

TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE APIZACO

Ingeniería en Tecnologías de la información y Comunicaciones

Tech Up: Innovación y Progreso

Nombre de la empresa:

Grupo Transnacional Catapulta

Autor:

Yair Atenco Hernández

Asesor interno:

Docente Felipe Pascual Rosario Aguirre

Asesor externo:

Ingeniero José Luis Hernández Miranda

Tlaxcala, Apizaco a 10 de enero de 2024

Agradecimientos

Dando por concluido mi proyecto, quisiera comenzar expresando mi más sincero agradecimiento a Dios, primeramente, ya que, gracias a él, que por su amor a mí no solo ha preservado mi vida durante mi etapa universitaria, sino que, también es quien me ha dado la gracia de poder aprender y desarrollar mis conocimientos, a pesar de lo arduo que fue este trayecto para mí. Sin él, esto no me habría sido posible.

Gracias a mi familia, mi esposa y mis hijos, que son quienes me han inspirado a seguir adelante con mayor fuerza y a terminar esta etapa, siempre acompañándome en cada paso con amor y confiando en mí, enseñándome el valor de la familia y el pilar sumamente importante que son para mí.

Gracias a mis padres, que por su trabajo me brindaron la oportunidad de poder estudiar la carrera que elegí, y por su cariño, por el cual me educaron e inculcaron la importancia, el valor y el privilegio de culminar mis estudios.

Gracias a mis docentes, por su admirable trabajo que realizaron al transmitirme sus conocimientos con paciencia, por medio de los cuales pude llegar hasta aquí, mostrándome también que siempre puedo seguir preparándome.

Y, por último, gracias a la empresa con quienes trabaje en conjunto con este proyecto, que me dieron la confianza y la oportunidad de poder llevar a cabo la creación de una aplicación móvil así cubriendo una necesidad importante que se tenía en la empresa.

Resumen

Se creo una IP completamente nueva, basada en las solicitudes y necesidades de la empresa para aumentar la eficiencia dentro de la empresa en la que se llevaron a cabo minuciosas investigaciones para el uso de las tecnologías que se adapten mejor al proyecto, hicimos comparativas con aplicaciones con la misma primicia.

En la cual partimos desde la creación de usuarios, administración y gestión de información para empezar el nombre asignado a la aplicación fue acuerdo comercial (Merchant Agreement) como su nombre lo dice realizamos un formulario comercial dirigido para los ciudadanos de los Estados Unidos en un enfoque practico; clientes dueños de bares, restaurantes, barberías etcétera con el fin de ofrecerles un servicio futuro o contratación a un punto de venta (POS). Se trabajo de manera remota con la complicitad de las tecnologías nuevas y emergentes para hacer esto mucho más práctico.

Figura 1

Ilustración 1 Terminal, Catapulta Pay, 10 diciembre 2024, catapultapay.com.



La aplicación tiene la necesidad de dos tipos de perfiles: el administrador y el usuario. El administrador tiene la oportunidad de crear, leer, actualizar y borrar. Al iniciar sesión se podrá consultar los folios de los usuarios que están registrados, editar los registros actualizando su información subiendo nueva información o documentos que no estuvieron a la mano a la hora del registro, al igual que consultar la información, y de ser necesario se podrá borrar el registro, datos, documentos y fechas.

Por otro lado, el usuario observa la página principal (Home) el botón para empezar el formulario, llenar el acuerdo comercial con sus datos los cuales tiene algunos datos opcionales y otros que en ese momento son de requerimiento. Contestara tanto preguntas personalizadas para el negocio como preguntas para configurar su entorno de trabajo. Enviara sus datos para revisión del personal encargado de esa área.

Índice

Agradecimientos-----	1
Resumen -----	2
1.- Introducción -----	7
2.- Descripción de la empresa y del puesto o área de trabajo del estudiante -----	8
2.1.- Área de trabajo del estudiante -----	9
3.- Problemas a Resolver, Priorizándolos -----	18
4.- Objetivos (General y Específicos)-----	19
4.1.- Objetivos Específicos -----	19
5.- Justificación -----	20
6.- Marco Teórico (fundamentos teóricos). -----	21
6.1.- Historia de Kotlin -----	21
6.2.- ¿Qué es Kotlin? -----	22
6.2.1.- Estructura del proyecto -----	24
6.2.2.- Sistema de compilación de Gradle -----	26
6.3.- Android Studio -----	27
7.- Procedimiento y descripción de las actividades realizadas. -----	30
7.1.- Arquitectura del software-----	30
7.2.- Documentación del código-----	34
7.3.- Segunda página principal-----	46
8.- Estructura del proyecto e interfaz grafica -----	60
9.- Conclusión del análisis de vulnerabilidades y arquitectura del software -----	74
10.- Referencias -----	75

Índice ilustrativo

Ilustración 1 Terminal, Catapulta Pay, 10 diciembre 2024, catapultapay.com.....	2
Ilustración 2 representación gráfica full stack, de Aaron Smolenskaskas, 1 de enero del 2024	11
Ilustración 3 Terminal, Catapulta Pay, 10 diciembre 2024, catapultapay.com	13
Ilustración 4 metodología ágil, de ilunion, 24 de enero de 2025	15
Ilustración 5 Icono Kotlin, JetBrains, 5 May 2021, commons.wikimedia.org.	23
Ilustración 6 Archivos del proyecto en la vista de proyectos de Android, 11 de junio del 2025, developer.android.com.....	25
Ilustración 7 Adarsh Fernando, Group Product Manager, 2 de marzo del 2024	29
Ilustración 8, Modelo Vista Controlado, precognis, 10 de agosto de 2022	32
Ilustración 9, Captura de pantalla, estructura del proyecto, 10 de enero de 2024	60
Ilustración 10, Captura de pantalla, Home UI, 10 de enero de 2024	61
Ilustración 11, Captura de pantalla, Home UI, 10 de enero de 2024.....	62
Ilustración 12, Captura de pantalla, página principal, 10 de enero de 2024	63
Ilustración 13, Captura de pantalla, SelecOneMenu, 10 de enero de 2024	64
Ilustración 14, Captura de pantalla, información del negocio, 10 de enero de 2024.....	65
Ilustración 15, Captura de pantalla, telefono/dirección, 10 de enero de 2024	66
Ilustración 16, Captura de pantalla, dirección email, 10 de enero de 2024	67
Ilustración 17, Captura de pantalla, fecha, 10 de enero de 2024.....	68
Ilustración 18, Captura de pantalla, selección de fecha, 10 de enero de 2024	69

Ilustración 19, Captura de pantalla, información personal segunda página, 10 de enero de 2024.....	70
Ilustración 20, Captura de pantalla, recolección de datos, 10 de enero de 2024	71
Ilustración 21, Captura de pantalla, contacto del proveedor, 10 de enero de 2024	72
Ilustración 22, Captura de pantalla, formato numérico, 10 de enero de 2024	73

1.- Introducción

Un software a la medida para la empresa en este caso es una necesidad, la solución proporcionada y conveniente es la creación de una aplicación móvil con la tecnología emergente para el desarrollo de aplicaciones Kotlin en su última versión, se trabajó el IDE de programación Android Studio.

Para llegar al resultado esperado del proyecto se cotejaron distintas posibilidades y opciones, tecnologías potentes he investigaciones de compatibilidad con el hardware y software existente en la empresa se atendieron necesidades que vienen implementadas en la aplicación, se trabajó un desarrollo Full-Stack con una breve investigación en la plataforma AWS para el levantamiento de la aplicación en un nivel ya de producción.

El proyecto tiene un modelo de arquitectura de software MVC, por sus siglas modelo vista controlador. El patrón Modelo Vista-Controlador (MVC) es una forma de organizar el código de una aplicación separando las responsabilidades en tres partes principales: el Modelo, que maneja los datos y la lógica de negocio; la Vista, que se encarga de mostrar la información al usuario; y el Controlador, que actúa como intermediario recibiendo las acciones del usuario y actualizando el Modelo y la Vista. Esta separación facilita el desarrollo, mantenimiento y reutilización del código, haciendo que la aplicación sea más escalable y fácil de entender.

Kotlin es un lenguaje de programación moderno y conciso, diseñado para ser seguro y fácil de usar, que se ejecuta en la máquina virtual de Java y también puede compilarse a código JavaScript o nativo. Destaca por su interoperabilidad total con Java, lo que permite a los desarrolladores utilizar bibliotecas y frameworks existentes sin problemas. Kotlin introduce características como la inferencia de tipos, las corrutinas para programación asíncrona y la

seguridad nula, lo que ayuda a prevenir errores comunes y a escribir código más limpio y eficiente.

2.- Descripción de la empresa y del puesto o área de trabajo del estudiante

Catapulta Pay, es una empresa que se enfoca en negocios latinos, manteniendo seguras sus finanzas, gestionando sus ingresos y favoreciendo su ahorro, siendo ese su mayor objetivo.

Ofrece servicios de procesamiento de pagos por medio de tecnología sofisticada de Hardware y Software, así como también, ofrece asesoría financiera y venta de procesadores TVP.

La empresa proporciona una interfaz de pago que permite a los comerciantes aceptar pagos de una variedad de fuentes, incluidas tarjetas de débito, tarjetas de crédito, billeteras móviles, Venmo, PayPal, Cámara de Compensación Automatizada (ACH), transferencias bancarias y compre ahora, pague después (BNPL) transacciones e incluso criptomonedas. Brindamos servicios de pago tanto al final de la transacción del cliente como del comerciante.

Las pasarelas de pago son sistemas seguros de pago en línea que conectan su tienda en línea con su servicio de procesamiento de tarjetas para verificar y completar transacciones en línea.

Algunos procesadores tienen portales en línea seguros que permiten una variedad de opciones modernas de pago en línea, incluidos pagos de facturas en línea, pagos recurrentes y de suscripción automatizados, pagos con códigos QR y enlaces de pago enviados por mensaje de texto o correo electrónico.

Casi todos los servicios de procesamiento de tarjetas admiten pagos ingresados a través de un panel en línea basado en la nube, generalmente llamado terminal virtual.

Muchos servicios de procesamiento de tarjetas ofrecen una aplicación de pago móvil que permite realizar pagos virtuales mediante tabletas y teléfonos inteligentes. Algunos proveedores incluyen esto en una aplicación POS que admite pagos tanto con llave como con lector de tarjetas.

2.1.- Área de trabajo del estudiante

Desarrollador full-stack Kotlin (móvil) en sistemas POS Responsabilidades del puesto (front-end Android y back-end) Desarrollo de la aplicación móvil (Android): Diseñar y codificar la interfaz y la lógica del sistema POS en Kotlin, implementando flujos de venta, pantalla de cobro, impresión de recibos, manejo de devoluciones, etc. Esto incluye integrar hardware como lectores de tarjetas, impresoras y dispositivos NFC. Como indica un anuncio de trabajo, el desarrollador POS “crea aplicaciones Android POS complejas, integrándolas con cajas registradoras y sistemas fiscales”.

Back-end y servicios: Construir y mantener las APIs y servicios necesarios para el POS (por ejemplo, consultas de productos, sincronización con inventario, comunicaciones con pasarelas de pago, gestión de usuarios). Emplear frameworks (Ktor, Spring Boot u otros) y bases de datos relacionales (PostgreSQL) para el servidor. Se encarga de la lógica de negocio, la seguridad y la persistencia de datos (p.ej. ventas registradas). Gestión de ciclo de vida del software: Participar en la definición de requerimientos funcionales y no funcionales, revisiones de diseño y arquitectura, estimaciones de esfuerzo, documentación técnica y control de versiones. Hacer escritura de pruebas unitarias/UI. Un perfil completo debe asegurar “código de alta calidad y pruebas unitarias, y asistir en pruebas de integración y aceptación”.

Calidad y estándares: Garantizar que la solución cumpla con estándares del dominio (por ejemplo, normas de pagos electrónicos). El anuncio citado resalta que el desarrollador POS “asegura el cumplimiento con estándares y prácticas del pago”. Esto implica implementar cifrado,

autenticación, manejo de errores, y cumplir con normativas de fiscalización (en algunos países, TPV móviles deben emitir facturas electrónicas o reportar transacciones).

Soporte y mantenimiento: Resolver errores y actualizar la aplicación. Dar soporte postlanzamiento, para corregir bugs reportados y mejorar funcionalidades existentes. También, participar en la monitorización del sistema en producción y en la incorporación de nuevos requerimientos (por ejemplo, nuevas formas de pago o legislaciones). Herramientas y tecnologías comunes (POS Android/Kotlin)

Entorno de desarrollo: Android Studio (IDE oficial), Gradle (sistema de compilación) y herramientas de depuración como ADB y emuladores de Android. Control de versiones con Git.

Lenguaje y librerías base: Kotlin (principal) y/o Java. Uso intensivo de Android SDK y bibliotecas.

Android Jetpack: por ejemplo, Jetpack Compose o vistas XML para interfaz, ViewModel/LiveData (MVVM) para arquitecturas limpias, Room o SQLite para persistencia local, y Navigation, WorkManager, Paging, etc., para manejo de navegación, trabajos en segundo plano e hilos. Frameworks o patrones como MVVM o Clean Architecture garantizan modularidad y escalabilidad.

Inyección de dependencias: Dagger/Hilt o alternativas (Koin, Toothpick) para gestionar dependencias y facilitar pruebas.

Redes y comunicación: Retrofit (sobre OkHttp) para llamadas REST/HTTP, Parseadores JSON (Gson/Moshi), y posiblemente clientes GraphQL. Pruebas de API con Postman o Charles Proxy. Hardware de pago e integraciones POS: SDKs específicos de terminales bancarios (por ejemplo, el PAX SDK) y librerías para comunicación NFC, BLE o USB con periféricos (lectores de

tarjetas, balanzas, impresoras). La lista de un caso de estudio incluye explícitamente “PAX SDK, NFC” como integraciones clave. Esto permite procesar pagos con tarjeta, contactless, imprimir tickets, etc. Bases de datos: PostgreSQL o MySQL en servidor, y bases locales (Room/SQLite) en el móvil. A veces NoSQL para datos offline o caché.

