición de datos*, finalizando con la aplicación móvil que pertenece a la categoría *Aplicación interactiva basada en transacción*.

Complementos (Plugins)

Son una serie de funciones/métodos que sirven para simplificar una única tarea específica, añade una funcionalidad adicional o una nueva característica al software.

Los complementos que serán utilizados en el presente trabajo de tesis son los siguientes: *cupertino_icons* (v0.1.2), *generic_bloc_provider* (v1.0.9), *provider* (v3.1.0), *curved_navigation_bar* (v0.3.2), *google_sign_in* (v4.2.0), *firebase_storage* (v3.1.3),

flutter_localizations (v1.0.0), flutter_toast (v4.0.1), wave_progress_widget (v0.0.1), simple_animations (v1.3.11), fl_chart (v0.9.0).

Librerías de software

Son un conjunto de funciones/métodos muy concretos que sirven para simplificar tareas complejas. Pueden ser implementadas en el código fuente, siempre que se respeten los métodos proporcionados en la documentación de la misma, no hay necesidad de adaptar/modificar la estructura de una aplicación, a diferencia de los complementos, las librerías cuentan con más métodos de acceso y tienen mayor funcionalidad. Cuando se utiliza una librería, el desarrollador tiene el control y la librería es la que se adapta a su código.

Se utiliza la librería *Firebase Authentication (v0.14.0+5)* en el presente trabajo para llevar a cabo la autenticación dentro de la aplicación móvil.

Marcos de referencia (Frameworks)

Aporta una estructura completa para trabajar un proyecto, existen diferentes tipos de marcos de referencia que realizan diferentes funciones, dependerá del objetivo final, la decisión de utilizar uno u otro. En este tipo de herramientas, el usuario inserta bloques de código propio a una estructura predefinida, permitiendo implementar una lógica concreta de la aplicación. Es mucho más que una librería. Impone condiciones a la aplicación e incluso puede llegar a definir su arquitectura. Es un marco en donde se definen las piezas que lo componen y se siguen sus reglas.

Servicios Web (Web Services)

Los servicios web proporcionan un medio estándar de interoperación entre diferentes aplicaciones de software, que se ejecutan en una variedad de plataformas y/o marcos de referencia, se caracterizan por su gran interoperabilidad y extensibilidad, así como por sus descripciones procesables por máquina gracias al uso de *XML* y *JSON*.

Un servicio web puede describirse como un método para intercambiar/comunicar información entre dispositivos a través de una red (Halili & Ramadani, 2018).

Existen muchos modelos que permiten generar servicios web, como *RMI*, *CORBA* o *DCOM*, sin embargo, cuando se trata de seguridad o compatibilidad estas tecnologías suelen causar muchos problemas. Las aplicaciones modernas se basan en dos nuevos modelos que brindan la posibilidad de crear servicios web: *SOAP* y *REST* (Tihomirovs & Grabis, 2017).

SOAP (Protocolo de Acceso Simple de Objeto).

Su operación es la siguiente: un proveedor de servicios pública una descripción del servicio o una interfaz en el registro del servicio, de modo que el solicitante del servicio puede encontrar una instancia de servicio correcta y usarla. Para garantizar el transporte de datos en *SOAP*, se utilizan protocolos como *HTTP*, *SMTP*⁴⁵, entre otros, mientras que los datos se envían en formato *XML*.

REST (Estado Representacional de Protocolo de Transferencia).

Es un enfoque más reciente con respecto a *SOAP*, utiliza el protocolo *HTTP* para transmitir datos, mientras que los datos están encapsulados en formatos *XML*, *JSON*, entre otros. Simplifica el acceso a los servicios web mediante el uso de los estándares existentes y conocidos en lugar de agregar nuevas capas de procesamiento de datos en la pila de transmisión y comunicación. Por lo tanto, *REST* tiende a ser una alternativa más ligera al protocolo *SOAP*. Los servicios *REST* se basan en recursos autodefinidos donde se utiliza el protocolo *HTTP* para llegar a ellos. Este servicio se proporciona como un recurso que puede ser identificado por URL.

En el presente trabajo de tesis se optó por utilizar este protocolo, ya que los complementos que serán utilizados *Firebase (v3.1.3)* y *Google Sign In (v4.2.0)*, retornan la información en formato *JSON* utilizando este protocolo.

Interfaz de programación de Aplicaciones (API)

Una *API* es un conjunto de funciones y procedimientos que cumplen una o muchas funciones con el fin de ser utilizadas por otro software sin necesidad de programarlas de nuevo. Las siglas *API* vienen del inglés Application Programming Interface. En términos de programación es una capa de abstracción.

Las *API* modernas son formas flexibles de proyectar sus capacidades a una audiencia fuera de su propio equipo de desarrollo. Cuando se hace correctamente, permiten a las empresas innovar más rápido y llegar a nuevas audiencias (Jensen, 2015).

Es importante mencionar que una *API* siempre podrá ser un servicio web, pero un servicio web no siempre puede ser una *API*.

2.4.2 Paradigmas en el desarrollo de aplicaciones

Dentro del desarrollo de aplicaciones de software, existen diferentes paradigmas que permiten la generación de un producto, cada uno de ellos ofrece una particularidad y singularidad que es importante identificar, dependiendo del producto deseado, el tiempo, calidad y costo pueden ser factores importantes para inclinarse por uno u otro desarrollo,

⁴⁵ Protocolo para Transferencia Simple de Correo. Es un protocolo de comunicación que permite el envío de correos electrónicos en internet.

en general se pueden clasificar en 4: Aplicaciones nativas, Aplicaciones multiplataforma, Aplicaciones híbridas y Aplicaciones generadas.

Aplicaciones Nativas

Se centra en la plataforma donde el producto de software será instalado. En este paradigma los desarrolladores regularmente utilizan un lenguaje base y herramientas para desarrollar la aplicación, por ejemplo: *Java*, *Kotlin*, o *Dart* para *Android*; *Object-C* y *Swift* para *iOS*; *C#* y *Visual Basic* para *Windows*, entre otras (Que et al., 2017).

Ventajas:

- Máximo rendimiento de los dispositivos que utilizan el software.
- Mejora la experiencia del usuario.

Desventajas:

- Se debe crear una aplicación diferente para cada plataforma.
- El costo y mantenimiento depende del número de versiones que se manejen para el software.

Aplicaciones Multiplataforma

El principal concepto de las soluciones multiplataforma es desarrollar la aplicación una vez, y ejecutarla en cualquier lugar (El-Kassas et al., 2017). Algunos de los lenguajes de programación que permiten el desarrollo de aplicaciones multiplataforma son: *JAVA*, *JavaScript*, *Python*, *Ruby*, *Dart*.

Ventajas:

- Mantenimiento sencillo.
- Costo de desarrollo menor.
- Código reutilizable.
- Funciona igual en todas las plataformas.

Desventajas:

- No son tan flexibles
- La experiencia de usuario es menor en comparación con una aplicación nativa.

Para la construcción de la aplicación móvil, se ha seleccionado esta opción, debido a que, uno de los objetivos del presente proyecto, es desarrollar una aplicación que funcione con *Android*, esto deja fuera las aplicaciones para el sistema operativo *iOS*, sin

embargo, al crear una aplicación multiplataforma nos permitirá, como trabajo futuro realizando cambios menores, la posibilidad de realizar el desarrollo en este entorno faltante.

Aplicaciones Híbridas

Generan aplicaciones específicas de la plataforma a partir de una base de código común en tiempo de compilación, como soluciones basadas en modelos y compilación cruzada (El-Kassas et al., 2017). Estas aplicaciones combinan tecnología web como HTML, CSS y JavaScript que no son aplicaciones móviles, de escritorio o web verdaderamente nativas, son empaquetadas utilizando un WebView⁴⁶ ejecutado dentro de un contenedor nativo, esto permite que funcione como una aplicación multiplataforma, las aplicaciones híbridas también pueden ser escalables entre sistemas operativos, algunos ejemplos son:

1. *Aplicación híbrida entre móvil y web.* Marcos de referencia que permiten desarrollar este tipo de aplicaciones son: Ionic, Xamarin, Phonegap, React Native, Cordova, entre otros.
2. *Aplicación híbrida entre web y escritorio.* Marcos de referencia que permiten el desarrollo de este tipo de aplicaciones son: Meteor, Electron, NWjs, entre otros.

Ventajas:

- Su funcionamiento es multiplataforma.
- Es posible utilizar la misma base de código (móvil, web y escritorio).
- Su mantenimiento es menos complicado que aplicaciones nativas o multiplataforma.

Desventajas:

- Funciones muy limitadas.
- Generalmente requieren conexión a internet para su funcionamiento.

Aplicaciones Generadas

Son aplicaciones donde el desarrollo utiliza técnicas y lenguajes específicos de la herramienta, luego se genera la aplicación en el lenguaje de la plataforma destino, por último, se compila con herramientas nativas. También se les conoce como aplicaciones falsas nativas.

Marcos de referencia que operan bajo esta filosofía son: GeneXus, Scratch, App Inventor.

⁴⁶ Es una aplicación que hace de visor de páginas web.

Ventajas:

- La curva de aprendizaje es rápida.
- El tiempo de desarrollo es corto.

Desventajas:

- El nivel de personalización no es muy amplio.
- Son limitadas las operaciones que se pueden realizar con estos programas.

2.4.3 Patrones de diseño

Un patrón de diseño define una posible solución correcta para un problema dentro de un contexto dado, describiendo las cualidades invariantes de todas las soluciones.

La base de los patrones de diseño llevados a cabo en los sistemas informáticos toma como base la Arquitectura, la forma en la que se hace un análisis y se empaquetan ciertos elementos que son recurrentes.

Cada patrón describe un problema que ocurre infinidad de veces en el entorno, así como la solución al mismo, de tal modo que podamos utilizar esta solución un millón de veces más adelante sin tener que volver a pensarla otra vez (Blancarte, 2016).

Existen 3 tipos de patrones:

1. *Patrones creacionales.* Son patrones de diseño relacionados con la creación o construcción de objetos. Estos patrones intentan controlar la forma en que los objetos son creados implementando mecanismos que eviten la creación directa de objetos, algunos patrones que se encuentran en esta categoría son: Patrón Método de Fábrica, Patrón de Fábrica Abstracta, Patrón Semifallo, Patrón Constructor, Patrón Prototipo y Patrón Conjunto de Objetos.
2. *Patrones estructurales.* Son patrones que tienen que ver con la forma en que las clases se relacionan con otras clases. Estos patrones ayudan a dar un mayor orden a las clases, ayudando a crear componentes más flexibles, algunos patrones dentro de esta categoría son: Patrón Adaptador, Patrón Puente, Patrón Compuesto, Patrón Decorador y Patrón Fachada.
3. *Patrones de comportamiento.* Son patrones que están relacionados con procedimientos y con la asignación de responsabilidad a los objetos. Los patrones de comportamiento engloban también patrones de comunicación entre ellos, algunos ejemplos son: Patrón Iterador, Patrón de Mando, Patrón Observador,

Patrón Método de Plantilla, Patrón Estrategia, Patrón Modelo Vista Controlador, Patrón Componentes de Lógica de Negocios.

Patrón de Componentes de Lógica de Negocios (BLoC)

Es un patrón de diseño desarrollado por el equipo de desarrollo de Google, presentado por Paolo Soares y Cong Hui, en la *DartConf* 2018. Este patrón facilita compartir código entre plataformas (móvil, web, escritorio, servidor y más), separando la lógica de negocio de la interfaz gráfica en uno o más *BLoCs* (Suri, 2018). Un *BLoC* es una clase que contiene la lógica de un componente de una aplicación, ayudando a administrar el estado y realizar el acceso a los datos desde un lugar centralizado.

La arquitectura de un *BLoC* está compuesta por 4 elementos, ver Figura 2-16.



Figura 2-16. Arquitectura BLoC.

1. **Vista.** Interface final que el usuario visualiza cuando un componente es cargado, en este apartado, se concentra toda la interacción con las vistas, es importante recordar que un componente puede tener N subcomponentes, a pesar de que visualmente se puede observar un solo resultado.
2. **BLoC.** Concentra la lógica del negocio, se establecen todas las acciones que pertenecen a un componente. Es importante que toda vista que contenga una lógica de negocio deberá contener su propio *BLoC* y encapsule cada uno de los casos de uso pertenecientes a ella.
3. **Repositorio.** Dentro de una aplicación compleja, es común utilizar diferentes fuentes de datos. La capa de repositorio, deberá ser capaz de cambiar la fuente de forma dinámica, en este apartado se agregan las clases que se conectan a dicha fuente pudiendo ser una *API*, Bases de datos, *Endpoints*⁴⁷, entre otros.
4. **Datos.** En esta capa son establecidos los modelos que ayudan a manipular los datos, cada modelo cuenta con los atributos necesarios de cada clase y el comportamiento de los mismos.

En el presente trabajo de tesis, se utilizará el patrón *BLoC* para generar la arquitectura de la aplicación, esto posibilita una escritura de código fuente más limpia y

⁴⁷ Es un dispositivo informático remoto que se comunica con una red a la que está conectado.

ordenada, permitiendo reutilizar diferentes bloques de código y haciendo el mantenimiento sencillo y eficiente.

2.4.4 Capa de presentación (Front-End)

Es la capa dentro del desarrollo de aplicaciones que se enfoca en el usuario, recolecta datos de entrada, y permite, qué acciones pueden ser realizadas, en otras palabras, es la interacción entre el cliente y la aplicación, comúnmente llamada capa de presentación.

Su principal finalidad es simplificar y hacer más eficiente la interacción entre una persona y la aplicación desarrollada, el código siempre es ejecutado del lado del cliente.

Dentro de la capa de presentación hay 3 técnicas que deben ser consideradas para el desarrollo de software de calidad: Los patrones de diseño, la experiencia de usuario y la interface de usuario. Las técnicas mencionadas anteriormente son tomadas en consideración para la implementación de la capa de presentación del presente trabajo de tesis, a continuación, se analizan cada una de ellas.

2.4.5 Experiencia de Usuario (UX)

Se trata de la experiencia que tiene un usuario al utilizar un producto o servicio ya sea digital o físico. Se enfoca en que las acciones que realice el usuario sean intuitivas y con la menor cantidad de pasos para realizar una acción (W. Lidwell, 2007).

Para llevar a cabo una buena experiencia de usuario es recomendable tomar en cuenta los siguientes elementos:

1. *Accesibilidad.* Los objetos y entornos deben diseñarse para que puedan ser utilizados sin modificaciones por tantas personas como sea posible.
2. *Asequibilidad.* Cuando la disponibilidad de un objeto o entorno se corresponde con su función prevista, el diseño tendrá un rendimiento más eficiente y será más fácil de utilizar.
3. *Alineación.* Los elementos de un diseño deben alinearse con uno o más elementos, esto crea un sentido de unidad y cohesión, lo que contribuye a la estética general y la estabilidad percibida del diseño.
4. *Cierre.* Afirma que siempre que sea posible, las personas tienden a percibir un conjunto de elementos individuales como un patrón único y reconocible, en lugar de múltiples elementos individuales. La tendencia a percibir un patrón es tan fuerte que las personas cerrarán los huecos y completarán la información que falta para completar el patrón si es necesario.

5. *Disonancia cognitiva*. Es el estado de malestar mental que ocurre cuando las actitudes, pensamientos o creencias de una persona entran en conflicto. Si dos ideas coinciden entre sí, hay consonancia y se produce un estado de comodidad. Si dos cogniciones no están de acuerdo el uno con el otro, hay disonancia y se produce un estado de incomodidad.
6. *Comparación*. La gente entiende la forma en que funciona el mundo al identificar las relaciones y los patrones en o entre los sistemas. Uno de los métodos más poderosos para identificar y entender estas relaciones es representar la información de manera controlada para que se puedan hacer comparaciones.
7. *Confirmación*. Es una técnica utilizada para acciones críticas, entradas o comandos. Proporciona un medio para verificar que una acción o entrada es intencional y correcta antes de ser ejecutada.
8. *Control*. Las personas deben poder ejercer control sobre lo que hace un sistema, pero el nivel de control debe estar relacionado con su competencia y experiencia en el uso del sistema.

2.4.6 Interface de Usuario (UI)

Alude a la interfaz con la que interactúan los usuarios, es definida desde la planeación del producto y representa los puntos de contacto entre un elemento digital o análogo y el usuario. Se enfocan en brindar un diseño y ambiente agradable a la vista, con colores que sean equilibrados y vayan con la filosofía del diseño que se desea realizar (W. Lidwell, 2007).

Para llevar a cabo una buena experiencia de usuario es recomendable tomar en cuenta los siguientes elementos:

1. *Color*. Se utiliza en el diseño para atraer la atención, agrupar elementos, indicar la inclinación y mejorar la estética.
2. *Efecto Estética – Usabilidad*. Los diseños estéticos son percibidos como más fáciles de utilizar que los diseños menos estéticos.
3. *Sesgo atractivo*. Las personas atractivas son generalmente percibidas más positivamente que las personas no atractivas.
4. *Sesgo a cara de bebés*. Existe una tendencia a ver a las personas y a las cosas con características de cada de bebé como más ingenuas, indefensas y honestas que aquellas con rasgos maduros.

5. *Cortar en pedazos*. Una técnica de combinar muchas unidades de información en una cantidad limitada de unidades o trozos, para que la información sea más fácil de procesar y recordar. Por ejemplo: la mayoría de las personas pueden recordar una lista de 5 palabras durante 30 segundos, pero pocas pueden recordar una lista de 10 palabras durante 30 segundos.
6. *Consistencia*. Los sistemas son más utilizables y comprensibles cuando partes similares se expresan de manera similar.
7. *Constancia*. Las personas tienden a percibir los objetos como constantes e invariables, a pesar de los cambios en la perspectiva, la iluminación, el color o el tamaño. Por ejemplo, una persona vista a distancia produce una imagen más pequeña en la retina que a la misma persona de cerca, pero la percepción del tamaño de la persona es constante.

2.4.7 Aplicaciones Móviles

Son programas de software desarrollados específicamente para dispositivos móviles como teléfonos inteligentes, tabletas o asistentes digitales personales.

El propósito de estas aplicaciones abarca una gran variedad de objetivos, desde la utilidad, productividad, navegación, medición, entretenimiento, estado físico, comunicación, entre muchas otras. A diferencia de los sitios web, una aplicación móvil suele tener un alcance más pequeño, ofrece mayor interactividad y presenta información más específica en un formato fácil e intuitivo de interpretar.

Las aplicaciones móviles pueden ser descargadas desde diferentes plataformas, dependiendo del sistema operativo para el cual fueron desarrolladas, como se mencionó en el tema 2.4, la elección del paradigma a desarrollar va de la mano con el público al cual va dirigida la aplicación.

Estas aplicaciones son programas empaquetados que son instalados mediante una tienda de aplicaciones o de forma manual con el consentimiento del usuario. Funcionan como una aplicación de escritorio, creando un acceso directo en el móvil, permitiendo al usuario decidir cuándo utilizarla.

Para el presente trabajo de investigación se propone crear una aplicación móvil que funcione en el sistema operativo Android y opere como una aplicación nativa, utilizando un marco de referencia de nueva generación llamado *Flutter*.

2.4.8 Marco de desarrollo Flutter

Flutter es un *SDK*⁴⁸ creado por la empresa *Google* que ayuda a la construcción de aplicaciones nativas para móviles, web y escritorio a partir de una sola base de código (Google, 2018). Incluye un marco de referencia moderno de estilo reactivo, un motor de renderizado 2D, *widgets*⁴⁹ listos y herramientas de desarrollo, permitiendo depurar, construir, diseñar y probar aplicaciones, el lenguaje de programación con el que se trabaja es *Dart*⁵⁰.

Arquitectura de Flutter

Flutter está basado en un sistema de capas, cada una construyéndose sobre la capa anterior, ver Figura 2-17:

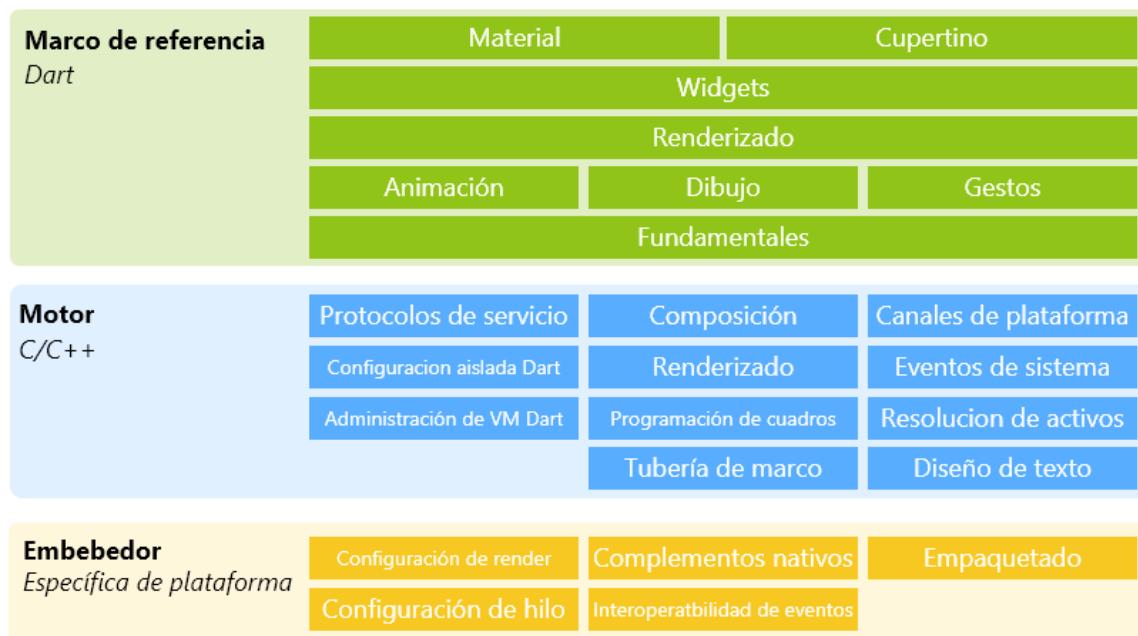


Figura 2-17. Arquitectura de Flutter.

El objetivo del diseño por capas es ayudar a la escritura menor de código. Por ejemplo, la capa *Material* de la Figura 2-17, se construye componiendo *widgets* básicos a partir de la capa de *widgets* y la capa de *widgets* se construye organizando los objetos de nivel inferior a partir de la capa de renderizado.

⁴⁸ Kit de herramientas de desarrollo que permite al usuario crear una aplicación informática para un sistema concreto.

⁴⁹ Son elementos básicos de una aplicación en Flutter, es una declaración inmutable del código.

⁵⁰ Es un lenguaje de programación de código abierto que utiliza un lenguaje estructurado pero flexible, desarrollado por la empresa Google.

Dentro de la arquitectura de *Flutter*, ver Figura 2-17, en el desarrollo de una aplicación se trabaja con la capa superior perteneciente al marco de referencia. La capa intermedia *Motor* y la inferior *Embebedor* realizan operaciones internas para trasladar código escrito en el lenguaje *Dart* hacia *C* y posteriormente compilarlo correctamente en la plataforma del sistema operativo del computador que ejecuta las instrucciones.

En Flutter todo es un Widget

A diferencia de otros marcos de referencia que separan vistas, controladores, capas y otras propiedades, *Flutter* tiene un modelo de objeto unificado y consistente llamado widget. Los widgets forman una jerarquía basada en composición, cada widget se integra en el interior y hereda las propiedades de su padre, el widget raíz sirve como entrada principal al programa a ejecutar.

El marco de referencia *Flutter*, contiene una gran cantidad de *widgets* diseñados para su uso inmediato, sin embargo, es posible crear widgets personalizados y construidos por el usuario para que reaccionen a un comportamiento deseado, los *widgets* son clasificados en 2 tipos, ver Figura 2-18.

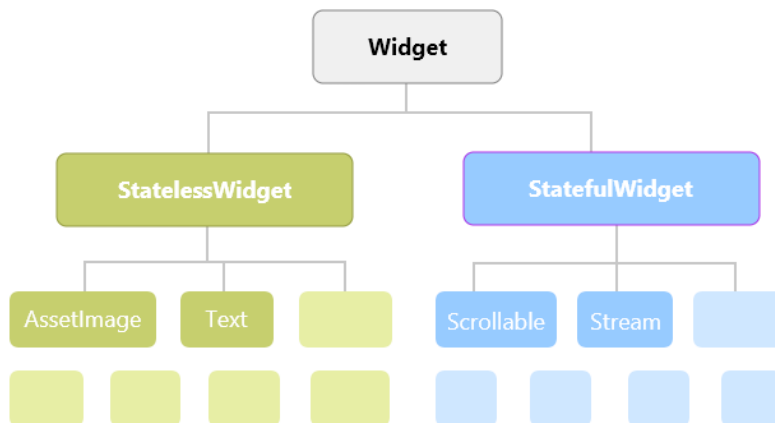


Figura 2-18. Clasificación de Widgets en Flutter.

StatelessWidget. Son conocidos como *widgets* sin estado, estos elementos permanecen inmutables una vez que son creados, en la Figura 2-18 se observa el widget *Text* que sirve para mostrar una cadena de texto dentro de la aplicación.

StatefulWidget. Son *widgets* interactivos que, si pueden cambiar su estado dependiendo de la funcionalidad programada, en la Figura 2-18 se observa el widget *Scrollable*, donde su función implementa un modelo de interacción para un widget desplazable, permitiendo modificar de forma dinámica sus valores según el usuario interactúa con él.

Un *StatelessWidget* puede pasar a ser un *StatefulWidget* dependiendo de su funcionalidad, pero no viceversa. Por ejemplo, en la Figura 2-18, se observa el *StatelessWidget Text*, si la operación deseada, es que un texto una vez que la aplicación ha sido inicializada al presionar un botón sea cambiado, el tipo de *widget* necesario es un *StatefulWidget*, porque será necesario un cambio de estado para modificar el texto, si no se desea que el texto sufra ninguna modificación, el widget recomendado es *StatelessWidget*.

Flujos (Streams)

Son una secuencia de eventos asíncronos, que permiten mejorar el funcionamiento de los *widgets*, el *flujo* es definido en un lugar y en otro escucha la adición de los datos para actualizar algún *widget*, son similares a los *observables*⁵¹ del lenguaje *JAVA* o a los *Sockets*⁵² de *C*.

Tomando en consideración que se utilizará *BLoC* como patrón de diseño para el desarrollo de toda la aplicación móvil, la implementación de los flujos deberá tener la siguiente estructura, ver Figura 2-19.

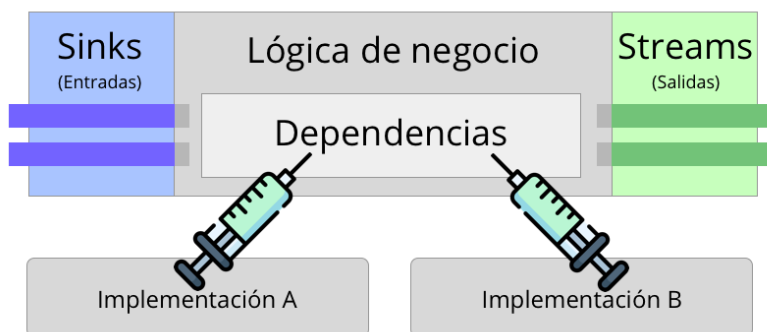


Figura 2-19. Uso de flujos con el patrón de diseño BLoC.

Como fue mencionado al inicio del capítulo, el presente proyecto se subdivide en 4 componentes principales, después de analizar y comparar las diferentes alternativas existentes, se han tomado las siguientes decisiones:

1. Prototipo hidropónico (Componente 1). Se utilizará la técnica de Flujo y relujo (tema 2.1.1), como base para la construcción del sistema hidropónico.
2. Automatización hidropónica (Componente 2). Fueron seleccionados los siguientes componentes para automatizar el proceso hidropónico: los sensores (tema 2.2.3) *AM2302*, *HC-SR04*. El *relevador 5v* que será utilizado como un actuador (tema 2.2.4). Los módulos (tema 2.2.2) *RTC AT24C32*, *pH-4502C* y finalmente *NodeMCU* basado en el microcontrolador (tema 2.2.1) *ESP8266*.

⁵¹ Representan una colección de futuros valores, es decir, información que probablemente no estemos recibiendo en un sistema informático.

⁵² Es un método para la comunicación entre un programa del cliente y un programa del servidor en la red.

3. Base de datos (Componente 3). Se utilizará una base de datos no relacional basada en el *SGBD* Firebase (tema 2.3.3).
4. Aplicación móvil (Componente 4). Su arquitectura estará basada en el patrón de diseño *BLoC* (tema 2.4.3), el marco de referencia para su desarrollo será *Flutter* (tema 2.4.8) y finalmente el diseño de la aplicación tomará como base los conceptos de experiencia e interface de usuario (temas 2.4.5 y 2.4.6).

Cada uno de los componentes previamente mencionados, serán analizados e implementados en los siguientes capítulos.

CAPÍTULO 3

Análisis y diseño

Capítulo 3 Análisis y diseño

La arquitectura propuesta para dar solución al presente trabajo de tesis consta de 4 componentes (Estructura hidropónica, Automatización, Base de datos, Aplicación móvil), ver Figura 3-1. Cada componente determina un apartado que brinda la solución a una problemática específica.

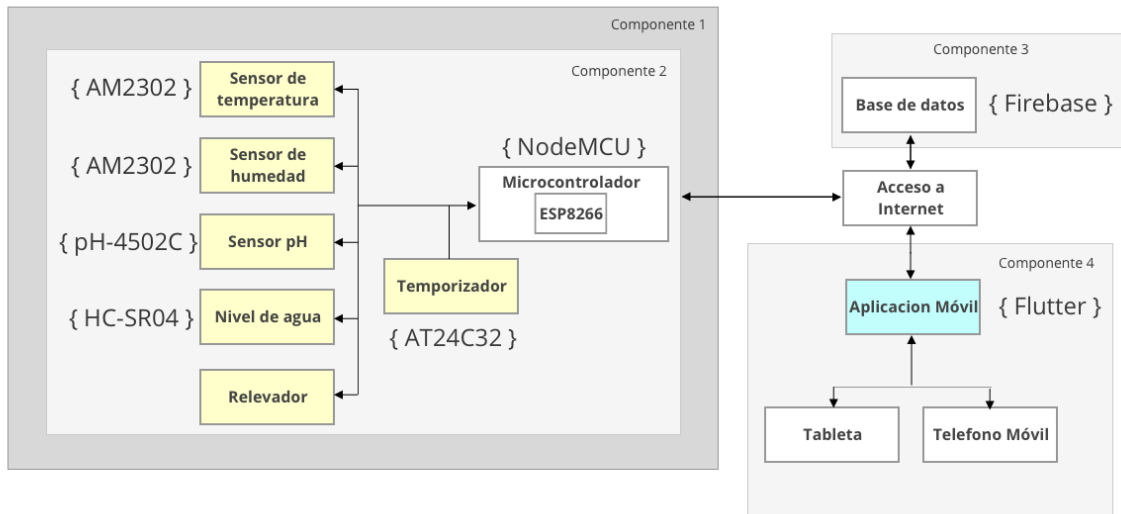


Figura 3-1. Arquitectura de componentes del proyecto Tekax

1. **Estructura hidropónica (Componente 1).** Comprende la técnica a utilizar, y la forma en la que fue construido el prototipo.
2. **Automatización (Componente 2).** Abarca el apartado del circuito electrónico: los sensores, actuadores y microcontroladores, que son utilizados dentro del proyecto para recolectar la información del entorno y automatizar procesos.
3. **Base de datos (Componente 3).** Comprende la configuración y estructura de los documentos dentro de la base de datos.
4. **Aplicación móvil (Componente 4).** El proceso de desarrollo de la aplicación móvil, la metodología a utilizar y el alcance de la misma.

3.1 Prototipo hidropónico (Componente 1)

Todo el prototipo hidropónico esta conformado por una serie de elementos que se describirán a continuación.

3.1.1 Estructura de soporte hidropónico

Esta estructura sirve como soporte para la cama de cultivo, el microcontrolador, los sensores y todos los elementos físicos que son utilizados para el control del proceso hidropónico propuesto.

La técnica hidropónica a utilizar, tiene por nombre *Flujo y Reflujo*, considerada una de las mejores debido a que permite optimizar de mejor forma recursos como el agua y los nutrientes (Sayara et al., 2016), fue documentada en el punto 2.1.1.

Para dar soporte a la cama de cultivo, se crea un trípode de madera, permitiendo que la movilidad del prototipo sea más sencilla. Tomando como base las consideraciones obtenidas en tema 2.1.2, se utilizan 4 fajillas de madera de 150 cm de 1/2" de 10 cm de ancho, y 3 fajillas de madera de 50 cm de 1/2" de 10 cm de ancho, además de 18 pijas de 2", tuerca de 2", ver Figura 3-2.

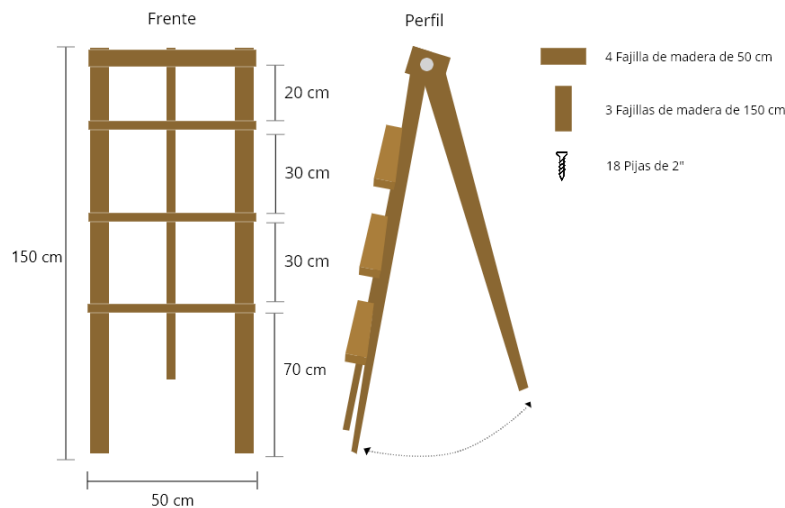


Figura 3-2. Estructura de soporte hidropónico.

