Arquitectura de una Plataforma Web para predicción de productividad de árboles frutales en predios agrícolas

Bastián Berríos Pérez

bastian.berrios@usm.cl

Lais San Martín Navarro
Profesora Guía

Francis Fuentes Chacana
Profesora Correferente

Resumen

El sector agrícola chileno enfrenta desafíos relacionados con la incertidumbre en la producción debido a la variabilidad climática y la necesidad de optimizar recursos limitados. En este contexto, la empresa **Neering** ha identificado la oportunidad de desarrollar una plataforma web que integra predicción de rendimiento frutal mediante Machine Learning.

El grupo de trabajo **Treering**, compuesto por 5 estudiantes de la universidad Federico Santa María han trabajado en el diseño y evaluación de la arquitectura de software de **AgroPredict** que se propone como solución a esta oportunidad.

La principal contribución de este trabajo es una arquitectura monolítica modular implementada en Django.

La arquitectura diseñada permite integración eficiente entre frontend web, servicios de Machine Learning y herramientas de cálculo agrícola, facilitando toma de decisiones relacionadas con selección de cultivos, planificación de cosecha y optimización de recursos hídricos para clientes de nivel gerencial en el área agrícola.

Palabras Clave: Arquitectura de Software; Sistemas Web; Microservicios; Predicción Agrícola; Django Framework

---

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): X Memoria o trabajo de título □ Tesis de Postgrado

Título del trabajo: Arquitectura de una Plataforma Web para predicción de productividad de árboles frutales en predios agrícolas

Nombre del candidato(a): Bastián Enrique Berríos Pérez

Carrera / Grado: Ingeniería en Informática

Campus: Viña del Mar **Departamento:** Electrotecnia e Informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Lais San Martín Navarro, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

X El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

□ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por **(marcar una opción)**:

□ 6 meses □ 12 meses □ 2 años □ 3 años □ 5 años □ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

_____

_____

_____

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 13/04/2026 **Firma:** _____

Estudiante o Candidato(a):

Fecha: 27/03/2026 **Firma:** _____

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

Glosario

API (Application Programming Interface): Interfaz de programación de aplicaciones que permite la comunicación entre diferentes componentes de software.

C4 Model: Modelo de documentación arquitectónica que estructura la visualización del software en cuatro niveles: Context (Contexto), Container (Contenedores), Component (Componentes) y Code (Código).

Django: Framework web de Python que sigue el patrón MVT (Model-View-Template) y proporciona funcionalidades integradas para desarrollo rápido.

FastAPI: Framework web moderno de Python optimizado para la creación de APIs de alto rendimiento con soporte asíncrono nativo.

Machine Learning (ML): Disciplina de la inteligencia artificial que permite a los sistemas aprender y mejorar automáticamente a partir de datos sin ser programados explícitamente.

Microservicios: Patrón arquitectónico que estructura una aplicación como una colección de servicios independientes, débilmente acoplados y desplegables de forma autónoma.

MVC (Model-View-Controller): Patrón arquitectónico que separa la lógica de datos (Model), presentación (View) y control de flujo (Controller).

MVP (Minimum Viable Product): Producto mínimo viable que contiene las funcionalidades esenciales para validar un concepto con usuarios reales.

ODEPA (Oficina de Estudios y Políticas Agrarias): Organismo del Ministerio de Agricultura de Chile que genera información estadística y estudios del sector agrícola.

ORM (Object-Relational Mapping): Técnica de programación que convierte datos entre sistemas de tipos incompatibles usando objetos en lenguajes orientados a objetos.

REST (Representational State Transfer): Estilo arquitectónico para sistemas distribuidos basado en el protocolo HTTP.

RPS (Requests Per Second): Métrica de rendimiento que mide cuántas solicitudes HTTP puede procesar un servidor por segundo.

Tabla de contenido

Introducción 4

Capítulo 1: Definición del Problema 7

1.1 Contexto General 7

1.2 Problemática Tecnológica 7

1.3 Necesidad de Herramientas Digitales 8

1.4 Actores Involucrados 8

1.5 Competencia y Estado Actual 9

1.6 Objetivos y Alcances del Trabajo de Título 9

Capítulo 2: Marco Conceptual, Estado del Arte y Rol de Trabajo 10

2.1 Arquitecturas de Sistemas Web Modernos 10

2.2 Frameworks Web para Python en Contextos de Machine Learnin 12

2.3 Sistemas Predictivos en Agricultura: Perspectiva Arquitectónica 13

2.4 Evaluación de Arquitecturas de Software 14

2.5 Metodologías de Evaluación Arquitectónica 15

2.6 Rol del Arquitecto de Software en Proyectos Complejos 16

Capítulo 3: Diseño y Desarrollo de la Solución 17

3.1 Metodología de Desarrollo 17

3.2 Proceso de Selección de Framework 17

3.3 Diseño Arquitectónico del Sistema 21

3.4 Stack Tecnológico Seleccionado 23

3.5 Implementación del Prototipo 24

3.6 Funcionalidades Adicionales 26

3.7 Integración con Componentes del Equipo 27

Capítulo 4: Validación de la Solución 29

4.1 Metodología de Validación 29

4.2 Pruebas de Rendimiento de Frameworks 29

4.3 Limitaciones Identificadas 30

Capítulo 5: Conclusiones y Recomendaciones 31

5.1 Alcances y Logros del Trabajo 31

5.2 Contribuciones del Trabajo 32

5.3 Limitaciones y Desafíos Enfrentados 33

5.4 Aprendizajes Personales 34

5.5 Trabajo Futuro y Recomendaciones 35

5.6 Impacto Esperado 36

Introducción

El presente trabajo documenta el proceso de diseño, evaluación e implementación de la arquitectura de software para AgroPredict, una plataforma web que incluye herramientas de predicción de producción frutal destinada al sector agrícola chileno. Este documento describe el recorrido técnico realizado para fundamentar decisiones arquitectónicas críticas que garantizan la viabilidad, escalabilidad y mantenibilidad del sistema.

Contexto y Motivación

El sector agrícola chileno representa uno de los pilares económicos del país, generando empleo y exportaciones significativas. Sin embargo, enfrenta una creciente incertidumbre productiva debido a la variabilidad climática, escasez hídrica y un mercado volátil. Esta situación demanda herramientas tecnológicas que permitan a los productores tomar decisiones informadas basadas en datos y predicciones confiables.

En este contexto, surge la necesidad de desarrollar sistemas digitales accesibles que hagan uso de tecnologías predictivas.

La empresa **Neering**, patrocinadora de este proyecto, identificó la oportunidad de crear una plataforma que integre inteligencia artificial, datos geoespaciales y tecnologías web modernas para ofrecer predicciones precisas de producción frutal.

Definición del Problema Abordado

El desafío central de este trabajo radica en la definición de una arquitectura de software robusta para un sistema que debe:

- Integrar datos heterogéneos de múltiples fuentes (climáticos, catastrales, geoespaciales)
- Comunicarse eficientemente con modelos de Machine Learning para entregar predicciones en tiempo real
- Proporcionar una experiencia de usuario fluida a través de una interfaz web intuitiva
- Garantizar escalabilidad para soportar crecimiento futuro de usuarios y funcionalidades
- Mantener costos de desarrollo y operación manejables para una *startup* tecnológica

La complejidad técnica del proyecto requiere seleccionar con cuidado las herramientas tecnológicas y definir patrones arquitectónicos que equilibren rendimiento, velocidad de desarrollo y mantenibilidad a largo plazo.

Propuesta de Solución y Objetivos

Este trabajo propone un enfoque sistemático para el diseño arquitectónico de AgroPredict, fundamentado en:

1. **Análisis comparativo de frameworks web**: Evaluación técnica de Django, FastAPI y Laravel mediante criterios objetivos de rendimiento, ecosistema de herramientas y curva de aprendizaje.

2. **Diseño arquitectónico documentado**: Aplicación del modelo C4 para documentar la arquitectura del sistema en múltiples niveles de abstracción, facilitando la comunicación técnica con el equipo de desarrollo.

3. **Implementación de prototipo funcional**: Desarrollo de un MVP que valida las decisiones arquitectónicas y demuestra la viabilidad técnica de la solución propuesta.

El objetivo general del trabajo es diseñar y evaluar la arquitectura de un sistema web enfocado en la producción frutal, comparando frameworks de desarrollo y analizando su impacto en el rendimiento, escalabilidad y mantenimiento del software, para sustentar la implementación exitosa de la plataforma AgroPredict.

Marco Teórico Adoptado

El desarrollo arquitectónico de AgroPredict se fundamenta en principios y patrones reconocidos de la ingeniería de software:

- **Patrones arquitectónicos**: Uso del patrón MVC (Model-View-Controller) para separación de responsabilidades y facilitar el mantenimiento del código.

- **Principios SOLID**: Aplicación de principios de diseño orientado a objetos que promueven cohesión alta y acoplamiento bajo entre componentes.

- **Modelo C4 para documentación**: Uso del modelo *Context-Container-Component-Code* para visualizar la arquitectura en diferentes niveles de detalle, facilitando la comunicación con *stakeholders* técnicos y no técnicos.

- **Arquitectura API-First**: Diseño de interfaces de programación bien definidas que permiten la integración eficiente entre frontend, backend y servicios de Machine Learning.

Metodología Aplicada

El trabajo fue distribuido en investigación aplicada, análisis comparativo y validación experimental, que a su vez se desarrollaron por fases:

Fase 1: Investigación y Análisis Comparativo - Revisión sistemática de literatura sobre arquitecturas de software y frameworks web modernos - Definición de criterios de evaluación objetivos (rendimiento, escalabilidad, ecosistema, curva de aprendizaje) - Análisis comparativo mediante matrices de decisión.

Fase 2: Diseño Arquitectónico - Aplicación de metodologías de diseño arquitectónico (ATAM - Architecture Tradeoff Analysis Method) - Documentación mediante modelo C4 en múltiples niveles.

Fase 3: Validación Experimental - Implementación de prototipo funcional
- Pruebas de benchmarking de rendimiento entre frameworks candidatos.