Figura 2

Ilustración 2 representación gráfica full stack, de Aaron Smolenskys, 1 de enero del 2024



Otros: Herramientas de testing (JUnit, Espresso, Mockito), análisis estático de código (Lint, SonarQube), CI/CD (Jenkins, GitHub Actions) para integraciones continuas, y servicios en la nube (AWS, Azure, GCP) si el backend es cloud. También se aprovechan frameworks de pago externo como Google Pay o pasarelas en la nube.

Fuente: Por ejemplo, un estudio de caso de desarrollo POS en Android detalla el uso de “Kotlin, Moxy, RxJava, Toothpick, Cicerone, Retrofit” más integraciones específicas “PAX SDK, NFC”. Otras referencias mencionan Android Studio, ADB, SQLite, Gradle, Dagger y Retrofit como herramientas típicas del desarrollo Android moderno.

Habilidades técnicas requeridas Dominio de Kotlin y Android: Conocimiento profundo del lenguaje Kotlin (sus conceptos de programación funcional, corutinas para concurrencia, tipado y extensión) y la plataforma Android (ciclo de vida de actividades/fragments, componentes de IU, manejo de hilos, servicios, broadcast receivers, etc.). Se valora experiencia previa en apps comerciales Android nativas.

Desarrollo de UI/UX móvil: Experiencia creando interfaces intuitivas para pantallas táctiles grandes (tablets/terminales) y optimizadas para rapidez en el flujo de venta. Conocimiento de Material Design o principios similares.

Habilidad para implementar interfaces reactivas (Jetpack Compose o XML) y manejar adaptabilidad de UI. Patrones de arquitectura: Manejo de arquitecturas limpias (MVVM, MVP, Clean Architecture) para separar responsabilidades y facilitar pruebas. Conocimiento de inyección de dependencias (Dagger/Hilt, Koin) y uso de MVVM con ViewModel, LiveData/StateFlow o frameworks similares. Concurrencia y asíncrono: Uso de Kotlin Coroutines (Flows) o RxJava para operaciones asíncronas (llamadas a red, acceso a BD). Se requiere optimizar tareas intensivas para no bloquear la IU. Redes y APIs: Construir y consumir APIs RESTful, procesar JSON/XML. Conexiones seguras (HTTPS, OAuth/JWT si es necesario). Experiencia con Retrofit/OkHttp o clientes HTTP equivalentes en Kotlin.

Bases de datos: Saber diseñar esquemas y consultas SQL, uso de ORM de Android (Room). En backend, manejar bases de datos relacionales (PostgreSQL, MySQL) o NoSQL (Mongo, Firebase) según arquitectura.

Pruebas y calidad: Escribir pruebas unitarias (JUnit, Mockito), de integración y pruebas de interfaz (Espresso). Automatizar testing para evitar regresiones. Control de calidad con revisiones de código.

Figura 3

Ilustración 3 Terminal, Catapulta Pay, 10 diciembre 2024, catapultapay.com



Seguridad en pagos: Conocer estándares de seguridad de la industria financiera. En particular, cumplir con PCI DSS cuando se manejan datos de tarjetas. Esto implica cifrar datos sensibles (AES/TLS), no almacenar CVV, validar dispositivos seguros, y seguir buenas prácticas de codificación. También aplicar principios generales de ciberseguridad (input sanitization, autenticación segura, etc.) y respetar normativas como GDPR si aplica a datos personales.

Habilidades complementarias: Dominio de herramientas de control de versiones (Git), plataformas de CI/CD, y metodologías ágiles (Scrum/Kanban). Habilidad de análisis de requerimientos y comunicación efectiva con stakeholders. Actitud de aprendizaje continuo (Kotlin evoluciona rápidamente).

Fuente: Se esperan conocimientos sólidos de Kotlin/Android (lenguaje, SDK, desarrollo de apps) y herramientas asociadas, así como experiencia con Android Studio, corutinas y bibliotecas estándar. Además, es crítico entender normas de seguridad de pagos (PCI DSS). Flujo de trabajo típico (POS móvil) En proyectos de POS móvil se suele trabajar con un equipo multidisciplinario (gestor de proyecto, analista de negocio, desarrollador Android, diseñador UI/UX, QA, etc.) en un esquema iterativo. Por ejemplo, un caso real describe un equipo que “reunió un gestor de proyecto, analista de negocio, desarrollador Android, diseñador UI/UX e ingeniero de QA” trabajando con metodología Scrum y sprints de 2–3 semanas.

El proceso típico incluye; Análisis y diseño: Inicialmente se obtienen los requerimientos del cliente (funcionales y no funcionales) y se redactan las especificaciones. Se definen casos de uso (p.ej. ventas, devoluciones, cierre de caja) y flujos de pantalla. En paralelo, el equipo de UI/UX diseña prototipos de las interfaces.

Figura 4

Ilustración 4 metodología ágil, de ilunion, 24 de enero de 2025



Desarrollo iterativo: En cada sprint (iteración corta) se implementan funcionalidades nuevas o mejoras. Se realiza codificación en Kotlin siguiendo el diseño establecido (patrones MVVM, clean code), se realizan pull requests y revisiones de código para garantizar calidad. Se usan entornos de testing (incluyendo sandboxes de terminal POS como el PAX sandbox para emular hardware de pago) y herramientas de prueba (Postman, Charles Proxy) para validar la integración con APIs y hardware. **Pruebas e integración:** Se ejecutan pruebas unitarias y de interfaz continuamente. Además, se integran con sistemas externos (pasarelas de pago, servidores fiscales, gestión de inventario). En esta fase se verifica el procesamiento correcto de pagos, impresión de tickets y robustez de la app bajo diferentes condiciones (offline/online, errores de red, etc.).

Despliegue y mantenimiento: Al finalizar un conjunto de historias de usuario, la aplicación se compila para despliegue (p.ej. subida a Google Play o distribución empresarial). Tras el lanzamiento, se monitorea el funcionamiento y se corrigen incidentes.

Se entrega soporte continuo: en el ejemplo citado, “la solución está desplegada, e Innowise proporciona mantenimiento y asistencia tras el lanzamiento”. Esto incluye actualizaciones por cambios legales (fiscalización) o nuevas necesidades (nuevos métodos de pago). Este flujo ágil garantiza que el POS móvil evolucione según las necesidades del negocio, con ciclos cortos de entrega y constante retroalimentación con el cliente. Buenas prácticas específicas del dominio POS móvil

Configurabilidad del sistema: Permitir parametrizar aspectos clave (tasas de impuestos, tipos de pago, descuentos, formatos de recibo, roles de usuario, etc.). Esto garantiza que el POS pueda adaptarse fácilmente a distintos clientes o países.

Integraciones planificadas: Diseñar la arquitectura para integrar de forma robusta con terceros (pasarelas de pago, sistemas de contabilidad, puntos de inventario, CRM). Definir claramente qué datos se intercambian y con qué frecuencia, usando APIs seguras y controladas.

Seguridad y cumplimiento: Aplicar cifrado (datos en tránsito y reposo) y mecanismos de autenticación fuertes. Asegurar el cumplimiento de PCI DSS para pagos con tarjeta, usando tokenización cuando sea posible.

Pruebas exhaustivas: Automatizar pruebas unitarias e integración para evitar regresiones. Realizar pruebas de rendimiento (carga/stress) dado que el sistema maneja transacciones críticas. Actualizar regularmente librerías y componentes (parches de seguridad).

Soporte y documentación; Planificar mantenimiento continuo: monitorear el POS en producción, gestionar incidencias y actualizaciones tecnológicas. Documentar el sistema (manuales de usuario y técnicos) para facilitar transferencias y emergencias.

Experiencia de usuario: Optimizar la interfaz para rapidez de uso en el punto de venta (botones grandes, navegación simple), minimizando pasos. Validar prototipos con usuarios finales (cajeros, meseros) para asegurar usabilidad y eficiencia. Estas prácticas se basan en estudios técnicos del área POS, que enfatizan la necesidad de un sistema configurado para cada negocio, seguro y bien probado.

Expectativas del mercado laboral actual el perfil de desarrollador full-stack Kotlin/Android sigue con alta demanda, especialmente en sectores retail, hostelería y fintech que modernizan sus puntos de venta. Kotlin se ha consolidado como “el estándar de facto para Android”: más del 60% de desarrolladores Android profesionales lo usan y cerca del 70% de las 1000 apps más populares ya incluyen código Kotlin. En ellas se valoran tanto habilidades móviles como conocimientos de integración de pagos. Dado que los desarrolladores Kotlin “pueden escribir código más rápido que Java”, empresas buscan estos perfiles para mejorar productividad y reducir costos. Además, la empresa espera que el candidato cubra tanto front-end (app Android) como aspectos back-end o cloud. Se valoran conocimientos en arquitecturas escalables (microservicios, contenedores), prácticas DevOps y herramientas modernas (Docker, Kubernetes, CI/CD), así como habilidad para adaptarse a nuevas tecnologías de pago digital. En resumen, se exige un desarrollador Android/Kotlin competente en toda la pila (UI, APIs, bases de datos), actualizado en tendencias móviles y normas de seguridad del comercio electrónico.

3.- Problemas a Resolver, Priorizándolos

Como punto número uno y a sazón de uno de los motivos principales ya antes visto en este tipo de empresas puede existir o no un equipo de desarrollo de software, como en este caso la empresa se caracteriza (el giro de la empresa) por sus micro transacciones, la instalación, creación, administración y mantenimiento de *puntos de venta* (POS).

Alquiler de un Software de Terceros

La situación de la empresa y problema principal es el tener una “fuga” de activo al contratar un servicio de terceros ya desde hace algún tiempo, lo que se prevé es un ahorro de mayor connotación así haciendo un traje a la medida y priorizando cada requerimiento específico de la empresa.

Llevar Registro y Organización de los Clientes

Por medio de este salvo conducto que la empresa está adquiriendo llevara su registro detallado de cada cliente, el cual se debe subir primeramente a la base de datos de la empresa por consiguiente trabajaremos de diversas formas esta información, la principal y con requerimiento en la empresa es la creación de un archivo PDF del cliente si esta así lo llegase a necesitar.

Microservicios Ahora Existentes. De menor prioridad sin embargo no menos importante se encuentran los microservicios que ofrece la empresa con la información recabada. Se encuentra gran escalabilidad y flexibilidad en la aplicación puesto que Kotlin es un lenguaje de programación de código abierto. Esto significa que su código fuente está disponible públicamente y puede ser utilizado, modificado y distribuido por cualquier persona. Aunque JetBrains es el principal impulsor de Kotlin, el lenguaje cuenta con una gran comunidad de desarrolladores que

contribuyen con mejoras y bibliotecas. Google también apoya activamente Kotlin, especialmente para el desarrollo de aplicaciones Android.

4.- Objetivos (General y Específicos)

Organizar, automatizar, brindar confort y hacer intuitivos los procedimientos dentro de la empresa con el enfoque respecto a la aplicación móvil desarrollada desde cero para su complicidad en los resultados que se ofrecen.

Desplegué de la aplicación con sus requerimientos específicos el éxito de esta aplicación se mide de la forma en que los archivos que se usan de manera primitiva en *Google Drive* pasen a ser archivos convertidos a PDF evitando el uso de software con licencias (de terceros).

4.1.- Objetivos Específicos

El cliente evita entrar en sitios o URL con distinto dominio que el existente en la empresa esto genera desconfianza y poco profesionalismo para grandes marcas o clientes potencialmente interesados. Se genera un ambiente de confianza y reciprocidad con el cliente, marca o empresa que use nuestros servicios.

Se propone una arquitectura de software limpia (clean architecture) conveniente al proyecto se espera la fácil comprensión del código fuente para su futura evolución (escalabilidad) módulos nuevos implementación de vanguardia, etc.

Hablemos de lo intuitivo del proyecto lo que va a facilitar en demasía el trabajo que están haciendo actualmente los empleados de la empresa, en el uso de grandes cantidades de datos.

5.- Justificación

El propósito de este proyecto es el dar solución a los procesos en los que existe un cuello de botella en la empresa; automatizarlos, ofrecer una opción más viable y reducir significativamente el gasto existente en cuestión de licencias.

El trabajo en equipo con el personal de la empresa es un punto clave para el proyecto cada actualización de requerimiento ya sea un módulo nuevo, una funcionalidad extra. Porque la necesidad de la empresa puede ir cambiando en base al tiempo que se está trabajando.

Este proyecto optimiza el tráfico de datos y lleva un registro para cada usuario de ser necesario la información del cliente llegaría en formato PDF a través de la aplicación.

La tecnología que se está utilizando es de código abierto quiere decir que tiene soporte por la comunidad y las empresas filiales difícilmente la aplicación podría quedar obsoleta, sino al contrario la escalabilidad que esta ofrece es muy satisfactoria.

6.- Marco Teórico (fundamentos teóricos).

6.1.- Historia de Kotlin

En julio de 2011 JetBrains reveló el Proyecto Kotlin, un nuevo lenguaje para la JVM, el cual había estado bajo desarrollo por un año. El líder de JetBrains, Dmitry Jemerov, dijo que la mayoría de lenguajes no tienen las características que buscaban, con la excepción de Scala. Sin embargo, citó el lento tiempo de compilación de Scala como una deficiencia obvia. Uno de los objetivos establecidos de Kotlin es el de compilar tan rápido como Java.

En febrero de 2012, JetBrains liberó el código fuente del proyecto bajo la Licencia Apache 2.

En 2017, recibe un importante impulso al ser nombrado por Google como lenguaje oficial para Android al mismo nivel que Java.

JetBrains es la empresa detrás de Kotlin, un lenguaje de programación moderno y versátil. Originalmente creado para uso interno, Kotlin se ha convertido en un lenguaje de código abierto muy popular, especialmente para el desarrollo de aplicaciones Android. JetBrains no solo desarrolló el lenguaje, sino que también proporciona herramientas de desarrollo de alta calidad, como IntelliJ IDEA y Android Studio, que ofrecen un excelente soporte para Kotlin.

6.2.- ¿Qué es Kotlin?

Kotlin es un lenguaje de programación orientado a objetos multiplataforma, estáticamente tipado, de alto nivel y propósito general con inferencia de tipos. Kotlin está diseñado para ser totalmente interoperable con Java, y la versión JVM de la biblioteca estándar de Kotlin depende de la Java Class Library, pero la inferencia de tipos permite que su sintaxis sea más concisa.

