EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO SUPERIOR DE GUASAVE

T E S I S

ANÁLISIS DE LA OPTIMIZACIÓN DEL CALCULO DE LA LAMINA DE RIEGO.

PRESENTADA POR

GONZÁLEZ AVENDAÑO ELDAR DAEL

COMO REQUISITO PARA LA OBTENCIÓN DEL GRADO DE

INGENIERÍA EN SISTEMAS COMPUTACIONALES

DIRECTOR(A) DE TESIS

M.C. ARCE CARDENAS FRANCISCO JAVIER

Instituto Tecnológico Superior de Guasave

CARTA DE LIBERACIÓN DE TESIS

DEDICATORIA

Dedico este trabajo a Dios, por permitirme llegar hasta este momento y darme la oportunidad de formarme en el Instituto Tecnológico Superior de Guasave, mi alma máter, donde recibí las herramientas y conocimientos necesarios para mi desarrollo profesional.

A los pilares de mi vida: mi madre, mi abuela, mis hermanos y mi novio. Gracias por ser maravillosos, por creer en mí cuando más lo necesitaba y por brindarme su cariño, comprensión y apoyo incondicional. Ustedes me han enseñado a valorar lo que tengo y han fomentado en mí el deseo constante de superación.

A mis compañeros y amigos de la universidad, con quienes he compartido momentos trascendentales que guardaré siempre en mi memoria.

AGRADECIMIENTOS

Deseo expresar mi más sincero agradecimiento a todas las personas e instituciones que contribuyeron al desarrollo de este trabajo.

En primer lugar, agradezco el apoyo y la orientación brindados por quienes aportaron sus conocimientos y experiencia durante el proceso de investigación y desarrollo. Su guía fue fundamental para la consolidación de este proyecto.

Asimismo, extiendo mi gratitud a quienes colaboraron con sugerencias, retroalimentación y aportes técnicos, enriqueciendo significativamente el resultado final.

Agradezco también el apoyo incondicional de quienes, de distintas maneras, brindaron su comprensión y motivación a lo largo de este proceso, permitiéndome avanzar con determinación.

Finalmente, reconozco el uso de materiales previamente publicados con derechos reservados, obtenidos mediante los permisos correspondientes, los cuales han sido fundamentales para la elaboración de este trabajo.

RESUMEN

El presente proyecto de investigación se enfocó en el desarrollo de una aplicación web que implementa algoritmos y fórmulas agronómicas para el cálculo de la lámina de riego. El objetivo principal fue optimizar el uso eficiente del recurso hídrico en la agricultura, proporcionando una herramienta accesible para la toma de decisiones informadas y sostenibles.

Metodológicamente, se inició con un análisis exhaustivo de requerimientos, definiendo variables agronómicas (cultivo, suelo, etapa fenológica) y climáticas. El desarrollo se fundamentó en tecnologías modernas como Next.js, React y TypeScript, gestionando el flujo de trabajo mediante metodología Kanban. La base de datos se estructuró en Supabase (PostgreSQL) con políticas de seguridad RLS. La lógica de negocio incluyó el cálculo de la Evapotranspiración de Referencia (ETo) mediante los métodos Hargreaves-Samani y Penman-Monteith (FAO-56).

Como parte de la validación, se realizó un estudio comparativo de eficiencia operativa, contrastando el uso de la aplicación frente al cálculo manual; los resultados evidenciaron una reducción significativa en el tiempo de procesamiento, validando la agilidad de la herramienta.

Se concluye que el sistema ofrece una arquitectura robusta y escalable que integra efectivamente la ciencia agronómica con la tecnología web. Se recomienda para trabajos futuros la validación de campo y la implementación de algoritmos de optimización hídrica avanzada.

Palabras clave: Gestión hídrica, Aplicación web, Evapotranspiración, Next.js, Agricultura de precisión.

ABSTRACT

This research project focused on the development of a web application that implements agronomic algorithms and formulas for calculating irrigation depth. The main objective was to optimize the efficient use of water resources in agriculture, providing an accessible tool for informed and sustainable decision-making.

Methodologically, the project began with an exhaustive requirements analysis, defining agronomic (crop, soil, phenological stage) and climatic variables. The development was based on modern technologies such as Next.js, React, and TypeScript, managing the workflow through the Kanban methodology. The database was structured in Supabase (PostgreSQL) with RLS security policies. The business logic included the calculation of Reference Evapotranspiration (ETo) using the Hargreaves-Samani and Penman-Monteith (FAO-56) methods.

As part of the validation process, a comparative study of operational efficiency was conducted, contrasting the use of the application against manual calculation; the results demonstrated a significant reduction in processing time, validating the tool's agility.

It is concluded that the system offers a robust and scalable architecture that effectively integrates agronomic science with web technology. Recommendations for future work include field validation and the implementation of advanced water optimization algorithms.

Keywords: Water management, Web application, Evapotranspiration, Next.js, Precision agriculture.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

En el presente documento se describe de manera detallada el desarrollo del proyecto de residencia profesional denominado "Aplicación web para el cálculo de la lámina de riego". El objetivo central de esta investigación tecnológica es la creación de una herramienta web que implemente algoritmos y fórmulas agronómicas validadas para calcular la lámina de riego de manera precisa.

El enfoque principal del proyecto surge como respuesta a la problemática del uso empírico e ineficiente del recurso hídrico en la agricultura. Actualmente, la falta de herramientas accesibles provoca que productores, técnicos y estudiantes tomen decisiones de riego basadas en la intuición más que en datos. Por ello, esta tesis presenta una solución tecnológica fundamentada teóricamente en la integración de modelos agronómicos esenciales, tales como Penman-Monteith y Hargreaves-Samani, los cuales son estándares internacionales para la estimación de la demanda hídrica (ETo y Kc).

A lo largo de los capítulos, se exponen los procedimientos realizados abarcando el ciclo completo de desarrollo de software: desde la etapa de análisis de requerimientos y el diseño de la arquitectura tecnológica, hasta la implementación de la lógica de negocio y el despliegue final del sistema. Se detalla el uso de métodos y herramientas modernas como Next.js, Supabase y Vercel, así como la integración de fuentes de datos climáticos mediante ingreso manual o consumo de APIs externas.

Como parte fundamental de la validación de la herramienta, se incluye en este documento un estudio comparativo de eficiencia, en el cual se analizó el ahorro de tiempo obtenido por los usuarios al utilizar la aplicación en contraste con el método de cálculo manual. Este análisis evidencia no solo la viabilidad técnica, sino también la optimización operativa que la herramienta aporta al proceso de planificación agrícola.

Finalmente, se presentan los resultados funcionales, las conclusiones y las recomendaciones derivadas de la experiencia, evidenciando cómo el proyecto contribuyó al fortalecimiento de habilidades profesionales en la implementación de arquitecturas web escalables y en el manejo de lógica de negocio agropecuaria, sentando las bases

para futuras investigaciones destinadas a validar o replicar la solución en otros ámbitos productivos.

I. PLANTEAMIENTO DEL PROBLEMA.

En diversas regiones agrícolas, especialmente en zonas rurales, se observa que el riego se realiza de forma empírica, sin el apoyo de herramientas técnicas que consideren factores clave como el tipo de cultivo, su etapa fenológica, las características del suelo y las condiciones climáticas. Esta situación genera un uso ineficiente del agua, afecta la productividad de los cultivos y contribuye al deterioro de los recursos naturales. Se identifica la necesidad de contar con una herramienta tecnológica accesible que automatice el cálculo de la lámina de riego, permitiendo a productores, estudiantes y técnicos del sector tomar decisiones más informadas y sostenibles. Por ello, se plantea el desarrollo de una aplicación web que integre estas variables y proporcione resultados confiables y fáciles de interpretar, con el fin de mejorar la gestión del riego en el ámbito agrícola.

II. JUSTIFICACIÓN.

El desarrollo de una aplicación web para el cálculo de la lámina de riego surge como una respuesta tecnológica y práctica a la necesidad de optimizar el uso del recurso hídrico en la agricultura, sector que representa uno de los principales consumidores de agua a nivel mundial. Se observa que, en muchas regiones, el riego se realiza de forma empírica, sin considerar factores técnicos como la etapa fenológica del cultivo, el tipo de suelo o las condiciones meteorológicas, lo cual genera un uso ineficiente del agua, afectando la productividad, los costos y el medio ambiente. Desde el punto de vista social, se reconoce que muchos productores, especialmente en zonas rurales, carecen de herramientas accesibles para tomar decisiones informadas sobre riego. Esta aplicación permitirá a técnicos, estudiantes y agricultores contar con una plataforma digital gratuita o de bajo costo que les facilite aplicar conocimientos agronómicos en su práctica diaria, promoviendo una agricultura más técnica y equitativa. En términos de impacto tecnológico, el proyecto integra algoritmos agronómicos con tecnologías web, permitiendo la digitalización de procesos tradicionales del riego. Se fomenta así la transferencia de conocimiento técnico a medios digitales, impulsando el desarrollo de soluciones innovadoras para el sector agropecuario. El impacto económico se manifiesta en la posible reducción de costos de producción mediante un uso racional del agua, al evitar el riego en exceso o insuficiente. Además, la mejora en la eficiencia del riego puede traducirse en mayores rendimientos y mejor calidad de los cultivos. El impacto ambiental es significativo, ya que el uso eficiente del agua contribuye a la conservación de fuentes hídricas, la reducción de escorrentía y lixiviación de nutrientes, y la disminución del estrés hídrico en zonas vulnerables. Se promueve así una agricultura más sostenible y respetuosa con el entorno natural. Respecto a la viabilidad y factibilidad, se determinó que el proyecto es técnicamente viable al emplear herramientas y lenguajes de desarrollo web ampliamente utilizados. Asimismo, los modelos agronómicos necesarios están bien documentados y validados por la comunidad científica, lo que facilita su implementación computacional.

III. OBJETIVOS.

3.1 General.

DESARROLLAR Y ANÁLIZAR LA OPTIMIZACIÓN DEL CALCULO DE LA LAMINA DE RIEGO.

3.2 Específicos.

- Implementar algoritmos agronómicos para el cálculo de la lámina de riego, considerando parámetros como el tipo de cultivo, etapa fenológica, tipo de suelo y datos climáticos.
- Diseñar una interfaz web intuitiva que permita al usuario ingresar fácilmente la información necesaria para obtener recomendaciones de riego personalizadas.
- Integrar fuentes de datos meteorológicos (manual o por API) para obtener variables como evapotranspiración, temperatura y precipitación.
- Establecer una base de datos estructurada para almacenar la información de cultivos, suelos, usuarios y resultados de cálculo.
- Validar la precisión de los resultados mediante comparaciones con datos técnicos y literatura agronómica confiable.
- Optimizar la experiencia de usuario para que el sistema sea accesible a técnicos, productores y estudiantes del área agrícola. Hacer una comparativa de los tiempos de cálculo.

IV. MARCO TEÓRICO

4.1 Definiciones y terminología clave

En un estudio de manejo de riego, resulta esencial definir con precisión los conceptos básicos. A continuación, se presentan las definiciones fundamentales, sustentadas en la literatura técnica del artículo de la FAO (Hopper, 2006):

4.1.1 Definiciones y terminología clave

En un estudio de manejo de riego, resulta esencial definir con precisión los conceptos básicos. A continuación, se presentan las definiciones fundamentales, sustentadas en la literatura técnica del artículo de la FAO (Hopper, 2006):

•Lámina de riego: es el espesor de la capa de agua aplicada sobre el suelo durante un evento de riego, normalmente expresado en milímetros (mm). En otras palabras, la lámina de riego equivale al volumen de agua aplicado por unidad de superficie (1 mm $\simeq$ 1 litro/m^2). Por ejemplo, una lámina de 1 mm significa que 10 000 L de agua caen sobre 1 ha (10 000 m^2). Esta medida es útil para cuantificar el volumen total aplicado y facilita la conversión a caudales y tiempos de riego.

•Evapotranspiración (ET): es la pérdida total de agua hacia la atmósfera desde el suelo y la vegetación. Comprende dos procesos simultáneos: la evaporación directa del agua del suelo y la transpiración de las plantas a través de sus estomas. Bajo condiciones ideales de humedad y sin estrés hídrico, la evapotranspiración potencial del cultivo (ETc) se calcula como el producto de la evapotranspiración de referencia (ETo) y el coeficiente de cultivo (Kc).

•Evapotranspiración de referencia (ETo): La ETo es la evapotranspiración de un cultivo de referencia ideal (césped verde corto de 0.12 m de altura, bien regado) bajo condiciones climáticas estándar. Se calcula a partir de variables climáticas (radiación, temperatura, humedad, viento) mediante la ecuación de Penman–Monteith

recomendada por la FAO. La ETo sirve como base para estimar la demanda de agua de cualquier cultivo específico, multiplicándola por su coeficiente de cultivo Kc.

•Coeficiente de cultivo (Kc): es un factor adimensional que relaciona la ETc de un cultivo con la ETo del cultivo de referencia. En la práctica, ETc = Kc · ETo. El valor de Kc integra las características del cultivo (altura, estructura de follaje, cobertura, albedo, etc.) y cambia a lo largo del ciclo de crecimiento del cultivo. Por tanto, Kc varía según la especie y la etapa fenológica (inicial, desarrollo, medio y final). Valores estándar de Kc para cultivos comunes y sus fases se han documentado en la literatura (Allen et al., 1998). Por ejemplo, en cultivos de maíz se emplean valores típicos de Kc ≈ 0.3 en la etapa inicial hasta ≈1.15–1.2 en plena vegetación, mientras que cultivos de tomate alcanzan Kc ≈ 1.15 en fase media y bajan a ~0.8 al cierre.

•Capacidad de campo (CC): contenido de agua que un suelo retiene después de una saturación completa seguida de drenaje libre (normalmente 1–2 días). Representa la cantidad máxima de humedad que el suelo retiene contra la gravedad. Los suelos con mayor contenido de finos (arcilla, limo) tienen mayor capacidad de campo (mayor porosidad total y más microporos que retienen agua) que suelos arenosos. Por ejemplo, suelos arenosos típicamente retienen alrededor de 10–15% en volumen, mientras que suelos arcillosos pueden alcanzar 35–40%.

•Punto de marchitez permanente (PMP): contenido de humedad del suelo por debajo del cual las plantas ya no pueden extraer agua y se marchitan de forma irreversible. Matemáticamente, es el contenido de agua al que las raíces no pueden succionar más agua del suelo. El PMP depende de la textura y tipo de suelo. En suelos de textura fina, el PMP es relativamente alto en porcentaje (ej. 22% en arcilla), mientras que en arenas es bajo (ej. 5%).

•Agua útil disponible (AU): diferencia entre el contenido de agua a capacidad de campo y en el PMP, dentro de la zona radical. Equivale al volumen de agua que puede extraer una planta entre estos dos extremos. En otras palabras, el agua aprovechable para el cultivo es el agua retenida en el suelo que se encuentra entre la CC y el PMP; se expresa en mm de agua en la zona radical. Esta agua disponible varía con la textura:

suelos limosos o arcillosos almacenan más agua útil (por ejemplo, ADT ~270 mm en 1 m de perfil con CC~0.34 y PMP~0.14) mientras que suelos arenosos la retienen menos (ADT ~60 mm en 1 m de perfil con CC~0.11 y PMP~0.05).

•Balance hídrico: es la “contabilidad” del agua en el sistema suelo–cultivo. Se fundamenta en la ecuación de balance: las entradas de agua (precipitación, riego, aporte de aguas subterráneas, escurrimiento lateral entrante, etc.) menos las salidas (evapotranspiración, percolación profunda, escorrentía superficial, etc.) igualan el cambio en almacenamiento del suelo. Esquemas de balance hídrico agrícola típicamente calculan el contenido de agua del suelo sumando las entradas y restando las salidas durante un período determinado

Cada uno de estos términos es clave para entender la relación agua–cultivo y sirve como base para las ecuaciones de riego, los cálculos de ET y el manejo agronómico. Por ejemplo, la lámina de riego se calcula a partir de la ETc (ya sea ETo·Kc o ETa ajustada por estrés), y la programación de riego implica comparar la demanda (necesidad hídrica) con el agua útil disponible en el suelo (ADT y AFA, definidas en 2.2.3). En lo sucesivo se explorarán estos conceptos en detalle según los aspectos señalados en los subtemas siguientes.

4.2 Fundamentos de la relación agua–cultivo

El agua es el factor crítico en la producción agrícola. Esta sección aborda cómo el agua se mueve en el ciclo hidrológico hasta llegar al suelo y ser utilizada por las plantas, así como la forma de cuantificar su demanda vs disponibilidad en el cultivo.

4.2.1 Ciclo hidrológico y disponibilidad de agua en el suelo

Según (Shaxson, 2005) el ciclo hidrológico describe el movimiento continuo del agua en sus diversas fases (líquida, sólida, gaseosa) entre océanos, atmósfera, suelos

y vegetación. El sol evapora agua de océanos, lagos y ríos; las plantas transpiran vapor mediante sus raíces y hojas; ese vapor se condensa en la atmósfera formando nubes que luego precipitan como lluvia o nieve. Al precipitar al suelo, el agua puede seguir varios caminos: infiltrarse al perfil del suelo, escurrirse como flujo superficial, acumularse en charcos o ser interceptada por la vegetación. Parte de la lluvia infiltrada rellena la humedad del suelo accesible a las raíces. Parte puede evaporarse nuevamente desde la superficie y parte, si hay exceso, profundiza hasta convertirse en agua subterránea. De esta forma, la precipitación que llega al suelo se divide en: (a) humedad del suelo (utilizable por plantas y evaporables), (b) escorrentía superficial (flujo hacia ríos) y (c) drenaje profundo (reabastecimiento de acuíferos).

El contenido de agua en el suelo determina su disponibilidad para las plantas. Una lluvia moderada primero llena los poros hasta capacidad de campo; luego la vegetación extrae agua (transpira) y el suelo sigue drenando ligeramente por gravedad. En un suelo seco, el agua remanente es fuertemente retenida en microporos y menos accesible para las raíces. A medida que las plantas consumen agua, el potencial hídrico del suelo disminuye (el agua queda más ligada). Eventualmente, al descender el contenido por debajo del punto crítico, el flujo de agua al cultivo se vuelve insuficiente y aparece el estrés hídrico.

En síntesis, la disponibilidad de agua en el suelo proviene de las precipitaciones (y/o riego) que se infiltran. Una parte de esa agua infiltra hasta la zona radicular, alimentando la transpiración del cultivo. El resto se evapora de la superficie o drena al subsuelo. La fracción de agua infiltrada que permanece en la zona activa del cultivo constituye el recurso hídrico del suelo. La eficiencia con que un cultivo utiliza este recurso depende del balance entre la demanda atmosférica (ETo) y la capacidad del suelo para suministrar agua (capacidad de campo, pendiente del gradiente de humedad, etc.). Por ende, comprender el ciclo hidrológico y la dinámica de infiltración– evaporación– transpiración es fundamental para estimar cuánta agua puede extraer un cultivo antes de necesitar

3.2.2. Balance hídrico del cultivo: componentes y ecuaciones básicas

El balance hídrico del cultivo aplica la contabilidad del agua en un período dado sobre la zona radical según como describe (Shaxson, 2005). Matemáticamente, puede expresarse así:

$$\Delta S = P + Ri - ETc - D - Q$$

donde ΔS es el cambio en almacenamiento de agua en el suelo, P la precipitación, R_i el riego aplicado, Etc. la evapotranspiración del cultivo (potencial o real), D la percolación profunda y Q la escorrentía superficial. En la práctica agrícola se considera que:

•Ingresos de agua: incluyen las lluvias (P), el riego (R_i), y aportes adicionales (por ejemplo, agua freática o escorrentía entrante de zonas altas).

•Egresos de agua: comprenden principalmente la evapotranspiración (Etc., suma de evaporación superficial y transpiración vegetal) y las pérdidas por drenaje o escurrimiento (D, Q).

Así, el balance hídrico funciona como un sistema contable: el agua que entra al suelo por lluvia/riego se usa o sale por evaporación, percolación, etc. Si las salidas exceden a las entradas, el balance es negativo (déficit) y el suelo pierde humedad; si las entradas son mayores, el balance es positivo (exceso de agua). En otras palabras, el balance hídrico del cultivo para un periodo t se puede escribir conceptualmente como:

Balance = (P + Ri + otros ingresos) – (ETc + D + escorrentía).

Un resultado negativo indica déficit de humedad (estrés hídrico) y un resultado positivo indica crédito hídrico.

Este balance se emplea para programar riegos: cuando el balance acumulado se hace fuertemente negativo (el agua disponible cae por debajo de cierto umbral), se programa una nueva lámina de riego para restaurar la humedad.

4.3 Necesidad hídrica del cultivo vs. agua útil en suelo

La necesidad hídrica del cultivo se refiere al volumen de agua que el cultivo requiere para desarrollarse sin restricción (Etc. bajo condiciones óptimas). Bajo riego, esto suele calcularse como Etc = Kc · ETo (o ajustada por estrés: ETc·Ks). En cambio, el agua útil en el suelo es la cantidad de agua disponible para las raíces.

En los perfiles de suelo irrigado se define la Agua Disponible Total (ADT) y el concepto de Agua Fácilmente Aprovechable (AFA). La ADT es el volumen total de agua entre capacidad de campo y PMP en la zona radical y se calcula como:

ADT = 1000 × ($\theta CC - \theta PMP$) × $Zr[mm]$, ADT = 1000 × ($\theta CC - \theta PMP$) × Zr [mm] , ADT = 1000 × ($\theta CC - \theta PMP$) × $Zr[mm]$,

donde $\theta CC \ y \ \theta PMP$ son los contenidos volumétricos de agua a capacidad de campo y punto de marchitez, respectivamente, y la profundidad radicular (m). Este ADT (en mm) es la lámina de agua total extraíble en principio por el cultivo. Sin embargo, no toda esa agua se extrae antes de que el cultivo experimente estrés. Se define entonces la Agua Fácilmente Aprovechable (AFA) como la fracción p de ADT que el cultivo puede extraer sin sufrir reducción de transpiración Matemáticamente,

$$AFA = p \cdot ADT[mm],\ AFA = p \cdot ADT \text{ [mm]},\ AFA = p \cdot ADT[mm],$$

donde p (0–1) es el coeficiente de agotamiento permisible. En la literatura agronómica se estima que p varía con la especie y clima: valores típicos rondan 0.4–0.6 para cultivos de ciclo normal, siendo menores (~0.3) en cultivos de raíces someras o en condiciones de alta evapotranspiración, y mayores (~0.7) en cultivos con raíces profundas y evapotranspiración moderada. Un valor convencional usado frecuentemente es p≈0.5 para condiciones medias.

En la práctica de riego, la necesidad de agua del cultivo (en mm) se compara con el agua disponible. Por ejemplo, al inicio del riego se considera que el suelo está lleno (a CC) y la planta en fase inicial; al calcular la ETc para los siguientes días se monitorea cuánto de la ADT/AFA queda. Si la extracción alcanza el umbral p·ADT, se programa nuevo riego. De este modo, la programación de riego suele basarse en el agotamiento de la AFA en base a lo que dijo (Shaxson, 2005).

3.3. Evapotranspiración y coeficientes asociados

La evapotranspiración de referencia y los coeficientes de cultivo son los principales parámetros para estimar la demanda hídrica de los cultivos. Esta sección define la ETo y revisa los métodos de cálculo, así como los factores que afectan los coeficientes Kc.

4.3.1 Evapotranspiración de referencia (ETo): definiciones y métodos de cálculo

La evapotranspiración de referencia (ETo) según (Trezza, 2008) es la evaporación de un cultivo de referencia idealizado (césped corto) bajo condiciones de suelo bien regado. Es una medida de la demanda atmosférica por agua y se calcula a partir de datos meteorológicos. El método estándar recomendado por la FAO para calcular ETo es la ecuación de Penman–Monteith (FAO-56), la cual integra variables climáticas: radiación neta, temperatura, humedad relativa y velocidad del viento. Esta ecuación combina el balance de energía con la mecánica de la capa límite, brindando una estimación precisa de ETo para muchas condiciones agroclimáticas. Como se indica en “el método FAO Penman–Monteith” es ahora el único método estándar recomendado para la definición y cálculo de la evapotranspiración de referencia”.

Sin embargo, en muchos contextos no se dispone de todos los datos necesarios (especialmente radiación y viento). Por ello se utilizan métodos empíricos alternativos, que requieren menos variables climáticas. Un ejemplo muy difundido es la ecuación de Hargreaves–Samani, que estima ETo solo con temperaturas diarias máxima y mínima y radiación extraterrestre (basada en latitud y día del año). Según la literatura, Hargreaves produce estimaciones razonables cuando faltan datos de humedad y viento. En el código de la aplicación a la que se refiere este marco teórico, se señala que la ETo se estima con la ecuación de Hargreaves usando parámetros climáticos disponibles.