Fase 4: Integración con Equipo Multidisciplinario - Coordinación con metodología Scrum del proyecto grupal - Entregas incrementales mediante sprints semanales - Iteración basada en retroalimentación del equipo y patrocinador.

Organización del Documento

El presente documento se estructura en los siguientes capítulos:

Capítulo 1: Definición del Problema - Describe el contexto del sector agrícola chileno, la problemática tecnológica identificada, los actores involucrados y el alcance del trabajo de título.

Capítulo 2: Marco Conceptual, Estado del Arte y Rol de Trabajo - Presenta los fundamentos teóricos sobre arquitecturas de software modernas, frameworks web para Python, sistemas predictivos en agricultura, metodologías de evaluación arquitectónica y el rol de un arquitecto de software en un proyecto.

Capítulo 3: Diseño y Desarrollo de la Solución - Documenta el proceso de selección de framework, el diseño arquitectónico del sistema mediante modelo C4, la definición del stack tecnológico y la implementación del prototipo funcional.

Capítulo 4: Validación de la Solución - Dispone los resultados de las pruebas de rendimiento, la evaluación de calidad arquitectónica, el análisis de trade-offs y las limitaciones identificadas.

Capítulo 5: Conclusiones y Recomendaciones - Resume los alcances y logros del trabajo, las contribuciones realizadas, los aprendizajes obtenidos y las recomendaciones para trabajo futuro.

Capítulo 1: Definición del Problema

1.1 Contexto General

El sector agrícola chileno enfrenta una incertidumbre creciente en la planificación productiva debido a múltiples factores: variabilidad climática, escasez hídrica, fluctuaciones de mercado y la necesidad de optimizar recursos limitados. Esta situación puede generar pérdidas económicas significativas y dificulta la toma de decisiones estratégicas en las empresas frutícolas del país.

La agricultura de precisión y las tecnologías de predicción emergen como soluciones prometedoras para enfrentar estos desafíos. Sin embargo, el acceso a estas tecnologías ha estado históricamente limitado a grandes productores con recursos suficientes para invertir en sistemas costosos. Por lo que existe una necesidad de universalizar el acceso a herramientas predictivas mediante plataformas digitales de bajo costo y alta accesibilidad.

1.2 Problemática Tecnológica

La empresa **Neering**, patrocinadora del proyecto AgroPredict, ha identificado la necesidad de desarrollar una plataforma tecnológica que permita a los gerentes y administradores de empresas agrícolas predecir la producción frutal de sus predios.
El desafío principal radica en crear una arquitectura de software robusta, escalable y mantenible que integre datos heterogéneos (climáticos, catastrales, geoespaciales) y proporcione predicciones precisas a través de una interfaz web intuitiva.

Los retos técnicos específicos incluyen:

- **Integración de datos heterogéneos**: El sistema debe consumir información de múltiples fuentes con formatos diversos, incluyendo datos de la ODEPA y variables climáticas de modelos ERA5.
- **Comunicación eficiente con servicios de ML**: La arquitectura debe facilitar la comunicación en tiempo real entre la aplicación web y modelos de Machine Learning complejos, minimizando latencia.
- **Escalabilidad técnica**: El diseño debe contemplar crecimiento futuro en número de usuarios, volumen de datos procesados y complejidad de modelos predictivos.
- **Mantenibilidad a largo plazo**: La arquitectura debe facilitar la incorporación de nuevas funcionalidades, actualización de modelos y corrección de errores sin comprometer la estabilidad del sistema.

1.3 Necesidad de Herramientas Digitales

Si bien en el sector agrícola existe el uso de sensores, drones y otras tecnologías, la alta inversión inicial detiene a una buena parte de este sector en incluir estas herramientas, por lo que deben utilizar métodos tradicionales, como modelos estadísticos simples y la experiencia propia.

Debido a lo indicado existe una demanda de sistemas digitales de bajo costo, que permitan el uso de tecnologías predictivas avanzadas

Los productores agrícolas chilenos requieren herramientas que:

- Proporcionen predicciones confiables basadas en datos científicos y modelos validados
- Sean accesibles mediante interfaces web intuitivas que no requieran conocimientos técnicos especializados
- Ofrezcan información accionable que facilite la toma de decisiones gerénciales (planificación de cosecha, gestión de recursos)
- Mantengan costos accesibles para pequeños y medianos productores

1.4 Actores Involucrados

El proyecto AgroPredict involucra múltiples actores con roles y expectativas específicas:

Patrocinador: Empresa Neering

-Empresa tecnológica chilena enfocada en soluciones digitales a la medida.
-Provee dirección estratégica.
-Espera un producto mínimo viable (MVP) técnicamente robusto y comercialmente factible.

Usuarios Finales: Gerentes y Administradores Agrícolas

-Profesionales con formación técnica agrícola, pero conocimientos tecnológicos variables.
-Requieren información precisa y oportuna para planificación productiva y comercial.
-Valoran simplicidad de uso y confiabilidad de predicciones sobre complejidad técnica.

Equipo de Desarrollo

El equipo de trabajo de este proyecto es **Treering**, que está compuesto por 5 integrantes con diversos roles de trabajo:

- **Matías Aguayo (UI/UX)**: Diseño de interfaces centradas en el usuario para perfiles gerenciales, validación de flujos de interacción.

- **Bastián Berríos (Arquitectura de Software)**: Diseño y evaluación de la estructura tecnológica, selección de frameworks.

- **Sebastián Lagos (Machine Learning)**: Desarrollo de redes neuronales para predicción de producción frutal, integración de datos de ODEPA y ERA5.

- **Javier Lucero (Desarrollo Back-End)**: Implementación de funcionalidades backend, integración de componentes.

- **Álvaro Marchant (Gestión de Proyecto)**: Implementación de metodologías ágiles Scrum, coordinación del equipo y seguimiento de entregables.

1.5 Competencia y Estado Actual

El mercado de tecnologías agrícolas en Chile presenta soluciones fragmentadas con limitaciones significativas:

Herramientas Tradicionales

–Como anteriormente mencionados, se utilizan métodos empíricos basados en experiencia histórica de productores.
- Modelos estadísticos simples que no capturan relaciones no lineales complejas.
- Baja precisión y falta de personalización por características específicas de cada predio.

Soluciones Tecnológicas Existentes

- Plataformas internacionales con datos no adaptados al contexto chileno.
- Sistemas costosos dirigidos exclusivamente a grandes productores.
- Herramientas especializadas que requieren conocimientos técnicos avanzados.

Oportunidad de Innovación AgroPredict se posiciona como una solución que:

- Utiliza datos específicos del contexto agrícola chileno (ODEPA).
- Ofrece accesibilidad mediante interfaz web simple e intuitiva.
- Mantiene costos bajos mediante arquitectura optimizada y tecnologías open-source.
- Proporciona predicciones basadas en Machine Learning con datos satelitales.

1.6 Objetivos y Alcances del Trabajo de Título

Objetivo General

Diseñar y evaluar la arquitectura de un sistema web de predicción de producción frutal, comparando frameworks de desarrollo y analizando su impacto en el rendimiento, escalabilidad y mantenimiento del software, para sustentar la implementación exitosa de la plataforma AgroPredict.

Objetivos Específicos

Objetivo 1: Análisis Comparativo de Frameworks Web

- Comparar frameworks destacados en creación de aplicaciones web (Django, FastAPI, Laravel)
- Investigar metodologías de evaluación sistemática de tecnologías web
- Evaluar qué criterios se usan en sistemas que integran Machine Learning

Objetivo 2: Diseño Arquitectónico del Sistema

- Definir la arquitectura del sistema AgroPredict considerando patrones escalables
- Evaluar la adopción de arquitecturas monolíticas vs. microservicios
- Diseñar las interfaces entre componentes del sistema (frontend, backend y ML)

Objetivo 3: Evaluación de Rendimiento y Calidad Arquitectónica

- Establecer métricas objetivas para evaluar la arquitectura propuesta
- Analizar el impacto de las decisiones técnicas en la mantenibilidad del software
- Validar el sistema mediante pruebas de rendimiento y benchmarking
- Documentar recomendaciones para futuras iteraciones

Alcances del Trabajo

Dentro del Alcance:

- Diseño arquitectónico completo del sistema web AgroPredict
- Comparación técnica fundamentada de frameworks de desarrollo web
- Implementación de prototipo funcional (MVP) con funcionalidades básicas
- Documentación arquitectónica mediante modelo C4
- Validación mediante pruebas de rendimiento y benchmarking

Fuera del Alcance:

- Desarrollo del modelo de Machine Learning
- Diseño detallado de interfaces de usuario
- Despliegue en producción y operación del sistema
- Implementación de arquitectura de microservicios completa

(Pues son las tareas asignadas a otros integrantes del grupo de trabajo)

Capítulo 2: Marco Conceptual, Estado del Arte y Rol de Trabajo

2.1 Arquitecturas de Sistemas Web Modernos

Patrones Arquitectónicos Fundamentales

Los sistemas web modernos se fundamentan en patrones arquitectónicos que proporcionan estructura y escalabilidad. En este proyecto se destacan los siguientes patrones:

Modelo-Vista-Controlador (MVC): Patrón fundamental que separa la lógica de datos (Model), presentación (View) y control de flujo (Controller), facilitando el mantenimiento y la escalabilidad. Este patrón ha demostrado ser efectivo para aplicaciones web complejas al permitir que equipos multidisciplinarios trabajen en diferentes capas del sistema sin generar conflictos significativos.

Arquitectura en Capas: Organización del sistema en capas lógicas (presentación, lógica de negocio, acceso a datos) que mejora la modularidad y facilita pruebas independientes de cada capa. La separación clara de responsabilidades reduce el acoplamiento y aumenta la cohesión dentro de cada capa.

API-First Design: Enfoque que prioriza el diseño de interfaces de programación para facilitar la integración con sistemas externos y servicios de Machine Learning. Este patrón es especialmente relevante para AgroPredict, donde la comunicación eficiente entre la aplicación web y los servicios de predicción es crítica.