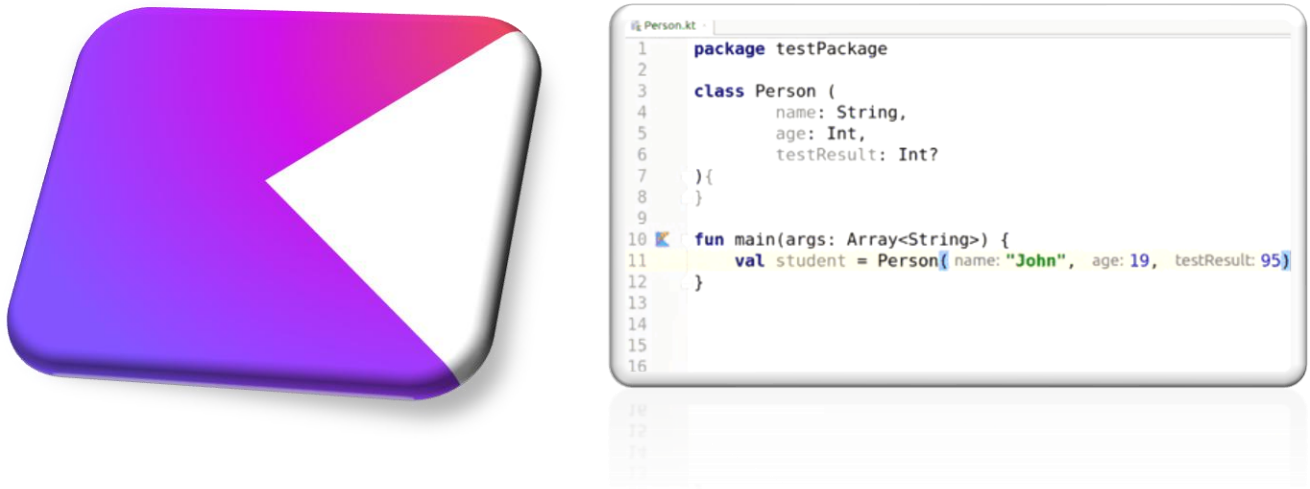
Kotlin se dirige principalmente a la JVM, pero también compila a JavaScript (por ejemplo, para aplicaciones web front-end que utilizan React) o código nativo a través de LLVM (por ejemplo, para aplicaciones nativas de iOS que comparten lógica de negocio con aplicaciones Android). Los costes de desarrollo del lenguaje corren a cargo de JetBrains, mientras que la Fundación Kotlin protege la marca comercial Kotlin.

El 7 de mayo de 2019, Google anunció que el lenguaje de programación Kotlin es ahora el lenguaje preferido para los desarrolladores de apps Android, y desde el lanzamiento de Android Studio 3.0 en octubre de 2017, Kotlin se ha incluido como alternativa al compilador Java estándar. El compilador Kotlin de Android produce bytecode Java 8 por defecto (que se ejecuta en cualquier JVM posterior), pero permite al programador elegir como objetivo Java 9 a 18 para su optimización, o permite más características; tiene soporte para la interoperabilidad bidireccional de clases de registro para la JVM, introducida en Java 16, considerada estable a partir de Kotlin 1.5.

Kotlin tiene soporte para la web; compilando a JavaScript (es decir, Kotlin/JS con el backend clásico, declarado estable a partir de la versión 1.3), mientras que el más nuevo Kotlin/JS (basado en lenguaje intermedio) está en beta a partir de la versión 1.5.30. Kotlin/Native (por ejemplo, para compatibilidad de Apple Silicon) está en beta desde la versión 1.3

Figura 5

Ilustración 5 Icono Kotlin, JetBrains, 5 May 2021, commons.wikimedia.org.



Android Studio es el entorno de desarrollo integrado (IDE) oficial que se usa en el desarrollo de apps para Android. Basado en el potente editor de código y las herramientas para desarrolladores de IntelliJ IDEA, Android Studio ofrece aún más funciones que mejoran tu productividad cuando compilas apps para Android, como las siguientes:

- Un sistema de compilación flexible basado en Gradle
- Un emulador rápido y cargado de funciones
- Un entorno unificado donde puedes desarrollar para todos los dispositivos Android
- Ediciones en vivo para actualizar elementos componibles en emuladores y dispositivos físicos, en tiempo real
- Integración con GitHub y plantillas de código para ayudarte a compilar funciones de apps comunes y también importar código de muestra
- Variedad de marcos de trabajo y herramientas de prueba

- Herramientas de Lint para identificar problemas de rendimiento, usabilidad y compatibilidad de versiones, entre otros
- Compatibilidad con C++ y NDK
- Compatibilidad integrada con Google Cloud Platform, que facilita la integración con Google Cloud Messaging y App Engine

6.2.1.- Estructura del proyecto

Cada proyecto de Android Studio incluye uno o más módulos con archivos de código

Módulos de apps para Android

Módulos de biblioteca

Módulos de Google App Engine

De manera predeterminada, Android Studio muestra los archivos de tu proyecto en la vista de proyecto de Android, como se ve en la figura 3. Esta vista está organizada en módulos para que puedas acceder rápidamente a los archivos fuente clave de tu proyecto. Puedes ver todos los archivos de compilación en el nivel superior, en Gradle Scripts.

Cada módulo de app contiene las siguientes carpetas:

manifests: Contiene el archivo AndroidManifest.xml.

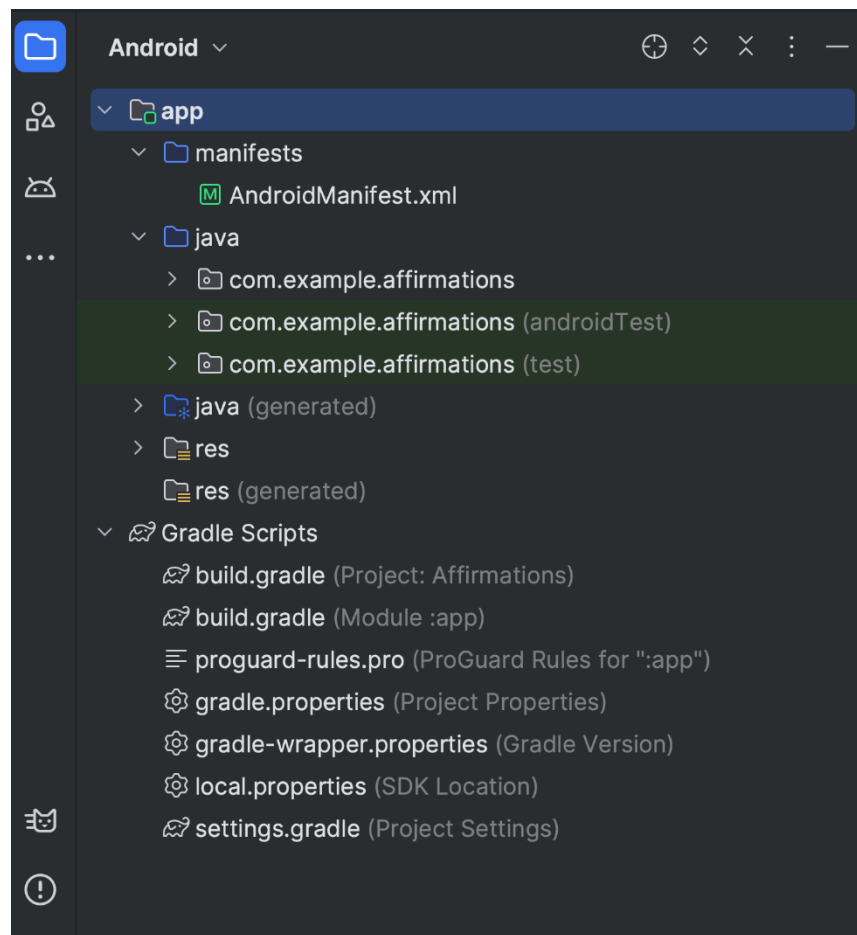
java: Contiene los archivos de código fuente de Kotlin y Java, incluido el código de prueba JUnit.

res: Contiene todos los recursos sin código, como cadenas de IU y, además, imágenes de mapa de bits.

La estructura del proyecto de Android en el disco difiere de esta representación plana. Para ver la estructura real de archivos del proyecto, selecciona Project en lugar de Android en el menú Project.

Figura 6

Ilustración 6 Archivos del proyecto en la vista de proyectos de Android, 11 de junio del 2025, developer.android.com



6.2.2.- Sistema de compilación de Gradle

Android Studio usa Gradle como base del sistema de compilación, y el complemento de Android para Gradle proporciona capacidades específicas de Android. Este sistema de compilación se ejecuta en una herramienta integrada desde el menú de Android Studio, y lo hace independientemente de la línea de comandos. Puedes usar las funciones del sistema de compilación para lo siguiente:

Personalizar, configurar y extender el proceso de compilación

Crear varios APKs para tu app; diferentes funciones usan el mismo proyecto y los mismos módulos

Volver a utilizar códigos y recursos entre conjuntos de orígenes

Gracias a la flexibilidad de Gradle, puedes lograrlo sin modificar los archivos fuente de tu app.

Los archivos de compilación de Android Studio se denominan `build.gradle.kts` si usas Kotlin (recomendado) o `build.gradle` si usas Groovy. Son archivos de texto sin formato que usan la sintaxis de Kotlin o Groovy para configurar la compilación con elementos que proporciona el complemento de Android para Gradle. Cada proyecto tiene un archivo de compilación de nivel superior para todo el proyecto y archivos de compilación de nivel de módulo independientes para cada módulo. Cuando importas un proyecto existente, Android Studio genera automáticamente los archivos de compilación necesarios.

6.3.- Android Studio

Android Studio es el entorno de desarrollo integrado oficial para la plataforma Android. Fue anunciado el 16 de mayo de 2013 en la conferencia Google I/O, y reemplazó a Eclipse como el IDE oficial para el desarrollo de aplicaciones para Android. La primera versión estable fue publicada en diciembre de 2014.

Está basado en el software IntelliJ IDEA Community Edition de JetBrains y ha sido publicado de forma gratuita a través de la Licencia Apache 2.0. Está disponible para las plataformas Microsoft Windows, macOS y GNU/Linux. Ha sido diseñado específicamente para el desarrollo de Android.

Android Studio es mucho más que un simple programa; es el entorno de desarrollo integral y oficial que Google pone a disposición de los programadores para crear aplicaciones que funcionen en cualquier dispositivo Android. Imagina que quieres construir una casa. Necesitarías un terreno, planos, herramientas específicas y un lugar donde guardarlo todo. Android Studio es ese lugar: un taller completo que te proporciona todo lo necesario para "construir" aplicaciones.

Un Espacio de Trabajo Completo para desarrolladores en su corazón, Android Studio está construido sobre una potente plataforma llamada IntelliJ IDEA. Esto significa que viene con un editor de código muy inteligente, capaz de ayudarte a escribir código de manera más rápida y con menos errores, sugiriéndote opciones, corrigiendo fallos y reorganizando tu código para que sea más legible. Puedes programar principalmente en Kotlin o Java, los dos lenguajes más comunes para el desarrollo de apps en Android, y también tiene soporte para C++ si necesitas un control más profundo del hardware.

Pero el desarrollo de aplicaciones no es solo escribir código. También necesitas diseñarlas. Android Studio incluye un diseñador visual que te permite arrastrar y soltar elementos de la interfaz, como botones, textos e imágenes, para crear cómo se verá tu aplicación, y lo hace mientras te muestra una previsualización en tiempo real. Así, puedes ver cómo lucirá en diferentes tamaños de pantalla, desde un teléfono pequeño hasta una tableta grande.

Probando y pulido tus creaciones una vez que has escrito el código y diseñado la interfaz, necesitas probar tu aplicación. Android Studio cuenta con un emulador de Android muy potente. Esto es como tener una colección de teléfonos y tabletas virtuales en tu computadora. Puedes simular cómo se comporta tu aplicación en diferentes dispositivos, sin la necesidad de comprarlos físicamente, lo que es increíblemente útil para asegurarte de que tu app funciona bien en cualquier lugar.

Además, cuando algo no funciona como esperas, el entorno te ofrece herramientas de depuración avanzadas que te permiten revisar el código paso a paso para encontrar y solucionar problemas. También tiene herramientas de rendimiento que te ayudan a optimizar tu aplicación, asegurando que use la batería de forma eficiente y que sea rápida y fluida para el usuario final.

Siempre conectado y en constante evolución Android Studio se integra perfectamente con el sistema de construcción Gradle, que es como un capataz inteligente que se encarga de organizar todo el proceso para convertir tu código en una aplicación lista para instalar. También facilita la conexión con servicios de Google, como Firebase (una plataforma para el desarrollo de aplicaciones móviles) o Google Cloud Platform, lo que te permite añadir funcionalidades avanzadas a tus apps, como bases de datos en la nube o autenticación de usuarios.

Figura 7

Ilustración 7 Adarsh Fernando, Group Product Manager, 2 de marzo del 2024



Incluso en sus versiones más recientes, Android Studio incorpora asistentes de inteligencia artificial, como Gemini, que pueden ayudarte a generar código, corregir errores o incluso responder preguntas sobre el desarrollo de Android. En definitiva, Android Studio es la herramienta esencial y más completa para cualquier persona que desee crear aplicaciones robustas y de alta calidad para el vasto mundo de dispositivos Android.

7.- Procedimiento y descripción de las actividades realizadas.

7.1.- Arquitectura del software

La arquitectura de software es la estructura fundamental para el sistema, define cómo se organizan sus componentes y cómo interactúan. Una buena arquitectura determina qué partes existen (módulos, servicios, capas) y cómo se relacionan para cumplir requisitos funcionales y no funcionales (rendimiento, seguridad, escalabilidad, etc.). Una arquitectura sólida guía las decisiones de diseño, facilita la evolución del sistema y sirve de columna vertebral que sostiene la calidad técnica a largo plazo.