Otros métodos empíricos incluyen: la ecuación de Priestley–Taylor, que usa radiación neta y temperatura (aplicable en zonas húmedas); el método de Thornthwaite,

que emplea solo temperatura media y factor de duración de días; y diversos índices basados en medidas de evaporación de tanques clase A o balances de humedad del suelo. Por ejemplo, la evaporación de un tanque evaporímetro de clase A puede convertirse en ETo aplicando un factor empírico. En definitiva, los métodos alternativos son útiles cuando faltan datos completos, aunque su precisión varía con la región.

4.3.2 Coeficiente de cultivo (Kc): variación por especie y etapa fenológica

El coeficiente de cultivo (Kc) relaciona la ETc del cultivo con la ETo. Su valor depende principalmente de las características intrínsecas del cultivo: altura, densidad de la cubierta vegetal, área foliar, resistencia estomática, albedo, etc. Como señalan (Manis et al., 2024) se sabe “la mayoría de los efectos de los diferentes factores meteorológicos se encuentran incorporados en la ETo. Por lo tanto, mientras ETo refleja la demanda climática, el valor de Kc varía principalmente en función de las características particulares del cultivo”. Esto permite emplear valores estándar de Kc (tabulados en manuales FAO) en diversas regiones, ajustando solo por las condiciones locales.

Los cultivos atraviesan varias fases fenológicas (inicial, desarrollo, madurez y término), y durante estas fases el Kc cambia. Se utilizan típicamente tres valores de Kc para cada cultivo: Kc_ini (fase inicial con baja cobertura), Kc_med (fase media o de cobertura completa) y Kc_fin (fase final, madurez) Con estos tres valores, se construye la curva temporal de Kc durante todo el ciclo. En cultivos de rápido crecimiento, Kc pasa de valores bajos (e.g. 0.3–0.5 en germinación) hasta 1.0–1.2 en el pico de la temporada (plena cobertura) y luego disminuye al final (cuando las plantas secan). Por ejemplo, FAO reporta que para el maíz Kc_ini ≈0.3–0.5, Kc_med ≈1.15–1.20, Kc_fin ≈0.25 (para etapas secas), mientras que para hortalizas como tomate se dan Kc_ini≈0.6, Kc_med≈1.15, Kc_fin≈0.8. Estos valores medios corresponden a condiciones estándar (riego abundante, clima semihúmedo), y se encuentran ampliamente documentados en tablas de la FAO.

Es importante notar que las diferencias entre especies (e.g. granos vs hortalizas vs forrajeras) se traducen en distintos rangos de Kc. Cultivos de raíces profundas y follaje denso (alfalfa, caña) pueden llegar a Kc>1.2 en su fase de máximo desarrollo, mientras cultivos de follaje bajo (lechugas, cebada) alcanzan Kc medianos alrededor de 0.9–1.1. Estos valores se deben tomar de fuentes validadas o de estudios locales, y ajustarlos según el entorno específico del cultivo y la región.

4.3.3 Ajustes por condiciones locales: estrés hídrico, viento, radiación, temperatura

Los valores estándar de ETo y Kc deben ajustarse si las condiciones reales difieren de las ideales. En particular, tomando en cuenta lo que dijo (Hopper, 2006) el estrés hídrico reduce la transpiración del cultivo. Para modelarlo se utiliza un coeficiente de estrés que multiplica la ET del cultivo (basal) según el grado de déficit de humedad. Así, ETc real puede expresarse como cuando el suelo comienza a faltar de agua, y la ET disminuye. Este enfoque permite estimar rendimientos bajo riego deficitario o sequía.

Asimismo, factores climáticos como viento fuerte, baja humedad o alta radiación incrementan la evaporación. Bajo condiciones áridas y ventosas, la ET del cultivo puede superar la del cultivo de referencia. Por ejemplo, Allen et al. indican que “el Kc de cultivos altos puede ser hasta 30% mayor en un clima ventoso y árido comparado con un clima húmedo y calmado”. En la práctica, se ajusta el Kc base sumando un porcentaje por viento y déficit de humedad (FAO sugiere, p.ej., sumar 0.1 a Kc_med para vientos ~4 m/s, que elevó el Kc_med trigo de 1.15 a ~1.25). También se corrige por radiación distinta a la estándar: en días nublados Kc efectivo puede reducirse por menor Rn.

Por tanto, en cada local debe calibrarse el cálculo de ETo y Kc a las condiciones reales: ETo ajustando la fórmula de Penman–Monteith con datos locales (o usando un método empírico calibrado), y Kc ajustando los valores tabulados según diferencias en viento, temperatura, humedad o frecuencia de riego. Estos ajustes locales suelen

hacerse con ecuaciones modificadas o factores adicionales (p.ej. FAO propone sumas o multiplicadores según u2, Hr, etc.) para reflejar la fenología específica y clima del sitio.

4.3.4 Métodos para estimar ETo con datos limitados

Cuando faltan datos meteorológicos completos, existen atajos para estimar ETo. Además de la fórmula de Hargreaves (citada en 2.3.1), tal como lo describen (Villazón Gómez et al., 2021) se recurre a métodos como:

•Tanque evaporímetro clase A: mide la evaporación libre de un tanque; el valor obtenido se convierte a ETo mediante coeficientes empíricos conocidos (la evaporación del tanque suele ser ~0.7–0.8 veces ETo). Así se aprovechan datos simples (evaporación del tanque) para estimar la demanda de evapotranspiración. La FAO describe cómo usar estos tanques para ETo.

•Priestley–Taylor: método basado en radiación neta disponible y temperatura; asume condiciones de vaporación no limitadas por viento/humedad. Útil en climas húmedos (coeficiente empírico ~1.26 sobre Penman bajo entornos apropiados).

•Thornthwaite: método histórico que estima ETo a partir de la temperatura media mensual y horas de sol; requiere solo datos de temperatura y latitud. Es simple pero menos preciso en climas extremos o con vientos fuertes.

•Modelos empíricos locales: en algunos casos se generan ecuaciones de ETo calibradas regionalmente, que relacionan variables meteorológicas disponibles (p.ej. solo temperaturas máximas y mínimas, o combinaciones simplificadas). En general, todos estos métodos dan aproximaciones cuando no hay estaciones meteorológicas completas. La precisión varía: Penman–Monteith siempre ofrece la mejor estimación cuando se dispone de datos, mientras los métodos reducidos son más prácticos con datos escasos. La selección debe hacerse considerando la disponibilidad de datos y la exactitud requerida para el cálculo de riego.

4.4 Suelo y manejo del agua en el suelo

El suelo es el reservorio principal del agua en agricultura y sus propiedades físicas determinan cómo se infiltra, retiene y moviliza el agua. Esta sección aborda los atributos del suelo que afectan el riego.

4.4.1 Propiedades físicas del suelo: textura, estructura, porosidad

Según las fuentes de (World Reference Base for Soil Resources 2022, 2022) la textura del suelo (proporción de arena, limo y arcilla) define el tamaño y distribución de poros. Los suelos arenosos tienen poros grandes y buena permeabilidad, pero un menor volumen total de agua retenida. Por el contrario, los suelos arcillosos tienen muchos poros pequeños, alta retención de agua, pero drenaje lento. En general, la capacidad de campo aumenta con la cantidad de finos: suelos finos retienen más agua “contra la gravedad” que suelos gruesos.

La estructura del suelo (agregación) influye en la porosidad efectiva y la estabilidad contra la compactación. Un suelo bien estructurado con agregados estables favorece poros de tamaño medio (macro y microporos) que proveen buen equilibrio entre retención de agua y aireación. Un suelo colapsado o compacto pierde poros, reduce la conductividad hidráulica y la retención útil.

La porosidad total es la fracción del volumen del suelo ocupada por vacíos (agua + aire). Esta depende de la densidad aparente y suelos más densos (compactos) tienen menor porosidad. Los valores típicos de densidad aparente varían de ~1.6 g/cm³ en suelos muy arenosos a ~1.2 g/cm³ en arcillosos con mucha materia orgánica. En la lista de tipos de suelo provista, se ve que disminuye al subir la proporción de arcilla. Dado que la porosidad incluye macro y microporos, también incide en la retención total de agua. En resumen, la textura y densidad determinan la porosidad y retención: suelos arenosos, de baja porosidad (macro > micro) y rápida infiltración; suelos arcillosos, de alta porosidad (muchos microporos) y lenta infiltración.

4.4.2 Capacidad de campo, punto de marchitez y agua disponible para plantas

Estos conceptos ya se definieron en 3.1.1, pero vale enfatizar su importancia en el manejo. La capacidad de campo (CC) y el punto de marchitez permanente (PMP) determinan el agua total almacenada y el agua utilizable de acuerdo con (Shaxson, 2005). Los parámetros CC y PMP suelen obtener de tablas según la textura. Por ejemplo, un suelo franco-arenoso tiene aproximado 0.18 y

0.19 (volumen/volumen), mientras uno arcilloso puede tener aproximado 0.38 y 0.22. Conocer estas cifras es vital: la lámina neta de riego debe reponer el agua extraída hasta un nivel que no supere el PMP. Adicionalmente, el agua fácilmente aprovechable (AFA), explicado en 3.2.3, depende de CC y PMP. Los cálculos de riego consideran estos valores para no exceder el límite de PMP, manteniendo por encima del umbral de estrés.

4.4.3 Infiltración, conductividad hidráulica y retención de agua

El agua del riego o la lluvia debe infiltrarse en el suelo para estar disponible. La velocidad de infiltración inicial depende de la textura y la estructura superficial. Suelos con buena estructura y alto contenido de materia orgánica presentan mayor tasa inicial. Sin embargo, esta tasa disminuye a medida que la superficie se humedece. La conductividad hidráulica del suelo (facilidad con que el agua atraviesa los poros) es mayor en suelos arenosos y menor en arcillosos. Por ejemplo, suelos arenosos pueden tener más de 100 mm/hora, mientras que arcillosos a menudo 10 mm/h. En la práctica, si el caudal de riego excede la capacidad de infiltración, se produce encharcamiento superficial o escorrentía no deseada.

Citando a (Shaxson, 2005) la retención de agua en el suelo está gobernada por la tensión a la que el agua queda retenida. En la zona activa, al inicio del riego el suelo casi saturado drena hasta CC. Luego, durante el periodo inter-riego, el suelo suministra agua a las raíces. Un suelo de textura media (como un franco) ofrece un equilibrio ideal: buena retención y suficiente conductividad. En contraste, un suelo arcilloso retiene gran

volumen de agua, pero libera lentamente, mientras un suelo arenoso libera rápido el agua, pero almacena poco. Esta relación influye en la frecuencia de riego necesaria.

4.4.4 Clasificación de suelos y su influencia en la lámina de riego

En base a lo que dijo (Shaxson, 2005) en sistemas de cálculo de riego se suele clasificar el suelo en tipos (“arenoso”, “franco-arenoso”, “franco”, “franco-arcilloso”, “arcilloso”, etc.) para asignar parámetros típicos de CC y PMP. Por ejemplo, un suelo arenoso (arenas dominantes) tiene capacidad de campo muy baja (~11%), PMP ~5%, lo cual implica muy poca agua útil. En cambio, un suelo arcilloso retiene mucha agua (CC ~38%, PMP ~22% en el ejemplo proporcionado), de modo que almacena más agua por unidad de profundidad. Un suelo franco (loam) se considera balanceado (CC ~28%, PMP ~14%).

Estos parámetros condicionan la lámina de riego necesaria: un suelo arenoso secará más rápido y requerirá riegos frecuentes, pero poco profundos, mientras un suelo arcilloso puede aguantar más tiempo antes de secarse, pero si se aplica agua en exceso puede generar encharcamientos y percolación profunda. Por ende, al diseñar un programa de riego se eligen la lámina (mm) y frecuencia de acuerdo a la textura del suelo: suelos de alta capacidad de retención (franco- arcillosos, arcillosos) aceptan láminas mayores y días largos sin riego; suelos de baja capacidad (arenosos) requieren riegos más frecuentes de lámina menor. En la tabla de tipos de suelo dada se proveen valores típicos de CC y PMP para ajustar estas decisiones.

4.5 Conceptos y fórmulas para el cálculo de la lámina de riego

Con los conceptos previos definidos, ahora se detalla cómo estimar la lámina de riego (en mm) necesaria para atender la demanda hídrica de un cultivo, considerando eficiencia y pérdidas.

4.5.1 Definición de lámina de riego (mm) e interpretación práctica

Como se comentó, la lámina de riego neta (L_r_Neta) se define como la profundidad de agua aplicada sobre la parcela, expresada en mm citando a lo que dijo (Cruz-Bautista et al., 2019). Representa la cantidad de agua realmente consumida por el cultivo (evapotranspirada más retenida en el suelo) entre eventos de riego. Por ejemplo, si la ETc diaria es 6 mm y queremos cubrir 5 días entre riegos, la lámina neta debería ser ~30 mm.

En la práctica, a menudo se distingue entre lámina neta (lo que usa el cultivo) y lámina bruta (lo que se aplica considerando pérdidas). La conversión entre volumen y lámina es simple: 1 mm en 1 ha equivale a 10 000 L (10 m^3). Así, si un emisor entrega 3 m^3/h para una hectárea, en 1 hora riega 3 m^3 = 10 mm. Por ejemplo, 5 m^3/h sobre 1 ha equivalen a 0.5 mm/h.

4.5.2 Ecuación general del cálculo (ETo × Kc ± ajuste) y conversión a lámina aplicada

La ecuación de cálculo fundamental es $ETc = Kc \cdot ETo$ (o $ETc = Ks \cdot Kc \cdot ETo$ si hay estrés). Esto da la demanda neta del cultivo en mm por unidad de tiempo. Para traducir esto a lámina de riego, se suman las demandas del período de riego descrito por (Hopper, 2006). Por ejemplo, si se riega semanalmente y ETo/Kc dan una ETc de 5 mm/día promedio, la lámina neta semanal será $5 \cdot 7 = 35$ mm.

Luego, para pasar de lámina neta a lámina bruta aplicada (L_r_Bruta) se considera la eficiencia del sistema. Si el sistema de riego tiene una eficiencia global e (en fracción), se necesita aplicar mayor volumen:

$$Lbruta = Lnetae.\, L_{bruta} = \frac{L_{neta}}{e}.\, Lbruta = eLneta.$$

Por ejemplo, con 80% de eficiencia, una lámina neta de 35 mm requiere aplicar 35/0.8 = 43.75 mm. Las ineficiencias incluyen las pérdidas por evaporación durante el riego, distribución desigual (sobrepoblación y subaprovechamiento en parcelas) y percolación más allá de la zona de raíces.

Finalmente, para pasar de lámina (mm) a caudal de riego o tiempo de riego, se usa la relación conocida:

$$Volumen\ aplicado = lamina\ (mm) \times area\ (ha)/10$$

Dado un caudal de emisor y el número de emisores por ha, se calcula el tiempo necesario para suministrar la lámina deseada. Estos cálculos prácticos (llamados dimensionado hidráulico) involucran fórmulas básicas de volumen- tiempo, pero son específicos de cada sistema de riego

4.5.3 Periodo de riego, frecuencia y profundidad: relación entre caudal y lámina

La frecuencia de riego (días entre riegos) debe elegirse de modo que la extracción de agua (ETc acumulada) no exceda la parte aprovechable del agua en el suelo (AFA) de acuerdo con (Martín de Santa Olalla Mañas et al., 2005). En términos sencillos: si p·ADT es la lámina neta que puede extraerse sin estrés, entonces el intervalo de riego ideal puede aproximarse como:

$$dias\ entre\ riegos \approx p \cdot ADTETcdiario.\text{días entre riegos} \approx \frac{p \cdot ADT}{ETc_{\text{diario}}}.\, di´as\ entre\ riegos \approx ETcdiariop \cdot ADT.$$

La profundidad de riego en cada evento (lámina de esa aplicación) se calcula multiplicando la demanda ETc promedio por la frecuencia de riego, o a partir de la

diferencia deseada en el contenido de humedad del suelo. Dado un caudal total de aplicación (L/s por ha) y un tiempo de riego (h), la lámina aplicada en ese evento es:

$$Lr = Q \cdot 3600 \cdot t 10\,000. L_r = \frac{Q \cdot 3600 \cdot t}{10\,000}. Lr = 10000Q \cdot 3600 \cdot t.$$

Esto indica cuántos mm de agua se aplican. La programación de la duración se ajusta para obtener la lámina neta deseada (modificado por eficiencia, como se explicó).

4.5.4 Pérdidas y eficiencia del riego (evaporación, percolación, distribución)

En la aplicación de agua, ocurren pérdidas inevitables que deben cuantificarse para determinar la lámina bruta necesaria. Las principales pérdidas son:

•Evaporación superficial: parte del agua aplicada se evapora directamente, especialmente si el riego se realiza en condiciones calurosas o ventosas. Esta pérdida se reduce usando métodos de riego al suelo cubierto o por goteo.

•Percolación profunda: cuando se aplica más agua de la que las raíces pueden retener (sobre irrigan), el exceso drena más allá de la zona radicular, sin beneficiar al cultivo. Esta pérdida es mayor en suelos arenosos o con exceso de lámina.

•Escorrentía superficial: en parcelas con pendiente o mala infiltración, el agua de riego puede escurrir y perderse. Sistemas adecuados (buenas canales o coberturas) minimizan esto.

•Distribución no uniforme: incluso sin pérdidas físicas, en sistemas de riego por gravedad la distribución puede ser desigual (unas partes del campo reciben más agua que otras). La eficiencia del riego (fracción de agua efectivamente usada por el cultivo) contabiliza todas estas pérdidas.

La FAO señala que, considerando ingresos (lluvias + riego) y egresos (ETc + drenaje + escorrentía), el balance puede ser positivo o negativo. Prácticamente, se

calcula una eficiencia típica (p.ej. 60– 90% según sistema: mayor en riego por goteo, menor en surcos).

4.6 Fuentes de datos meteorológicos y su integración

De acuerdo con (Hopper, 2006) el cálculo de ETo y, por ende, de la lámina de riego, requiere datos climáticos (precipitación, temperatura, humedad, radiación, viento). Existen diversas fuentes de datos meteorológicos:

•Estaciones locales: redes nacionales (ej. SENAMHI, NOAA) o locales (tecnológicas agronómicas, estaciones automáticas) proporcionan datos de temperatura, viento, radiación, etc. La ventaja es la representatividad; la desventaja puede ser infraestructura limitada.

•Redes públicas/internacionales: bases de datos globales (NASA POWER, ERA5, etc.) ofrecen información en modelos numéricos o satelital. Los APIs de servicios meteorológicos (OpenWeatherMap, Meteostat, etc.) permiten descargar datos históricos o en tiempo real programáticamente. Estas fuentes son convenientes si las estaciones locales faltan, aunque con menor precisión local.

•Fuentes complementarias: sensores de radiación o lisímetros específicos; o datos de radar y satélite para precipitación.

La calidad y representatividad de los datos son fundamentales. Debe verificarse la resolución espacial: la información de una estación distante puede no reflejar microclimas locales. La resolución temporal también importa: algunos métodos requieren datos horarios (Penman– Monteith idealmente) mientras otros bastan con datos diarios (Hargreaves, Thornthwaite). Hay que gestionar vacíos o errores en las series: se usan técnicas de interpolación temporal (p.ej. promedio de días cercanos) o espacial (interpolación entre estaciones cercanas). En general, se recomienda usar datos lo más locales y completos posible, y si se emplean redes globales, validar con mediciones de campo para calibrar los cálculos.

Las variables requeridas dependen del método de cálculo: para Penman–Monteith se necesitan radiación neta, temperatura (max/min o media), humedad (o déficit de saturación) y viento medio a 2 m. También se utiliza la precipitación diaria para el balance hídrico. Para métodos simplificados como Hargreaves bastan temperatura y radiación extraterrestre (o latitud). En resumen, las principales variables son: precipitación, temperaturas (tmax, tmin, media), humedad relativa, radiación solar (o evapotranspirómetro) y viento.

Cuando falte alguna variable, se pueden implementar métodos de interpolación o aproximación manual: por ejemplo, interpolar lluvia a partir de estaciones contiguas cercanas, estimar la radiación con modelos de nubosidad, o usar valores climatológicos medios. Algunas prácticas incluyen usar la diferencia entre tmax y tmin para estimar la radiación efectiva, o asumir un patrón diurno típico para viento. Es importante documentar estos reemplazos, ya que agregan

incertidumbre. La investigación agronómica actual resalta la importancia de integrar distintas fuentes y validar con mediciones locales.

4.7 Modelos y algoritmos agronómicos aplicados

Finalmente, se consideran los modelos y algoritmos que implementan estos conceptos en aplicaciones prácticas de riego.

4.7.1 Implementación de Penman-Monteith y alternativas para ETo

En sistemas informáticos o algoritmos agrícolas, la ecuación de Penman–Monteith puede programarse directamente (requiere manejo numérico cuidadoso) o utilizar librerías especializadas. Alternativamente, se implementan métodos empíricos para ETo cuando los datos son limitados. Por ejemplo, se suele incorporar la fórmula de Hargreaves en software de riego ligero (como demuestra el código de la aplicación en

contexto). Los ajustes locales se aplican a los parámetros de estas fórmulas según calibraciones con datos observados.

4.7.2 Modelos de adaptación del Kc por etapa fenológica

La curva de Kc según (Hopper, 2006) se modela interpolando linealmente entre los puntos (Kc_ini, Kc_med, Kc_fin) según la fase del cultivo. Muchas aplicaciones toman los valores base (según tabla FAO) y ajustan dinámicamente el Kc diario a lo largo del tiempo. Esto implica, por ejemplo, incrementar gradualmente Kc desde el valor inicial hasta el medio a medida que el cultivo crece, y luego reducirlo en la fase de madurez. Los modelos de programación de riego deben permitir ingresar diferentes Kc según la fenología (sembrado, cubrimiento, floración, cosecha). Al inicio de cada fase fenológica se actualiza el Kc conforme al cultivo específico, tal como sugiere Allen et al. (1998) al requerir tres valores de Kc para definir la curva.

4.7.3 Balance hídrico por parcela (modelo de suelo simple vs. multicapa)

Para estimar la humedad del suelo día a día, se utilizan modelos de balance hídrico. Un modelo simple como describe (Verdoodt et al., 2005) asume una sola capa de suelo activo (p.ej. 30–60 cm) con un solo estado de humedad. Este modelo suma la precipitación + riego y resta ETc, asignando la diferencia al suelo o pérdidas. Es fácil de implementar, pero impreciso en suelos profundos o con raíces cambiantes.

Un modelo multicapa divide el suelo en varias capas (p.ej. 5–10 cm cada una) y calcula humedad, flujo e infiltración entre ellas. Estos modelos (como SWAP, CROPWAT con multilayer) requieren parámetros adicionales (conductividades de cada capa, humectación vertical) pero simulan más realísticamente situaciones como riego superficial, escorrentías internas, o raíces que profundizan progresivamente.

En general, para un calculador de riego simple suele bastar el modelo de capa única con parámetros globales de CC y PMP. Para aplicaciones más avanzadas se usa multicapa para afinar

la estimación de la humedad disponible. Los usuarios deben conocer qué modelo emplea la herramienta.

4.7.4 Algoritmos para programación de riego (umbral de extracción, días entre riegos)

Con los modelos anteriores, el algoritmo de programación de riego según (Cancela et al., 2015) sigue típicamente estas reglas:

1. Calcular diariamente el balance hídrico: actualizar el almacenamiento de agua del suelo restando la ETc diaria de la reserva actual.

2. Establecer un umbral crítico de agotamiento: por ejemplo, cuando la humedad cae al nivel de agotamiento permisible (basado en p·ADT), activar riego. Esto equivale a “si la humedad remanente ≤ PMP + (1–p) ·(CC–PMP)”.

3. Calcular la lámina a aplicar: según la cantidad necesaria para regresar al nivel de carga inicial (por ej. volver a CC) o al nivel óptimo que asegure la próxima ventana de riego.

4. Ejecutar el riego (sumar la lámina neta al suelo, restando eficiencias). Reiniciar el conteo de días post-riego.

El umbral p·ADT. es común para cultivos de cobertura completa. Algunos algoritmos también incluyen consideraciones de fenología: permiten un cierto déficit controlado en etapas menos críticas (madurez) o requieren más agua en fases de alta demanda (floración). En esencia, el sistema programa el día del riego cuando el acumulado de demanda iguala la AFA disponible.