Microservicios vs. Arquitectura Monolítica

La decisión entre diseñar arquitecturas monolíticas y de microservicios es crítica para sistemas como AgroPredict.

Arquitecturas Monolíticas:

- Ofrecen simplicidad inicial y menor costo de mantenimiento en las primeras fases del proyecto
- Facilitan el desarrollo rápido al eliminar la complejidad de comunicación entre servicios
- Reducen costos operacionales en términos de despliegue y monitoreo
- Son más eficientes para equipos pequeños con recursos limitados

Arquitecturas de Microservicios:

- Proporcionan escalabilidad independiente de componentes específicos
- Ofrecen mayor resiliencia al aislar fallos en servicios individuales
- Facilitan la actualización independiente de componentes sin afectar el sistema completo
- Requieren mayor conocimiento técnico y herramientas de orquestación sofisticadas

Para el inicio de este proyecto, se propuso crear una *estructura monolítica* bien estructurada, con el fin de cumplir con el limitado tiempo de trabajo. Se considera la migración a una arquitectura de microservicios como propuesta futura. Esto se verá más justificado en los capítulos siguientes.

Principios SOLID en Arquitectura Web

La correcta aplicación de principios SOLID en el diseño de sistemas web contribuye significativamente a la mantenibilidad:

Responsabilidad Única (Single Responsibility): Cada componente debe tener una razón específica para cambiar. En AgroPredict, esto implica separar claramente la lógica de autenticación, procesamiento de predicciones y presentación de resultados.

Abierto/Cerrado (Open-Closed): Los módulos deben estar abiertos para extensión, pero cerrados para modificación. La arquitectura de AgroPredict debe permitir agregar nuevos tipos de cultivos sin modificar el código existente.

Sustitución de Liskov (Liskov Substitution): Los objetos derivados deben poder sustituir a sus objetos base sin afectar la funcionalidad del sistema.

Segregación de Interfaces (Interface Segregation): Interfaces específicas son preferibles a interfaces generales. Las APIs de AgroPredict deben proporcionar endpoints especializados para diferentes tipos de consultas.

Inversión de Dependencias (Dependency Inversion): Los módulos de alto nivel no deben depender de módulos de bajo nivel; ambos deben depender de abstracciones. Esto facilita la sustitución de componentes como la base de datos o el servicio de Machine Learning.

2.2 Frameworks Web para Python en Contextos de Machine Learning

Django Framework

Django se ha consolidado como una opción robusta para la creación de aplicaciones web con una filosofía "batteries included", donde las funcionalidades del framework son de fácil acceso e implementación.

Características Relevantes para AgroPredict

Sistema de Autenticación Robusto: Manejo seguro de usuarios, sesiones y permisos incorporados en el framework.

Arquitectura MVT Clara: Variante del patrón MVC adaptada para desarrollo web que separa Models, Views y Templates.

Django Rest Framework: Extensión que facilita la creación de APIs REST para integración con modelos de Machine Learning.

Ventajas para Desarrollo Rápido: Django permite implementar prototipos funcionales rápidamente gracias a su amplio conjunto de herramientas integradas. Para un equipo con recursos limitados y plazos ajustados, esta característica representa una ventaja significativa.

Integración con ML: Frameworks como Django Rest Framework facilitan la exposición de modelos de Machine Learning como servicios web mediante APIs REST. La integración con bibliotecas como Scikit-learn y TensorFlow es directa gracias al ecosistema Python común.

Flask Framework

Flask ofrece un enfoque minimalista y flexible para el desarrollo web, proporcionando las herramientas esenciales y permitiendo al desarrollador seleccionar componentes adicionales según necesidad.

Ventajas:

- Flexibilidad arquitectónica máxima que permite personalización total del stack tecnológico.
- Curva de aprendizaje suave ideal para desarrolladores principiantes.
- Control granular sobre componentes y configuración del sistema.

Desafíos:
Requiere toma de decisiones arquitectónicas adicionales que pueden retrasar el desarrollo.
Menos funcionalidades incluidas por defecto, necesitando integración manual de bibliotecas externas.

Contexto ML: Flask es ampliamente usado en proyectos de ciencia de datos por su simplicidad para crear APIs ligeras que exponen modelos predictivos.

FastAPI

FastAPI representa una opción moderna optimizada para la creación de APIs de alto rendimiento con características avanzadas.

Ventajas:

- **Documentación Automática**: Genera documentación interactiva OpenAPI automáticamente.
- **Validación Automática de Datos**: Usa Python type hints para validar datos de entrada y salida.

Aplicabilidad: Especialmente adecuado para servicios de Machine Learning que requieren procesamiento asíncrono y APIs de alta frecuencia. Su arquitectura está optimizada para microservicios y comunicación entre componentes distribuidos.

Para este proyecto, si bien FastAPI tiene buen rendimiento, su implementación tiene un nivel aumentado de complejidad comparado al resto.

2.3 Sistemas Predictivos en Agricultura: Perspectiva Arquitectónica

Requisitos Específicos del Dominio Agrícola

Los sistemas agrícolas predictivos presentan características únicas que influyen en decisiones arquitectónicas:

- **Procesamiento de Datos Heterogéneos**: El sistema debe integrar datos climáticos (temperatura, precipitación, humedad), satelitales (índices de vegetación NDVI, EVI), catastrales (superficie plantada, tipo de suelo) y administrativos (histórico de producción, prácticas de manejo). Cada tipo de dato tiene formatos, frecuencias de actualización y niveles de confiabilidad diferentes.

- **Escalabilidad Estacional**: Los sistemas agrícolas experimentan picos de uso durante épocas específicas del año (temporadas de cosecha, períodos de planificación). La arquitectura debe soportar estas variaciones sin degradación significativa de rendimiento.

- **Latencia Variable Aceptable**: A diferencia de sistemas financieros o de comercio electrónico, las predicciones agrícolas no requieren latencia en milisegundos. Tiempos de respuesta de 2-5 segundos son generalmente aceptables, permitiendo arquitecturas menos complejas.

- **Actualización de Modelos**: Los modelos predictivos pueden requerir reentrenamiento periódico con nuevos datos. La arquitectura debe facilitar la actualización de modelos sin interrumpir el servicio.

Patrones de Integración con Machine Learning

Para lograr integración efectiva entre sistemas web y modelos de Machine Learning se recomiendan patrones específicos:

- **API Gateway**: Centralización del acceso a servicios de predicción con gestión de autenticación. Este patrón simplifica la comunicación entre frontend y servicios de backend especializados.

- **Model Serving**: Encapsulación de modelos ML como servicios independientes que pueden escalarse y actualizarse sin afectar la aplicación web principal.

2.4 Evaluación de Arquitecturas de Software

Métricas de Calidad Arquitectónica

Para realizar evaluación objetiva de arquitecturas de software se requieren métricas específicas y medibles:

Métricas de Mantenibilidad (ISO 25010):

- **Cohesión de Módulos**: Grado en que elementos dentro de un módulo están relacionados funcionalmente. Se evalúa mediante análisis estático de código.
- **Acoplamiento entre Componentes**: Nivel de dependencia entre módulos diferentes, medido a través de análisis de dependencias. Acoplamiento bajo facilita cambios sin efectos colaterales.

Métricas de Escalabilidad:

- **Rendimiento (Throughput)**: Medido en solicitudes por segundo (RPS) bajo diferentes cargas de trabajo
- **Utilización de Recursos**: Consumo de CPU y memoria durante operación normal y picos de carga
- **Tiempo de Respuesta**: Latencia promedio y percentiles bajo carga variable

Métricas de Disponibilidad:

- **Uptime**: Porcentaje de tiempo que el sistema está operativo
- **MTBF (Mean Time Between Failures)**: Tiempo promedio entre fallos
- **MTTR (Mean Time To Recover)**: Tiempo promedio para recuperación después de fallos.

Trade-offs Arquitectónicos

Toda decisión arquitectónica implica compromisos entre atributos de calidad:

Rendimiento vs Mantenibilidad: Optimizaciones de rendimiento frecuentemente aumentan complejidad del código, dificultando mantenimiento. Para AgroPredict, se prioriza mantenibilidad en fases iniciales, optimizando rendimiento solo en componentes críticos.

Flexibilidad vs Complejidad: Arquitecturas altamente flexibles (como microservicios) introducen complejidad operacional significativa. Para un MVP, se acepta menor flexibilidad a cambio de simplicidad.

Costos de Desarrollo vs Costos Operacionales: Frameworks con muchas funcionalidades integradas (Django) aceleran desarrollo, pero pueden generar costos innecesarios en producción. Frameworks minimalistas (Flask) requieren más desarrollo inicial, pero ofrecen control fino sobre recursos.

2.5 Metodologías de Evaluación Arquitectónica

Architecture Tradeoff Analysis Method (ATAM)

ATAM proporciona un marco sistemático para evaluar arquitecturas de software identificando cómo las decisiones arquitectónicas afectan atributos de calidad:

Fases del Método:

1. **Presentación**: Exposición de los drivers de negocio y arquitectura propuesta.
2. **Investigación y Análisis**: Identificación de escenarios de atributos de calidad y enfoques arquitectónicos.
3. **Pruebas**: Análisis de sensibilidad y trade-offs.
4. **Reporte**: Documentación de hallazgos, riesgos y no-riesgos.

Aplicación a AgroPredict: Este método fue adoptado para evaluar los frameworks seleccionados, identificando que Django ofrece el mejor balance entre velocidad de desarrollo y escalabilidad futura.

Documentación Arquitectónica con Modelo C4

El modelo C4 (Context, Containers, Components, Code) ofrece una aproximación estructurada para documentar arquitecturas de software en múltiples niveles de abstracción:

Nivel 1 - Context (Contexto): Muestra el sistema como una caja rodeada de usuarios y sistemas externos con los que interactúa. Para AgroPredict, identifica usuarios finales (gerentes agrícolas), servicios externos (ERA5, ODEPA) y la plataforma web como sistema central.