Para la construcción de la cama de soporte se realizan los siguientes pasos:

1. Colocar dos fajillas de madera de 150 cm verticalmente en forma paralela con una separación de 50 cm, en la parte superior de las 2 fajillas verticales se coloca una fajilla en forma horizontal de 50 cm por encima de estas, posteriormente las 3 fajillas son fijadas con 2 pijas de 2" en cada extremo, permitiendo formar un marco.

2. Se toma de referencia la fajilla horizontal superior y se miden *20 cm* hacia abajo, donde se coloca una segunda fajilla de *50 cm* en forma paralela a la fajilla superior.
3. Se deja un espacio de *30 cm*, entre la segunda fajilla y la tercera, para colocarla de forma paralela a las 2 anteriores, dando un efecto de escalera.
4. La última fajilla también se coloca a *30 cm* de distancia de la 3era fajilla, dejando un espacio de *70 cm* entre esta y el suelo.
5. Por la parte trasera de la estructura, en la parte superior se colocan 2 trozos de madera de *10 cm x 10 cm*, de forma paralela con una separación de *2.5 cm*.
6. Entre las 2 piezas de madera colocadas en el punto 5, se introduce la fajilla de *150 cm* faltante, se realiza una perforación, posibilitando que una tuerca de *2"* atraviese las piezas de madera del punto 5 y la fajilla de madera del punto 6, esta acción permitirá que la última fajilla colocada tenga movimiento, permitiendo crear un trípode.

3.1.2 Cama de cultivo

La cama de cultivo es la sección del prototipo hidropónico donde son colocadas las semillas germinadas, son rodeadas por una esponja e introducidas en un recipiente plástico, posteriormente se colocan en un espacio vacío de la misma, permitiendo la absorción de los nutrientes que fluyen en su interior, la oxigenación de las plantas y el flujo de agua constante de forma adecuada.

Los materiales necesarios para la construcción de la cama de cultivo propuesta, son los siguientes: 3 tubos *PVC* de *2"* de *60 cm*, 6 tapones para cople de *2"*, 6 codos de *90°* para tubo *PVC* de *1/2"*, 2 de tubos *PVC* de *1/2"* de *30 cm*, 2 tubos *PVC* de *1/2"* de *10 cm*, pegamento para tubo *PVC*, 1 llave de paso de agua para tubo *1/2"*, 1 adaptador de *PVC 1/2"* a manguera, taladro y broca para cerradura de *2"*, ver Figura 3-3.

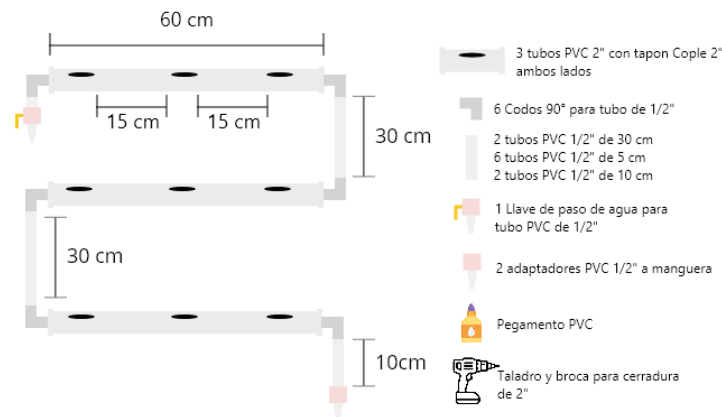


Figura 3-3. Cama de cultivo

La construcción de la cama de cultivo, se lleva a cabo realizando las siguientes actividades:

1. Los 3 tubos *PVC* de 2" de 60 cm, reciben perforaciones con una broca de 2", la separación recomendada entre cada perforación es de 15 – 20 cm, por lo tanto, solo se realizan 3.
2. Se colocan los tapones de cople 2" en cada extremo de los 3 tubos y son sellados utilizando pegamento para *PVC*.
3. Al centro de cada tapón de cople 2" se realiza una perforación circular y se introduce un tubo de *PVC* de 1/2" de 5cm de largo.
4. Al anexo de tubo agregado en el punto 3, conecta a un codo 90°.
5. Se colocan los 3 tubos *PVC* del paso uno, de forma vertical.
6. En el tubo *PVC* 2" de 60 cm de la parte superior, a su izquierda, se agrega una llave de paso, que permitirá regular la cantidad agua entrante al prototipo hidropónico.
7. A la llave de paso se agrega un adaptador de tubo *PVC* 1/2" a manguera.
8. Al tubo mencionado en el paso 6, en su derecha, al codo de 90° se anexa un tubo *PVC* de 1/2" de 30 cm.
9. Al tubo *PVC* 2" de 60 cm ubicado en el centro, del lado derecho, se conecta el tubo anexado en el paso 8, al codo 90°.
10. Al tubo del paso 9, en su lado izquierdo, al codo 90°, se anexa un tubo de *PVC* de 1/2" de 30 cm.
11. Al tubo *PVC* 2" de 60 cm ubicado en la parte inferior de la estructura hidropónica, del lado izquierdo, se conecta el tubo anexado en el paso 10.
12. El tubo del paso 11, en su lado derecho, al codo 90°, se agrega un tubo *PVC* de 1/2" de 10 cm.
13. Al tubo agregado del paso 12, se anexa un adaptador de tubo de 1/2" a manguera.

3.1.3 Contenedor y sistema de drenaje del prototipo hidropónico

El sistema de drenaje sirve para captar el flujo de agua sobrante y redirigirlo hacia un contenedor con la solución nutritiva para posteriormente ser reutilizada.

El prototipo propuesto, utiliza 2 mangueras de 1/2" de 1.5m de largo y un contenedor rectangular oscuro, ver Figura 3-4, evitando así, la generación de algas en la solución nutritiva, siguiendo las recomendaciones del tema 2.1.3.



Figura 3-4. Contenedor y sistema de drenaje del prototipo hidropónico.

3.1.4 Prototipo hidropónico propuesto

En conjunto los elementos previamente mencionados: la estructura de soporte hidropónico, la cama de cultivo, el sistema de drenaje y el contenedor de la solución nutritiva, forman el prototipo hidropónico del presente trabajo de investigación, ver Figura 3-5.

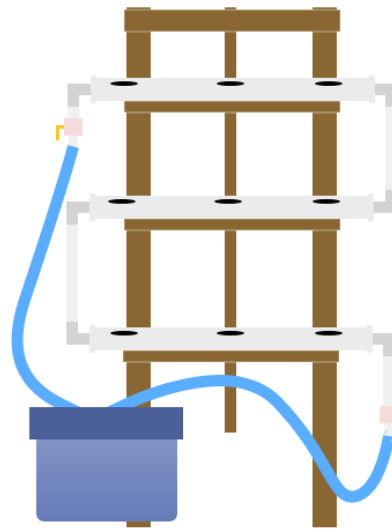


Figura 3-5. Prototipo hidropónico propuesto.

3.2 Automatización hidropónica (Componente 2)

En el presente trabajo de tesis, los elementos que serán monitorizados constantemente son: pH, temperatura, humedad, nivel de agua y el flujo constante de la misma.

Cada cultivo necesita umbrales que garanticen condiciones óptimas para un crecimiento adecuado de las plantas, por ejemplo: la lechuga orejona, necesita que el pH de la solución nutritiva permanezca entre 5.5 – 6.6, la temperatura del ambiente entre los 24°C - 30°C, la humedad relativa entre el 55% - 70% (Zagal et al., 2016).

Para medir los elementos que necesitan supervisión dentro del cultivo hidropónico y monitorizar de manera correcta que permanezcan en sus respectivos umbrales, se utilizarán los componentes que se muestran en la Tabla 3-1.

Componente	Cantidad	Descripción
NodeMCU	1	Microcontrolador reprogramable que permite conectar diferentes sensores, módulos y actuadores, cuenta con un procesador Tensilica.

AM2302	1	Sensor de temperatura y humedad, su rango de medición de temperatura es -40°C a 80°C , y humedad de $0 - 100\%$.
HC-SR04	1	Sensor ultrasónico, es un emisor y receptor de ultrasonidos que trabaja una frecuencia de 40 KHz .
AT24C32	1	Es un RTC (Real Time Clock), funciona con una batería de 3V y permite contar horas, minutos y segundos de forma precisa.
pH-4502C	1	Es un electrodo que sirve para medir el nivel de pH de una solución, mide rangos desde $0 - 14\text{ pH}$.
Relevador 5v	1	Es un interruptor que puede ser accionado electrónicamente.

Tabla 3-1. Componentes a utilizar para automatización y control del sistema hidropónico.

El diagrama de conexión de cada uno de los componentes mencionados en la Tabla 3.1 y su integración con el microcontrolador, puede ser visualizado en la Figura 3-6.

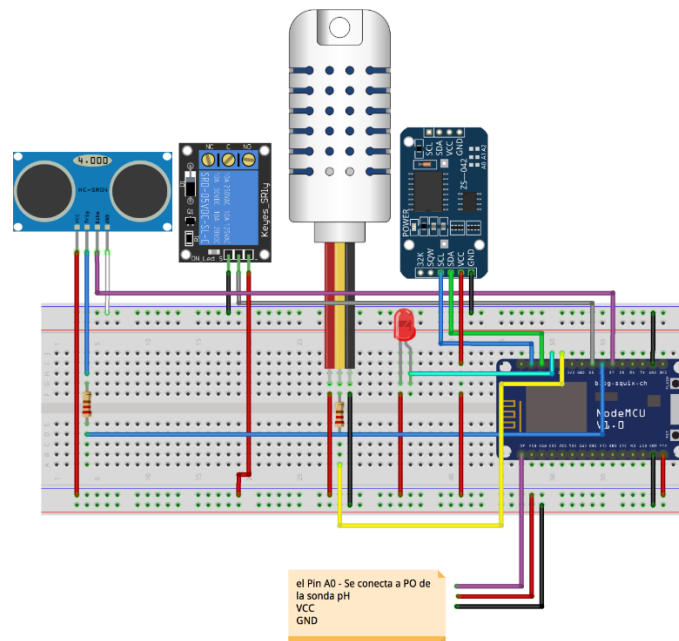


Figura 3-6. Diagrama de conexión entre módulos, sensores y microcontrolador del componente 2.

3.2.1 NodeMCU y el sensor AM2302

La temperatura y humedad son elementos que son monitorizados en el presente proyecto, para llevar a cabo esta tarea, ha sido seleccionado el sensor *AM2302*, en el tema 2.2.3 podemos encontrar los pines de conexión y sus características.

Siguiendo el diagrama de conexión de módulos, sensores y microcontrolador, ver Figura 3.6. Se conecta el pin *DATA* del sensor *AM2302* al pin *D4* del *NodeMCU* y se agrega una resistencia de 220 Ohms al mismo, las conexiones a tierra y corriente que el sensor

AM2302 necesita, son conectadas a las líneas generales que se crearon del módulo *NodeMCU*.

Se presenta el Algoritmo 3-1, que muestra en forma generalizada la secuencia de instrucciones necesarias para obtener las lecturas de temperatura y humedad del ambiente:

Inicio

- Incluir librería DHT
- Asignar el pin del *NodeMCU* que se reserva para el sensor AM2302
- Definir el tipo de sensor DHT a utilizar
- Crear variables globales para almacenar temperatura y humedad
- Inicializar el sensor DHT
- Obtener valores de temperatura y humedad del sensor
 - Si temperatura y humedad != nulo entonces
 - Actualiza variables de temperatura y humedad
 - Si no
 - Imprime error

Fin

Algoritmo 3-1 Módulo NodeMCU y el sensor AM2302.

3.2.2 *NodeMCU* y el módulo RTC AT24C32.

Dentro del proyecto existen procesos que necesita ser ejecutado en lapsos de tiempo interrumpidos, por ejemplo: la circulación de la solución nutritiva por todo el prototipo hidropónico y el almacenamiento de información en la base de datos, ambas son actividades que dependen del tiempo, para llevar a cabo las operaciones mencionadas con anterioridad se ha seleccionado el módulo *AT24C32*, en el tema 2.2.2 se puede encontrar el diagrama de conexión y sus características. Se utiliza el pin *SCL* del módulo *AT24C32* que envía la señal de reloj y se conecta con el pin *D1* del *NodeMCU*, luego usa el pin *SDA* del módulo *AT24C32* que envía los datos y se conecta al pin *D2* del *NodeMCU*, posteriormente los pines *VCC* y *GND* del módulo *AT24C32* son conectados a las líneas generadas de corriente y tierra del *NodeMCU*, ver Figura 3.6. Es importante mencionar, que este módulo solo obtiene el tiempo (fecha y hora), no activa o apaga algún actuador, ni guarda información en la base de datos, sin embargo, funciona como complemento de otros elementos que hacen uso de esta función.

El Algoritmo 3-2, es utilizado como referencia para configurar la interacción entre el *NodeMCU* y el módulo *AT24C32*.

Inicio

- Incluir librería RTC
- Se crea la constante que ejecuta las funciones del temporizador
- Inicializa el temporizador
 - Si inicia correctamente entonces
 - Establecer fecha y hora
 - Si no
 - Imprimir error
- Almacenar en una variable de tipo DateTime la fecha y hora actual

Fin

Algoritmo 3-2. Interacción entre NodeMCU y el módulo AT24C32.

3.2.3 NodeMCU y el relevador

Un relevador es un actuador que internamente contiene una bobina, esta al ser excitada con un pulso electromagnético puede cambiar su estado, permitiendo abrir o cerrar el flujo de la corriente eléctrica, en el tema 2.2.4, se describe el diagrama de conexión y las características del relevador que será utilizado en el presente proyecto. El pin *IN* del relevador es conectado al pin *D5* del *NodeMCU*, los pines *VCC* y *GND* del relevador son conectados a las líneas de corriente y tierra del *NodeMCU*, ver Figura 3.6, donde se muestra el diagrama de conexión entre módulos, sensores y microcontrolador.

Al combinar el reloj en tiempo real utilizando el módulo *AT24C32* junto con el relevador, se puede automatizar el flujo de agua en el prototipo hidropónico, el Algoritmo 3-3, detalla el funcionamiento de este proceso:

Inicio

- Se fija el pin que usará el relevador para la bomba de agua
- Establece variable del estado de la bomba
- Establece variable para activar la bomba manual
- Establecer pin bomba en modo escritura
- Obtener el valor del reloj en tiempo real
- Ejecutar función para encender bomba
 - Obtener el valor de la bomba manual
 - Si bomba manual = falso entonces
 - Si hora actual ≥ 8 y ≤ 19 entonces
 - Si minuto actual ≥ 0 y ≤ 20 entonces
 - Estado de la bomba = encender
 - Si no
 - Estado de la bomba = apagar
 - Si no
 - No hacer nada
 - Si no
 - Estado de la bomba = encender

Fin

Algoritmo 3-3. Automatización de bomba de agua utilizando NodeMCU y el relevador.

3.2.4 NodeMCU y el sensor HC-SR04

El sensor *HC-SR04* utiliza ondas ultrasónicas para medir la distancia que existe entre este y un objeto que se encuentre frente a él. En el presente proyecto de tesis se utiliza el sensor para conocer la cantidad de líquido almacenada en el recipiente contenedor de la solución nutritiva, el sensor ultrasónico cuenta con 2 transductores, el primero se utiliza para enviar una señal ultrasónica, cuando la señal encuentra con un obstáculo (el agua) rebota y el segundo transductor la recibe. La respuesta que provee el sensor, es la duración del tiempo en microsegundos, que tarda la señal en ir del transductor principal al secundario, al obtener solo el tiempo, se debe aplicar una fórmula para obtener la distancia.

La fórmula para calcular la velocidad es la siguiente:

$$velocidad = \frac{distancia}{tiempo}$$

Haciendo un despeje, para conocer la distancia:

$$distancia = velocidad \times tiempo$$

La velocidad del sonido es equivalente a:

$$velocidad\ del\ sonido = 343.2\ m/s$$

El sensor *HC-SR04* provee la información del tiempo en microsegundos (*ms*), y la fórmula del sonido está en metros (*m*), se realizará una transformación para cambiar *m/s* a *cm/ms*.

$$1\ s = 1000000\ ms$$

$$1\ m = 100\ cm$$

$$1\ m/s = 0.0001\ cm/ms$$

Por lo tanto, la velocidad del sonido calculada en *cm/ms* sería:

$$velocidad\ del\ sonido = 343.2 \times 0.0001$$

$$velocidad\ del\ sonido = 0.03432\ cm/ms$$

Finalmente, debido a la forma de operar del sensor *HC-SR04* el sonido viaja de ida y vuelta, por lo tanto, recorre 2 veces la misma distancia, para obtener el valor correcto, se aplica la siguiente fórmula:

$$distancia = \left(\frac{0.03432}{2} \right) \times tiempo$$

Se utiliza como referencia el diagrama de conexión entre módulos, sensores y microcontrolador, ver Figura 3.6, para conectar el sensor de proximidad *HC-SR04* a *NodeMCU*. El pin *Trig* del sensor *HC-SR04* que se conecta a una resistencia de *220 Ohms* y después llega al pin *D6* del *NodeMCU*, el pin *Echo* del sensor *HC-SR04* se conecta directamente al pin *D7* del *NodeMCU*. Por último, los pines *VCC* y *GND* del sensor *HC-SR04* se conectan a las líneas de *VCC* y *GND* proporcionadas por *NodeMCU*.

El Algoritmo 3-4, define la secuencia que será utilizada dentro del microcontrolador para obtener el valor de la distancia entre el sensor *HC-SR04* y la solución nutritiva:

Inicio

Reservar el pin que recibe el Trig en NodeMCU
Reservar el pin que recibe el Echo en NodeMCU
Establecer pin Trig en modo escritura
Establecer pin Echo en modo lectura
Enviar pulso Trig
tiempo = valor pulso Echo
distancia = $(0.03432/2) * \text{tiempo}$

Fin

Algoritmo 3-4. Obtener distancia entre el sensor HC-SR04 y la solución nutritiva, utilizando NodeMCU.

3.2.5 NodeMCU y el módulo pH-4502C

El módulo análogo *pH-4502C* como su nombre lo indica sirve para obtener lecturas de pH en soluciones líquidas, en el tema 2.2.2 se describen sus características y pines de conexión disponibles.

La conexión utilizada en el presente proyecto, comienza con la conexión del pin *PO* en el módulo *pH-4502C* con el pin *A0* del *NodeMCU*, los pines *VCC* y *GND* del módulo *pH-4502C* se conectan con las líneas compartidas *VCC* y *GND* de *NodeMCU*, visualizar el diagrama de conexión entre módulos, sensores y microcontrolador, Figura 3-6.

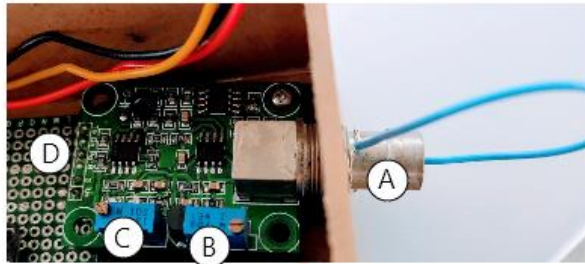
El pH se mide en una escala de 0 – 14, las soluciones que contienen valores en un rango de 0-6 son consideradas ácidas, soluciones con valores de 7 se consideran neutras y con valores entre 8-14 alcalinas, sin embargo, el módulo *ph-4502C* al ser analógico no devuelve una salida digital. El microcontrolador *NodeMCU* contiene un *ADC*⁵³ que lee el cambio de voltaje en un número entre 0 – 1023, por lo tanto, para convertir el valor de la señal analógica a digital se debe utilizar la siguiente fórmula:

$$\text{voltaje} = \text{valorAnalogico} * \left(\frac{5.0}{1023}\right)$$

⁵³ Convertidor de señal Analógica a Digital, es un dispositivo capaz de convertir señal analógica a digital, ya sea tensión o corriente, en una señal digital mediante un cuantificador.

Por defecto en módulo *pH-4502C* no viene calibrado, es necesario configurarlo para que las lecturas que toma la sonda, sean correctas, se utiliza el siguiente procedimiento para llevar a cabo esta acción:

1. Se conecta un cable en la parte central del conector *BNC* y el otro extremo del cable se conecta a la parte exterior, esto generara un pequeño cortocircuito que restaura los valores iniciales, se ajusta el valor del potenciómetro de compensación, ver Figura 3-7, hasta alcanzar los 2.5V (0V representa pH 0, 5V representa pH 14).



- A. Conector BNC
- B. Potenciómetro de compensación
- C. Potenciómetro de limite pH
- D. Pines de conexión

```
void setup(){  
  Serial.begin(115200);  
}  
  
void loop(){  
  int sensorValue = analogRead(A0);  
  float voltage = sensorValue * (5.0 / 1023.0);  
  Serial.println(" Voltaje: "+ String(voltage));  
  delay(300);  
}
```

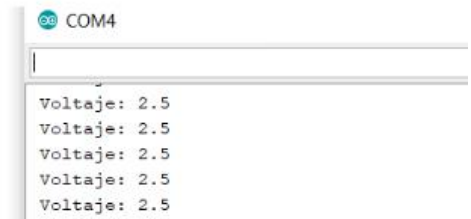


Figura 3-7. Calibración del módulo *ph-4502C*.

2. Ahora es necesario utilizar al menos 2 soluciones pH base, en el presente proyecto se utilizan soluciones en polvo de: pH 4.01 y pH 6.86, al realizar las lecturas con el módulo *ph-4502C* se han obtenido los siguientes voltajes 3.04V y 2.54V respectivamente, con la obtención de esa información es posible trazar una función lineal y así poder calcular el pH de cualquier solución, ver figura 3-8.

No.	pH	voltaje
1	15.41	1.04
2	12.56	1.54
3	9.71	2.04
4	6.86	2.54
5	4.01	3.04
6	1.16	3.54

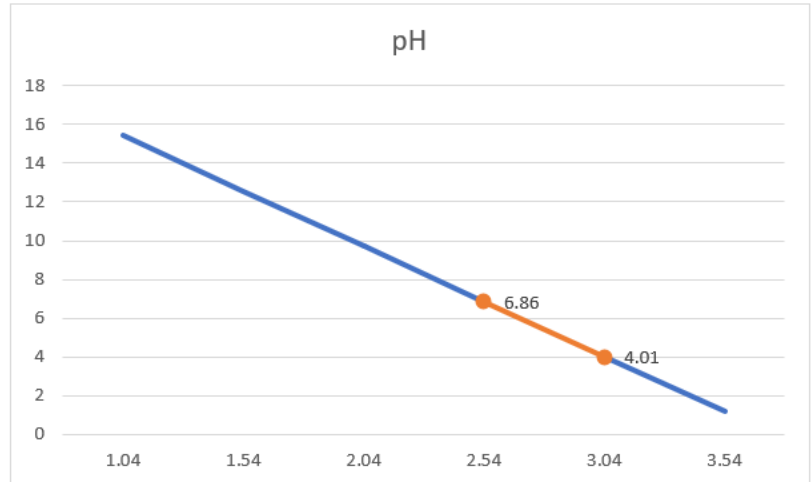


Figura 3-8. Función lineal para el cálculo de pH.

3. La fórmula para obtener el pH de cualquier punto de la pendiente es la siguiente:

$$pH = pendienteInclinacion * voltaje + intercepcion$$

De momento solo se conocen los valores de pH (4.01 y 6.86) y los voltajes (3.04 y 2.54), para conocer la pendiente de inclinación se realiza un despeje a la fórmula proporcionada en el paso 3:

$$pendienteInclinacion = \frac{(pH - intercepcion)}{voltaje}$$

Como ya se conoce el valor de 2 puntos de pH (4.01 y 6.86) y sus respectivos voltajes (3.04 y 2.54) no se necesita la intercepción, por lo tanto, se omite de la fórmula y se sustituye para obtener el valor de la pendiente de inclinación.

$$pendienteInclinacion = \frac{(4.01 - 6.86)}{(3.04 - 2.54)}$$

$$pendienteInclinacion = -5.7$$

4. Ya que se conoce el pH, voltaje y la pendiente de inclinación, de la fórmula descrita en el punto 3, se realiza el despeje necesario para obtener el valor de la intercepción.

$$intercepcion = pH - pendienteInclinacion \times voltaje$$

Sustituyendo valores:

$$intercepcion = 4.01 - (-5.7 \times 3.04)$$

$$\text{intercepcion} = 21.338$$

5. Al conocer el valor de todas las variables, se aplica la siguiente fórmula para conocer el valor de pH en cualquier punto de la recta proporcionando únicamente el voltaje, que es la información que el módulo *ph-4502C* entrega como respuesta.

$$\text{pH} = (-5.7 \times \text{voltaje}) + 21.338$$

El Algoritmo 3-5, se utiliza para obtener el valor de pH, utilizando el módulo *ph-4502C* y el microcontrolador *NodeMCU*.

Inicio

Definir pinpH = A0
Establecer pinpH en modo lectura
voltaje = lecturaSensor(pinPh)
valorPh = -5.7 x voltaje + 21.338

Fin

Algoritmo 3-5. NodeMCU y el módulo ph-4502C.

3.2.6 NodeMCU y Firebase

El módulo *NodeMCU* se encarga de recolectar los valores de temperatura, humedad, nivel de agua, estado de la bomba y el pH, de sus respectivos módulos y sensores, sin embargo, no guarda estos datos en ningún lado, por lo tanto, se pierden. Para almacenar la información, es necesario utilizar una base de datos. En el tema 2.3.3 se analizan diferentes *SGBD*, Firebase fue seleccionada debido a que la información puede ser accedida en tiempo real y provee una *API* adecuada para integrarse con el presente proyecto.

El Algoritmo 3-6, presenta la secuencia de configuración que se propone utilizar para conectar el microcontrolador *NodeMCU* y la comunicación con la base de datos Firebase, además del almacenamiento de la información de los sensores.

Inicio

```
// Librerías
Cargar librería FirebaseArduino
Cargar librería ArduinoJson
// Constantes
Definir variable firebase anfitrión
Definir variable firebase autenticación
Definir identificador Wi-Fi
Definir contraseña Wi-Fi
// Función Setup() Arduino
Iniciar conexión Wi-Fi ingresando el identificador y su contraseña
Iniciar conexión a Firebase ingresando el anfitrión y la autenticación
Cargando configuraciones en la base de datos
  Configuración del contenedor de solución nutritiva
  Configuración del estado de bomba
  Configuración de la bomba manual
// Función Loop() Arduino
Si nuevoCultivo entonces
  Establecer estructura para nuevoCultivo en Firebase
si no
  Crear objeto JSON con valores de todos los sensores
  Si minuto == 50 entonces
    Guardar información en el histórico de sensores Firebase
  si no
    Guardar información de sensores en Firebase
```

Fin

Algoritmo 3-6. Integración de NodeMCU y Firebase.

3.2.7 Diagrama de flujo del programa dentro del módulo NodeMCU

Para que la operación del componente 2 funcione de forma correcta, se deben realizar todas las acciones desde el punto 3.2.1 hasta el 3.2.8. La metodología de programación que es utilizada dentro de los microcontroladores es de tipo declarativa, por tal motivo, se propone integrar cada acción de los puntos 3.2.1 hasta el 3.2.8 en pequeñas funciones, de tal forma que, el mantenimiento de la aplicación sea más sencillo, en la Figura 3-9 se muestra el diagrama de flujo propuesto que permitirá realizar todas las operaciones de dicho componente.

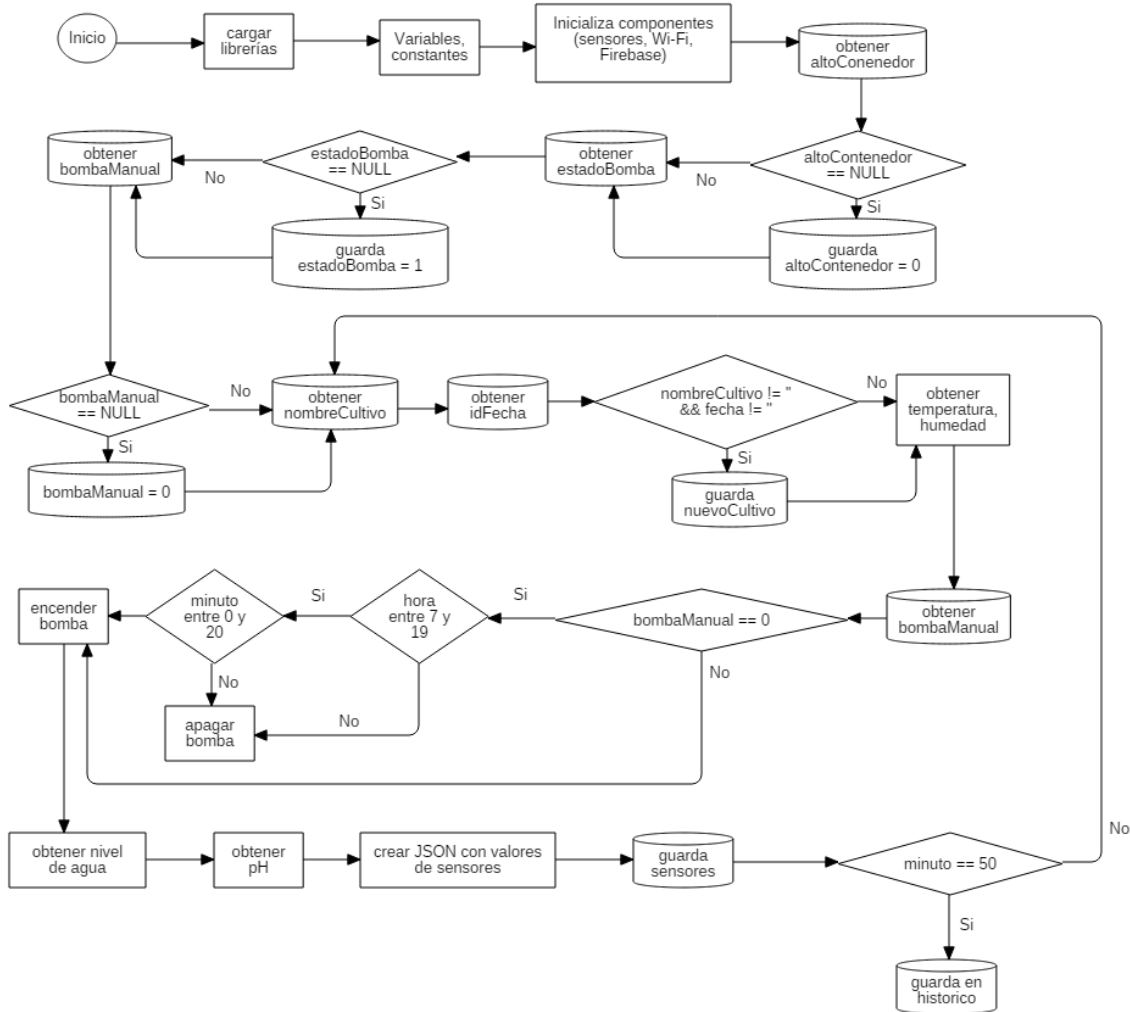


Figura 3-9. Diagrama de flujo del programa en el módulo NodeMCU.

En el proceso *cargar librerías*, ver Figura 3-9, deben ser incluidas todas las librerías externas que serán utilizadas por los sensores o funciones especiales, tales como *ESP8266WiFi*, *FirebaseArduino*, *ArduinoJSON*, *DHT*, *RTC_lib*.

Para el proceso *Variables, constantes*, ver Figura 3-9, se debe definir el nombre del prototipo, los pines a utilizar para cada sensor y actuador, las credenciales de conexión a *Firebase* y a la red *Wi-Fi*, las variables para almacenar la temperatura, humedad, nivel de agua, pH, estado de la bomba y bomba manual.

Para todos los procesos futuros que almacenen o accedan a información dentro de la base de datos, será necesario que el microcontrolador cuente con una conexión a internet adecuada, este procedimiento será realizado en el proceso *Inicializa componentes*.

3.3 Base de datos (Componente 3)

La base de datos es la estructura que permite que la información almacenada pueda ser recuperada en todo momento, siempre y cuando las condiciones del cliente, el servidor y la conexión a internet así lo permitan. Los datos que el presente proyecto de tesis debe almacenar, necesita responder los siguientes cuestionamientos:

- ¿Cuáles son los dispositivos (Componente 2) que están disponibles?
- ¿Quién es el usuario que está utilizando el componente 2 seleccionado?
- ¿Cuál es el cultivo actual?
- ¿Cuál es el listado de cultivos realizados por usuario?
- ¿Cuál es el valor actual de los sensores integrados en el componente 2?
- ¿Cuál es el estado de la bomba de agua?
- ¿Cuál es el nivel de agua del contenedor?
- Histórico de cada cultivo
- ¿Quién es el usuario autenticado?

En consideración a los cuestionamientos previamente mencionados, se propone el siguiente diseño de la base de datos. Es importante conocer, que la información contenida en ella, es ingresada de 2 formas totalmente independientes, utilizando el componente 2 y 4, cada uno de ellos accede a documentos fijos en la estructura de la base de datos, ver Figura 3-10.