La arquitectura define la distribución de recursos (cómputo, memoria, red) y la forma de procesar datos, lo cual impacta directamente en la velocidad de respuesta. Un diseño eficiente (p. ej. uso de cachés, bases de datos optimizadas o microservicios especializados) reduce la latencia y mejora la experiencia de usuario. Por ejemplo, integrar caché en memoria o CDN para contenido estático descarga a los servidores y agiliza el acceso a datos frecuentes. Sin arquitectura pensada para el rendimiento, la aplicación puede atascarse bajo carga: largos tiempos de carga frustran usuarios y pueden traducirse en pérdida de clientes. Buenas prácticas: usar capas de caché (memoria, CDN), balanceo de carga, bases de datos indexadas y consultas optimizadas. Diseñar servicios ligeros que respondan rápido (por ejemplo, APIs REST que devuelvan sólo lo necesario) y paralelizar procesos cuando sea posible. Errores comunes: una única base de datos centralizada sin caché, o servicios monolíticos masivos mal optimizados, generan cuellos de botella. Por ejemplo, aplicaciones de e-commerce con arquitectura monolítica simple suelen colapsar en eventos de alto tráfico (Cyber Monday) porque no pueden procesar suficientes peticiones concurrentes.

Una buena arquitectura facilita el escalado horizontal (añadir más instancias) y vertical (mejorar hardware), así como la partición adecuada de datos. Por ejemplo, una arquitectura modular o basada en microservicios permite escalar sólo las partes críticas (aumentar réplicas de la base de datos de usuarios) sin tener que duplicar todo el sistema.

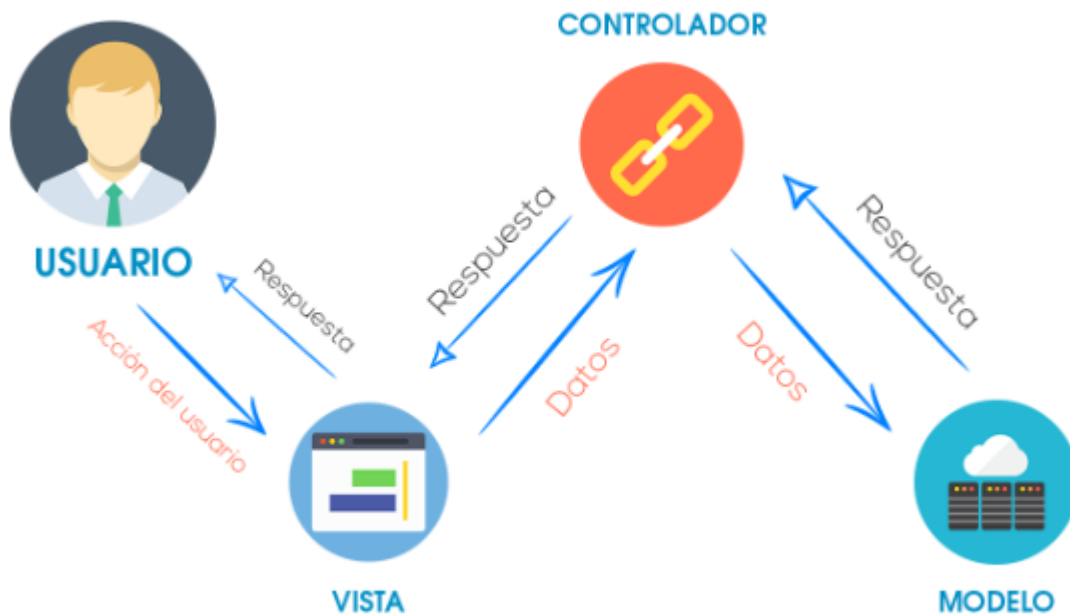
Esto mantiene el rendimiento incluso con tráfico creciente, un diseño monolítico denso suele obligar a escalar la aplicación entera (duplicar toda la máquina) lo que resulta ineficiente. Ejemplo real: Netflix enfrentó este reto. En 2009, su plataforma monolítica no soportaba el rápido crecimiento de usuarios de streaming. Migró sus centros de datos privados a la nube y rediseñó la aplicación en microservicios independientes. Esto le permitió manejar miles de despliegues diarios y escalar cada servicio por separado (por ejemplo, distinto escalado para el servicio de recomendación versus el de reproducción de video), evitando cuellos de botella generales. En cambio, bajo una arquitectura monolítica antigua la compañía hubiera tenido que reiniciar toda la aplicación ante cada cambio, ralentizando la innovación.

Errores comunes: no prever crecimiento provoca que la arquitectura se convierta en cuello de botella. Un diseño rígido con alto acoplamiento (p. ej. un único componente que maneja todo) impedirá el escalado individual. De hecho, en un monolito “no se pueden escalar componentes individuales” y “si hay un error en un módulo, puede afectar a toda la aplicación”, lo que reduce la disponibilidad bajo alto uso.

Una arquitectura en capas (presentación, negocio, datos) permite modificar la interfaz de usuario sin tocar la capa de acceso a datos, mejorando la mantenibilidad. Además, seguir principios de diseño (SOLID: responsabilidad única, abierto/cerrado, etc.) y escribir código limpio facilita que nuevos desarrolladores comprendan el sistema.

Figura 8

Ilustración 8, Modelo Vista Controlado, precognis, 10 de agosto de 2022



Sin embargo, una arquitectura mal planificada produce deuda técnica. Decidir atajos para cumplir plazos (ignorar la calidad interna o pruebas) encarece luego cada cambio. Como advierten expertos, “priorizar funcionalidades inmediatas sobre una buena arquitectura o código mantenible hace que sea cada vez más difícil añadir mejoras, ralentizando el proyecto y aumentando su coste”. En el largo plazo, esto crea un sistema frágil donde cualquier modificación puede desencadenar efectos en cadena (el “efecto dominó”).

fomentar la modularidad y bajo acoplamiento; mantener código limpio y documentado; usar sistemas de control de versiones (Git) y revisiones de código; y aplicar pruebas unitarias/automatizadas que faciliten el refactorizado. De este modo se evita que corregir un error o agregar una funcionalidad requiera reescribir gran parte del sistema.

Incluso con buen desempeño, cualquier sistema puede fallar. La arquitectura define mecanismos de tolerancia a fallos que evitan que una falla aislada cause un colapso total. Esto incluye redundancia de componentes críticos (múltiples servidores o réplicas de base de datos). Un sistema resiliente puede seguir funcionando parcial o completamente pese a errores puntuales. Por ejemplo, en arquitecturas de microservicios cada servicio es independiente: “si un microservicio falla, el resto del sistema puede seguir funcionando”. Además, se diseñan despliegues activos (failover automático) y balanceadores de carga que detectan nodos caídos y redistribuyen tráfico. Incluso se puede optar por degradación gradual: funciones no críticas se desactivan bajo fallo, pero la aplicación principal sigue operativa.

Netflix adoptó la ingeniería del caos: utilizó herramientas (Chaos Monkey) que provocan fallos aleatorios en producción para probar y reforzar la resiliencia. Esto “obliga a los ingenieros a diseñar sistemas con redundancia y tolerancia a fallos en mente”, asegurando que perder un nodo no deje a los usuarios sin servicio. Buenas prácticas: implementar redundancia y balanceo (auto-escalado de réplicas). Usar patrones de reintentos con backoff exponencial y circuit breakers para aislar servicios con problemas. Diseñar para degradación controlada (por ejemplo, cachear respuestas críticas si la capa de negocio falla). Realizar pruebas de estrés y simulación de fallos (chaos testing) para comprobar la tolerancia. Errores comunes: una arquitectura sin redundancia ni monitoreo de fallos provoca efectos en cascada. Por ejemplo, si una base de datos sin réplicas cae, puede detener toda la aplicación. No prever failsafes convierte pequeños problemas (un servidor caído) en interrupciones mayores.

7.2.- Documentación del código

En el proyecto se utiliza la herramienta compose y sus componentes cada una de las declaraciones tiene uso adecuado y con sentido, por ejemplo, LazyColumn una herramienta semejante a Column con la diferencia que en este proyecto el scroll que tiene la aplicación es robusto la tecnología LazyColumn Carga los componentes de una aplicación conforme se va mostrando en pantalla.

1. Imports y anotaciones

- Utiliza componentes de Material3 (TextField, Button, DatePickerDialog, ExposedDropDownMenuBox, etc.), layouts (Column, LazyColumn, Row), y APIs de estado composable.
- Marca la función con `@OptIn(ExperimentalMaterial3Api::class)`.

2. Función principal

3. @Composable

4. fun LongForm(navController: NavController) { ... }

- Recibe un NavController para la navegación tras enviar el formulario.

5. Estados de los campos

Para cada campo del formulario se declara un `mutableStateOf(TextFieldValue(""))` (y su flag de error), por ejemplo:

6. var name by remember { mutableStateOf(TextFieldValue("")) }

7. var nameError by remember { mutableStateOf(false) }

Se repite para decenas de campos: nombre, dirección, ciudad, estado, código postal, teléfono, email, EIN, etc.

8. Lista de identificadores de campo

9. `val formFields = listOf(`

10. `"Name", "AsName", "Street", "City", ..., "ContactE"`

11. `)`

Se itera con `LazyColumn { items(formFields) { field -> ... } }` y, usando un `when(field)`, se renderiza cada sección del formulario.

12. Componentes de UI por sección

- `TextField` para textos y números, con validación en tiempo real (`isError` y mensajes de error).
- `DatePickerDialog` para seleccionar la fecha de fundación del negocio.
- `RadioButton` para opciones binarias o múltiples (p. ej. “Yes/No”, tipo de edificio, zona, rango de ingresos).
- `ExposedDropDownMenuBox` para elegir el “Ownership / Entity Type” entre “Restaurant” o “Retail”.
- Botón final de “Submit” que, al pulsarse, navega a la pantalla principal (`AppScreens.PrincipalActivity.route`).

13. Validación y navegación

- Cada campo actualiza un flag de error si queda en blanco.

- Se controla isValid, aunque en el código actual solo se compara nameError y emailError.
 - Al enviar, redirige con NavController.navigate(...).
-

En conjunto, el archivo define un formulario dinámico para recopilar datos detallados de un negocio (datos generales, contacto, ubicación, información fiscal y operativa), aprovechando la potencia de Compose para gestionar estado, validación y navegación de forma reactiva y declarativa.

```
import androidx.compose.foundation.layout.Arrangement
```

```
import androidx.compose.foundation.layout.Column
```

```
import androidx.compose.foundation.layout.Row
```

```
import androidx.compose.foundation.layout.Spacer
```

```
import androidx.compose.foundation.layout.fillMaxSize
```

```
import androidx.compose.foundation.layout.fillMaxWidth
```

```
import androidx.compose.foundation.layout.height
```

```
import androidx.compose.foundation.layout.padding
```

```
import androidx.compose.foundation.lazy.LazyColumn
```

```
import androidx.compose.foundation.lazy.items
```

```
import androidx.compose.foundation.text.KeyboardOptions
```

```
import androidx.compose.material3.Button

import androidx.compose.material3.DatePicker

import androidx.compose.material3.DatePickerDialog

import androidx.compose.material3.DropdownMenuItem

import androidx.compose.material3.ExperimentalMaterial3Api

import androidx.compose.material3.ExposedDropdownMenuBox

import androidx.compose.material3.ExposedDropdownMenuDefaults

import androidx.compose.material3.MaterialTheme

import androidx.compose.material3.OutlinedButton

import androidx.compose.material3.RadioButton

import androidx.compose.material3.Scaffold

import androidx.compose.material3.Text

import androidx.compose.material3.TextField

import androidx.compose.material3.rememberDatePickerState

import androidx.compose.runtime.Composable

import androidx.compose.runtime.getValue

import androidx.compose.runtime.mutableStateOf

import androidx.compose.runtime.remember

import androidx.compose.runtime.setValue

import androidx.compose.ui.Alignment
```

```
import androidx.compose.ui.Modifier

import androidx.compose.ui.text.font.FontWeight

import androidx.compose.ui.text.input.KeyboardType

import androidx.compose.ui.text.input.TextFieldValue

import androidx.compose.ui.text.style.TextAlign

import androidx.compose.ui.unit.dp

import androidx.navigation.NavController

import com.example.tiptime.navigation.AppScreens

import java.time.Instant

import java.time.ZonedDateTime

/**

 * Pantalla de formulario largo que recopila información detallada de un negocio.

 *

 * @param navController Controlador de navegación para cambiar de pantalla al enviar el
formulario.

 */

@OptIn(ExperimentalMaterial3Api::class)

@Composable

fun LongForm(navController: NavController) {
```

// Estados de los campos de texto y sus indicadores de error

```
var asNameBus by remember { mutableStateOf(TextFieldValue("")) }
```

```
var asNameError by remember { mutableStateOf(false) }
```

```
var name by remember { mutableStateOf(TextFieldValue("")) }
```

```
var nameError by remember { mutableStateOf(false) }
```

```
var city by remember { mutableStateOf(TextFieldValue("")) }
```

```
var cityError by remember { mutableStateOf(false) }
```

```
var email by remember { mutableStateOf(TextFieldValue("")) }
```

```
var emailError by remember { mutableStateOf(false) }
```

```
var estate by remember { mutableStateOf(TextFieldValue("")) }
```

```
var estateError by remember { mutableStateOf(false) }
```

```
var postal by remember { mutableStateOf(TextFieldValue("")) }
```

```
var postalError by remember { mutableStateOf(false) }
```

```
var main by remember { mutableStateOf(TextFieldValue("")) }
```

```
var mainError by remember { mutableStateOf(false) }
```

```
var point by remember { mutableStateOf(TextFieldValue("")) }
```

```
var pointError by remember { mutableStateOf(false) }
```

```
var pointLast by remember { mutableStateOf(TextFieldValue("")) }
```

```
var pointLastError by remember { mutableStateOf(false) }
```

```
var pointPhone by remember { mutableStateOf(TextFieldValue("")) }
```

```
var pointPhoneError by remember { mutableStateOf(false) }
```

```
var emails by remember { mutableStateOf(TextFieldValue("")) }
```

```
var emailsError by remember { mutableStateOf(false) }
```

```
var website by remember { mutableStateOf(TextFieldValue("")) }
```

```
var websiteError by remember { mutableStateOf(false) }
```

```
// Estado para Dropdown (Ownership / Entity Type)
```

```
var selectedState by remember { mutableStateOf("") }
```

```
var expanded by remember { mutableStateOf(false) }
```

```
val states = listOf("Restaurant", "Retail")
```

```
// Identificador de fecha
```

```
var showDate by remember { mutableStateOf(false) }
```

```
val state = rememberDatePickerState()
```

```
// Opciones de radio buttons para diversas secciones del formulario
```

```
var selectedOption by remember { mutableStateOf("Opción 1") }
```

```
var selectedOption1 by remember { mutableStateOf("Opción 1") }
```

```
var selectedOption2 by remember { mutableStateOf("Opción 1") }
```

```
var selectedOption3 by remember { mutableStateOf("Opción 1") }
```

```
var selectedOption4 by remember { mutableStateOf("Opción 1") }
```

```
// Lista de secciones del formulario
```

```
val formFields = listOf(
```

```
    "Name", "AsName", "Street", "City", "State", "Postal",
```

```
    "Main", "Point", "PointLast", "PointPhone", "Emails",
```

```
    "Website", "What", "Tax", "Customer", "TelNu", "CuSer",
```

```
    "Fax", "ContactE"
```

```
)
```

```
// Layout principal con LazyColumn para scroll
```

```
LazyColumn(
```

```
    modifier = Modifier
```

```
        .fillMaxSize()
```

```
        .padding(top = 40.dp, start = 16.dp, end = 16.dp, bottom = 50.dp),
```

```
    horizontalAlignment = Alignment.Start,
```

```
    verticalArrangement = Arrangement.Center
```

```
) {
```

```
    items(formFields) { field ->
```

```
        when (field) {
```

```
"Name" -> {  
  
    // Sección de información básica del negocio  
  
    Text(  
  
        text = "Business Information",  
  
        modifier = Modifier.padding(bottom = 8.dp),  
  
        textAlign = TextAlign.Start,  
  
        fontWeight = FontWeight.Bold  
  
    )  
  
    Text(  
  
        text = "We need to know a little about your business, please complete the  
following",  
  
        modifier = Modifier.padding(bottom = 8.dp),  
  
        textAlign = TextAlign.Start  
  
    )  
  
    Text(text = "Corporate / Legal Name **", textAlign = TextAlign.Start)  
  
    TextField(  
  
        value = name,  
  
        onChange = {  
  
            name = it  
  
            nameError = it.text.isBlank()  
  
        }  
  
    )  
}
```

```
    },  
  
    isError = nameError,  
  
    keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Text),  
  
    modifier = Modifier.fillMaxWidth()  
  
)  
  
if (nameError) {  
  
    Text(  
  
        text = "The name is required",  
  
        color = MaterialTheme.colorScheme.error,  
  
        style = MaterialTheme.typography.bodySmall  
  
    )  
  
}  
  
}  
  
// Similar documentación debe aplicarse a cada bloque "AsName", "Street", etc.  
  
// Por brevedad, omito la repetición aquí, pero se recomienda documentar cada  
sección  
  
"ContactE" -> {  
  
    // Sección final: correo de contacto y botón de envío  
  
    Text(text = "Contact Email")  
  
    TextField(  

```

```
value = website,

onValueChange = {

    website = it

    websiteError = it.text.isBlank()

},

isError = websiteError,

keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Email),

modifier = Modifier.fillMaxWidth()

)

if (websiteError) {

    Text(

        text = "The field is required",

        color = MaterialTheme.colorScheme.error,

        style = MaterialTheme.typography.bodySmall

    )

}

Text(

    text = "All fields marked with (*) are required",

    modifier = Modifier.padding(top = 12.dp, bottom = 8.dp)

)
```

```
Button(  
  
    onClick = { navController.navigate(route = AppScreens.PrincipalActivity.route) },  
  
    modifier = Modifier.fillMaxWidth()  
  
){  
  
    Text(text = "Submit")  
  
}  
  
}  
  
}  
  
Spacer(modifier = Modifier.height(16.dp))  
  
}  
  
}  
  
}
```

7.3.- Segunda página principal

/**

* Pantalla `SecondPage` de la aplicación TipTime.

*

* Esta pantalla muestra un formulario de varios campos (nombre, nacionalidad, fecha,

* licencia, dirección, etc.) y valida la entrada antes de permitir el envío.

* Utiliza Jetpack Compose con Material3 y un `LazyColumn` para recorrer dinámicamente los campos.

*/

@OptIn(ExperimentalMaterial3Api::class)

@Composable

fun SecondPage(navController: NavController) {

 // Estado de validación general del formulario

 var isValid by remember { mutableStateOf(false) }

 // Estados individuales para cada campo de texto y su indicador de error

 var firstName by remember { mutableStateOf(TextFieldValue("")) }

 var firstNameError by remember { mutableStateOf(false) }

 var national by remember { mutableStateOf(TextFieldValue("")) }

```
var nationalError by remember { mutableStateOf(false) }

var lastName by remember { mutableStateOf(TextFieldValue("")) }

var lastNameError by remember { mutableStateOf(false) }

var title by remember { mutableStateOf(TextFieldValue("")) }

var titleError by remember { mutableStateOf(false) }

var personal by remember { mutableStateOf(TextFieldValue("")) }

var personalError by remember { mutableStateOf(false) }

var day by remember { mutableStateOf(TextFieldValue("")) }

var dayError by remember { mutableStateOf(false) }

var drivers by remember { mutableStateOf(TextFieldValue("")) }

var driversError by remember { mutableStateOf(false) }

var license by remember { mutableStateOf(TextFieldValue("")) }

var licenseError by remember { mutableStateOf(false) }

var address by remember { mutableStateOf(TextFieldValue("")) }

var addressError by remember { mutableStateOf(false) }

var country by remember { mutableStateOf(TextFieldValue("")) }

var countryError by remember { mutableStateOf(false) }

var tab by remember { mutableStateOf(TextFieldValue("")) }

var tabError by remember { mutableStateOf(false) }

var telep by remember { mutableStateOf(TextFieldValue("")) }
```

```
var telepError by remember { mutableStateOf(false) }

var percentage by remember { mutableStateOf(TextFieldValue("")) }

var percentageError by remember { mutableStateOf(false) }

var isPer by remember { mutableStateOf(TextFieldValue("")) }

var isPerError by remember { mutableStateOf(false) }

var radio by remember { mutableStateOf(TextFieldValue("")) }

var radioError by remember { mutableStateOf(false) }


// Estado para mostrar el selector de fecha

var showDate by remember { mutableStateOf(false) }


// Opciones seleccionadas para botones de radio

var selectedOption by remember { mutableStateOf("Opción 1") }

var selectedOption1 by remember { mutableStateOf("Opción 1") }


// Estado para el DatePicker

val state = rememberDatePickerState()


// Lista de identificadores de campos para iterar

val formFields = listOf(
```

"First",

"National",

"LastName",

"Title",

"Personal",

"Dob",

"Drivers",

"License",

"Take",

"AddressBo",

"Dao",

"Country",

"Postal",

"Telep",

"Email",

"Percentage",

"IsPer",

"Radio"

)

// Contenedor principal que despliega los elementos en una lista vertical desplazable

LazyColumn(

 modifier = Modifier

 .fillMaxSize()

 .padding(top = 40.dp, start = 16.dp, end = 16.dp, bottom = 50.dp),

 horizontalAlignment = Alignment.Start,

 verticalArrangement = Arrangement.Center

) {

 items(formFields) { field ->

 when (field) {

 // Sección de información principal

 "First" -> {

 Text(

 text = "Principal information",

 fontWeight = FontWeight.Bold,

 modifier = Modifier.padding(bottom = 8.dp)

)

 Text(

 text = "Please list all principals who...",

```
        modifier = Modifier.padding(bottom = 8.dp)

    )

    // Campo de texto: Nombre

    Text(text = "First Name *")

    TextField(

        value = firstName,

        onChange = {

            firstName = it

            firstNameError = it.text.isBlank()

            isValid = !firstNameError

        },

        isError = firstNameError,

        keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Text),

        modifier = Modifier.fillMaxWidth()

    )

    if (firstNameError) {

        Text(

            text = "The name is required",

            color = MaterialTheme.colorScheme.error,
```

```
        style = MaterialTheme.typography.bodySmall

    )

}

}

// Apellido

"LastName" -> {

    Text(text = "Last Name *")

    TextField(

        value = lastName,

        onChange = {

            lastName = it

            lastNameError = it.text.isBlank()

            isValid = !firstNameError && !lastNameError

        },

        isError = lastNameError,

        keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Text),

        modifier = Modifier.fillMaxWidth()

    )

    if (lastNameError) {
```

```
Text(  
  
    text = "The name is required",  
  
    color = MaterialTheme.colorScheme.error,  
  
    style = MaterialTheme.typography.bodySmall  
  
)  
  
}  
  
}  
  
  
// Nacionalidad  
  
"National" -> {  
  
    Text(text = "Nationality *")  
  
    TextField(  
  
        value = national,  
  
        onChange = {  
  
            national = it  
  
            nationalError = it.text.isBlank()  
  
            isValid = !firstNameError && !lastNameError && !nationalError  
  
        },  
  
        isError = nationalError,  
  
        keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Text),  

```

```
        modifier = Modifier.fillMaxWidth()

    )

    if (nationalError) {

        Text(

            text = "The nationality is required",

            color = MaterialTheme.colorScheme.error,

            style = MaterialTheme.typography.bodySmall

        )

    }

}

// Título

"Title" -> {

    Text(text = "Title *")

    TextField(

        value = title,

        onChange = {

            title = it

            titleError = it.text.isBlank()
```

```
isFormValid = !firstNameError && !lastNameError && !nationalError &&
!titleError

},

isError = titleError,

keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Text),

modifier = Modifier.fillMaxWidth()

)

if (titleError) {

    Text(

        text = "The title is required",

        color = MaterialTheme.colorScheme.error,

        style = MaterialTheme.typography.bodySmall

    )

}

}

// Información personal: SSN

"Personal" -> {
```

```
Text(  
  
    text = "Personal information",  
  
    fontWeight = FontWeight.Bold,  
  
    modifier = Modifier.padding(bottom = 8.dp)  
  
)  
  
Text(text = "Social Security Number **")  
  
TextField(  
  
    value = personal,  
  
    onChange = {  
  
        personal = it  
  
        personalError = it.text.isBlank()  
  
        isValid = personalError.not()  
  
    },  
  
    label = { Text("XXX-XX-") },  
  
    isError = personalError,  
  
    keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number),  
  
    modifier = Modifier.fillMaxWidth()  
  
)  
  
if (personalError) {
```

```
Text(  
  
    text = "The SSN is required",  
  
    color = MaterialTheme.colorScheme.error,  
  
    style = MaterialTheme.typography.bodySmall  
  
)  
  
}  
  
}
```

// Fecha de nacimiento con DatePickerDialog

```
"Dob" -> {  
  
    Text(text = "Day of Birth *")  
  
    TextField(  
  
        value = day,  
  
        onChange = {  
  
            day = it  
  
            dayError = it.text.isBlank()  
  
            isFormValid = dayError.not()  
  
        },  
  
        isError = dayError,
```

```
keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number),

modifier = Modifier.fillMaxWidth()

)

if (dayError) {

    Text(

        text = "The birth date is required",

        color = MaterialTheme.colorScheme.error,

        style = MaterialTheme.typography.bodySmall

    )

}

}
```

```
// Otros campos omitted for brevity...
```

```
// Botón de envío al final del formulario
```

```
"Radio" -> {

    Text(

        text = "All fields marked with (*) are required",

        modifier = Modifier.padding(top = 12.dp, bottom = 8.dp)
```

```
)  
  
Button(  
  
    onClick = { navController.popBackStack() },  
  
    modifier = Modifier.fillMaxWidth()  
  
){  
  
    Text(text = "Submit")  
  
}  
  
}  
  
}  
  
Spacer(modifier = Modifier.height(16.dp))  
  
}  
  
}  
  
}
```

8.- Estructura del proyecto e interfaz grafica

Figura 9

Ilustración 9, Captura de pantalla, estructura del proyecto, 10 de enero de 2024

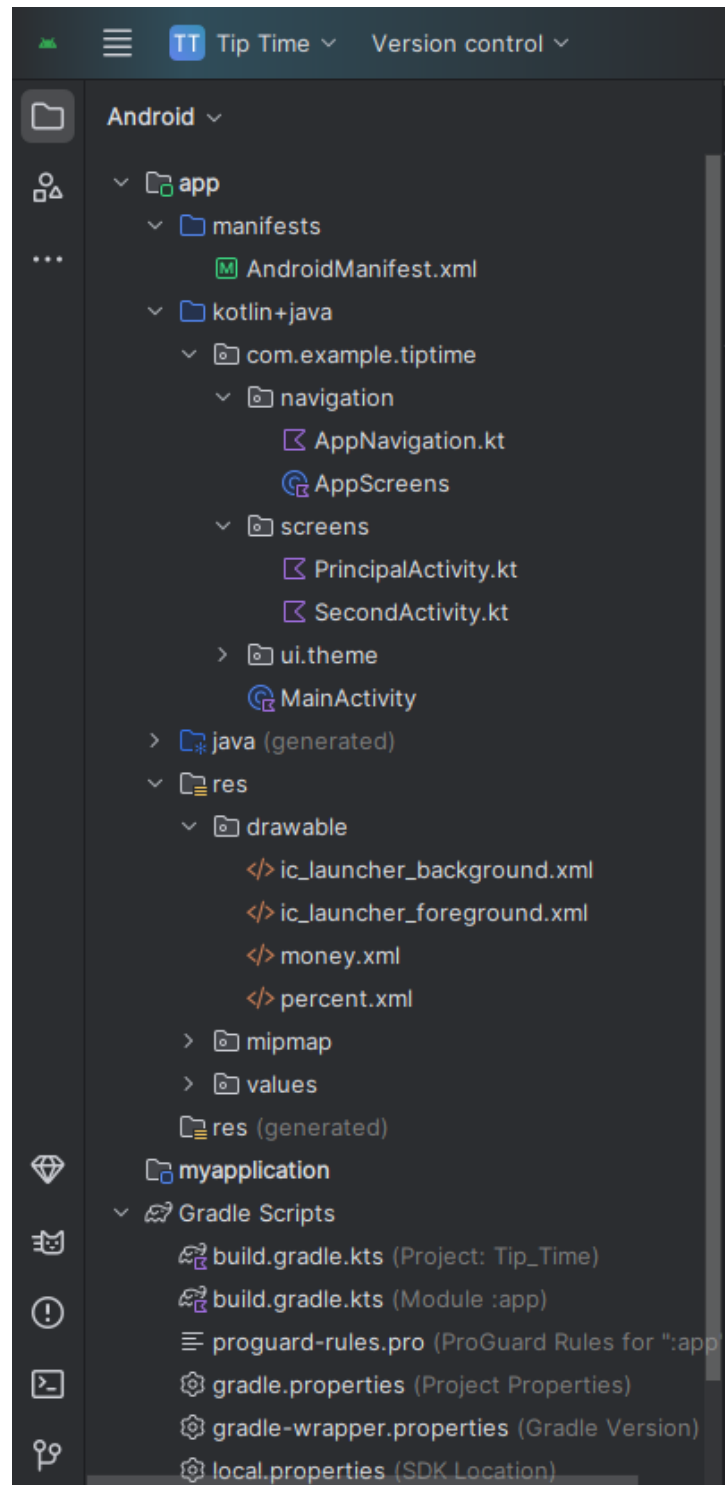


Figura 10

Ilustración 10, Captura de pantalla, Home UI, 10 de enero de 2024

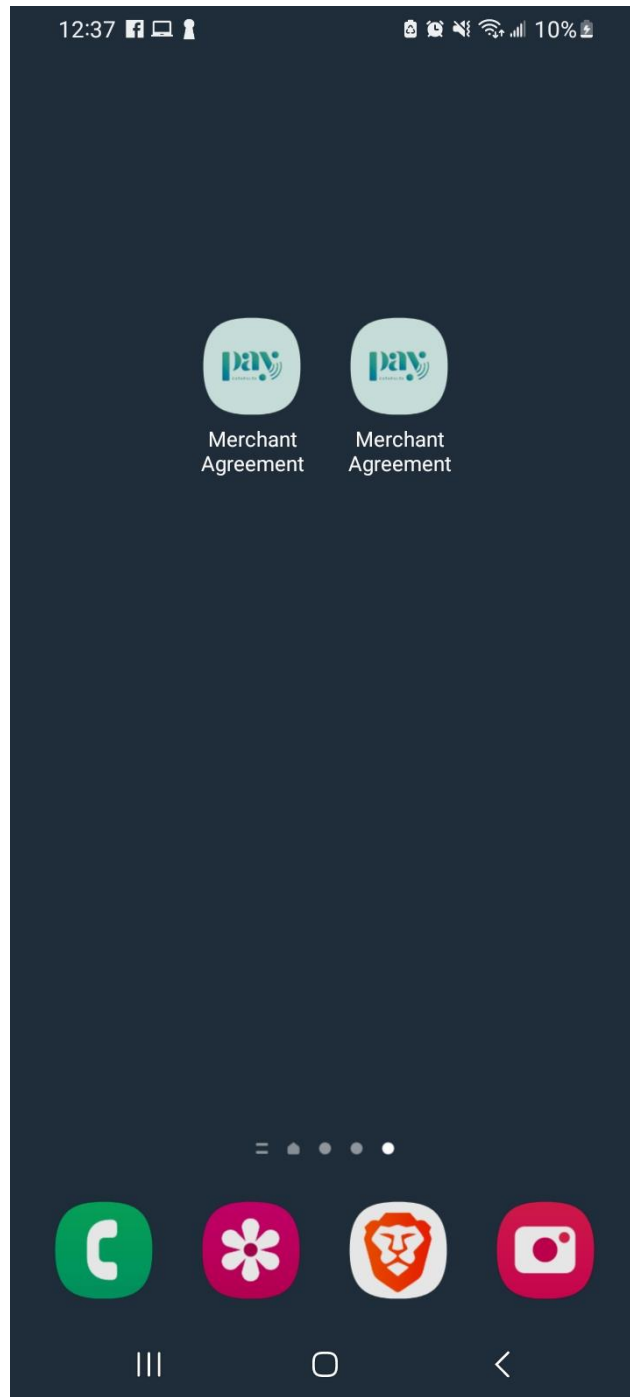


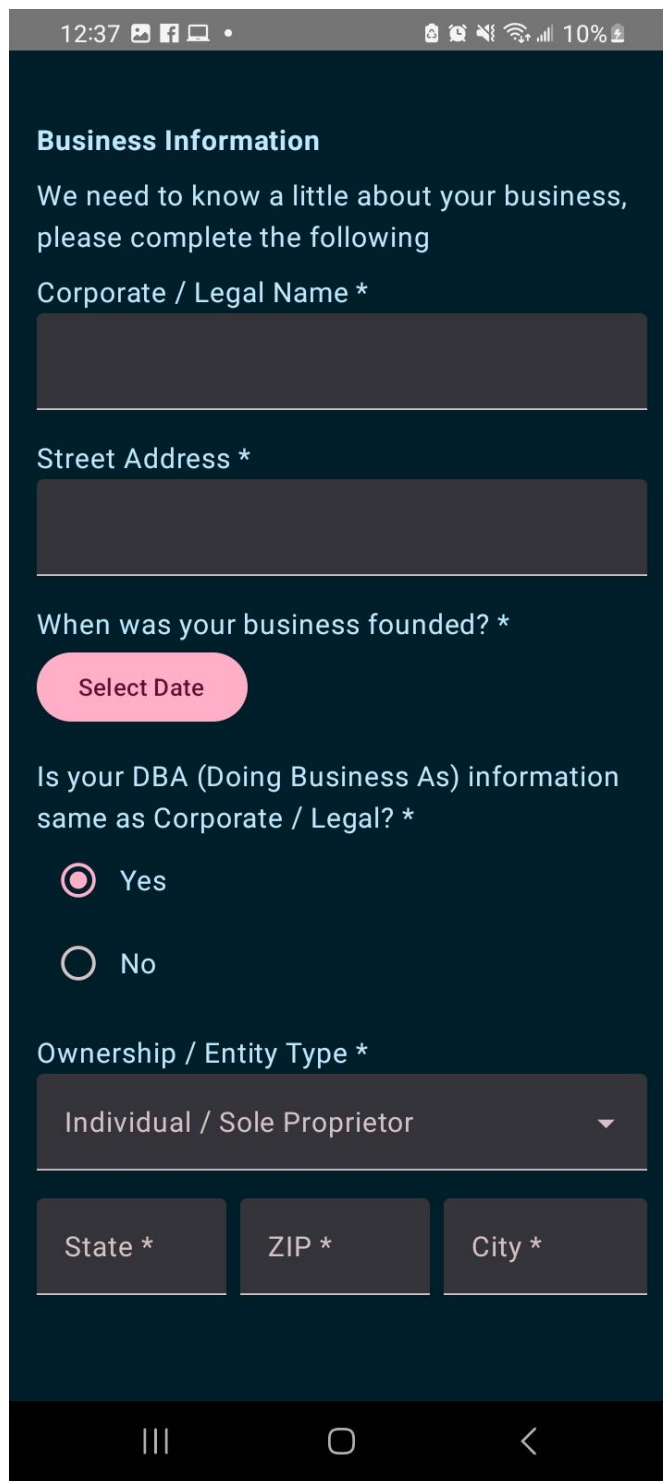
Figura 11

Ilustración 11, Captura de pantalla, Home UI, 10 de enero de 2024



Figura 12

Ilustración 12, Captura de pantalla, página principal, 10 de enero de 2024



12:37 [Icons] 10%

Business Information

We need to know a little about your business, please complete the following

Corporate / Legal Name *

Street Address *

When was your business founded? *

Select Date

Is your DBA (Doing Business As) information same as Corporate / Legal? *

☒ Yes

☐ No

Ownership / Entity Type *

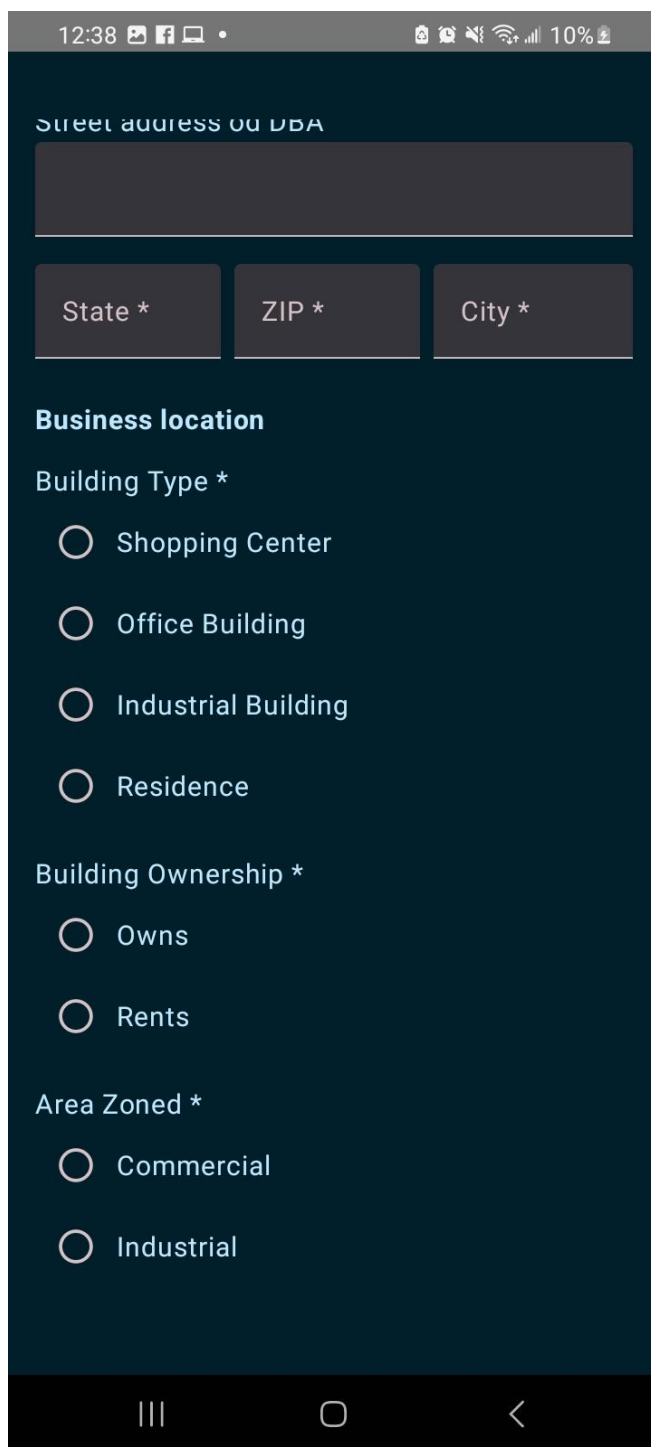
Individual / Sole Proprietor ▼

State * ZIP * City *

[Icons]

Figura 13

Ilustración 13, Captura de pantalla, SelecOneMenu, 10 de enero de 2024



12:38 [Icons] 10%

Street address or DBA

[Text Input Field]

State * ZIP * City *

Business location

Building Type *

- ☐ Shopping Center
- ☐ Office Building
- ☐ Industrial Building
- ☐ Residence

Building Ownership *

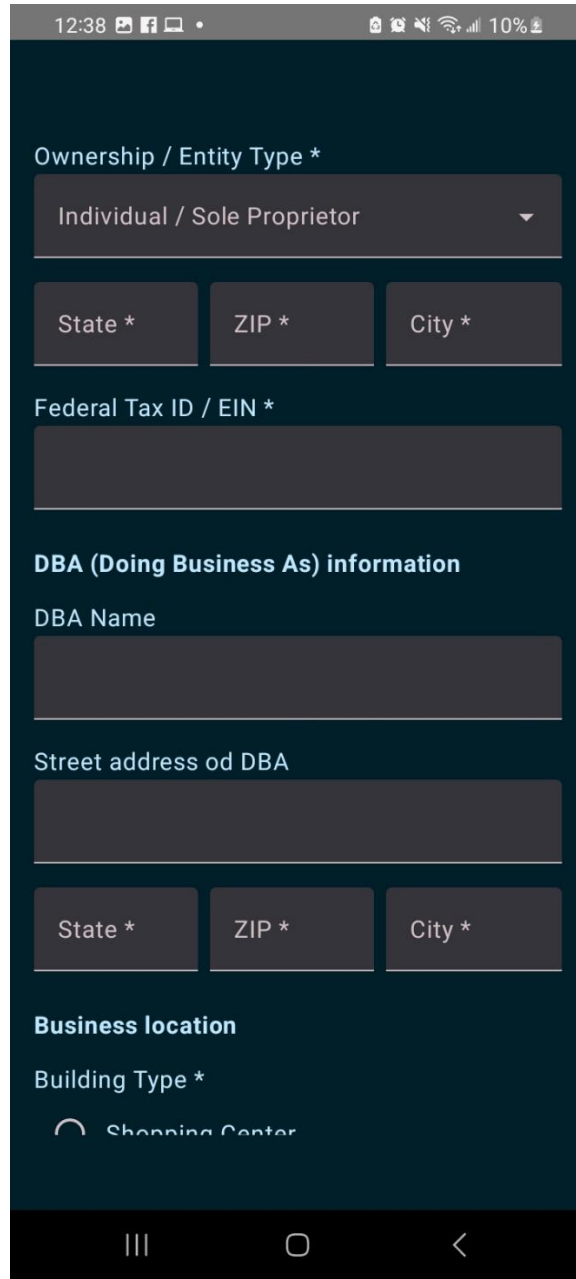
- ☐ Owns
- ☐ Rents

Area Zoned *

- ☐ Commercial
- ☐ Industrial

Figura 14

Ilustración 14, Captura de pantalla, información del negocio, 10 de enero de 2024



12:38 [Icons] 10%

Ownership / Entity Type *

Individual / Sole Proprietor

State * ZIP * City *

Federal Tax ID / EIN *

DBA (Doing Business As) information

DBA Name

Street address od DBA

State * ZIP * City *

Business location

Building Type *

☐ Shopping Center

Figura 15

Ilustración 15, Captura de pantalla, telefono/dirección, 10 de enero de 2024

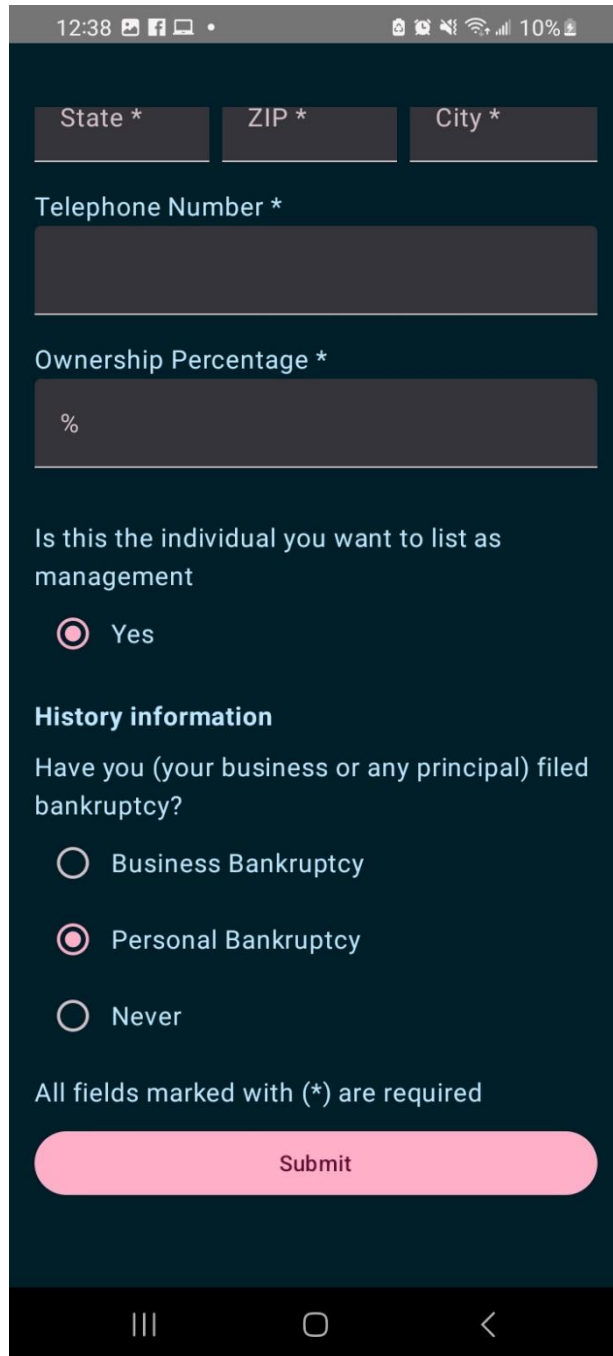
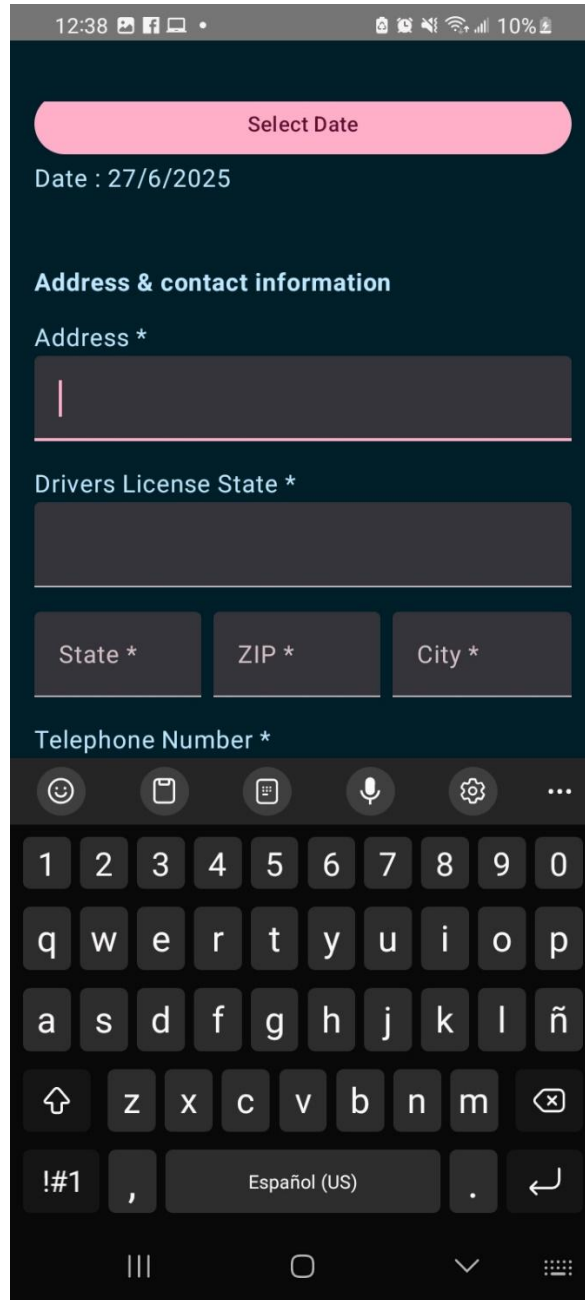


Figura 16

Ilustración 16, Captura de pantalla, dirección email, 10 de enero de 2024



12:38 [Icons] 10%

Select Date

Date : 27/6/2025

Address & contact information

Address *

Drivers License State *

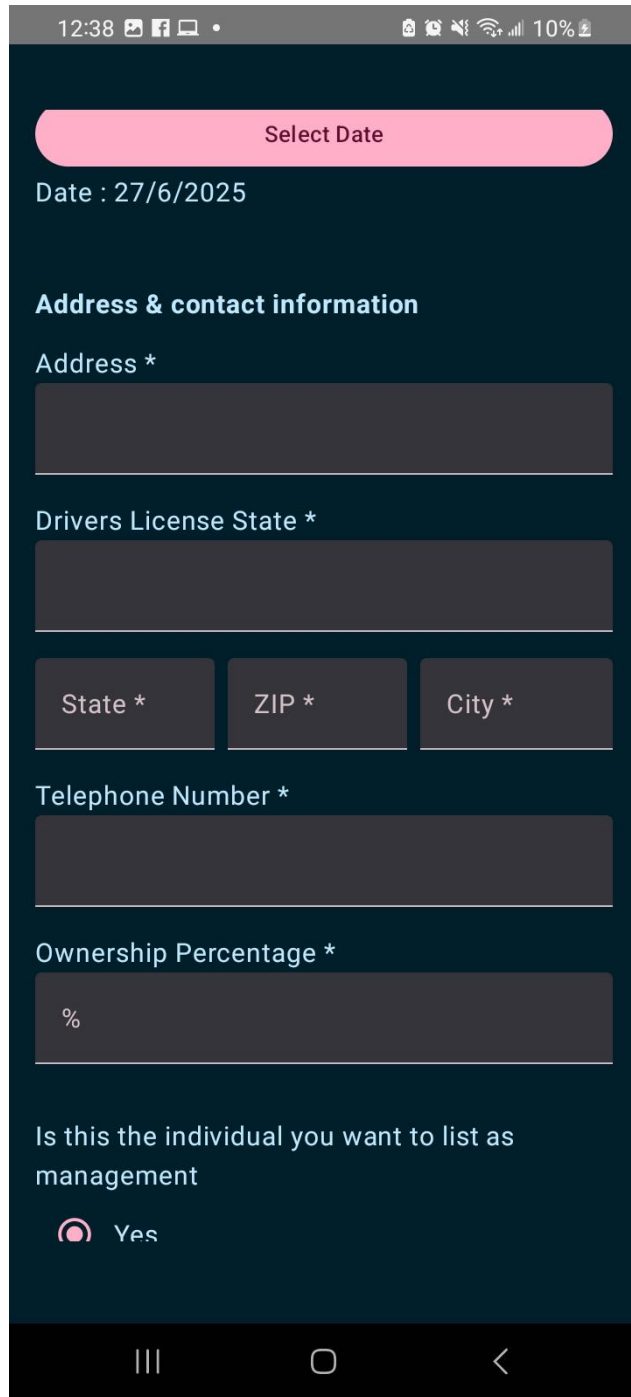
State * ZIP * City *

Telephone Number *

[Keyboard: Spanish (US)]

Figura 17

Ilustración 17, Captura de pantalla, fecha, 10 de enero de 2024



12:38 [Icons] 10%

Select Date

Date : 27/6/2025

Address & contact information

Address *

Drivers License State *

State * ZIP * City *

Telephone Number *

Ownership Percentage *

%

Is this the individual you want to list as management

☒ Yes

Figura 18

Ilustración 18, Captura de pantalla, selección de fecha, 10 de enero de 2024

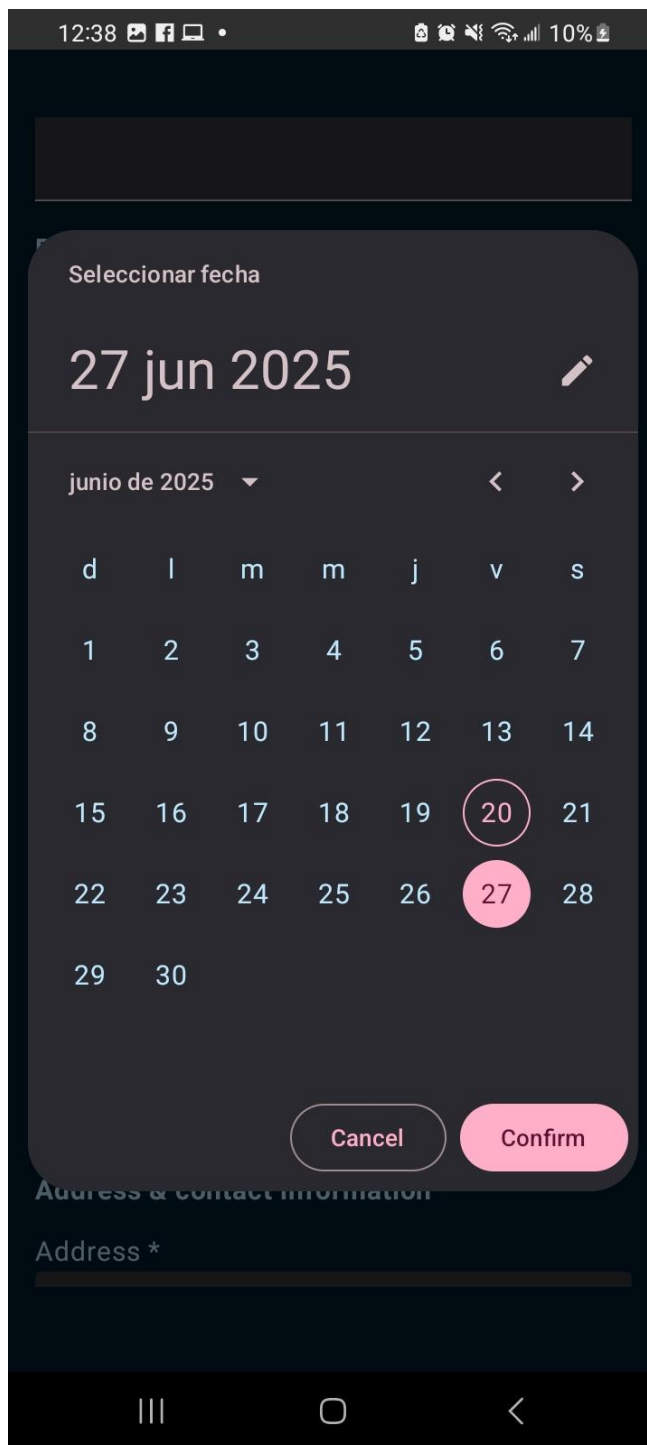
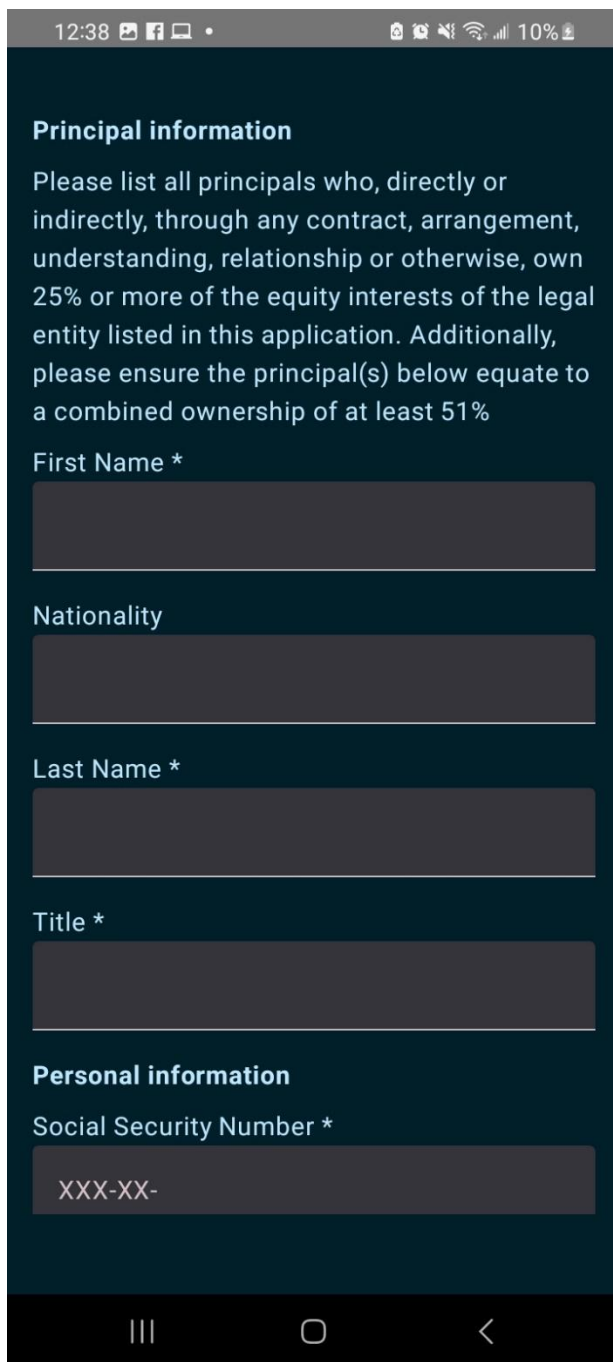


Figura 19

Ilustración 19, Captura de pantalla, información personal segunda página, 10 de enero de 2024



12:38 [Icons] 10%

Principal information

Please list all principals who, directly or indirectly, through any contract, arrangement, understanding, relationship or otherwise, own 25% or more of the equity interests of the legal entity listed in this application. Additionally, please ensure the principal(s) below equate to a combined ownership of at least 51%

First Name *

Nationality

Last Name *

Title *

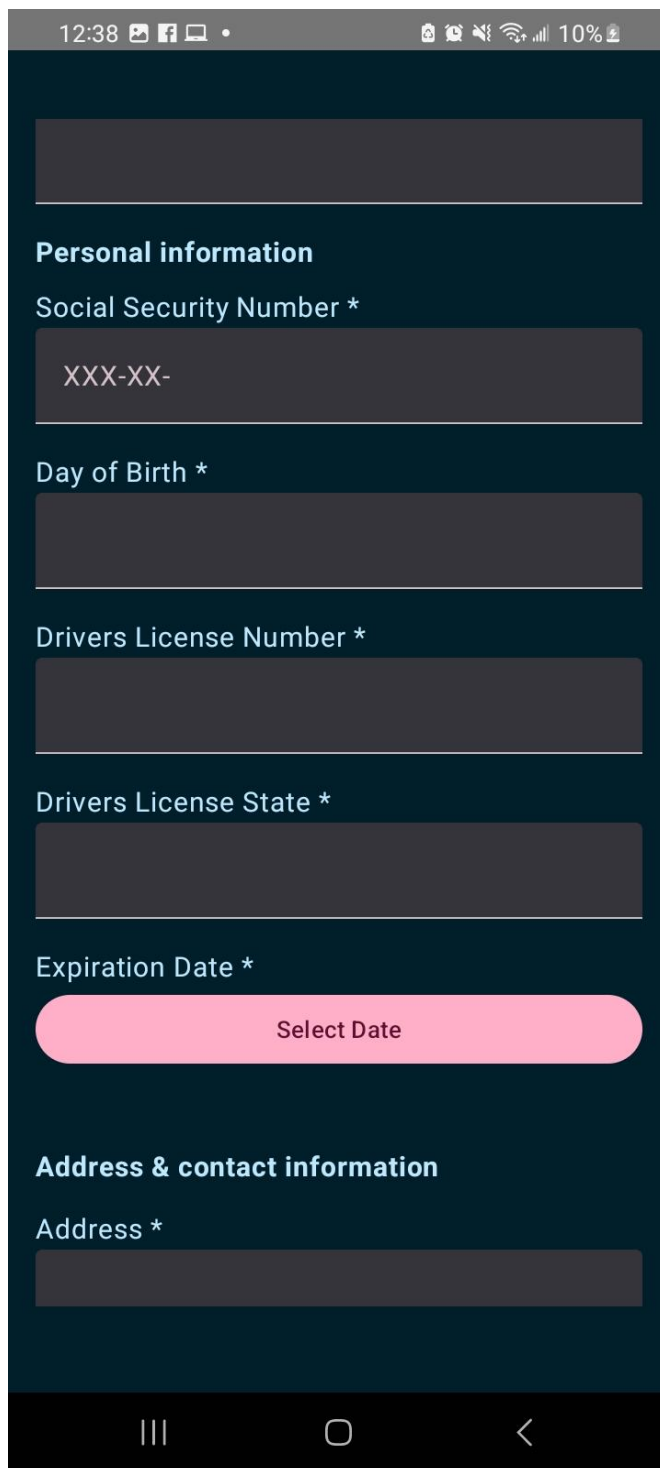
Personal information

Social Security Number *

XXX-XX-

Figura 20

Ilustración 20, Captura de pantalla, recolección de datos, 10 de enero de 2024



12:38

Personal information

Social Security Number *

XXX-XX-

Day of Birth *

Drivers License Number *

Drivers License State *

Expiration Date *

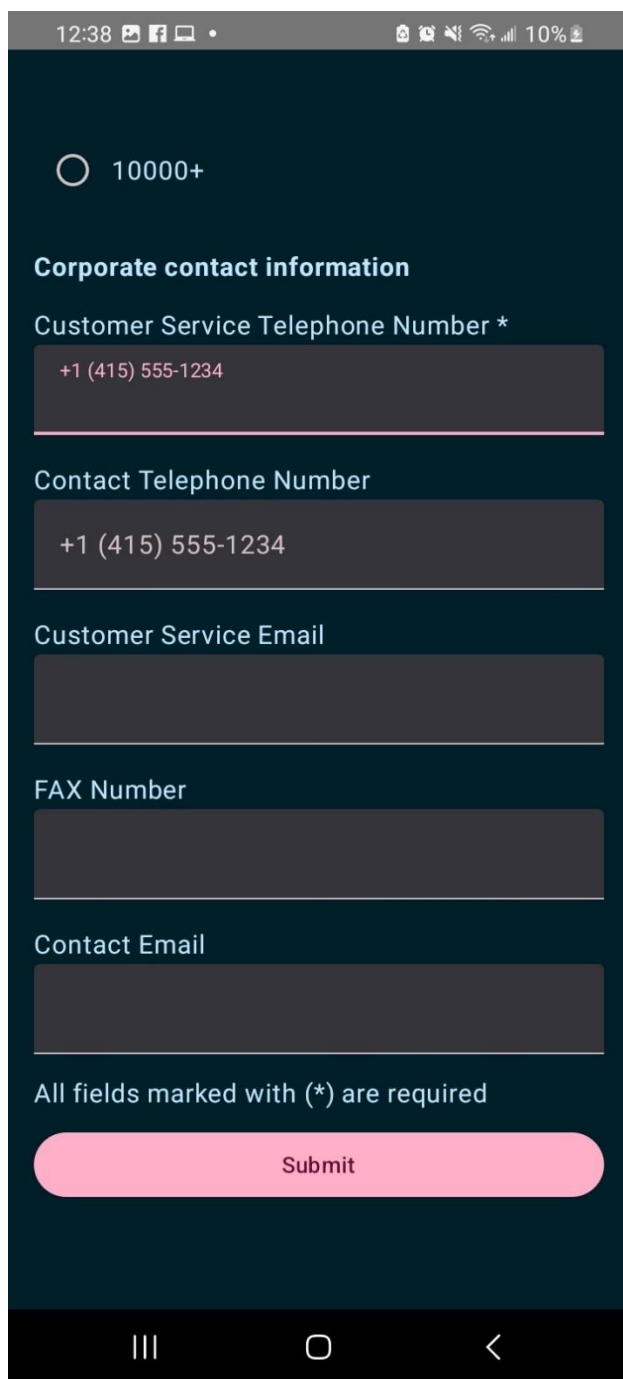
Select Date

Address & contact information

Address *

Figura 21

Ilustración 21, Captura de pantalla, contacto del proveedor, 10 de enero de 2024



12:38 [Icons] 10%

☐ 10000+

Corporate contact information

Customer Service Telephone Number *

+1 (415) 555-1234

Contact Telephone Number

+1 (415) 555-1234

Customer Service Email

FAX Number

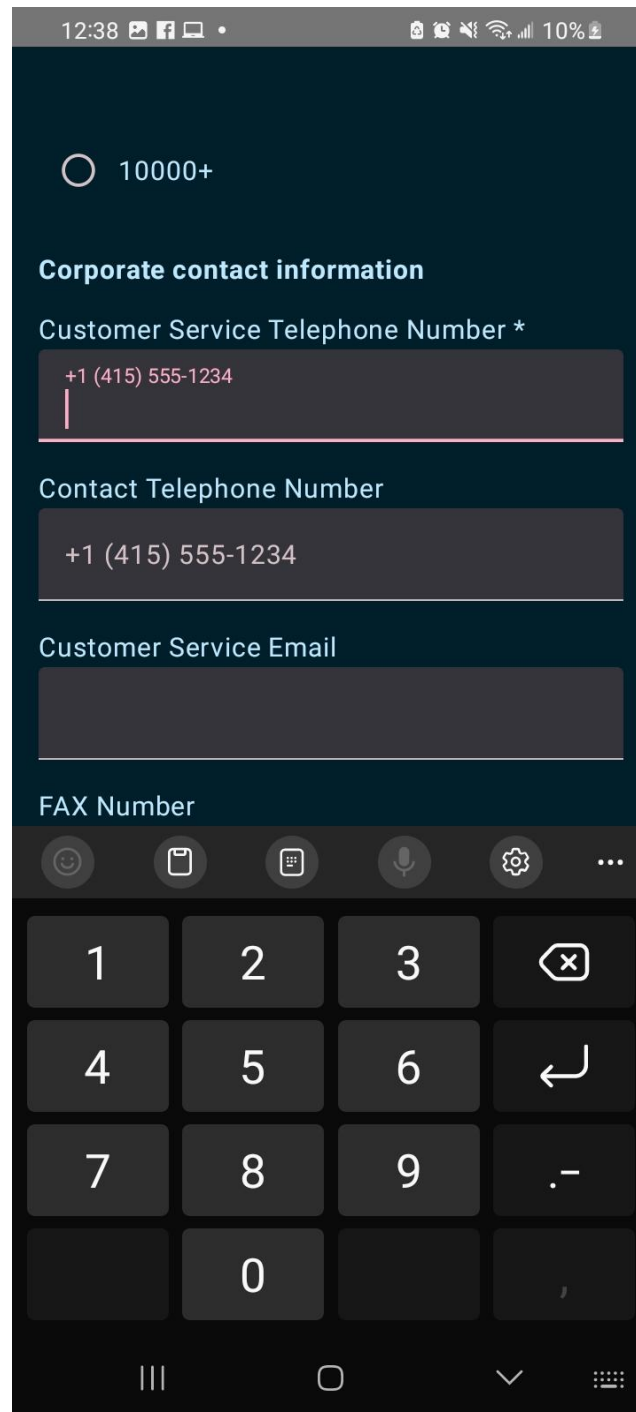
Contact Email

All fields marked with (*) are required

Submit

Figura 21

Ilustración 22, Captura de pantalla, formato numérico, 10 de enero de 2024



12:38 [status icons] 10%

☐ 10000+

Corporate contact information

Customer Service Telephone Number *

+1 (415) 555-1234

Contact Telephone Number

+1 (415) 555-1234

Customer Service Email

FAX Number

[navigation icons]

1 2 3 [back]

4 5 6 [undo]

7 8 9 [decimal]

[space] 0 [comma]

[bottom navigation icons]

9.- Conclusión del análisis de vulnerabilidades y arquitectura del software

La lista de elementos visualizada en tu imagen representa un conjunto exhaustivo de vectores de ataque y patrones comunes de vulnerabilidad que afectan a aplicaciones web y de escritorio. Entre ellos destacan técnicas como la inyección de código (SQL, comandos, scripts), buffer overflows, fallos en la manipulación de rutas y nombres de archivos, así como problemas de codificación y escape de caracteres, todos los cuales representan riesgos críticos para la seguridad de una aplicación.

Dentro del contexto del proyecto (una app móvil de punto de venta desarrollada en Kotlin), esta lista cobra especial importancia. Dado que el sistema maneja datos sensibles de usuarios y clientes, e incluso potencialmente transacciones financieras y documentos legales, una arquitectura segura y robusta no es opcional, sino indispensable.

La aplicación debe incluir mecanismos como:

- Validación estricta de entradas (contra inyecciones y datos maliciosos)
 - Codificación adecuada en interfaces web y API
 - Cifrado de datos sensibles en tránsito y en reposo (TLS, AES)
 - Controles de acceso y autenticación seguros
 - Pruebas de penetración y uso de herramientas de análisis estático
-

10.- Referencias

Android Studio (5 de diciembre de 2025). *Introducción a Android Studio*.

https://developer.android.com/studio/intro?hl=es-419#project_structure.

Android Studio (5 de noviembre de 2025). *Cómo comenzar a usar Jetpack Compose*

<https://developer.android.com/develop/ui/compose/documentation?hl=es-419>

Catapulta Pay (10 diciembre 2024) *Tu proximo procesador de pago*

<https://catapultapay.com/>

Robert C. Martin (12 de septiembre de 2017) *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall (una imprenta de Pearson)

Luiz Di Bella (5 de agosto de 2021) *¡Diez años de Kotlin!*.

<https://blog.jetbrains.com/es/kotlin/2021/08/ten-years-of-kotlin/>

Kotlin by JetBrains (5 de agosto de 2021) *Creating The Best Programming Language:*

The Story of Kotlin. <https://www.youtube.com/watch?v=uE-1oF9PyiY>

Katrina Walker (25 de febrero de 2025) *What exactly is Kotlin?*. [https://codeop.tech/what-exactly-is-](https://codeop.tech/what-exactly-is-kotlin/#:~:text=Like%20a%20spiced%20Dup%20Java,macOS%20and%20iOS%20platforms%20too)

[kotlin/#:~:text=Like%20a%20spiced%20Dup%20Java,macOS%20and%20iOS%20platforms%20too](https://codeop.tech/what-exactly-is-kotlin/#:~:text=Like%20a%20spiced%20Dup%20Java,macOS%20and%20iOS%20platforms%20too).

NetMentor (27 de febrero de 2024) *Explicación al patrón o arquitectura MVC | Modelo*

vista controlador. <https://www.youtube.com/watch?v=RjAK93TD6dc&t=4s>

Steve Smith (17 de junio de 2024) *Overview of ASP.NET Core MVC.*

<https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-10.0>

Precognis (10 de agosto de 2022) *Modelo Vista Controlador.*

<https://es.linkedin.com/pulse/modelo-vista-controlador-precognis>

Android Studio (2 de marzo de 2024) *Developers.*

<https://developer.android.com/studio?hl=es-419>