Nivel 2 - Container (Contenedores): Muestra las aplicaciones y servicios que componen el sistema. AgroPredict se descompone en; aplicación web frontend, API backend Django, base de datos PostgreSQL y servicio de ML.

Nivel 3 - Component (Componentes): Detalla los módulos principales de cada contenedor. El backend Django se divide en; módulo de autenticación, API de predicciones, gestor de datos y orquestador de servicios ML.

Nivel 4 - Code (Código): Muestra clases e implementación específica, generalmente mediante diagramas UML. Este nivel se documenta para componentes críticos o complejos.

Utilizado para facilitar la comunicación entre integrantes del equipo de trabajo **Treering**.

2.6 Rol del Arquitecto de Software en Proyectos Complejos

El arquitecto de software es responsable de traducir requisitos técnicos complejos en estructuras viables que faciliten desarrollo coordinado.
A diferencia del desarrollador que implementa código específico, el arquitecto define la "columna vertebral" del sistema que permite que múltiples roles especializados trabajen coherentemente.

En proyectos como AgroPredict, donde coexisten especialidades en Machine Learning, diseño UI/UX, desarrollo backend y gestión de proyecto, el arquitecto actúa como "integrador técnico" que:

- Define criterios de evaluación para selecciones tecnológicas críticas
- Documenta la estructura del sistema mediante metodologías reconocidas
- Diseña interfaces claras entre componentes para permitir trabajo paralelo
- Anticipa decisiones que habilitan evolución futura del sistema
- Facilita comunicación técnica entre roles especializados

Las siguientes secciones del Capítulo 3 documentan cómo estas responsabilidades se aplicaron específicamente en el diseño de AgroPredict.

Capítulo 3: Diseño y Desarrollo de la Solución

3.1 Metodología de Desarrollo

Integración con Scrum

El desarrollo de AgroPredict se integró con la metodología ágil Scrum adoptada por el equipo completo, facilitando coordinación entre roles y permitiendo entregas incrementales de valor.

- **Organización de Sprints**: El proyecto se estructuró en sprints semanales con ceremonias regulares:
- **Sprint Planning**: Definición de objetivos arquitectónicos y tareas técnicas para la semana
- **Daily**: Sincronización diaria con equipo para identificar bloqueos y dependencias
- **Sprint Review**: Demostración de avances arquitectónicos y prototipos funcionales
- **Sprint Retrospective**: Identificación de mejoras en procesos y comunicación técnica

Herramientas de Colaboración

GitHub para Control de Versiones: Repositorio centralizado que facilitó:

- Versionado de código del prototipo
- Revisión de código entre pares
- Documentación técnica mediante README y wikis
- Seguimiento de problemas técnicos

ClickUp para Gestión de Tareas: Sincronización de tareas con backlog general del proyecto, permitiendo visibilidad de progreso para todo el equipo y patrocinador.

Discord para Comunicación: Canal dedicado para discusiones técnicas, resolución rápida de dudas y coordinación con equipo de Neering. Se implementó bot para transcripción automática de reuniones importantes.

3.2 Proceso de Selección de Framework

Definición de Criterios de Evaluación

La selección del framework web para AgroPredict requirió definir criterios objetivos alineados con las necesidades del proyecto:

Criterio 1: Rendimiento (Requests Per Second)

- Métrica: Solicitudes por segundo (RPS) soportadas bajo carga típica
- Relevancia: Garantizar experiencia fluida para múltiples usuarios concurrentes

Criterio 2: Curva de Aprendizaje

- Métrica: Tiempo de inicio dado el plazo ajustado del proyecto
- Evaluación: Experiencia previa del equipo y disponibilidad de documentación

Criterio 3: Ecosistema de Machine Learning

- Métrica: Facilidad de integración con bibliotecas ML
- Relevancia: Comunicación eficiente entre aplicación web y servicios de predicción
- Evaluación: Disponibilidad de bibliotecas de integración y ejemplos documentados

Criterio 5: Escalabilidad Futura

- Métrica: Capacidad de evolución hacia arquitecturas distribuidas
- Relevancia: Permitir crecimiento del sistema sin reescritura completa

Matriz de Comparación

Framework	Rendimiento (RPS)	Curva Aprendizaje	Ecosistema ML	Velocidad Desarrollo	Escalabilidad	Decisión
Django	~4,500	Media	Excelente	Alta	Buena	**MVP**
FastAPI	~6,250	Baja	Excelente	Media	Excelente	Futuro
Flask	~5,000	Baja	Bueno	Media-Baja	Variable	Descartado

Tabla 1: Comparación de frameworks web evaluados para AgroPredict.

Fuente: Elaboración propia basada en benchmarking interno.

Decisión Fundamentada

Selección para MVP: Django

Django fue seleccionado como framework principal para el MVP de AgroPredict basándose en:

1. **Balance óptimo para fase inicial**: Aunque FastAPI ofrece rendimiento superior (~40% más RPS), Django proporciona velocidad de desarrollo significativamente mayor gracias a funcionalidades integradas (ORM, autenticación, admin panel).

2. **Ecosistema Python unificado**: Tanto la aplicación web como el modelo de Machine Learning usan Python, facilitando integración y uso compartido de código entre componentes.

3. **Reducción de riesgo técnico**: La madurez de Django y su extensa documentación minimizan riesgo de bloqueos técnicos en un proyecto con plazo ajustado.

3.3 ¿Cuándo usar estructuras monolíticas vs microservicios?

Aunque la migración a microservicios es una propuesta válida para fases futuras de AgroPredict, es importante reconocer que la arquitectura monolítica representa una decisión apropiada para el contexto específico AgroPredict.
La selección de un patrón arquitectónico debe ser función directa del contexto del proyecto, incluyendo factores como el tamaño del equipo, la complejidad del dominio, el plazo disponible y la madurez del producto.

Equipos Pequeños y Multidisciplinarios:
Las arquitecturas monolíticas son efectivas cuando el equipo de desarrollo es reducido (3-10 desarrolladores) y los miembros trabajan en múltiples capas del sistema.
En el caso de AgroPredict, con tres desarrolladores la arquitectura monolítica facilita la comunicación directa entre componentes sin necesidad de canales de comunicación complejos entre servicios independientes.
Este beneficio se vuelve aún más relevante considerando que el equipo no tiene experiencia previa en uso de containers o administración de infraestructura distribuida.

Fases Iniciales y Validación de Conceptos:
La arquitectura monolítica es óptima durante las fases de validación de producto (MVP y primeras iteraciones), donde el objetivo principal es demostrar viabilidad técnica y comercial del concepto con el menor tiempo posible de desarrollo.
AgroPredict se encuentra precisamente en esta fase inicial: donde se busca validar que un modelo de Machine Learning integrado con una interfaz web puede proporcionar predicciones útiles a gerentes agrícolas.

Requisitos de Baja Latencia Intra-Sistema:
Una ventaja técnica importante de los monolíticos es que la comunicación entre módulos ocurre dentro del mismo proceso, eliminando la latencia de red y serialización de datos que caracteriza a la comunicación entre microservicios.
Para AgroPredict, donde operaciones como validación de datos, consultas a base de datos y llamadas a servicios de predicción deben completarse en tiempos de respuesta razonables (2-5 segundos), esta ventaja es significativa.
Si bien microservicios otorgan más beneficios, el tiempo de latencia no se ajusta a la baja cantidad de clientes previstas para el MVP.

Costos de Operación y Despliegue Simplificados:
Mantener una arquitectura monolítica en producción requiere menos infraestructura y herramientas que una arquitectura de microservicios completa.
Un único contenedor o servidor ejecutando la aplicación Django es significativamente más simple de desplegar, monitorear y mantener que un conjunto de servicios independientes que requieren orquestación con Kubernetes, sistemas de rastreo entre módulos.

Dominio de Negocio Medianamente Complejo sin Equipos Independientes:
Las arquitecturas de microservicios son especialmente ventajosas cuando múltiples equipos independientes trabajan en el mismo sistema y necesitan iterar a diferentes velocidades.
Sin embargo, AgroPredict es actualmente un sistema donde todas las funcionalidades (predicción de producción, calculadoras agrícolas, gestión de usuarios, visualización de datos) están fuertemente integradas en torno a un core de dominio: predicción agrícola.
No existe una división clara de dominio que justifique equipos completamente independientes en microservicios

Matriz de Decisión: Monolítico vs Microservicios

La siguiente matriz presenta criterios clave para evaluar cuándo es apropiada cada arquitectura, permitiendo identificar en qué contextos la decisión por monolítico es justificada.

Tabla 2: Tabla de comparación arquitecturas de software.
Fuente: Elaboración propia.

Criterio	Favorece Monolítico	Favorece Microservicios
Tamaño del equipo	< 10 desarrolladores	> 20 desarrolladores
Madurez del producto	MVP, concepto en validación	Producto maduro en producción
Uniformidad de tecnología	Un stack tecnológico principal	Múltiples tecnologías especializadas
Velocidad de desarrollo requerida	Rápido (< 6 meses)	Flexible (escalonado)
Requisitos de escalabilidad	Horizontal simple (replicar instancia)	Escalamiento independiente de componentes
Experiencia del equipo	DevOps/Kubernetes opcional; despliegue simple en servidor único	DevOps avanzado; Kubernetes requerida para manejo de múltiples servicios
Presupuesto de operación	Limitado	Abundante
Complejidad operacional aceptable	Baja	Media-Alta

3.3 Diseño Arquitectónico del Sistema

Decisión: Arquitectura Monolítica Modular

Para el MVP de AgroPredict se adoptó una **arquitectura monolítica modular** en lugar de microservicios, ya que:

- **Simplicidad Operacional**: Un sistema monolítico bien estructurado reduce complejidad de despliegue.
- **Equipo Pequeño**: Con 3 desarrolladores, la coordinación de microservicios independientes agregaría complejidad innecesaria
- **MVP Enfocado**: La funcionalidad inicial no justifica la complejidad adicional de arquitectura distribuida

Modularidad Interna: Aunque monolítico, el sistema mantiene separación clara de responsabilidades:

- Módulo de a autenticación y gestión de usuarios
- Módulo de interfaz web
- Módulo de API REST para integración con ML
- Módulo de gestión de datos agrícolas
- Módulo de comunicación con servicios externos

Esta estructura modular facilita futura migración hacia microservicios si el crecimiento del sistema lo justifica.