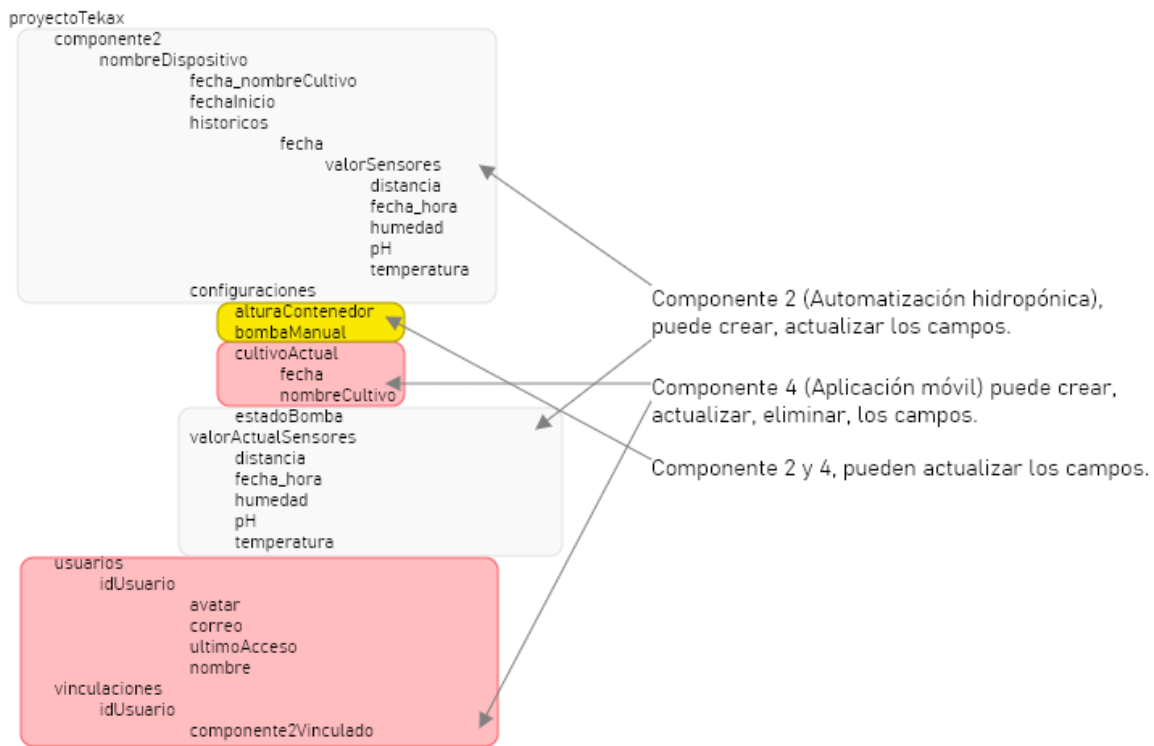


Figura 3-10. Esquema de la base de datos.

Como se mencionó en el tema 2.3.3 el *SGBD* seleccionado para el presente proyecto es *Firebase*, una base de datos *NoSQL*, por tal motivo, el diseño utilizado es escalar y representado en formato *JSON*, en la Figura 3-10, es posible observar diferentes indentaciones⁵⁴, estas representan el nivel jerárquico de cada documento, siendo los elementos ubicados más a la izquierda de mayor nivel, mientras que los de la derecha representan menor nivel, por otra parte al tener acceso a un elemento de jerarquía mayor, se puede obtener toda la información de los hijos de dicho elemento.

3.4 Aplicación móvil (Componente 4)

La aplicación móvil desarrollada para el presente proyecto de tesis debe cubrir algunos requisitos:

- Generación de nuevo usuario para acceso al sistema.
- Para acceder a los módulos el usuario debe autenticarse.
- La aplicación debe permitir la vinculación entre usuario y componente 2.
- Permitir la creación de nuevos cultivos.
- Brindar información actualizada de los sensores que integran el componente 2.
- Capacidad de activar o desactivar la bomba de agua manualmente.
- Permitir establecer la altura del contenedor de la solución nutritiva.
- Capacidad de mostrar históricos de la información recolectada de los sensores que integran el componente 2.
- Permitir el cierre de sesión

Analizando la información requerida para la construcción de la aplicación móvil, se diseña el diagrama de casos de uso, ver Figura 3-11 que permitirá la construcción de cada elemento necesario.

⁵⁴ Término utilizado en informática para representar el movimiento de un bloque de texto hacia la derecha, insertando espacios o tabulaciones.

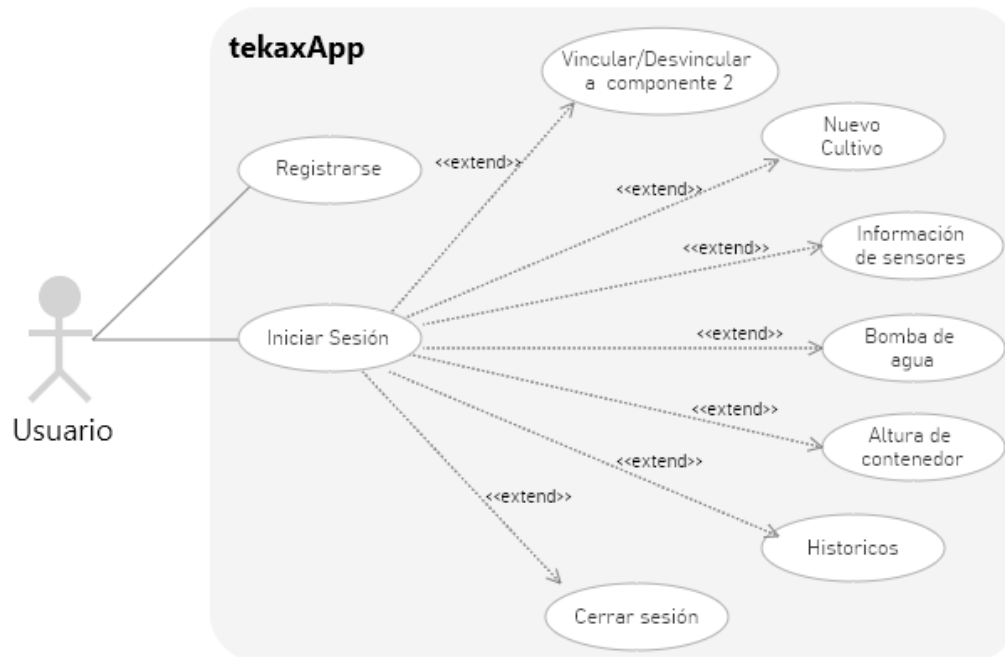


Figura 3-11. Diagrama de casos de uso de la aplicación móvil.

3.4.1 Diseño de interfaz y experiencia de usuario

Antes de comenzar el desarrollo de la aplicación se realiza un bosquejo del diseño de la interfaz y la experiencia de usuario, tomando en consideración los puntos que fueron planteados en el tema 2.4.5 y 2.4.6. Es importante definir la estructura y el diseño de cada pantalla ya que permite trabajar sobre algo previamente planeado y no realizar modificaciones al momento del desarrollo, esto evitará pérdida de tiempo en reestructuraciones innecesarias.

La herramienta en la que se realizaron los bosquejos, cuenta con licencia Open Source⁵⁵ y tiene por nombre *Lunacy*, la versión con la que se trabajó en el presente proyecto es la 5.5 y funciona únicamente para el Sistema Operativo Windows.

Paleta de colores

Los colores deben reflejar armonía con el concepto de la aplicación, el objetivo de la misma es administrar cultivos hidropónicos, cuando se habla de plantas y agua, colores que sirven a este propósito son el verde y azul, por este motivo, la paleta de colores a utilizar

⁵⁵ Es un código diseñado de manera que sea accesible al público: todos pueden ver, modificar, y distribuir el código de manera que consideren conveniente.

está basada en estos dos, sus colores monocromáticos⁵⁶ y algunos complementarios⁵⁷, ver Figura 3-12.

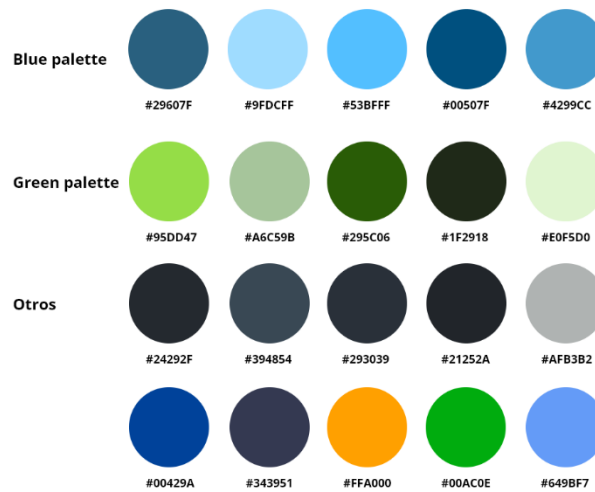


Figura 3-12. Paleta de colores para el diseño de la aplicación.

Logotipo

Para el desarrollo del logotipo, el objetivo era crear algo sencillo que cumpliera con 2 características:

1. Que al visualizarlo se interpretara de inmediato plantas y agua.
2. Que al ser reducido se distingan todos sus componentes perfectamente.

El logotipo diseñado, ver Figura 3-13, es muy importante ya que está presente en muchas partes de la aplicación, se encuentra como un icono, como botón, como logo, entre otros, además, los colores utilizados formarán la base de todas las pantallas en la aplicación móvil, permitiendo una cohesión de la misma.



Figura 3-13. Logotipo del proyecto Tekax.

⁵⁶ Son colores que derivan de un solo color base que es extendida mediante el uso de tonalidades claras y oscuras del mismo color.

⁵⁷ Son colores que se encuentran en una posición oponible dentro del círculo cromático.

Diseño de la aplicación

A continuación, se muestra el UI y UX creado, utilizando la paleta de colores propuesta y tomando como referencia el diagrama de casos de uso, ver Figura 3-11. Se realiza el diseño conceptual de cada pantalla dentro de la aplicación móvil, ver Figura 3-14, permitiendo seguir una guía clara del objetivo deseado.



Figura 3-14. Diseño de interfaz y experiencia de usuario.

3.4.2 Estructura superior de widgets

Como se mencionó en el tema 2.4.8 la piedra angular de todo proyecto en Flutter son los widgets, su principal característica es que funcionan de forma reactiva y hacen uso de la herencia.

Casi todas las acciones que el usuario puede realizar dentro de la aplicación necesitan previa autenticación, por tal motivo, es necesario crear un mecanismo que permita de forma eficiente verificar el estado del usuario (si está autenticado o no). En la Figura 3-15, se observa la estructura principal de widgets que se propone utilizar como capa superior para restringir el acceso a las acciones que requieren autenticación.

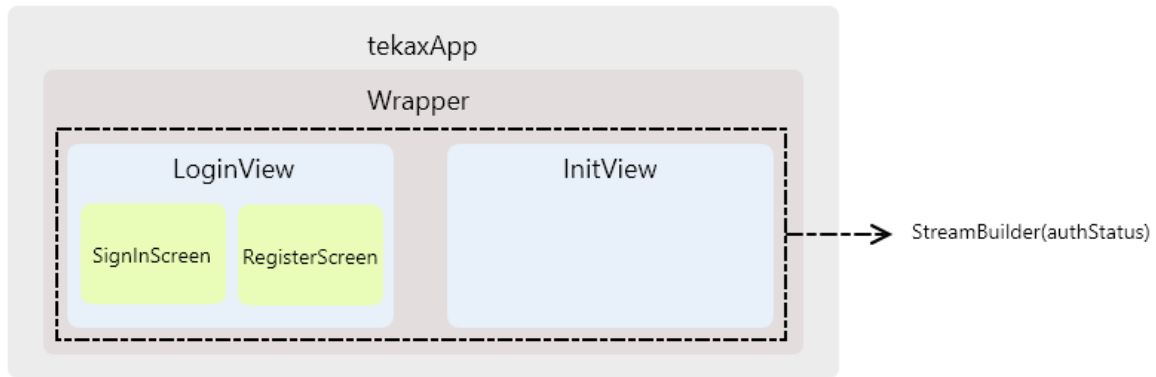


Figura 3-15. Conjunto de widgets que integran la estructura superior.

Se diseñó el siguiente diagrama de secuencia, ver Figura 3-16, que permite conocer si el usuario que utiliza la aplicación está autenticado o no. El widget *StreamBuilder* se encargará de verificar el estado de la autenticación, utilizando la variable *authStatus*, dependiendo de su valor la aplicación mostrará el widget *loginView* encargado del inicio de sesión o *initView* si el usuario cuenta con las credenciales correctas. A partir de este punto, se integrarán como hijos, los widgets que dan vida al resto de la aplicación.

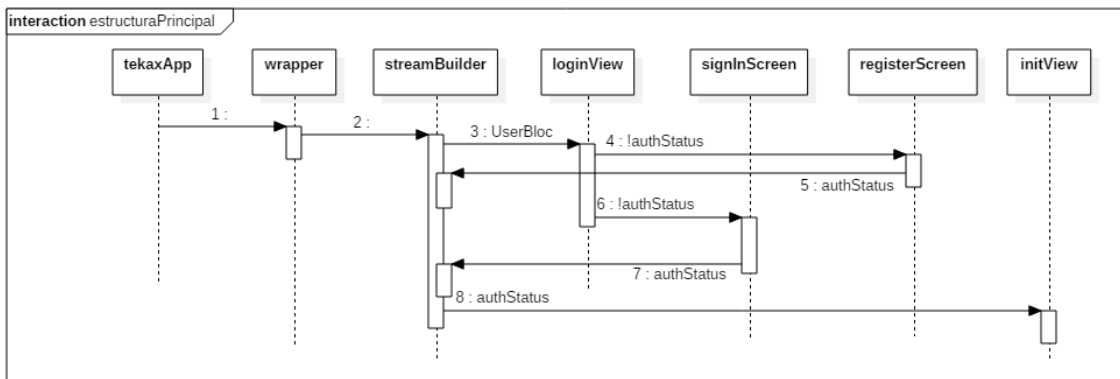


Figura 3-16. Diagrama de secuencia que permite verificar el estado del usuario en la aplicación móvil.

3.4.3 Nuevo usuario

Como se observa en el diagrama de casos de uso de la aplicación, documentado en el tema 3.4, la única acción que el usuario tiene permitido realizar sin estar autenticado es el registro, la API de Firebase cuenta con un método que permite implementar este

procedimiento de forma sencilla, para su integración al presente proyecto se utiliza la siguiente estructura de widgets, ver Figura 3-17.

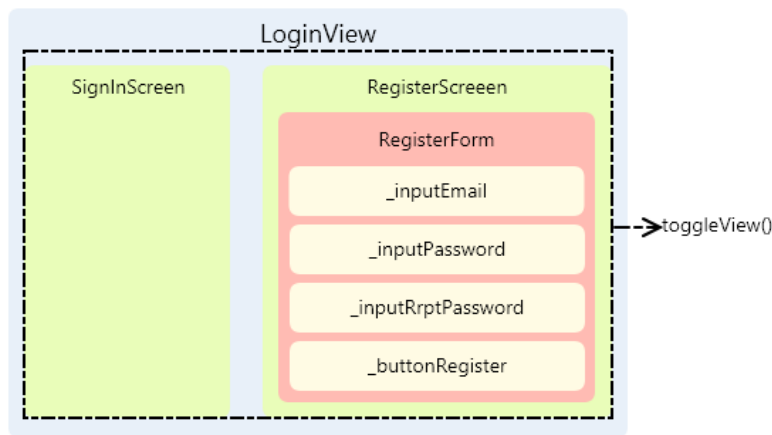


Figura 3-17 Estructura de widgets para la creación de nuevo usuario.

Para crear un nuevo usuario utilizando la API de Firebase, solo es necesario enviar 2 parámetros (correo y contraseña) hacia sus servidores, ellos se encargarán de recibir y procesar la información, devolviendo un objeto con el resultado de la operación. A continuación, ver Figura 3-18, se muestra un diagrama de secuencia de las actividades que se propone realizar para llevar a cabo la creación de un nuevo usuario en la aplicación móvil.

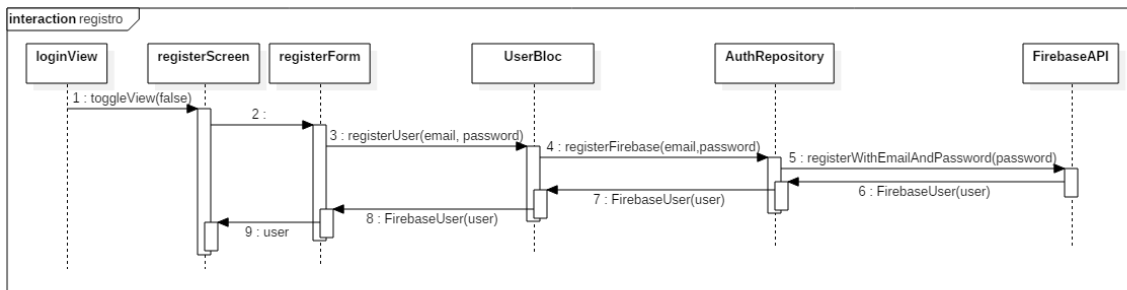


Figura 3-18 Diagrama de secuencia para la creación de nuevo usuario.

Lo importante de utilizar la API es que todo el procesamiento intermedio es realizado por ella, por ejemplo, la API valida que lo que fue ingresado como correo electrónico tenga formato de correo electrónico, que la contraseña tenga al menos 6 caracteres, verifica si el correo que el usuario está intentando registrar no existe, entre otras cosas.

3.4.4 Autenticación

Para garantizar que el acceso a la aplicación sea sencillo, se han implementado 2 mecanismos de autenticación:

1. *Usuario y contraseña.* Este mecanismo necesita que el usuario haya hecho un registro previo en la aplicación, como se explicó en el tema 3.4.3, posteriormente se ingresan las credenciales generadas y se envían hacia los servidores de Firebase, donde se verificará su autenticidad, devolviendo una respuesta que posteriormente la aplicación procesará permitiendo o denegando el acceso a la misma.
2. *Credenciales de Google.* Este mecanismo solo necesita una cuenta activa de Google, el sistema solicita al usuario sus credenciales, posteriormente la aplicación envía los datos hacia los servidores de Google, donde esta información es procesada permitiendo finalmente regresar un objeto con el resultado, si las credenciales son correctas, la aplicación móvil permitirá el acceso al sistema, en caso contrario lo negará.

En la Figura 3-19, se muestra la arquitectura de widgets propuesta para llevar a cabo la implementación de la autenticación dentro del proyecto, el método *toggleView* sirve para intercambiar las vistas entre la autenticación y el registro de un nuevo usuario.

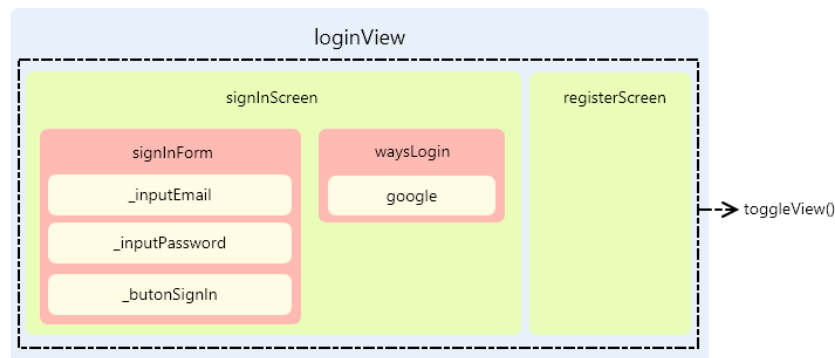


Figura 3-19. Estructura de widgets de la autenticación.

Diagrama de secuencia propuesto para llevar a cabo la implementación de la autenticación en la aplicación móvil, ver Figura 3-20.

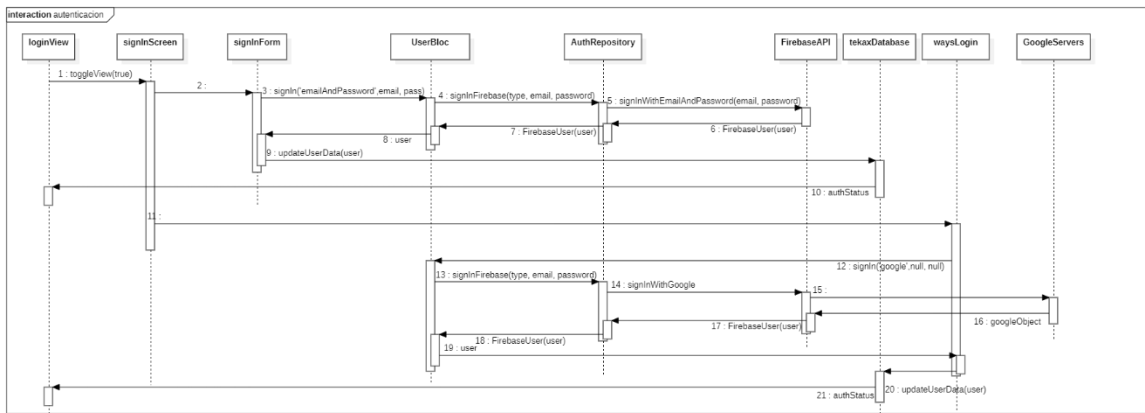


Figura 3-20 Diagrama de secuencia de la autenticación.

1. Si el widget *loginView* recibe como parámetro verdadero (*true*), entonces, llama un nuevo widget *signInScreen* (punto 1, ver Figura 3-20).
2. Dentro de *signInScreen* se encuentran 2 widgets que permiten el acceso a la aplicación utilizando diferentes métodos de acceso.
 - a. *signInForm* (punto 2), es el widget que contiene el formulario de inicio de sesión, utiliza el método tradicional, solicitando el usuario y contraseña que fueron registrados dentro de la aplicación, tema 3.4.3.
 - i. Una vez que el usuario introduce la información requerida en el formulario, presionará un botón que ejecutará el mecanismo para activar el inicio de sesión (puntos 3 – 8, ver Figura 3-20).
 - ii. Cuando la respuesta es procesada y se conoce el usuario que acceso a la aplicación, del objeto retornado por los servidores de Firebase, se extraen los elementos que serán almacenados en la base de datos de la aplicación móvil, creando un objeto de tipo *usuario*, únicamente con la información que la aplicación necesita (puntos 9 – 10, ver Figura 3-20).
 - b. *waysLogin* (punto 11, ver Figura 3-20), es el widget que contiene la autenticación utilizando una cuenta de terceros, en este caso particular de Google.
 - i. El usuario presiona el botón con el logotipo de Google, la aplicación se conectará con los servidores de esta empresa, pasando el control momentáneo de la aplicación a ellos, hasta que, una respuesta del proceso sea regresada (puntos 11 – 18, ver Figura 3-20)
 - ii. Cuando la respuesta llega a la aplicación móvil, el control vuelve, y se procesa la información recibida, finalmente, el usuario es actualizado permitiendo el acceso o denegándolo (puntos 19 – 21, ver Figura 3.20)

3.4.5 Patrón de diseño BLoC

Como fue mencionado en el tema 2.4.3, en el presente proyecto se implementa *BLoC* como patrón de diseño, su objetivo principal es separar la lógica del negocio de la interface del usuario. Todos los textos, iconos, logotipos, botones, entre otros componentes visuales deben ser separados en vistas, mientras que los métodos que ejecuten una acción, conexiones a fuentes de datos, operaciones en segundo plano, entre otros, deberán permanecer en el apartado de la lógica del negocio.

Para llevar a cabo la implementación, cada acción del diagrama de casos de uso, ver Figura 3-11, se propone crear la siguiente estructura de carpetas, ver Figura 3-21, permitiendo tener una arquitectura más limpia y ordenada, brindando la posibilidad de reutilizar componentes y dar mantenimiento de una forma más eficiente.

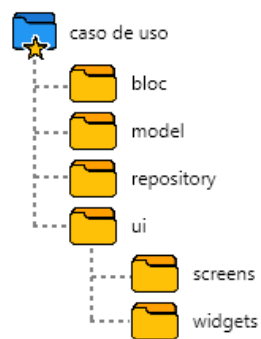


Figura 3-21. Estructura de carpetas utilizada para implementar el patrón de diseño BLoC.

La información que se almacena en cada carpeta es la siguiente:

- **bloc.** Es la capa que contiene la entrada a la lógica del negocio, se definen los casos de uso que pertenecen a la aplicación, por ejemplo: el usuario tiene la acción de autenticarse, cerrar sesión, entre otros.
- **model.** Un modelo es una estructura de datos que está diseñada para manejar la información dentro de la aplicación, tiene nombre, atributos y métodos que permiten manipular la información del mismo.
- **repository.** Realiza la definición de los métodos que se conectan a una fuente de datos, por ejemplo: *API*, Punto final de comunicación⁵⁸, bases de datos, entre otros.
- **ui.** Contiene el apartado visual, es separado en 2 subcarpetas
 - **screens.** Muestra las pantallas individuales que pertenecen al caso de uso, por ejemplo, inicio de sesión.
 - **widgets.** Son elementos que pueden ser compartidos permitiendo su reutilización en toda la aplicación, por ejemplo, un botón.

⁵⁸ Es un dispositivo informático remoto que se comunica con una red a la que está conectado

3.4.6 Emparejar dispositivo

La aplicación móvil hace uso de la información recolectada por cada sensor que integra el componente 2, por esta razón, para utilizarla de forma correcta, cuando el usuario ingresa con las credenciales necesarias y accede al sistema, la primera acción que debe realizarse es el emparejamiento de la aplicación móvil con el componente 2.

La sección de emparejamiento o sincronización, debe mostrar los componentes disponibles permitiendo al usuario seleccionar el dispositivo deseado, en caso de que el usuario ya cuente con algún componente vinculado, el sistema automáticamente deberá mostrar la pantalla de inicio.

La estructura de widgets propuesta para implementar esta acción es la siguiente, ver Figura 3-22.

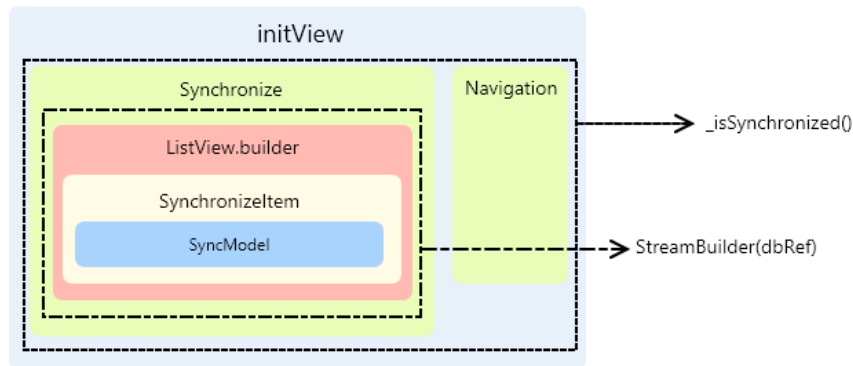


Figura 3-22. Estructura de widgets que se utiliza para emparejar el componente 2.

En la Figura 3-23, se muestra el diagrama de secuencia que se propone utilizar para llevar a cabo el emparejamiento de la aplicación móvil con el componente 2.

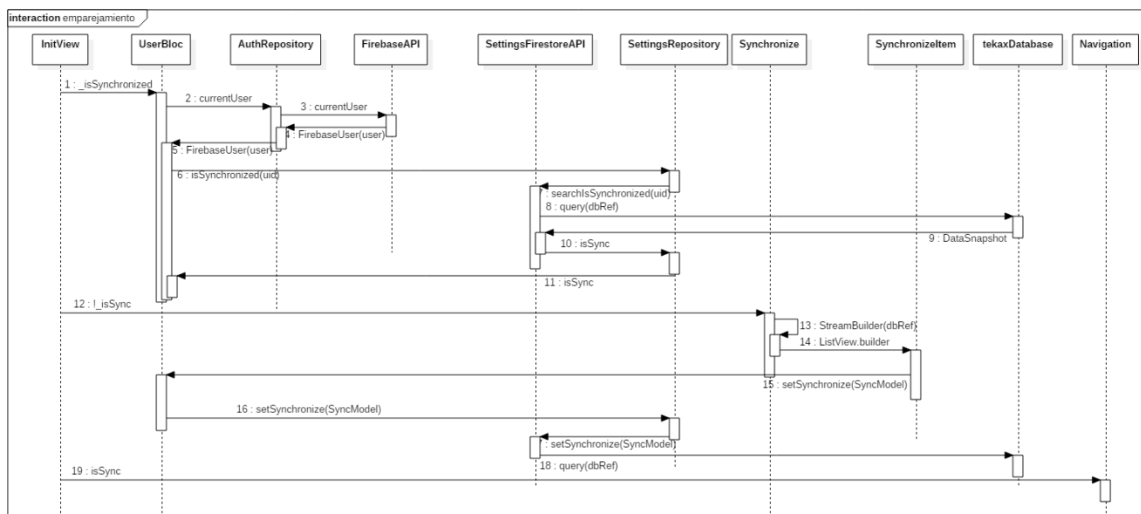


Figura 3-23. Diagrama de secuencia para emparejar el componente 2 con la aplicación móvil.

1. Verificar las credenciales del usuario (puntos 1-5, ver Figura 3-23)
2. Verificar si el usuario cuenta con un componente 2 emparejado (puntos 6 – 11, ver Figura 3-23)
 - a. Si el usuario no cuenta con un componente emparejado
 - i. Mostrar los componentes disponibles para emparejar (puntos 12 – 14, ver Figura 3-23)
 - ii. Seleccionar el componente con el que se desea emparejar (puntos 15 – 18, ver Figura 3-23)
 - b. Si el usuario ya cuenta con un componente emparejado
 - i. Ir hacia la navegación (punto 19, ver Figura 3-23)

3.4.7 Navegación

Siguiendo las recomendaciones de experiencia de usuario mencionadas en el tema 2.4.5, se diseña la aplicación para que funcione de forma intuitiva y que al usuario le resulte fácil moverse entre pantallas, por esta razón, se integra un widget llamado *Navigation*, ver Figura 3-24, que opera como menú en la aplicación y permite su desplazamiento.

Dentro de *Navigation* se utiliza el widget *Scaffold*, ver Figura 3-24, permitiendo crear una estructura predefinida para toda la pantalla de la aplicación, el widget *PageView* se encarga de encapsular todo lo que será visualizado en la ventana, mientras que el widget *CurvedNavigationBar*, reservará una sección de la pantalla que se superpondrá a los elementos encapsulados y será utilizado como menú de la aplicación

Se propone agregar 3 secciones principales que sean utilizadas como punto central, permitiendo el desplazamiento entre ellas y brindando fácil acceso a las mismas, estas secciones son las siguientes:

- **HomeView.** Es la pantalla inicial de la aplicación, lo que usuario observará una vez que ha iniciado sesión y emparejado un componente 2.
- **CurrentData.** Es el apartado donde serán mostradas las lecturas de los sensores del componente emparejado.
- **Settings.** Es la sección reservada para administrar las configuraciones que son permitidas dentro de la aplicación.

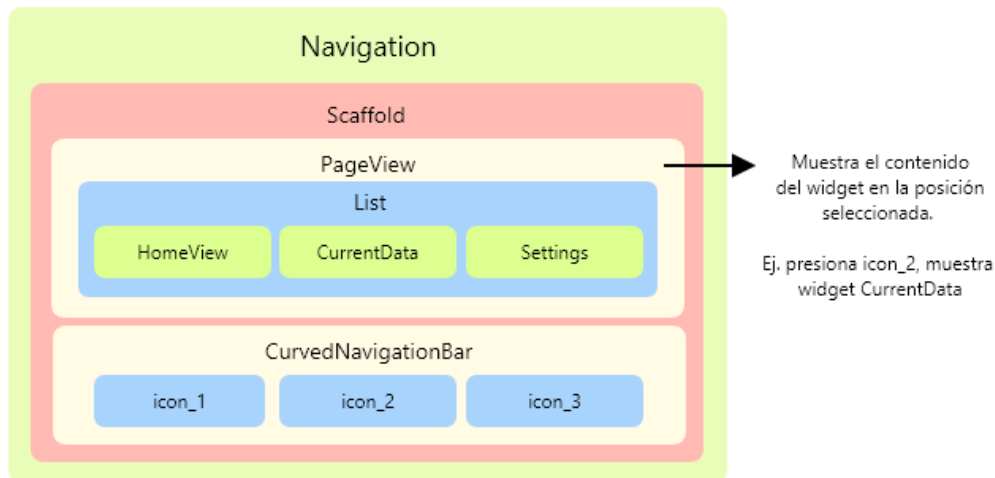


Figura 3-24. Estructura de widgets para implementar la navegación en la aplicación.

El funcionamiento de la navegación debe ser el siguiente:

1. Cada elemento del widget *CurvedNavigationBar* contará con un índice, comenzando por el número 0, ej. (icon_2 su índice es 1, ver Figura 3-24)
2. Cuando un elemento del widget *CurvedNavigationBar* es presionado, se toma su índice y se utiliza para acceder a los elementos agregados dentro de un objeto tipo *List* en el widget *PageView*.
3. Finalmente, el elemento del *objeto* que es accedido mediante el índice, toma el control y se coloca por encima de los demás, mostrando este elemento en el widget *PageView*.

3.4.8 Pantalla de inicio

Por defecto, la navegación mostrada en el punto 3.4.7 inicia con el widget *HomeView*, este componente se subdivide en 5 widgets más, que conforman las diferentes operaciones que el usuario puede realizar.

- **HistoryScreen.** Widget encargado de mostrar los históricos de los cultivos realizados.
- **PictureScreen.** Se encargará de mostrar fotografías de progreso en los cultivos, capturadas por el usuario.
- **HarvestScreen.** Es el widget encargado de crear nuevos cultivos en la aplicación.
- **PrototypeScreen.** Se encargará de ayudar al usuario a construir su propio prototipo hidropónico.
- **HomeScreen.** Es el widget que contiene los botones de activación para mostrar y ocultar los otros widgets dentro de *HomeView*.