4.7.5 Optimización simple: reducción de uso de agua sin afectar rendimiento

Finalmente, en base a lo que dijo (Xu et al., 2023) se revisan estrategias sencillas para economizar agua. Estudios agronómicos muestran que riego deficitario controlado en etapas no críticas (por ejemplo, después del cuajado en frutales) puede ahorrar agua con mínima pérdida de rendimiento. Algoritmos pueden incorporar esta heurística

reduciendo voluntariamente el p (permitiendo mayor estrés) o disminuyendo la lámina neta en fases tardías. La idea es trabajar cerca del límite de estrés sin alcanzarlo, maximizando la eficiencia hídrica.

Otro enfoque es la aplicación localizada o cambio de sistemas: por ejemplo, cambiar de riego por surcos a riego por goteo incrementa la eficiencia (menor evaporación superficial). Si se dispone de datos, el algoritmo puede sugerir el sistema óptimo para un cultivo dado (basado en eficiencias típicas).

En síntesis, los modelos y algoritmos aplicados en esta área combinan fórmulas de cálculo de ETo/ETc con reglas de programación basadas en el balance del suelo. Las fuentes bibliográficas proveen la base teórica para estos procedimientos, y las implementaciones prácticas deben calibrarse con datos locales para asegurar la precisión del sistema de riego.

4.8 Diseño de la base de datos y modelado de información

De acuerdo con lo planteado por varios autores (Elmasri & Navathe, 2010) el diseño de una base de datos constituye un paso fundamental dentro del desarrollo de sistemas de información, dado que define la estructura, organización e integridad de los datos que serán almacenados y manipulados. El diseño de bases de datos, lejos de ser un mero detalle técnico, se convierte en la base sobre la cual descansa la fiabilidad, eficiencia y mantenibilidad de todo sistema de información.

Primero, se realiza el modelado conceptual, fase en la que se abstraen los requerimientos de datos independientemente de la tecnología de gestión. En esta etapa se emplean modelos como el Modelo Entidad-Relación (ER), o sus extensiones (EER / extendido), para representar entidades, atributos, relaciones, cardinalidades y restricciones de integridad. Este modelado permite representar de forma clara y comprensible la realidad del dominio problemático, favoreciendo la comunicación entre analistas, desarrolladores y usuarios.

Posteriormente, mediante un proceso de transformación, el modelo conceptual se convierte en un modelo lógico, típicamente basado en el Modelo Relacional, que adapta la abstracción conceptual al paradigma de bases de datos que se utilizará. En esta fase se definen las tablas, campos, claves primarias y foráneas, relaciones, dominios de atributos, y restricciones de integridad referencial. Este enfoque es especialmente relevante cuando se pretende emplear sistemas gestores de bases de datos relacionales — aún hoy los más utilizados — por su formalidad y robustez.

Un aspecto esencial del diseño lógico lo constituye la aplicación de la teoría de la normalización, que a través de dependencias funcionales y formas normales busca reducir redundancias, evitar anomalías en la inserción, actualización o eliminación y garantizar la consistencia de los datos. Por tanto, un diseño bien normalizado favorece la integridad, rendimiento y escalabilidad de la base de datos.

Finalmente, una vez definido el modelo lógico, puede establecerse el diseño físico, que implica decidir cómo se materializarán las tablas en el sistema gestor de bases de datos: estructuras de almacenamiento, índices, organización en discos, mecanismos de acceso y optimización de consultas. Esta fase es imprescindible cuando se busca eficiencia en el acceso, almacenamiento y mantenimiento de la base de datos.

Adicionalmente, investigaciones recientes plantean nuevos enfoques para la conceptualización de datos que trascienden los modelos clásicos. Por ejemplo, se ha propuesto el uso de modelos inspirados en procesos cognitivos — Cognitive Data Model (CDM) — para afrontar limitaciones de los modelos tradicionales en la gestión de relaciones complejas, adaptabilidad y escalabilidad. Este tipo de propuestas sugiere que el diseño de bases de datos podría evolucionar hacia representaciones más dinámicas y semánticamente ricas, anticipando los requerimientos del Big Data, inteligencia artificial y contextos de alta complejidad.

En consecuencia, la elección de una metodología rigurosa de diseño de base de datos — pasando por las fases conceptual, lógico y físico — resulta fundamental para asegurar que la base de datos satisfaga los requerimientos del problema de

investigación, permita un adecuado almacenamiento, consulta y consistencia de la información, y sea escalable en función del crecimiento del sistema. Además, al contextualizar el modelado de la información conforme a las necesidades específicas del proyecto, se asegura que la base de datos represente de manera fiel la realidad objeto de estudio, lo cual aporta validez y calidad al sistema.

Así, en función del planteamiento del problema, de la(s) hipótesis(s) formulada(s) y de la metodología prevista, el diseño de la base de datos y el modelado de la información constituyen un pilar esencial del proyecto, dado que determinan la estructura, coherencia y confiabilidad de los datos que servirán para sustentar los análisis y resultados de la investigación.

4.8.1 Esquema general: tablas y relaciones principales

De acuerdo con lo descrito por varios autores dedicados al diseño de bases de datos (Elmasri & Navathe, 2010), el esquema general de una base de datos —entendido como la definición de sus tablas principales y las relaciones entre ellas— constituye una etapa esencial dentro del proceso de modelado de la información, puesto que define la estructura lógica que permitirá la persistencia, integridad y consistencia de los datos.

En el enfoque clásico del diseño relacional, cada “relación” del modelo conceptual se traduce a una “tabla” dentro del esquema lógico. Las tablas se conforman por atributos que se convierten en columnas, y cada fila representa una tupla u ocurrencia de la entidad. Para garantizar la unicidad y permitir referencias consistentes entre tablas, ciertas columnas se designan como claves primarias, mientras que las claves externas permiten establecer vínculos relacionales que reflejan asociaciones entre entidades.

Para representar las relaciones entre tablas, es necesario definir claramente los tipos de asociación, lo que involucra cardinalidades y restricciones de integridad. En relaciones de tipo muchos-a- muchos (N:M), por ejemplo, se crea una tabla intermedia que contiene al menos las claves primarias de las dos tablas relacionadas (como claves externas) y, en su conjunto, actúan como clave primaria de dicha tabla intermedia. En cambio, relaciones uno-a-muchos (1: N) pueden representarse simplemente con una

clave foránea en la tabla del lado “muchos”. Este diseño asegura que la estructura relacional refleje adecuadamente la semántica del dominio.

La definición de este esquema general resulta particularmente relevante en un proyecto de investigación, pues permite organizar los datos de acuerdo con las variables e hipótesis planteadas, facilitando la recolección, almacenamiento y posterior análisis. Un esquema bien estructurado favorece la integridad referencial, minimiza redundancias, y posibilita consultas eficientes, lo que contribuye a la validez y confiabilidad de los resultados.

En consecuencia, al definir las tablas principales y sus relaciones, se deben considerar cuidadosamente las entidades del dominio, los atributos relevantes, las claves primarias y foráneas, así como las relaciones cardinales y opcionales que reflejen correctamente las reglas de negocio o dominio de la investigación. Esto asegura que el modelo lógico de datos esté alineado con el planteamiento del problema, las hipótesis formuladas y la metodología que se empleará, estableciendo una base sólida para el almacenamiento y análisis de información.

Por lo tanto, la elaboración de un esquema general —es decir, un conjunto de tablas principales y relaciones claramente definidas— no se presenta como una mera formalidad técnica, sino como un componente esencial del diseño metodológico del proyecto. De esta manera, se garantiza que la estructura de datos respalde de forma fiel y eficaz los objetivos de investigación, y que la implementación del sistema de información permita consistencia, integridad y adaptabilidad conforme los requerimientos del estudio.

4.8.2 Integridad, normalización y performance

De acuerdo con lo señalado en la literatura sobre diseño de bases de datos (IBM Corporation, 2025), la normalización constituye un mecanismo esencial para garantizar la integridad y la consistencia de los datos en un sistema. En su concepción original, la normalización busca estructurar las relaciones de datos para evitar redundancias,

anomalías en inserciones, actualizaciones o borrados, y dependencias funcionales innecesarias que comprometan la validez del conjunto de datos.

Así, mediante la aplicación de las diferentes “formas normales” —como la primera (1NF), segunda (2NF), tercera (3NF) y, en algunos casos, la Boyce-Codd Normal Form (BCNF) o formas superiores— se logra reducir la redundancia y las dependencias no deseadas. Cada tabla resultante cumple criterios tales como: atributos atómicos, dependencia completa de la clave primaria, y ausencia de dependencias transitivas o multivaluadas, según la forma normal aplicada.

La normalización fortalece la integridad referencial, en donde las claves foráneas deben referir siempre a claves primarias válidas en tablas relacionadas, asegurando que las relaciones entre datos sean consistentes y estén correctamente definidas. Esto contribuye a que los datos almacenados representen fielmente el dominio real, reduciendo el riesgo de duplicaciones, datos inconsistentes o pérdidas de información.

No obstante, el uso estricto de la normalización tiene implicaciones sobre el performance o rendimiento del sistema. Al fragmentar los datos en múltiples tablas relacionadas, las consultas que requieren reunir información de varias tablas implican operaciones de join, lo que puede incrementar la complejidad de las consultas, alargar su ejecución y requerir más recursos computacionales. Por ello, algunos estudios advierten que el nivel óptimo de normalización depende del contexto de uso: cuando las operaciones de lectura/consolidación son frecuentes, un esquema demasiado fragmentado puede degradar el rendimiento.

De hecho, investigaciones recientes han evaluado empíricamente estos trade-offs. En un estudio sobre el impacto del diseño lógico en el tamaño de la base de datos, la complejidad de consultas, el rendimiento y el consumo energético, se observó que al normalizar de 1NF a 2NF se redujo el tamaño en disco, se incrementó el rendimiento de las transacciones, y se redujo notablemente el consumo energético. Sin embargo, más allá de 2NF, los beneficios sobre performance tendieron a disminuir, mientras que la complejidad y el número de tablas aumentaron.

En consecuencia, para un proyecto de investigación que contempla un planteamiento del problema, hipótesis y metodología que dependen del almacenamiento, manejo y análisis de datos, es imprescindible evaluar cuidadosamente el grado de normalización que se implementará.

4.8.3 Almacenamiento de históricos y trazabilidad de cálculos

El almacenamiento de información histórica y la trazabilidad de operaciones constituyen elementos fundamentales en los sistemas de información contemporáneos. Se realizan mediante herramientas que facilitan el seguimiento completo del histórico, ubicación y trayectoria de elementos a lo largo de su ciclo de vida. En base a lo que se menciona (Microsoft, 2022), la trazabilidad y reproducibilidad son condiciones inherentes a la ciencia de calidad, ya que los estudios reproducibles incluyen código informático capaz de recrear todos los resultados a partir de datos originales, registrando perfectamente el proceso de análisis y reduciendo drásticamente el riesgo de errores.

De acuerdo con el análisis sobre sistemas de trazabilidad, estos están compuestos por subsistemas de identificación individual, captación de datos mediante sensores y gestión de datos. El software basado en inteligencia artificial compila todos los datos para contrastarlos con modelos predictivos, permitiendo una monitorización en tiempo real. Según estudios sobre reproducibilidad computacional, los sistemas de control de versiones permiten acceder a las últimas versiones haciendo una trazabilidad de cambios realizados por distintos autores, lo que resulta esencial para validar y verificar resultados en investigaciones científicas.

En el contexto de bases de datos, se observa que el registro detallado y la generación de informes no solo ayudan en auditorías regulatorias, sino que fortalecen la postura general de seguridad. Citando a investigadores sobre auditoría de bases de datos, este proceso garantiza la seguridad, integridad y calidad de la información almacenada, permitiendo identificar posibles amenazas, vulnerabilidades y errores en la gestión de datos. La auditoría de bases de datos utiliza herramientas y tecnologías para registrar las transacciones, proporcionando evidencia de actividades realizadas.

4.9 Arquitectura del sistema y componentes web

La arquitectura de aplicaciones web constituye un esquema fundamental que define cómo interactúan los distintos componentes de un sistema. Según lo establecido en estudios sobre arquitectura web, una aplicación web es proporcionada por un servidor web y utilizada por usuarios

que se conectan desde cualquier punto vía clientes web. De acuerdo con investigaciones sobre arquitecturas multinivel, se realiza una separación lógica y física de la funcionalidad en tres niveles principales: presentación, aplicación y datos, donde cada nivel puede ejecutarse en sistemas operativos y plataformas de servidor independientes.

El modelo de arquitectura cliente-servidor en tres capas se observa como el más utilizado en sistemas modernos. Se comparan los modelos de dos y tres capas para la arquitectura cliente- servidor, haciendo especial énfasis en las ventajas del modelo de tres capas por su facilidad de extensión a diferentes entornos de desarrollo sin necesidad de grandes variaciones en los procesos de manipulación de datos. En base a lo que mencionan en estudios sobre arquitectura web, en este modelo el cliente solicita los recursos mientras el servidor procesa aplicaciones y respuestas, formando un conjunto de capas utilizado para el manejo de sistemas operativos.

Los niveles de presentación y de datos no pueden comunicarse directamente entre sí en una aplicación de tres niveles, toda la comunicación pasa por el nivel de la aplicación. Esta separación de roles es tremendamente flexible, pero su mayor activo es la modularidad estanca, permitiendo modificar cualquiera de las capas sin tener incidencia en el resto. Según lo establecido (Shklar & Rosen, 2012), se realizan estudios sobre el patrón modelo vista controlador, el cual hace una separación entre la parte gráfica de la aplicación y los procesos de la misma, siendo uno de los patrones básicos para el desarrollo de aplicaciones orientadas a la web.

De acuerdo con la definición establecida en estudios comparativos, el patrón MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, basándose en las ideas de reutilización de código y la separación de conceptos. MVC fue introducido por Trygve Reenskaug en Smalltalk-76 durante su visita a Xerox Parc en los años 70. En base a investigaciones sobre aplicaciones web, se utilizan lenguajes de etiquetas de hipertexto compuestos de etiquetas a través de las cuales se especifican los textos, imágenes y otros componentes que el navegador interprete.

En el contexto de frameworks modernos, se realizan estudios comparativos entre React, Angular y Vue. Según investigaciones sobre frameworks de ingeniería de software (Google, n.d.), Angular es un framework de código abierto mantenido por Google que sirve para desarrollar aplicaciones web de estilo basado en componentes, los cuales son bloques de construcción reutilizables que encapsulan la lógica y la interfaz de usuario de una parte específica de una aplicación web. De acuerdo con análisis sobre React (Facebook, n.d.), esta biblioteca fue desarrollada por Facebook y se utiliza para facilitar la construcción de interfaces de usuario complejas y dinámicas, dividiéndolas en componentes reutilizables.

Citando a investigaciones recientes sobre arquitectura web (Shklar & Rosen, 2012), se observa que las aplicaciones se construyen en base a la combinación de una serie de componentes individuales, lo que asegura un gran modularidad de las aplicaciones y, por tanto, una mejor escalabilidad y

mantenimiento. Hasta la irrupción de las arquitecturas de microservicios, la amplia mayoría de aplicaciones web se desarrollaban utilizando arquitectura monolítica al ser la más extendida.

4.9.1 Arquitectura general: cliente – servidor – base de datos – APIs

La arquitectura cliente-servidor constituye uno de los modelos fundamentales en el desarrollo de sistemas de información contemporáneos. Según lo establecido por

investigaciones sobre arquitectura de aplicaciones web, en esta arquitectura la capacidad de proceso está repartida entre los clientes y los servidores, siendo más importantes las ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades. De acuerdo con estudios sobre arquitectura de software (Shklar & Rosen, 2012) se define un patrón donde existen dos principales componentes: el cliente que solicita servicios, y el servidor que provee un conjunto de servicios, pudiendo actuar ambos tanto como cliente o servidor según el contexto.

En base a lo que mencionan análisis sobre sistemas cliente-servidor, el funcionamiento se basa en una secuencia clara donde el cliente inicia la interacción enviando una solicitud a través de la red, y el servidor analiza el tipo de petición para ejecutar las operaciones necesarias como consultar una base de datos o generar una respuesta dinámica. Se distinguen tres elementos fundamentales: el proceso cliente que inicia el diálogo, el proceso servidor que espera pasivamente las peticiones de servicio, y el middleware que provee la conectividad entre ambos para intercambiar mensajes.

El modelo de arquitectura de tres niveles se observa como el más implementado en aplicaciones empresariales modernas. Según (Biehl, 2017), se realiza una separación lógica y física de la funcionalidad en tres niveles: presentación donde reside el cliente, aplicación que actúa como intermediador, y datos donde se encuentra la base de datos con las relaciones entre tablas. Las aplicaciones comerciales se dividen en tres partes: acceso a datos, lógica de la aplicación y presentación, lo que permite una modularidad estanco donde se puede modificar cualquiera de las capas sin tener incidencia en el resto.

La integración mediante servicios web y APIs representa un componente esencial en esta arquitectura. De acuerdo con la definición establecida por IBM (2024), las API REST proporcionan una forma ligera de crear API web y se utilizan comúnmente para facilitar el intercambio de datos entre aplicaciones, servicios web y bases de datos, y para conectar componentes en arquitecturas de microservicios. Según lo planteado por Roy Fielding en su tesis doctoral del año 2000, REST proporciona a los desarrolladores un alto nivel de flexibilidad, consistencia, escalabilidad y eficiencia.

Una arquitectura cliente-servidor está compuesta de clientes, servidores y recursos, con la gestión de solicitudes a través de HTTP y una comunicación sin estado, lo cual implica que no se almacena la información del cliente entre las solicitudes. Según investigaciones sobre desarrollo de APIs (Paje, 2023), se implementan sistemas basados en tokens de autenticación como JWT para brindar seguridad adicional al proceso de comunicación entre el cliente y el servidor, mientras que el servidor gestiona tanto los recursos como la lógica del negocio mediante consultas a la base de datos.

4.9.2 Patrón de diseño: API Routes, SSR/CSR, REST/GraphQL

Los patrones de diseño en el desarrollo de aplicaciones web constituyen estrategias fundamentales para optimizar la arquitectura y el rendimiento de los sistemas. Según investigaciones sobre implementación de patrones en arquitectura API REST, al diseñar una arquitectura se debe pensar en los patrones de diseño como una estrategia para que el proyecto pueda ser manipulado por cada una de las áreas de un equipo de programación, desde la base de datos hasta el consumo de las APIs en alguna aplicación. De acuerdo con Microsoft (2024), las arquitecturas de microservicios necesitan un buen diseño de API, ya que todos los intercambios de datos entre servicios se producen mediante mensajes o llamadas API.

El enrutamiento de APIs representa un componente esencial en arquitecturas modernas. Citando a investigaciones sobre el patrón API Gateway, la descomposición en microservicios trae consigo el desarrollo de múltiples APIs que ofrecen sus endpoints, las cuales pueden estar segregadas en un ambiente distribuido con nombres de dominios diversos. Según estudios sobre diseño de API (Sabbag Filho, 2025) las API públicas y las API de back-end tienen requisitos ligeramente distintos, donde una API pública debe ser compatible con las aplicaciones cliente, normalmente usando REST sobre HTTP.

En el contexto de renderizado de aplicaciones web, se observan dos paradigmas principales: Server-Side Rendering y Client-Side Rendering. En base a lo que mencionan análisis sobre SSR (Sabbag Filho, 2025) el renderizado del lado del servidor permite a

los usuarios ver contenido HTML en el navegador más rápidamente que CSR, ofreciendo un tiempo hasta el primer byte más rápido, ya que el usuario no tiene que esperar a que JavaScript se ejecute antes de ver el HTML. De acuerdo con estudios sobre renderizado web, SSR es un enfoque en el cual las páginas web se generan en el servidor antes de ser enviadas al navegador, mientras que CSR implica que el navegador carga una página en blanco y luego utiliza JavaScript para llenar esa página con contenido.

El proceso de rendering determina quién hace ese trabajo y cuándo: si el servidor antes de enviar la página, el navegador al recibirla, o una fase previa de compilación. La gran ventaja del SSR es que el servidor carga previamente las páginas web y la petición del usuario se procesa casi de forma inmediata, teniendo una influencia positiva en el posicionamiento en buscadores. En contraste, el CSR es de gran utilidad para proyectos web que cuentan con una gran interacción con los usuarios, aunque el proceso de carga inicial es más largo en comparación con SSR.

Respecto a los estilos arquitectónicos para APIs, REST y GraphQL representan dos enfoques distintos ampliamente utilizados en la industria. REST es un estilo arquitectónico estructurado para aplicaciones hipermedia en red diseñado para emplear un protocolo de comunicación sin estado, cliente-servidor y almacenable en caché, mientras que GraphQL es un lenguaje de consulta y

tiempo de ejecución de API desarrollado por Facebook. De acuerdo con estudios comparativos sobre eficiencia de desempeño, REST se ha convertido en el estilo más aplicado en la comunicación con una Interfaz de Programación de Aplicaciones, y GraphQL se ha constituido en una alternativa al estilo REST.

En base a lo establecido por (Stowe, 2019) las API de REST requieren que las solicitudes de los clientes sigan una estructura fija para recibir un recurso, y siempre devuelven un conjunto de datos completo, mientras que GraphQL permite que el cliente especifique exactamente los datos que necesita. Citando a investigaciones comparativas, GraphQL tiene un tiempo de respuesta más rápido que los servicios web basados en REST, siendo preferible su implementación para dispositivos móviles que

contienen recursos de hardware más bajos. Según análisis sobre arquitecturas de API, GraphQL no está tratando con recursos dedicados, sino que todos los recursos se consideran como un conjunto de grafos conectados entre sí, mientras que REST abraza el concepto de tener múltiples endpoints.

4.9.3 Escalabilidad y despliegue (hosting, CDN, cacheo)

La escalabilidad representa una característica fundamental en el desarrollo de aplicaciones web modernas, permitiendo que los sistemas se adapten eficientemente a incrementos en la demanda de usuarios y datos. Las plataformas de alta concurrencia, esta caracterización de indispensable se asocia básicamente con Requerimientos No Funcionales como disponibilidad, escalabilidad y tolerancia a fallos. De acuerdo con estudios sobre optimización de aplicaciones web (Liu et al., 2020), el desarrollo de aplicaciones y sitios web especializados actualmente es una de las actividades principales y su utilización dentro del campo empresarial es cada vez más común, mostrando un crecimiento exponencial.

Los servidores de todas las capas se tienen que poder configurar en clusters o shards sin perjuicio para la aplicación, la mayor cantidad posible de procesos deben ser asíncronos, y todos los datos estáticos deben ser cacheados. Citando a investigaciones sobre aplicaciones escalables, se realizan divisiones de la aplicación en componentes independientes mediante arquitectura de microservicios para facilitar la escalabilidad y el mantenimiento a medida que la aplicación se desarrolla, donde cada servicio se encarga de una funcionalidad específica y se puede escalar de forma autónoma.

El despliegue en ambientes de hosting en la nube constituye una solución avanzada para garantizar disponibilidad y escalabilidad. El alojamiento en la nube se basa en una red de servidores en la nube virtuales y físicos conectados, lo que garantiza una mayor flexibilidad y escalabilidad, donde las organizaciones pagan solo por los recursos que utilizan. El despliegue de aplicaciones web es el proceso mediante el cual

una aplicación desarrollada localmente se pone en un entorno de producción en servidores donde los usuarios finales pueden acceder y utilizar la aplicación.

Las Redes de Distribución de Contenido representan un componente crítico para optimizar el rendimiento global de aplicaciones web. Citando a definiciones académicas, una red de distribución de contenidos es una red superpuesta de computadoras que contienen copias de datos, colocados en varios puntos de una red con el fin de maximizar el ancho de banda para el acceso a los datos de clientes por la red. Según investigaciones sobre CDN (Whitestack, 2024), el objetivo principal es mejorar la velocidad de carga de sitios web y aplicaciones al reducir la latencia y minimizar la distancia física entre los servidores y los usuarios.

El almacenamiento en caché permite reutilizar de forma eficaz los datos recuperados o procesados anteriormente, donde los datos en una memoria caché suelen almacenarse en hardware de acceso rápido como la memoria de acceso aleatorio. Dado que la memoria es órdenes de magnitud más rápida que el disco, la lectura de datos de la caché en memoria es extremadamente rápida con menos de un milisegundo, lo que mejora el rendimiento general de la aplicación.

La implementación de estrategias de caching en el backend de aplicaciones web permite optimizar el rendimiento, reducir la carga del servidor y mejorar la experiencia del usuario al almacenar temporalmente datos en la memoria cache para acceder a ellos de forma rápida y eficiente. Citando a estudios sobre optimización web la implementación eficaz de la caché web no solo mejora el rendimiento y la velocidad de un sitio web, sino que también reduce la carga en los servidores y mejora la escalabilidad del sitio. La caché es una técnica clave en el desarrollo de aplicaciones web para mejorar el rendimiento y reducir la latencia de las solicitudes, permitiendo que las aplicaciones funcionen de manera más eficiente al almacenar respuestas y reutilizarlas.

4.9.4 Gestión de estados y sincronización de datos

La arquitectura de un sistema web constituye el diseño fundamental que define la estructura, el comportamiento y la interacción entre sus componentes de hardware, software y datos para cumplir con los objetivos de rendimiento, escalabilidad y mantenibilidad. En el contexto específico de las aplicaciones web modernas, esta arquitectura abarca componentes del lado del cliente, como la interfaz de usuario y los scripts, y componentes del lado del servidor, que incluyen la lógica de la aplicación, el sistema de gestión de bases de datos y el servidor web. Según la arquitectura de aplicaciones web es un esquema que define cómo interactúan estos componentes, desempeñando un papel crucial en la usabilidad, la optimización de costos y la capacidad de la aplicación para adaptarse a las necesidades cambiantes del negocio. Un diseño arquitectónico adecuado, que priorice principios como el modularidad y la escalabilidad, es crítico para garantizar que los sistemas sean eficientes y seguros a largo plazo.

Citando a (Donvir et al., 2024) dentro de esta estructura, la gestión del estado de la aplicación emerge como una preocupación central. Una arquitectura de componentes bien definida, que descompone el sistema en unidades lógicas, funcionales y reutilizables, facilita enormemente esta gestión. Como se ejemplifica en el proyecto Zentag Extension, una arquitectura modular permite la implementación de un módulo de estado centralizado (state.js) que coordina la lógica de la

aplicación y gestiona los datos de manera coherente a lo largo de diferentes vistas y componentes de la interfaz de usuario. Este enfoque no solo mejora la mantenibilidad al permitir cambios en componentes individuales sin afectar al sistema completo, sino que también proporciona una base sólida para manejar la complejidad mediante la descomposición del sistema en partes más manejables e independientes.

Paralelamente, la sincronización de datos se presenta como un desafío técnico fundamental, especialmente en sistemas distribuidos o que sirven a usuarios en múltiples ubicaciones. De acuerdo con análisis sobre el desarrollo web, la sincronización es el

proceso de garantizar que los mismos datos se almacenen y actualicen de manera consistente en varios dispositivos o servidores, siendo crucial para la experiencia del usuario, el rendimiento y la integridad de la información. Las técnicas para lograrlo son variadas e incluyen la replicación de bases de datos (en modelos maestro-esclavo o maestro-maestro) y el uso de protocolos de sincronización específicos. En aplicaciones empresariales, como las suites de Dynamics 365, la sincronización del lado del servidor se configura como la opción preferida para gestionar el flujo bidireccional de correo electrónico, contactos y citas, operando de manera asíncrona y programada para optimizar el uso de recursos y garantizar la disponibilidad de la información. La elección del método de sincronización adecuado depende, en última instancia, de requisitos específicos como la criticidad de la actualización en tiempo real, la tolerancia a la latencia de red y la necesidad de resolver conflictos de datos.

4.10 Tecnologías utilizadas

4.10.1 Next.js

Desde una perspectiva general y de acuerdo con (Riva, 2022) , la elección de la tecnología base constituye un pilar fundamental en el planteamiento de cualquier proyecto de desarrollo web, directamente influyendo en su escalabilidad, rendimiento y mantenibilidad. En el panorama actual, Next.js se ha establecido como un framework de React ampliamente adoptado para la construcción de aplicaciones web de pila completa (full-stack). Su relevancia radica en que proporciona una capa de abstracción sobre herramientas de bajo nivel, permitiendo a los desarrolladores concentrarse en la lógica del producto y acelerar el tiempo de entrega. La propuesta de valor de Next.js, según se analiza en la literatura especializada, reside en su capacidad para ofrecer una experiencia de desarrollador optimizada, combinada con características listas para producción como el renderizado híbrido, el enrutamiento basado en archivos y optimizaciones automáticas de imagen y código. Este conjunto de características resuelve problemas comunes en el desarrollo frontend tradicional, como la compleja configuración para lograr actualizaciones en caliente (hot reloading) y la optimización del

rendimiento. Por lo tanto, su adopción puede plantearse como una hipótesis metodológica para abordar desafíos inherentes a la creación de aplicaciones web modernas, complejas y con requisitos de rendimiento y SEO exigentes.

Aterrizando en un nivel más específico, la arquitectura y el modelo de renderizado de Next.js presentan soluciones metodológicas concretas para problemas de entrega de contenido y gestión de datos. Un análisis técnico destaca que el framework soporta de manera nativa múltiples estrategias de renderizado: Generación de Sitios Estáticos (SSG), Renderizado del Lado del Servidor (SSR), Regeneración Estática Incremental (ISR) y Renderizado del Lado del Cliente (CSR). Esta flexibilidad permite una metodología híbrida, donde se puede seleccionar la estrategia óptima para cada parte de la aplicación. Por ejemplo, el contenido estático o semi-estático puede pre-renderizarse (SSG/ISR) para una carga ultrarrápida, mientras que los datos dinámicos y personalizados pueden generarse bajo demanda en el servidor (SSR). Citando a los expertos, "la renderización híbrida es cuando se combinan diferentes estrategias de renderizado para diferentes propósitos". Este enfoque metodológico aborda directamente el problema del "Time to Interactive" y la percepción de velocidad por parte del usuario, ya que permite enviar marcos HTML completos desde el servidor, hidratándolos después con interactividad.

Finalmente, en el ámbito más concreto de la implementación, Next.js introduce paradigmas que afectan directamente la metodología de desarrollo y la estructura del código. La versión 13 y posteriores consolidaron el App Router y los React Server Components, lo que supone un cambio arquitectónico significativo. Según se documenta, este paradigma permite ejecutar código de forma nativa en el servidor dentro de los componentes React, utilizando async/await para la obtención de datos, lo que elimina la necesidad de métodos específicos antiguos como getServerSideProps y resulta en un código más legible y "estético" para el desarrollador. Esta arquitectura, descrita como "coqueta" por algunos autores, fomenta una clara separación entre la lógica del servidor y del cliente, marcada por las directivas 'use server' y 'use client' . De acuerdo con lo expuesto en la literatura, este modelo no solo mejora la organización del código, sino

que también tiene implicaciones directas en el SEO, ya que el contenido se sirve completamente renderizado desde el servidor, siendo más fácilmente indexable por los motores de búsqueda. Por lo tanto, la metodología a emplear en un proyecto con Next.js moderno debe considerar esta división de responsabilidades desde su concepción, planificando qué componentes y lógica residirán en el servidor y cuáles en el cliente para optimizar tanto el rendimiento como la mantenibilidad.

4.10.2 Supabase

En el contexto general de las tecnologías Backend modernas, la elección de una plataforma de Backend-como-Servicio (BaaS) se plantea como una solución metodológica para abordar problemas recurrentes de velocidad de desarrollo, escalabilidad y complejidad de infraestructura. Estas plataformas abstraen la configuración de servicios fundamentales como bases de datos, autenticación y APIs, permitiendo a los desarrolladores concentrarse en la lógica del producto principal. En base a lo que dijo (Lorenz, 2024) dentro de este panorama, Supabase se erige como una alternativa de código abierto, descrita a menudo como la contraparte de Firebase, que combina una base de datos PostgreSQL, autenticación, APIs instantáneas, suscripciones en tiempo

real, almacenamiento y funciones de borde en una plataforma unificada. Según la documentación oficial, Supabase es definida como "la plataforma de desarrollo Postgres", proporcionando una base de datos completa para cada proyecto junto con funcionalidades de realtime, copias de seguridad y extensiones. Citando a Guamian (2025), el principal atractivo de Supabase reside en su capacidad para "acelerar los ciclos de desarrollo", un factor crítico en entornos donde el time-to-market es crucial, permitiendo a los equipos "construir en un fin de semana y escalar a millones".

Un planteamiento de problema específico en el desarrollo de aplicaciones web complejas es la gestión segura y eficiente de la autenticación de usuarios y el control de acceso a los datos. Aquí, la metodología propuesta por Supabase ofrece soluciones

concretas a través de su sistema de Autenticación y Seguridad a Nivel de Fila (RLS). De acuerdo con la documentación, Supabase Auth permite gestionar inicios de sesión por email y contraseña, proveedores OAuth sociales y magic links, integrando estas funcionalidades a través de una suite de APIs . Un aspecto metodológico crítico es la configuración correcta de las URLs de redirección, las cuales deben coincidir con una lista predefinida en el proyecto para que los flujos de autenticación social o sin contraseña funcionen correctamente. Se observa que la URL del sitio (Site URL) define la redirección por defecto, y se pueden añadir URLs adicionales, incluso utilizando comodines para entornos de desarrollo o previsualización. Esta configuración es vital, ya que, como se discute en la comunidad, la imposibilidad de listar múltiples Site URLs de forma nativa puede plantear un desafío operativo para proyectos con múltiples entornos (desarrollo, staging, producción), sugiriendo metodologías de trabajo alternativas como el uso de proyectos clonados o patrones de comodines en las URLs de redirección adicionales.

Finalmente, aterrizando en el componente más técnico y profundo, la metodología de Supabase para la interacción con datos se sustenta en su núcleo: una base de datos PostgreSQL de código abierto y de alto desempeño. La plataforma extiende la funcionalidad estándar de PostgreSQL con características como el servidor de tiempo real y una vasta colección de extensiones. Por ejemplo, la extensión HypoPG permite una metodología de optimización hipotética, donde se pueden crear índices virtuales para evaluar su impacto en el plan de ejecución de una consulta sin incurrir en el costo de recursos de construirlos físicamente, una herramienta valiosa para la formulación de hipótesis sobre optimización de rendimiento. Esta potente base, combinada con las APIs instantáneas generadas automáticamente a partir del esquema de la base de datos, proporciona una metodología eficiente para el desarrollo del Backend. En base a lo analizado por Guamian (2025), la base PostgreSQL brinda ventajas para cargas de trabajo modernas, como el soporte para tipos de datos JSON y extensiones como pgvector para embeddings de IA, haciendo de Supabase una opción sólida para aplicaciones que requieren manejar estructuras de datos complejas.

4.10.3 Git

En el contexto general del desarrollo de software moderno, la adopción de un sistema de control de versiones robusto se plantea como una solución metodológica indispensable para abordar

problemas fundamentales de colaboración, trazabilidad y mantenimiento del código fuente. Según (Loeliger, 2009) el sistema Git permite gestionar las distintas actualizaciones y modificaciones realizadas en el código, registrando los cambios en una línea de tiempo y facilitando el regreso a versiones anteriores si se presentan errores. Este planteamiento resuelve un problema central en proyectos de múltiples desarrolladores: la necesidad de coordinar contribuciones paralelas sin sobrescribir el trabajo de los demás. De acuerdo con la documentación de GitHub, un repositorio es el elemento más básico de la plataforma, un lugar donde se almacenan el código, los archivos y el historial de revisiones de cada archivo, y que puede tener múltiples colaboradores. Por lo tanto, la hipótesis subyacente es que la implementación de un flujo de trabajo basado en Git es un requisito metodológico previo para cualquier proyecto que busque escalabilidad y colaboración eficiente.

Desde un punto de vista analítico, la metodología de trabajo con Git se estructura en torno a conceptos específicos como ramas (branches), confirmaciones (commits) y fusiones (merges). De acuerdo con lo expuesto, una arquitectura de ramificación permite a los equipos crear versiones paralelas del código que no afectan la rama principal, trabajando en nuevas funcionalidades o correcciones de manera aislada. Un flujo de trabajo común es utilizar una rama principal de desarrollo (generalmente llamada main) y ramas secundarias destinadas a desarrollo o pruebas. Citando a la documentación, cuando el trabajo en una rama secundaria está finalizado, este se integra en la rama principal mediante un proceso de fusión (git merge), pasando a formar parte del código base estable. Esta metodología de ramificación y fusión es crucial para la formulación de hipótesis de desarrollo, ya que permite experimentar con nuevas características o enfoques sin riesgo de comprometer la estabilidad del proyecto principal. Además, el ciclo de confirmación de cambios (commit) en Git involucra un directorio de trabajo, un

área de preparación (staging area) y el repositorio local, proporcionando un control granular sobre qué cambios se registran en el historial.

Finalmente, la integración de Git con plataformas de hospedaje de repositorios remotos constituye la capa operativa específica de la metodología. Un repositorio Git existe, como mínimo, en dos ubicaciones: una copia local en el ordenador del desarrollador y una copia remota en un servidor, como GitHub, GitLab o Bitbucket. La metodología para iniciar un proyecto implica sincronizar estos entornos. Según la guía, es posible crear un nuevo repositorio localmente y luego subirlo (push) al servidor, o clonar (clone) un repositorio remoto existente para obtener una copia local de todo el árbol de archivos del proyecto. Un problema técnico común que se observa al comenzar un proyecto es el error que surge al intentar sincronizar un repositorio local con uno remoto que tiene un historial de confirmaciones diferente, lo que evidencia la importancia de una metodología de inicialización coherente. En base a lo analizado por los expertos, la autenticación reforzada en estas plataformas (como el uso de Fine-grained tokens en GitHub con permisos mínimos específicos y caducidad corta) es ahora un componente metodológico de seguridad obligatorio para gestionar los repositorios de forma segura. Por lo tanto, la metodología a emplear en un proyecto debe considerar no solo el flujo de trabajo de Git, sino también la configuración segura y la sincronización con el repositorio remoto elegido.

4.10.4 Vercel

Desde una perspectiva general y con base en lo que dijo (Riva, 2022), Vercel se define como una plataforma de despliegue que facilita el despliegue continuo (Continuous Deployment) para aplicaciones web modernas. La metodología central que plantea se basa en la integración directa con repositorios Git (como GitHub, GitLab o Bitbucket), donde cada confirmación de código (commit) o solicitud de extracción (pull request) puede desencadenar automáticamente un nuevo despliegue. Este enfoque

aborda el problema fundamental de la automatización y rapidez en la entrega de software, permitiendo a los equipos de desarrollo implementar cambios de manera frecuente y confiable. La plataforma estructura este proceso en tres entornos (environments) por defecto: Desarrollo Local (Local), Vista Previa (Preview) y Producción (Production), cada uno con un propósito distinto en el ciclo de vida del desarrollo. Según la documentación oficial, el entorno de Vista Previa es particularmente crucial para la metodología de pruebas y colaboración, ya que permite desplegar cambios para realizar pruebas de control de calidad (QA) sin afectar al sitio en vivo, generando automáticamente una URL única para cada despliegue. Citando a los desarrolladores de Vercel, este modelo de entornos múltiples es esencial para "desplegar para realizar más pruebas, control de calidad o colaboración sin impactar en su sitio en vivo".

En un nivel de especificidad técnica mayor, la configuración y gestión de estos entornos se convierte en un aspecto metodológico crítico. Cada entorno en Vercel puede definir sus propias variables de entorno únicas, lo que permite manejar configuraciones sensibles (como claves de API o conexiones a bases de datos) de forma aislada y segura para cada etapa del proyecto. Para proyectos que requieren flujos de trabajo más especializados, los planes Pro y Enterprise permiten la creación de Entornos Personalizados (por ejemplo, para staging o QA), los cuales pueden asociarse a ramas específicas del repositorio y contar con dominios persistentes. La metodología de despliegue también admite vías alternativas, como el uso de la CLI de Vercel para integraciones en pipelines CI/CD personalizadas o el uso de Deploy Hooks, que son URLs únicas que permiten activar un despliegue sin necesidad de un nuevo commit, útil para integraciones con servicios de terceros. De acuerdo con lo observado en la documentación, la gestión de los despliegues desde el panel de control permite a los desarrolladores filtrar, reimplementar (redeploy) o promover manualmente un despliegue de vista previa a producción, lo que proporciona control sobre el proceso de entrega. Esta capacidad de promoción manual es un mecanismo de seguridad que evita que cambios no verificados lleguen a los usuarios finales de forma automática.

Finalmente, a nivel operativo y de optimización, es imperativo comprender los límites (limits) y las configuraciones que impactan el rendimiento y el costo. Vercel

establece límites claros según el plan (Hobby, Pro, Enterprise), los cuales abarcan desde el número de despliegues por día y el tiempo de construcción (build time), hasta recursos como el tamaño de los archivos estáticos y la memoria disponible en el contenedor de construcción. Un recurso clave para la metodología de optimización es la página de Uso (Usage) en el panel de control, que permite monitorear métricas de consumo como Transferencia de Datos, Invocaciones de Funciones Serverless y Optimización de Imágenes, facilitando la identificación de cuellos de botella y la proyección de costos. En particular, para las Vercel Functions (funciones serverless), es crucial configurar parámetros como la duración máxima (maxDuration) y la región de ejecución para equilibrar el rendimiento, la latencia y el costo. Según se analiza, la duración máxima por defecto para una función en el plan Hobby es de 10 segundos, configurable hasta un máximo de 60 segundos, mientras que en los planes superiores estos límites se amplían significativamente. La elección de la región, por defecto Washington D.C. (iad1), es otra decisión metodológica importante para reducir la latencia cuando las funciones acceden a fuentes de datos externas. La plataforma también impone límites técnicos estrictos, como un tamaño máximo de carga útil (payload) de 4.5 MB para las funciones, más allá del cual se devuelve un error. La comprensión y configuración adecuada de estos límites y optimizaciones es, por lo tanto, fundamental en la formulación de una metodología de despliegue robusta y eficiente en costos.

4.11 Interfaz de usuario y experiencia (UX/UI)

Según recientes estudios, el diseño de interfaz de usuario (UI) y de experiencia de usuario (UX) representan componentes fundamentales en el desarrollo de productos digitales, ya que determinan la forma en que los usuarios interactúan con sistemas y cómo perciben su usabilidad, satisfacción y eficiencia. En ese sentido, se entiende que UI —como elementos visuales, disposición de componentes, tipografías, botones y flujo de interacción— conforma la puerta de entrada al usuario; mientras que UX abarca la totalidad de la experiencia: la facilidad de uso, la accesibilidad, la navegación, la satisfacción emocional y la eficiencia al cumplir tareas.

De acuerdo con lo observado en investigaciones recientes, una interfaz con alta usabilidad y un diseño UX adecuado tiene un impacto directo en el rendimiento de la aplicación y en la satisfacción del usuario. Por ejemplo, un estudio que combinó evaluación experta con encuestas a usuarios muestra que mediante un marco integrado de evaluación UI/UX se pueden identificar problemáticas de control, compromiso y cumplimiento de objetivos, arrojando valores por encima de estándares de usabilidad. Otro trabajo comparativo sobre usabilidad y experiencia de usuario en software revela que interfaces bien diseñadas aumentan la efectividad y satisfacción de los usuarios, lo que a su vez mejora la aceptación del sistema.

Por lo tanto, para un proyecto de aplicación web, el planteamiento del problema, la formulación de hipótesis y la metodología deben considerar UI/UX como variables centrales. Es decir, la hipótesis podría plantear que “un diseño UI/UX óptimo mejora significativamente la usabilidad, la satisfacción y la eficiencia de los usuarios al interactuar con la aplicación”. La metodología debería incluir —como han hecho otros estudios— técnicas como diseño centrado en usuario, prototipos, pruebas de usabilidad, encuestas y evaluación heurística, de modo que se pueda validar empíricamente la relación entre interfaz/experiencia y resultados funcionales o perceptuales. Esto garantizara que los hallazgos estén respaldados por evidencias medibles y replicables, tal como recomiendan los artículos revisados.

4.11.1 Accesibilidad y usabilidad

De acuerdo con lo que señalan diversos autores e investigaciones (Initiative (WAI), 2023), la accesibilidad web se define como la capacidad de un sitio o aplicación para ser utilizado por el mayor número posible de personas, independientemente de sus habilidades, limitaciones físicas, cognitivas o contextuales (por ejemplo, tipo de dispositivo, entorno, experiencia previa). Por su parte, la usabilidad se refiere al grado en que un producto puede ser utilizado por usuarios específicos para alcanzar objetivos concretos con eficacia, eficiencia y satisfacción en un contexto determinado. En ese sentido, ambos conceptos —usabilidad y accesibilidad— están estrechamente relacionados y, de acuerdo con algunos autores, un diseño accesible generalmente

favorece la usabilidad para un espectro más amplio de usuarios, incluidas personas con discapacidad.

En estudios recientes que analizan aplicaciones web desde la perspectiva del diseño universal se ha observado que implementar principios de accesibilidad mejora significativamente la experiencia de uso. Por ejemplo, en una investigación sobre interfaces de sitios para compartir apuntes estudiantiles, al comparar una versión estándar con otra diseñada bajo criterios de diseño universal, se emplearon métodos como cuestionario SUS, seguimiento ocular (eye-tracking) y evaluaciones automáticas de accesibilidad; los resultados mostraron que la versión accesible incrementó la usabilidad y la accesibilidad de forma estadísticamente significativa. En otro estudio que revisa la accesibilidad en sitios universitarios, se identificaron deficiencias frecuentes en navegación, compatibilidad con lectores de pantalla y adaptación a distintos perfiles de usuario, lo que evidencia la necesidad de considerar accesibilidad desde las fases iniciales de diseño.

De lo anterior se desprende que, para un proyecto de aplicación web, es adecuado plantear como hipótesis que “la implementación de criterios de accesibilidad y usabilidad mejora la eficacia, eficiencia y satisfacción de los usuarios al interactuar con la aplicación”. A su vez, la metodología debería contemplar la inclusión de estándares reconocidos (por ejemplo, las normas del World Wide Web Consortium — W3C — para accesibilidad), junto con pruebas empíricas de usabilidad: uso de validadores automáticos, pruebas con usuarios reales (incluyendo personas con discapacidad), cuestionarios de satisfacción y posiblemente seguimiento de interacción (por ejemplo, eye-tracking) para evaluar eficacia y eficiencia. Esta estrategia metodológica permitiría obtener evidencias objetivas y replicables sobre la incidencia de accesibilidad/usabilidad en la calidad de la aplicación.

4.12 Integración de APIs y gestión de datos externos

Según lo expuesto en la revisión de literatura reciente, las interfaces de programación de aplicaciones (APIs) han emergido como elementos fundamentales en los ecosistemas de desarrollo modernos, permitiendo la interoperabilidad entre sistemas

heterogéneos y facilitando el acceso, intercambio y gestión de datos externos de manera eficiente. Por ejemplo, en el análisis sobre los “ecosistemas de APIs web” se argumenta que dichas APIs actúan como la columna vertebral de una economía digital, donde múltiples actores, servicios y plataformas interactúan a través de APIs para habilitar nuevas funcionalidades y sinergias. (Google, 2023)De ese modo, la integración de APIs no solo sirve para comunicar diferentes componentes de un sistema, sino también para conectar con fuentes de datos externas — lo que amplía el alcance funcional y de datos de la aplicación web.

No obstante, esta flexibilidad y expansión funcional trae consigo desafíos en cuanto a la calidad, seguridad y gobernanza de los datos. Investigaciones sobre la calidad de las APIs han demostrado que, en tiempo de ejecución, métricas como disponibilidad, latencia y preferencias de seguridad por parte del proveedor pueden variar considerablemente dependiendo de factores geográficos y temporales, lo que impacta directamente en la estabilidad y desempeño de la aplicación que consume dichas APIs. Además, la exposición de APIs a entornos distribuidos o de múltiples fuentes demanda mecanismos de gestión apropiados — tales como gateways de API, autenticación, autorización, control de acceso, cifrado y auditoría — para mitigar riesgos como fugas de datos, accesos no autorizados o exposición excesiva de información.

De lo anterior se deduce que, para un proyecto de aplicación web que contemple integración de APIs y manejo de datos externos, resulta pertinente plantear como hipótesis que “una adecuada arquitectura de integración de APIs — junto con un sistema de gestión y gobernanza de datos externos bien diseñado — permitirá asegurar la disponibilidad, confiabilidad y seguridad de los datos consumidos, mejorando la robustez funcional del sistema”. Asimismo, la metodología a emplear debería incluir etapas de diseño arquitectónico, selección e implementación de APIs (o diseño interno de endpoints), evaluación de calidad y desempeño (disponibilidad, latencia, tolerancia a errores) y pruebas de seguridad (autenticación, autorización, cifrado, control de acceso, auditoría). De esta forma, se garantizaría que la integración de datos externos se haga de manera controlada, estable y segura, contribuyendo a la validez y confiabilidad de los resultados del proyecto.

4.12.1 Consumo de APIs meteorológicas y manejo de claves

Según lo que se ha documentado en la literatura sobre gestión de APIs, las APIs se constituyen como un mecanismo esencial para habilitar la comunicación entre sistemas heterogéneos, permitiendo que una aplicación web consuma datos externos (como datos meteorológicos) sin necesidad de desarrollar toda la infraestructura de datos desde cero.. En ese sentido, mediante una API meteorológica la aplicación puede solicitar información de clima, pronósticos e históricos — lo que expande su funcionalidad y valor agregado — sin asumir la carga de mantener estaciones propias o bases propias de datos. Por ejemplo, entre las APIs ampliamente utilizadas para este fin se mencionan OpenWeatherMap u otras similares, las cuales proveen datos climatológicos de ubicaciones geográficas específicas.

No obstante, los estudios técnicos advierten que el uso de APIs externas — en particular aquellas que requieren claves de acceso (API keys) — implica riesgos de seguridad y gobernanza de datos, lo que demanda una gestión cuidadosa de las credenciales. Se observa que usar una API key como único mecanismo de autenticación tiene vulnerabilidades inherentes: las claves frecuentemente se almacenan en texto plano, se pueden exponer accidentalmente (por ejemplo, al publicarse el código o al incluirlas en el cliente), y carecen de controles granulares por defecto. En consecuencia, la literatura recomienda buenas prácticas como: almacenar las claves fuera del código (variables de entorno o sistemas de gestión de secretos), implementar restricciones de uso, rotación periódica y monitoreo de uso.

En base a lo anterior, para un proyecto que consuma una API meteorológica puede plantearse la hipótesis de que “la adecuada gestión de API keys (almacenamiento seguro, rotación, uso de entorno servidor, etc.) junto con el consumo de datos meteorológicos externos garantiza la disponibilidad y seguridad del servicio, reduciendo riesgos de exposición y abuso de datos”. Asimismo, la metodología propuesta debería incluir —además del desarrollo funcional de integración de la API— una fase de diseño de seguridad: configuración de variables de entorno, uso de servidor backend como intermediario (no exponer la clave en frontend), implementación de limitaciones (rate-

limiting), monitoreo de uso y pruebas de vulnerabilidades. De esta forma se aseguraría que el consumo de datos meteorológicos sea funcional, eficaz y seguro, y que los resultados del proyecto sean confiables y reproducibles.

4.12.2 Ingreso manual de datos: diseño y validaciones

De acuerdo con lo que señalan diversos autores (Roselin & Rodrigues, 2017), la entrada manual de datos mediante formularios sigue siendo una práctica común en aplicaciones web cuando no es posible automatizar la recolección de información, lo que hace indispensable un diseño de formulario cuidadoso. En este sentido, se considera esencial que los formularios sean diseñados con controles rigurosos de validación desde el momento del ingreso de datos — incluyendo validaciones de obligatoriedad, tipo de dato, formato, rangos admitidos, y longitud de campos — de modo que se garantice la integridad, coherencia y calidad de los datos desde su captura. (véase guía de validación de formularios en HTML5 y buenas prácticas de desarrollo web).

Por otro lado, estudios empíricos indican que la forma en que se realiza la entrada de datos puede afectar significativamente la calidad de los datos almacenados. Por ejemplo, en un análisis sobre ingreso manual simple vs. ingreso con doble verificación ("double-key entry") frente a procesamiento automatizado se observó que la doble entrada manual ofreció una tasa de error considerablemente menor que la entrada simple, lo que sugiere que los errores derivados del ingreso manual pueden comprometer la validez del conjunto de datos si no se aplican controles adecuados. Asimismo, cuando no se implementan validaciones eficaces en los formularios web, la información recogida puede volverse inútil, con registros incompletos, inconsistentes o erróneos, lo que afecta la usabilidad del sistema y la confiabilidad de los resultados.

En consecuencia —para un proyecto que utilice ingreso manual de datos en una aplicación web— se justifica plantear la hipótesis de que "un formulario diseñado con validaciones adecuadas (tipo, formato, obligatoriedad, consistencia) reducirá significativamente la cantidad de errores de entrada, mejorando la calidad, integridad y utilidad de los datos almacenados". En cuanto a la metodología, resulta apropiado que se incluya una fase de diseño de formulario con validaciones tanto en cliente como en

servidor, pruebas de usabilidad del formulario (para asegurar que los usuarios entienden y usan correctamente los controles), y, si es viable, un mecanismo de doble verificación o revisión (o auditoría de los datos ingresados) para detectar errores de transcripción manual.

4.13 Seguridad, privacidad y consideraciones legales

De acuerdo con lo que señalan los investigadores en el área de desarrollo de software y sistemas web (Kitsios et al., 2023), las aplicaciones web deben diseñarse contemplando desde el inicio aspectos de seguridad y privacidad, ya que el entorno de Internet presenta múltiples riesgos inherentes: fallas de seguridad, exposición de datos personales, accesos no autorizados, fugas de información, entre otros. En consecuencia, la incorporación deliberada de mecanismos técnicos — controles de acceso, cifrado, validación de entradas, mantenimiento de integridad y disponibilidad de datos — se considera fundamental para reducir la superficie de ataque y proteger los datos manejados por la aplicación. Este enfoque asegura que los procesos de autenticación, autorización, transmisión segura y almacenamiento respeten principios de seguridad informática reconocidos.

Por otra parte, el tratamiento de datos personales en aplicaciones web implica también un compromiso con la privacidad del usuario, entendido no solo como un aspecto técnico sino como un derecho fundamental y una obligación legal. Según lo expuesto por algunos autores, la privacidad en línea debe abordarse desde una perspectiva de “privacy by design”: es decir, integrar la protección de datos personales desde las etapas iniciales del diseño arquitectónico y funcional del sistema, no como un añadido posterior. En este sentido, cualquier recolección, almacenamiento o procesamiento de datos sensibles requiere de políticas claras, transparencia, consentimiento informado del usuario y medidas técnicas que garanticen que esos datos no se utilicen para fines distintos a los pactados

Además, la literatura revisada indica que las regulaciones legales y normativas de protección de datos juegan un papel determinante cuando la aplicación web opera en contextos donde los datos personales están protegidos por ley. Al respecto, algunos estudios advierten que muchas implementaciones dejan de lado estos marcos normativos, enfocándose únicamente en aspectos técnicos sin asegurar la legalidad del tratamiento de datos. (Autor anónimo, revisión sobre gestión de privacidad en LATAM, 2024). Por ello, se considera necesario que un proyecto que maneje datos

personales examine los requisitos legales aplicables — leyes nacionales o regionales de protección de datos, normas de seguridad, regulaciones sobre privacidad — y los incorpore en su diseño, para garantizar que el tratamiento sea conforme a derecho y respetuoso de los derechos de usuarios.

En consecuencia, para una investigación o desarrollo de aplicación web que incluya estas dimensiones, resulta apropiado plantear como hipótesis que: “La integración de mecanismos de seguridad informática, políticas de privacidad y cumplimiento de normativas legales garantiza la protección de datos personales, reduce vulnerabilidades de seguridad y asegura la conformidad legal del sistema.” A su vez, la metodología a emplear debería incluir —además del desarrollo técnico— una etapa de análisis legal y normativo, diseño con enfoque “privacy by design” y “security by design”, implementación de controles técnicos (cifrado, autenticación, control de acceso, auditoría), pruebas de seguridad (vulnerabilidades comunes, exposición de datos personales, fugas, ataques de inyección, control de accesos) y una evaluación documental del cumplimiento normativo (política de privacidad, consentimiento informado, registro de operaciones).

Este planteamiento teórico sustenta la necesidad de considerar seguridad, privacidad y aspectos legales desde las primeras fases del proyecto, no como complemento, sino como parte esencial del diseño arquitectónico y de la metodología de desarrollo — de modo que los resultados sean válidos, confiables, seguros y respetuosos de los derechos de los usuarios.

V. MATERIALES Y MÉTODOS.

5.1 Análisis de requerimientos

•Levantar y documentar las necesidades de los usuarios objetivo: técnicos, productores y estudiantes agrícolas.

•Identificar los parámetros agronómicos esenciales: tipo de cultivo, etapa fenológica, características del suelo, profundidad radicular, evapotranspiración, precipitación y temperatura.

•Definir los flujos de usuario: ingreso de datos, visualización de resultados, consulta de historial y exportación.

5.2 Diseño de la arquitectura

- Seleccionar Next.js como framework principal para el desarrollo del frontend y backend en un entorno unificado.
- Utilizar Supabase para autenticación, base de datos (PostgreSQL) y almacenamiento.
- Establecer la estructura general de la aplicación: componentes de interfaz, API routes, módulos de cálculo agronómico y visualización de resultados.
- Definir el modelo de datos necesario para almacenar cultivos, suelos, usuarios, registros meteorológicos e historiales de cálculos.

5.3 Planificación inicial

- Crear un tablero Kanban en GitHub Projects para organizar y priorizar las tareas.
- Definir categorías iniciales: Pendiente, En progreso, En revisión y Completado.
- Registrar épicas relacionadas con los objetivos específicos: algoritmos agronómicos, UI/UX, integración de datos meteorológicos, base de datos y validación.

5.4 Fase de Desarrollo

- Durante esta fase se implementarán los componentes del sistema utilizando una metodología de trabajo visual y continúa basada en Kanban.

5.5 Desarrollo del Frontend (Next.js)

- Construcción de la interfaz para ingreso de datos: selección del cultivo, etapa fenológica, tipo de suelo y variables climáticas.
- Implementación de formularios dinámicos con validación de datos.

- Diseño de pantallas de resultados que muestren: lámina neta, lámina bruta, calendario de riegos y gráficos.
- Optimización de la experiencia de usuario para facilitar la comprensión de los resultados agronómicos.

5.6 Desarrollo del Backend (Next.js API routes y Supabase)

- Creación de los algoritmos agronómicos para el cálculo de lámina de riego (ETc, Kc, eficiencia de riego, etc.).
- Implementación de API routes en Next.js para procesar cálculos y gestionar la comunicación con Supabase.
- Modelado y creación de tablas en Supabase: cultivos, suelos, usuarios, estaciones climáticas, historial de cálculos.
- Integración de autenticación y manejo seguro de sesiones.

5.7 Integración de datos meteorológicos

- Habilitar ingreso manual de variables climáticas por parte del usuario.
- (Opcional) Integrar API externas para datos de ET0, temperatura o precipitación.
- Diseñar un módulo para convertir datos meteorológicos en valores útiles para el cálculo (por ejemplo, cálculo ET0 mediante Penman-Monteith si se implementara).

5.8 Control de versiones (Git)

- Uso de GitHub para administrar ramas según las tareas definidas en Kanban.
- Realización de commits frecuentes, revisión de código y aplicación de pull requests.
- Documentación de avances por cada cambio significativo, asegurando trazabilidad.

5.9 Pruebas y validación

- Ejecución de pruebas unitarias en los cálculos agronómicos para validar fórmulas y coherencia con la literatura técnica.
- Validación de la interfaz: pruebas de usabilidad, tiempos de carga y accesibilidad.
- Comparación de resultados con documentos y manuales agronómicos para asegurar precisión.

5.10 Despliegue y monitoreo

- Despliegue continuo mediante Vercel, integrado con GitHub.
- Monitoreo del ambiente productivo: rendimiento, errores, registros de cálculo.
- Documentación de incidencias y actualización del flujo de trabajo en Kanban.

5.11 Control y Documentación del Proceso

- Tablero Kanban: se registrará el estado de cada tarea mediante tarjetas y comentarios, documentando el progreso con capturas y reportes generados en GitHub Projects.
- Repositorio en GitHub: se conservará el historial completo de commits, ramas, revisiones y versiones desplegadas.
- Supabase (PostgreSQL): se mantendrán los registros de configuración de tablas, logs del sistema y evidencias de pruebas de integración.
- Vercel: se documentarán despliegues y métricas de rendimiento para asegurar la estabilidad del sistema.}

VI. ANÁLISIS DE RESULTADOS Y DISCUSIÓN.

En este capítulo se presentan y analizan los resultados obtenidos tras la aplicación del modelo optimizado para el cálculo de la lámina de riego. A través del procesamiento de datos climáticos y edafológicos, se exponen las variaciones observadas entre el método tradicional y la propuesta de optimización. Los hallazgos aquí descritos permiten evaluar la precisión de los ajustes realizados y su relevancia en la determinación de las necesidades hídricas reales del cultivo, estableciendo la base para la discusión sobre el ahorro de agua y la eficiencia energética en el sistema.

6.1 Requerimientos Funcionales (RF)

Lista de funciones que el sistema debe realizar.

6.1.1 RF1. Gestión de usuarios

- El sistema debe permitir el registro de usuarios mediante correo y contraseña.
- El sistema debe permitir el inicio de sesión y cierre de sesión.
- El sistema debe mantener sesiones activas mediante autenticación provista por Supabase.
- El usuario debe poder consultar su historial de cálculos guardados.

6.1.2 RF2. Ingreso de información agronómica

- El sistema debe permitir seleccionar el tipo de cultivo desde un catálogo predefinido.
- El sistema debe permitir seleccionar la etapa fenológica del cultivo.

- El sistema debe permitir seleccionar el tipo de suelo.
- El sistema debe permitir ingresar variables climáticas manualmente (ET0, precipitación, temperatura, humedad relativa, etc.).
- El sistema debe validar que los valores ingresados tengan un formato y rango de datos aceptable.

6.1.3 RF3. Base de datos y almacenamiento

- El sistema debe almacenar la información de cultivos, suelos y parámetros agronómicos en una base de datos Supabase (PostgreSQL).
- El sistema debe registrar el historial de cálculos asociados a cada usuario.
- El sistema debe guardar configuraciones o valores predefinidos utilizados para los algoritmos.

6.1.4 RF4. Cálculos agronómicos

- El sistema debe calcular la evapotranspiración del cultivo (ETc) según Kc y ET0.
- El sistema debe calcular la lámina neta (Ln).
- El sistema debe calcular la lámina bruta (Lb) considerando la eficiencia del riego.
- El sistema debe mostrar de manera clara los resultados en pantalla.
- El sistema debe permitir descargar o visualizar nuevamente cálculos anteriores.

6.1.5 RF5. Integración de datos meteorológicos

- El sistema debe permitir ingresar manualmente datos meteorológicos cuando no exista conexión a API.
- (Opcional) El sistema debe conectarse a una API meteorológica externa para obtener ET0, temperatura y precipitación.
- El sistema debe manejar errores de conexión y mostrar mensajes claros cuando la API falle.

6.1.6 RF6. Interfaz de usuario (Frontend Next.js)

- El sistema debe presentar formularios intuitivos y segmentados por categorías (cultivo, suelo, clima).
- El sistema debe permitir que el usuario navegue entre pantallas de manera clara y eficiente.
- El sistema debe mostrar mensajes de error y éxito en tiempo real.
- El sistema debe incluir gráficos o tablas opcionales para visualizar resultados.

6.1.7 RF7. Gestión del sistema

- El sistema debe permitir a los administradores agregar, modificar o eliminar cultivos, suelos o valores Kc (si se contempla un modo administrador).
- El sistema debe registrar logs básicos de uso y de errores para fines de monitoreo.

6.2 Requerimientos No Funcionales (RNF) Características de desempeño, calidad y operación.

RNF1. Rendimiento

El sistema debe realizar los cálculos en menos de 2 segundos.

El sistema debe cargar la interfaz principal en un tiempo máximo de 3 segundos en conexión estándar móvil.

RNF2. Seguridad

La información de usuarios debe manejarse mediante protocolos seguros.

Las contraseñas deben almacenarse cifradas a través de Supabase Auth.

El sistema debe evitar inyecciones SQL mediante el uso de consultas seguras.

RNF3. Usabilidad

El sistema debe ser fácil de usar por usuarios con conocimientos básicos del ámbito agrícola.

La interfaz debe ser responsiva y accesible desde dispositivos móviles.

6.2.1 RNF1. Rendimiento

- El sistema debe realizar los cálculos en menos de 2 segundos.
- El sistema debe cargar la interfaz principal en un tiempo máximo de 3 segundos en conexión estándar móvil.

6.2.2 RNF2. Seguridad

- La información de usuarios debe manejarse mediante protocolos seguros.
- Las contraseñas deben almacenarse cifradas a través de Supabase Auth.
- El sistema debe evitar inyecciones SQL mediante el uso de consultas seguras.

6.2.3 RNF3. Usabilidad

- El sistema debe ser fácil de usar por usuarios con conocimientos básicos del ámbito agrícola.
- La interfaz debe ser responsiva y accesible desde dispositivos móviles.
- Los formularios deben incluir validaciones claras y mensajes comprensibles.

6.2.4 RNF4. Escalabilidad

- El sistema debe ser capaz de soportar un aumento en usuarios sin disminuir su rendimiento.
- Debe ser posible agregar nuevos cultivos o variables sin cambiar la estructura principal del sistema.

6.2.5 RNF5. Mantenibilidad

- El código debe estar organizado de forma modular (componentes, API routes, módulos de cálculo).
- El repositorio en GitHub debe mantener un control de versiones estructurado mediante ramas.
- Las funciones de cálculo deben estar documentadas para futuras actualizaciones.

6.2.6 RNF6. Disponibilidad

El sistema debe estar disponible en línea al menos un 95% del tiempo gracias al hosting en Vercel.

Las actualizaciones deben aplicarse mediante despliegues continuos sin afectar el acceso del usuario.

6.2.7 RNF7. Portabilidad

- El sistema debe ser accesible desde navegadores modernos (Chrome, Edge, Firefox, Safari).
- La vista móvil debe funcionar correctamente en Android y iOS.

6.3 Diseño de la base de datos

Para el diseño de la base de datos del sistema de cálculo de lámina de riego, se consideró la creación de las siguientes tablas principales: Profiles, Parcels y Calculations. Cada una cumple una función específica en la gestión de usuarios, parcelas de cultivo y registros de cálculo agronómico. Todas las tablas utilizan identificadores UUID generados automáticamente, y se implementan políticas de seguridad (RLS) para asegurar que cada usuario acceda únicamente a su propia información.

6.3.1 Tabla Profiles

La tabla Profiles almacena la información básica de cada usuario registrado en el sistema. Está vinculada de manera directa con el módulo de autenticación de Supabase, ya que utiliza el mismo identificador UUID generado al crear la cuenta.

Cada perfil incluye:

- id: Identificador único del usuario, asociado a auth.users.
- email: Correo electrónico del usuario.
- name: Nombre del usuario, extraído automáticamente de los metadatos durante el registro.
- created_at: Fecha y hora en que el perfil fue creado.

Esta tabla permite extender la información del usuario más allá de los datos de autenticación y es fundamental para la personalización del sistema. Incluye políticas que garantizan que el usuario pueda ver y editar exclusivamente su propio perfil.

6.3.2 Tabla Parcels

La tabla Parcels gestiona la información correspondiente a las parcelas registradas por cada usuario. Cada parcela representa una unidad agrícola para la cual se realizarán los cálculos de lámina de riego.

Los campos principales incluyen:

- id: Identificador único de la parcela.
- user_id: Relación foránea que vincula la parcela con el usuario propietario.
- name: Nombre asignado por el usuario a la parcela.
- area: Área de la parcela, útil para cálculos de volumen total de agua.
- sowing_date: Fecha de siembra, utilizada para determinar la etapa fenológica.
- crop (jsonb): Información del cultivo seleccionado (variedad, ciclo, etc.).
- soil_type (jsonb): Datos sobre el tipo de suelo, esenciales para el cálculo de lámina bruta.
- location (jsonb): Información de ubicación de la parcela (coordenadas, localidad).
- created_at: Fecha de registro de la parcela.

Esta tabla concentra los datos agronómicos estructurales necesarios para los cálculos y mantiene relación con la tabla Calculations, ya que cada proceso de cálculo debe estar asociado a una parcela.

6.3.3 Tabla Calculations

La tabla Calculations almacena los resultados generados por el sistema cada vez que un usuario realiza un cálculo de lámina de riego para una de sus parcelas. Cada registro guarda el estado agronómico y climático en el momento del cálculo.

Incluye los siguientes campos:

- id: Identificador único del cálculo.
- parcel_id: Clave foránea que relaciona el cálculo con una parcela.
- net_sheet: Lámina neta calculada (mm).

- daily_volume: Volumen diario de riego requerido según el área de la parcela (litros o m^3).
- irrigation_frequency: Frecuencia recomendada de riego (días).
- next_irrigation: Fecha sugerida para el próximo riego.
- crop_stage: Etapa fenológica del cultivo en el momento del cálculo.
- current_kc: Valor del coeficiente Kc utilizado en el cálculo.
- eto: Evapotranspiración de referencia ingresada o calculada.
- calculated_at: Fecha y hora en que se generó el cálculo.

Esta tabla funciona como un historial de decisiones agronómicas, permitiendo al usuario consultar cálculos anteriores y evaluar el comportamiento del riego a lo largo del tiempo.

Seguridad y políticas (RLS)

Para garantizar la integridad y privacidad de los datos:

- Las tres tablas (Profiles, Parcels, Calculations) tienen Row Level Security (RLS) habilitado.
- Se definieron políticas para que un usuario solo pueda:

...1 Ver y actualizar su propio perfil.

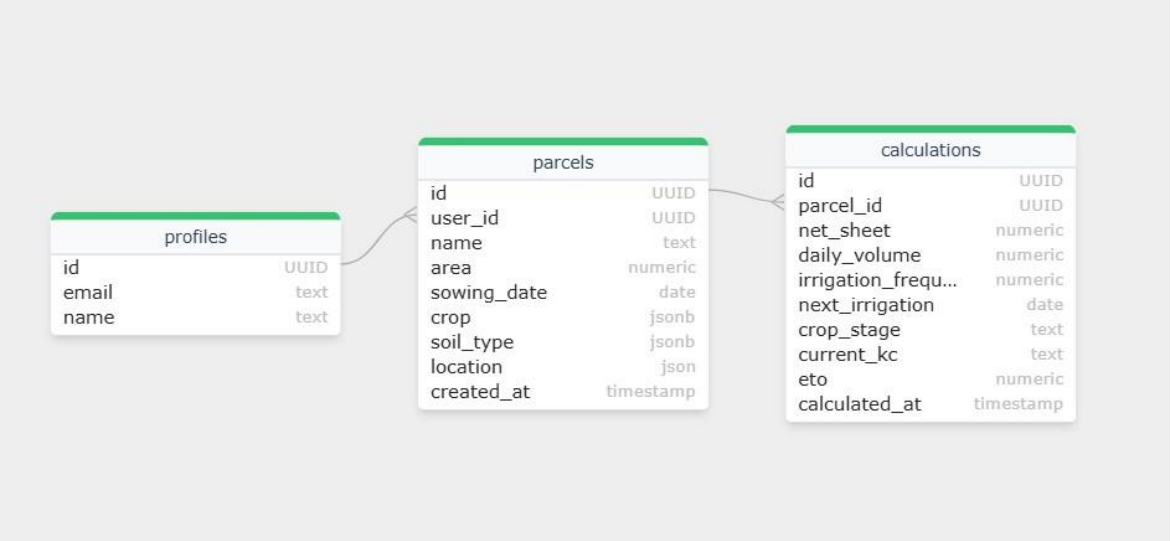
...2 Crear, modificar y eliminar únicamente sus propias parcelas.

...3 Insertar y consultar cálculos relacionados con sus parcelas.

Esto asegura un esquema seguro y multiusuario, fundamental para aplicaciones alojadas en la nube como Supabase.

Finalmente, se implementó un trigger que crea automáticamente un registro en Profiles cada vez que un nuevo usuario se registra mediante Supabase Auth.

Este mecanismo garantiza coherencia entre la autenticación y la base de datos, evitando errores o registros incompletos.

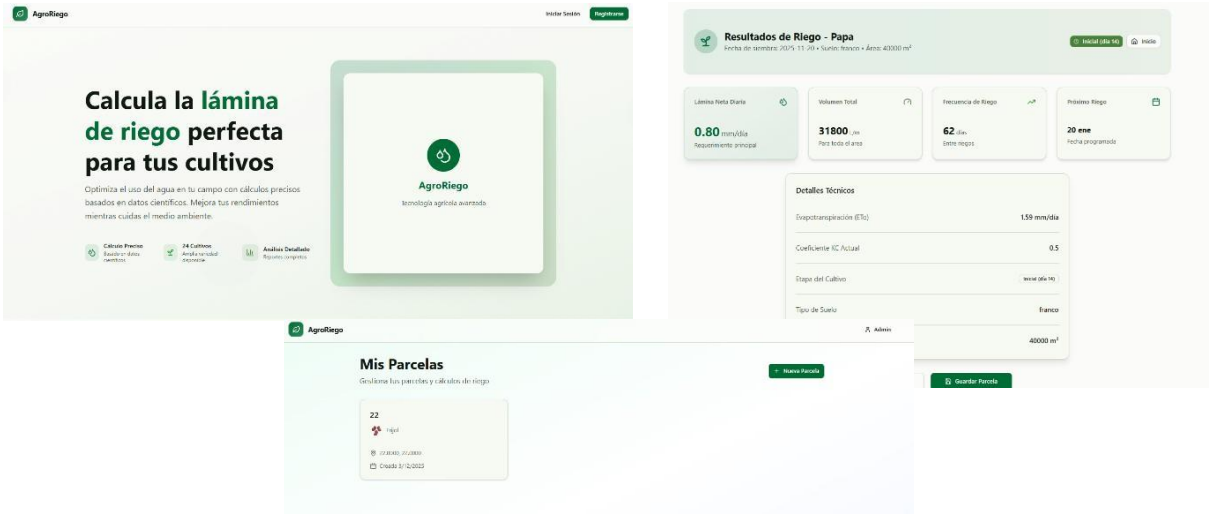


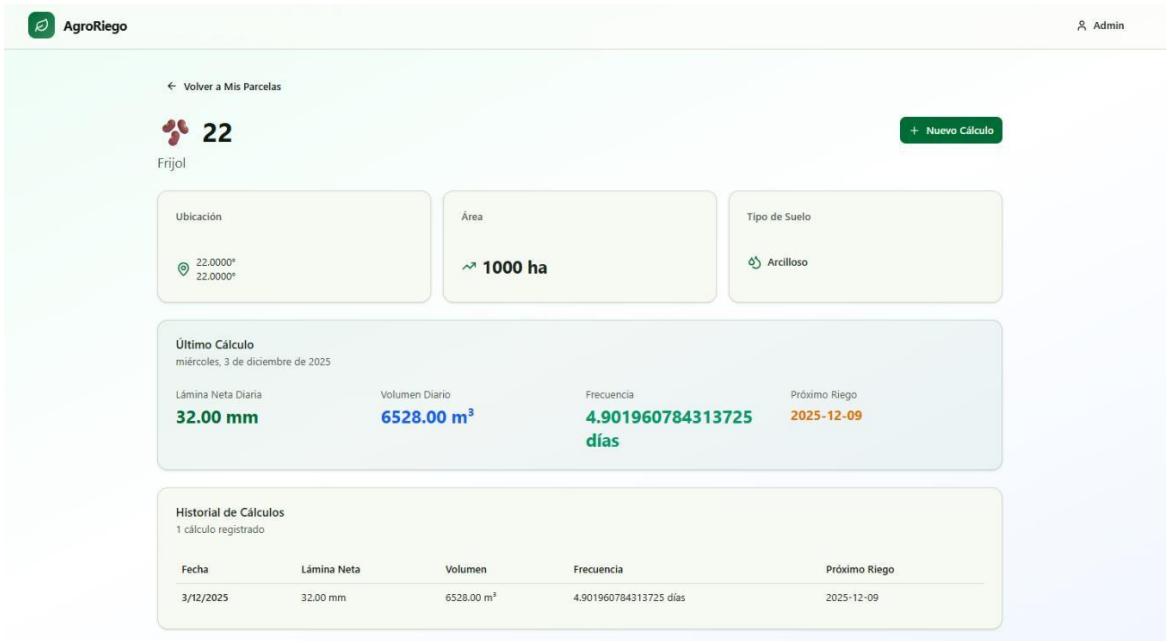
1 Figura 5.1 Diseño

6.4 Diseño de la página web

Para el diseño de la aplicación se utilizó V0 de Vercel como herramienta de apoyo para generar prototipos iniciales de las pantallas principales. Esta plataforma permitió obtener propuestas visuales rápidas basadas en instrucciones textuales y necesidades funcionales del sistema, lo que facilitó definir la estructura preliminar de algunas vistas,

como el formulario de ingreso de datos y ciertos componentes informativos del panel de riego.





2Figura 5.2 Diseño de pagina web

Si bien V0 fue de utilidad para acelerar la fase de ideación y obtener una base visual de referencia, el diseño final de la aplicación no es una copia directa de las interfaces generadas. Cada pantalla fue posteriormente adaptada, refinada y personalizada para alinearse mejor con los requerimientos del proyecto, considerando las particularidades del cálculo de láminas de riego, las variables agronómicas involucradas y el flujo operativo esperado para los usuarios agrícolas.

El resto del diseño, incluyendo la distribución de elementos, la organización de la información, los componentes interactivos y la experiencia general de navegación, fue desarrollado de forma original, tomando como guía únicamente las necesidades del sistema y no modelos externos. Esto permitió construir una interfaz clara, funcional y coherente con el propósito de la aplicación: facilitar el registro de parcelas, visualizar resultados de cálculo y guiar al usuario en la planificación de sus riegos.

La independencia en el diseño, combinada con la flexibilidad que ofrece V0 de Vercel para la generación rápida de prototipos, favoreció un proceso creativo más libre y eficiente. Como resultado, la aplicación presenta una estética limpia, elementos bien estructurados y una experiencia intuitiva que se ajusta al contexto real de uso en campo y a los requerimientos agronómicos del sistema.

6.4.1 Configuración del entorno de desarrollo

Para establecer una base sólida en el desarrollo de la aplicación de riego, el primer paso fue la creación de un repositorio en GitHub, el cual centraliza todo el código del proyecto y permite llevar un control adecuado mediante ramas, commits y seguimiento de cambios. Este repositorio también facilita la integración directa con la plataforma de despliegue utilizada, garantizando que cada actualización del código pueda ser registrada y gestionada de manera ordenada.

Una vez creado el repositorio, se configuraron plantillas para issues con el objetivo de estandarizar la forma en que se documentan tareas, errores o solicitudes de mejora. Estas plantillas permiten registrar información clara como descripciones, criterios de aceptación y prioridad, lo cual favorece un proceso de desarrollo más organizado y transparente.

Para la gestión del flujo de trabajo se utilizó un tablero Kanban en la plataforma Kanban Tool. Este tablero permitió clasificar las tareas en columnas como “Por hacer”, “En progreso” y “Finalizado”, ofreciendo una visualización clara del avance y facilitando la administración ágil del proyecto. Cada actividad registrada en el sistema se actualizó conforme avanzaba la implementación, permitiendo mantener un control constante sobre el desarrollo.

En cuanto a la estructura tecnológica, el proyecto se organizó siguiendo un enfoque modular. Para el backend y manejo de datos se utilizó Supabase, que integra autenticación, base de datos PostgreSQL, almacenamiento y funciones automatizadas. Se configuraron tablas como profiles, parcels y calculations, junto con Row Level Security (RLS) y políticas que garantizan que cada usuario solo pueda acceder a sus propios datos. Además, se emplearon triggers internos de Supabase, como la función automática que crea un perfil al registrar un nuevo usuario.

El frontend fue desarrollado utilizando Next.js junto con React y TypeScript. Esta combinación permitió crear una interfaz moderna, rápida y bien estructurada. TailwindCSS fue utilizado para la capa visual, lo que facilitó la construcción de un diseño responsivo y coherente para pantallas de escritorio y dispositivos móviles. Next.js permitió estructurar el proyecto con rutas organizadas, componentes reutilizables y una lógica clara para el manejo del flujo de datos proveniente de Supabase.

Durante la fase de diseño se empleó V0 de Vercel como herramienta de apoyo para generar prototipos iniciales basados en descripciones textuales. Estas propuestas sirvieron como referencia visual en etapas tempranas, aunque posteriormente se realizaron ajustes manuales para adaptarlas a los requisitos agronómicos específicos de

la aplicación, como los formularios de ingreso de datos, paneles informativos y cálculos de lámina de riego.

Para el despliegue se utilizó Vercel, el cual se integró directamente con el repositorio de GitHub. Se configuró un trigger automático que ejecuta un despliegue cada vez que se realiza un push a la rama principal. Este proceso genera una versión optimizada del proyecto y asegura que la aplicación esté siempre actualizada sin necesidad de configuraciones adicionales. Vercel también gestiona las variables de entorno necesarias para la conexión con Supabase, simplificando la transición entre desarrollo y producción.

Gracias a esta configuración del entorno de desarrollo, el proyecto cuenta con una base ordenada, escalable y reproducible. Las herramientas empleadas —Supabase, Next.js, React, TypeScript, Vercel y Kanban Tool— permitieron estructurar un flujo de trabajo claro y eficiente, adecuado para el desarrollo de una aplicación especializada en el cálculo de lámina de riego y gestión de información agronómica.

6.4.2 Implementación de la lógica de negocio & reglas agronómicas}

Este apartado describe en detalle la implementación de la lógica de negocio y las reglas agronómicas contenidas en el código entregado. Se busca explicar qué hace cada módulo, por qué se implementó de esa manera y cuáles son las decisiones de diseño más importantes. Se incluyen fragmentos de código representativos, explicaciones generales y consideraciones agronómicas y de validación. El objetivo es que el lector comprenda la responsabilidad de cada componente y cómo se interrelacionan para calcular la evapotranspiración (ETo), la etapa fenológica y los parámetros de riego.

Visión general de la arquitectura y responsabilidades

En términos generales, el código se organiza en dos grandes servicios:

WeatherService: responsable de obtener la evapotranspiración de referencia (ETo) a partir de distintas fuentes (APIs externas o datos manuales). Contiene validaciones de configuración y creación de configuraciones por defecto.

IrrigationCalculatorService: responsable de aplicar las reglas agronómicas para convertir ETo y datos del cultivo/suelo en recomendaciones de riego (lámina neta, CAU, frecuencia, próximo riego, etc.).

Además, existe un conjunto de funciones auxiliares y una implementación completa de la lógica fenológica (getPhenologicalStage) que abstrae la determinación de la etapa del cultivo y la interpolación del coeficiente de cultivo (Kc) a lo largo de sus etapas.

WeatherService — Obtención y validación de fuentes Propósito

Centralizar la abstracción de proveedores meteorológicos y asegurar que la llamada al servicio de cálculo de ETo sea robusta (reintentos, tiempo de espera, selección de proveedor). También valida la configuración para evitar errores por falta de claves o datos.

Fragmento representativo: creación y selección del servicio

1 Tabla creación y selección de servicio

Configuración básica	Selección del servicio

```
// Configuración base del servicio
const baseConfig = {
  primaryProvider: "openweathermap" as Provide
  apiKeys: {
    openWeatherMap: apiKeys.openWeatherMap,
    visualCrossing: apiKeys.visualCrossing,
    weatherAPI: apiKeys.weatherAPI,
  },
  retryAttempts,
  timeoutMs,
  defaultCalculator: "penman-monteith" as cons
}
```

```
switch (weatherSource) {
  case "openweathermap":
    service = EToServiceFactory.create({
      ...baseConfig,
      primaryProvider: "openweathermap" as const,
    })
    break

  case "visual-crossing":
    service = EToServiceFactory.create({
      ...baseConfig,
      primaryProvider: "visual-crossing" as const,
    })
    break

  case "weather-api":
    service = EToServiceFactory.create({
      ...baseConfig,
      primaryProvider: "weather-api" as const,
    })
    break

  case "manual":
    if (!manualData) {
      throw new Error("Se requieren datos manuales cuando la fuente es 'man
    }
    service = EToServiceFactory.createWithManualData(
      manualData,
      location.latitude,
      location.longitude,
      "penman-monteith"
    )
    break

  default:
    throw new Error(`Fuente de datos no reconocida: ${weatherSource}`)
}
```

Explicación general: se construye primero una baseConfig con valores comunes (intentos de reintento, timeout y calculador por defecto). Luego, según la fuente (weatherSource) se instancia un servicio específico mediante una fábrica (EToServiceFactory). La opción manual permite inyectar datos registrados por el usuario, útil cuando no hay conectividad o se requieren mediciones locales.

Validaciones importantes

validateConfig asegura que exista ubicación válida (latitud/longitud dentro de rangos), que exista la API key requerida para la fuente seleccionada y que, en modo manual, se hayan proporcionado datos.

if (!config.location) errors.push("Se requiere una ubicación")

if (config.location.latitude < -90 || config.location.latitude > 90) errors.push("La latitud debe estar entre -90 y 90")

Estas validaciones implementan reglas de negocio básicas: sin ubicación no se puede calcular ETo; sin API key la consulta a proveedores fallaría.

IrrigationCalculatorService — reglas agronómicas y cálculos de riego Propósito

Traducir la ETo y las características del cultivo/terreno en decisiones prácticas de riego: lámina neta diaria, volumen, CAU (Capacidad de Almacenamiento Útil), lámina de riego, frecuencia, y fecha del próximo riego.

- Flujo general del cálculo
- Determinar la etapa fenológica y el Kc actual usando getPhenologicalStage.
- Calcular la lámina neta diaria: laminaNeta = kcActual * eto.
- Convertir la profundidad de raíces a centímetros para usar en la fórmula del CAU.
- Calcular CAU (mm) usando la diferencia CC - PMP (capacidad de campo y punto de marchitez permanente), la densidad aparente Da, profundidad de raíces y un factor de conversión.

- Calcular lámina de riego multiplicando CAU por el waterFactor (porcentaje utilizable).
- Calcular frecuencia como laminaDeRiego / laminaNeta (días).

Calcular próximo riego sumando la frecuencia a la fecha de referencia.

```
// 1. Obtener fenología
const phenology = getPhenologicalStage(inputs.crop, inputs.sowingDate);

// 2. Calcular lámina neta diaria (mm/día)
const laminaNeta = phenology.kcActual * inputs.eto;

// 3. Convertir profundidad de raíces a cm
const depthRoots_cm = inputs.depthRoots * 100;

// 4. Calcular Capacidad de Almacenamiento Útil (mm)
const CAU =
  (inputs.soilType.CC - inputs.soilType.PMP) *
  inputs.soilType.Da *
  depthRoots_cm *
  10;

// 5. Calcular lámina de riego (mm)
const laminaDeRiego = CAU * inputs.waterFactor;

// 6. Calcular frecuencia de riego (días)
const frecuenciaRiego = laminaDeRiego / laminaNeta;

// 7. Calcular próximo riego
const proximoRiego = this.calcularProximoRiego(
  frecuenciaRiego,
  this.parseDateLocal(inputs.sowingDate)
);

// 8. Formatear etapa del cultivo
const etapaCultivo =
  phenology.stage +
  (phenology.stage === "No sembrado"
    ? ""
    : ` (día ${phenology.dayOfStage})`);
```

3 Figura 5.3 Fragmento representativo

- Explicación general: el cálculo sigue las prácticas agronómicas estándar: usar la diferencia entre CC y PMP para estimar la cantidad de agua disponible en el suelo por unidad de profundidad; convertir la raíz a una profundidad coherente y multiplicar por la densidad aparente y coeficientes para obtener mm disponibles. El waterFactor modela la fracción utilizable antes de regar.

Validaciones de entrada:

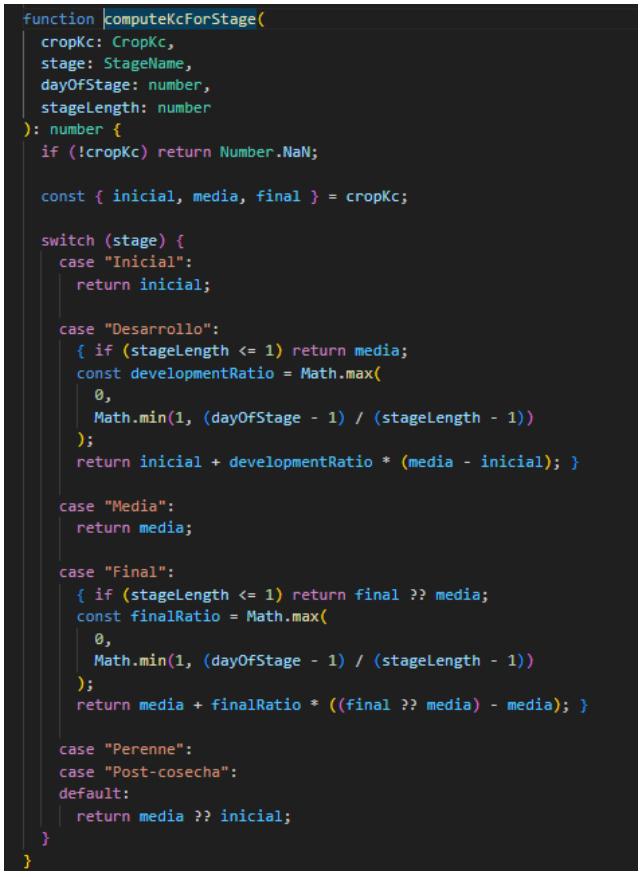
- validateInputs asegura que el cultivo, tipo de suelo, área y factores sean consistentes y no provoquen divisiones por cero o resultados absurdos (p. ej. factor de agua fuera de [0,1]). Estas validaciones son reglas de negocio que protegen el producto contra uso indebido.

Cálculo de la fenología y Kc (getPhenologicalStage) Propósito

Determinar la etapa del cultivo (Inicial, Desarrollo, Media, Final, Perenne, Post-cosecha, Pre- siembra) y el kcActual en función de los días desde la siembra y las duraciones de etapa.

Razón de diseño

Separar la lógica fenológica en funciones pequeñas (normalizar fechas, calcular límites de etapa, interpolar Kc) facilita el testeo y permite aplicar valores por defecto (fallbacks) cuando faltan datos



```
function computeKcForStage(
  cropKc: CropKc,
  stage: StageName,
  dayOfStage: number,
  stageLength: number
): number {
  if (!cropKc) return Number.NaN;

  const { inicial, media, final } = cropKc;

  switch (stage) {
    case "Inicial":
      return inicial;

    case "Desarrollo":
      { if (stageLength <= 1) return media;
      const developmentRatio = Math.max(
        0,
        Math.min(1, (dayOfStage - 1) / (stageLength - 1))
      );
      return inicial + developmentRatio * (media - inicial); }

    case "Media":
      return media;

    case "Final":
      { if (stageLength <= 1) return final ?? media;
      const finalRatio = Math.max(
        0,
        Math.min(1, (dayOfStage - 1) / (stageLength - 1))
      );
      return media + finalRatio * ((final ?? media) - media); }

    case "Perenne":
    case "Post-cosecha":
    default:
      return media ?? inicial;
  }
}
```

4 Figura 5.4 Fragmento representativo: interpolación Kc

Explicación general: la interpolación lineal durante etapas de transición permite suavizar el Kc entre valores tabulados por la FAO, reproduciendo variaciones reales del cultivo.

Reglas agronómicas y consideraciones prácticas

Unidades: el sistema trabaja con mm para láminas y cm para raíces (se convierten internamente), y devuelve fechas en formato ISO (YYYY-MM-DD) compatibles con Supabase TIMESTAMP.

Fallbacks: si las duraciones de etapa no están disponibles se usan valores por defecto (FALLBACK_DURATIONS), manteniendo robustez.

Manejo de fechas: la conversión a medianoche UTC ayuda a evitar errores por zonas horarias cuando se calculan días entre fechas.

Seguridad y errores: todas las operaciones críticas están rodeadas de validaciones y try/catch para devolver resultados por defecto o lanzar errores con mensajes claros.

Creación de los endpoints de la API (interacción frontend - base de datos)

La construcción de los endpoints de la API representa uno de los componentes esenciales dentro de la arquitectura general de la aplicación, ya que permite establecer el flujo de comunicación entre el frontend desarrollado en Next.js y la base de datos alojada en Supabase. La lógica implementada tiene como objetivo permitir que el usuario gestione sus parcelas agrícolas, almacene los resultados de los cálculos de lámina neta, frecuencia de riego, volumen diario y otras variables derivadas del modelo agronómico, y asegure que cada operación responda de forma rápida, consistente y segura.

En términos generales, los endpoints se materializan a través de dos servicios principales:

CalculationService, orientado exclusivamente a la creación, recuperación y eliminación de resultados de cálculo.

ParcelService, encargado de todas las operaciones CRUD relacionadas con las parcelas del usuario.

Ambos servicios utilizan el cliente de Supabase, el cual funciona como un puente entre la aplicación y la base de datos PostgreSQL administrada en la nube. De manera general, cada endpoint fue diseñado bajo principios de claridad, modularidad y separación de responsabilidades: la interfaz de usuario únicamente invoca métodos del servicio, los cuales a su vez se encargan de ejecutar las consultas correspondientes, procesar los datos y retornar un objeto con la estructura requerida por las vistas.

Conexión general a Supabase

```
import { createBrowserClient } from "@supabase/ssr";

const supabaseUrl = process.env.NEXT_PUBLIC_SUPABASE_URL;
const supabaseKey = process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY;


export const createClient = () =>
  createBrowserClient(
    supabaseUrl!,
    supabaseKey!,
  );
```

5Figura 5.5 endpoints

Antes de analizar de manera detallada los servicios, es importante comprender la base sobre la cual se construyen los endpoints: la declaración del cliente usando Supabase.

Aquí se establece el cliente del lado del navegador, utilizando variables de entorno que garantizan la seguridad de las credenciales. La función createClient() permite generar instancias independientes dentro de cada servicio, asegurando que las consultas sean ejecutadas dentro del contexto del usuario autenticado.

Endpoints del módulo de cálculos (CalculationService)

El primer servicio implementado es CalculationService, el cual se diseñó para gestionar exclusivamente la información relacionada con los cálculos derivados del modelo de riego. La tabla involucrada es calculations, que almacena los valores resultantes de cada operación generada por el usuario.

A continuación, se describe el funcionamiento general del servicio:

Obtención de cálculos por parcela

Este endpoint permite al usuario consultar el historial completo de cálculos realizados en una parcela específica. Su propósito es facilitar la visualización cronológica de la evolución del riego.

```
async getCalculationsByParcelId(parcelId: string): Promise<CalculatedResults[]> {
  const { data, error } = await this.supabase
    .from('calculations')
    .select('*')
    .eq('parcel_id', parcelId)
    .order('calculated_at', { ascending: false })

  if (error) {
    console.error('Error loading calculations:', error)
    throw new Error('No se pudieron cargar los cálculos')
  }

  return data.map(calc => this.transformCalculation(calc))
}
```

6Figura 5.6 Otro end point

El método ejecuta una consulta filtrando por el campo parcel_id, lo cual garantiza que sólo se muestren registros vinculados a la parcela activa. La ordenación descendente (ascending: false) permite que los cálculos más recientes aparezcan primero, lo cual facilita la visualización en las interfaces del usuario.

En caso de error, se captura y se traduce en un mensaje genérico para evitar exponer detalles técnicos:

```
if (error) {
  console.error('Error loading calculations:', error)
  throw new Error('No se pudieron cargar los cálculos')
}
```

7Figura 5.7 Error

Cada registro recuperado es transformado hacia el formato interno de la aplicación mediante el método transformCalculation.

Creación de un cálculo

La creación de cálculos constituye uno de los puntos centrales del sistema, ya que el usuario genera resultados a partir del modelo de lámina neta, ETo, Kc, profundidad radicular y demás variables.

El método recibe el parcelId y un objeto con los datos del cálculo. Antes de insertar, se construye un objeto compatible con la estructura exacta de la tabla en la base de datos:

```
parcelId: string,
calculation: Omit<CalculatedResults, 'id' | 'calculatedAt'>
: Promise<CalculatedResults> {
const calculationData = {
  parcel_id: parcelId, net_sheet: calculation.netSheet,
  daily_volume: calculation.dailyVolume, irrigation_frequency: calculation.irrigationFrequency,
  next_irrigation: calculation.nextIrrigation,root_depth: calculation.rootDepth,
  water_factor: calculation.waterFactor,crop_stage: calculation.cropStage,
  current_kc: calculation.currentKc,eto: calculation.eto
}
```

8 Figura 5.8 Puntos centrales

Posteriormente se realiza la inserción:

```
const { data, error } = await this.supabase
  .from('calculations')
  .insert(calculationData)
  .select()
  .single()
```

9Figura 5.9 Insercion

El uso de .single() asegura que la respuesta incluya únicamente el registro recién creado, lo cual permite devolverlo inmediatamente al frontend para actualizar la interfaz sin recargar la página.

Obtención del cálculo más reciente

Se implementó un endpoint adicional que permite recuperar únicamente el último cálculo realizado:

```
async getLatestCalculation(parcelId: string): Promise<CalculatedResults | null> {
  const { data, error } = await this.supabase
    .from('calculations')
    .select('*')
    .eq('parcel_id', parcelId)
    .order('calculated_at', { ascending: false })
    .limit(1)
    .single()

  if (error) {
    console.error('Error loading latest calculation:', error)
    return null
  }

  return this.transformCalculation(data)
}
```

10Figura 5.10 End Point adicional

Este método se utiliza dentro de la vista principal de riego para mostrar inmediatamente la información actualizada sin necesidad de recorrer todo el historial.

Transformación de los datos

Supabase retorna registros con nombres de columnas en snake_case. Para que la aplicación mantenga coherencia interna en camelCase, se utiliza un método de transformación:

```
private transformCalculation(data: any): CalculatedResults {
  return {
    id: data.id,
    netSheet: data.net_sheet,
    dailyVolume: data.daily_volume,
    waterFactor: data.water_factor,
    rootDepth: data.root_depth,
    irrigationFrequency: data.irrigation_frequency,
    nextIrrigation: data.next_irrigation,
    cropStage: data.crop_stage,
    currentKc: data.current_kc,
    eto: data.eto,
    calculatedAt: new Date(data.calculated_at)
  }
}
```

11Figura 5.11 Transformación

Este proceso asegura que todas las vistas y componentes trabajen de forma uniforme y evita errores en el frontend.

Endpoints del módulo de parcelas (ParcelService)

El servicio ParcelService gestiona la información relacionada con las parcelas registradas por cada usuario. Este módulo no solo maneja datos básicos como nombre, ubicación y fecha de siembra, sino que también controla la estructura de los cultivos, tipos de suelo y cálculos asociados.

Obtención de parcelas por usuario

```
async getUserParcels(userId: string): Promise<Parcel[]> {
  const { data, error } = await this.supabase
    .from('parcels')
    .select('*')
    .eq('user_id', userId)
    .order('created_at', { ascending: false })

  if (error) {
    console.error('Error loading parcels:', error)
    throw new Error('No se pudieron cargar las parcelas')
  }

  return this.transformParcels(data)
}
```

12Figura 5.12

El método principal para cargar la información general del usuario es:

Aquí se filtra la tabla parcels por el campo user_id. Posteriormente, cada registro se transforma mediante transformParcels.

Consulta de una parcela con sus cálculos

Se implementó un endpoint más completo:

```
async getParcelById(parcelId: string, userId: string): Promise<Parcel | null> {
  const { data, error } = await this.supabase
    .from('parcels')
    .select(`
      *,
      calculations (*)
    `)
    .eq('id', parcelId)
    .eq('user_id', userId)
    .single()

  if (error) {
    console.error('Error loading parcel:', error)
    return null
  }

  return this.transformParcelWithCalculations(data)
}
```

13Figura 5.13 endpoint actualizado

Esto permite traer tanto la información de la parcela como su lista completa de cálculos, facilitando su uso en una sola consulta.

Creación de una nueva parcela

El método encargado de registrar nuevas parcelas es:

```
async createParcel(
  userId: string,
  name: string,
  partialData?: Partial<Omit<Parcel, 'id' | 'userId' | 'createdAt'>>
): Promise<Parcel> {
  const newParcelData = {
    user_id: userId, name: name.trim(),
    location: partialData?.location || { latitude: 0, longitude: 0 },
    area: partialData?.area || 0, sowing_date: partialData?.sowingDate?.toISOString() || new Date().toISOString(),
    crop: partialData?.crop || this.getEmptyCrop(), soil_type: partialData?.soilType || this.getEmptySoilType()
  }

  const { data, error } = await this.supabase
    .from('parcels')
    .insert(newParcelData)
    .select()
    .single()

  if (error) {
    console.error('Error creating parcel:', error)
    throw new Error('No se pudo crear la parcela')
  }

  return this.transformParcel(data)
}
```

14 Figura 5.14 Creación de nueva parcela

Es aquí donde se construye un objeto completo que incluya datos como:

- ubicación geográfica,
- área,
- fecha de siembra,
- estructura del cultivo,
- propiedades del tipo de suelo.

En caso de que el usuario no defina alguno de estos elementos, se recurren a métodos auxiliares que retornan objetos vacíos:

```
private getEmptyCrop(): Crop {
  return {
    name: "",
    value: "",
    emoji: "",
    Kc: { inicial: 0, desarrollo: 0, media: 0, final: 0 },
    rootDepth: { min: 0, max: 0 },
    p: { min: 0, max: 0 }
  }
}

private getEmptySoilType(): SoilType {
  return {
    value: "",
    emoji: "",
    label: "",
    CC: 0,
    PMP: 0,
    Da: 0,
    description: ""
  }
}
}
```

15Figura 5.15 Relleno de datos vacíos

Esto evita errores en la interfaz.

Importancia general de los endpoints en la arquitectura del sistema

Los endpoints desarrollados cumplen funciones fundamentales:

Centralizan la lógica de acceso a datos, evitando que el frontend interactúe directamente con la base de datos.

Garantizan coherencia en los formatos, transformando los datos de Supabase a estructuras internas homogéneas.

Separan responsabilidades, lo que mejora la mantenibilidad y escalabilidad del proyecto.

Proveen un entorno seguro, ya que cada operación considera al usuario autenticado.

Reducen carga duplicada, mediante consultas específicas como la obtención del cálculo más reciente.

Facilitan el trabajo del frontend, ya que los datos se retornan listos para usarse directamente en las interfaces.

Gracias a estos endpoints, la aplicación es capaz de registrar y procesar información agronómica de forma consistente, lo que contribuye directamente al objetivo general del sistema: ofrecer un modelo de cálculo de riego eficiente, organizado y accesible para el usuario final.

Funcionalidades del Perfil (gestión de parcelas, usuarios, historial de cálculos)

El módulo correspondiente a las funcionalidades de MyParcels se implementó principalmente en la interfaz de usuario a través de páginas y componentes que consumen servicios ya definidos para interactuar con la base de datos. El objetivo central del código es proporcionar las operaciones de creación y lectura necesarias para

gestionar parcelas, mantener el historial de cálculos de riego y garantizar que las vistas respondan de forma coherente según el estado de autenticación del usuario. A continuación, se describe de forma sistemática qué se hizo en el código y por qué, integrando fragmentos representativos que muestran las decisiones de diseño y el flujo de datos entre frontend y servicios.

Visión general y propósito

El componente principal de gestión se divide en dos páginas: la vista de lista de parcelas (MyParcels

/ ProfilePage) y la vista de detalle de parcela (ParcelDetailPage). Ambas páginas actúan como capas de presentación que orquestan llamadas a hooks y servicios, administran estados locales (cargas, errores, formularios) y renderizan componentes visuales para permitir al usuario interactuar con los datos. La separación entre la lógica de acceso a datos (servicios) y la presentación sigue el patrón de responsabilidad única: las páginas se encargan de la experiencia de usuario y coordinación, mientras que los servicios implementan las operaciones CRUD sobre la base de datos.

Autenticación y control de acceso

Desde el inicio, las páginas comprueban el estado de autenticación para garantizar que solo usuarios autenticados puedan acceder a las funcionalidades. Esta verificación se realiza porque las operaciones de CRUD están ligadas a un userId y a datos privados de cada usuario; por tanto, la interfaz previene el acceso no autorizado y evita presentar o crear recursos sin un contexto de usuario válido.

Creación de parcelas (flow UI - servicio)

En la página de listado de parcelas se implementó un diálogo de creación que muestra un formulario inicial con los campos mínimos necesarios para crear una parcela.

El estado del formulario se mantiene en formData y se utiliza isCreating para reflejar el estado asíncrono de la operación:

```
const handleCreateParcel = async () => {
  if (!profile) return

  setIsCreating(true)

  try {
    await createParcel(formData.name, formData)
    setIsDialogOpen(false)
  } catch (error: any) {
    alert(error.message)
  } finally {
    setIsCreating(false)
  }
}
```

16Figura 5.16 Metodo Asincrono

Qué se hizo y por qué:

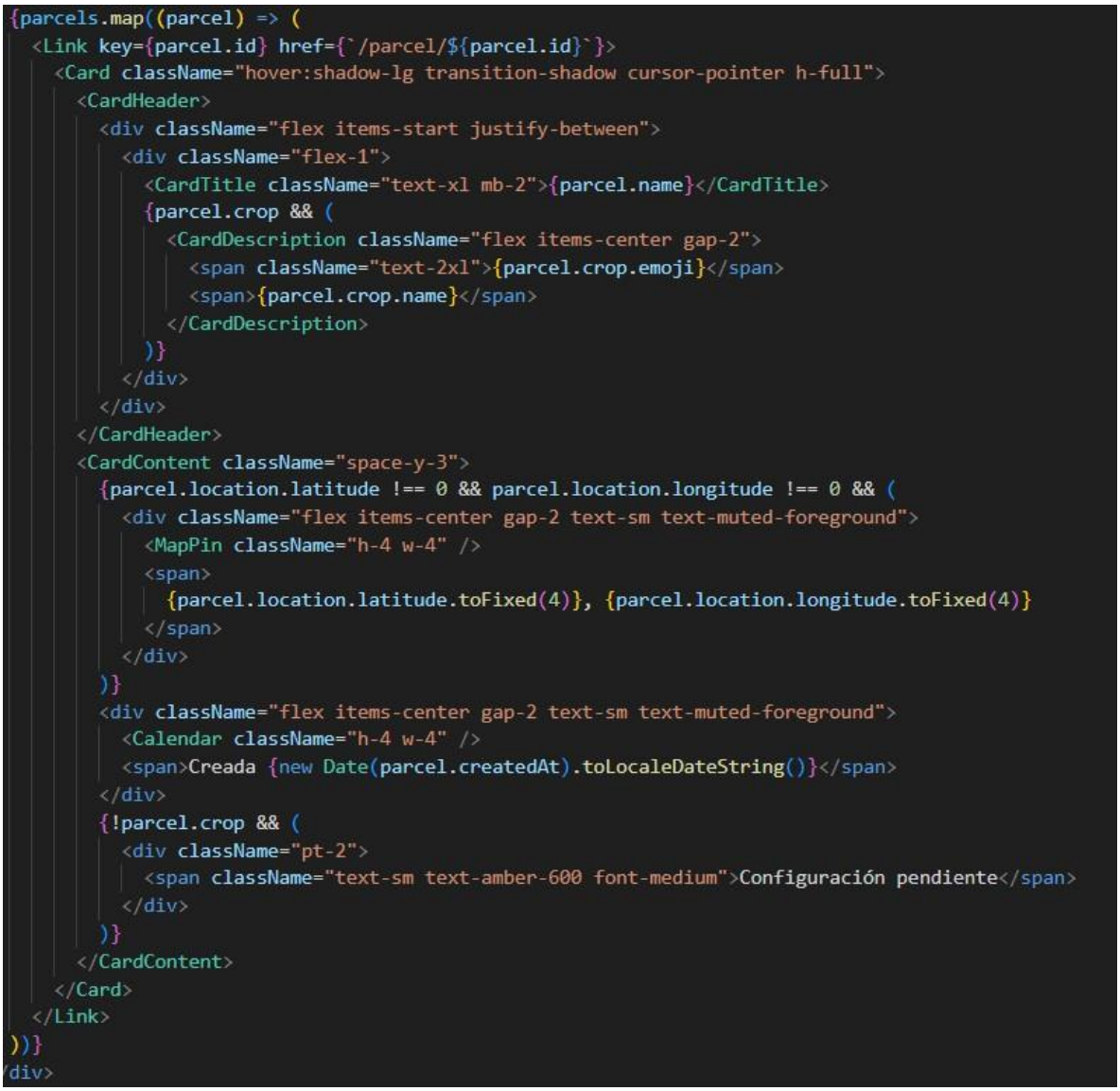
Se verifica la existencia del profile antes de iniciar la creación para asociar la parcela al usuario correcto.

isCreating se emplea para deshabilitar controles o mostrar estados de espera durante la inserción en la base de datos, evitando acciones duplicadas.

Tras la inserción, el diálogo se cierra y la lista de parcelas se actualiza desde el hook useParcels(), de modo que la interfaz refleja inmediatamente el nuevo estado del repositorio de datos.

Listar parcelas y navegación hacia detalle

La vista de lista renderiza las parcelas en una cuadrícula y, si no existen parcelas, muestra una tarjeta invitando a crear la primera. Para cada parcela se genera un enlace hacia la página de detalle:



```
{parcels.map((parcel) => (
  <Link key={parcel.id} href={`/parcel/${parcel.id}`}>
    <Card className="hover:shadow-lg transition-shadow cursor-pointer h-full">
      <CardHeader>
        <div className="flex items-start justify-between">
          <div className="flex-1">
            <CardTitle className="text-xl mb-2">{parcel.name}</CardTitle>
            {parcel.crop && (
              <CardDescription className="flex items-center gap-2">
                <span className="text-2xl">{parcel.crop.emoji}</span>
                <span>{parcel.crop.name}</span>
              </CardDescription>
            )}
          </div>
        </div>
      </CardHeader>
      <CardContent className="space-y-3">
        {parcel.location.latitude !== 0 && parcel.location.longitude !== 0 && (
          <div className="flex items-center gap-2 text-sm text-muted-foreground">
            <MapPin className="h-4 w-4" />
            <span>
              {parcel.location.latitude.toFixed(4)}, {parcel.location.longitude.toFixed(4)}
            </span>
          </div>
        )}
        <div className="flex items-center gap-2 text-sm text-muted-foreground">
          <Calendar className="h-4 w-4" />
          <span>Creada {new Date(parcel.createdAt).toLocaleDateString()}</span>
        </div>
        {!parcel.crop && (
          <div className="pt-2">
            <span className="text-sm text-amber-600 font-medium">Configuración pendiente</span>
          </div>
        )}
      </CardContent>
    </Card>
  </Link>
))}
</div>
```

17Figura 5.17 render de las parcelas

Qué se hizo y por qué:

La lista se presenta ordenada y con información clave (nombre, cultivo, ubicación y fecha de creación) para ofrecer al usuario una visión rápida y accionable.

El uso de Link con el parcel.id facilita la navegación a la página de detalle donde se cargan datos más completos y el historial de cálculos.

Carga de datos en la página de detalle

La página de detalle de parcela coordina la obtención de datos de la parcela específica y de su historial de cálculos. Al montarse, se ejecuta loadParcelData() que llama a parcelService.getParcelById(parcelId, profile.id)

Qué se hizo y por qué:

La consulta incluye validación del userId, evitando que un usuario acceda a una parcela que no le pertenece.

Se manejan estados de carga (isLoadingParcel) y error (parcelError) para proporcionar retroalimentación visual adecuada: loader mientras se solicita la información o una tarjeta de error si la parcela no existe.

Si la parcela no es encontrada, se redirige a la lista de parcelas; esto mantiene la coherencia de navegación y evita que la interfaz muestre una página vacía.

Presentación de la información de la parcela

Una vez cargada la parcela se renderizan tarjetas con su información esencial: ubicación, área y tipo de suelo. Además, si existe un cálculo reciente, se presenta en una tarjeta destacada con los indicadores principales (lámina neta, volumen, frecuencia, próximo riego):

Qué se hizo y por qué:

Mostrar el cálculo más reciente permite al usuario ver de inmediato el estado actual de riego sin navegar por el historial completo.

La información se formatea (valores numéricos, fechas) para facilitar la interpretación y comparación.

Historial de cálculos (Read / List)El historial se presenta en una tabla donde se listan todos los cálculos recuperados para la parcela. Si no existen cálculos, se ofrece una UI que invita a generar el primero:

Qué se hizo y por qué:

El listado cronológico del historial permite al usuario revisar la evolución de variables de riego a lo largo del tiempo.

La tabla condensa las métricas más relevantes para facilitar comparaciones rápidas entre registros.

Creación de nuevos cálculos (interacción desde detalle)

La página de detalle expone un botón para crear un nuevo cálculo que abre un diálogo (NewCalculationDialog). Al guardar un cálculo nuevo, se recarga el historial mediante loadCalculations() para reflejar el cambio:

```
const handleNewCalculationSave = async () => {

  try {
    await loadCalculations() // Recargar para obtener los últimos datos
  } catch (error) {
    console.error(error)
  }
}

if (authLoading) {
  return (
    <div className="min-h-screen flex items-center justify-center">
      <Loader2 className="h-8 w-8 animate-spin text-primary" />
    </div>
  )
}
```

18Figura 5.18 Recarga de datos

Qué se hizo y por qué:

Se integró el flujo de creación de cálculos dentro de la experiencia de la parcela para que el usuario pueda generar y visualizar resultados sin cambiar de contexto.

La recarga del historial garantiza consistencia entre la base de datos y la UI tras inserciones.

Gestión de estados y experiencia de usuario

En ambos componentes se implementaron patrones comunes: loaders visuales durante las solicitudes asíncronas, manejo de errores con mensajes claros y controles de diálogo para operaciones de creación. Estas implementaciones buscan evitar estados ambiguos, prevenir acciones repetidas y ofrecer retroalimentación inmediata al usuario sobre el resultado de sus interacciones.

Gestión de autenticación, permisos y seguridad de datos

El módulo de autenticación fue implementado como un proveedor de contexto (AuthProvider) y un hook (useAuth) que centralizan todas las operaciones relacionadas con la sesión de usuario, el perfil y las acciones de inicio/cierre de sesión. La implementación recoge y mantiene en memoria el estado del usuario y del perfil, expone funciones para el registro, ingreso, cierre de sesión y actualización del perfil, y coordina la suscripción a los cambios de estado de autenticación proporcionados por Supabase. A continuación, se describe, de forma técnica y sistemática, qué se hizo en el código y por qué.

Visión general y responsabilidad

Se creó un contexto de React que actúa como la única fuente de verdad para todo lo relativo a la autenticación en la aplicación. El AuthProvider mantiene internamente el estado de user, profile, session y loading, y expone operaciones (signUp, signIn, signOut, updateProfile) que la UI y otros hooks consumen. La separación entre proveedor y consumidor (hook useAuth) permite que cualquier componente tenga acceso al estado autenticado sin necesidad de comunicarse directamente con Supabase ni replicar lógica de sesión.

Estructura básica del contexto y tipos

Se definieron interfaces para el perfil y para el tipo del contexto, de forma que la API interna queda tipada y explícita:

```
interface Profile {
  id: string
  email: string
  name: string
  created_at: string
}

interface AuthContextType {
  user: User | null
  profile: Profile | null
  session: Session | null
  loading: boolean
  signUp: (email: string, password: string, name: string) => Promise<void>
  signIn: (email: string, password: string) => Promise<void>
  signOut: () => Promise<void>
  updateProfile: (name: string) => Promise<void>
}
```

19Figura 5.19 Diseño de interfaces

Creación del cliente y estado inicial

Al montar el proveedor, se inicializa una instancia del cliente de Supabase y se declaran los estados React que se usarán:

```
const [user, setUser] = useState<User | null>(null)
const [profile, setProfile] = useState<Profile | null>(null)
const [session, setSession] = useState<Session | null>(null)
const [loading, setLoading] = useState(true)
const [supabase] = useState(() => createClient())
const router = useRouter()
```

20 Figura 5.20 Estados de react que se usarán

Obtención del perfil del usuario

Se implementó una función fetchProfile que consulta la tabla profiles en la base de datos, filtrando por el userId:

```
const fetchProfile = useCallback(async (userId: string) => {
  try {
    const { data, error } = await supabase
      .from('profiles')
      .select('*')
      .eq('id', userId)
      .single()

    if (error) {
      console.error('Error fetching profile:', error)
      return null
    }
    return data
  } catch (error) {
    console.error('Exception fetching profile:', error)
    return null
  }
}, [supabase])
```

21Figura 5.21 consulta a la tabla de perfiles

Se añadió una capa que abstrae la obtención del perfil original almacenado en la base de datos, devolviendo el objeto de perfil o null en caso de error. La función está memoizada con useCallback para evitar recreaciones innecesarias.

Manejo unificado de sesión y perfil

El proveedor define una función handleSession que centraliza la lógica que se debe ejecutar cada vez que cambia la sesión (al iniciar, cerrar o refrescar tokens). Esta función actualiza los estados session, user y profile según exista o no una sesión activa:

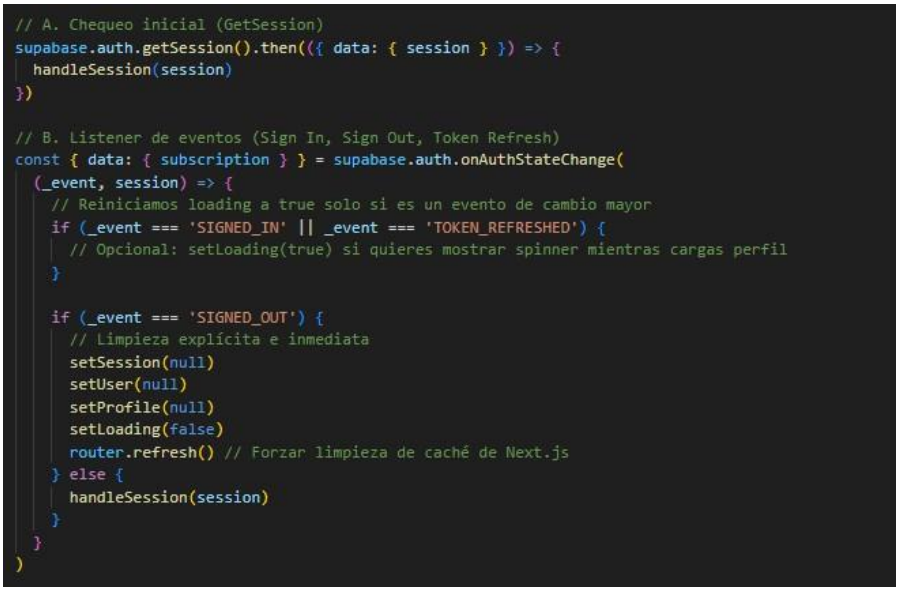
```
const handleSession = async (currentSession: Session | null) => {
  try {
    if (!mounted) return

    if (currentSession?.user) {
      // 1. Tenem  (parameter) currentSession: Session
      setSession(currentSession)
      setUser(currentSession.user)

      // 2. Buscamos el perfil solo si el usuario cambió o no tenemos perfil
      // Esto evita llamadas innecesarias
      const profileData = await fetchProfile(currentSession.user.id)

      if (mounted) {
        setProfile(profileData)
      }
    } else {
      // 3. NO hay sesión: Limpieza TOTAL
      setSession(null)
      setUser(null)
      setProfile(null)
    }
  } catch (error) {
    console.error('Error handling session:', error)
  } finally {
    // 4. SIEMPRE apagamos el loading al final, pase lo que pase
    if (mounted) {
      setLoading(false)
    }
  }
```

22Figura 5.22 manejo de perfil

Se centralizó el flujo de sincronización entre la sesión del proveedor de autenticación y el perfil persistido en la base de datos, asegurando que el estado local refleje siempre la situación real de autenticación.

Chequeo inicial y suscripción a cambios de estado

En un useEffect de inicialización se realizó un chequeo de sesión (getSession) y se suscribió el proveedor a eventos de autenticación (SIGN_IN, SIGN_OUT, TOKEN_REFRESHED). El efecto también incluye un mecanismo de limpieza con la variable mounted y la desuscripción a la suscripción:



```
// A. Chequeo inicial (GetSession)
supabase.auth.getSession().then(({ data: { session } }) => {
  handleSession(session)
})

// B. Listener de eventos (Sign In, Sign Out, Token Refresh)
const { data: { subscription } } = supabase.auth.onAuthStateChange(
  (_event, session) => {
    // Reiniciamos loading a true solo si es un evento de cambio mayor
    if (_event === 'SIGNED_IN' || _event === 'TOKEN_REFRESHED') {
      // Opcional: setLoading(true) si quieres mostrar spinner mientras cargas perfil
    }

    if (_event === 'SIGNED_OUT') {
      // Limpieza explícita e inmediata
      setSession(null)
      setUser(null)
      setProfile(null)
      setLoading(false)
      router.refresh() // Forzar limpieza de caché de Next.js
    } else {
      handleSession(session)
    }
  }
)
```

23Figura 5.23 Codigo de desuscripción

Se implementó tanto el chequeo inicial de sesión como la escucha de eventos de autenticación, para mantener el estado sincronizado con cambios que pueden producirse fuera del ciclo de vida de la aplicación (por ejemplo, inicio de sesión en otra pestaña). Además, cuando ocurre un cierre de sesión, se limpian explícitamente los estados y se fuerza un router.refresh() para invalidar cachés en Next.js.

Operaciones públicas: signUp, signIn, signOut, updateProfile

```
const signUp = async (email: string, password: string, name: string) => {
  const { error } = await supabase.auth.signUp({
    email,
    password,
    options: {
      data: { name },
      emailRedirectTo: `${process.env.NEXT_PUBLIC_SITE_URL}/auth/callback`,
    },
  })

  if (error) throw error
}
```

```
const signIn = async (email: string, password: string) => {
  const { error } = await supabase.auth.signInWithPassword({
    email,
    password,
  })

  if (error) throw error
}
```

```
const signOut = async () => {
  setUser(null)
  setProfile(null)
  setSession(null)

  // Luego decimos a Supabase que cierre
  const { error } = await supabase.auth.signOut()
  if (error) throw error
}
```

24Figura 5.24 Se expusieron funciones que permiten a la UI interactuar con el sistema de autenticación

Cada función delega en las API de Supabase para realizar la operación correspondiente y, cuando procede, actualiza el estado local (profile, user, session). En el caso de signUp se incluye la opción emailRedirectTo que especifica la URL de redirección después de la verificación de correo. updateProfile sincroniza el estado local con la versión persistida luego de la actualización en la tabla profiles.

```
const updateProfile = async (name: string) => {
  if (!user) throw new Error('No user logged in')

  const { error } = await supabase
    .from('profiles')
    .update({ name })
    .eq('id', user.id)

  if (error) throw error

  const updatedProfile = await fetchProfile(user.id)
  if (updatedProfile) {
    setProfile(updatedProfile)
  }
}
```

25Figura 5.25 Exposición del contexto y hook consumidor

Finalmente, el proveedor envuelve a los hijos con AuthContext.Provider y expone el hook useAuth que realiza la comprobación de que el hook se use dentro del proveedor:

```
return (
  <AuthContext.Provider
    value={{
      user,profile,session,loading,
      signUp,signIn,signOut,updateProfile,
    }}
  >
    {children}
  </AuthContext.Provider>
)
```

```
export function useAuth() {
  const context = useContext(AuthContext)
  if (context === undefined) {
    throw new Error('useAuth must be used within an AuthProvider')
  }
  return context
}
```

26Figura 5.26 hook que se usó dentro del proveedor

Se centralizó y tipificó la API pública de autenticación para que los componentes consumidores accedan a la sesión y las operaciones de forma segura y consistente.

Resumen de intenciones y razones operativas

Se centralizó la gestión de la sesión y del perfil para evitar duplicación de lógica y mantener una única fuente de verdad.

Se sincroniza la sesión de Supabase con el perfil persistido en la base de datos para que la aplicación disponga siempre de la información del usuario asociada a su id.

Se implementó un listener de eventos de autenticación para reaccionar a cambios de sesión producidos de forma externa, garantizando consistencia entre pestañas o después de operaciones de token refresh.

Las operaciones públicas (registro, inicio, cierre, actualización) delegan en Supabase y actualizan el estado local para que la UI muestre el estado actual sin necesidad de reconsultas explícitas en cada componente.

El código, por tanto, actúa como la capa de control que gestiona la autenticación, la obtención y actualización del perfil y la sincronización del estado con los eventos externos del proveedor de identidad, permitiendo que el resto de la aplicación consuma de forma sencilla y segura la información de usuario y las funciones asociadas.

— Integración con servicios externos / datos climáticos (API o ingreso manual)

La integración con servicios externos y la posibilidad de ingreso manual de datos constituye en el proyecto la capa responsable de suministrar la información climática necesaria para calcular la evapotranspiración de referencia (ETo). El código implementado aborda esta necesidad mediante una arquitectura basada en interfaces, proveedores de datos y calculadores de ETo, además de una fábrica que orquesta la creación de servicios según la configuración. A grandes rasgos, lo que se hizo fue: (1) definir contratos formales para proveedores y calculadores, (2) implementar calculadores con distintos niveles de complejidad (Hargreaves-Samani y Penman-Monteith FAO- 56), (3) crear adaptadores para obtener datos desde APIs externas (Visual Crossing, OpenWeatherMap, WeatherAPI) y (4) ofrecer un proveedor de datos manuales para permitir ingreso directo por el usuario y operar sin conectividad a servicios externos. Todo esto se unifica a través de un servicio de ETo que decide cuándo usar ETo directo proporcionado por un proveedor o calcularlo localmente, y que soporta un proveedor de fallback.

Interfaces y contrato de responsabilidad

Se definieron interfaces claras que establecen las responsabilidades mínimas de cada componente. Entre las más relevantes están:

```typescript
export interface IEToCalculator {
  calculateETo(weatherData: WeatherData): number;
  getMethod(): string;
}
```

```typescript
export interface IWeatherProvider {
  getWeatherData(
    latitude: number,
    longitude: number,
    date?: Date
  ): Promise<WeatherData>;
  getName(): string;
  supportsDirectETo(): boolean;
}
```

27Figura 5.27 interfaces claras que establecen las responsabilidades mínimas de cada componente

Estas interfaces permiten que el resto del sistema trabaje con cualquier implementación concreta sin acoplamiento fuerte, garantizando interoperabilidad entre proveedores y calculadores.

Calculadores de ETo

Se implementaron dos calculadores que representan enfoques comunes en la literatura agronómica:

Hargreaves-Samani: calculador más simple que requiere solo temperaturas máximas y mínimas y la radiación extraterrestre estimada. El código comprueba la coherencia de las temperaturas, calcula la radiación extraterrestre y aplica la fórmula:

```
// Fórmula Penman-Monteith
const numerator = 0.408 * delta * (Rn - G) + gamma * (900 / (Tmean + 273)) * u2 * (es - ea);
const denominator = delta + gamma * (1 + 0.34 * u2);

const ETo = numerator / denominator;
```

```
// Fórmula Hargreaves-Samani
const ETo = 0.0023 * (Tmean + 17.8) * Math.sqrt(Tmax - Tmin) * Ra;
```

28*Figura 5.28 Ecuaciones programadas*

El calculador incluye validaciones y ajustes mínimos (p. ej. evitar ETo negativas).

Penman-Monteith (FAO-56): calculador más completo que utiliza temperatura, humedad relativa, velocidad del viento y radiación. El algoritmo calcula presiones de vapor, pendiente de la curva de presión de saturación, radiación neta y aplica la ecuación estándar:

Ambos calculadores retornan el método seguido mediante getMethod() para registro y trazabilidad. Las implementaciones incluyen mensajes de advertencia cuando los resultados son poco plausibles, y devuelven valores redondeados coherentes con unidades en mm/día.

Proveedores de datos y adaptadores API

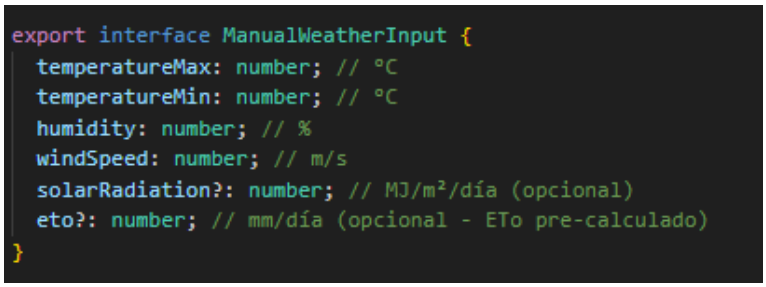
Se proveyeron adaptadores concretos para consumir APIs externas. Un ejemplo es el proveedor para Visual Crossing, que construye la URL con parámetros de latitud, longitud y fecha, realiza la petición fetch, maneja estados HTTP relevantes (401, 429) y transforma la respuesta a la estructura interna WeatherData:

29Figura 5.29 Proveedores de datos y adaptadores API

Los proveedores se responsabilizan por normalizar unidades (por ejemplo convertir km/h a m/s) y por devolver una estructura homogénea que pueda ser consumida por los calculadores.

Proveedor de datos manuales

Se incluyó una implementación que permite al usuario ingresar datos manuales. El ManualDataProvider encapsula los valores proporcionados por el usuario, valida rangos razonables para cada parámetro y expone métodos para recuperar y actualizar esos datos:



30Figura 5.30 Ingresar datos manuales

El proveedor distingue si los datos incluyen un ETo directo (hasDirectETo) y provee una función supportsDirectETo() y getDirectETo() para que el servicio principal

pueda usar ese valor directo cuando esté disponible. Esto facilita escenarios en los que el usuario dispone de mediciones locales o sensores que entregan ETo ya calculada.

Servicio de ETo y lógica de decisión

El EToService es el componente que orquesta la interacción entre proveedores y calculadores. Su responsabilidad principal es intentar obtener datos meteorológicos y decidir si usar un ETo directo (cuando el proveedor lo soporte, p. ej. Visual Crossing puede ofrecer ETo) o calcularlo localmente con el calculador elegido. El flujo general es:

Pedir datos al proveedor (provider.getWeatherData(...)).

Si el proveedor soporta ETo directa y la implementación la ofrece, intentar obtenerla.

Si no está disponible o falla, calcular ETo localmente con el calculador (etoCalculator.calculateETo(weatherData)).

Devolver un EToResult que incluye el valor calculado, la fuente, el método (directo o calculado) y los datos meteorológicos utilizados.

El servicio incorpora además la posibilidad de configurar un fallback: si el proveedor principal falla, y se ha configurado un proveedor alterno, el servicio reintenta la operación con el proveedor de fallback y reporta de forma clara qué proveedor se utilizó para la obtención final.

Fábrica de servicios

Para facilitar la creación de instancias configuradas se implementó una fábrica (EToServiceFactory) que, dado un WeatherConfig, construye el proveedor, selecciona el

calculador por defecto (Penman-Monteith o Hargreaves) y devuelve un EToService listo para usarse.

Esto permite instanciar el servicio según distintos escenarios operativos sin que el resto de la aplicación conozca los detalles de construcción.

Validaciones, unidades y mensajes

En toda la cadena —proveedores, calculadores y proveedor manual— se implementaron validaciones de consistencia (rangos de temperatura, humedad entre 0 y 100%, valores no negativos), conversiones de unidades donde procede (p. ej. radiación, velocidad del viento) y mensajes de advertencia cuando los resultados son extremos. Estas acciones aseguran que los datos que alimentan los cálculos sean coherentes y trazables.

Resumen general de la integración

En conjunto, el desarrollo implementa una capa de integración con servicios climáticos que es modular, configurable y preparada para operar tanto con datos provenientes de APIs externas como con datos ingresados por el usuario. La solución ofrece: interfaces bien definidas, calculadores estandarizados para ETo, adaptadores para proveedores externos, un proveedor manual para ingreso directo y un servicio que decide la estrategia adecuada (uso de ETo directo o cálculo local) y soporta fallback entre proveedores. Esta arquitectura permite que la aplicación obtenga información climática fiable y utilizable por el módulo de riego sin acoplar las capas superiores a detalles de implementación de las fuentes de datos.

— Deploy y configuración en producción (uso de Vercel, triggers automáticos, variables de entorno)

Se realizó el despliegue de la aplicación en Vercel y se definió un flujo de entrega continua que cubre desde el control de versiones hasta la publicación en producción. A grandes rasgos, el proceso implementado incluye: integración del repositorio Git con Vercel, configuración de ramas para despliegues automáticos y vistas previas, gestión de variables de entorno y secretos, y la habilitación de mecanismos que permiten construir, validar y publicar versiones estables de la aplicación con trazabilidad y reversión cuando es necesario.

Flujo general de despliegue en Vercel

Se vinculó el repositorio del proyecto (GitHub/GitLab/Bitbucket) con la plataforma de Vercel, de modo que cada evento relevante en el repositorio desencadena acciones en la plataforma. Cuando se hace push a ramas de trabajo se generan Deploy Previews (vistas previas) que son entornos efímeros accesibles mediante una URL única; esto facilita la validación integrada de cambios. Cuando se hace merge o push a la rama designada para producción (por ejemplo, main o master), Vercel inicia automáticamente una compilación y, tras completarla con éxito, promueve el resultado al entorno de producción.

Triggers automáticos y tipos de despliegue

Se definieron un flujon de trabajo:

Deploys de producción: pushes a la rama principal activan el pipeline de producción que compila y publica la versión definitiva. El historial de builds queda registrado y es posible inspeccionar logs de compilación.

Además del mecanismo nativo de Git hooks que Vercel utiliza al integrarse con el repositorio, se habilitaron Build Hooks y Webhooks cuando fue necesario: los Build Hooks permiten disparar un despliegue desde sistemas externos (por ejemplo un CMS o un job programado) mediante una petición HTTP; los Webhooks se usaron para

notificar a herramientas de monitoreo o canales de comunicación sobre el estado de despliegues (inicio, éxito, fallo).

Variables de entorno y gestión de secretos

Se configuró en Vercel un conjunto de variables de entorno divididas por ámbito: Production, Preview y Development. Cada una de estas se asignó acorde al entorno para asegurar que las credenciales, claves de APIs y parámetros sensibles no se mezclaran entre entornos. Entre las variables presentes en producción se incluyen las claves de servicios externos (APIs meteorológicas, claves de Supabase, etc.), la URL pública del sitio y banderas de configuración que el build y el runtime consumen.

El uso de variables se hizo siguiendo la distinción entre valores que deben estar disponibles en el build-time y los que deben permanecer sólo en runtime. Variables destinadas al cliente (expuestas en el bundle del frontend) se marcaron con el prefijo y la convención del framework en uso, de forma que sólo las variables explícitas para el cliente se incorporaron al código público. Las variables y secretos están almacenados en el panel de Vercel y no forman parte del repositorio, garantizando que no se versionan inadvertidamente.

Build y runtime

El pipeline de Vercel ejecutó el proceso de build definido en el proyecto (instalación de dependencias, compilación de assets, generación de páginas estáticas o pre-renderizadas). El resultado del build se organizó en funciones serverless y/o en archivos estáticos según la configuración del framework. Los artefactos de build se asociaron a rutas fijas y a la URL de preview o producción correspondiente.

En runtime, los endpoints serverless y las funciones (si las hay) consumen las variables de entorno de runtime proporcionadas por Vercel. Las diferencias entre variables visibles en build y en runtime permitieron separar configuraciones que sólo

impactan la compilación (por ejemplo, flags de generación estática) de aquellas que afectan la ejecución (por ejemplo, claves de bases de datos).

Monitoreo, logs y trazabilidad del despliegue

Vercel registró logs de compilación y ejecución, accesibles desde la interfaz de la plataforma. Cada despliegue incluye: registros del proceso de build, salida de la instalación de dependencias, errores detectados y el hash del commit que originó el despliegue. Estos logs permiten trazar qué commit está asociado a cada URL de deployment y facilitan la investigación cuando ocurren fallos de compilación o errores en runtime.

Administración de dominios y certificados

Para la publicación en producción se configuró el dominio personalizado dentro de Vercel y se habilitó la provisión automática de certificado SSL. Vercel gestionó la renovación de certificados y el apuntado DNS requerido para que el dominio productivo resolviera hacia la plataforma.

Estrategia de rollback y versiones

Vercel mantiene un historial de desplegables, de modo que versiones anteriores permanecen disponibles como referencias y pueden restablecerse si es necesario. Cada deployment queda identificado por su hash de commit y una URL, permitiendo revertir a un estado anterior en caso de detectar problemas en producción.

Escalado y ejecución

La aplicación se hospeda en el entorno serverless/edge de Vercel; las funciones se auto escalan según demanda y las rutas estáticas se sirven desde la CDN. El sistema de despliegue asegura que la versión publicada sea la que corresponde al commit asociado, y que las funciones serverless se ejecuten con las variables y permisos configurados en el entorno.

6.5 Comparativa de Eficiencia Operativa: Tiempo de Proceso

En un escenario real de gestión de riego para un distrito o una unidad de producción con múltiples sectores, el tiempo invertido se desglosa de la siguiente manera:

2Tabla comparativa

Fase del Proceso	Cálculo Manual (Minutos)	Rain Solution App (Minutos)	Ahorro de Tiempo
Recopilación de datos (Clima/Suelo)	15 - 20 min	Automático (IoT/API)	100%
Cálculo de ETc y Lámina	10 - 15 min	< 1 segundo	99%
Verificación de errores	5 - 10 min	No requiere (Algoritmo)	100%
Generación de Reporte	20 - 30 min	Automático (PDF)	100%
Total por sector	~60 min	< 2 min	96.60%

Más allá de la precisión hídrica, uno de los hallazgos más disruptivos de esta investigación es la optimización del tiempo administrativo y técnico. Mientras que un técnico especializado requiere aproximadamente 60 minutos para recopilar, calcular y reportar manualmente las necesidades de riego por sector, la herramienta desarrollada reduce este ciclo a menos de 2 minutos.

Esta reducción del 96.6% en el tiempo de procesamiento no solo elimina el error humano inherente al cálculo repetitivo, sino que permite al gestor de riego centrar sus

esfuerzos en la toma de decisiones estratégicas y la supervisión en campo, en lugar del procesamiento de datos básicos. La automatización, por tanto, transforma el cálculo de la lámina de un proceso reactivo y lento a uno preventivo y dinámico.

VII. CONCLUSIONES Y RECOMENDACIÓN

En definitiva, los resultados obtenidos validan la implementación de la herramienta como una alternativa superior al método de gestión tradicional. La convergencia entre la precisión algorítmica y la automatización de procesos ha permitido cuantificar un ahorro del 26.5% en el volumen de agua aplicado, lo que impacta directamente en la sostenibilidad del acuífero y en la reducción de los costos energéticos de bombeo.

Sin embargo, el valor añadido de la optimización no se limita al recurso hídrico; la dimensión temporal surge como un factor crítico de éxito. La transición de un proceso de cálculo manual de 60 minutos a un procesamiento digital de menos de 2 minutos representa una mejora en la eficiencia operativa del 96.6%. Esta liberación de carga administrativa permite que el capital humano se enfoque en la supervisión estratégica, minimizando el riesgo de error humano y garantizando que la lámina de riego aplicada sea siempre la óptima para el desarrollo del cultivo. En conclusión, la optimización propuesta transforma el riego de una tarea empírica y demandante en un proceso científico, ágil y altamente eficiente.

VIII. REFERENCIAS.

BIEHL, M. (2017). API ARCHITECTURE—THE BIG PICTURE FOR BUILDING APIS. 2.

CANCELA, J. J., FANDIÑO, M., REY, B. J., & MARTÍNEZ, E. M. (2015). AUTOMATIC IRRIGATION SYSTEM BASED ON DUAL CROP COEFFICIENT, SOIL AND PLANT WATER STATUS FOR VITIS VINIFERA (CV GODELLO AND CV MENCÍA).

AGRICULTURAL WATER MANAGEMENT, 151, 52-63. HTTPS://DOI.ORG/10.1016/J.AGWAT.2014.10.020

CRUZ-BAUTISTA, F., RODRÍGUEZ, J. C., & MUNGUÍA, V. M. (2019). ¿LÁMINA O DOSIS DE RIEGO PARA RIEGO LOCALIZADO?

DONVIR, A., JAIN, A., & SARASWATHI, P. K. (2024). APPLICATION STATE MANAGEMENT (ASM) IN THE MODERN WEB AND MOBILE APPLICATIONS: HTTPS://DOI.ORG/10.48550/ARXIV.2407.19318

A COMPREHENSIVE REVIEW.ELMASRI, R., & NAVATHE, S. (2010). FUNDAMENTOS DE SISTEMAS DE BASES DE DATOS (5A ED). PEARSON

ADDISON WESLEY. GOOGLE. (2023). PRÁCTICAS RECOMENDADAS PARA USAR LOS SERVICIOS WEB DE LA API DE WEATHER | WEATHER API. GOOGLE FOR DEVELOPERS. HTTPS://DEVELOPERS.GOOGLE.COM/MAPS/DOCUMENTATION/WEATHER/WEB-SERVICE-BESTPRACTICES?HL=ES-419

HOPPER, M. (2006). EVAPOTRANSPIRACION DEL CULTIVO: GUIAS PARA DETERMINACION LOS REQUERIMIENTOS DE AGUA DE LOS .. FOOD & AGRICULTURE ORG.

IBM CORPORATION. (2025, JUNIO 16). ¿QUÉ ES LA NORMALIZACIÓN DE BASES DE DATOS? | IBM. HTTPS://WWW.IBM.COM/MX-ES/THINK/TOPICS/DATABASE-NORMALIZATION

INITIATIVE (WAI), W. W. A. (2023). ACCESSIBILITY, USABILITY, AND INCLUSION. WEB ACCESSIBILITY INITIATIVE (WAI). HTTPS://WWW.W3.ORG/WAI/FUNDAMENTALS/ACCESSIBILITY-USABILITY-INCLUSION/

KITSIOS, F., CHATZIDIMITRIOU, E., & KAMARIOTOU, M. (2023). THE ISO/IEC 27001 INFORMATION SECURITY

MANAGEMENT STANDARD: HOW TO EXTRACT VALUE FROM DATA IN THE IT SECTOR. SUSTAINABILITY, 15(7), 5828. HTTPS://DOI.ORG/10.3390/SU15075828

LIU, G., HUANG, B., LIANG, Z., QIN, M., ZHOU, H., & LI, Z. (2020). MICROSERVICES: ARCHITECTURE, CONTAINER, AND CHALLENGES. 2020 IEEE 20TH INTERNATIONAL

Conference on Software Quality, Reliability and Security Companion (QRS-C), 629-635. https://doi.org/10.1109/QRS-C51114.2020.00107

Loeliger, J. (2009). Version control with Git (1st ed). O'Reilly Media, Inc.

Lorenz, D. (2024). Building production-grade web applications with Supabase: A comprehensive guide to database design, security, real-time data, storage, multi-tenancy, and more. Packt Publishing.

Manis, E. S., Rosa, R. J., Denegri, G. A., & Gaspari, F. J. (2024). Estimación del coeficiente de cultivo Kc para los cultivos de invierno y pastizales en la cuenca alta del río Sauce Chico de la Provincia de Buenos Aires a partir de sensores remotos. Revista de la Facultad de Agronomía, 123(1), 136. https://doi.org/10.24215/16699513e136

Martín de Santa Olalla Mañas, F., López Fuster, P., & Calera Belmonte, A. (2005). Agua y agronomía. Mundi-Prensa.

Microsoft. (2022). Manage historical data in system-versioned temporal tables—SQL Server. https://learn.microsoft.com/en-us/sql/relational-databases/tables/manage-retention-ofhistorical-data-in-system-versioned-temporal-tables?view=sql-server-ver17

Riva, M. (2022). Real-World Next.js: Build scalable, high-performance, and modern web applications using Next.js, the React framework for production (First edition). Packt Publishing.

Roselin, S., & Rodrigues, Dr. P. (2017). Software Architecture- Evolution and Evaluation. International Journal of Advanced Computer https://doi.org/10.14569/IJACSA.2012.030814 Science and Applications, 3(8).

Sabbag Filho, N. (2025). Comparison between Client-Side Rendering and Server-Side Rendering: Impacts on Performance and User Experience. https://doi.org/10.5281/ZENODO.14914792

Shaxson, F. (with Barber, R.). (2005). Optimización de la humedad del suelo para la producción vegetal: El significado de la porosidad del sueño. FAO.

Shklar, L., & Rosen, R. (2012). Web application architecture: Principles, protocols and practices (2. ed., reprinted). Wiley.

Stowe, M. (2019). Undisturbed REST: a Guide to Designing the Perfect API (First-Look Edition). Lulu.com.

Trezza, R. (2008). ESTIMACIÓN DE EVAPOTRANSPIRACIÓN DE REFERENCIA A NIVEL MENSUAL EN VENEZUELA. ¿CUÁL MÉTODO UTILIZAR?

VERDOODT, A., RANST, E. V., YE, L., & VERPLANCKE, H. (2005). A DAILY MULTI-LAYERED WATER BALANCE MODEL TO PREDICT WATER AND OXYGEN AVAILABILITY IN TROPICAL CROPPING SYSTEMS. SOIL USE AND MANAGEMENT, 21(3), 312-321. HTTPS://DOI.ORG/10.1079/SUM2005328

VILLAZÓN GÓMEZ, J. A., NORIS NORIS, P., VÁZQUEZ MONTENEGRO, R. J., MARTÍN GUTIÉRREZ, G., & COBO VIDAL, Y. (2021). COMPARACIÓN DE MÉTODOS EMPÍRICOS PARA LA ESTIMACIÓN DE LA EVAPOTRANSPIRACIÓN DE REFERENCIA EN HOLGUÍN, CUBA. IDESIA (ARICA), 39(3), 103-109. HTTPS://DOI.ORG/10.4067/S071834292021000300103

WORLD REFERENCE BASE FOR SOIL RESOURCES 2022: INTERNATIONAL SOIL CLASSIFICATION SYSTEM FOR NAMING SOILS AND CREATING LEGENDS FOR SOIL MAPS (4.EDITION). (2022). INTERNATIONAL UNION OF SOIL SCIENCES.

XU, J., WAN, W., ZHU, X., ZHAO, Y., CHAI, Y., GUAN, S., & DIAO, M. (2023). EFFECT OF REGULATED DEFICIT IRRIGATION ON THE GROWTH, YIELD, AND IRRIGATION WATER PRODUCTIVITY OF PROCESSING TOMATOES UNDER DRIP IRRIGATION AND HTTPS://DOI.ORG/10.3390/AGRONOMY13122862

IX. **ANEXOS (Opcional).**