Diagrama de Contexto (C4 - Nivel 1)

El diagrama de contexto muestra AgroPredict como sistema central interactuando con actores externos:

Actores Humanos

- **Gerente Agrícola**: Usuario principal que consulta predicciones para toma de decisiones gerenciales.
- **Administrador de Sistema**: Usuario con permisos elevados para gestión de datos y configuración.

Sistemas Externos:

- **ERA5**: Proveedor de datos satelitales y climáticos de alta resolución.
- **ODEPA**: Fuente de datos agrícolas históricos de Chile (producción, superficie plantada, precios).
- **Servicio de Machine Learning**: Componente que ejecuta modelos predictivos (inicialmente integrado).

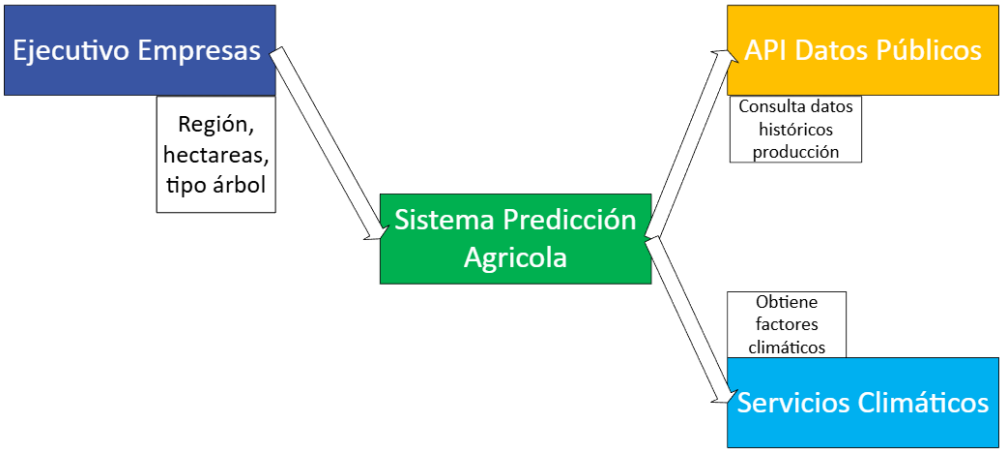

Figura 1: Diagrama de Contexto C4 del Sistema AgroPredict.
Fuente: Elaboración propia.

Diagrama de Contenedores (C4 - Nivel 2)

El nivel de contenedores descompone AgroPredict en sus aplicaciones y servicios principales:

Frontend Web Application (Django Templates):

- Renderiza interfaz de usuario HTML/CSS/JavaScript.
- Maneja sesiones de usuario y autenticación.
- Presenta resultados de predicciones en formato visual comprensible.

Backend API (Django + Django Rest Framework):

- Expone endpoints REST para operaciones CRUD.
- Orquesta comunicación con servicio de ML.
- Gestiona lógica de negocio y validaciones.
- Implementa autenticación basada en tokens.

Base de Datos (PostgreSQL/SQLite):

- Almacena usuarios, sesiones y configuraciones
- Persiste histórico de predicciones realizadas.
- Configuración: SQLite para desarrollo, PostgreSQL para futuro.

Servicio ML

- Carga modelos entrenados de Machine Learning.
- Ejecuta inferencias sobre datos de entrada del usuario.
- Retorna predicciones con intervalos de confianza.

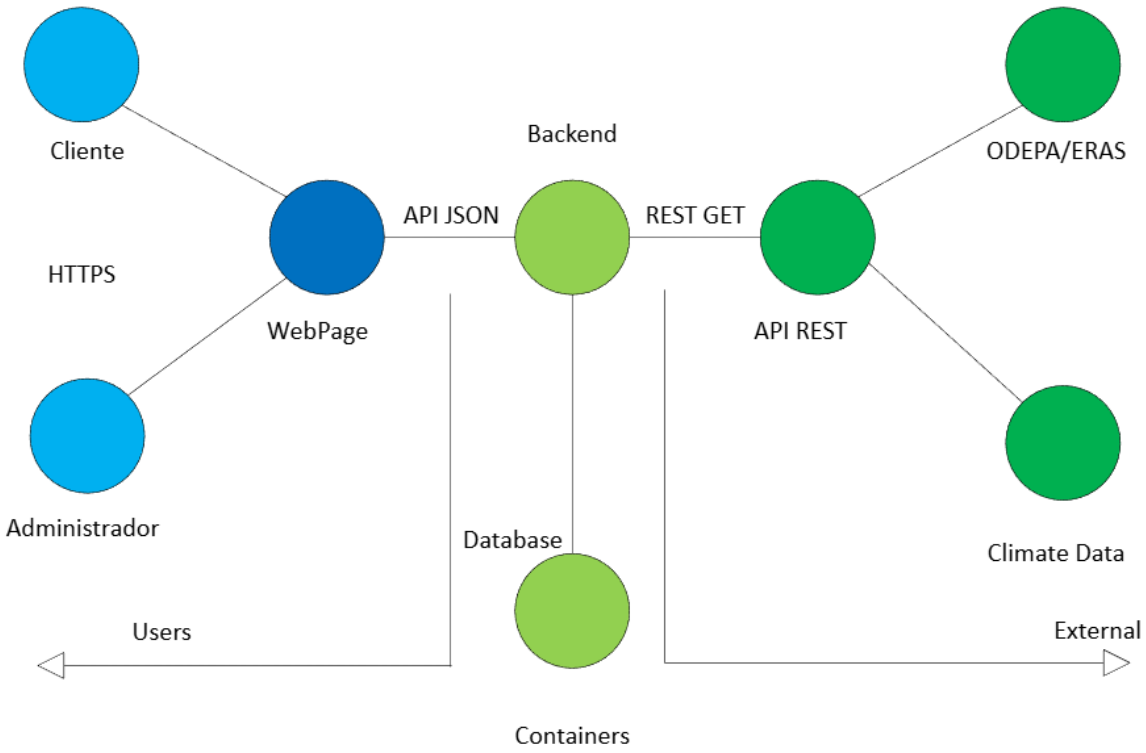

Figura 2: Diagrama de Contenedores C4 del Sistema AgroPredict.
Fuente: Elaboración propia.

3.4 Stack Tecnológico Seleccionado

Componentes del Stack

Backend Framework: Django 4.2+
- Framework web completo con ORM, autenticación y admin
- Django Rest Framework para APIs REST
- Configuración: SQLite (desarrollo), PostgreSQL (producción)

Frontend:
- Enfoque inicial: Django templates
- Bootstrap 5 para diseño responsivo
- JavaScript vanilla para interactividad básica -

Base de Datos: SQLite y PostgreSQL 14+
- Base de datos relacional para producción
- Soporte para tipos de datos avanzados

Machine Learning: Integración mediante API REST entre Django y servicio ML.

Control de Versiones: Git + GitHub - Repositorio centralizado para código.

Herramientas de Desarrollo: - Python 3.10+ - VS Code como IDE principal del equipo.

3.5 Implementación del Prototipo

Módulos Desarrollados

Módulo 1: Sistema de Autenticación
- Registro de usuarios con validación de email
- Login/Logout con manejo seguro de sesiones
- Perfiles de usuario con información agrícola (región, tipo de cultivo)

Módulo 2: Dashboard Administrativo
- Panel de control mostrando resumen de predicciones recientes
- Estadísticas de uso del sistema (número de consultas, usuarios activos)
- Gestión de datos de cultivos disponibles
- Implementación: Django admin personalizado con templates propios