La estructura de widgets propuesta para la construcción de la pantalla de inicio es la siguiente, ver Figura 3-25.

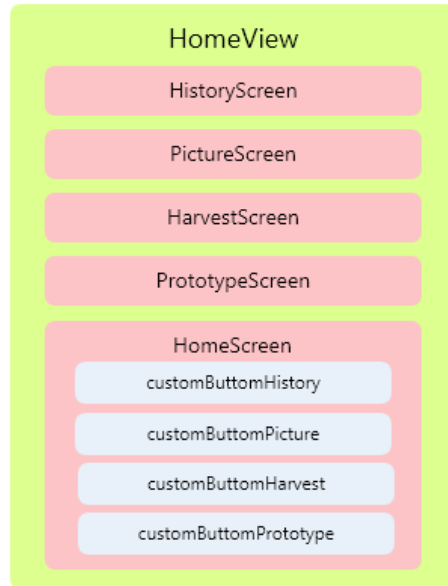


Figura 3-25. Estructura de widgets de la pantalla de inicio.

Cuando se accesa a *HomeView*, los widgets hijos *HistoryScreen*, *PictureScreen*, *HarvestScreen*, *PrototypeScreen* permanecen ocultos, mientras que *HomeScreen* es mostrado, este widget contiene 4 botones personalizados que al ser presionados realizan internamente un cambio de variable y muestran un widget específico, en la Figura 3-26, se muestra el diagrama de secuencia que realiza esta operación.

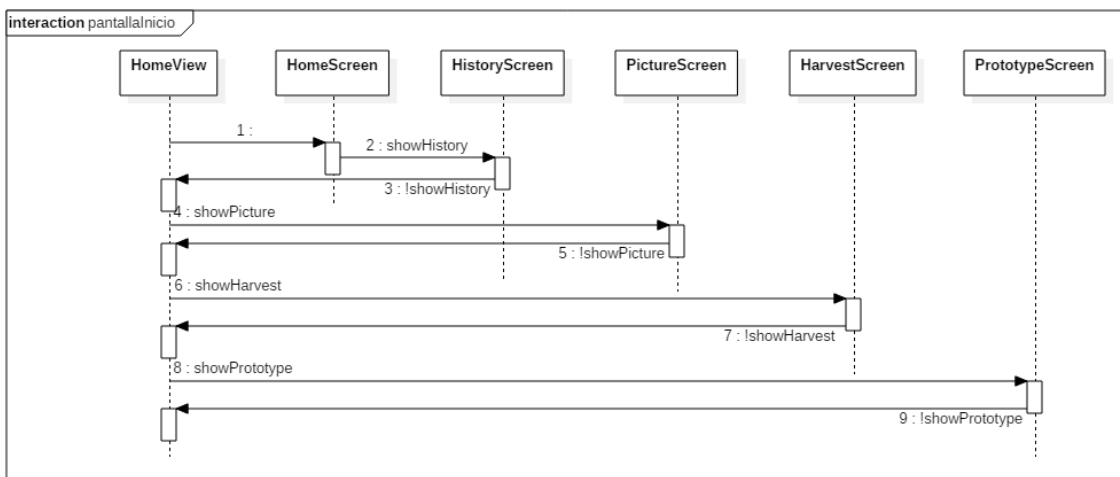


Figura 3-26. Diagrama de secuencia de la pantalla de inicio.

3.4.9 Nuevo Cultivo

Dentro de la aplicación es necesario conocer los cultivos que están en producción, la sección *Cultivo* permitirá, generar toda la estructura necesaria para la creación de un nuevo cultivo y almacenar estos cambios en la base de datos. El nuevo cultivo comenzará con el widget *HarvestScreen* y su estructura propuesta es la siguiente, ver Figura 3-27:

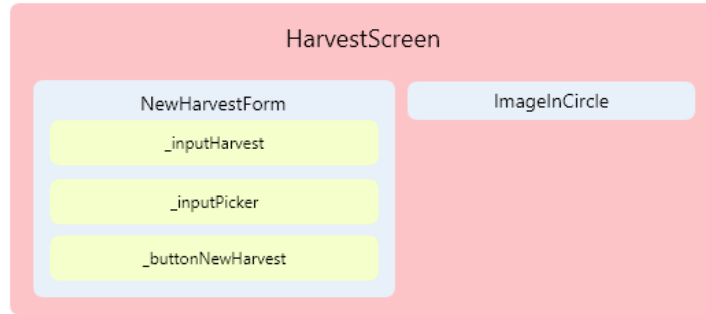


Figura 3-27. Estructura de widgets para la creación de nuevo cultivo.

Para crear un nuevo cultivo dentro de la aplicación, deben conocerse 4 valores:

- El usuario que está creando el cultivo.
- El nombre del dispositivo del componente 2, donde será anexado el nuevo cultivo.
- El nombre del nuevo cultivo
- La fecha de inicio para el cultivo.

El diagrama de secuencia propuesto para llevar a cabo la creación de nuevos cultivos es el siguiente, ver Figura 3-28:

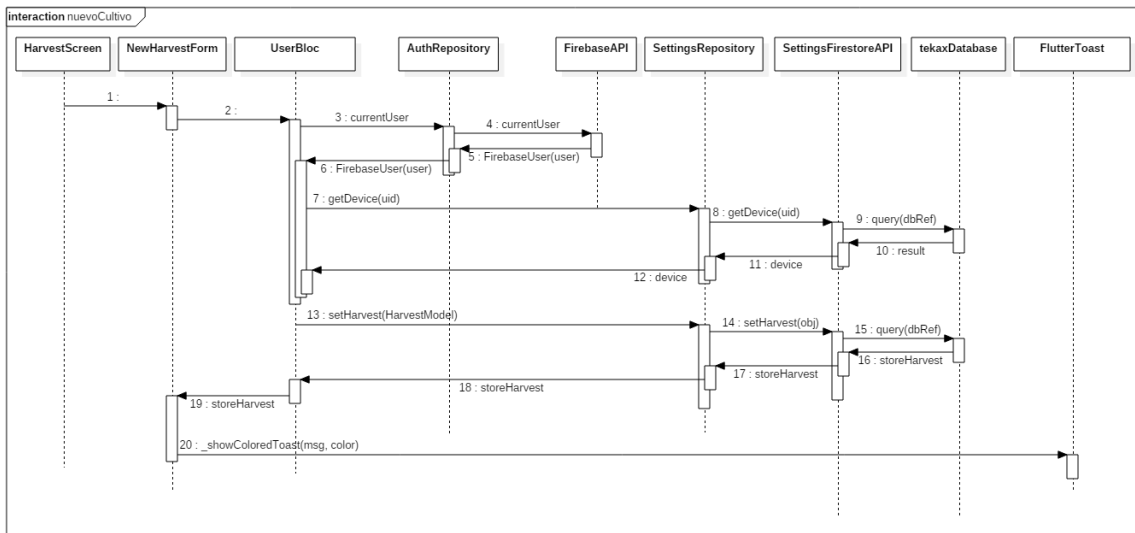


Figura 3-28 Diagrama de secuencia para la creación de nuevos cultivos

1. Acceder a la sección nuevo cultivo (puntos 1-2, ver Figura 3-28).
2. Verificar ¿Qué usuario está utilizando la aplicación móvil? (puntos 3 – 6, ver Figura 3-28).
3. Obtener el dispositivo emparejado con el usuario activo (puntos 7 – 12, ver Figura 3-28).
4. Ingresar el nombre del nuevo cultivo y su fecha de inicio.
5. Almacenar la información del nuevo cultivo en la base de datos y devolver una respuesta del estado de la transacción (puntos 13 – 19, ver Figura 3-28).
6. Notificar al usuario el resultado de la transacción (punto 20, ver Figura 3-28).

3.4.10 Información de los sensores

Esta sección será la encargada de mostrar las lecturas de los sensores que conforman el componente 2, se accederá desde el menú de navegación, seleccionando la segunda opción *icon_2*, ver Figura 3-24, su widget principal es *CurrentData* y está conformado por los siguientes widgets, ver Figura 3-29:



Figura 3-29. Estructura de widgets para mostrar la información de los sensores.

El diagrama de secuencia propuesto, para la implementación de la obtención de los sensores, inicia con el widget *currentData*, contiene el UI principal de la sección y posteriormente ejecuta diferentes operaciones, ver Figura 3-30.

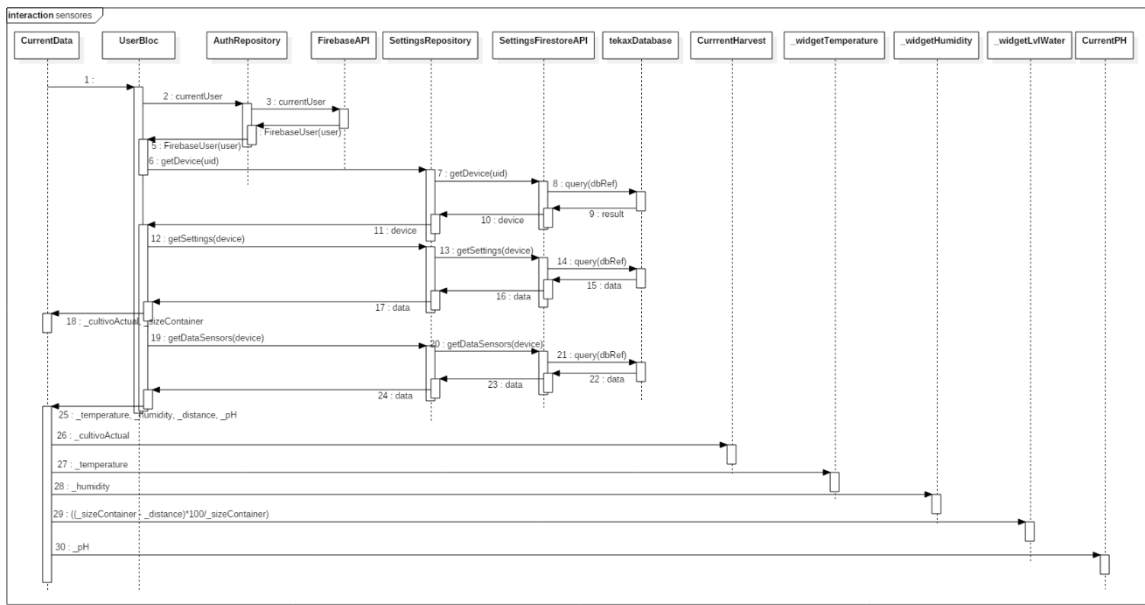


Figura 3-30 Diagrama de secuencia para mostrar la información de los sensores

1. Obtener el usuario activo (puntos 1 -5, ver Figura 3-30).
2. Obtener el dispositivo del componente 2 emparejado con el usuario activo (puntos 6 – 12, ver Figura 3-30).
3. Obtener las configuraciones que tiene el dispositivo emparejado (puntos 12 – 18, ver Figura 3-30).
4. Obtener la información de todos los sensores pertenecientes al dispositivo emparejado (puntos 19 – 25, ver Figura 3-30).
5. Retornar por parámetro las variables `_cultivoActual`, `_temperature`, `_humidity`, `_sizeContainer`, `_pH` hacia sus widgets correspondientes, que serán los encargados de agregar el UI a la información recibida y mostrarán el resultado en la aplicación móvil (puntos 26 – 30, ver Figura 3-30).

Se propone utilizar la siguiente fórmula para calcular la distancia que existe entre el sensor de proximidad *HC-SR04* y la solución nutritiva almacenada dentro del contenedor (punto 29, ver Figura 3-30):

$$distancia = (_sizeContainer - _distance) \times 100 / _sizeContainer$$

El valor de la distancia es lo que se almacenará en la base de datos, este elemento será utilizado en un futuro, para realizar el cálculo del nivel de agua dentro del contenedor.

3.4.11 Configuraciones

La sección de configuraciones funcionará de forma diferente a los demás módulos, algunos solo actualizan información en la base de datos (nuevo usuario, autenticación, emparejamiento, nuevo cultivo), otros solo realizan lecturas (información de sensores), esta

La sección de configuraciones contará con 3 widgets que funcionarán como subsecciones:

1. **UserInfo**. Se encargará de mostrar la información de la cuenta que ha iniciado sesión, independientemente del método de autenticación realizado.
2. **Operations**. En esta sección serán agrupadas las configuraciones que tiene permitido el usuario realizar para el componente 2 al cual esta emparejado.
3. **ImageInCircle**. Es una imagen dentro de un botón que al ser presionada se encargará de cerrar la sesión activa.

La sección de configuraciones funcionará de la siguiente forma:

1. Inicia con el widget *Settings* que contiene la estructura para implementar el UI diseñado y carga los 3 widgets hijos que componen al padre.
2. Se carga el widget *UserInfo*.
 - a. Obtiene la información del usuario activo (puntos 2 – 5, ver Figura 3-32)
 - b. Muestra la información dependiendo del tipo de usuario autenticado (punto 6, ver Figura 3-32).
3. Se carga el widget *Operations*.
 - a. Obtiene la información del usuario activo (puntos 2 – 5, ver Figura 3-32).
 - b. Obtiene el dispositivo emparejado (puntos 7 – 12, ver Figura 3-32).
 - c. Obtiene las configuraciones del dispositivo emparejado (puntos 13 – 19, ver Figura 3-32).
 - d. Agrega un botón para desvincular el dispositivo emparejado (punto 20, ver Figura 3-32).
 - e. Agrega un botón que funciona como interruptor para encender y apagar la bomba de agua (punto 21, ver Figura 3-32).
 - f. Agrega un control deslizante de rango, para indicar el tamaño del contenedor que almacena la solución nutritiva en el prototipo hidropónico (punto 22, ver Figura 3-32).
4. Se carga el widget *ImageInCircle*
 - a. Al ser presionado cierra la sesión del usuario en el dispositivo móvil (puntos 23 – 25, ver Figura 3-32)

3.4.12 Históricos

Dentro de la sección de históricos, el usuario podrá observar de forma general cuales han sido las lecturas obtenidas de cada sensor y serán mostradas en un gráfico lineal, cada línea deberá representar un parámetro de medición (temperatura, humedad, nivel de agua, pH).

Para acceder a los históricos, es necesario navegar hacia la pantalla de inicio (*HomeView*) mostrada en el tema 3.4.8 y posteriormente presionar el botón *Histórico*, esta acción establecerá como widget principal *HistoryScreen* mostrándolo en toda la pantalla de

la aplicación, la arquitectura de widgets que se propone utilizar para su implementación es la siguiente, ver Figura 3-33:

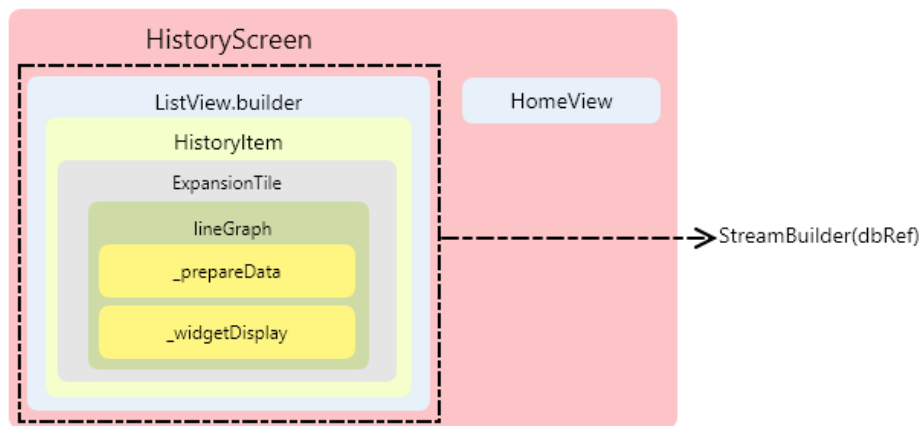


Figura 3-33. Estructura de widgets para la construcción de los históricos.

El diagrama de secuencia propuesto, ver Figura 3-34, establece la forma de interacción entre los diferentes componentes, permitirá visualizar el gráfico final del histórico de un cultivo determinado.

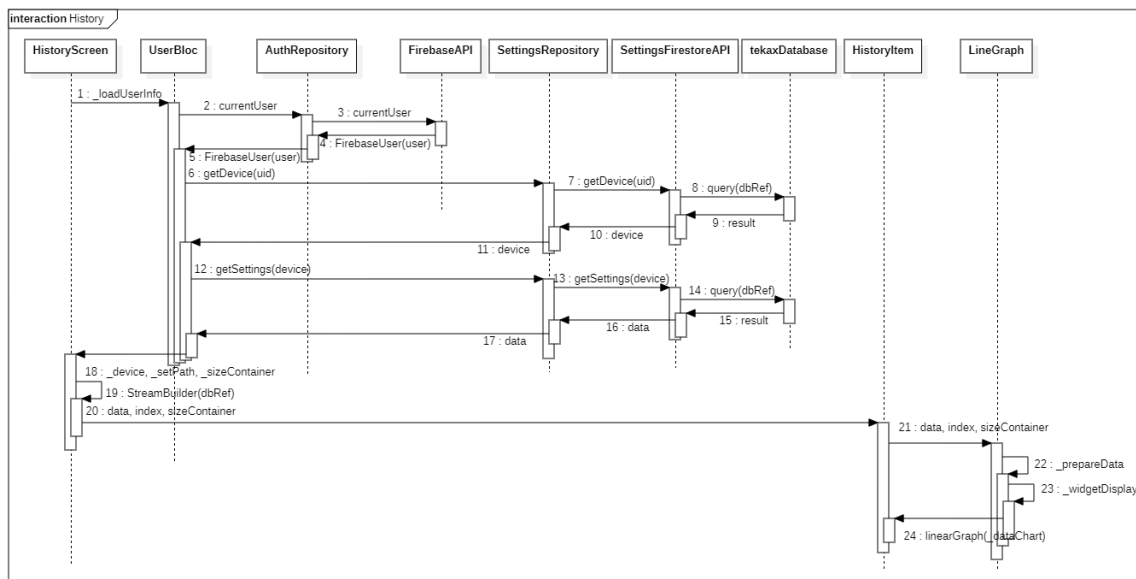


Figura 3-34. Diagrama de secuencia para mostrar los históricos.

1. Al cargar el contenido dentro del widget *HistoryScreen* la primera acción ejecutada es el método *_loadUserInfo*.
 - a. Solicita el usuario activo de la sesión actual (puntos 2-5, ver Figura 3-34).
 - b. Solicita el dispositivo al cual el usuario esta emparejado (puntos 6 – 11, ver Figura 3-34)

- c. Se obtienen las configuraciones que el usuario y el dispositivo emparejados poseen (puntos 12 – 18, ver Figura 3-34).
2. Se creará un *StreamBuilder* que permitirá escuchar los cambios en determinados campos de la base de datos, definidos por la ruta establecida en *dbRef* (punto 19, ver Figura 3-34).
 - a. Se carga el widget *HistoryItem* pasando por parámetro la información del histórico del cultivo específico (*data*), el nombre del cultivo (*index*), y el tamaño del contenedor (*sizeContainer*), punto 20, ver Figura 3-34.
3. Dentro de *HistoryItem*, se creará la estructura UI en la aplicación que permitirá mostrar el gráfico. Posteriormente se envía toda la información de los históricos (*data*), el nombre del cultivo (*index*) y el alto del contenedor (*sizeContainer*) hacia el widget *LineGraph* (punto 21, ver Figura 3-34).
4. El widget *LineGraph* se encargará de preparar y estructurar la información del histórico y para que al momento de ser utilizada se presente de forma coherente en el gráfico, contendrá 2 elementos que ayudarán a realizar estas operaciones:
 - a. El método *_prepareData* será el encargado de preprocesar toda la información que llega a través de la variable *data*, elimina toda la información innecesaria y agrupa los elementos del histórico segmentando la información por día, punto 22, ver Figura 3-34.
 - b. El widget *_widgetDisplay* (punto 23, ver Figura 3-34), se encargará de crear la estructura del gráfico lineal, tomando como referencia la documentación de la librería *fl_chart*, e integrará los datos preparados con el método *_prepareData*.
5. Finalmente, cuando los datos han sido preprocesados y la estructura del gráfico ha sido establecida, será mostrado en la aplicación, utilizando el método *linearGraph* (punto 24, ver Figura 3-34).

CAPÍTULO 4

Metodología y casos de estudio

Capítulo 4 Implementación

Después de realizar un análisis de cada componente del proyecto, el siguiente paso es su implementación, de esta forma se podrá verificar su funcionamiento y posibles errores.

4.1 Prototipo hidropónico (Componente 1)

Siguiendo las especificaciones mencionadas en el punto 3-1 se realiza la construcción del prototipo hidropónico, ver Figura 4-1, una vez que está en operación es importante considerar los siguientes 2 puntos:

1. Cada vez que termine el proceso hidropónico y se obtenga la cosecha, se debe realizar una limpieza de la cama de cultivo, el contenedor y el sistema de drenaje. Por tal motivo no es recomendable agregar pegamento para PVC a todas las uniones, se recomienda el uso de cinta de aislar posibilitando el desensamble de algunos elementos.
2. Prestar mucha atención con las uniones de los coples y el sistema de drenaje, eliminando por completo las fugas de agua que pueda presentar el prototipo hidropónico.



Figura 4-1 Implementación del prototipo hidropónico.

4.2 Automatización hidropónica (Componente 2)

Dentro de la automatización hidropónica se deben integrar y configurar los diferentes módulos, sensores y actuadores junto con el microprocesador, que serán los encargados de monitorizar el entorno que rodea al prototipo, también debe realizarse el almacenamiento de cada valor obtenido del componente 2 dentro de una base de datos de forma adecuada.

4.2.1 Configuración Arduino IDE

Como se mencionó en el tema 2.2.2 el módulo central utilizado en el presente proyecto es *NodeMCU*, placa que permite ser programada utilizando *ARDUINO IDE*, sin embargo, al no ser una placa construida por la empresa *ARDUINO*, su configuración no viene precargada en el entorno de desarrollo, por tal motivo, es necesario integrar el microcontrolador *ESP8266* a las configuraciones del IDE para que éste lo reconozca, el procedimiento para realizar dicha acción es el siguiente:

1. Contar con una versión de *ARDUINO IDE* igual o superior a la 1.6.0.
2. Abrir el *ARDUINO IDE*.
3. Entrar en el menú Archivo → Preferencias.
4. En la ventana que se abre, localizar el apartado “Gestor de URLs adicionales de tarjeta” y añadir la siguiente URL
http://arduino.esp8266.com/stable/package_esp8266com_index.json, presionar *OK*.
5. Ir al menú Herramientas → Placa y buscar la opción que diga “Gestor de Tarjetas”.
6. Aparece una nueva ventana, y buscar la siguiente “esp8266 by ESP8266 Community”, dar doble clic para comenzar a instalar.
7. Reiniciar *Arduino IDE*.
8. Ir nuevamente al menú Herramientas → Placa, deberán aparecer varias placas nuevas, dentro de esta lista, buscar *NodeMCU 1.0 (ESP-12E Module)*, placa con la que se trabajará en el presente proyecto.

4.2.2 Configuración NodeMCU.

El módulo *NodeMCU*, es el componente principal del proceso de automatización, al tener integrado el microcontrolador *ESP8266*, todos los sensores que serán utilizados deben conectarse a él, ver Figura 3-6. En el tema 3.2.1 se configuró el *ARDUINO IDE* para reconocer el módulo, la siguiente acción es verificar que funcione correctamente, realizando una prueba sencilla.

1. Utilizar un cable USB para conectar el módulo *NodeMCU* a la computadora.
2. Seleccionamos la placa a utilizar, en el menú Herramientas → Placa → *NodeMCU 1.0 (ESP-12E Module)*.

3. Configuración la velocidad de subida, en el menú Herramientas → Upload Speed → *115200*.
4. Seleccionamos el Puerto, menú Herramientas → Puerto → Depende del computador, puede ser *COM1, COM4, COM5*.
 - a. Si el computador no reconoce el dispositivo, lo más probable es que este dañado o no tenga los controladores.
 - i. Buscar los controladores genéricos *CH340G*, instalarlos y repetir el proceso desde el paso 1.
 - ii. Cambiar de módulo *NodeMCU*.
5. Ir al menú Archivo → Ejemplos → 01.Basics → Blink.
6. Cargar el proyecto, menú Programa → Subir.

Cuando el archivo de ejemplo ha sido cargado de forma correcta en el microcontrolador, un *LED* integrado directamente en el módulo *NodeMCU* deberá comenzar a parpadear con una luz azul.

Los sensores utilizados en el presente proyecto funcionan con *5V*, como se indicó en el tema 2.2.2 el módulo *NodeMCU* funciona con *3V*, sin embargo, cuenta con un pin de salida de *5V*, este pin es *Vim*, para alimentar a todos los sensores, se crea una línea de corriente eléctrica que provea esos *5V*, de tal forma que todos los sensores deberán conectarse a ella. Un proceso similar ocurre con la conexión a tierra, con la diferencia de que el módulo si cuenta con múltiples pines *GND*, sin embargo, la mejor manera es utilizar una sola línea, y que los sensores también se conecten a ella, ver el diagrama de conexión entre sensores, módulos, actuadores y microcontrolador en la Figura 3-6.

4.2.3 Implementando la lectura de la humedad y temperatura

En el tema 3.2.1 se detalla el diagrama de conexión y el algoritmo propuesto para realizar la implementación del sensor *AM2302*, encargado de monitorizar la temperatura y humedad e integrarlo a *NodeMCU*, a continuación, se muestra el código fuente utilizado:

```

1 //Librerías
2 #include <DHT.h>
3 #include <DHT_U.h>
4 // Constantes y definiciones
5 #define DHTTYPE DHT21
6 #define DHTPIN 2
7 // Inicializaciones
8 DHT dht(DHTPIN, DHTTYPE);
9 // Variables
10 float temp, hum;
11 // Funciones
12 void obtenerTemperatura() {
13     delay(5000);
14     hum = dht.readHumidity();

```

```

15     temp = dht.readTemperature();
16     if(isnan(temp) || isnan(hum))
17         Serial.println("Fallo en la lectura del sensor AM2301");
18 }
19 // Configuración inicial
20 void setup() {
21     Serial.begin(115200);
22     dht.begin();
23 }
24 // Ciclo
25 void loop() {
26     obtenerTemperatura();
27 }

```

4.2.4 Implementando el temporizador

Se utiliza el módulo *AT24C32* para acceder a las funciones de fecha y hora que son utilizadas dentro del microcontrolador, para integrar el módulo con *NodeMCU*, se utiliza el diagrama, las conexiones y el algoritmo planteados en el tema 3.2.2, la implementación se realiza con el siguiente código:

```

1 // Librerías
2 #include "RTCLib.h"
3 // Constantes y definiciones
4 RTC_DS3231 TIMER;
5 // Configuración inicial
6 void setup() {
7     Serial.begin(115200);
8     if (!TIMER.begin()) {
9         Serial.println(F("No se pudo encontrar el RTC"));
10        while (1);
11    }
12 }
13 // Ciclo
14 void loop() {
15     // Fecha actual del microcontroller
16     DateTime now = TIMER.now();
17 }

```

4.2.5 Implementando el relevador

Una acción que el sistema debe ser capaz de realizar, es la circulación de la solución nutritiva dentro del prototipo hidropónico, para llevar a cabo esta tarea se utiliza una bomba de agua que siempre permanecerá conectada a una toma de corriente, mientras que el encendido y apagado de la misma será determinado por un relevador, elemento que será automatizado utilizando el temporizador implementado en el punto 4.2.4 o controlado desde la aplicación móvil. En el tema 3.2.3 se explica la conexión utilizada entre el relevador

y el módulo *NodeMCU*, también un algoritmo general de su funcionamiento, a continuación, se anexa el código fuente utilizado para llevar a cabo dicha tarea:

```
// Constantes y definiciones
1 #define PINBOMBA 14
2 // Variables
3 int estadoBomba;
4 // Funciones
5 void encenderBomba(DateTime date){
6     if(bombaManual == false){
7         if( date.hour() >= 7 && date.hour() <= 19 ){
8             if(date.minute() >= 0 && date.minute() <= 20){
9                 estadoBomba = LOW;
10                Serial.println("RTC bomba encendida");
11            }else{
12                estadoBomba = HIGH;
13                Serial.println("RTC bomba apagada");
14            }
15
16            digitalWrite(PINBOMBA, estadoBomba);
17        }else{
18            digitalWrite(PINBOMBA, HIGH); //Bomba manual false -> siempre
19 apagada a deshoras
20        }
21    }else{
22        Serial.println("Bomba siempre encendida");
23        digitalWrite(PINBOMBA, LOW); // Bomba manual true -> siempre
24 encendida;
25    }
26 }
27 // Configuración inicial
28 void setup(){
29     pinMode(PINBOMBA, OUTPUT);
30 }
31 // Ciclo
32 void loop(){
33     DateTime now = TIMER.now();
34     encenderBomba(now);
35 }
```

4.2.6 Implementando el sensor ultrasónico

Como se menciona en el tema 3.2.4, se utiliza un sensor ultrasónico *HC-SR04* para calcular la distancia entre el sensor y la solución nutritiva almacenada dentro del contenedor, también se explica el diagrama de conexión entre este sensor y el módulo *NodeMCU*.

Es de vital importancia tener en consideración que, el dato obtenido con el sensor ultrasónico es la distancia y no el nivel de agua del contenedor, sin embargo, a partir de la distancia, en un futuro, se puede calcular el nivel de agua que existe dentro del mismo, por lo tanto, en este punto de la automatización obtendremos únicamente la distancia, utilizando el siguiente código:

```
1 // Constantes y definiciones
2 #define PINTRIG 12
3 #define PINECHO 13
4 // Variables
5 float tiempo, distancia;
6 // Funciones
7 void obtenerNivelAgua() {
8     digitalWrite(PINTRIG, LOW);
9     delayMicroseconds(2);
10    digitalWrite(PINTRIG, HIGH);
11    delayMicroseconds(10);
12    digitalWrite(PINTRIG, LOW);
13
14    tiempo = pulseIn(PINECHO, HIGH);
15    distancia = (0.3432 / 2) * tiempo;
16 }
17 // Configuración inicial
18 void setup() {
19     pinMode(PINECHO, INPUT);
20     pinMode(PINTRIG, OUTPUT);
21 }
22 // Ciclo
23 void loop() {
24     obtenerNivelAgua();
25 }
```

4.2.7 Implementando el medidor de pH

El componente electrónico utilizado en el presente proyecto es el *pH-4502C*. La sonda de *pH* devuelve un valor de voltaje, sin embargo, utilizaremos la fórmula propuesta en el tema 3.2.5 donde también se muestra el diagrama de conexión entre la sonda y *NodeMCU*, permitiendo realizar la conversión de voltaje hacia los valores correctos de pH, a continuación, se muestra el código fuente implementado para llevar a cabo esta tarea.

```
1 // Constantes y definiciones
2 #define PINPH A0
3 // Variables
4 float pHValue;
5 // Funciones
6 void obtenerPH() {
7     voltage = analogRead(PINPH);
```

```

8     pHValue = sensorValue * (5.0 / 1023.0);
9 }
10 // Configuración inicial
11 void setup(){
12     pinMode(PINPH, INPUT);
13 }
14 // Ciclo
15 void loop(){
16     obtenerPH();
17 }

```

4.2.8 Lectura y escritura de datos en Firebase

Una vez que se ha implementado con éxito la lectura de los sensores que comprenden el componente 2, el último paso es guardar la información en una base de datos. Dentro de la implementación con *Firebase*, además de enviar la información de los sensores, se establecen las configuraciones y lecturas a la base de datos, entre otras cosas, en el punto 3.2.7 se detalla el diagrama de flujo propuesto, a continuación, se muestra su implementación en el programa almacenado dentro del módulo *NodeMCU*:

```

1 // Librerías
2 #include <ESP8266WiFi.h>
3 #include <FirebaseArduino.h>
4 #include <ArduinoJson.h>
5
6 // Constantes y definiciones
7 #define PROTOTIPO "nombre_prototipo"
8 // Credenciales Firebase
9 #define FIREBASE_HOST "proyecto.host"
10 #define FIREBASE_AUTH "tokenAuth"
11 // Credenciales Wi-Fi
12 #define WIFI_SSID "ssid"
13 #define WIFI_PASSWORD "passwd"
14
15 // Funciones
16 void configuraciones(){
17     FirebaseContenedor();
18     FirebaseEstadoBomba();
19     FirebaseBombaManual();
20 }
21
22 void sensoresFirebase(String id, DateTime date){
23     String fecha = String(date.year()) + "-" + String(date.month()) + "-" +
24 String(date.day());
25     String hora = String(date.hour()) + ":" + String(date.minute());
26     String pathSensores = "/info_tekax/" + String(PROTOTIPO) + "/" + id
27 + "/logs_sensores/" + fecha;
28     String fecha_hora = fecha + "*" + hora;

```

```

29
30   StaticJsonBuffer<200> jsonBuffer;
31   JsonObject& obj = jsonBuffer.createObject();
32   obj["temperatura"] = temp;
33   obj["humedad"] = hum;
34   obj["distancia"] = dist;
35   obj["pH"] = pHValue;
36   obj["fecha_hora"] = fecha_hora;
37
38   Firebase.set( "/info_tekax/"+ String(PROTOTIPO) +"/sensores", obj);
39   if (Firebase.failed()) {
40       Serial.print("Actualizando datos actuales de los sensores en
41 firebase");
42       return;
43   }
44
45   if( date.minute()%50 == 0 ){
46       Firebase.push( pathSensores, obj );
47       if (Firebase.failed()) {
48           Serial.print("Error al almacenar la información de los sensores en
49 Firebase");
50           return;
51       }
52   }
53 }
54
55 // Configuración inicial
56 void setup(){
57   WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
58   while( WiFi.status() != WL_CONNECTED ){
59       delay(500);
60       Serial.print(".");
61   }
62   Serial.println("WiFi Conectado!");
63   Serial.println(WiFi.localIP());
64   Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
65   configuraciones();
66 }
67
68 // Ciclo
69 void loop(){
70   String nombreCultivo = Firebase.getString( "/info_tekax/"+
71 String(PROTOTIPO) +"/configs/cultivo_actual/nombreCultivo");
72   if (Firebase.failed()) {
73       Serial.print("Configuración nombre cultivo");
74       return;
75   }
76
77   String idFecha = Firebase.getString( "/info_tekax/"+ String(PROTOTIPO)
78 +"/configs/cultivo_actual/idFecha");

```

```

79     if (Firebase.failed()) {
80         Serial.print("Configuración nombre cultivo");
81         return;
82     }
83     delay(1000);
84
85     if(nombreCultivo != "" && idFecha != ""){
86         String fechaCultivo = idFecha + "_" + nombreCultivo;
87
88         if( ctrlFechaCultivo != fechaCultivo ){
89             Firebase.setString( "/info_tekax/" + String(PROTOTIPO) + "/" +
90 fechaCultivo + "/fechaInicio", idFecha);
91             if (Firebase.failed()) {
92                 Serial.print("Nuevo cultivo error al insertar en la db");
93                 return;
94             }
95             ctrlFechaCultivo = fechaCultivo;
96         }
97         sensoresFirebase(fechaCultivo, now);
98     }else{
99         Serial.println("Esperar a que se configuren cosas");
100    }
101
102    delay(1000);
103 }

```

4.3 Base de datos

Para dar solución a los cuestionamientos mencionados en el tema 3.3, se realiza la implementación y construcción de la estructura planteada, realizando las pruebas necesarias para su correcto acceso.

4.3.1 Dispositivos de hardware disponibles

Cuando el componente 2, contiene las credenciales a *Firebase* y Wi-Fi correctas, comienza a enviar información hacia la base de datos, posteriormente, crea un documento con el valor de la constante *PROTOTIPO*, el documento generado a partir de ahora será conocido como *nombreDispositivo*, dentro de este documento se almacenará toda la información recolectada por el componente 2, ver Figura 4-2.

Para conocer los dispositivos a los que un usuario se puede conectar, simplemente debe acceder al documento *info_tekax*, y listar las llaves de los elementos que aparezcan en lugar de *nombreDispositivo*, ver Figura 4-2.



Figura 4-2. Dispositivos de hardware (componente 2) disponibles.

4.3.2 Usuario vinculado

Si se desea emparejar un usuario con el componente 2, se utiliza el documento de *vinculaciones* de la base de datos, para realizar este registro solo se debe crear un objeto que tenga como llave el identificador de usuario y en su interior el nombre del dispositivo a vincular ej. *idUsuario { device: 'nombre_dispositivo_comp_2' }*, ambos elementos deberán ser de tipo *String*, ver Figura 4-3.



Figura 4-3. Vinculación entre usuario y dispositivo en la base de datos.

4.3.3 Cultivo actual

Dentro del documento raíz *nombreDispositivo*, existe un documento hijo llamado *configs* que almacena las configuraciones establecidas para el dispositivo emparejado, dentro de este último documento existe otro llamado *cultivo_actual*, que almacena la información que su nombre indica, es determinado por el usuario y solo consta de 2 elementos, el nombre del cultivo y la fecha de inicio, ambos elementos son de tipo *String*, ver Figura 4-4.



Figura 4-4. Almacenamiento del cultivo actual en la base de datos.

4.3.4 Listado de cultivos por dispositivo

Los cultivos que ha realizado el dispositivo, están almacenados dentro del documento *nombreDispositivo*, ver Figura 4-5, su característica principal es que son elementos que siguen la regla:

$$\text{cultivo} = \text{idFecha_NOMBRECULTIVO}$$



Figura 4-5. Listado de cultivos del usuario.

4.3.5 Valor actual de los sensores

Para conocer el valor almacenado de cada sensor integrando al componente 2, solo se necesita acceder al documento *sensores*, que está dentro del documento *nombreDispositivo*, ver Figura 4-6.



Figura 4-6. Valor almacenado en la BD de cada sensor del componente 2.

Dentro de los valores que *NodeMCU* almacena en la base de datos, es posible encontrar los campos *distancia*, *humedad*, *pH* y *temperatura* que son de tipo *Float*, mientras que el campo *fecha_hora*, es un campo concatenado de tipo *String*, utilizando la siguiente regla:

$$fecha_hora = yyyy - m - d ** h:m$$

4.3.6 Estado de la bomba de agua

El documento llamado *configs* almacena la información del estado de la bomba en la variable *estadoBomba*, el tipo de dato que utiliza es *Integer*, 1 significa que esta encendida y 0 que esta apagada, ver Figura 4-7.

Es importante considerar que el valor *estadoBomba* es actualizado de 2 formas totalmente diferentes:

1. Si el atributo *bombaManual*, ver Figura 4-7, está en *false*, significa que el componente 2 utilizará el temporizador integrado y activará la bomba 20 minutos cada hora, posteriormente será desactivada de forma automática, como se explica en el tema 3.2.5.
2. Si el atributo *bombaManual*, está en *true*, significa que el usuario puede controlar la bomba de forma manual, utilizando la aplicación móvil el usuario podrá encender o apagar la bomba por tiempo indeterminado, intercambiando este atributo de forma dinámica.

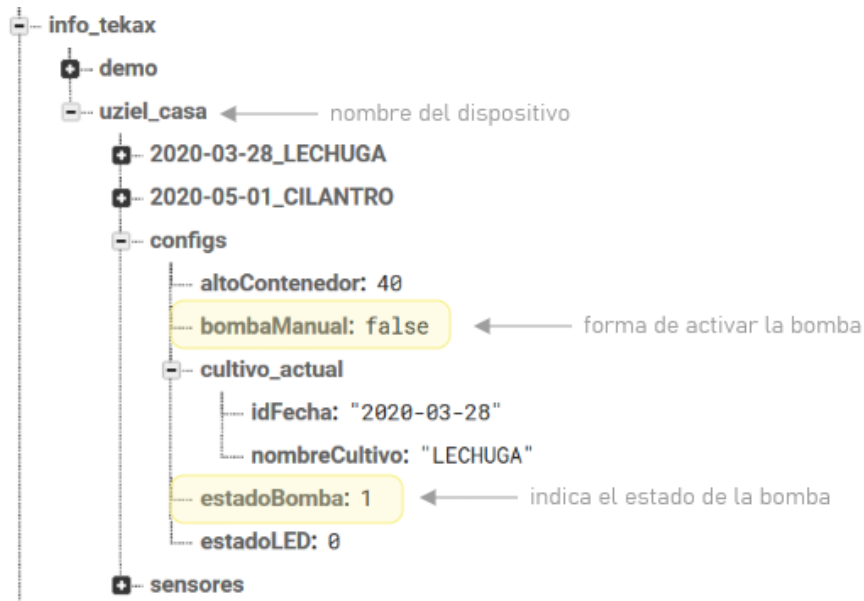


Figura 4-7. Obtener el estado de la bomba.

4.3.7 Histórico de cultivos

Cuando un usuario agrega un nuevo cultivo, se genera un documento que puede ser encontrado en el listado de dispositivos, como se explicó en el tema 3.3.4, dentro de este documento agregado, se almacenará la información de su histórico, al llegar el minuto 50 de cada hora, se actualiza su documento hijo llamado *logs_sensores*, que guarda el objeto que contiene la información de los sensores proveniente del componente 2, separando la información por fecha, el Algoritmo 4-1, muestra el detalle de la operación realizada:

Inicio

- Si minuto = 50 entonces
 - Insertar fecha actual en documento *logs_sensores*
 - Insertar objeto sensores en documento fecha actual dentro de *logs_sensores*
- Si no
 - Guardar objeto sensores en documento *sensores*

Fin

Algoritmo 4-1. Almacenar información de históricos.

La información almacenada en el documento *log_sensores*, es estructurada de la siguiente manera, ver Figura 4-8.

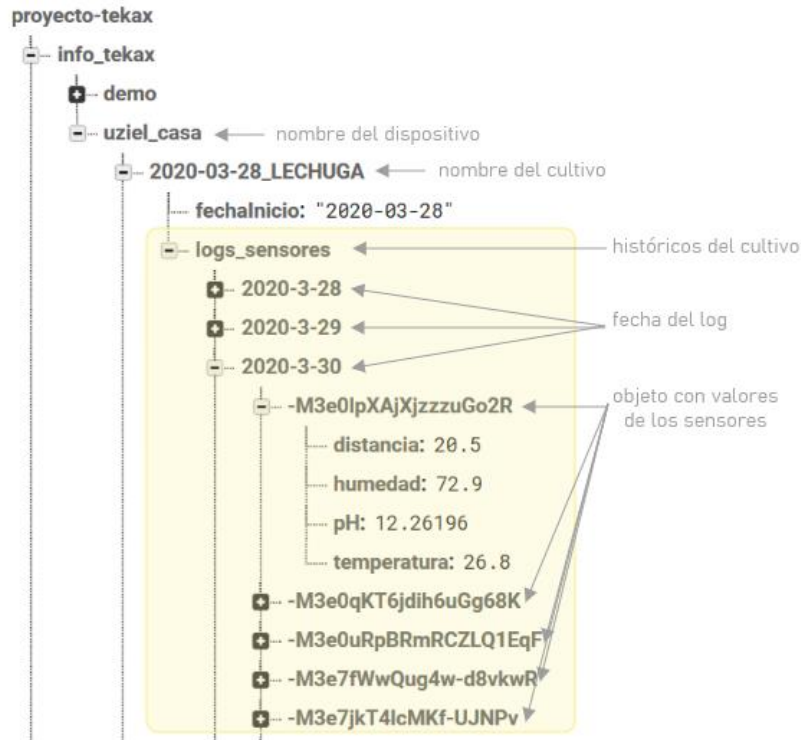


Figura 4-8. Histórico de cultivos en la base de datos.

4.3.8 Usuario autenticado

Cuando un usuario se autentica, la información recibida es almacenada dentro del documento *usuarios*, si se utiliza el método tradicional ingresando usuario y contraseña, el objeto a guardado cumple las siguientes características:

$$idUsuario = \{ correo: "correoDelUsuario", \quad lastSignIn: "FechaHora" \}$$

Si el usuario se autentica utilizando credenciales de Google, se almacena un objeto con la siguiente estructura:

```

idUsuario = {
  avatar: "urlImagen",
  correo: "correoDelUsuario",
  lastSignIn: "FechaHora",
  nombre: "nombreCompleto"
}

```

El tipo de dato para los elementos *idUsuario*, *avatar*, *correo* y *nombre* es de tipo *String*, mientras que el tipo de dato para *lastSingIn* es *DateTime*, ver Figura 4-9.



Figura 4-9. Almacenamiento de usuarios en la base de datos.

4.4 Aplicación móvil

Es el último componente del presente proyecto de tesis, permite visualizar y controlar la información en tiempo real de lo ocurrido en el prototipo hidropónico, en la Figura 3-1, se puede observar la arquitectura de componentes del proyecto Tekax. El microcontrolador y la aplicación móvil, se encargan de enviar y recibir información a la base de datos, cuando se detecta un cambio, la aplicación móvil realiza una actualización en tiempo real permitiendo la visualización de los mismos.

Con el dispositivo electrónico (componente 2) se obtienen los valores de cada sensor y se almacenan en la base de datos (componente 3), la aplicación móvil (componente 4) funciona como un complemento permitiendo la administración del prototipo hidropónico, realizando acciones que el componente 2 no puede, por ejemplo: solo usuarios permitidos pueden acceder a su información, vincular/desvincular un usuario con un componente 2, crear cultivos, visualizar la información de los sensores, visualizar el histórico de los cultivos, activar la bomba de agua, establecer el tamaño del contenedor y cerrar sesión. En el tema 3.4 se realiza un análisis de cada una de las secciones que se integrarán a la aplicación móvil, basándose en el diagrama de casos de uso, ver Figura 3-11.

4.4.1 Instalación y configuración de Flutter

Como se menciona en el tema 2.4.8, la construcción de la aplicación móvil utiliza *Flutter*, un marco de desarrollo creado por la empresa Google, esta tecnología brinda todos los elementos necesarios para la creación de una aplicación móvil moderna y de calidad, la instalación fue realizada en un computador con el sistema operativo *Windows*, sin embargo, funciona para *OS* y *Linux*. Los requerimientos básicos para su instalación son los siguientes:

Elemento	Requisito
Sistema Operativo	Windows 7 SP1 (Profesional, Empresarial, Estudiante) de 64-bit
Espacio en disco	400 Mb (No incluye IDE ni herramientas)
Herramientas	Windows Power Shell 5.0

Tabla 4-1. Requerimientos para la instalación de Flutter.

El IDE que se utiliza para el desarrollo de la aplicación móvil es *Visual Studio Code*, los pasos para integrar Flutter son los siguientes:

1. Se crea la siguiente carpeta *C:/Android*
2. Se descargan los binarios de Java para el funcionamiento de la herramienta, se utiliza el siguiente enlace: https://github.com/AdoptOpenJDK/openjdk8-binaries/releases/download/jdk8u202-b08/OpenJDK8U-jdk_x64_windows_hotspot_8u202b08.zip
3. Se descomprime la información en la siguiente ruta *C:/Android/openjdk*
4. Se descarga el SDK⁵⁹ de Flutter de la siguiente URL <https://flutter.dev/docs/get-started/install/windows>
5. Descomprimir el SDK de Flutter en la siguiente ruta *C:/Android/flutter*
6. Descargar las herramientas de Android, desde la siguiente URL: <https://developer.android.com/studio/#command-tools>
7. Descomprimir las herramientas de Android en la siguiente ruta *C:/Android/tools*
8. Agregar los directorios creados a las variables de entorno, ejecutar desde la terminal los siguientes comandos:
 - a. `setx JAVA_HOME "C:\Android\openjdk"`
 - b. `setx ANDROID_HOME "C:\Android"`
 - c. `setx ANDROID_SDK_ROOT "C:\Android\tools"`
 - d. `setx path %path%;"C:\Android\sdk;C:\Android\tools\bin;C:\Android\flutter\bin"`
9. Descargar la imagen de Android a utilizar, en este caso en particular se trabajará con Android 8, se abre la terminal y se ejecutan los siguientes comandos:
 - a. `sdkmanager "system-images;android-27;default;x86_64" "platforms;android-28"`
 - b. `sdkmanager "platform-tools"`
 - c. `sdkmanager "build-tools;28.0.3"`
 - d. `sdkmanager "platforms;android-28"`
 - e. `sdkmanager emulator`
10. Aceptar las licencias de Android, ejecutar el siguiente comando `sdkmanager --licenses`
11. Indicar a Flutter donde se encuentra el SKD, ejecutando el siguiente comando `flutter config --android-sdk C:/Android`
12. Ejecutar el comando `flutter doctor -v` para verificar que todo esté en orden, ver Figura 4-10.

⁵⁹ Por sus siglas en inglés Software Development Kit, es generalmente un conjunto de herramientas de desarrollo de software, que permite a un desarrollador crear una aplicación informática para un sistema concreto.

```

c:\Android>flutter doctor -v
[✓] Flutter (Channel stable, v1.9.1+hotfix.2, on Microsoft Windows [VersiÃ³n 10.0.17763.737], locale es-MX)
    • Flutter version 1.9.1+hotfix.2 at C:\Android\flutter
    • Framework revision 2d2a1ffec9 (9 days ago), 2019-09-06 18:39:49 -0700
    • Engine revision b863200c37
    • Dart version 2.5.0

[✓] Android toolchain - develop for Android devices (Android SDK version 28.0.3)
    • Android SDK at C:/Android
    • Android NDK location not configured (optional; useful for native profiling support)
    • Platform android-28, build-tools 28.0.3
    • ANDROID_HOME = C:\Android
    • ANDROID_SDK_ROOT = C:\Android\sdk
    • Java binary at: C:\Android\openjdk\bin\java
    • Java version OpenJDK Runtime Environment (AdoptOpenJDK)(build 1.8.0_202-b08)
    • All Android licenses accepted.

```

Figura 4-10. Verificando la instalación de Flutter.

4.4.2 Estructura superior de widgets

Como se observa en el diagrama de casos de uso, ver Figura 3-11, la mayoría de las operaciones que se realicen dentro de la aplicación móvil, deben contar con previa autenticación, por tal motivo, se diseñó una estructura que permite implementar y controlar este tipo de restricción. En el punto 3.4.2 se muestra el análisis de los widgets a utilizar y se diseñó un diagrama de secuencia que permitirá la implementación esta sección.

Para la redacción del presente documento, en las partes de código fuente, cuando se encuentren puntos suspensivos (...), se indica que existe más código en esa región, sin embargo, se ha ocultado debido a que no es tan relevante para la explicación del funcionamiento.

Archivo wrapper.dart

```

1  ...
2  class Wrapper extends StatelessWidget {
3    UserBloc userBloc;
4    @override
5    Widget build(BuildContext context) {
6      userBloc = BlocProvider.of(context);
7      return _handleCurrentSession();
8    }
9
10   Widget _handleCurrentSession() {
11     return StreamBuilder(
12       stream: userBloc.authStatus,
13       builder: (BuildContext context, AsyncSnapshot snapshot) {
14         if(!snapshot.hasData || snapshot.hasError){
15           return LoginView();
16         } else {
17           return InitView();
18         }
19       }
20     );
21   }

```


22 }

El widget *Wrapper* se encarga de manejar la sesión activa del usuario, se utiliza la variable *userBloc* (línea 6) para encapsular todos los métodos que pertenecen al patrón de diseño separando la lógica del negocio, posteriormente se utiliza el *StreamBuilder* (línea 11) que permanece escuchando los cambios que existan dentro del atributo *authStatus* (línea 12), dependiendo de su valor redireccionará al widget *LoginView* (línea 15) si el usuario esta autenticado, o al widget *InitView* (línea 17) si el usuario no cuenta con las credenciales necesarias.

Archivo login_view.dart

```
1 ...
2 class LoginView extends StatefulWidget {
3   @override
4   _LoginViewState createState() => _LoginViewState();
5 }
6 class _LoginViewState extends State<LoginView> {
7   bool showSignIn = true;
8   void toggleView() {
9     setState( () => showSignIn = !showSignIn );
10  }
11  @override
12  Widget build(BuildContext context) {
13    if(showSignIn){
14      return SignInScreen(toggleView: toggleView);
15    }else{
16      return RegisterScreen(toggleView: toggleView);
17    }
18  }
19 }
```

Se utiliza una variable booleana llamada *showSignIn* (línea 7) para cambiar el widget a mostrar, dependiendo de su valor, se muestra la ventana de inicio de sesión (*SignInScreen*) o la ventana de registro (*RegisterScreen*), como se aprecia en el código fuente de las líneas 13 - 17, finalmente el método *toggleView* (línea 8) es el encargado de realizar el cambio en el valor de la variable *showSignIn*.

4.4.3 Nuevo usuario

Es necesario implementar una sección que permita al usuario registrarse para poder tener acceso a la aplicación móvil, esta acción es la única permitida sin estar autenticado, como se muestra en el diagrama de casos de uso del punto 3-11.

En el tema 3.4.3 se detalla la estructura de widgets y el diagrama de secuencia utilizado para llevar a cabo esta implementación, partiendo del widget *RegisterScreen*.

Archivo register_screen.dart

```
1 ...
2
3 class _RegisterScreenState extends State<RegisterScreen> {
4   @override
5   Widget build(BuildContext context) {
6
7     return Scaffold(
8       body: ...,
9       children: <Widget> [
10        RegisterForm(),
11        ...
12      ]
13    );
14  }
15 }
```

El widget *RegisterScreen* integra toda la estructura del UI para implementar el registro de usuario, en la línea 10 se realiza el llamado al widget *RegisterForm*.

Archivo register_form.dart

```
1 ...
2 class _RegisterFormState extends State<RegisterForm> {
3   ...
4   Widget _inputEmail() {...}
5   Widget _inputPassword() {...}
6   Widget _inputRptPassword() {...}
7   Widget _buttonRegister() {
8     return Container(
9       ...
10      child: InkWell(
11        onTap: () async {
12          if(_formKey.currentState.validate()) {
13            userBloc.registerUser(newUser, email,
14 password).then((FirebaseUser user) => print("El usuario es:
15 ${user.toString()}"));
16          }
17        },
18        ...
19      )
20    );
21  }
22  @override
23  Widget build(BuildContext context) {
24    ...
25    final form = Form(
26      key: _formKey,
27      child: Column(
28        children: <Widget>[
```

```

29         ...
30         FadeIn(0.4, _inputEmail()),
31         FadeIn(0.6, _inputPassword()),
32         FadeIn(0.8, _inputRptPassword()),
33         FadeIn(1, _buttonRegister())
34     ]
35 )
36 );
37 return form;
    }
}

```

El widget *RegisterForm* es el encargado de contener el formulario que se utiliza para crear un nuevo registro, se agregan los widgets que contienen el diseño de los campos para la entrada de: correo electrónico (*_inputEmail*, línea 4), contraseña (*_inputPassword*, línea 5), repetir contraseña (*_inputRptPassword*, línea 6). El widget *_buttonRegister* (línea 7) funciona como un botón, que al ser presionado ejecuta el método *registerUser* (línea 13) perteneciente a la clase *userBloc* integrando el patrón de diseño. Finalmente, todo el formulario es encapsulado dentro del widget *Form* (líneas 25 – 36), que contiene el llamado a los widgets *_inputEmail*, *_inputPassword*, *_inputRptPassword* y *_buttonRegister*.

Cuando se presiona el botón creado con el widget *_buttonRegister*, se activa el método *onTap* (línea 11) que ejecuta una función asíncrona y realiza lo siguiente:

1. Verifica que el formulario este validado, corroborando la siguiente información:
 - a. El correo tenga formato de correo electrónico
 - b. La contraseña sea mayor o igual a 6 caracteres
 - c. El campo contraseña y repetir contraseña deben ser iguales
2. Si el paso 1 es correcto entonces se ejecuta el método *registerUser* (línea 13) pasando por parámetro: *email* y *password*.
3. Si el paso 1 es incorrecto, se agrega un estilo diferente a los campos que tengan error y se visualiza en la pantalla de la aplicación.

Archivo *bloc_user.dart*

```

1 ...
2 class UserBloc implements Bloc {
3   final _authRepository = AuthRepository();
4   ...
5   Future<FirebaseUser> registerUser(String email, String password) {
6     return _authRepository.registerFirebase(email, password);
7   }
8 }

```

Este archivo es el encargado de implementar el patrón de diseño *BLoC* y contiene la clase *UserBloc*, como se mencionó en el tema 2.4.3, este patrón, se encarga de separar la lógica de negocio de la vista. Los widgets previamente mencionados *RegisterScreen* y

RegisterForm, contienen la parte visual, mientras que lo que suceda a partir del archivo *bloc_user.dart*, formará parte del acceso a la información y diferentes repositorios, esto se conoce como la lógica de negocio.

En la línea 2 se crea la clase *UserBloc* y se utiliza el método *implements*⁶⁰, implementando todos los métodos pertenecientes a la clase *Bloc*, que es el patrón de diseño utilizado en todo el proyecto. Posteriormente dentro de la clase *UserBloc* (línea 5), se crea el método *registerUser* (línea 5) pasando por parámetro: *email* y *password*. Finalmente se utiliza el objeto *_authRepository* (línea 7) ejecutando el método *registerFirebase* pasando por parámetro: *email* y *password*.

Archivo *auth_repository.dart*

```
1 ...
2 class AuthRepository {
3   final _firebaseAuthAPI = FirebaseAuthAPI();
4   ...
5   Future<FirebaseUser> registerFirebase(String email, String password) {
6     return _firebaseAuthAPI.registerWithEmailAndPassword(email, password);
7   }
8   ...
9 }
```

El widget *AuthRepository* se encarga de realizar las peticiones hacia la API de Firebase que tienen relación con la autenticación, primero se crea un objeto de tipo *FirebaseAuthAPI* (línea 3) y posteriormente se crea el método *registerFirebase* (línea 5) que recibe como parámetro: *email* y *password*, finalmente al ser llamado, ejecuta del objeto *_firebaseAuthAPI* (línea 6) el método *registerWithEmailAndPassword* pasando por parámetro nuevamente: *email* y *password*. Este último método se comunica con el servidor de Firebase, procesa la información y devuelve un objeto de tipo *FirebaseUser* con el resultado de la operación.

4.4.4 Autenticación

Para implementar la autenticación, se utiliza el diagrama de secuencia propuesto en el tema 3.4.4. Cuando la estructura superior de widgets ha sido implementada en el apartado 4.4.2, se cuenta con el widget *LoginView*, la variable *showSignIn* por defecto comienza en verdadero (*true*), esta acción permite que se muestre el widget *SignInScreen* (inicio de sesión), por otra parte, si su valor es falso (*false*), se mostrará el widget *RegisterScreen* (nuevo usuario) que fue documentado en el punto 4.4.3 del presente trabajo.

⁶⁰ Es un método propio del lenguaje Dart, sirve para implementar clases, funciona como la herencia de otros lenguajes como C o JAVA.

Cuando la aplicación se ejecuta y el usuario no ha iniciado sesión, el sistema mostrará el widget *SignInScreen*, permitiendo seleccionar el tipo de acceso deseado, a continuación, se muestra la implementación realizada para llevar a cabo esta acción.

Archivo `sign_in_screen.dart`

```
1 ...
2 class SignInScreen extends StatefulWidget {
3   final Function toggleView;
4   SignInScreen({ this.toggleView });
5   ...
6 }
7 class _SignInScreenState extends State<SignInScreen> {
8   @override
9   Widget build(BuildContext context) {
10    return Scaffold(
11      body: LayoutBuilder(builder: (context, constraints) {
12        return SingleChildScrollView(
13          child: ConstrainedBox(
14            constraints: BoxConstraints(minWidth: constraints.maxWidth,
15 minHeight: constraints.maxHeight),
16            child: IntrinsicHeight(
17              child: Container(
18                ...
19              child: Column(
20                children: <Widget>[
21                  SignInForm(),
22                  FadeIn(0.8, WaysLogin()),
23                ],
24              ),
25            ),
26          )
27        );
28      );
29    });
30  );
31 }
```

El widget *SignInScreen* contiene la estructura de widgets necesarios para implementar de forma exitosa el UI diseñado para esta sección, dentro del mismo, hay 2 widgets que deben ser analizados por separado:

- *SignInForm* (línea 21), contiene la estructura del formulario de inicio de sesión que solicitará información al servidor de Firebase.
- *WaysLogin*(línea 22), contiene la sección establecida para agregar métodos de autenticación, utilizando un mecanismo de terceros, ej. Google, Facebook, Twitter, entre otros.

Archivo signin_form.dart

```
1 ...
2 class _SignInState extends State<SignInForm> {
3   UserBloc userBloc;
4   final _formKey = GlobalKey<FormState>();
5   String email = '', password = '', error = '';
6   Widget _inputEmail () {...}
7   Widget _inputPassword () {...}
8   Widget _buttonSignIn() {
9     return Container(...),
10    child: InkWell(
11      onTap: () {
12        if(_formKey.currentState.validate()) {
13          userBloc.signIn(emailAndPassword, email,
14 password).then((FirebaseUser user) {
15            userBloc.updateUserData(User(
16              uid: user.uid,
17              name: null,
18              email: user.email,
19              photoUrl: null
20            ));
21          });
22        }
23      },
24      ...
25    )
26  );
27 }
28 Widget build(BuildContext context){
29   userBloc = BlocProvider.of(context);
30   final form = Form(
31     key: _formKey,
32     child: Column(
33       children: <Widget>[
34         FadeIn(0.5, _inputEmail()),
35         FadeIn(0.6, _inputPassword()),
36         FadeIn(0.7, _buttonSignIn())
37       ],
38     ),
39   );
40   return form;
41 }
```

Al ingresar al widget *SignInForm* lo primero que se realiza es crear una instancia de la clase que almacena nuestro patrón de diseño *userBloc* (línea 3), posteriormente se crean los widgets *_inputEmail* e *_inputPassword*, que contienen el diseño y funcionalidad necesaria para mostrar el correo electrónico (línea 6) y contraseña (línea 7) respectivamente, también se anexa el widget *_buttonSignIn* (línea 8) encargado de mostrar

el botón de inicio de sesión, finalmente se crea el formulario (líneas 29 – 38), que integra los widgets previamente creados `_inputEmail`, `_inputPassword` y `_buttonSignIn`.

Cuando el usuario presiona el botón `_buttonSignIn` se activa el método `onTap` (línea 11) y se realiza una validación, que verifica la existencia de información en el campo `_inputEmail` e `_inputPassword`, si todo está correcto entonces, se ejecuta el método `signIn` (línea 13) del objeto `userBloc` previamente instanciado, pasando por parámetro los valores de: `'emailAndPassword'`, `email` y `password`.

Archivo `ways_login.dart`

```
1  ...
2  class _WaysLoginState extends State<WaysLogin> {
3    UserBloc userBloc;
4    @override
5    Widget build(BuildContext context) {
6      userBloc = BlocProvider.of(context);
7      final waysLogin = Row(
8        mainAxisAlignment: MainAxisAlignment.center,
9        children: <Widget>[
10       SizedBox(width: 15.0),
11       ImageInCircle(
12         ...
13         onPressed: () {
14           userBloc.signIn('google', null, null).then((FirebaseUser user) {
15             userBloc.updateUserData(User(
16               uid: user.uid,
17               name: user.displayName,
18               email: user.email,
19               photoUrl: user.photoUrl
20             ));
21           });
22         }
23       ),
24     ],
25   );
26   return waysLogin;
27 }
```

El widget `WaysLogin`, es el encargado de mostrar el UI con las alternativas de inicio de sesión posteriores al método tradicional, en este punto, solo se ha implementado el inicio de sesión con Google, sin embargo, la estructura del widget está diseñada para integrar algún otro tipo de autenticación, como Facebook, Twitter, Github, entre otros, realizando ajustes sencillos.

El widget, crea una instancia de la clase `UserBloc` (línea 3), permitiendo tener acceso a los métodos del patrón de diseño creado, posteriormente se realiza un llamado al método

signIn (línea 13) pasando por parámetro: *'google', null, null*. A partir de este punto, tanto el acceso tradicional como el inicio de sesión de terceros, siguen la misma ruta.

Archivo `bloc_user.dart`

```
...
1 class UserBloc implements Bloc {
2   final _authRepository = AuthRepository();
3   final _userFirestoreRepository = UserFirestoreRepository();
4   Future<FirebaseUser> signIn(String typeLogin, String email, String password)
5 {
6   return _authRepository.signInFirebase(typeLogin, email, password);
7 }
8 void updateUserData(User user) =>
9 _userFirestoreRepository.updateUserDataFirestore(user);
}
```

Dentro de la clase *UserBloc*, se agrega un método llamado *signIn* (línea 4) que recibe por parámetro 3 valores: *typeLogin*, *email* y *password*. La variable *typeLogin* indica el tipo de inicio de sesión que será realizado. Cuando el método *signIn* es llamado, ejecuta el método *signInFirebase* perteneciente al objeto *_authRepository* (línea 7) enviando por parámetro las mismas 3 variables recibidas: *typeLogin*, *email* y *password*.

Archivo `auth_repository.dart`

```
...
1 class AuthRepository {
2   final _firebaseAuthtAPI = FirebaseAuthAPI();
3
4   Future<FirebaseUser> signInFirebase(String typeLogin, String email, String
5 password) {
6     switch(typeLogin){
7       case 'google': return _firebaseAuthtAPI.signInWithGoogle(); break;
8       case 'emailAndPassword': return
9 _firebaseAuthtAPI.signInWithEmailAndPassword(email, password); break;
10    }
11    return null;
12  }
}
```

Dentro de la clase *AuthRepository* se agrega un nuevo método llamado *signInFirebase* (línea 4), que permitirá realizar una petición hacia los servidores de Firebase, para realizar esta acción, primero es necesario crear una instancia de la clase *FirebaseAuthAPI* (línea 2) y posteriormente ejecutar el método deseado.

La aplicación móvil permite realizar la autenticación utilizando 2 formas de acceso diferentes:

1. Autenticación tradicional. Utilizando el correo electrónico y contraseña registrados en el punto 4.4.3. Si el valor de la variable `typeLogin` (línea 6) contiene `'emailAndPassword'` se ejecuta el método `signInWithEmailAndPassword` (línea 9) del objeto `_firebaseAuthAPI`.
2. Autenticación Google. Utiliza las credenciales de la cuenta de Google del usuario. Si el valor de la variable `typeLogin` (línea 6) es `'google'`, se ejecuta el método `signInWithGoogle` (línea 7), esta acción activará una nueva ventana, que preguntará al usuario sus credenciales, al presionar el botón verificar, las envía a sus servidores, finalmente cuando la información es procesada, devuelve un objeto con la respuesta.

Cuando una acción del paso 1 o 2 es ejecutada, la *API* de Firebase procesa internamente las respuestas y devuelve un único objeto de tipo *FirebaseUser*.

Cuando el objeto de *FirebaseUser* es retornado hacia los métodos que ejecutaron la petición *SignInForm* o *WayLogin*, se ejecuta el método `updateUserData` del patrón de diseño, enviando por parámetro el objeto `user`:

1. Autenticación tradicional. Retorna el objeto `user` con los atributos `uid` y el `email`.
2. Autenticación por Google. Retorna el objeto `user` con los atributos `uid`, `name`, `email`, `photoUrl`.

Archivo `user_firestore_api.dart`

```

1 ...
2 class UserFirestoreAPI {
3   static const String USERS = 'usuarios';
4   final dbRef = FirebaseDatabase.instance.reference();
5   void updateUserData(User user) async {
6     String setPath = "$USERS/${user.uid}";
7     await dbRef.child(setPath).set({
8       'nombre': user.name,
9       'correo': user.email,
10      'avatar': user.photoUrl,
11      'lastSignIn': DateTime.now().toString()
12    });
13  }
14 }

```

La clase *UserFirestoreAPI* contiene el método `updateUserData` (línea 5) que permite actualizar la información de un usuario específico dentro de Firebase, recibe como parámetro un objeto de tipo *User*, con los atributos: `uid`, `name`, `email`, `photoUrl`.

4.4.5 Patrón de diseño

En el tema 3.4.5 se plantea el diseño que será utilizado dentro del proyecto para organizar los archivos y tener la arquitectura *BLoC*, permitiendo tener una estructura limpia y adecuada, posibilitando un mantenimiento mucho más sencillo, ver Figura 4-11.

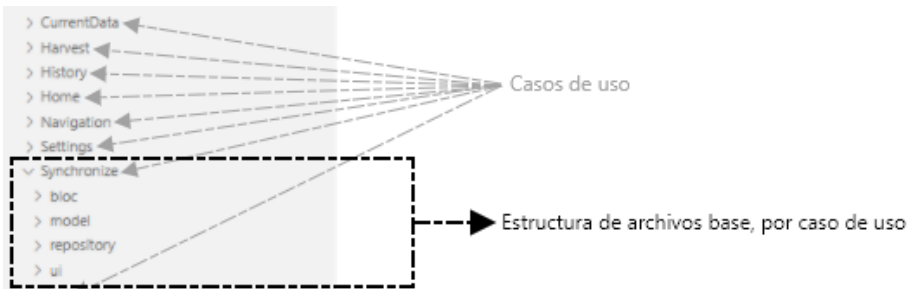


Figura 4-11 Estructura de carpetas del proyecto Tekax.

Para cada caso de uso, se utiliza la siguiente estructura de archivos: *bloc*, *model*, *repository* y *ui*. Como se describió en el tema 3.2.5, cada carpeta almacena un tipo de archivo determinado, es necesario conocer la acción que se desea realizar e integrarlos dentro de su carpeta correspondiente, ver Figura 4-12:

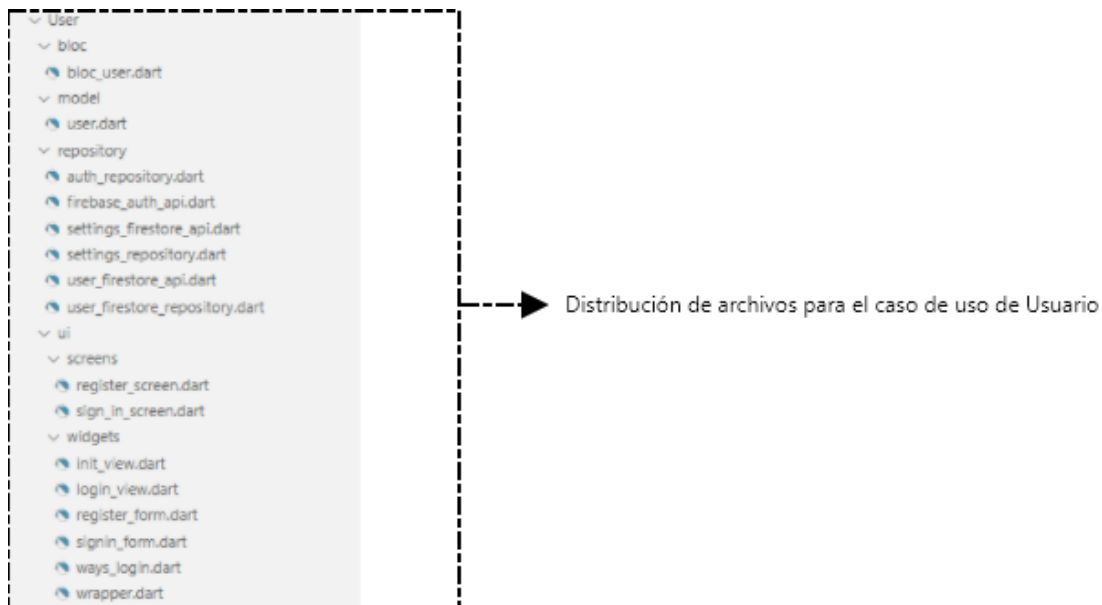


Figura 4-12 Asignación de archivos dentro de las carpetas del patrón BLoC.

4.4.6 Emparejar dispositivo

La función emparejar dispositivo, sirve para vincular el usuario con el componente 2, partiendo del análisis realizado en el tema 3.4.6, se utiliza como referencia el diagrama de secuencia. Su implementación comienza con la construcción del widget *InitView*.

Archivo `init_view.dart`

```
1 ...
2 class _InitViewState extends State<InitView> {
3   UserBloc userBloc;
4   bool _isSync = false;
5   _isSynchronized() async {
6     await userBloc.currentUser().then( (FirebaseUser user){
7       if (user == null ) return;
8       userBloc.isSynchronized(user.uid).then( (bool isSync){
9         if(!mounted) return;
10        setState((){ _isSync = isSync; });
11      });
12    });
13  }
14  @override
15  Widget build(BuildContext context) {
16    userBloc = BlocProvider.of<UserBloc>(context);
17    _isSynchronized();
18    if(_isSync)
19      return Navigation();
20    else
21      return Synchronize();
22  }
23 }
```

El widget *InitView* verifica si el usuario cuenta con un dispositivo emparejado o no, dependiendo del resultado, la aplicación redireccionará hacia el widget *Navigation* (línea 19) o *Synchronize* (línea 21), antes de llegar a eso, se realizan una serie de operaciones.

Cuando el widget *InitView* carga, se crea una instancia de la clase *UserBloc* (línea 3) que permitirá el acceso a los métodos del patrón de diseño, posteriormente se ejecuta el método asíncrono *_isSynchronized* (línea 17). La primera acción en realizarse, consiste en conocer si el usuario está autenticado, ejecutando el método *currentUser* (línea 6) del objeto *userBloc*, la respuesta de este método es un usuario de tipo *FirebaseUser*. Si el usuario no existe, entonces no se realiza ninguna acción, por otra parte, si el usuario existe, se procede a verificar si cuenta con un componente emparejado, esta acción es realizada utilizando el método *isSynchronized* (línea 8) enviando por parámetro el identificador del usuario (*uid*), cuando del método *isSynchronized* termina, retorna un objeto booleano *isSync* que determina si el usuario cuenta con dispositivo emparejado o no, finalmente dependiendo del valor de *isSync* (línea 18) se mostrará el widget *Navigation* o *Synchronize*.

Cuando se ejecutan los métodos *currentUser* (línea 6) e *isSynchronized* (línea 8), ocurren una serie de procesos antes de retornar una respuesta, estos procesos serán analizados a continuación.

Archivo bloc_user.dart

```
1 ...
2 class UserBloc implements Bloc {
3   final _authRepository = AuthRepository();
4   final _userSettings = SettingsRepository();
5   Future<FirebaseUser> currentUser() {
6     return _authRepository.currentUser();
7   }
8   Future<bool> isSynchronized(String uid) => _userSettings.isSynchronized(uid);
9   void setSynchronize(SyncModel obj) => _userSettings.setSynchronize(obj);
}
```

Para verificar que el usuario este autenticado y sincronizado se agregan 3 métodos nuevos a la clase `UserBloc`: `currentUser` (línea 5), `isSynchronized` (línea 8) y `setSynchronized` (línea 9).

Cuando el método `currentUser` es llamado, se ejecuta otro método con el mismo nombre perteneciente al objeto `_authRepository` (línea 6) de la clase `AuthRepository` (línea 3), su objetivo es verificar el usuario activo.

Cuando el método `isSynchronized` es llamado, se ejecuta el método `isSynchronized` del objeto `_userSettings` (línea 8) enviando como parámetro el identificador del usuario (`uid`), perteneciente a la clase `SettingsRepository`, su objetivo es verificar si el usuario esta emparejado con un dispositivo del componente 2.

Cuando el método `setSynchronize` es llamado, se ejecuta un método con el mismo nombre del objeto `_userSettings` (línea 9) enviando como parámetro `SyncModel`, perteneciente a la clase `SettingsRepository`, su objetivo es establecer el emparejamiento entre un usuario y un dispositivo del componente 2.

Archivo sync_model.dart

```
1 import 'package:flutter/material.dart';
2 class SyncModel {
3   final String uid;
4   final String device;
5   SyncModel({
6     Key key,
7     @required this.uid,
8     @required this.device
9   });
10 }
```

El widget `SyncModel` contiene el modelo de los datos necesarios para realizar la vinculación entre un usuario y un dispositivo, el atributo `uid` (línea 3) almacena el

identificador del usuario, mientras que el atributo *device* (línea 4) almacena el nombre del dispositivo del componente 2.

Archivo `auth_repository.dart`

```
1 ...
2 class AuthRepository {
3   final _firebaseAuthAPI = FirebaseAuthAPI();
4   ...
5   Future<FirebaseUser> currentUser() => _firebaseAuthAPI.currentUser();
6   ...
7 }
```

La clase *AuthRepository* es actualizada para utilizar el método *currentUser* del objeto *_firebaseAuthAPI* (línea 5) perteneciente a la clase *FirebaseAuthAPI* (línea 3), esta acción permitirá a la aplicación comunicarse con los servidores de Firebase y verificar el usuario activo, retornando un objeto de tipo *FirebaseUser*.

Archivo `settings_repository.dart`

```
...
1 class SettingsRepository {
2   final _settingsFirestoreAPI = SettingsFirestoreAPI();
3   Future<bool> isSynchronized(String uid) =>
4   _settingsFirestoreAPI.searchIsSynchronized(uid);
5   void setSynchronize(SyncModel obj) =>
6   _settingsFirestoreAPI.setSynchronize(obj);
7 }
```

Este archivo fue creado para establecer un enlace entre la capa de datos de la aplicación y la base de datos. Se crea una instancia a la clase *SettingsFirestoreAPI* (línea 2) que se comunicará directamente con la base de datos solicitando información, posteriormente cuando se llama el método *isSynchronized* (línea 3) se ejecuta otro método *searchIsSynchronized* perteneciente al objeto *settingsFirestoreAPI* (línea 4).

Cuando el usuario desea realizar un emparejamiento, se utiliza el método *setSynchronize* (línea 5), que forma parte del objeto *_settingsFirestoreAPI* y se envía como parámetro el widget *SyncModel*.

Archivo `settings_firestore_api.dart`

```
1 ...
2 class SettingsFirestoreAPI {
3   ...
4   Future<bool> searchIsSynchronized(String uid) async {
5     bool isSync = false;
6     try{
7       await dbRef.child(SYNCHRONIZED).child(uid).once().then( (DataSnapshot
8 snapshot){
```

```

9         isSync = snapshot.value == null ? false : true;
10    });
11  } catch(e) {
12    print("Error: ${e.toString()}");
13  }
14  return isSync;
15  }
16  ...
17  void setSynchronize(SyncModel obj) async {
18    String setPath = "$SYNCHRONIZED/${obj.uid}";
19    await dbRef.child(setPath).set({
20      "device": obj.device,
21    });
22  }
23  ...
24  }

```

Este archivo es creado para realizar una conexión directa con la base de datos, se agregan los siguientes 2 métodos: *searchIsSynchronized* y *setSynchronized*.

Cuando se ejecuta el método asíncrono *searchIsSynchronized* (línea 4), permitirá verificar si existe un usuario emparejado con un dispositivo del componente 2, esta acción observará una ruta especificada por *dbRef* (línea 7), retornando un objeto de tipo *DataSnapshot*, el objeto *snapshot* es leído y dependiendo de su contenido, un booleano es asignado a la variable *isSync* (línea 9), finalmente si no hay errores el método retornará *isSync* (línea 14).

El método *setSynchronize* (línea 17), permitirá establecer un vínculo entre usuario y dispositivo, cuando el método es ejecutado, debe recibir un objeto de tipo *SyncModel*, del cual se obtendrán los atributos *uid* y *device*, que serán almacenados en la base de datos utilizando el método *set* (línea 19), cuando el proceso ha terminado, el usuario activo será emparejado con el componente 2 enviado.

Archivo *synchronized.dart*

```

1  ...
2  class _SynchronizeState extends State<Synchronize> {
3    ...
4    UserBloc userBloc;
5    @override
6    Widget build(BuildContext context) {
7      userBloc = BlocProvider.of(context);
8
9      return Scaffold(
10         resizeToAvoidBottomPadding: false,
11         body: Container(...),
12         child: Column(
13           children: <Widget>[

```

```

14         ...
15     Expanded(
16         child: StreamBuilder(
17             stream: dbRef.child('info_tekax').limitToLast(10).onValue,
18             builder: (context, snapshot) {
19                 if(snapshot.hasData && !snapshot.hasError){
20                     title = 'Dispositivos';
21                     Map data = snapshot.data.snapshot.value;
22                     List keys = [];
23                     data.forEach( (index, data) => keys.add(index) );
24                     return ListView.builder(
25                         itemCount: data.length,
26                         itemBuilder: (context, index) => SynchronizeItem(title:
27 keys[index], bottom: 10,
28                             onPressed: (){ userBloc.currentUser().then(
29 (FirebaseUser user){
30                                 userBloc.setSynchronize(SyncModel(uid: user.uid,
31 device: keys[index]));
32                             });
33                         })
34                     );
35                 }else{
36                     ...
37                 }
38             }
39         ),
40     ),
41 ],
42 ),
43 ),
);
}
}

```

Este archivo contiene el widget *Synchronized*, se encarga de implementar el UI diseñado para la sección de sincronizaciones, comienza agregando una instancia al patrón de diseño *UserBloc* (línea 4), posteriormente se genera el *StreamBuilder* (línea 16) que estará escuchando lo que ocurre en la sección de la base de datos determinada por *dbRef* (línea 17), finalmente se crea un *ListView.builder* (línea 24) que procesará cada elemento y se lo asignará al widget *SynchronizeItem* (línea 26).

Archivo `synchronize_item.dart`

```

1  ...
2  class _SynchronizeItemState extends State<SynchronizeItem> {
3      @override
4      Widget build(BuildContext context) {
5          return Column(
6              children: <Widget>[

```

```

7     Container(
8         ...
9         child: Row(
10            mainAxisAlignment: MainAxisAlignment.spaceBetween,
11            children: <Widget>[
12                CustomText( title: widget.title, fontColor: Color(0xFF71798C),
13 fontSize: 18 ),
14                SyncButton(title: 'Vincular', color: Color(0xFF74A93A),
15 onPressed: widget.onPressed),
16            ],
17        ),
18    ),
19 ],
20 );
21 }
    }

```

Este archivo contiene el widget *SynchronizerItem*, permite dar formato a cada elemento del componente 2 encontrado en la base de datos y lo muestra dentro aplicación móvil.

Se agrega el widget *CustomText* (línea 12) que muestra el nombre del componente 2 disponible, mientras que el widget *SyncButton* (línea 14), agrega un botón que al ser presionado permitirá vincular el al usuario con el componente 2, ejecutando el método *setSynchronize*, enviando como parámetro el modelo *SyncModel*.

4.4.7 Navegación

La navegación nos permite desplazarnos entre ventanas de la aplicación de una forma rápida y sencilla, como se analizó en el tema 3.4.7 sus secciones principales son 3, cada una de ellas, contienen operaciones y acciones que son ejecutadas de forma independiente.

Para implementar la navegación se utiliza el widget *Navigation*, que contiene los elementos necesarios para hacer el cambio entre diferentes pantallas.

Archivo navigation.dart

```

1 ...
2 class _NavigationState extends State<Navigation> {
3   PageController _pageController;
4   ...
5   @override
6   Widget build(BuildContext context) {
7     return Scaffold(
8       bottomNavigationBar: CurvedNavigationBar(
9         ...
10        items: <Widget>[

```



```

11         Icon(Icons.home, size: 35, color: Colors.white),
12         Icon(Icons.local_florist, size: 35, color: Colors.white),
13         Icon(Icons.settings, size: 35, color: Colors.white),
14     ],
15     ...
16     onTap: (index) {
17         setState(() {
18             _pageController.jumpToPage(index);
19         });
20     },
21 ),
22 body: PageView(
23     controller: _pageController,
24     scrollDirection: Axis.vertical,
25     children: _showWidgets(),
26     onPageChanged: (int index) {
27         setState(() {
28             _pageController.jumpToPage(index);
29         });
30     }
31 ),
32 );
33 }
34 List<Widget> _showWidgets() {
35     return <Widget>[
36         HomeView(),
37         CurrentData(),
38         Settings(),
39     ];
40 }
41 }

```

Se crea una instancia de la clase *PageController* (línea 3), esta clase permite conocer que elemento será mostrado en la aplicación. Posteriormente se utiliza el widget *Scaffold* para construir una estructura predefinida de widgets en la aplicación, dentro de *Scaffold* se utiliza *bottomNavigationBar* (línea 8), esta propiedad permite establecer en la parte inferior una sección que será utilizada para agregar el menú de navegación. Dentro del menú de navegación, los iconos mostrados se encuentran dentro del atributo *items* (línea 10). En la propiedad *body* (línea 22) del *Scaffold*, se agregará el contenido de la ventana en general, dependiendo del valor de *_pageController* (línea 23) se mostrará una u otro ventana. Las ventanas disponibles serán almacenadas como widgets dentro de un objeto tipo *List* llamado *_showWidgets* (líneas 34 - 40).

4.4.8 Pantalla de inicio

La pantalla de inicio, está diseñada para ser la entrada a las diferentes acciones que el usuario tiene permitido realizar, para acceder a ella necesita pasar por el widget

HomeView, en el tema 3.4.8 se muestra el diagrama de secuencia propuesto para su implementación.

Para tener acceso a esta sección el usuario debe estar autenticado y contar con un dispositivo emparejado, una vez que se cuente con ambos procesos, la aplicación redireccionará hacia el widget *Navigation* que por defecto carga otro widget llamado *HomeView*, a partir de este widget se comienza la construcción de la página de inicio.

Archivo `home_view.dart`

```
1 ...
2 class _HomeViewState extends State<HomeView> {
3   bool showHarvest = false;
4   bool showHistory = false;
5   void harvestView() { setState( () => showHarvest = !showHarvest ); }
6   void historyView() { setState( () => showHistory = !showHistory ); }
7   @override
8   Widget build(BuildContext context) {
9     if(showHarvest){
10      return HarvestScreen(harvestView: harvestView);
11    } else if(showHistory){
12      return HistoryScreen(historyView: historyView);
13    } else {
14      return HomeScreen(harvestView: harvestView, historyView: historyView);
15    }
16  }
17 }
```

El widget *HomeView* es el encargado de verificar que pantalla será mostrada en la ventana de inicio, funcionando como un interruptor, cuando activa una ventana, desactiva las demás. La variable booleana *showHarvest* (línea 3) se utiliza para controlar el acceso a nuevos cultivos, la variable *showHistory* (línea 4) controlará el acceso a los históricos, por defecto la ventana de inicio siempre mostrará el widget *HomeScreen* (línea 14).

Archivo `home_screen.dart`

```
1 ...
2 class _HomeScreenState extends State<HomeScreen> {
3   ...
4   @override
5   Widget build(BuildContext context) {
6
7     return Scaffold(
8       body: Container(
9         child: Column(
10          children: <Widget>[
11            Row(
12              children: <Widget>[
13                FadeIn(0.3, CustomGradientButton(
```

```

14         ...,
15         title: 'Historicos'
16     )),
17     FadeIn(0.3, CustomGradientButton(
18         ...,
19         title: 'Fotografías'
20     )),
21     ],
22     ),
23     FadeIn(0.7, CustomGradientButton(
24         ...,
25         title: 'Cultivo'
26     )),
27     FadeIn(1, CustomGradientButton(
28         ...,
29         title: "Construye tu prototipo hidropónico"
30     )),
31     ],
32     ),
33     ),
34     );
35 }
36 }

```

El widget *HomeScreen* contiene la implementación y funcionalidad del UI de la pantalla de inicio, cuenta con 4 botones: Históricos (línea 13), Fotografías (línea 17), Cultivo (línea 25) y Construye tu prototipo hidropónico (línea 27). Cada botón al ser presionado cambia el estado de las variables del widget *HomeView*, mostrando el widget deseado.

4.4.9 Nuevo cultivo

Es la sección diseñada dentro de la aplicación móvil para crear cultivos, por defecto, el componente 2 envía los datos actualizados de cada cultivo al componente 3 y este a su vez, almacena esta información en la sección de cultivo actual, posteriormente la aplicación móvil crea un cultivo nuevo, la sección de configuraciones en la base de datos será actualizada. Cuando el componente 2 detecta que hay un nuevo cultivo, internamente realiza todos los cambios necesarios para almacenar la información de todos los sensores en el nuevo cultivo, esta parte es detallada en el tema 4.2.8.

La implementación de la sección nuevo cultivo, toma como referencia el análisis y diagrama de secuencia propuesto en el tema 3.4.9, es necesario acceder a la pantalla de inicio y posteriormente presionar el botón *Cultivo*, como se documentó en el tema 4.4.8, esta acción, trasladará el widget *HarvestScreen* a la capa superior, tornándolo visible.

Archivo harvest_screen.dart

```
1 ...
2 class HarvestScreen extends StatefulWidget {
3   final Function harvestView;
4   HarvestScreen({ this.harvestView });
5   ...
6 }
7
8 class _HarvestScreenState extends State<HarvestScreen> {
9   @override
10  Widget build(BuildContext context) {
11
12    return Container(...),
13    child: SingleChildScrollView(
14      child: Column(
15        mainAxisAlignment: MainAxisAlignment.spaceBetween,
16        children: <Widget>[
17          Container(...),
18          child: NewHarvestForm(),
19        ],
20        FadeIn(1,
21          ImageInCircle(
22            width: 200, height: 200,
23            bckColor: Colors.transparent,
24            pathImg: 'assets/images/leaf2.png',
25            onPressed: () { widget.harvestView(); }
26          )),
27        ],
28      ),
29    ),
30  );
31
32 }
33 }
```

El widget *HarvestScreen* se encarga de implementar el UI diseñado para la sección de nuevo cultivo, contiene el widget *NewHarvestForm* (línea 18) que realiza el llamado al formulario, por otra parte, el widget *ImageInCircle* (línea 21) muestra una imagen, que al ser presionada, traslada a la capa superior el widget *HomeView*, mostrando la página de inicio.

Archivo new_harvest_form.dart

```
1 ...
2 class _NewHarvestFormState extends State<NewHarvestForm> {
3   UserBloc userBloc;
4   String harvest = '', error = '';
5   DateTime _dateTime = DateTime.now();
6   void _selectDate() { showDatePicker(...); }
```

```

7 void _showColoredToast(String msg, Color color) {
8   Fluttertoast.showToast(...);
9 }
10 Widget _inputHarvest() {...}
11 Widget _inputPicker() {...}
12 Widget _buttonNewHarvest() {
13   return Container(...),
14     child: InkWell(
15       onTap: () async {
16         if(_formKey.currentState.validate()) {
17           userBloc.currentUser().then( (FirebaseUser user){
18             userBloc.getDevice(user.uid).then( (String device){
19               userBloc.setHarvest(
20                 HarvestModel(
21                   device: device,
22                   idFecha: DateFormat('yyyy-MM-dd').format(_dateTime),
23                   nombreCultivo: harvest
24                 )
25               ).then( (bool storeHarvest){
26                 if(storeHarvest){
27                   _showColoredToast('Tu cultivo ha sido activado',
28 Color(0xFF74A93A));
29                 }else{
30                   _showColoredToast('Ocurrio un error, verifica
31 informacion', Color(0xFFFF0000));
32                 }
33               } );
34             } );
35           } );
36         } );
37       }
38     ),
39     ...
40   )
41 );
42 }
43 @override
44 Widget build(BuildContext context) {
45   userBloc = BlocProvider.of<UserBloc>(context);
46   final form = Form(
47     key: _formKey,
48     child: Column(
49       children: <Widget>[
50         ...
51         FadeIn(0.4, _inputHarvest()),
52         FadeIn(0.6, _inputPicker()),
53         FadeIn(0.8, _buttonNewHarvest())
54       ],
55     )
56 );

```

```

57     return form;
    }
}

```

El widget *NewHarvestForm* contiene el formulario para la creación de nuevos cultivos, el método *_selectDate* (línea 6) lanza un componente que agrega un calendario a la pantalla, permitiendo seleccionar la fecha de forma visual. Los widgets *_inputHarvest* (línea 10) e *_inputPicker* (línea 11) agregan el estilo del UI a los campos de entrada del formulario. El widget *_buttonNewHarvest* (línea 12) contiene el UI para la construcción del botón *Crear cultivo*. Finalmente se utiliza el widget *Form* (líneas 46 - 56), que construye todo el formulario.

Cuando el widget *_buttonNewHarvest* (línea 12) es presionado, se ejecuta el método *onTap* (línea 15) esta acción ejecutará de forma asíncrona diferentes procesos y cuando todos hayan finalizado, continuará la ejecución de la aplicación. El primer proceso a ejecutar es la validación de la información del formulario (línea 16), posteriormente verificará el usuario activo (línea 17), después obtiene el nombre del dispositivo (línea 18), finalizando con el almacenamiento del nuevo cultivo (línea 19), el resultado es devuelto en la variable *showHarvest* (línea 25) donde, dependiendo de su valor, lanzará una notificación a la pantalla utilizando el método *_showColoredToast* (línea 27 o 30).

Archivo *harvest_model.dart*

```

1 class HarvestModel {
2   final String device;
3   final String idFecha;
4   final String nombreCultivo;
5
6   HarvestModel({
7     Key key,
8     @required this.device,
9     @required this.idFecha,
10    @required this.nombreCultivo
11  });
12 }

```

El widget *HarvestModel* contiene la estructura necesaria para crear un nuevo cultivo, la variable *device* (línea 2) almacenará el nombre del dispositivo del componente 2 al cual será asignado el nuevo cultivo, *idFecha* (línea 3) contiene la fecha de inicio de cultivo, *nombreCultivo* (línea 4) almacena el nombre designado por el usuario para el cultivo.

Archivo *bloc_user.dart*

```

1 ...
2 class UserBloc implements Bloc {
3   ...
4   Future<String> getDevice(String uid) => _userSettings.getDevice(uid);
5   Future<bool> setHarvest(HarvestModel obj) => _userSettings.setHarvest(obj);

```

```
6 ...
7 }
```

El método *currentUser* y toda su implementación ya fue documentado previamente en el tema 4.4.6 por lo tanto, a partir de ahora solo será documentado código anexo a sus respectivas clases.

El método *getDevice* (línea 4) sirve para obtener el dispositivo emparejado a un usuario, recibe como parámetro el identificador del usuario (*uid*), utiliza una función de flecha que posteriormente ejecuta el método *getDevice* de la instancia *_userSettings* perteneciente a la clase *SettingsRepository*, cuando todo el proceso termina, el método *getDevice* devuelve el nombre del dispositivo emparejado en una variable tipo cadena (String).

El método *setHarvest* (línea 5) sirve para almacenar el nuevo cultivo en el dispositivo adecuado dentro de la base de datos, recibe como parámetro un objeto de tipo *HarvestModel* y utiliza una función de flecha que ejecuta el método *setHarvest* de la clase *SettingsRepository*, cuando todo el proceso termina, retorna una variable booleana indicando si el cultivo ha sido registrado.

Archivo `settings_repository.dart`

```
1 ...
2 class SettingsRepository {
3   ...
4   Future<String> getDevice(String uid) => _settingsFirestoreAPI.getDevice(uid);
5   Future<bool> setHarvest(HarvestModel obj) =>
6   _settingsFirestoreAPI.setHarvest(obj);
7   ...
}
```

La clase *SettingsRepository* se actualiza con los métodos *getDevice* (línea 4) y *setHarvest* (línea 5), retornando el dispositivo emparejado y el resultado de la transacción efectuada respectivamente, ambos elementos ejecutan métodos de la clase *SettingsFirestoreAPI* que se comunicará directamente con la base de datos.

Archivo `settings_firestore_api.dart`

```
1 ...
2 class SettingsFirestoreAPI {
3   ...
4   Future<String> getDevice(String uid) async {
5     String result = '';
6     await dbRef.child(SYNCHRONIZED).child(uid).once().then( (DataSnapshot
7 snapshot) {
8       result = snapshot.value == null ? '' : snapshot.value['device'];
9     });
10    return result;
}
```

```

11 }
12 Future<bool> setHarvest(HarvestModel obj) async {
13     bool storeHarvest = false;
14     try{
15         String setPath = "$DEVICES/${obj.device}/configs/cultivo_actual";
16         await dbRef.child(setPath).set({
17             'idFecha': obj.idFecha,
18             'nombreCultivo': obj.nombreCultivo.toUpperCase()
19         });
20         storeHarvest = true;
21     } catch(e) {
22         print("Error: ${e.toString()}");
23     }
24     return storeHarvest;
25 }
26 ...
}

```

El método *getDevice* (línea 4) es ejecutado como una función asíncrona, recibe como parámetro el identificador del usuario (*uid*), posteriormente con el valor recibido se crea una referencia (línea 6), donde son anexados los valores y la sección específica de la base de datos a modificar, cuando la consulta es ejecutada, la base de datos devuelve un objeto de tipo *DataSnapshot*, la respuesta debe ser procesada para obtener únicamente la información deseada y se asigna a la variable *result* (línea 7), finalmente esta variable es retornada como respuesta del método.

El método *setHarvest* (línea 12), se ejecuta como un método asíncrono, recibe como parámetro un objeto de tipo *HarvestModel*, la información del objeto es utilizada para generar la referencia del documento buscado y se asigna a la variable *setPath* (línea 15), que será usada para actualizar la base de datos, si la variable *showHarvest* (línea 20) cambia su estado a verdadero significa que la transacción ha sido correcta, finalmente esta variable es retornada terminando la funcionalidad del método.

4.4.10 Información de los sensores

Es la sección diseñada para que el usuario pueda consultar la información que es recolectada en tiempo real por el componente 2, visualizándola de forma gráfica y entendible por el usuario. El desarrollo de esta sección se lleva a cabo tomando como referencia el análisis y diagrama de secuencia documentado en el tema 3.4.10.

Para tener acceso a este apartado, se debe utilizar el menú de navegación, presionando el icono del centro, ver Figura 3-24, esta acción realiza los ajustes necesarios para mostrar en primer plano el widget *CurrentData*.

Archivo current_data.dart

```
1 ...
2 class _CurrentDataState extends State<CurrentData> with
3 SingleTickerProviderStateMixin {
4   ...
5   _loadUserInfo() async {
6     await userBloc.currentUser().then( (FirebaseUser user){
7       if (user == null ) return;
8       userBloc.getDevice(user.uid).then( (String device){
9         if(!mounted) return;
10        setState(){
11          userBloc.getSettings(device).then( (Map data){
12            if( data == null ) return;
13            if(data['cultivo_actual'] != null) _cultivoActual =
14 data['cultivo_actual']['nombreCultivo'];
15            _sizeContainer = data['altoContenedor'].toDouble();
16          });
17
18          userBloc.getDataSensors(device).then( (Map data){
19            if(data != null){
20              _temperature = data['temperatura'].toDouble();
21              _humidity = data['humedad'].toDouble();
22              _distance = data['distancia'].toDouble();
23              _pH = (data['pH'].toDouble() >= 14) ? 14 :
24 data['pH'].toDouble();
25            }
26          });
27        });
28      });
29    });
30  }
31
32 Widget _widgetLvlWater() {...}
33 Widget _widgetTemperature() {...}
34 Widget _widgetHumidity() {...}
35
36 @override
37 Widget build(BuildContext context) {
38   userBloc = BlocProvider.of(context);
39   _loadUserInfo();
40   return Container(
41     child: SingleChildScrollView(
42       child: Column(
43         mainAxisAlignment: MainAxisAlignment.spaceBetween,
44         children: <Widget>[
45           CustomNavbar(),
46           Padding(
47             padding: EdgeInsets.all(10),
48             child: FadeIn(0.3, CurrentHarvest( cultivoActual:
49 (_cultivoActual != null) ? _cultivoActual : ' ' )),
```

```

50         ),
51         Padding(
52             padding: EdgeInsets.symmetric(horizontal: 10, vertical: 0),
53             child: Row(
54                 mainAxisAlignment: MainAxisAlignment.spaceBetween,
55                 children: <Widget>[
56                     FadeIn(0.5, _widgetTemperature()),
57                     FadeIn(0.5, _widgetHumidity()),
58                 ],
59             ),
60         ),
61         Padding(
62             padding: EdgeInsets.all(10),
63             child: FadeIn(0.7, _widgetLvlWater()),
64         ),
65         Padding(
66             padding: EdgeInsets.all(10),
67             child: FadeIn(1, CurrentPH(pH: _pH,)) )
68         ),
69         SizedBox(height: 15)
70     ],
71     ),
    );
}
}

```

El widget *CurrentData* se encarga de implementar el UI diseñado para la sección *Información de los sensores*, mostrando los valores obtenidos de cada sensor del componente 2 en tiempo real, sin embargo, antes de mostrar el resultado, primero deben ser ejecutados ciertos procesos.

Cuando el widget es iniciado, se crea una instancia del patrón de diseño *userBloc* (línea 38), posteriormente se ejecuta la función *_loadUserInfo* (línea 39), que se encarga de ejecutar los diferentes procesos necesarios para obtención de información, misma que será mostrada como resultado en cada widget de la pantalla final.

El primer valor necesario, es el usuario activo de la aplicación, para ello se utiliza el método *currentUser* (línea 6), este método fue documentado en el tema 4.4.6.

El segundo valor necesario es el dispositivo del componente 2 con el que esta emparejado el usuario, para lo cual, se utiliza el método *getDevice* (línea 8) enviando por parámetro el identificador del usuario, este método ha sido documentado en el tema 4.4.10.

El tercer valor necesario son las configuraciones asignadas del dispositivo, se obtienen ejecutando el método *getSettings* (línea 11) del patrón de diseño, enviando por parámetro el dispositivo emparejado.

El último elemento a consultar es la información de los sensores, el método que se utiliza es *getDataSensors* (línea 18) enviando por parámetro el nombre del dispositivo y retornando la información de los sensores.

Cuando toda la información necesaria ha sido recolectada, se mostrará un widget personalizado para cada elemento: *_widgetTemperature* (línea 56) muestra la temperatura, *_widgetHumidity* (línea 57) muestra la humedad, *_widgetLvlWater* (línea 63) muestra el nivel de agua del contenedor y *CurrentPH* (línea 67) muestra el valor del pH, dentro de la aplicación móvil.

Archivo `bloc_user.dart`

```
1 ...
2 class UserBloc implements Bloc {
3   ...
4   Future<Map> getSettings(String device) => _userSettings.getSettings(device);
5   Future<Map> getDataSensors(String device) =>
6   _userSettings.getDataSensors(device);
7   ...
}
```

Se actualiza la clase *UserBloc* anexando 2 métodos nuevos:

- *getSettings* (línea 4) se encarga de consultar las configuraciones de un dispositivo determinado, recibe por parámetro el nombre del dispositivo (*device*) y ejecuta otro método de la clase *SettingsRepository* con el mismo nombre, devuelve un objeto de tipo *Map*, con las configuraciones del dispositivo requerido.
- *getDataSensors* (línea 5) se encarga de consultar la información obtenida de cada sensor, recibe por parámetro el nombre del dispositivo y ejecuta un método de la clase *SettingsRepository* con el mismo nombre, devuelve un objeto de tipo *Map*, con la información de los valores actualizados de cada sensor.

Archivo `settings_repository.dart`

```
...
1 class SettingsRepository {
2   final _settingsFirestoreAPI = SettingsFirestoreAPI();
3   ...
4   Future<Map> getSettings(String device) =>
5   _settingsFirestoreAPI.getSettings(device);
6   Future<Map> getDataSensors(String device) =>
7   _settingsFirestoreAPI.getDataSensors(device);
}
```

Se actualiza la clase *SettingsRepository* agregando 2 métodos nuevos: *getSettings* (línea 4) y *getDataSensors* (línea 6), ambos reciben como parámetro el nombre del dispositivo (*device*) y ejecutan métodos de la clase *SettingsFirestoreAPI* encargada de consultar de forma directa a la base de datos.

Archivo `settings_firestore_api.dart`

```
1 ...
2 class SettingsFirestoreAPI {
3   ...
4   Future<Map> getSettings(String device) async {
5     String setPath = "$DEVICES/$device/configs";
6     Map data;
7     try{
8       await dbRef.child(setPath).once().then( (DataSnapshot snapshot){
9         data = snapshot.value;
10      });
11    } catch(e) {
12      print("Error: ${e.toString()}");
13    }
14    return data;
15  }
16
17  Future<Map> getDataSensors(String device) async{
18    String setPath = "$DEVICES/$device/sensores";
19    Map data;
20    try{
21      await dbRef.child(setPath).once().then( (DataSnapshot snapshot){
22        data = snapshot.value;
23      });
24    } catch(e) {
25      print("Error: ${e.toString()}");
26    }
27    return data;
28  }
29
30 }
```

La clase *SettingsFirestoreAPI* es actualizada con 2 métodos nuevos:

- *getSettings* (línea 4) es un método asíncrono que recibe como parámetro el dispositivo emparejado con el usuario activo, comienza estableciendo el documento que se desea consultar de la base de datos con la variable *setPath* (línea 5), posteriormente se realiza la consulta utilizando la referencia *dbRef* y la ruta *setPath* esperando un resultado (línea 8), la *API* de Firebase se encarga de procesar la solicitud y retorna una respuesta dentro de un objeto de tipo *DataSnapshot* que se asigna a la variable *data* (línea 9), el método finaliza cuando se regresa dicha variable (línea 14).

- `getDataSensors` es un método asíncrono que recibe como parámetro el dispositivo emparejado, la variable `setPath` (línea 18) se utiliza para asignar el documento de la base de datos que será consultado, la ejecución del método este método generará la consulta a la base de datos, el resultado es retornado como un objeto de tipo `DataSnapshot` y es asignado a la variable `data` (línea 22), el método concluye cuando esta es retornada (línea 27).

4.4.11 Configuraciones

Para llevar a cabo la implementación de la sección de configuraciones se utiliza el diagrama de secuencia propuesto en el tema 3.4.11, el widget utilizado como punto de partida es `Settings`, a continuación, se analizarán cada uno de los componentes que integran su implementación.

Archivo `settings.dart`

```

1  ...
2  class Settings extends StatelessWidget {
3    UserBloc userBloc;
4    @override
5    Widget build(BuildContext context) {
6      ...
7      return Scaffold(
8        body: Stack(
9          alignment: Alignment.center,
10         children: <Widget>[
11           GradientBack(height: null, initColor: Color(0xFFFF5F6FB), endColor:
12             Color(0xFFFDFF)),
13           SingleChildScrollView(
14             child: Column(
15               mainAxisAlignment: MainAxisAlignment.spaceEvenly,
16               children: <Widget>[
17                 FadeIn(0.3, UserInfo()),
18                 FadeIn(0.7, Operations()),
19                 FadeIn(1, ImageInCircle(
20                   ...,
21                   onPressed: () { userBloc.signOut(); } ) ),
22                 ...
23               ],
24             ),
25           ),
26         ],
27       );
28     }
29   }

```

El widget *Settings* contiene la implementación del UI diseñado para la sección de configuraciones, reserva una sección dentro de la pantalla para cada widget que conformará el módulo de configuraciones: la información del usuario con el widget *UserInfo* (línea 16), las acciones del usuario con el widget *Operations* (línea 17), el cierre de sesión con el widget *ImageInCircle* (línea 18).

Archivo user_info.dart

```

1  ...
2  class _UserInfoState extends State<UserInfo> {
3  ...
4  _loadUserInfo() async {
5    await userBloc.currentUser().then( (FirebaseUser user){
6      if(!mounted) return;
7      setState( (){
8        if( user != null ){
9          _photoUrl = (user.photoUrl != null ) ? user.photoUrl : 'https:...';
10         _displayName = (user.displayName != null) ? user.displayName :
11         'Correo electrónico';
12         _email = user.email;
13       }
14     } );
15   });
16 }
17 Widget _displayAvatar() {...}
18 @override
19 Widget build(BuildContext context) {
20   userBloc = BlocProvider.of(context);
21   _loadUserInfo();
22   return Stack(
23     alignment: Alignment.center,
24     children: <Widget>[
25       Container(
26         ...
27         child: Column(
28           children: <Widget>[
29             CustomText(title: _displayName,
30               fontColor: Colors.white, fontSize: 24,
31               fontWeight: FontWeight.w400, letterSpacing: 1.0, ),
32             CustomText(title: _email, fontColor: Colors.white,
33               fontSize: 18, fontWeight: FontWeight.w300, )
34           ],
35         ),
36       ),
37       _displayAvatar(),
38     ],
39   );
40 }
41 }

```

El widget *UserInfo* se encarga de mostrar la información del usuario que esta autenticado en la aplicación, en el tema 4.4.4 se documentan las 2 alternativas que existen para tener acceso a la aplicación: 1) Método tradicional mediante usuario y contraseña previamente registrados en la aplicación. 2) Usando las credenciales de Google. La sección de configuraciones independientemente del método utilizado al autenticarse, debe encargarse de mostrar la información del usuario.

Cuando se accede al widget *UserInfo*, se ejecuta el método *_loadUserInfo* (línea 21), posteriormente se obtiene la información del usuario activo utilizando el método *currentUser* (línea 5) perteneciente a la clase *UserBloc*, finalmente cuando el usuario es retornado se obtienen los valores: *_photoUrl* (línea 9), *_displayName* (línea 10) y *_email* (línea 12), que serán mostrados en pantalla utilizando el widget *_displayAvatar* (línea 37).

Archivo operations.dart

```

1 ...
2 class _OperationsState extends State<Operations> {
3   ...
4   _loadUserInfo() async {
5     await userBloc.currentUser().then( (FirebaseUser user){
6       if (user == null ) return;
7       userBloc.getDevice(user.uid).then( (String device){
8         if(!mounted) return;
9         setState( ){
10          _uid = user.uid;
11          _device = device;
12          _setPath = 'info_tekax/$_device/configs';
13          userBloc.getSettings(device).then( (Map data){
14            if(data == null) return;
15            _settings = data;
16            _bombaManual = data['_bombaManual'];
17            _sizeContainer = data['_altoContenedor'].toDouble();
18          });
19        });
20      });
21    });
22  }
23
24  @override
25  Widget build(BuildContext context) {
26    ...
27    _loadUserInfo();
28    return Container(
29      ...
30      child: Column(
31        children: <Widget>[
32          CustomText(title: 'Acciones', fontColor: Color(0xFF626A7E),
33          fontSize: 20),
34          Row(

```

```

35     mainAxisAlignment: MainAxisAlignment.spaceBetween,
36     children: <Widget>[
37         CustomText(title: (_device == '') ? '' : _device, ... ),
38         SyncButton(
39             title: 'Desvincular', color: Color(0xFF9C100),
40             onPressed: (){
41                 userBloc.removeSynchronize(_uid);
42             }
43         ),
44     ],
45 ),
46 Row(
47     mainAxisAlignment: MainAxisAlignment.spaceBetween,
48     children: <Widget>[
49         CustomText(
50             title: (_bombaManual) ? 'Bomba manual (Encendida)' : 'Bomba
51 manual (con timer)', ...
52         ),
53         Switch(value: _bombaManual,
54             onChanged: (newVal){ setState(){
55                 data[_bombaManual] = newVal;
56                 data[_estadoBomba] = (newVal) ? 0 : 1;
57                 userBloc.setSettings(data, _device);
58             });
59         });
60     ],
61 ),
62 Column(
63     crossAxisAlignment: CrossAxisAlignment.start,
64     children: <Widget>[
65         CustomText(title: 'Altura del contenedor $_sizeContainer cm
66 (mover slider)', ... ),
67         SliderTheme(
68             data: ...,
69             child: Slider(
70                 ...
71                 label: '$_sizeContainer',
72                 onChanged: (value) {
73                     setState() {
74                         if(_settings == null) return;
75                         if(_device == null) return;
76                         _settings[_altoContenedor] = value.toDouble();
77                         userBloc.setSettings(_settings, _device);
78                     });
79                 },
80             ),
81         ),
82     ],
83 )
84 ],

```



```

85         )
        );
    }
}

```

El widget *Operations* se encarga de mostrar en el apartado de las configuraciones, las operaciones que el usuario tiene permitido realizar. Antes de poder ejecutar cualquier acción, es necesario conocer el usuario activo utilizando el método *currentUser* (línea 5), el dispositivo vinculado con el método *getDevice* (línea 7) y las configuraciones del usuario con el método *getSettings* (línea 13), todo esto encapsulado dentro del método asíncrono *_loadUserInfo* (líneas 4 - 22) que es llamado antes de la construcción del UI (línea 27).

Cuando el método *_loadUserInfo* termina de solicitar toda la información necesaria a la base de datos, el UI es mostrado, permitiendo al usuario cambiar el estado de las siguientes configuraciones:

1. *Desvincular usuario*. Como su nombre lo indica desvincula el usuario asociado al componente 2 previamente seleccionado, para llevar a cabo esta tarea solo es necesario presionar el botón *Desvincular*, esta acción ejecuta el método *removeSynchronize* (línea 41) enviando por parámetro el identificador del usuario *_uid*.
2. *Bomba manual*. Permite al usuario alternar entre activar la bomba utilizando el temporizador (opción por defecto) o una activación manual. Con el temporizador la bomba es encendida 20 minutos cada hora de 8 am a 7 pm, mientras que, activando la bomba de forma manual, está siempre permanecerá encendida. Para alternar entre los diferentes estados de la bomba, solo es necesario presionar el botón interruptor, que ejecutará el método *onChanged* (línea 54) permitiendo comenzar el proceso de actualización del valor de la bomba en la base de datos utilizando el método *setSettings* (línea 57) y enviando por parámetro los nuevos valores (*data*) y el dispositivo asociado (*_device*).
3. *Altura del contenedor*. Permite establecer una altura para el contenedor de la solución nutritiva utilizado por el usuario, esta acción es realizada agregando un control deslizante, que al cambiar su estado se ejecuta el método *onChanged* (línea 72), obtiene los nuevos valores de la altura del contenedor y los envía a la base de datos utilizando el método *setSettings* (línea 77).

4.4.12 Históricos

En esta sección el usuario tiene la posibilidad de visualizar los históricos de sus cosechas, mostrando una gráfica lineal, el sistema permite visualizar el progreso de los elementos que son monitorizados dentro de los cultivos (pH, porcentaje de la solución nutritiva dentro del contenedor, temperatura y humedad). La Implementación de los históricos toma como referencia el análisis y diagrama de secuencia propuesto en el tema 3.4.12. Para acceder a esta sección, el usuario debe utilizar el menú de navegación e

ingresar al widget *HomeScreen*, posteriormente presionar la opción *Históricos*, que permitirá visualizar el widget *HistoryScreen* mostrando la interface inicial.

Archivo `history_screen.dart`

```
1 ...
2 class _HistoryScreenState extends State<HistoryScreen> {
3   UserBloc userBloc;
4   final dbRef = FirebaseDatabase.instance.reference();
5   ...
6   _loadUserInfo() async {
7     await userBloc.currentUser().then( (FirebaseUser user){
8       userBloc.getDevice(user.uid).then( (String device){
9         setState(){
10          userBloc.getSettings(device).then( (Map data){
11            if( data == null ) return;
12            _device = device;
13            _setPath = 'info_tekax/$_device';
14            _sizeContainer = data['altoContenedor'].toDouble();
15          });
16        });
17      });
18    });
19  }
20  @override
21  Widget build(BuildContext context) {
22    userBloc = BlocProvider.of(context);
23    _loadUserInfo();
24
25    return Scaffold(
26      ...
27      body: Container(
28        ...
29        child: Column(
30          children: <Widget>[
31            ...
32            Expanded(
33              child: StreamBuilder(
34                stream: (_device == '') ? null :
35 dbRef.child(_setPath).limitToLast(10).onValue,
36                builder: (context, snapshot) {
37                  if(snapshot.hasData && !snapshot.hasError){
38                    Map data = snapshot.data.snapshot.value;
39                    List keys = [];
40                    data.forEach( (index, data) => keys.add(index) );
41
42                    return ListView.builder(
43                      itemCount: data.length,
44                      itemBuilder: (context, index) {
45                        return HistoryItem(data: data[keys[index]],
46                          index: keys[index],
```

```

47         sizeContainer: _sizeContainer,
48         bottom: 10);
49     }
50     );
51     }
52     }
53     ),
54     ),
55     ],
56     ),
57     ),
58     );
59 }

```

El widget *HistoryScreen* contiene la implementación del UI diseñado para la sección de los históricos, una vez que ha sido cargado se crea una instancia al método *BlocProvider* que permite establecer la relación entre el código y el patrón de diseño (línea 22), posteriormente se realizan las peticiones necesarias a los diferentes servicios para conocer el usuario activo (línea 7), dispositivo emparejado (línea 8) y configuraciones establecidas (líneas 10 – 15).

Una vez que la aplicación obtiene los elementos necesarios, consultará en la base de datos por las cosechas pertenecientes al usuario activo y el dispositivo emparejado. Utilizando un *StreamBuilder* (líneas 33 – 52) todas las respuestas serán procesadas y segmentadas de tal forma que es posible enviar la información de los históricos (*data*), el nombre del cultivo (*index*) y el tamaño del contenedor (*sizeContainer*) hacia el widget *HistoryItem*.

Archivo `history_item.dart`

```

1 ...
2 class _HistoryItemState extends State<HistoryItem> {
3   @override
4   Widget build(BuildContext context) {
5     return Column(
6       children: <Widget>[
7         Container(
8           ...
9         child: Theme(
10          ...
11         child: ExpansionTile(
12          ...
13         children: <Widget>[
14         Container(
15          ...
16         child: SafeArea(
17         child: PageView(

```

```

18         children: <Widget>[
19           Container(
20             child: ListView(
21               children: <Widget>[
22                 Padding(
23                   padding: EdgeInsets.symmetric(horizontal:
24 0, vertical: 0),
25                   child: LineGraph(data: widget.data, index:
26 widget.index, sizeContainer: widget.sizeContainer,)
27                 ),
28               ],
29             ),
30           ),
31         ],
32       ),
33     ),
34   ),
35 ],
36 ),
37 ),
38 ),
39   SizedBox(height: widget.bottom,)
40 ],
41 );
}
}

```

El widget *HistoryItem*, se encarga de presentar dentro de la aplicación móvil, cuáles son los cultivos que el usuario ha realizado, cada elemento contiene una estructura base definida por el UI diseñado, y un área específica (*safeArea*) delimitando el espacio que puede ocupar la gráfica en la aplicación móvil (línea 16), finalmente dentro de esta sección determinada, se realiza la petición hacia el widget *LineGraph* (línea 25) enviando como parámetros toda la información de los históricos (*data*), el nombre del cultivo (*index*) y el alto del contenedor(*sizeContainer*).

Archivo `line_graph.dart`

```

1 ...
2 class _LineGraphState extends State<LineGraph> {
3   ...
4   _prepareData(){
5     if (widget.data[logs_sensores] == null) return;
6     // Obtengo el listado de meses a introducir en la grafica
7     var logDays = widget.data[logs_sensores].keys.toList()..sort();
8     ..
9     for (var day in logDays) {
10      var daySplit = day.split('-');
11      String month = _nameMonth(int.parse(daySplit[1]));
12      if(!months.contains(month)) months.add(month);
13    }

```

```

14     int sizeLogs = 0;
15     double hum = 0, temp = 0, dist = 0, ph = 0, posX = 0;
16     sizeLogs = widget.data[logs_sensores][day].length;
17     widget.data[logs_sensores][day].forEach( (k, log){
18         dist += log[distancia];
19         hum += log[humedad];
20         ph += log[pH];
21         temp += log[temperatura];
22     } );
23
24     // Calculamos el promedio por dia
25     dist = ( (widget.sizeContainer - (dist/sizeLogs) ) * 100 /
26 widget.sizeContainer ) * 4 / 100;
27     hum = (hum/sizeLogs)*4/100;
28     ph = (ph/sizeLogs)*4/100;
29     temp = (temp/sizeLogs)*4/100;
30
31     // Calculamos la posicion del punto dentro del grafico
32     posX = noDay*14/logDays.length;
33
34     // Agregamos el punto a una lista de puntos
35     waterSpots.add(FlSpot(posX, dist));
36     humiditySpots.add(FlSpot(posX, hum));
37     phSpots.add(FlSpot(posX, ph));
38     temperatureSpots.add(FlSpot(posX, temp));
39
40     noDay++;
41 }
42 ...
43 }
44 ...
45 Widget _widgetDisplay(){
46     if( _dataLog){
47         return AspectRatio(
48             aspectRatio: 1.23,
49             child: Container(
50                 decoration: BoxDecoration(...),
51                 child: Stack(
52                     children: <Widget>[
53                         Column(
54                             children: <Widget>[
55                                 ...
56                                 Expanded(
57                                     child: Padding(
58                                         padding: EdgeInsets.only(right: 16, left: 6),
59                                         child: LineChart(
60                                             _dataChart()
61                                         ),
62                                     )
63                                 )

```

```

64         ],
65     )
66     ],
67     ),
68     ),
69     );
70 }
71 }
72 ...
73 @override
74 Widget build(BuildContext context) {
75     _prepareData();
76     return _widgetDisplay();
77 }
}

```

El widget *LineGraph* es el encargado de mostrar el gráfico, contiene múltiples métodos propios de la librería que realizan diferentes operaciones, como son: establecer el color de las líneas en el gráfico, dar formato a los títulos, establecer el formato de la gráfica en general. Sin embargo, hay 2 métodos que fueron desarrollados desde cero y que permiten generar el gráfico de forma dinámica:

1. El método *_prepareData*, se utiliza para procesar los datos de los históricos almacenados en la base de datos, por defecto, el componente 2 envía la información y la almacena cada hora, sin embargo, si existe un corte de energía o alguna falla en el internet, esta información no puede ser almacenada, por tal motivo, se almacena una cantidad de N lecturas diarias para cada sensor. El funcionamiento de este método es el siguiente:
 - i. Se verifica que exista información dentro del arreglo *data['logs_sensores']* que contiene todos los históricos de un cultivo en específico (línea 5).
 - ii. Los históricos son separados por día y se almacenan en un arreglo llamado *logDays* (línea 7).
 - iii. Se obtiene la cantidad de meses que ha durado el cultivo, y se almacena en una variable de tipo *List* de nombre *months* (línea 12).
 - iv. De todos los históricos de un día, se obtiene una media del valor de cada sensor y también se calcula la posición del punto en la gráfica lineal con la siguiente fórmula (líneas 17 – 29):
$$spot = (\sum_{i=m}^n sensorValue / sizeLogs) * 4 / 100$$
 - v. Cada punto obtenido con la fórmula del paso 1.iv es agregado a una variable de tipo *List* (líneas 35 - 38), la conjunción de todos los puntos forma una línea.
2. *_widgetDisplay*, se encarga de crear el gráfico, concentrando los widgets que fueron previamente establecidos como: el color de las líneas en el gráfico, los títulos, fondo del gráfico, y los datos que fueron procesados con el método *_prepareData*.

CAPÍTULO 5

Resultados

Capítulo 5 Resultados

En el presente capítulo se muestran los resultados obtenidos, posteriores a la implementación de cada uno de los elementos que componen el presente proyecto.

5.1 Automatización hidropónica

Para llevar a cabo la automatización hidropónica se integran los siguientes elementos: *NodeMCU*, *pH-4502C*, módulo Relay 5v, *AM2302*, *AT24C32*, en una placa Fenólica y se soldán por la parte inferior utilizando un caudín, pasta para soldar y soldadura, evitando así un corto circuito o falso contacto, ver Figura 5-1.

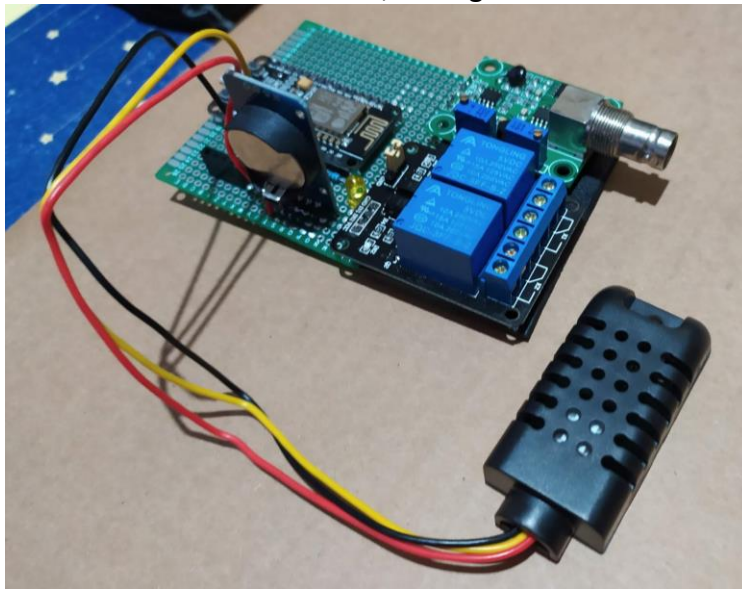


Figura 5-1 Circuito electrónico desarrollado.

Dentro de la automatización existen 2 elementos que pueden conectarse y desconectarse según las necesidades del usuario, ver Figura 5-2:

1. El sensor *pH-4502C* cuenta con una conexión tipo *BNC* que permite integrar la sonda de pH al sensor análogo para su fácil acoplamiento.
2. El sensor de proximidad *HC-SR04*, para su integración al circuito electrónico se soldán, un conjunto de pines hembra a la placa Fenólica, mismos que servirán de entrada para que el sensor de proximidad pueda ser acoplado o removido del circuito.



Figura 5-2 Elementos que pueden conectarse y desconectarse del componente 2.

Finalmente se diseña una caja de MDF que protegerá a todo el circuito desarrollado, integrando cada elemento previamente mencionado en un solo componente, ver Figura 5-3.

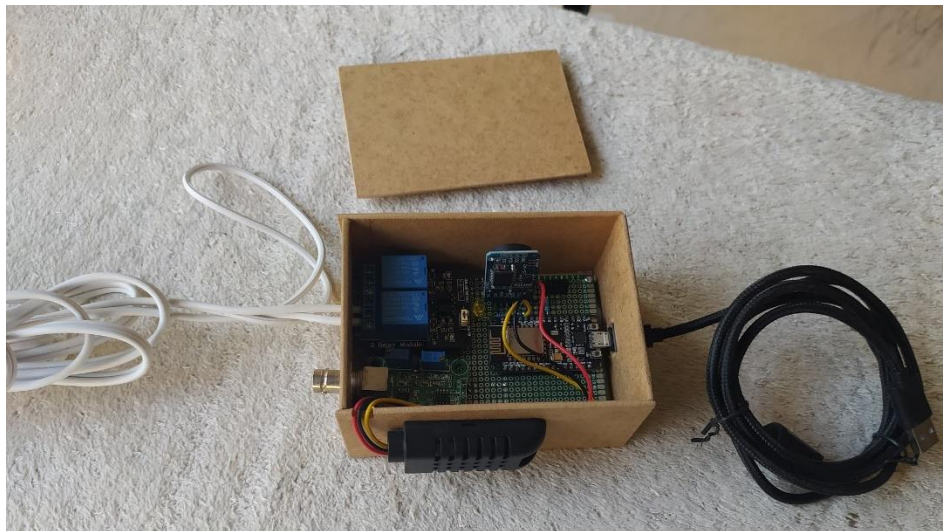


Figura 5-3 Caja MDF creada para proteger el circuito electrónico.

5.2 Aplicación móvil

La aplicación permite al usuario administrar y controlar las cosechas del cultivo hidropónico, realizando acciones como: acceso restringido a la información mediante un sistema de autenticación, encender o apagar la bomba de agua, actualizar el tamaño del contenedor de la solución nutritiva, desvincular el dispositivo, acceder en tiempo real a la información del entorno que rodea al prototipo hidropónico, crear nuevos cultivos, visualización de históricos y cerrar sesión.

Una de sus principales virtudes, es que la información puede ser consultada en todo momento, independientemente de la ubicación del usuario y el prototipo, también fue implementada una interface que cuida la experiencia de usuario, acorde a los estándares establecidos en el tema 2.4.5 y 2.4.6, esta aplicación móvil puede ser utilizada en un teléfono celular o una tableta y se adapta a las diferentes resoluciones y posiciones de la pantalla.

5.2.1 Autenticación

Por defecto las operaciones dentro de la aplicación, están restringidas para usuarios autenticados, con el objetivo de brindar una mayor facilidad de acceso, se implementaron 2 mecanismos que permiten el ingreso a la misma, ver Figura 5-4:

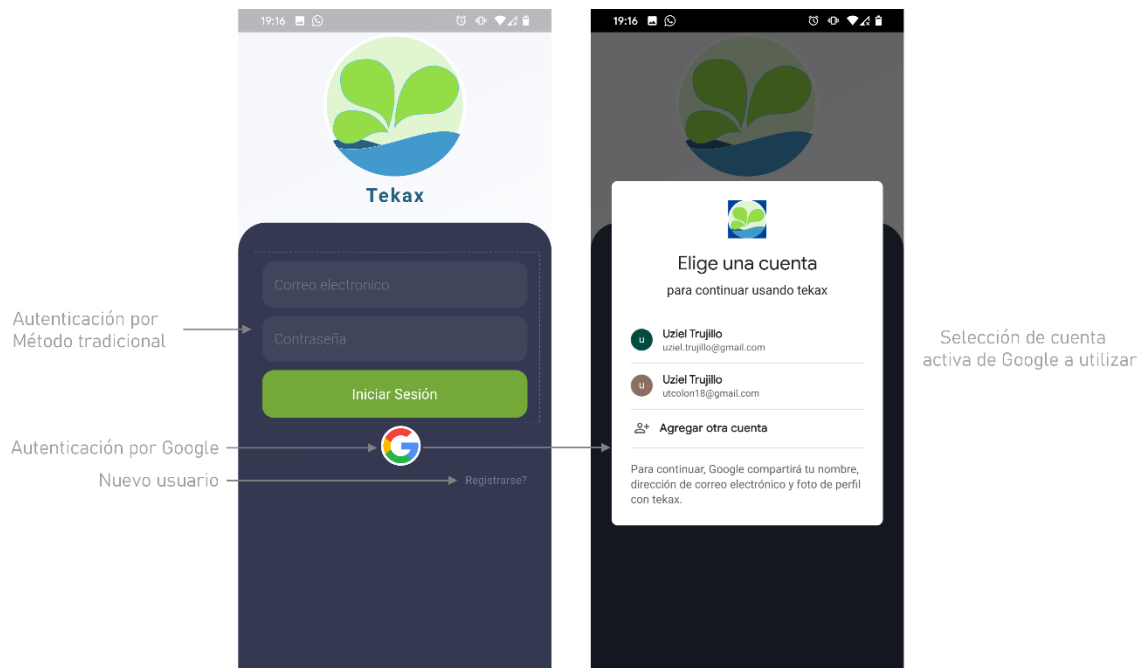


Figura 5-4 Autenticación en la aplicación móvil.

1. **Método tradicional.** El usuario ingresa sus credenciales dentro del formulario, posteriormente presiona el botón *Iniciar Sesión*, el sistema valida la información introducida y permite o deniega el acceso dependiendo del resultado.
2. **Google.** Para realizar la autenticación por Google, el usuario debe presionar el logotipo de la empresa, posteriormente se abrirá una nueva ventana donde le permitirá al usuario seleccionar la cuenta activa, o agregar una nueva cuenta, una vez realizado este procedimiento, el servidor de Google enviará una respuesta a la aplicación móvil y se permitirá o denegará el acceso.

Si el usuario no cuenta con los accesos del punto 1 y 2, entonces se brinda la opción de realizar un nuevo registro, solo necesita presionar la palabra *Registrarse* y el sistema redireccionará hacia el formulario de *Nuevo usuario*.

5.2.2 Nuevo usuario

Creada para registrar nuevos usuarios que se autenticarán en la aplicación móvil utilizando el método tradicional, consta de un formulario que solicita únicamente un correo electrónico y contraseña. Cuando el usuario ha terminado de ingresar la información solicitada, deberá presionar el botón *Registrar*, permitiendo a la aplicación crear un nuevo usuario, en caso de que ocurra algún problema, el sistema notificará los posibles errores, ver Figura 5-5.

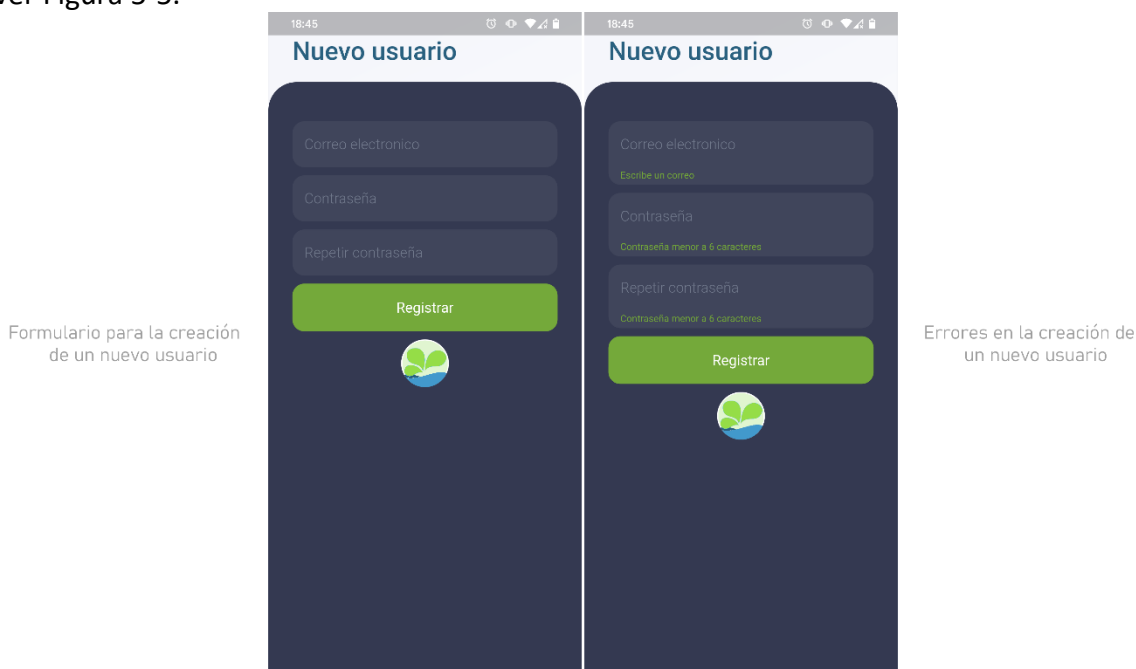


Figura 5-5 Creación de nuevo usuario en la aplicación móvil.

Una vez que el usuario se ha registrado, necesitará utilizar estas credenciales en el formulario de inicio de sesión para poder autenticarse y tener acceso a todas las funcionalidades restringidas de la aplicación móvil.

5.2.3 Emparejar dispositivo

La aplicación permite emparejar un usuario con un dispositivo electrónico (componente 2), esta acción posibilita que el usuario pueda consultar, verificar y administrar la información recolectada por dicho dispositivo desde la aplicación móvil, por lo tanto, realizar este paso es de vital importancia antes de continuar con cualquier otra acción. De forma automática cuando el usuario ha iniciado sesión adecuadamente, la aplicación verificará el identificador del usuario y revisará si existe un dispositivo

emparejado a él, si no existe ningún dispositivo asociado, entonces se mostrará la ventana de vinculación, mostrando todos los dispositivos disponibles, ver Figura 5-6.



Figura 5-6 Emparejamiento de usuario activo con el componente 2 desde la aplicación móvil.

Para emparejar un dispositivo, el usuario solo deberá presionar la opción *Vincular* y de forma automática el sistema realizará este proceso, posteriormente será redireccionado hacia la pantalla de inicio.

5.2.4 Pantalla de inicio

Esta ventana es mostrada siempre y cuando el usuario haya ingresado a la aplicación con las credenciales correctas y también posea un dispositivo emparejado. Muestra la interface principal de la aplicación, por defecto, se presenta la pantalla inicial, que consta de 4 sub secciones principales que permiten el acceso a: *Históricos*, *Fotografías*, *Cultivo*, *Construye tu prototipo hidropónico*. En la parte inferior se muestra el menú de navegación, que está dividido en 3 iconos que representan las secciones: *Pantalla de inicio*, *información de sensores*, *configuraciones*, ver Figura 5-7.



Figura 5-7 Pantalla de inicio de la aplicación móvil.

Para acceder a cualquier sección del menú de navegación o sub sección de la pantalla inicial, solo es necesario presionar la opción deseada.

5.2.5 Nuevo cultivo

La sub sección *nuevo cultivo*, permitirá al usuario terminar una cosecha y comenzar una nueva. Para acceder a ella, es necesario utilizar el menú de navegación y presionar la opción *pantalla de inicio*, posteriormente seleccionar la opción *Cultivo*.

Cuando el usuario accede de forma adecuada a la sección, la aplicación muestra el formulario base, solicitando el nombre del cultivo y la fecha de inicio, para evitar errores en la selección de fecha, se ha implementado un calendario. Cuando el usuario considera que la información es correcta, deberá presionar el botón *Crear cultivo*, el sistema verificará si los campos del formulario no están vacíos, de ser este el caso, la aplicación indicará en que campo se requiere información, si la información es correcta y los datos fueron almacenados en el componente 3 de forma adecuada, se notificará al usuario, ver Figura 5-8.

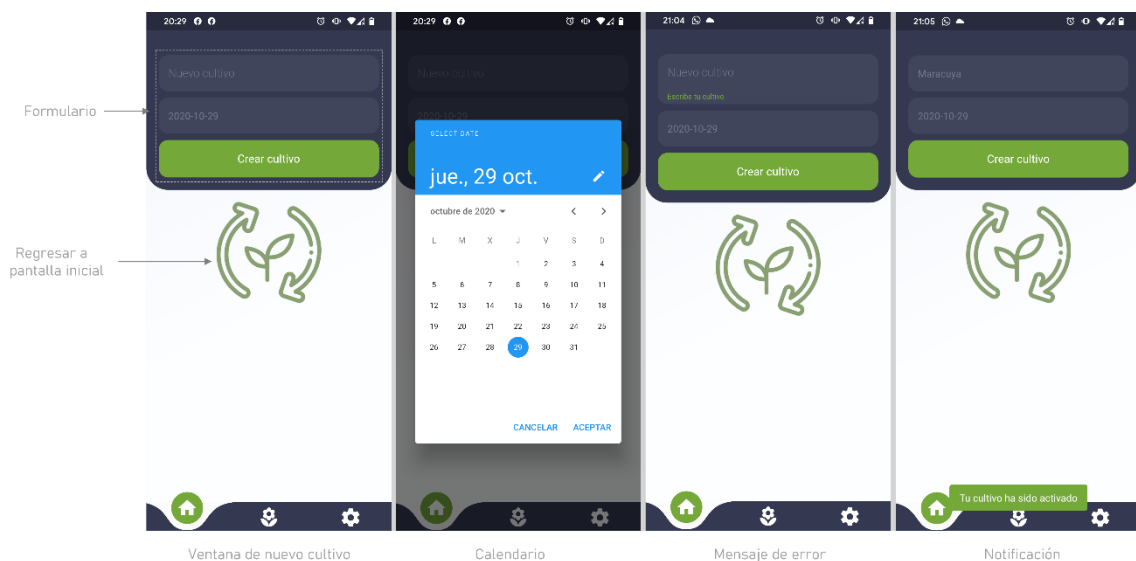


Figura 5-8 Creación de nuevo cultivo desde aplicación móvil.

5.2.6 Información de sensores

La aplicación móvil permite tener acceso a la información generada por el circuito electrónico (componente 2) y almacenada en la base de datos (componente 3), estas lecturas son en tiempo real, lo que permite visualizar cualquier cambio de información en el pH, temperatura, humedad y nivel de la solución nutritiva dentro del contenedor en todo momento, sin necesidad de recargar la aplicación ni cambiar de una ventana a otra, brindando al usuario información actualizada siempre que sea requerida, ver Figura 5-9.

Para acceder a esta sección, solo es necesario presionar el icono de *información de sensores* ubicado en el menú de navegación.



Figura 5-9 Información del estado actual de los sensores desde la aplicación móvil.

5.2.7 Configuraciones

Dentro de la ventana de configuraciones el usuario tiene la posibilidad de ajustar valores para mejorar la información que obtiene o para establecer parámetros que operen directamente en el circuito electrónico (componente 2), como lo son: desvincular el usuario con el componente 2, encender o apagar la bomba de agua de forma permanente, modificar la altura del contenedor de la solución nutritiva, cerrar la sesión.

Cuando se accede a la sección de configuraciones, el sistema detecta el usuario activo, independientemente del método de autenticación utilizado. Si el usuario cuenta con una imagen de perfil, esta será utilizada, de lo contrario establecerá, la inicial de su primer nombre, ver Figura 5-10.

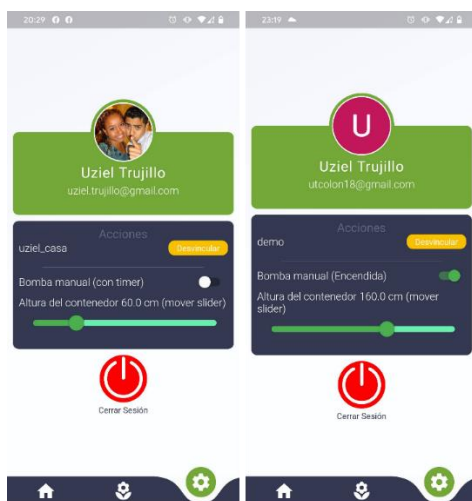


Figura 5-10 Configuraciones del prototipo hidropónico desde la aplicación móvil.

En las configuraciones existen diferentes controles, por ejemplo: 1) Para desvincular el usuario activo al componente electrónico o cerrar la sesión, solo es necesario presionar el botón para activar su funcionalidad, 2) Para encender o apagar la bomba de agua, se necesita presionar el interruptor, este alternara entre las 2 únicas funciones permitidas (encender o apagar), 3) Para modificar la altura del contenedor, solo se debe deslizar el punto sobre la barra de desplazamiento, hasta ajustar la medida deseada.

5.2.8 Históricos

Una parte importante para la administración de los cultivos hidropónicos es conocer el estado de las cosechas y su comportamiento con el paso del tiempo, por esta razón, el manejo de históricos es de suma importancia. La aplicación permite concentrar y mostrar de forma visual el progreso de los diferentes parámetros (pH, temperatura, humedad y nivel de agua) que son constantemente monitorizados en un solo gráfico.

Para acceder a los históricos es necesario navegar hacia la *pantalla de inicio*, posteriormente presionar el botón *Históricos*, finalmente, la aplicación mostrará las cosechas que han sido registradas mediante el circuito electrónico (componente 2) al cual el usuario está vinculado. Para visualizar el gráfico que contiene la información concentrada, es necesario presionar los botones ubicados en la parte derecha de las cosechas, esta acción permitirá mostrar u ocultar los elementos según lo requiera el usuario, ver Figura 5-11.

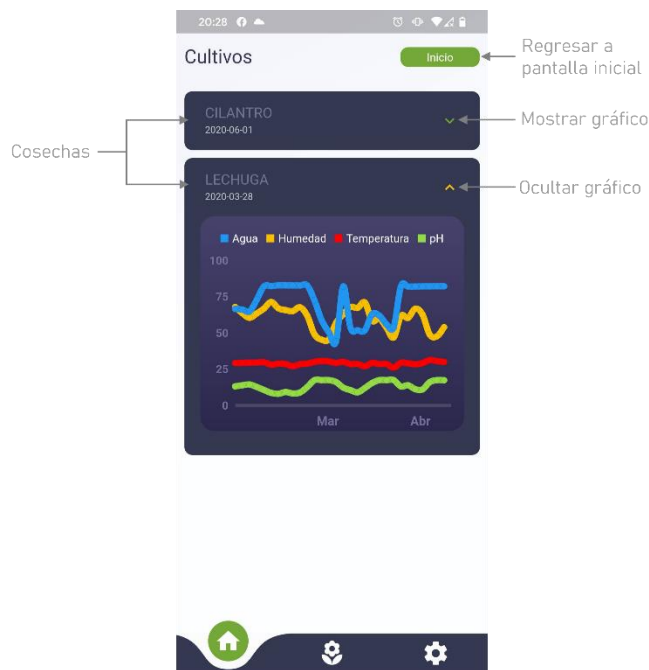


Figura 5-11 Históricos en la aplicación móvil.

5.3 Prototipo hidropónico

El prototipo hidropónico es el elemento que muestra los resultados finales del presente proyecto, para llevar a cabo el control de las cosechas, se utilizan elementos de automatización mediante el componente 2 y administración de los cultivos a través la aplicación móvil.

Se realizan 2 pruebas con diferentes tipos de hortalizas para comprobar la automatización, el funcionamiento de la aplicación móvil y cómo se comportan en conjunto con el prototipo hidropónico.

Las condiciones en las que fue llevado a cabo el experimento en ambas pruebas, son las siguientes:

- El prototipo hidropónico fue colocado en un espacio abierto, sin protección ni recubrimiento de ningún tipo.
- En ningún momento se agrega fertilizante o plaguicidas a los cultivos.
- Las horas aproximadas en las que el sol golpea directamente sobre las plantas es de 7 horas aproximadamente.

5.3.1 Lechuga orejona (Experimento 1)

Para la primera prueba se cosecha lechuga orejona, en la tabla 5-1, se muestran los resultados utilizando las herramientas y el prototipo hidropónico propuesto, regulando y monitorizando los valores de pH, temperatura, humedad y nivel de la solución nutritiva en el contenedor.

Día	Temperatura	Humedad	Solución nutritiva	pH	Tamaño
1-2	32°C	62%	-	-	1 cm
3-7	30°C	73%	-	-	3 cm
8-14	33°C	68%	75%	6	8 cm
15-24	29°C	61%	70%	6.1	13 cm
25-31	28°C	63%	60%	5.7	18 cm
32-38	31°C	65%	65%	6.2	25 cm
39-45	32°C	58%	61%	6	37 cm

Tabla 5-1 Evolución lechuga orejona utilizando el prototipo hidropónico y la aplicación móvil.

A continuación, se muestra la evolución de las plantas a través de los días, el que suscribe se ofreció a degustar la cosecha probando diferentes ensaladas, el resultado es comestible, la lechuga cuenta con un buen color, sin presencia de hongos o algún tipo de animal, su textura es suave y su sabor es agradable, ver Figura 5-12.



Figura 5-12 Resultados de lechuga orejona en el prototipo hidropónico.

5.3.2 Cilantro y maracuyá (Experimento 2)

Para la segunda prueba, son utilizadas 2 tipos de hortalizas: cilantro y maracuyá. En la tabla 5-2, se muestran los resultados de la cosecha.

Día	Temperatura	Humedad	Solución nutritiva	pH	Cilantro crecimiento	Maracuyá crecimiento
1-2	28°C	70%			1 cm	1 cm
3-7	30°C	65%			3 cm	5 cm
8-14	30°C	69%			5cm	10 cm
15-24	29°C	55%	80%	6	8 cm	15 cm
25-31	25°C	83%	80%	6	12 cm	22 cm
32-38	28°C	68%	75%	6.3	14 cm	35 cm
39-45	30°C	70%	70%	6.7	17 cm	45 cm
46 – 52	29°C	73%	80%	6	21 cm	60
52 - 60	27°C	67%	75%	6.5	24 cm	70 cm

Tabla 5-2 Evolución del cilantro y maracuyá en el prototipo hidropónico, utilizando la aplicación móvil.

Las hortalizas son insertadas dentro del prototipo hidropónico de forma no secuencial, ver Figura 5-13

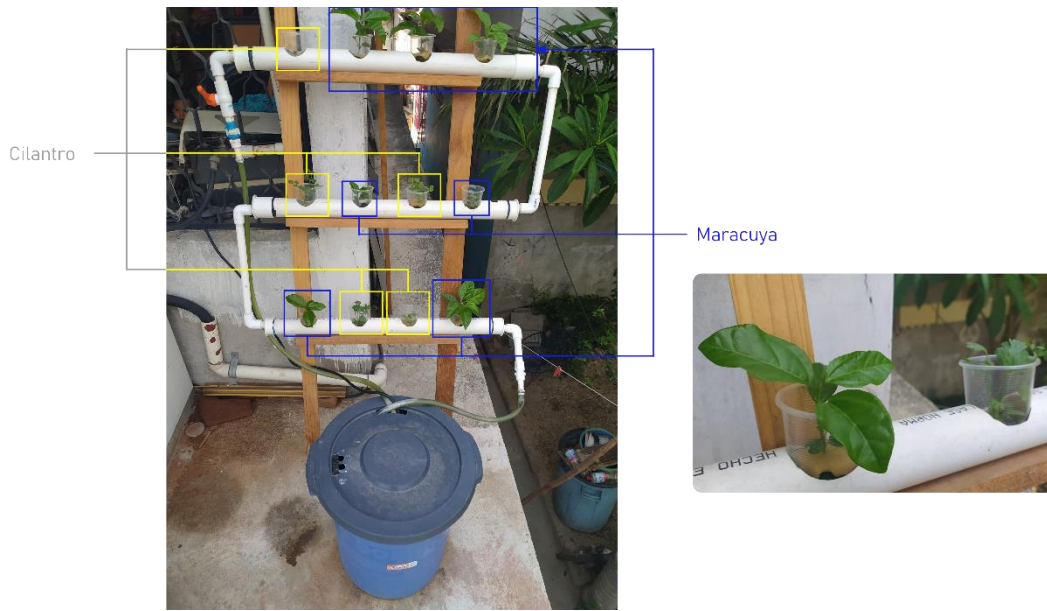


Figura 5-13 Disposición de cilantro y maracuyá en el prototipo hidropónico.

Hay crecimiento en ambos casos, sin embargo, no todos los resultados son los esperados.

Resultados de cosechar cilantro utilizando el prototipo hidropónico y la aplicación móvil propuesta, ver Figura 5-14:



Cilantro en buen estado

Cilantro en mal estado, hojas amarillas

Comparativa de tamaño respecto a la planta de maracuyá

Figura 5-14 Resultados de cosechar cilantro en el prototipo hidropónico.

- Su crecimiento conforme avanza el tiempo, es lento.
- En algunos casos las hojas se comienzan a tornar amarillas de las puntas y se marchitan.
- No existe evidencia de bichos.

Resultados de cosechar Maracuyá utilizando el prototipo hidropónico y la aplicación móvil propuesta, ver Figura 5-15.



Figura 5-15 Resultados de la cosecha de maracuyá en el prototipo hidropónico.

- Su crecimiento es acelerado.
- Las hojas tienen buen tamaño y color.
- No existe evidencia de hongos o bichos.

CAPÍTULO 6

Conclusiones

Capítulo 6 Conclusiones

El objetivo del presente proyecto de tesis consiste en brindar una alternativa que permita llevar el control y la administración en forma adecuada de los cultivos hidropónicos. Con las herramientas propuestas, el tiempo dedicado a la administración se ve reducido de forma considerable, ya que el usuario en todo momento, puede conocer el estado de la cosecha sin estar físicamente presente, esto es, debido a que toda la información recolectada por el componente 2 (automatización hidropónica) utilizando diferentes sensores, se envía hacia el componente 3 (base de datos) que almacena todos estos datos en la nube, estas acciones, permiten que la información pueda ser accedida desde cualquier parte del mundo, siempre y cuando se cuente con una conexión a internet y se posean las credenciales necesarias.

El control de los cultivos hidropónicos también es más cómodo de llevar desde el componente 4 (aplicación móvil), ya que permite realizar los siguientes ajustes: encender o apagar la bomba agua, creación de nuevos cultivos, establecimiento de la altura del contenedor de la solución nutritiva, emparejar o desvincular un usuario con el componente 2. Finalmente es posible acceder a la información almacenada en el componente 3 y así poder visualizar el estado actual de las cosechas o incluso generar los procesos necesarios para verificar su comportamiento de forma gráfica, mediante sus históricos.

Los materiales utilizados para la construcción del prototipo hidropónico y los sensores seleccionados son de bajo costo, esperando que cualquier persona pueda llevar a cabo este proyecto sin muchas dificultades.

La hidroponía es una técnica que puede ser utilizada para uso comercial o doméstico, la herramienta propuesta en el presente trabajo de tesis puede ser utilizada en ambos casos según lo requiera el usuario.

Como las evidencias documentadas en el capítulo de resultados, se muestra la factibilidad de realización para un sistema hidropónico, utilizando las herramientas propuestas que facilitan su administración y control, brindando al usuario la posibilidad de cultivar sus propias hortalizas de forma automatizada.

6.1 Trabajo futuro

A pesar de que el prototipo fue llevado a cabo con éxito, aún existen muchos elementos que pueden ser mejorados, a continuación, se enlistan las posibles mejoras de cada componente.

6.1.1 Prototipo hidropónico (Componente 1)

En la región, existe una gran cantidad de personas que aún se dedican a la agricultura tradicional, parte de la preocupación diaria es alimentar al ganado. Utilizando otra técnica hidropónica y realizando ajustes al prototipo, será posible cultivar forraje verde que pueda ser utilizado como alimento para la crianza de animales.

Algunos tubos de *PVC* requieren perforaciones que no vienen de fábrica, al realizar estos ajustes se debe tener cuidado, ya que hacerlo de forma inadecuada, pueden ocasionar fugas de agua. Es posible crear tubos y reducciones a medida utilizando una impresora *3D* para que el ensamble y desensamble sea más sencillo y así evitar que la construcción del prototipo hidropónico sea complicada.

6.1.2 Automatización hidropónica (Componente 2)

A continuación, se describen posibles mejoras con el fin de incrementar el objetivo, alcance y por supuesto, ventajas de la idea base aquí propuesta:

- Utilizar celdas fotovoltaicas que provean energía suficiente para encender la bomba de agua y el circuito electrónico automatizando aún más el consumo de los recursos.
- Crear un circuito impreso para que los componentes electrónicos y sensores utilizados en el presente proyecto, puedan ser integrados con mayor facilidad, consumiendo así, un menor tiempo de ensamble.
- Construir una estructura de tipo invernadero que permita proteger el prototipo hidropónico, agregando los sensores y actuadores necesarios, se proveerán las herramientas que controlen completamente el entorno de las cosechas, mejorando así la calidad de las mismas.

6.1.3 Base de datos (Componente 3)

La información de la base de datos se almacena de forma estructurada, sin embargo, si en un futuro se desea integrar la aplicación junto a un servicio externo, lo mejor sería crear una *API* que permita la lectura, escritura, actualización y eliminación de información en la base de datos de forma más eficiente, posibilitando su integración con otros desarrollos y reutilizando la información previamente generada.

6.1.4 Aplicación móvil (Componente 4)

Dentro de la aplicación, en la pantalla de inicio, quedaron 2 sub secciones pendientes, que no afectan su funcionamiento, pero si ofrecen un valor agregado, estas secciones son:

- *Módulo de fotografías*: Pensada para que el usuario registre el crecimiento de las plantas mediante la toma constante de fotografías, lo que le permitirá contar con una bitácora visual del progreso de sus cosechas.
- *Construye tu prototipo hidropónico*: La sección está pensada para aquellos usuarios que desconozcan la técnica hidropónica y deseen comenzar a cultivar sus hortalizas, la idea es integrar información que el usuario pueda consultar referente a diferentes temas hidropónicos, como pueden ser: la construcción del prototipo, el tiempo de germinación de las plantas, el rango de pH recomendado para cada cultivo, la duración de las cosechas, entre otras cosas.

La estructura de la aplicación móvil está pensada para integrar nuevos módulos de forma sencilla, por esta razón, es posible anexar nuevas funcionalidades, según sea requerido.

De momento la aplicación funciona sólo con el sistema operativo Android 5.0 o superior, migrar la aplicación al sistema operativo iOS es una opción viable, que permitirá captar una mayor cantidad de usuarios activos.

6.2 Participaciones y aportaciones

Durante el transcurso de los 2 años que duro el proyecto, se realizaron las siguientes participaciones y aportaciones en diferentes ámbitos:

1. Publicación de cartel en Expo-Proyecta (Tecnológico Nacional de México, campus Acapulco).
2. Artículo y ponencia en el Congreso Internacional de Investigación de Academia Journals Puebla 2019.
3. Artículo publicado en la 29a. Reunión Internacional de comunicaciones, computación, electrónica, automatización, robótica y exposición industrial. ROC&C 2020 IEEE, Acapulco de Juárez.
4. Publicación de cartel en el 2do. Congreso Internacional Multidisciplinario (Tecnológico Nacional de México, campus Xalapa).
5. Impartición de taller: “Introducción al desarrollo de aplicaciones móviles multiplataforma con Flutter”, dentro del 3er Congreso Internacional de Innovación, Tecnología y Sustentabilidad 2019 (Tecnológico Nacional de México, campus Acapulco).
6. Estancia realizada en la Secundaria General No. 4 (Acapulco de Juárez), donde fueron entregados:
 - a. Prototipo funcional
 - b. Aplicación móvil
 - c. Manual de usuario
 - d. Manual técnico

Fuentes de Información

- Abdullah, H. M., & Zeki, A. M. (2014). Frontend and backend web technologies in social networking sites: Facebook as an example. *Proceedings - 3rd International Conference on Advanced Computer Science Applications and Technologies, ACSAT 2014*, 85–89. <https://doi.org/10.1109/ACSAT.2014.22>
- Abraham Silberschatz, Henry F. Korth, S. S. (2006). *Fundamentos de base de datos* (5ta ed.). Mc Graw Hill.
- Atmel. (2003). *AT24C32 Datasheet*. 1–19.
- AUTOGROW-LAB. (2018). *Automation Solutions For Quality Crops | Grow anywhere | Autogrow*. <https://autogrow.com/>
- Balducci, F., Impedovo, D., & Pirlo, G. (2018). Machine Learning Applications on Agricultural Datasets for Smart Farm Enhancement. *Machines*, 6(3), 38. <https://doi.org/10.3390/machines6030038>
- Beltrano, J., & Gimenez, D. O. (2015). Cultivo en hidroponía. *Libros de Càtedra*, 1(978-950-34-1258-9), 181. http://sedici.unlp.edu.ar/bitstream/handle/10915/46752/Documento_completo.pdf?sequence=1
- Blancarte, O. (2016). *Introducción a los patrones de diseño: un enfoque practico* (Primera Ed). Amazon.
- Cambra, C., Sendra, S., Lloret, J., & Lacuesta, R. (2018). Smart system for bicarbonate control in irrigation for hydroponic precision farming. *Sensors (Switzerland)*, 18(5), 1–16. <https://doi.org/10.3390/s18051333>
- Chiara, D., Herrera, L., Vargas, P., & Mario Chauca, A. (2016). CULTIVO HIDROPÓNICO DE ESPINACA MEDIANTE TÉCNICA NFT E INVERNADERO PARA EL CONTROL DE VARIABLES AMBIENTALES Hydroponics culture of spinach through NFT technique and greenhouse to control environmental variables. *Nº*, 12(12), 49–60.
- Crisnapati, P. N., Wardana, I. N. K., Aryanto, I. K. A. A., & Hermawan, A. (2017). Hommons: Hydroponic management and monitoring system for an IOT based NFT farm using web technology. *2017 5th International Conference on Cyber and IT Service Management, CITSM 2017, August*. <https://doi.org/10.1109/CITSM.2017.8089268>
- DIY. (2017). *pH-4502C Datasheet*. 6, 0–5. <https://www.botshop.co.za/how-to-use-a-ph-probe-and-sensor/>

- Dunn, B. (2015). *Hydroponics*. July.
- El-Kassas, W. S., Abdullah, B. A., Yousef, A. H., & Wahba, A. M. (2017). Taxonomy of Cross-Platform Mobile Applications Development Approaches. *Ain Shams Engineering Journal*, 8(2), 163–190. <https://doi.org/10.1016/j.asej.2015.08.004>
- FARM.ONE. (2018). *Farm.One: Technology-Powered Vertical Farms for Specialty Produce*. <https://farm.one/>
- Ferentinos, K. P., & Albright, L. D. (2003). Fault detection and diagnosis in deep-trough hydroponics using intelligent computational tools. *Biosystems Engineering*, 84(1), 13–30. [https://doi.org/10.1016/S1537-5110\(02\)00232-5](https://doi.org/10.1016/S1537-5110(02)00232-5)
- Francis, F., Vishnu, P. L., Jha, M., & Rajaram, B. (2018). IOT-based automated aeroponics system. *Lecture Notes in Electrical Engineering*, 492, 337–345. https://doi.org/10.1007/978-981-10-8575-8_32
- FULLBLOOM HYDROPONICS. (2019). *Hydroponic Systems | Kits, Supplies & Stealth Growing Equipment*. [https://www.fullbloomhydroponics.net./](https://www.fullbloomhydroponics.net/)
- García Ulloa, M., León, C., Hernández F, & Chavéz R. (2005). Evaluación de un sistema experimental de acuaponia. *Avances En Investigación Agropecuaria*, 9(001), 1–5. http://bvirtual.ucol.mx/descargables/678_evaluacion_de_un_sistema_experimental.pdf
- Google. (2018). *Flutter - Beautiful native apps in record time*. <https://flutter.dev/>
- Guadarrama, R. B., Melitón, E., Rodríguez, R., Rodrigo, E., Cerón, V., Jorge, A., & Blancas, M. (2016). *Medidor De Ph , Electro-Conductividad Y. 120*, 736–758.
- Halili, F., & Ramadani, E. (2018). Web Services: A Comparison of Soap and Rest Services. *Modern Applied Science*, 12(3), 175. <https://doi.org/10.5539/mas.v12n3p175>
- INEGI. (2017). *Agricultura*. <https://www.inegi.org.mx/temas/agricultura/>
- INEGI. (2019). *Mapas. Climatológicos*. Instituto Nacional de Estadística y Geografía. INEGI. <https://www.inegi.org.mx/temas/mapas/climatologia/>
- IPN, C. (2018). *CeProBi - IPN*. www.Ceprobi.Ipn.Mx. <https://www.ceprobi.ipn.mx/conócenos/instalaciones.html#labs>
- Izquierdo, J., & Marulanda, C. (FAO). (2003). *La Huerta Hidropónica Popular - Manual técnico*.

- Jensen, C. T. (Ed.). (2015). *APIs for dummies* (Issue c). John Wiley & Sons, Inc.
- Liu, T. (2016). AM2302 Datasheet. *Adfruit*, 1–5. <https://cdn-shop.adafruit.com/datasheets/Digital+humidity+and+temperature+sensor+AM2302.pdf>
- Monk, S. (2012). *Programming Arduino Getting Started with Sketches by Simon Monk* (p. 365). Mc Graw Hill.
- Monk, S. (2017). *Hacking Electronics Learning Electronics with Arduino and Raspberry Pi* (2nd Editio). Mc Graw Hill.
- Morgan, E. J. (2014). HC-SR04 Datasheet. *Evaluation Tecnichal of Sensor*, Nov. 16 2014.
- NACIONES UNIDAS. (2012). *Tierra, recursos naturales y conflictos: Del infortunio a la oportunidad. Una Alianza UE-ONU en acción*. Naciones Unidas. <http://www.un.org/es/land-natural-resources-conflict/>
- NACIONES UNIDAS. (2015). *Población | Naciones Unidas*. <http://www.un.org/es/sections/issues-depth/population/index.html>
- Nishimura, T., Okuyama, Y., & Satoh, A. (2016). High-accuracy and low-cost sensor module for hydroponic culture system. *2016 IEEE 5th Global Conference on Consumer Electronics, GCCE 2016*, 5–8. <https://doi.org/10.1109/GCCE.2016.7800514>
- NIWA-ONE. (2018). *getniwa | Niwa: Get growing fruits and & vegetables at home!* <https://getniwa.com/>
- NodeMCU Datasheet. (2015). NodeMCU Datasheet. *Espressif Systems Datasheet*, 1–31. https://www.adafruit.com/images/product-files/2471/OA-ESP8266__Datasheet__EN_v4.3.pdf
- Nursyahid, A., Wibisono, M. R., Wardihani, E. D., Helmy, & Setyawan, T. A. (2018). Plant age identification system of outdoor hydroponic cultivation based on digital image processing. *Proceedings - 2017 4th International Conference on Information Technology, Computer, and Electrical Engineering, ICITACEE 2017, 2018-Janua*, 213–218. <https://doi.org/10.1109/ICITACEE.2017.8257705>
- Obay, M. A. M., Altrafi, G., & Ismail, M. O. (2014). Relational vs . NoSQL Databases : A Survey. *International Journal of Computer and Information Technology*, 03(03), 2279–2764.
- Palande, V., Zaheer, A., & George, K. (2018). Fully Automated Hydroponic System for Indoor Plant Growth. *Procedia Computer Science*, 129, 482–488.

<https://doi.org/10.1016/j.procs.2018.03.028>

POSTSCAPES. (2019). *Smart Greenhouse | 2019 Guide to best Sensors and Remote Automated Monitoring Software*. <https://www.postscapes.com/smart-greenhouses/>

Que, P., Guo, X., & Zhu, M. (2017). A Comprehensive Comparison between Hybrid and Native App Paradigms. *Proceedings - 2016 8th International Conference on Computational Intelligence and Communication Networks, CICN 2016*, 611–614. <https://doi.org/10.1109/CICN.2016.125>

Ruengittinun, S., Phongsamsuan, S., & Sureeratanakorn, P. (2017). Applied internet of thing for smart hydroponic farming ecosystem (HFE). *Ubi-Media 2017 - Proceedings of the 10th International Conference on Ubi-Media Computing and Workshops with the 4th International Workshop on Advanced E-Learning and the 1st International Workshop on Multimedia and IoT: Networks, Systems and Applications*. <https://doi.org/10.1109/UMEDIA.2017.8074148>

Saaid, M. F., Yahya, N. A. M., Noor, M. Z. H., & Ali, M. S. A. M. (2013). A development of an automatic microcontroller system for Deep Water Culture (DWC). *Proceedings - 2013 IEEE 9th International Colloquium on Signal Processing and Its Applications, CSPA 2013, March*, 328–332. <https://doi.org/10.1109/CSPA.2013.6530066>

Sayara, T., Amarnah, B., Saleh, T., Aslan, K., Abuhanish, R., & Jawabreh, A. (2016). Hydroponic and Aquaponic Systems for Sustainable Agriculture and Environment. *International Journal of Plant Science and Ecology*, 2(3), 23–29. <http://www.aiscience.org/journal/ijpsehttp://creativecommons.org/licenses/by/4.0/>

Sisyanto, R. E. N., Suhardi, & Kurniawan, N. B. (2018). Hydroponic smart farming using cyber physical social system with telegram messenger. *2017 International Conference on Information Technology Systems and Innovation, ICITSI 2017 - Proceedings, 2018-Janua*, 239–245. <https://doi.org/10.1109/ICITSI.2017.8267950>

Solís, P., & Benza, G. (2013). Clases sociales, pobreza y desigualdad durante los años de alternancia presidencial. *El Colegio de México*, 22. <http://www.inegi.org.mx/eventos/2013/Desigualdades/doc/P-PatricioSolis.pdf>

Sommerville, I. (2011). *Ingeniería de software* (9th ed.). Pearson Educación.

Suri, S. (2018). *Architect your Flutter project using BLOC pattern - FlutterPub - Medium*. <https://medium.com/flutterpub/architecting-your-flutter-project-bd04e144a8f1>

Systems, E. (2020). *ESP8266EX*.

Tihomirovs, J., & Grabis, J. (2017). Comparison of SOAP and REST Based Web Services

Using Software Evaluation Metrics. *Information Technology and Management Science*, 19(1), 92–97. <https://doi.org/10.1515/itms-2016-0017>

W. Lidwell, K. H. and J. B. (2007). W. Lidwell, K. Holden and J. Butler: Universal Principles of Design. Educational Technology Research and Development (Vol. 55). <http://doi.org/10.1007/s11423-007-9036-7>. In *Educational Technology Research and Development* (Vol. 55). <https://doi.org/10.1007/s11423-007-9036-7>

Zagal, M., Martínez, S., Salgado, S., Escalera, F., Peña, B., Carrillo, F., Zagal-Tranquilin, M., Martínez-González, & Carrillo-Díaz, F. (2016). Hydroponics maize green forage production with watering every 24 hours. *Abanico Veterinario*, 6(1), 29–34. http://www.scielo.org.mx/scielo.php?script=sci_arttext&pid=S2448-61322016000100029&lng=es&nrm=iso&tlng=es

Zobel, R. W., Del Tredici, P., & Torrey, J. G. (2008). Method for Growing Plants Aeroponically. *Plant Physiology*, 57(3), 344–346. <https://doi.org/10.1104/pp.57.3.344>