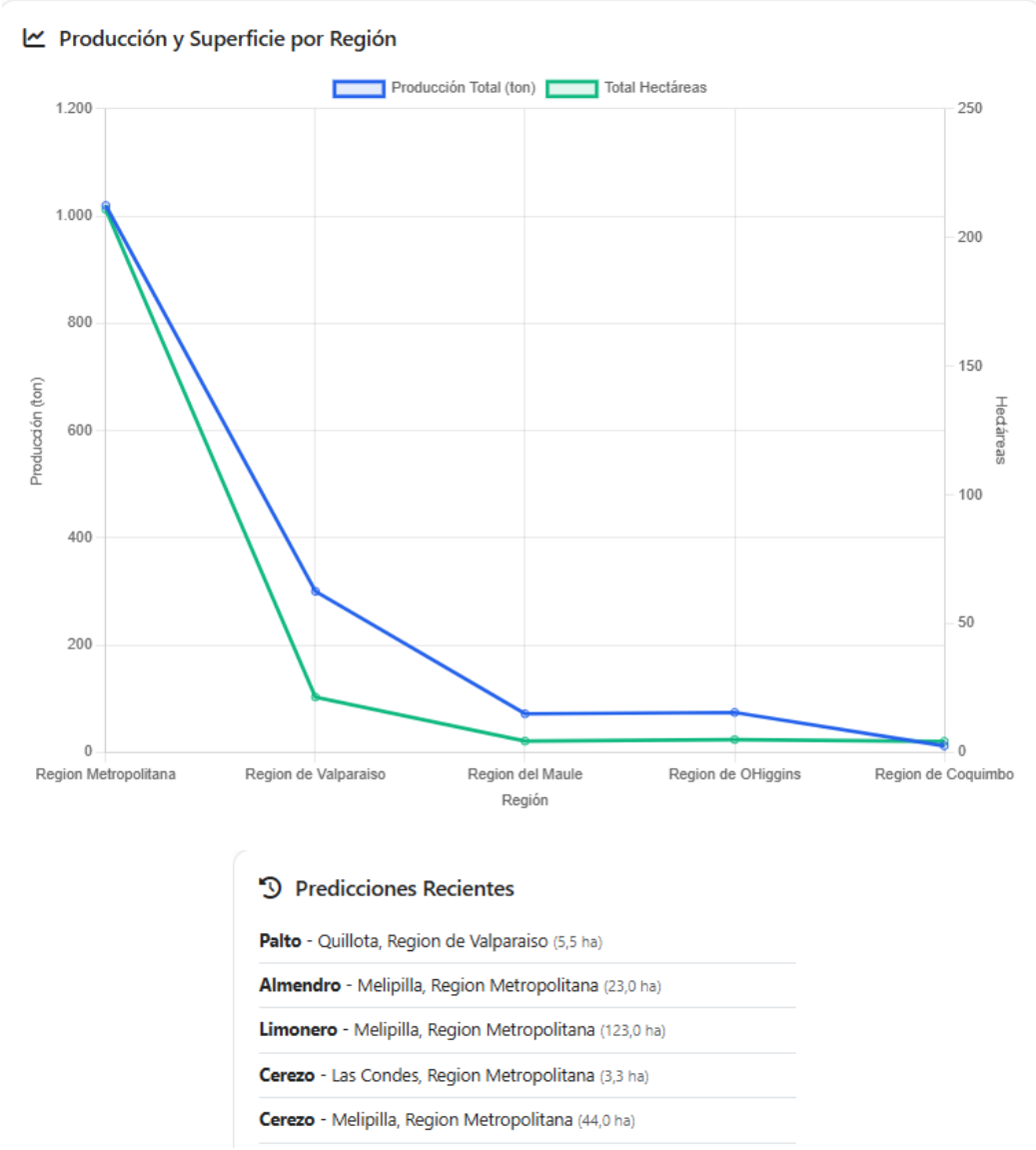

Figura 3: Captura de Dashboard Principal de AgroPredict mostrando interfaz de usuario gerencial
Fuente: Elaboración propia

Módulo 3: Formulario de Predicción

Interfaz para ingreso de datos de cultivo:

- Región geográfica - Superficie plantada (hectáreas)- Tipo de árbol frutal - Validación de datos en frontend y backend

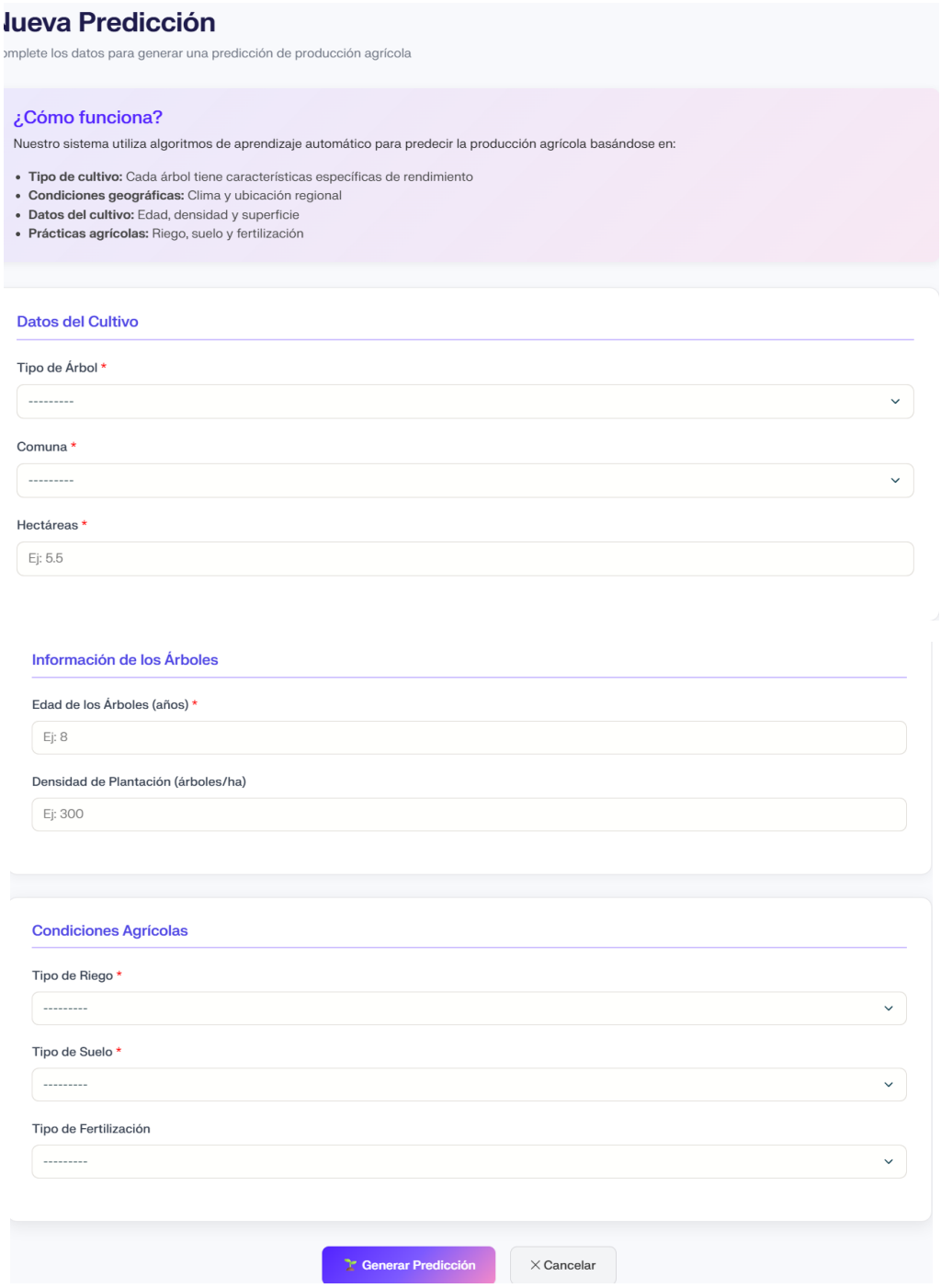

Figura 4: Formulario de ingreso de datos para predicción de producción frutal.
Fuente: Elaboración propia

Módulo 5: Presentación de Resultados

- Visualización de predicción en toneladas esperadas

Figura 5: Pantalla de resultados mostrando predicción de producción con intervalos de confianza
Fuente: Elaboración propia

3.6 Funcionalidades Adicionales

Además de la funcionalidad principal, la predicción con el modelo de ML, a lo largo del desarrollo del progreso, se recibió feedback por parte de profesores, el cual se acogió de la siguiente manera:

Por la limitante de tiempo y el limitado acceso a datos de calidad para generar un modelo predictivo robusto y preciso, se nos aconsejó añadir más herramientas a la página web.

Treering, decidió implementar calculadoras de uso frecuente en el área agrícola, como una calculadora de riego, balance hídrico y asistente de fertilización, por mencionar algunas.

Estas funcionalidades ayudan a la página a convertirse en una *"navaja suiza"* para el cliente, dándoles más información y una razón para el uso frecuente de la página además de las predicciones.

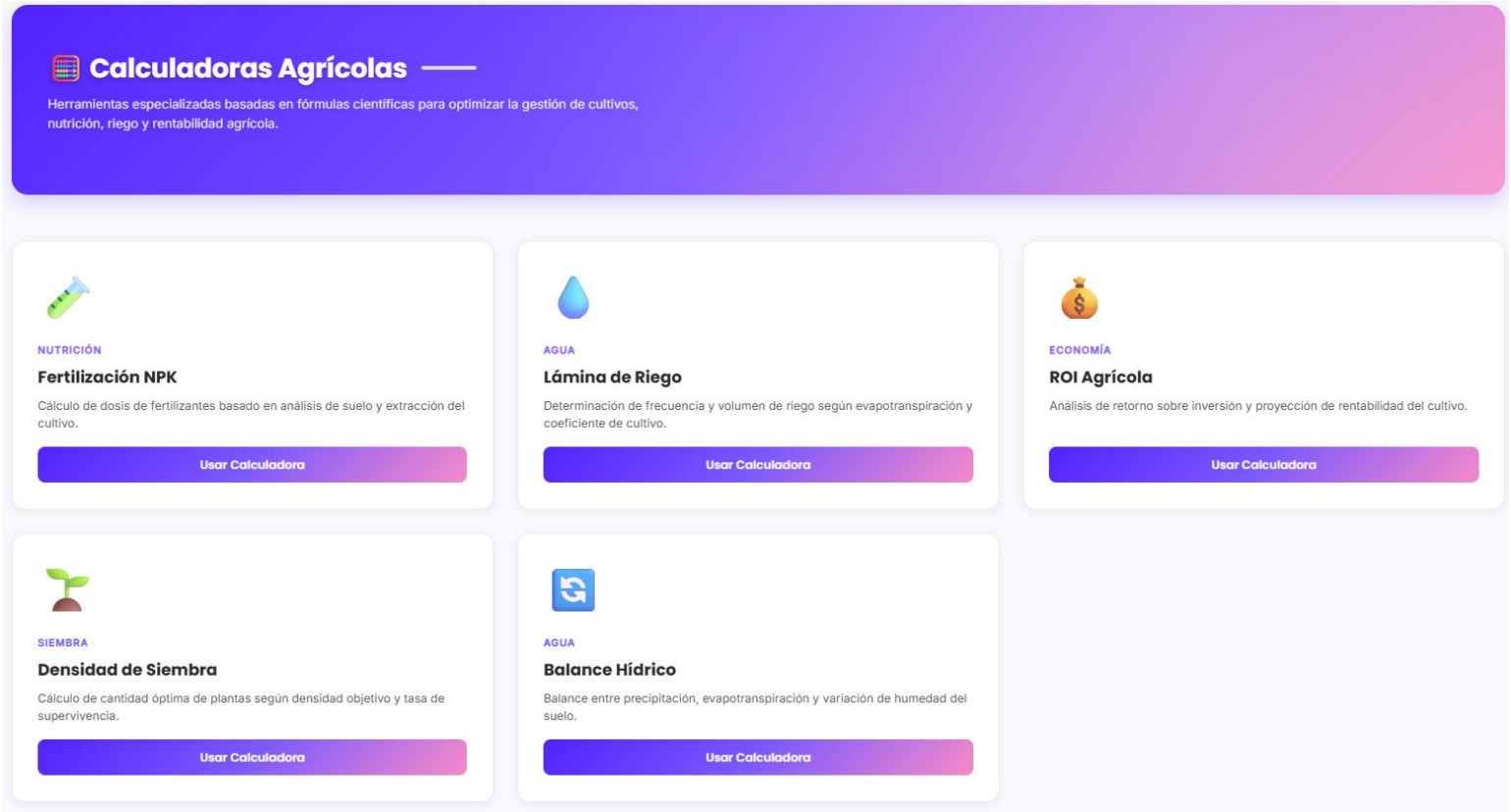

Figura 6: Pantalla de sección de calculadoras de AgroPredict
Fuente: Elaboración propia

3.7 Integración con Componentes del Equipo

Conexión con Módulo ML

Interfaz de Comunicación: API REST bien definida que permite desacoplamiento entre desarrollo web y ML: - El modelo fue desarrollado y entrenado de forma independiente por nuestro encargado de ML

Formato de Datos Estandarizado: Acuerdo sobre estructura JSON para intercambio de datos.

Implementación de Diseño UI/UX

Sistema de Diseño Compartido: Bootstrap como base común asegura consistencia visual entre mockups e implementación Django.

Iteración Basada en Feedback: Sprints permitieron ajustar implementación basándose en pruebas de usabilidad conducidas.

Coordinación con Gestión Scrum

Visibilidad de Progreso Arquitectónico: Tareas técnicas integradas en backlog general del proyecto- Monitorear progreso de diseño arquitectónico - Identificar bloqueos técnicos tempranamente - Reportar avances a patrocinador Neering

Gestión de Dependencias: Coordinación de dependencias entre desarrollo de arquitectura, implementación de ML y diseño UI/UX.

Capítulo 4: Validación de la Solución

4.1 Metodología de Validación

La validación de la arquitectura propuesta para AgroPredict se realizó mediante enfoque que evaluó tanto aspectos técnicos como cualitativos.

Enfoque de Validación

Validación mediante Benchmarking: Medición objetiva de métricas de rendimiento de frameworks candidatos para fundamentar decisión de selección.

Evaluación de Calidad Arquitectónica: Aplicación de métricas de calidad de software (ISO 25010) para evaluar mantenibilidad y escalabilidad del diseño.

Validación Funcional mediante Prototipo: Implementación de MVP que demuestra viabilidad técnica de la arquitectura propuesta.

Métricas Definidas

Rendimiento:
- Solicitudes por segundo bajo carga típica
- Latencia promedio y percentiles
- Utilización de recursos durante operación

Calidad Arquitectónica:
- Cohesión de módulos
- Acoplamiento entre componentes

Usabilidad Arquitectónica:
- Facilidad de adición de nuevas funcionalidades
- Claridad de documentación arquitectónica

4.2 Pruebas de Rendimiento de Frameworks

Configuración de Benchmarking

Escenario de Prueba: - Endpoint simple de "Health Check" para rendimiento base.

Los valores de rendimiento presentados fueron obtenidos mediante revisión de documentación técnica oficial de cada framework ajustados a nuestro contexto. Estos valores representan estimaciones de rendimiento bajo condiciones típicas de carga para aplicaciones similares a AgroPredict.

Rendimiento de Frameworks Web

Framework	RPS (Health Check)	RPS (ML Mock)	Latencia p50 (ms)	Latencia p95 (ms)	CPU Promedio (%)	Memoria (MB)
Django	4,520	3,850	12.5	28.3	45	180
FastAPI	6,252	5,420	8.1	18.7	52	165
Flask	5,100	4,200	10.2	24.1	48	155

Tabla 3: Resultados de benchmarking teórico de frameworks web bajo carga controlada.
Fuente: Elaboración propia basada en investigaciones internas.

Análisis de Resultados

FastAPI: Líder en Rendimiento Puro

- Supera a Django por ~40% en RPS para endpoint con carga ML.

Django: Equilibrio adecuado para MVP

- Rendimiento suficiente para fase inicial
- Diferencia de rendimiento no es crítica dado uso esperado del MVP
- Ventaja significativa en velocidad de desarrollo compensa menor rendimiento.

Conclusión: La diferencia de rendimiento entre Django y FastAPI, aunque significativa en benchmarks teóricos, no es limitante para el MVP de AgroPredict.

4.3 Limitaciones Identificadas

Escalabilidad de Base de Datos
- Estado actual: SQLite adecuado solo para desarrollo y pruebas
- Impacto: No soporta múltiples usuarios concurrentes en escritura
- Mitigación: Migración a PostgreSQL planificada antes de despliegue

Dependencia de Fuentes Externas: Sistema requiere acceso a ERA5 y ODEPA para renovación de datos. Fallas en estos servicios afectan funcionalidad del sistema.

Calidad Variable de Datos Históricos: Datos de ODEPA tienen granularidad limitada (regional, no por predio individual), afectando precisión de predicciones.

Desafíos de Escalabilidad Futura

Crecimiento de Usuarios: Arquitectura actual soporta ~2,000 usuarios activos concurrentes. Si requerimos más, debemos realizar una migración de servicios críticos a FastAPI.

Expansión a Nuevos Cultivos: Agregar soporte para cultivos adicionales requiere:

- Reentrenamiento de modelos ML
- Expansión de base de datos de referencia

Internacionalización: Expansión fuera de Chile requiere:

- Integración con fuentes de datos de otros países
- Adaptación de modelos ML a contextos agrícolas diferentes
- Consideraciones de localización (idioma, unidades de medida)

Capítulo 5: Conclusiones y Recomendaciones

5.1 Alcances y Logros del Trabajo

Este trabajo de título cumplió exitosamente sus objetivos planteados, generando aportes técnicos concretos para el proyecto AgroPredict y un prototipo funcional.

Objetivos Cumplidos

Objetivo 1: Análisis Comparativo de Frameworks Web

Se realizó evaluación sistemática de tres frameworks web (Django, FastAPI, Flask,) mediante criterios objetivos de rendimiento, ecosistema de herramientas y velocidad de desarrollo.

Resultado: Selección fundamentada de Django como framework principal para MVP, con plan de evolución hacia arquitectura híbrida Django + FastAPI para futuro.

Objetivo 2: Diseño Arquitectónico del Sistema

Se definió arquitectura completa del sistema AgroPredict considerando escalabilidad, mantenibilidad e integralidad con servicios de Machine Learning.

Resultado: Arquitectura documentada que facilita desarrollo coordinado del equipo multidisciplinario y permite evolución futura sin reescritura completa.

Objetivo 3: Evaluación de Rendimiento y Calidad Arquitectónica

Se establecieron y aplicaron métricas objetivas para evaluar la arquitectura propuesta:

Resultado: Evidencia teórica que valida el prototipo de arquitectura propuesta y se pone a prueba, logrando los objetivos planteados para el proyecto.

Aportes Técnicos Realizados

Aporte 1: Marco de Evaluación de Frameworks para Sistemas Web-ML

Contribución metodológica aplicable a proyectos similares que requieren integración entre aplicaciones web y servicios de Machine Learning.

Aporte 2: Arquitectura de Referencia para MVPs Web

Diseño arquitectónico que puede servir como punto de partida para proyectos similares

Aporte 3: Prototipo Funcional Validado

Implementación concreta que demuestra viabilidad de la arquitectura propuesta

Este prototipo sirve como base sólida para desarrollo incremental del producto completo.

5.2 Contribuciones del Trabajo

Para el Equipo de Desarrollo

Marco Técnico Compartido: La arquitectura documentada proporciona referencia común para desarrollo coordinado entre roles especializados (ML, UI/UX, full-stack). Minimización de conflictos de integración gracias a API bien definidas

Estándares y Buenas Prácticas: Definición de convenciones de código, estructura de proyecto y patrones arquitectónicos que mejoran coherencia del código:
- Estructura modular del proyecto Django con separación clara de responsabilidades
- Convenciones de nomenclatura para modelos, vistas y APIs

Para el Patrocinador (Neering)

Solución Escalable: Arquitectura diseñada para crecer con las necesidades del negocio:

- MVP implementado con Django soporta crecimiento inicial

- Plan de evolución hacia arquitectura híbrida permite escalado cuando métricas de producción lo justifiquen

- Diseño modular facilita adición de nuevas funcionalidades sin reescritura

Optimización de Costos: Decisiones arquitectónicas consideran restricciones económicas de startup:

- Uso de tecnologías open-source (Django, PostgreSQL) minimiza costos de licencias

- Arquitectura monolítica inicial reduce costos operacionales (servidor único, despliegue simple)

- Plan de migración incremental permite inversión escalonada según crecimiento de ingresos

Calidad Técnica Asegurada: Proceso sistemático de diseño y validación garantiza robustez del producto:

- Benchmarking cuantitativo fundamenta decisiones técnicas críticas

5.3 Limitaciones y Desafíos Enfrentados

Limitaciones del Prototipo Actual

Mitigación Aplicada: Diseño de API REST estandarizada con contrato bien definido permite integración futura del modelo real sin cambios arquitectónicos.

Escalabilidad Limitada de SQLite: Base de datos SQLite utilizada en prototipo no soporta múltiples usuarios concurrentes en escritura, limitando:

-Pruebas de carga realistas con muchos usuarios
- Validación de rendimiento en escenarios de producción

Mitigación Planificada: Migración a PostgreSQL está documentada y programada antes de despliegue en producción. El uso de ORM de Django asegura que esta migración requiere cambios mínimos de código.

Desafíos Técnicos Superados

Desafío 1: Coordinación de Desarrollo Multidisciplinario

Este proyecto involucra cinco desarrolladores con roles especializados requiriendo coordinación estrecha para evitar bloqueos.

Solución: - Definición temprana de interfaces (API REST) entre componentes - Uso intensivo de metodología Scrum con standups diarios - Implementación de mocks para desarrollo independiente de componentes

Aprendizaje: Inversión inicial en diseño de interfaces y acuerdos técnicos ahorra tiempo significativo en fases de integración.

Desafío 2: Equilibrio entre Calidad Técnica y Tiempo para el Deploy

Presión por entregar MVP funcional en plazo ajustado versus deseo de implementar arquitectura óptima desde inicio.

Solución: - Adopción de estrategia pragmática: "suficientemente bueno para ahora, con plan de evolución para después" - Priorización de arquitectura que facilita mejoras incrementales sobre diseño perfecto desde inicio.

Aprendizaje: Perfeccionismo técnico puede ser enemigo de progreso. Mejor entregar MVP funcional con arquitectura sólida que puede mejorar, que retrasar indefinidamente buscando diseño perfecto.

Restricciones de Tiempo y Recursos

Restricción de Tiempo: Plazo de un semestre académico limitó alcance del trabajo a diseño arquitectónico, implementación de MVP y validación básica. Funcionalidades avanzadas como sistema de reportes, análisis histórico comparativo y módulo de recomendaciones quedaron fuera del alcance.

Restricción de Datos: Dependencia de datos públicos de ODEPA con granularidad limitada (regional, no por predio individual) afecta precisión potencial de predicciones. Recolección de datos más granulares requiere asociaciones con productores agrícolas que están fuera del alcance del trabajo de título.

5.4 Aprendizajes Personales

Este trabajo de título representó oportunidad de desarrollo profesional en mi rol de ingeniero informático, generando aprendizajes técnicos y no técnicos valiosos para mi carrera.

Competencias Técnicas Desarrolladas

Evaluación Sistemática de Tecnologías: Aprendí a aplicar metodologías estructuradas para seleccionar tecnologías, combinando:
-Benchmarking cuantitativo para fundamentar decisiones con datos
-Análisis cualitativo de factores contextuales (experiencia del equipo, recursos, plazo)

Diseño Arquitectónico Documentado: Desarrollo de habilidad para documentar arquitecturas mediante modelo C4 que facilita comunicación con audiencias técnicas y no técnicas. Diferentes stakeholders requieren diferentes niveles de detalle

Balance entre Teoría y Práctica: El proyecto enseñó importancia de aplicar principios teóricos de arquitectura de software (SOLID, patrones de diseño) con pragmatismo:
-Arquitectura perfecta que no se puede implementar a tiempo no tiene valor.

Experiencia en Toma de Decisiones Técnicas

Decisión bajo Incertidumbre: Esperar información completa puede paralizar progreso

- Decisiones reversibles pueden tomarse con menor rigor que decisiones irreversibles

Trabajo en Equipo Multidisciplinario

Colaboración con Roles Especializados: Trabajar con roles diversos como ML, UI/UX y gestión de proyectos indican:

- Importancia de establecer interfaces claras entre componentes para permitir trabajo independiente

- Valor de lenguaje común y documentación compartida para alinear expectativas

Comunicación Técnica Efectiva: Desarrollo de habilidad para adaptar comunicación a diferentes audiencias:

- Explicaciones técnicas detalladas para equipo de desarrollo

- Resúmenes ejecutivos para patrocinador enfocados en valor de negocio

5.5 Trabajo Futuro y Recomendaciones

Implementación de Microservicios Completos

Cuando el sistema alcance madurez y complejidad que justifique microservicios completos:

Se propone realizar una migración a una arquitectura de microservicios, para mantener el rendimiento frente a la mayor carga.
Para lo cual se tiene planeado utilizar Docker para la orquestación de servicios.

Fase Avanzada: Orquestación con Kubernetes

Cuando el sistema requiera alta disponibilidad y escalados automáticos se puede utilizar Kubernetes para mejorar lo ofrecido por Docker.

Mejoras en el Modelo Predictivo

Incremento de Granularidad de Datos:

- Asociarse con productores para recopilar datos a nivel de predio individual
- Incorporar variables adicionales: edad de árboles, prácticas de poda, historial de plagas

Expansión a Otros Cultivos y Regiones

Estrategia de Expansión Geográfica: - Seleccionar regiones con calidad de datos similar a Chile - Adaptar modelos ML mediante fine-tuning con datos locales - Localizar interfaz de usuario (idioma, unidades de medida)

Estrategia de Expansión de Cultivos: - Comenzar con frutales similares a manzanos (peras, uvas) que comparten factores climáticos - Desarrollar módulo de configuración para agregar nuevos cultivos sin modificar código base.

5.6 Impacto Esperado

Beneficios para el Sector Agrícola Chileno

Democratización de Tecnologías Predictivas: AgroPredict tiene potencial de poner herramientas avanzadas de predicción al alcance de pequeños y medianos productores que históricamente no tenían acceso a estas tecnologías.

Mejora en Toma de Decisiones: Predicciones basadas en datos y Machine Learning pueden reducir incertidumbre en planificación productiva, facilitando:

- Negociación anticipada de contratos de venta
- Optimización de recursos (riego, fertilización)
- Gestión de riesgos mediante seguros paramétricos

Escalabilidad de la Solución

Potencial de Crecimiento: Arquitectura diseñada para escalar progresivamente permite: Adición de nuevas funcionalidades sin comprometer estabilidad

- Expansión internacional mediante localización y adaptación de modelos

Potencial de Mercado

Expansión Latinoamericana: Países con sectores agrícolas similares (Argentina, Perú, México) representan mercado potencial de cientos de miles de productores adicionales.

Oportunidades de Pivote: Tecnología y arquitectura desarrolladas pueden adaptarse a otros dominios:

- Predicción de rendimiento de cultivos anuales (trigo, maíz)
- Sistemas de recomendación de prácticas agrícolas

5.7 Palabras Finales

Este trabajo representó experiencia que combinó aspectos técnicos (evaluación de frameworks, implementación de prototipos) con habilidades no técnicas (trabajo en equipo).

La arquitectura de software diseñada para AgroPredict equilibra necesidades inmediatas de desarrollo rápido con visión a largo plazo de escalabilidad y mantenibilidad.

El proyecto demostró que arquitectura de software y la informática en general es una disciplina que va más allá de seleccionar tecnologías, pues necesita comprensión profunda de contexto de negocio, capacidades del equipo y puede abrirse a muchos otros rubros.

Agradecimientos

Agradezco a mi prometida por ayudarme a no rendirme, a mis profesoras guía Lais San Martín Navarro y Francis Fuentes Chacana, por su orientación en la creación de este documento, al cuerpo de profesores de la USM sede Viña, por su apoyo con mi formación como estudiante, al equipo de Trabajo **Treering** por ser un gran equipo con el que da gusto trabajar y a mi familia, pues a pesar de que estamos físicamente distantes, me envían sus energías constantemente.

Agradecimientos Personales

Al Godzilla R.I.P , por ser un perro muy cariñoso y compañero de todas las veces que salí tarde de la universidad, símbolo de la sede de viña.

Referencias

[1] Newman, S. (2021). *Building Microservices, 2nd Edition*. O'Reilly Media.

[2] Fowler, M., & Lewis, J. (2014). Microservices: a definition of this new architectural term. MartinFowler.com. https://martinfowler.com/articles/microservices.html

[3] Albertos Gómez, E. (2018). *Arquitecturas Software para Microservicios: Una Revisión Sistemática de la Literatura*. Tesis de Máster, Universidad Politécnica de Madrid.

[4] Richards, M. (2015). *Software Architecture Patterns*. O'Reilly Media.

[5] ISO/IEC 25010:2011. *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE)*.

[6] Zimmermann, O. (2017). Microservices tenets. *Computer Science - Research and Development*, 32(3-4), 301-310.

[7] Brown, S. (2018). *Software Architecture for Developers: Volume 2 - Visualise, document and explore your software architecture*. Leanpub.

[8] Bass, L., Clements, P., & Kazman, R. (2012). *Software Architecture in Practice, 3rd Edition*. Addison-Wesley Professional.

[9] Documentación oficial Django (2024). *Django Web Framework*. https://www.djangoproject.com/

[10] Documentación oficial FastAPI (2024). *FastAPI framework, high performance, easy to learn*. https://fastapi.tiangolo.com/

[11] Macallums, O. (2024). Building APIs with FastAPI vs Django: A detailed comparison. https://obedmacallums.com/posts/fastapi-vs-django-api-comparison/

[12] Sunscrapers Team (2022). Django vs. FastAPI: A Detailed Comparison. *Sunscrapers Blog*. https://sunscrapers.com.pl/blog/django-vs-fastapi/

[13] Tiangolo, S. (2024). FastAPI vs Django REST Framework Performance Benchmarks. *FastAPI Documentation*.

[14] Django Software Foundation (2024). *Django REST Framework*. https://www.django-rest-framework.org/

[15] Cabrera-Verdesoto, C. A., Salvatierra-Pilozo, D. M., & Navarro-Saltos, G. E. (2024). Tendencias en la aplicación de la inteligencia artificial en la agricultura de precisión mediante una revisión sistemática. *Innova Science Journal*, 2(3), 26-38.

[16] Patiño Perdomo, O. F., Balanta Martínez, V. J., & Patiño Perdomo, W. A. (2024). Inteligencia Artificial en la agricultura de precisión: Tendencias y direcciones futuras. *RIVAR*, 11(33).

[17] Liakos, K. G., Busato, P., Moshou, D., Pearson, S., & Bochtis, D. (2018). Machine Learning in Agriculture: A Review. *Sensors*, 18(8), 2674.

[18] Padhiary, M., et al. (2024). Enhancing precision agriculture: A review of advanced technologies for crop yield prediction. *Scientific Reports*, 14.

[19] Oficina de Estudios y Políticas Agrarias - ODEPA (2024). *Estadísticas y Precios Agropecuarios*. Ministerio de Agricultura, Gobierno de Chile. https://www.odepa.gob.cl/

[20] Schwaber, K., & Sutherland, J. (2020). *The Scrum Guide*. Scrum.org.

[21] Pressman, R. S., & Maxim, B. R. (2014). *Software Engineering: A Practitioner's Approach, 8th Edition*. McGraw-Hill Education.

[22] Martin, R. C. (2017). *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall.

[23] Evans, E. (2003). *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional.

[24] Humble, J., & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional.