

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE



“EVALUACIÓN DE ARQUITECTURA DE SOFTWARE PARA BIOPLANNER ”

CLAUDIO BENJAMÍN FRANCESCO VARELA CHAPA

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Pablo Cruz
Profesor Correferente: Por asignar

Agosto - 2026



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo: Evaluación de Arquitectura de Software para BioPlanner

Nombre del candidato(a): Claudio Benjamín Francesco Varela Chapa

Carrera / Grado: Ingeniería Civil en Informática

Campus: Casa Central **Departamento:** Informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Pablo Cruz Navea, 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)

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

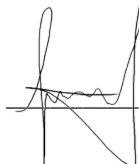
☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por (**marcar una opción**):

☐ 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: 5/8/2026 **Firma:** 

Estudiante o Candidato(a):

Fecha: 5 / 8 / 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.

DEDICATORIA

Esta memoria marca la culminación de cinco años y medio de estudio, cinco años y medio que pasaron volando, me enfrente a distintas situaciones tanto académicas como personales y estoy muy orgulloso de haberlas superado. Las personas que estuvieron junto a mi en este proceso me marcaron profundamente, mas que cualquier conocimiento que pude haber obtenido cursando la malla académica, es por esto que quiero demostrarles mi gratitud mencionándolos en este trabajo. Quiero agradecer a:

Geomara del Carmen Chapa Santis
Geomara Camila Varela Chapa
Camilo Alejandro Varela Chapa
Natalia Florentina Varela Chapa
Valentina Antonia Zamora Cortez
Juan Camilo Varela Chávez
Cristopher Jesús Mora Aravena
Raúl Astudillo

A todos mis primos y tías.
A todos mis amigos y amigas.

RESUMEN

Resumen— BioPlanner, desarrollado en un contexto académico bajo restricciones de tiempo y recursos, priorizó la implementación de funcionalidades por sobre la evaluación sistemática de sus atributos de calidad, por lo que no existía evidencia objetiva sobre si su arquitectura estaba preparada para operar en un entorno de producción real. Esta incertidumbre representa un riesgo directo para los objetivos de negocio del producto, al comprometer su capacidad de responder a escenarios de alta demanda, facilitar su mantenimiento y evolucionar de forma segura. Para abordarlo, se aplicó la extensión SAEMAT del Kernel de SEMAT, identificando los atributos de calidad priorizados (Seguridad, Escalabilidad y Mantenibilidad) y reconstruyendo cinco ADR a partir del conocimiento tácito del equipo, los cuales fueron evaluados mediante un procedimiento inspirado en el método DCAR. La evaluación reveló debilidades concretas en Seguridad, destacando la falta de autenticación entre el backend y el microservicio de recomendaciones y la aplicación inconsistente del mecanismo de autenticación en el backend, este último calificado como crítico por exponer datos de usuarios sin control de acceso; a partir de esto se formularon propuestas de mejora y un plan de implementación priorizado según severidad. Como trabajo futuro, se recomienda reevaluar la arquitectura tras incorporar las mejoras, extender el análisis a otros atributos de calidad y evaluar el cumplimiento de BioPlanner frente a la Ley 21.719 sobre protección de datos personales, vigente a partir de diciembre de 2026.

Palabras Clave— Arquitectura de software; evaluación arquitectónica; feria de software 33^a; SEMAT/SAEMAT.

ABSTRACT

Abstract— BioPlanner, developed in an academic context under time and resource constraints, prioritized the implementation of functionalities over the systematic evaluation of its quality attributes, meaning there was no objective evidence as to whether its architecture was ready to operate in a real production environment. This uncertainty represents a direct risk to the product’s business objectives, as it compromises its ability to respond to high-demand scenarios, facilitate maintenance, and evolve safely. To address this, the SAEMAT extension of the SEMAT Kernel was applied, identifying the prioritized quality attributes (Security, Scalability, and Maintainability) and reconstructing five ADRs from the development team’s tacit knowledge, which were then evaluated using a procedure inspired by the DCAR method. The evaluation revealed concrete weaknesses in Security, highlighting the lack of authentication between the backend and the recommendation microservice, and the inconsistent application of the authentication mechanism in the backend, the latter rated as critical for exposing user data without access control; based on this, improvement proposals and an implementation plan prioritized by severity were formulated. As future work, it is recommended to re-evaluate the architecture once the improvements have been incorporated, extend the analysis to other quality attributes, and assess BioPlanner’s compliance with Law 21.719 on personal data protection, which will take full effect in December 2026.

Keywords— Software architecture; architectural evaluation; 33rd software fair; SEMAT/SAEMAT.

GLOSARIO

ADR: Architecture Decision Record (Decisión Arquitectónica Documentada).
ARID: Active Reviews for Intermediate Designs.
ATAM: Architecture Trade-off Analysis Method.
CEV: Calificación Energética de Viviendas.
CORFO: Corporación de Fomento de la Producción.
CRUD: Create, Read, Update, Delete.
DCAR: Decision-Centric Architecture Review.
DI: Departamento de Informática.
EE: Eficiencia Energética.
FURPS: Functionality, Usability, Reliability, Performance, Supportability.
ISO/IEC: International Organization for Standardization / International Electrotechnical Commission.
JWT: JSON Web Token.
NoSQL: Not only SQL.
ODM: Object Document Mapper.
ODS: Objetivos de Desarrollo Sostenible.
OGUC: Ordenanza General de Urbanismo y Construcciones.
ONU: Organización de las Naciones Unidas.
REST: Representational State Transfer.
SAAM: Software Architecture Analysis Method.
SAEMAT: Software Architecture Evaluation MEthod and Theory.
SARA: Software Architecture Review and Assessment.
SEMAT: Software Engineering Method and Theory.
SMTP: Simple Mail Transfer Protocol.
SQuaRE: Systems and software Quality Requirements and Evaluation.
UTFSM: Universidad Técnica Federico Santa María.

ÍNDICE DE CONTENIDOS

RESUMEN	III
ABSTRACT	IV
GLOSARIO	V
ÍNDICE DE FIGURAS	VIII
ÍNDICE DE TABLAS	VIII
INTRODUCCIÓN	1
CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA	2
1.1 Objetivos	3
1.1.1 Objetivo general	3
1.1.2 Objetivos específicos	3
CAPÍTULO 2: MARCO CONCEPTUAL Y ESTADO DEL ARTE	4
2.1 Descripción de Bio Planner	4
2.2 Eficiencia Energética y Escala de Calificación	4
2.3 Calidad del Software	5
2.4 ¿Qué es la arquitectura de software?	6
2.5 Métodos para evaluar arquitecturas de software	7
2.5.1 Métodos basados en escenarios	7
2.5.2 Métodos orientados al análisis de <i>trade-offs</i>	7
2.5.3 Métodos basados en decisiones arquitectónicas documentadas	7
2.5.4 Otros métodos de evaluación	8
2.6 SEMAT y Essence Kernel	8
2.6.1 Extensión del Kernel de SEMAT: SAEMAT	8
2.7 Restricciones legales	9
CAPÍTULO 3: PROPUESTA DE SOLUCIÓN	10
3.1 Identificación, <i>trade-offs</i> y priorización de requisitos no funcionales	10
3.1.1 Mantenibilidad	10
3.1.2 Seguridad	11
3.1.3 Flexibilidad (Escalabilidad)	11
3.1.4 Atributos de calidad priorizados	11
3.2 Objetivos del negocio	12
3.2.1 Integrarse en el mercado de reducción de la intensidad energética en Chile	12
3.3 Descripción de la arquitectura	12
3.3.1 Descripción de alto nivel de BioPlanner	16
3.3.2 Descripción de la Capa Frontend de BioPlanner	16

3.3.3	Descripción de la Capa Backend de BioPlanner	17
3.4	Descripción de la Capa de servicios externos de BioPlanner	18
3.5	Decisiones de arquitectura	19
3.6	Evaluación de la Arquitectura Centrada en Decisiones	24
3.7	Propuestas de mejoras en base a la evaluación inspirada en DCAR	28
3.7.1	Propuesta para ADR-003: Implementar <i>Athenticate Actors</i> entre back-end y servicios externos.	28
3.7.2	Propuesta para ADR-004: Externalizar parámetros de calculo a la per- sistencia	28
3.7.3	Propuesta para ADR-005: Centralizar la autenticación mediante guard global	29
CAPÍTULO 4: PLAN DE IMPLEMENTACIÓN Y FACTIBILIDAD		30
CAPÍTULO 5: CONCLUSIONES		34
REFERENCIAS BIBLIOGRÁFICAS		36

ÍNDICE DE FIGURAS

1	Vista de alto nivel de la arquitectura de BioPlanner.	13
2	Diagrama de componentes — Capa Frontend.	14
3	Diagrama de componentes — Capa Backend.	15
4	Diagrama de integración con servicios externos.	15

ÍNDICE DE TABLAS

1	Modelo de calidad del producto ISO/IEC 25010:2023 [ISO/IEC, 2023].	6
2	ADR-001: Arquitectura modular.	20
3	ADR-002: Persistencia NoSQL.	21
4	ADR-003: Microservicio externo para el motor de recomendaciones sin autenticación.	22
5	ADR-004: Parámetros normativos embebidos en el servicio de cálculo.	23
6	ADR-005: Autenticación no centralizada a nivel de backend.	24
7	Esquema de semáforo DCAR. Fuente: adaptado de [van Heesch <i>et al.</i> , 2014].	25
8	Plan de implementación para la mejora de ADR-003. Fuente: Elaboración propia.	31
9	Plan de implementación para la mejora de ADR-004. Fuente: Elaboración propia.	32
10	Plan de implementación para la mejora de ADR-005. Fuente: Elaboración propia.	33

INTRODUCCIÓN

La eficiencia energética en el sector de la construcción se ha convertido en un eje estratégico tanto a nivel nacional como internacional, en línea con los Objetivos de Desarrollo Sostenible de la ONU [Organización de las Naciones Unidas,] relacionados con ciudades sostenibles y acción por el clima. En Chile, esta tendencia se ha visto reforzada por la entrada en vigencia de una nueva reglamentación térmica para viviendas [Ministerio de Vivienda y Urbanismo, 2024], que exige a arquitectos y constructoras incorporar criterios de desempeño energético desde las etapas más tempranas del diseño.

En este contexto surge BioPlanner, una aplicación móvil orientada a arquitectos y constructores que permite estimar la demanda energética de un proyecto habitacional a partir de variables como su ubicación geográfica y la configuración de materiales utilizados. El producto fue desarrollado por un equipo de seis estudiantes de Ingeniería Civil Informática para la 33.^a Feria de Software de la Universidad Técnica Federico Santa María, e impulsado por su mandante, Danae Esper, arquitecta y clienta del proyecto, con financiamiento de CORFO para continuar su desarrollo.

Si bien BioPlanner demostró ser una propuesta de alto valor en un contexto académico, su tránsito hacia un entorno de producción —entendido como aquel en que usuarios reales interactúan con el sistema bajo condiciones operativas reales— plantea un desafío de ingeniería no trivial. Las restricciones de tiempo y recursos propias del desarrollo en una feria de software llevaron a que las decisiones arquitectónicas priorizaran la implementación de funcionalidades por sobre la evaluación sistemática de los atributos de calidad del sistema. Como consecuencia, no existía evidencia objetiva respecto de si la arquitectura actual era apropiado para adecuadamente a escenarios de alta demanda, facilitar tareas de mantenimiento o evolucionar de forma segura, lo cual representa un riesgo directo para los objetivos de negocio del producto.

Frente a este escenario, el presente trabajo propone realizar una evaluación de la arquitectura de Software de BioPlanner que permita identificar riesgos, analizar las decisiones de diseño adoptadas durante su desarrollo y proponer mejoras concretas orientadas a preparar el sistema para producción. Para ello, se adopta como marco metodológico la extensión SAEMAT del Kernel de SEMAT [Cruz *et al.*, 2023], la cual organiza el proceso de evaluación en torno a la identificación de atributos de calidad priorizados, objetivos del negocio, la descripción de la arquitectura del sistema y el análisis de las decisiones arquitectónicas documentadas. Sobre esta última dimensión se aplica un procedimiento inspirado en el método DCAR (Decision-Centric Architecture Review) [van Heesch *et al.*, 2014], que permite revisar de forma sistemática las decisiones de diseño adoptadas, identificar los riesgos asociados a cada una y formular propuestas de mejora fundamentadas en patrones y tácticas de diseño reconocidas.

CAPÍTULO 1

DEFINICIÓN DEL PROBLEMA

BioPlanner es un producto informático desarrollado para la 33.^a **Feria de Software** de la Universidad Técnica Federico Santa María, evento organizado por el Departamento de Informática.

El proyecto consiste en una aplicación móvil diseñada para arquitectos y constructores, la cual permite obtener una estimación de la demanda energética que requerirá una vivienda en función de variables clave, como su ubicación geográfica y la configuración de materiales utilizados.

Este proyecto destacó como una propuesta de alto valor en la feria gracias a su impacto positivo y su alineación estratégica con dos de los **Objetivos de Desarrollo Sostenible (ODS)** de la ONU [Organización de las Naciones Unidas,]: el compromiso con las ciudades sostenibles y la acción por el clima. Asimismo, la iniciativa se concibió incorporando de manera proactiva la nueva reglamentación térmica de las viviendas [Ministerio de Vivienda y Urbanismo, 2024], anticipándose incluso a su entrada en vigencia el 28 de noviembre de 2025.

El producto fue desarrollado por seis estudiantes de Ingeniería Civil Informática e impulsado por **Danae Esper**, creadora del proyecto, el cual obtuvo financiamiento por parte de CORFO para completar su desarrollo.

El problema se origina en la necesidad de asegurar que la arquitectura de software de la aplicación mantenga niveles adecuados de calidad frente a distintos escenarios operativos, como tareas de mantenimiento, incrementos abruptos de carga o la evolución natural del producto, con el objetivo de preparar el sistema para un entorno de producción, entendido como aquel en el que usuarios finales interactúan con la aplicación bajo condiciones reales de operación.

Debido a que el desarrollo se realizó bajo restricciones significativas de tiempo y recursos, las decisiones arquitectónicas priorizaron la implementación de funcionalidades por sobre la evaluación sistemática de los atributos de calidad. Como consecuencia, no existe evidencia objetiva que permita determinar si la arquitectura actual satisface los requisitos necesarios para operar adecuadamente en un entorno de producción, generando incertidumbre respecto a su capacidad para responder frente a escenarios de alta demanda, facilitar su mantenimiento o evolucionar de manera segura. Esta incertidumbre puede afectar negativamente objetivos del negocio, tales como ofrecer un servicio confiable y disponible para una gran base de usuarios.

Se trata de un problema **complejo de ingeniería**, ya que se busca llevar un prototipo funcional validado en un contexto académico hacia un sistema preparado para operar en condiciones reales. Para ello, es necesario realizar un análisis profundo del sistema, comprendiendo

de forma integral los requisitos del cliente y del negocio.

Este proceso requiere considerar los *trade-offs* asociados a cada decisión arquitectónica y lograr el mejor balance posible entre los distintos atributos de calidad. Al incorporar nuevas decisiones, resulta fundamental supervisar la adaptabilidad del sistema frente a los constantes cambios tecnológicos, los cuales pueden representar tanto oportunidades como amenazas. En este contexto, una evaluación arquitectónica permite identificar riesgos de forma temprana, fundamentar decisiones de mejora y aumentar la probabilidad de que la arquitectura satisfaga los objetivos técnicos y de negocio definidos para el producto.

En consecuencia, el problema no posee una única solución correcta, sino que demanda equilibrar simultáneamente diversas dimensiones técnicas y organizacionales, apoyando las decisiones en métricas objetivas y en una comprensión sistémica del entorno tecnológico. Por ello, resulta necesario realizar una evaluación sistemática de la arquitectura existente que permita identificar riesgos, analizar las decisiones arquitectónicas adoptadas y proponer mejoras orientadas a incrementar su preparación para un entorno de producción.

1.1. Objetivos

1.1.1. Objetivo general

Preparar a **BioPlanner** para su despliegue en un ambiente de producción, mediante la evaluación y mejora de su arquitectura de software, aplicando marcos teóricos reconocidos internacionalmente.

1.1.2. Objetivos específicos

- **Evaluar** la arquitectura actual de BioPlanner utilizando marcos teóricos reconocidos, identificando riesgos y oportunidades de mejora en los atributos de calidad importantes para el cliente.
- **Diseñar** una propuesta de mejora arquitectónica basada en patrones y tácticas de diseño que fortalezcan los atributos de calidad identificados.
- **Elaborar** un plan de implementación y factibilidad que permita, eventualmente, llevar las mejoras propuestas a un entorno de producción.

CAPÍTULO 2

MARCO CONCEPTUAL Y ESTADO DEL ARTE

2.1. Descripción de Bio Planner

Bio Planner es una aplicación móvil que permite estimar el rendimiento energético de un proyecto habitacional desde la idea inicial. Su propuesta central es:

“Transformar viviendas en hogares eficientes mediante decisiones tempranas basadas en simulación, datos territoriales y tecnologías digitales avanzadas.”

Para ello, la aplicación ofrece las siguientes funcionalidades:

- **Simulación energética automática** basada en parámetros del proyecto.
- **Datos climáticos y geoespaciales**, esenciales para ajustar el diseño a la zona donde se emplazará la vivienda.
- **Acceso a proveedores de materiales eficientes**, conectando la demanda (arquitectos, constructoras) con la oferta real de aislantes, ventanas eficientes y sistemas constructivos avanzados.

Esta combinación convierte a Bio Planner en una herramienta no solo técnica, sino también colaborativa, capaz de integrar actores de toda la cadena de valor de la construcción.

2.2. Eficiencia Energética y Escala de Calificación

La *Eficiencia Energética* (EE) se define como el uso óptimo de la energía para mantener condiciones de confort térmico. En el contexto nacional, el desempeño se cuantifica mediante el porcentaje de ahorro de energía respecto a una “vivienda base”, entendida como aquella edificación que cumple estrictamente con las exigencias mínimas de la Ordenanza General de Urbanismo y Construcciones (OGUC).

De acuerdo con el *Manual de Procedimientos de la CEV* del Ministerio de Vivienda y Urbanismo [Ministerio de Vivienda y Urbanismo, 2025], el significado de la escala de calificación es el siguiente:

- **Eficiencia Superior (A y B):** Viviendas con ahorros superiores al **70 %**, logrados mediante diseño bioclimático y alta hermeticidad.

- **Eficiencia Media (C y D):** Ahorros entre el **40 % y 70 %**, representando un estándar destacado que supera la normativa actual.
- **Estándar Normativo y Precario (E, F y G):** La letra **E** representa el cumplimiento mínimo legal (ahorro 0–40 %), mientras que las letras **F y G** indican un desempeño deficiente e inferior a la normativa vigente.

2.3. Calidad del Software

La calidad del software se define como el grado en que un sistema satisface las necesidades declaradas e implícitas de sus interesados —también conocidas como requisitos no funcionales—, aportando valor de forma consistente. A lo largo de la historia de la disciplina, diversos autores e instituciones han propuesto modelos para estructurar y evaluar este concepto:

- El modelo **McCall** [McCall *et al.*, 1977] presenta dos niveles de atributos de calidad: factores de calidad y criterios de calidad. Entre los factores se encuentran once características que se mantienen vigentes en modelos modernos, tales como la **mantenibilidad** y la **usabilidad**.
- El modelo de **Boehm** [Boehm, 1978] es similar al de McCall, pero incorpora una variedad más amplia de características distribuidas en tres niveles de abstracción (alto, medio y básico). Su uso principal es la evaluación de la utilidad, la mantenibilidad y la portabilidad del software.
- **FURPS** [Grady y Caswell, 1987] organiza la calidad del software en cinco dimensiones: funcionalidad, usabilidad, fiabilidad, rendimiento y soporte (*Functionality, Usability, Reliability, Performance, Supportability*).
- El modelo de **Dromey** [Dromey, 1995], al igual que el de Boehm, establece una conexión entre características de bajo nivel —como la estructura del código— y atributos de alto nivel, como la mantenibilidad.
- **ISO/IEC 9126:2001** [ISO/IEC, 2001] fue el primer estándar internacional de calidad de software. Incorporó criterios de modelos anteriores, como el de McCall, consolidando un marco de referencia común para la industria.
- **ISO/IEC 25010:2023** [ISO/IEC, 2023] es la evolución del estándar anterior y el modelo más ampliamente aceptado por la industria en la actualidad.

Para efectos de esta memoria, se adopta el estándar internacional **ISO/IEC :2023**, el cual descompone la calidad del producto en ocho características fundamentales [ISO/IEC, 2023]:

Característica	Descripción
Adecuación Funcional	Grado en que las funciones del sistema satisfacen las necesidades declaradas e implícitas del usuario.
Eficiencia del Desempeño	Desempeño relativo a la cantidad de recursos utilizados, incluyendo tiempo de respuesta y capacidad de procesamiento.
Compatibilidad	Capacidad del sistema para intercambiar información con otros sistemas y operar en entornos compartidos.
Usabilidad	Facilidad de aprendizaje, operabilidad y satisfacción del usuario final al interactuar con el sistema.
Fiabilidad	Capacidad del sistema para mantener su nivel de rendimiento bajo condiciones establecidas, incluyendo disponibilidad y tolerancia a fallos.
Seguridad	Protección de la información y los datos, garantizando confidencialidad, integridad y autenticidad.
Mantenibilidad	Facilidad con la que el software puede ser modificado, analizado o probado para adaptarse a nuevos requisitos o corregir defectos.
Portabilidad	Capacidad del software para ser transferido de un entorno de ejecución a otro con el mínimo esfuerzo.

Tabla 1: Modelo de calidad del producto ISO/IEC 25010:2023 [ISO/IEC, 2023].

2.4. ¿Qué es la arquitectura de software?

La arquitectura de software corresponde a la estructura —o conjunto de estructuras— que conforman un sistema, incluyendo sus componentes, las relaciones entre ellos y las propiedades que son externamente visibles [Clements *et al.*, 2002]. Como nivel de abstracción, la arquitectura omite los detalles internos de implementación y se centra en aquellos elementos relevantes para comprender el comportamiento y la organización general del sistema.

Además de describir la estructura del software, la arquitectura constituye un mecanismo de comunicación entre los distintos *stakeholders*, facilitando el entendimiento compartido del sistema. Asimismo, representa las decisiones de diseño más tempranas y significativas, las cuales condicionan la evolución futura del producto. Estas decisiones pueden ser reutilizadas y transferidas a otros proyectos, convirtiendo a la arquitectura en un activo estratégico para

el desarrollo de software.

2.5. Métodos para evaluar arquitecturas de software

Los métodos de evaluación arquitectónica pueden clasificarse según el enfoque utilizado para analizar una arquitectura. Entre los estilos más comunes se encuentran los métodos basados en escenarios, los orientados al análisis de *trade-offs* entre atributos de calidad y aquellos centrados en la evaluación de decisiones arquitectónicas documentadas. A continuación se presentan algunos de los métodos más representativos de cada categoría.

2.5.1. Métodos basados en escenarios

Estos métodos utilizan escenarios que representan situaciones reales o hipotéticas para analizar cómo la arquitectura responde ante determinados cambios, requisitos o condiciones operacionales.

- **SAAM (Software Architecture Analysis Method) [Kazman *et al.*, 1994]:** Metodología centrada en evaluar la modificabilidad del sistema, analizando el impacto de escenarios de cambio sobre los componentes arquitectónicos.
- **ARID (Active Reviews for Intermediate Designs) [Clements *et al.*, 2000]:** Enfocado en la evaluación de diseños intermedios, verifica si las interfaces y componentes son aptos para ser utilizados por los desarrolladores que implementarán el sistema.

2.5.2. Métodos orientados al análisis de *trade-offs*

Estos métodos buscan identificar y analizar los compromisos existentes entre distintos atributos de calidad, permitiendo comprender cómo una decisión arquitectónica puede beneficiar un atributo mientras afecta negativamente a otro.

- **ATAM (Architecture Trade-off Analysis Method) [Clements *et al.*, 2002]:** Método de nueve pasos que identifica riesgos y *trade-offs* entre atributos de calidad, aplicable en cualquier fase del ciclo de vida del software.

2.5.3. Métodos basados en decisiones arquitectónicas documentadas

Este enfoque evalúa la arquitectura a partir de las decisiones registradas durante su diseño y evolución, considerando sus motivaciones, restricciones, consecuencias e impacto sobre los

atributos de calidad del sistema.

- **DCAR (Decision-Centric Architecture Review)** [van Heesch *et al.*, 2014]: Método de evaluación arquitectónica centrado en los argumentos de las decisiones de diseño documentadas. Su objetivo es analizar la racionalidad, trazabilidad y consecuencias de las decisiones arquitectónicas, identificando riesgos, dependencias y posibles impactos sobre los atributos de calidad del sistema a lo largo de su evolución.

2.5.4. Otros métodos de evaluación

Existen además enfoques híbridos que combinan múltiples perspectivas de análisis e incorporan factores técnicos, organizacionales y de negocio dentro del proceso de evaluación.

- **SARA (Software Architecture Review and Assessment)** [SARA Working Group, 2002]: Marco de evaluación que integra el análisis técnico con el contexto organizacional, proporcionando una valoración holística de la arquitectura frente a sus metas estratégicas.

2.6. SEMAT y Essence Kernel

SEMAT (Software Engineering Method and Theory) [Jacobson *et al.*, 2013] es una iniciativa que tiene como objetivo fortalecer la disciplina de la ingeniería de software mediante una base teórica sólida, principios comprobados y prácticas ampliamente aceptadas. Uno de los principales resultados de SEMAT es **Essence**, un marco conceptual que define un *kernel* compuesto por los elementos fundamentales presentes en todo esfuerzo de desarrollo de software. Este kernel proporciona un lenguaje común para describir métodos, prácticas y proyectos. Los elementos fundamentales se organizan en **Alphas** (*Abstract-Level Progress Health Attribute*), definidos como atributos de salud y progreso a nivel abstracto que representan los aspectos clave de un proyecto de software, sirviendo como indicadores universales del estado general y el avance del proyecto.

Es importante destacar que el Kernel de SEMAT es **operacionalmente aplicable, extensible y práctico**.

2.6.1. Extensión del Kernel de SEMAT: SAEMAT

SAEMAT (Software Architecture Evaluation M^Ethod and Theory) es una extensión del Kernel de SEMAT [Cruz *et al.*, 2023] diseñada específicamente para proveer métodos y técnicas

orientados a la evaluación de la arquitectura de un sistema. Su contenido se organiza en torno a cinco dimensiones centrales denominadas *Sub-Alphas*, las cuales pueden estar vinculadas entre sí y presentan una progresión de estados definida.

- **Quality Attributes:** Define y desglosa las características de rendimiento, seguridad o mantenibilidad que debe cumplir el sistema.
- **Business Goals:** Identifica las metas estratégicas y de misión de la organización que guían el diseño técnico.
- **Architecture Description:** Estructura la representación y documentación de la arquitectura del sistema.
- **Architecture Decision:** Registra y evalúa las decisiones clave de diseño junto con sus justificaciones.
- **Evaluation Adoption:** Define el método que será utilizado para llevar a cabo la evaluación arquitectónica.

2.7. Restricciones legales

Además de las restricciones normativas descritas en la sección 2.2, relativas a la eficiencia energética y la reglamentación térmica, el sistema debe cumplir con la **Ley 19.628** [Ministerio Secretaría General de la Presidencia, 1999] sobre protección de la vida privada. Esta legislación establece que los datos personales de los usuarios deben ser tratados mediante medidas de seguridad adecuadas para prevenir accesos no autorizados, pérdida, alteración o divulgación indebida de la información.

En términos arquitectónicos, esta ley exige la implementación de mecanismos de autenticación y autorización, cifrado de datos en tránsito y en almacenamiento, gestión segura de credenciales, registro de eventos para auditoría y trazabilidad, así como controles que permitan garantizar la confidencialidad, integridad y disponibilidad de la información tratada por el sistema.

CAPÍTULO 3

PROPUESTA DE SOLUCIÓN

Para formular la solución se trabajará con la extensión del Kernel de SEMAT (Ver 2.6.1), siguiendo las progresiones establecidas específicamente para los *Alphas* descritos. A continuación, se identifican los atributos de calidad más críticos para el cliente, se analizan sus *trade-offs* asociados y, finalmente, se establece su prioridad para el sistema.

3.1. Identificación, *trade-offs* y priorización de requisitos no funcionales

La identificación de los atributos de calidad considerados en este trabajo se realizó a partir de la información obtenida durante las reuniones de levantamiento de requisitos y seguimiento con el cliente, complementada con el análisis del documento de negocio desarrollado para BioPlanner, el cual describe los objetivos del producto, los requerimientos del negocio y las necesidades de sus usuarios. A partir de estas fuentes se identificaron los atributos de calidad más relevantes para el contexto de la aplicación. Estos atributos fueron priorizados por el equipo de desarrollo considerando las necesidades manifestadas por el cliente, las restricciones del proyecto y el impacto esperado de cada atributo sobre la evolución del sistema hacia un entorno de producción.

La selección de atributos de calidad se realizó en función de las necesidades y prioridades identificadas por los stakeholders, por lo que los atributos restantes de ISO/IEC 25010:2023 [ISO/IEC, 2023] quedan fuera del alcance de esta etapa del proyecto.

3.1.1. Mantenibilidad

Se define la mantenibilidad como la facilidad con la que el software admite modificaciones para evolucionar el producto. Este atributo es vital, ya que el sistema debe permitir que personas ajenas al equipo original realicen tareas de soporte y mejora. Basándose en la norma ISO/IEC 25010:2023 [ISO/IEC, 2023], se destacan tres subcaracterísticas fundamentales:

- **Modularidad:** Capacidad del producto para ser modificado sin afectar negativamente a otras partes del sistema.
- **Analizabilidad:** Facilidad para evaluar el impacto potencial de un cambio propuesto en el sistema.
- **Modificabilidad:** Capacidad de realizar cambios de forma eficiente, sin introducir defectos ni degradar la calidad existente.

3.1.2. Seguridad

Dado que el sistema almacena y gestiona datos sensibles —como la ubicación de usuarios y configuraciones críticas de construcción—, es imperativo garantizar su protección. Las sub-características priorizadas son:

- **Confidencialidad:** Asegurar que solo los usuarios autorizados tengan acceso a datos específicos.
- **Integridad:** Proteger los datos frente a modificaciones o eliminaciones accidentales o no autorizadas.
- **Autenticidad:** Capacidad de verificar rigurosamente la identidad de quien intenta acceder al sistema.

3.1.3. Flexibilidad (Escalabilidad)

El cliente proyecta un crecimiento constante en la base de usuarios, lo que exige que el sistema sea capaz de adaptarse a esta demanda. Dentro de la característica de flexibilidad, según la actualización de la norma ISO/IEC 25010:2023 [ISO/IEC, 2023], se destaca:

- **Escalabilidad:** Capacidad para gestionar de manera efectiva el incremento en la carga de trabajo, el volumen de datos o el número de usuarios.

3.1.4. Atributos de calidad priorizados

Tras el análisis realizado, se ha establecido el siguiente orden de prioridad para el sistema:

1. **Seguridad.**
2. **Flexibilidad / Escalabilidad.**
3. **Mantenibilidad.**

El siguiente *Alpha* que desarrollaremos será el de *Business Drivers*. Este *Alpha* incluye en su progresión la identificación de los objetivos de negocio, el establecimiento de principios derivados de estos, y el análisis de cómo dichos principios afectan —o no— a la arquitectura del sistema.

3.2. Objetivos del negocio

3.2.1. Integrarse en el mercado de reducción de la intensidad energética en Chile

- **Identificado:** La meta nace del Plan Nacional de Eficiencia Energética [Ministerio de Energía de Chile, 2023]. Bio Planner identifica esta meta país como un driver de negocio al querer formar parte de este proceso.
- **Principio establecido:** Bio Planner adopta como principio rector que toda vivienda diseñada en la plataforma debe contribuir activamente a la reducción de la intensidad energética nacional, es por esto que la plataforma desarrollada debe llegar de forma efectiva a cada constructor o arquitecto que la pueda necesitar.
- **Impacto en la arquitectura:** Este principio exige que la arquitectura esté preparada para escalar y dar servicio simultáneo a una gran cantidad de usuarios (arquitectos, constructoras e inmobiliarias) a lo largo de todo el país, dado que el negocio busca masificar su adopción y no limitarse a clientes puntuales. Esto implica decisiones como infraestructura con capacidad de balanceo de carga, bases de datos que soporten alta concurrencia.

3.3. Descripción de la arquitectura

Para este *Alpha* se presentan cuatro diagramas distintos correspondientes a una vista en detalle de cada capa, junto a una Vista de alto nivel del producto.

Cabe destacar que los diagramas de arquitectura presentados no muestran la totalidad de los módulos del sistema de forma individual: con el objetivo de mantener una representación legible y centrada en los aspectos estructurales relevantes, algunos módulos de menor granularidad fueron agrupados dentro de componentes de mayor nivel que describen su función en términos generales, omitiendo así detalles de implementación que no aportan a la comprensión global de la arquitectura.

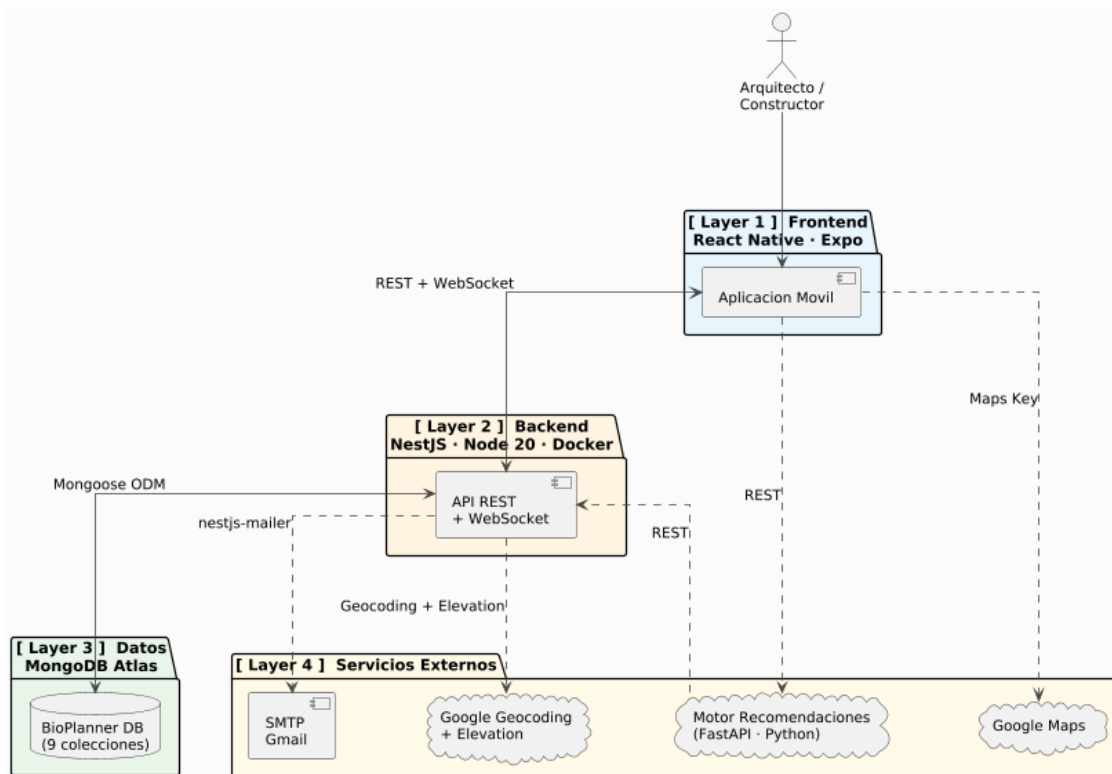


Figura 1: Vista de alto nivel de la arquitectura de BioPlanner.

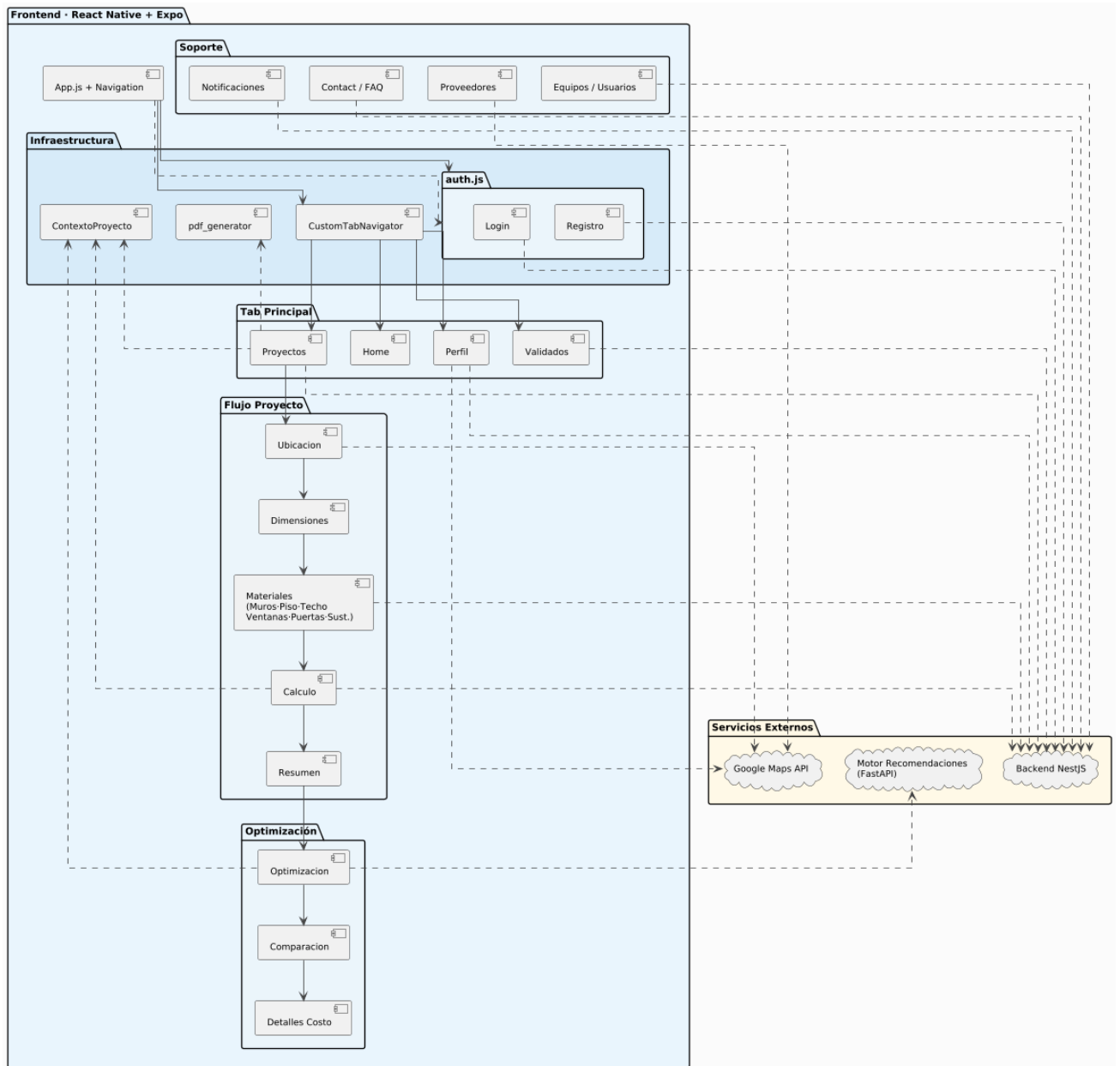


Figura 2: Diagrama de componentes — Capa Frontend.

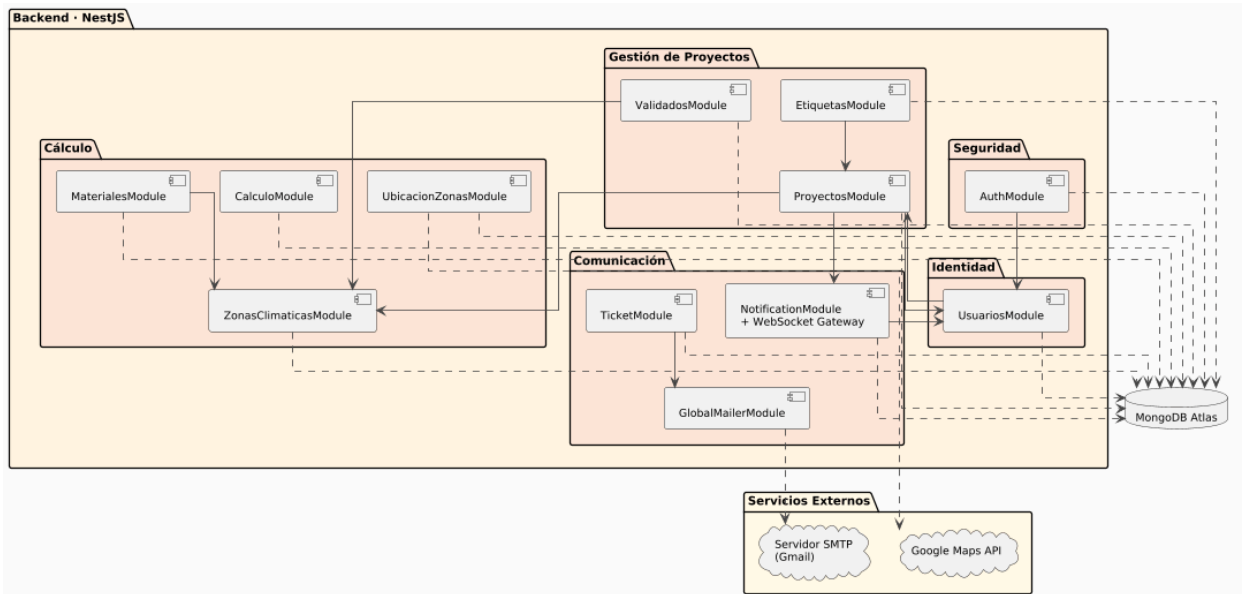


Figura 3: Diagrama de componentes — Capa Backend.

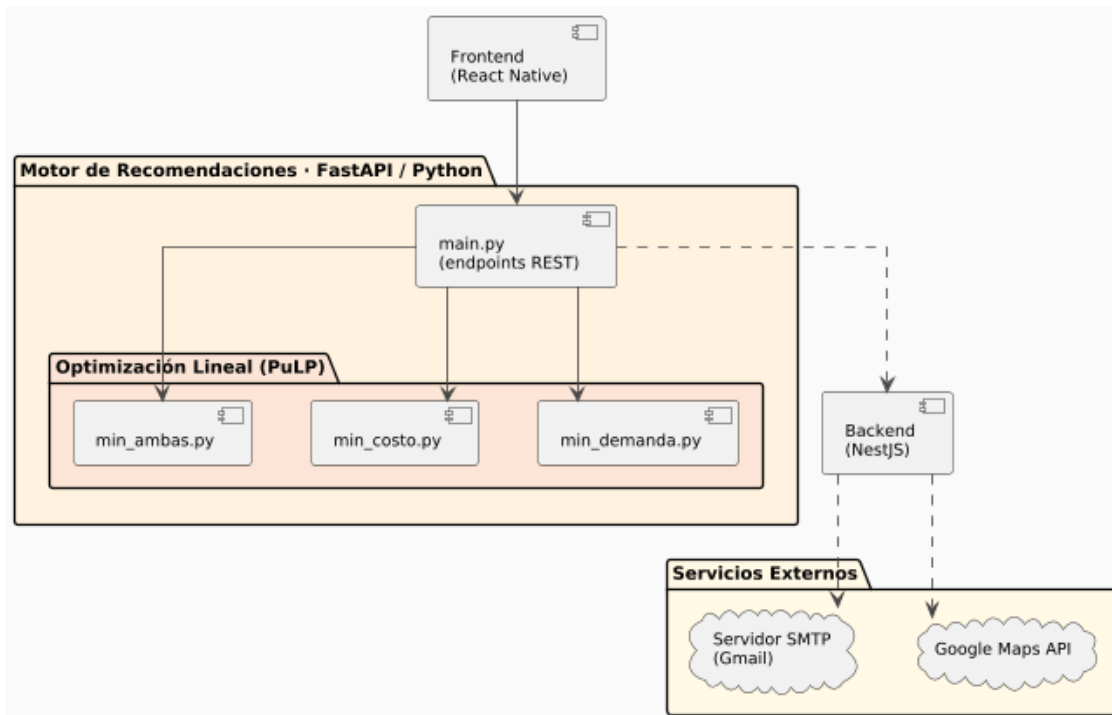


Figura 4: Diagrama de integración con servicios externos.

3.3.1. Descripción de alto nivel de BioPlanner

La Figura 1 refleja una arquitectura en capas (*Layered Architecture*) con cuatro niveles bien delimitados: presentación, lógica de negocio/API, persistencia y servicios externos.

Las pantallas se comunican con el backend via **REST + Bearer JWT** usando `fetch()`.

El flujo típico comienza con la interacción del usuario, que actualiza el contexto global y dispara una llamada HTTP al backend, el cual consulta MongoDB y retorna la respuesta para re-renderizar la vista. Las notificaciones, en cambio, fluyen de forma asíncrona por WebSocket. La pantalla de optimización es la excepción: se comunica directamente con el Motor de Recomendaciones (FastAPI), que a su vez consulta al backend para obtener materiales y datos de zonas climáticas antes de optimizar.

3.3.2. Descripción de la Capa Frontend de BioPlanner

La Figura 2 describe la capa del Frontend que está construido en **React Native + Expo**, siguiendo un patrón **Component-Based Architecture** propio del ecosistema React. La estructura se organiza en tres capas funcionales claramente identificables en el diagrama: una **capa de navegación** (`App.js` y `navigation.js`), una **capa de infraestructura compartida** (contexto global, autenticación y utilidades) y una **capa de presentación** compuesta por las pantallas (*screens*) agrupadas por dominio funcional.

El flujo típico de una acción de usuario es el siguiente:

1. El usuario navega desde Proyectos hacia el flujo de creación, pasando secuencialmente por Ubicación → Dimensiones → pantallas de Materiales.
2. Cada pantalla de materiales actualiza el contexto global con `actualizarProyecto()` al seleccionar un elemento.
3. Al llegar a `Calculo`, se recopila el estado acumulado del contexto y se envía al backend.
4. El usuario puede continuar hacia `Optimizacion`, donde el front llama directamente al motor de recomendaciones (FastAPI) enviando los datos del envoltorio térmico transformados localmente.
5. El resultado se muestra en `Comparacion` y `Detalles Costo`.

Frontera de comunicación externa: se identificaron tres integraciones reales con servicios externos:

- **Backend NestJS** (API_URL): consumido por prácticamente todas las pantallas para operaciones CRUD de proyectos, materiales, usuarios, tickets y notificaciones.
- **Motor de Recomendaciones FastAPI** (FASTAPI_URL): consumido exclusivamente desde `Optimizacion_Proyecto_Screen.js`.
- **Google Maps API**: consumida desde `Ubicacion`, `Proveedores` y `Perfil` para renderizado de mapas y selección de coordenadas.

3.3.3. Descripción de la Capa Backend de BioPlanner

La Figura 3 describe el Backend construido sobre **NestJS**, framework que impone de forma nativa una arquitectura **modular en capas**. Cada módulo de negocio sigue una estructura consistente de tres capas internas: **Controller** (capa de presentación / entrada HTTP), **Service** (capa de lógica de negocio) y **Schema** (capa de modelo de datos, mapeado a documentos de MongoDB vía Mongoose).

El diagrama agrupa los once módulos NestJS en cinco categorías funcionales:

- **Seguridad:** `AuthModule`, responsable de autenticación mediante JWT. Internamente compuesto por `AuthService` (emisión de tokens), `JwtStrategy` (validación del token entrante vía Passport) y `JwtAuthGuard` (guard aplicado a rutas protegidas).
- **Identidad:** `UsuariosModule`, gestiona el CRUD de usuarios y expone su servicio para ser consumido por otros módulos.
- **Gestión de Proyectos:** compuesta por `ProyectosModule` (núcleo de la aplicación, gestiona el ciclo de vida completo de un proyecto y su historial), `EtiquetasModule` (clasificación de proyectos) y `ValidadosModule` (casos de éxito validados).
- **Cálculo:** `CalculoModule` (motor de cálculo térmico, sin persistencia propia), `MaterialesModule` (catálogo de materiales constructivos), `ZonasClimaticasModule` (reglas térmicas por zona) y `UbicacionZonasModule` (resolución de zona climática a partir de coordenadas geográficas).
- **Comunicación:** `NotificationModule` (incluye un *WebSocket Gateway* para push en tiempo real), `TicketModule` (gestión de soporte) y `GlobalMailerModule` (envío de correo).

El flujo típico de una petición HTTP es:

1. La petición llega al **Controller** correspondiente.

2. Si la ruta está protegida, el `JwtAuthGuard` intercepta la petición y valida el token mediante `JwtStrategy` antes de continuar.
3. El `ValidationPipe` global (configurado en `main.ts` con `whitelist: true`) sanitiza el *payload*, rechazando cualquier propiedad no declarada en el DTO correspondiente.
4. El `Controller` delega la operación al `Service` inyectado.
5. El `Service` ejecuta la lógica de negocio, consultando otros servicios inyectados si es necesario (por ejemplo, `ProyectoService` consulta `ZonasClimaticasService`).
6. El `Service` interactúa con el modelo `Mongoose` correspondiente para persistir o consultar en **MongoDB**.
7. La respuesta se serializa y retorna al cliente.

Frontera de comunicación externa: se identificaron dos integraciones reales con servicios externos:

- **Google Maps API:** consumida desde `UbicacionZonaService` para resolver geocodificación y elevación a partir de coordenadas, necesarias para determinar la zona climática de un proyecto.
- **Servidor SMTP (Gmail):** consumido por `TicketModule` para el envío de correos de soporte.

3.4. Descripción de la Capa de servicios externos de BioPlanner

A diferencia del frontend y el backend, esta capa descrita en la Figura 4 no corresponde a un único sistema sino a la **integración entre tres servicios independientes**: el backend `NestJS` mantiene comunicaciones con un microservicio externo de optimización construido en **FastAPI (Python)**, y dos proveedores de infraestructura de terceros (Google Maps y un servidor SMTP). El motor de recomendaciones es un servicio desacoplado del backend principal, pero no es autónomo, ya que depende de él para obtener datos antes de poder ejecutar su lógica de negocio.

Internamente, el motor de recomendaciones sigue una estructura simple de **Script-based / Functional**. Un único punto de entrada orquesta llamadas a funciones de obtención de datos y luego delega a uno de tres módulos de optimización matemática.

Los tres módulos de optimización están contruidos sobre **PuLP**, una librería de programación lineal en Python que formula el problema como un `LpProblem` con variables de decisión binarias por cada elemento constructivo (muro, techo, piso, ventana, puerta).

El flujo de una solicitud de optimización es el siguiente:

1. El **frontend** envía un POST a uno de los tres endpoints, incluyendo los datos del proyecto, elementos fijados por el usuario y coordenadas geográficas.
2. `main.py` ejecuta secuencialmente tres funciones de enriquecimiento de datos: `get_elementos()` (consulta materiales disponibles no fijados por el usuario para la zona climática dada), `get_datos_zona()` (obtiene horas de calefacción, refrigeración y delta de temperatura) y `get_datos_demanda()` (obtiene la demanda de referencia según latitud/longitud).
3. Cada una de estas funciones realiza una llamada GET síncrona al backend NestJS usando la librería `requests`.
4. Una vez completados los datos, se invoca al módulo de optimización correspondiente (`min_demanda`, `min_costo` o `min_ambas`), que resuelve el problema de programación lineal y retorna la solución óptima.
5. El resultado se serializa como `LPResponse` y se retorna al frontend.

Frontera de comunicación externa: Su única dependencia externa es el **backend NestJS**.

3.5. Decisiones de arquitectura

Al igual que en el *Alpha* anterior, aquí solo se mostrará la última parte de la progresión, correspondiente a las decisiones arquitectónicas documentadas. Estas decisiones no existían formalmente, pero por medio de discusiones con el equipo de desarrollo fue posible levantar los siguientes ADRs:

ADR-001 — Arquitectura modular para el backend	
Estado	Implementada
Contexto	El backend de BioPlanner debía ser desarrollado por un equipo de seis estudiantes en paralelo bajo restricciones de tiempo propias de una feria de software universitaria.
Decisión	Se estructura el sistema en módulos independientes por dominio, cada uno con su propio controlador, servicio y <i>schema</i> de persistencia.
Consecuencias positivas	Separación clara de responsabilidades, facilidad para el desarrollo paralelo, convenciones explícitas que reducen decisiones de diseño <i>ad-hoc</i> , y soporte nativo para inyección de dependencias.
Consecuencias negativas	La modularidad traslada al equipo la responsabilidad de definir cuidadosamente los límites de cada módulo: si dos dominios se modelan con necesidades de conocerse mutuamente, se corre el riesgo de introducir dependencias circulares entre módulos.
Trade-off	Velocidad de desarrollo y convención sobre configuración, a costa de exigir disciplina activa en el diseño de límites entre módulos para evitar dependencias cruzadas a medida que el dominio crece.

Tabla 2: ADR-001: Arquitectura modular.

ADR-002 — Persistencia NoSQL	
Estado	Implementada
Contexto	El modelo de datos de BioPlanner es variable por naturaleza: un proyecto habitacional puede tener distintas combinaciones de materiales, orientaciones y configuraciones que no encajan fácilmente en un esquema relacional rígido.
Decisión	Persistencia con enfoque NoSQL, utilizando una base de datos documental (MongoDB Atlas), accedida de forma centralizada a través de Mongoose como ODM en todos los módulos de dominio.
Consecuencias positivas	Flexibilidad de esquema para modelar proyectos habitacionales complejos sin migraciones rígidas, disponibilidad de un servicio gestionado sin infraestructura adicional, y soporte de referencias entre colecciones.
Consecuencias negativas	A diferencia de una base de datos relacional, MongoDB no impone integridad referencial a nivel de motor: las relaciones entre colecciones dependen enteramente de que la aplicación mantenga la consistencia, sin garantía nativa de que un documento referenciado exista o no haya sido eliminado. Asimismo, al no usarse transacciones multi-documento, las escrituras que afectan a más de una colección quedan expuestas a estados parcialmente inconsistentes si una de ellas falla.
Trade-off	Flexibilidad de esquema y velocidad de desarrollo a costa de consistencia referencial garantizada por el motor de base de datos: la integridad de las relaciones entre colecciones recae completamente en la disciplina de la capa de aplicación.

Tabla 3: ADR-002: Persistencia NoSQL.

ADR-003 — Microservicio externo para el motor de recomendaciones sin autenticación	
Estado	Implementada
Contexto	BioPlanner requiere un motor de optimización capaz de recomendar combinaciones de materiales que minimicen la demanda energética, el costo de construcción, o ambos simultáneamente, dado un conjunto de restricciones normativas y de diseño.
Decisión	Microservicio independiente en Python utilizando FastAPI y PuLP como solver de programación lineal, consumido por el backend y el frontend mediante llamadas HTTP simples, sin mecanismo de autenticación entre servicios.
Consecuencias positivas	Separación de responsabilidades entre el backend principal y el motor de optimización, escalabilidad independiente de ambos servicios, y aprovechamiento del ecosistema científico de Python para programación lineal.
Consecuencias negativas	El motor de recomendaciones depende de otro servicio para obtener los datos que necesita antes de optimizar, lo que introduce latencia adicional y un punto de fallo distribuido. La ausencia de autenticación entre servicios es más grave: cualquier actor con acceso a la red podría invocar el microservicio o los datos que este consulta, sin que exista una capa de verificación de identidad entre componentes.
Trade-off	Separación tecnológica y escalabilidad independiente, a costa de latencia adicional y una frontera de comunicación entre servicios sin control de acceso.

Tabla 4: ADR-003: Microservicio externo para el motor de recomendaciones sin autenticación.

ADR-004 — Parámetros normativos embebidos en el servicio de cálculo	
Estado	Implementada
Contexto	El cálculo de demanda energética requiere valores definidos por la reglamentación térmica vigente, incluyendo coeficientes de transmitancia máxima, horas de calefacción y refrigeración por zona climática, y umbrales de calificación energética según la escala CEV.
Decisión	Los parámetros normativos son definidos como constantes directamente en el código fuente del CalculoService, sin externalización a base de datos ni a un archivo de configuración versionado.
Consecuencias positivas	Implementación simple y sin dependencias adicionales, acceso inmediato a los valores sin latencia de consulta, y menor complejidad de despliegue.
Consecuencias negativas	Cualquier actualización de la normativa térmica requiere modificar el código fuente, ejecutar un nuevo ciclo de pruebas y realizar un despliegue completo. Dado que BioPlanner entrega una estimación orientativa de la demanda energética y no sustituye la calificación oficial CEV realizada por un especialista, este período de desactualización no implica un riesgo legal directo, pero sí compromete la utilidad y precisión de la herramienta frente a los usuarios.
Trade-off	Simplicidad de implementación y velocidad de desarrollo, a costa de la capacidad de mantener la estimación alineada con la normativa vigente sin intervención de desarrollo.

Tabla 5: ADR-004: Parámetros normativos embebidos en el servicio de cálculo.

ADR-005 — Validación centralizada no forzada en el backend	
Estado	Implementada
Contexto	El backend define un mecanismo centralizado de autenticación mediante <code>JwtStrategy</code> y <code>JwtAuthGuard</code> , pensado para ser aplicado de forma uniforme sobre las rutas que requieren un usuario autenticado.
Decisión	La aplicación del guard centralizado se delega a cada módulo de forma individual, sin un mecanismo que obligue su uso de manera transversal (por ejemplo, un guard global a nivel de aplicación).
Consecuencias positivas	Cada módulo puede decidir con flexibilidad qué rutas requieren autenticación y cuáles son públicas (por ejemplo, registro o login), sin que el guard global deba excluirlas explícitamente.
Consecuencias negativas	Esta flexibilidad delega en cada desarrollador la responsabilidad de recordar aplicar el guard correctamente. En la práctica, esto derivó en una aplicación inconsistente: <code>UsuariosController</code> no utiliza <code>JwtAuthGuard</code> , sino una validación manual del token duplicada dentro del propio controlador, con un secreto por defecto en texto plano como respaldo si la variable de entorno no está definida; y el endpoint <code>GET /usuarios/:id</code> no valida autenticación alguna, exponiendo datos de cualquier usuario a quien conozca su identificador.
Trade-off	Flexibilidad de configuración por módulo, a costa de consistencia y de una superficie de ataque que depende enteramente de que cada desarrollador no omita la protección de una ruta.

Tabla 6: ADR-005: Autenticación no centralizada a nivel de backend.

3.6. Evaluación de la Arquitectura Centrada en Decisiones

La evaluación de arquitectura se realizó mediante un procedimiento inspirado en el método DCAR [van Heesch *et al.*, 2014], aplicado sobre los ADRs documentados en la sección anterior. DCAR fue seleccionado como referencia ya que se enfoca en repasar decisiones arquitectónicas documentadas, lo que hace sentido al tener un conjunto de decisiones reconstruidas previamente.

Con el propósito de reducir la subjetividad en el proceso de evaluación, la aplicación del procedimiento se realizó mediante una sesión en la que participaron los integrantes del equipo de desarrollo con mayor participación en el diseño de la arquitectura y conocimiento de las decisiones arquitectónicas adoptadas durante el proyecto. El consenso obtenido durante la

sesión permitió identificar las principales fortalezas, riesgos y oportunidades de mejora de la arquitectura, cuyos resultados se presentan a continuación.

A continuación se presenta la evaluación de cada decisión utilizando el esquema de semáforo propuesto por DCAR:

Color	Criterio
Verde	Aceptada sin reparos, ninguna fuerza en contra compromete un atributo priorizado; no se requiere acción.
Amarillo	Aceptada con reparos, existen fuerzas en contra sobre algún atributo priorizado, pero el riesgo es puntual y mitigable sin descartar la decisión.
Rojo	No aceptada, las fuerzas en contra afectan de forma estructural y simultánea a dos o más atributos priorizados, o comprometen críticamente al de mayor prioridad.

Tabla 7: Esquema de semáforo DCAR.
Fuente: adaptado de [van Heesch *et al.*, 2014].

ADR-001 — Arquitectura modular para el backend (Tabla 2)

- **Fuerzas a favor:** La modularización por dominio favorece la Mantenibilidad (prioridad 3) al permitir desarrollo paralelo entre los integrantes del equipo y establecer convenciones explícitas (controlador, servicio, schema por módulo), reduciendo el tiempo de incorporación de nuevos desarrolladores al proyecto y la necesidad de decisiones de diseño *ad-hoc*.
- **Fuerzas en contra:** La modularidad traslada al equipo la responsabilidad de trazar cuidadosamente los límites entre módulos; cuando dos dominios se modelan con necesidad de conocerse mutuamente, como ocurre entre `ProyectosModule` y `UsuariosModule`, se materializa una dependencia circular resuelta mediante `forwardRef()`. Esto reduce la Analizabilidad (Mantenibilidad, prioridad 3), aunque no compromete la capacidad de escalar el backend horizontalmente, ya que `forwardRef()` se resuelve en tiempo de arranque dentro de un mismo proceso y no depende de comunicación entre servicios separados.

Resultado: ■ **Aceptada sin reparos.** *Justificación:* la dependencia circular identificada es un hallazgo técnico real, pero el escenario que la convertiría en un problema concreto —la descomposición de `Proyectos` y `Usuarios` en microservicios independientes— no forma parte del alcance ni la hoja de ruta actual de BioPlanner. Proponer una mejora de diseño para mitigar un riesgo que no se materializará dentro del horizonte del proyecto no representa un

uso eficiente del esfuerzo de desarrollo, por lo que se documenta como punto de atención sin acción correctiva asociada.

ADR-002 — Persistencia NoSQL (Tabla 3)

- **Fuerzas a favor:** El esquema flexible de una base documental favorece la Flexibilidad/Escalabilidad (prioridad 2) al permitir representar la variabilidad de los proyectos habitacionales sin migraciones de esquema, y el uso de un servicio gestionado en la nube reduce la carga operativa, favoreciendo indirectamente la Mantenibilidad (prioridad 3). Además, el acceso a datos se mantiene centralizado a través de Mongoose como único ODM en todos los módulos de dominio, sin vías paralelas de conexión que escapen a esta gestión.
- **Fuerzas en contra:** al no imponerse integridad referencial a nivel de motor, la consistencia entre colecciones relacionadas depende enteramente de la disciplina de la capa de aplicación, y la ausencia de transacciones multi-documento expone al sistema a estados parcialmente inconsistentes ante fallos parciales de escritura. Esto representa un riesgo moderado para la Mantenibilidad (prioridad 3), pero no compromete directamente la Seguridad (prioridad 1).

Resultado: ■ **Aceptada sin reparos.** *Justificación:* la decisión de adoptar una base documental gestionada está alineada con la prioridad de Escalabilidad y no introduce riesgos sobre la Seguridad, dado que todo acceso a los datos transita por una única vía centralizada (Mongoose). La falta de integridad referencial estricta es un costo inherente y aceptado del modelo NoSQL, mitigable mediante disciplina en la capa de aplicación, sin que esto ponga en riesgo las operaciones críticas de BioPlanner a corto o mediano plazo.

ADR-003 — Microservicio externo para el motor de recomendaciones (Tabla 4)

- **Fuerzas a favor:** la separación del motor de optimización en un servicio independiente favorece la Escalabilidad (prioridad 2), al permitir escalar el componente de cálculo intensivo de forma independiente del backend principal, y reduce el acoplamiento tecnológico al aprovechar el ecosistema científico de Python sin comprometer la Mantenibilidad del backend en NestJS.
- **Fuerzas en contra:** el ADR no documenta un mecanismo explícito de autenticación o autorización entre el backend y el microservicio de recomendaciones, lo que constituye una fuerza en contra de la Seguridad (prioridad 1), dado que se introduce un canal de comunicación adicional sin controles de identidad descritos. Asimismo, la dependencia de red introduce un punto adicional de fallo que, si bien no corresponde a un atributo formalmente priorizado, debilita la disponibilidad percibida del sistema.

Resultado: ■ **Aceptada con reparos.** *Justificación:* la decisión de externalizar el motor de recomendaciones es coherente con la prioridad de Escalabilidad, pero la ausencia de un contrato de seguridad explícito entre servicios —en un sistema donde la Seguridad es el atributo de mayor prioridad— debe resolverse antes del despliegue en producción, por ejemplo mediante autenticación de servicio a servicio (API keys, JWT de servicio o TLS mutuo).

ADR-004 — Parámetros normativos embebidos en el servicio de cálculo (Tabla 5)

- **Fuerzas a favor:** La simplicidad de mantener los parámetros como constantes en el código reduce la complejidad de despliegue, favorece la velocidad de desarrollo y elimina dependencias adicionales, sin afectar directamente los atributos de Seguridad.
- **Fuerzas en contra:** Cualquier modificación en la normativa térmica o en el modelo de cálculo utilizado para estimar la demanda energética requiere un nuevo ciclo de desarrollo, pruebas y despliegue, afectando principalmente la Mantenibilidad (prioridad 3), en particular su subcaracterística de Modificabilidad. Además, mantener la lógica y los parámetros acoplados al servicio dificulta la evolución del sistema hacia alternativas más escalables, como la externalización de la configuración o la separación del motor de cálculo en un microservicio independiente.

Resultado: ■ **Aceptada con reparos.** *Justificación:* La decisión no representa un riesgo crítico para la operación de BioPlanner, dado que la aplicación entrega una estimación orientativa de la demanda energética y no reemplaza la calificación oficial CEV realizada por un especialista. Sin embargo, mantener los parámetros normativos y la lógica de cálculo embebidos en el código limita la capacidad de adaptar rápidamente el sistema frente a cambios en la normativa o a futuras mejoras del algoritmo de estimación. Como alternativa, se recomienda externalizar los parámetros a una fuente de configuración versionada y desacoplada del ciclo de despliegue, o bien evolucionar el módulo de cálculo hacia un servicio independiente que facilite su mantenimiento y actualización.

ADR-005 — Autenticación centralizada no forzada en el backend (Tabla 6)

- **Fuerzas a favor:** Delegar la aplicación del guard a cada módulo permite excluir con flexibilidad las rutas públicas (registro, login) sin necesidad de una lista de excepciones sobre un guard global, favoreciendo levemente la Mantenibilidad (prioridad 3) en los módulos donde sí se aplicó correctamente.
- **Fuerzas en contra:** La falta de un mecanismo que obligue el uso consistente del guard centralizado derivó en una protección desigual entre módulos: `UsuariosController` implementa una validación de token manual y duplicada, con un secreto por defecto en texto plano como respaldo, y el endpoint `GET /usuarios/:id` no valida autenticación alguna. Esto compromete directamente la Seguridad (prioridad 1).

Resultado: ■ **No aceptada.** *Justificación:* a diferencia de los hallazgos anteriores, este compromiso directamente el atributo de mayor prioridad del sistema —la Seguridad— mediante la exposición no autenticada de datos personales de usuarios, lo cual constituye un riesgo inmediato y no contingente a ningún escenario futuro. Se recomienda aplicar un guard global a nivel de aplicación (APP_GUARD) con excepciones explícitas mediante un decorador `@Public()` para las rutas que deban permanecer sin autenticación, eliminando la validación manual duplicada y unificando el uso de `JwtAuthGuard` en todos los módulos.

3.7. Propuestas de mejoras en base a la evaluación inspirada en DCAR

3.7.1. Propuesta para ADR-003: Implementar *Authenticate Actors* entre back-end y servicios externos.

La evaluación evidenció que la comunicación entre el backend y el microservicio de recomendaciones no incorpora mecanismos para autenticar el origen de las solicitudes, exponiendo la arquitectura a riesgos que afectan la seguridad, particularmente la confidencialidad y autenticidad de la información intercambiada.

Se propone incorporar autenticación mutua mediante JWT de servicio, utilizando un secreto compartido distinto al empleado para la autenticación de usuarios. Cada servicio adjunta un token firmado en las solicitudes que realiza y valida el token recibido antes de procesarlas.

Esta mejora implementa la táctica *Authenticate Actors*, permitiendo verificar la identidad de ambos servicios sin modificar la arquitectura existente ni incorporar infraestructura adicional. Asimismo, reutiliza bibliotecas ya disponibles tanto en NestJS como en FastAPI, facilitando su adopción.

Trade-off

Cada comunicación incorpora el costo asociado a la generación y validación de tokens, además de requerir la administración y rotación periódica del secreto compartido.

3.7.2. Propuesta para ADR-004: Externalizar parámetros de calculo a la persistencia

La evaluación identificó que los parámetros normativos utilizados durante el cálculo energético se encuentran definidos directamente en el código fuente, dificultando su actualización frente a futuras modificaciones de la normativa y afectando la mantenibilidad, particularmente la modificabilidad del sistema.

Se propone externalizar dichos parámetros hacia una colección de configuración en MongoDB, consultada dinámicamente por `CalculoService`, siguiendo el mismo mecanismo ya

utilizado para almacenar la información de las zonas climáticas.

Esta propuesta implementa la táctica de externalización de configuración, permitiendo adaptar futuras modificaciones normativas mediante cambios en los datos y no en el código fuente, reduciendo el esfuerzo de mantenimiento y evitando nuevos despliegues por cambios regulatorios.

Trade-off

Cada cálculo requiere realizar una consulta adicional a la base de datos, incrementando levemente el tiempo de ejecución.

Evolución futura

Si el cálculo energético evolucionara hacia procesos computacionalmente más exigentes, podría resultar conveniente desacoplar `CalculoService` como un servicio independiente, siguiendo el precedente establecido para el motor de recomendaciones.

3.7.3. Propuesta para ADR-005: Centralizar la autenticación mediante guard global

La evaluación identificó que la aplicación del mecanismo de autenticación se delega individualmente a cada módulo, sin un componente que obligue su uso de forma transversal, lo cual derivó en una protección desigual entre controladores y afecta directamente la seguridad, particularmente la confidencialidad de los datos de usuario. Se propone registrar `JwtAuthGuard` como guard global e incorporar un decorador `@Public()` que permita excluir explícitamente las rutas que deban permanecer sin autenticación (por ejemplo, registro o login), eliminando además la validación manual de token actualmente duplicada en `UsuariosController`. Esta propuesta implementa la táctica *Authenticate Actors* de forma centralizada, asegurando que toda nueva ruta agregada al backend quede protegida por defecto, sin depender de que cada desarrollador recuerde aplicar el guard correspondiente.

Trade-off

Toda ruta nueva queda protegida por defecto, por lo que cada endpoint que deba ser público requiere marcarse explícitamente con `@Public()`; omitir este decorador en una ruta que sí debía ser pública la dejaría inaccesible hasta detectarse en pruebas.

CAPÍTULO 4

PLAN DE IMPLEMENTACIÓN Y FACTIBILIDAD

En esta sección se presenta un plan de implementación para las cinco propuestas de mejora descritas en el punto anterior, abordando su factibilidad técnica y económica.

El orden de implementación sigue la severidad obtenida en la evaluación de la arquitectura. Se aborda primero la decisión con resultado Rojo, ya que compromete críticamente la Seguridad, atributo de mayor prioridad para el cliente. Le siguen las decisiones Amarillas, en el orden en que fueron documentadas, dado que representan riesgos puntuales y mitigables que no comprometen la operación actual de BioPlanner.

Para cada mejora se documenta el esfuerzo estimado en persona-horas, el riesgo de implementación, las dependencias respecto de otras mejoras, los requerimientos de infraestructura adicional y el criterio de aceptación que permite verificar que la mejora fue correctamente incorporada.

Supuestos de estimación. La estimación de esfuerzo en persona-horas se obtuvo mediante la descomposición de tareas y juicio experto, asumiendo un desarrollador familiarizado con el stack actual del proyecto. No se asume el uso de una metodología o herramienta de desarrollo específica (por ejemplo, asistentes de programación basados en IA); el esfuerzo real de implementación puede variar según las herramientas y prácticas que emplee el equipo que eventualmente ejecute este plan.

La tarifa sugerida ronda los \$8000 - \$14000 CLP por hora.

Campo	Detalle
Mejora de referencia	ADR-003 — <i>Authenticate Actors</i> entre backend y microservicio de recomendaciones
Resultado DCAR	Amarillo
Esfuerzo estimado	10–14 persona-horas: definición y firma del JWT de servicio en el backend NestJS (3 h), validación del token en el microservicio FastAPI mediante Depends (2–3 h), implementación de la autenticación en sentido inverso para las consultas del microservicio hacia el backend (3 h), pruebas de integración del flujo bidireccional y gestión del secreto compartido (2–3 h).
Valor monetario estimado	\$140.000–\$196.000 CLP
Riesgo de implementación	Medio. Al ser una autenticación bidireccional entre dos servicios desarrollados en stacks distintos (NestJS y FastAPI), un error en la validación del token puede dejar inoperativo el flujo completo de recomendaciones.
Dependencias	Ninguna respecto de otras mejoras del plan.
Infraestructura adicional	Ninguna nueva; requiere únicamente la gestión de un secreto compartido entre ambos servicios, distinto al utilizado para la autenticación de usuarios.
Criterio de aceptación	Ninguno de los endpoints <code>/recomendaciones/*</code> del microservicio debe procesar una solicitud sin un JWT de servicio válido; de igual forma, el backend debe rechazar las consultas entrantes del microservicio (<code>/materiales</code> , <code>/reglas-termicas/zona-climatica</code> , <code>/zonas/demandas</code>) que no incluyan un token válido.

Tabla 8: Plan de implementación para la mejora de ADR-003. Fuente: Elaboración propia.

Campo	Detalle
Mejora de referencia	ADR-004 — Externalización de parámetros de cálculo a la persistencia
Resultado DCAR	Amarillo
Esfuerzo estimado	6–10 persona-horas: diseño y creación de la colección de configuración normativa en MongoDB (2 h), modificación de <code>CalculoService</code> para consultar los umbrales en lugar de utilizarlos como constantes (2–3 h), migración de los valores actualmente hardcodeados y pruebas de regresión sobre el flujo de precalificación energética (2–5 h).
Valor monetario estimado	\$84.000–\$140.000 CLP
Riesgo de implementación	Bajo. El patrón ya está validado en el sistema para los coeficientes de <code>ZonaClimatica</code> , que sigue el mismo mecanismo de configuración externalizada.
Dependencias	Ninguna. Puede implementarse de forma independiente del resto de las mejoras.
Infraestructura adicional	Ninguna. Se utiliza la misma base de datos documental (MongoDB Atlas) ya empleada por el sistema, sin componentes externos adicionales.
Criterio de aceptación	<code>CalculoService</code> no debe contener umbrales o parámetros normativos definidos como constantes en el código fuente; toda actualización de la normativa debe poder realizarse mediante una modificación de datos en la colección de configuración, sin requerir un nuevo despliegue.

Tabla 9: Plan de implementación para la mejora de ADR-004. Fuente: Elaboración propia.

Campo	Detalle
Mejora de referencia	ADR-005 — Centralización de la autenticación mediante guard global
Resultado DCAR	Rojo
Esfuerzo estimado	12–16 persona-horas: registro de JwtAuthGuard como APP_GUARD global (1 h), implementación del decorador @Public() y su reflector asociado (2 h), refactorización de UsuariosController para eliminar la validación manual de token duplicada (3 h), revisión y marcado de rutas públicas en el resto de los controladores (3–4 h), pruebas de regresión sobre todos los endpoints del backend (3–5 h).
Valor monetario estimado	\$168.000–\$224.000 CLP
Riesgo de implementación	Medio. Al proteger todas las rutas por defecto, existe riesgo de dejar inaccesible involuntariamente alguna ruta que debía permanecer pública si no se marca correctamente con @Public(), lo que requiere una revisión exhaustiva de todos los controladores existentes.
Dependencias	Ninguna respecto de otras mejoras del plan.
Infraestructura adicional	Ninguna. Se utiliza el mecanismo de <i>providers</i> y <i>guards</i> ya disponible de forma nativa en NestJS.
Criterio de aceptación	Ninguna ruta del backend debe quedar accesible sin autenticación salvo que esté explícitamente marcada con @Public(); UsuariosController no debe contener lógica de validación de token manual ni secretos por defecto en el código fuente.

Tabla 10: Plan de implementación para la mejora de ADR-005. Fuente: Elaboración propia.

CAPÍTULO 5

CONCLUSIONES

El presente trabajo tuvo como objetivo preparar el diseño de la arquitectura de Software de **BioPlanner** para su eventual despliegue en un ambiente de producción, mediante la evaluación de su arquitectura de software utilizando marcos teóricos reconocidos internacionalmente. A partir de los resultados obtenidos, es posible afirmar que el objetivo general fue alcanzado: se logró evaluar sistemáticamente la arquitectura existente, identificar riesgos concretos sobre los atributos de calidad priorizados y formular propuestas de mejora fundamentadas en tácticas y patrones de diseño reconocidos.

En cuanto a los objetivos específicos, el primero —evaluar la arquitectura actual identificando riesgos y oportunidades de mejora— se cumplió mediante la aplicación de un procedimiento inspirado en el método *DCAR* sobre los cinco ADR documentados a partir de discusiones con el equipo de desarrollo. Este proceso permitió constatar que la arquitectura de BioPlanner, si bien funcional y coherente con las restricciones de tiempo bajo las cuales fue concebida, presenta debilidades concretas en el atributo de mayor prioridad para el sistema: la **Seguridad**. En particular, la ausencia de autenticación entre el backend y el microservicio de recomendaciones (**ADR-003**) y la aplicación inconsistente del mecanismo de autenticación a nivel de backend (**ADR-005**) constituyen los hallazgos más relevantes del proceso de evaluación, siendo este último calificado con severidad *roja* por exponer datos de usuarios sin control de acceso alguno.

El segundo objetivo específico, relativo al diseño de una propuesta de mejora arquitectónica, se materializó en cuatro propuestas concretas: la incorporación de autenticación de servicio mediante *JWT* entre backend y microservicio (*Authenticate Actors*), la centralización de la autenticación a través de un *guard* global y la externalización de los parámetros normativos del cálculo energético hacia la capa de persistencia. Cada una de estas propuestas se vinculó explícitamente con el atributo de calidad que buscaba fortalecer y con su correspondiente *trade-off*, evitando así presentar mejoras sin un análisis de sus costos asociados.

Respecto del tercer objetivo, se elaboró un plan de implementación para las mejoras priorizadas, ordenado según la severidad obtenida en la evaluación de la arquitectura, comenzando por la decisión de resultado *rojo*. Para cada mejora se documentó el esfuerzo estimado, el riesgo de implementación, las dependencias respecto de otras mejoras y un criterio de aceptación verificable, entregando así una hoja de ruta factible para que el equipo de desarrollo pueda incorporar estas mejoras de forma incremental.

Como principal aporte de este trabajo se destaca el haber dotado a Bio Planner de un conjunto de decisiones arquitectónicas formalmente documentadas, las cuales no existían previamente de manera explícita, sino que fueron reconstruidas a partir del conocimiento tácito del equipo de desarrollo. Esta documentación constituye en sí misma un activo para el proyecto, en la medida en que facilita la incorporación de nuevos desarrolladores y sien-

ta una base objetiva sobre la cual evaluar futuras decisiones de diseño. Resulta interesante notar, además, que los **ADR** levantados en este trabajo pueden interpretarse como una forma de *deuda técnica* previamente implícita en el sistema: decisiones tomadas bajo presión de tiempo que, al no estar documentadas, no podían ser evaluadas ni priorizadas de forma sistemática. Su formalización como ADR permitió justamente traer esta deuda a un plano visible y abordable, evidenciando que la documentación de decisiones arquitectónicas no solo aporta trazabilidad, sino que constituye en sí misma un mecanismo de gestión temprana de *deuda técnica*.

Entre las limitaciones del trabajo cabe mencionar que la evaluación se centró en un subconjunto acotado de atributos de calidad —**Seguridad, Flexibilidad/Escalabilidad y Mantenibilidad**—, priorizados según las necesidades manifestadas por el cliente, dejando fuera del alcance otras características del modelo ISO/IEC 25010:2023 [ISO/IEC, 2023] que podrían resultar relevantes en etapas posteriores del proyecto. Asimismo, el plan de implementación presentado corresponde a una estimación de esfuerzo basada en juicio experto, por lo que su validación definitiva requerirá de la ejecución efectiva de las mejoras propuestas.

Como trabajo futuro, sería de interés extender el alcance de la evaluación a otros atributos de calidad no abordados en este trabajo, como la **Usabilidad** o la **Compatibilidad**, a medida que el producto se aproxime a su entrada en producción. Asimismo, dado que la **Ley 21.719**[Ministerio Secretaría General de la Presidencia, 2024] sobre protección de datos personales entrará en plena vigencia el 1 de diciembre de 2026 se recomienda evaluar el cumplimiento de BioPlanner respecto de esta nueva normativa antes de su entrada en vigor, en particular en lo referido al tratamiento de datos personales de los usuarios de la plataforma.

REFERENCIAS BIBLIOGRÁFICAS

- [Boehm, 1978] Boehm, B. W. (1978). *Characteristics of Software Quality*. North Holland Press.
- [Clements *et al.*, 2002] Clements, P., Kazman, R., y Klein, M. (2002). *Evaluating Software Architectures: Methods and Case Studies*. SEI Series in Software Engineering. Addison-Wesley, Boston, MA.
- [Clements *et al.*, 2000] Clements, P. C., Kazman, R., y Klein, M. (2000). Evaluating software architectures with arid. Technical Report CMU/SEI-2000-TN-007, Software Engineering Institute, Carnegie Mellon University.
- [Cruz *et al.*, 2023] Cruz, H., Astudillo, H., y Zapata-Jaramillo, C. M. (2023). Extending the semat kernel to represent and assess software architecture evaluations. En *Proceedings of the 2023 XLIX Latin American Computer Conference (CLEI)*, pp. 1–8. IEEE.
- [Dromey, 1995] Dromey, R. G. (1995). A model for software product quality. *IEEE Transactions on Software Engineering*, 21(2):146–162.
- [Grady y Caswell, 1987] Grady, R. B. y Caswell, D. L. (1987). *Software Metrics: Establishing a Company-Wide Program*. Prentice-Hall, USA.
- [ISO/IEC, 2001] ISO/IEC (2001). ISO/IEC 9126: Information technology – product quality – part 1: Quality model.
- [ISO/IEC, 2023] ISO/IEC (2023). Systems and software engineering — systems and software quality requirements and evaluation (SQuaRE) — product quality model. Technical Report ISO/IEC 25010:2023, International Organization for Standardization / International Electrotechnical Commission, Geneva, Switzerland.
- [Jacobson *et al.*, 2013] Jacobson, I., Ng, P.-W., McMahon, P. E., Spence, I., y Lidman, S. (2013). *The Essence of Software Engineering: Applying the SEMAT Kernel*. Addison-Wesley Professional.
- [Kazman *et al.*, 1994] Kazman, R., Bass, L., Abowd, G., y Webb, M. (1994). Saam: A method for analyzing the properties of software architectures. En *Proceedings of the 16th International Conference on Software Engineering*, pp. 81–90. IEEE Computer Society.
- [McCall *et al.*, 1977] McCall, J. A., Richards, P. K., y Walters, G. F. (1977). Factors in software quality: Volume II. Technical report, Defense Technical Information Center.
- [Ministerio de Energía de Chile, 2023] Ministerio de Energía de Chile (2023). Plan nacional de eficiencia energética 2022-2026. Publicado en el Diario Oficial el 25 de abril de 2023, en cumplimiento del Artículo 1 de la Ley 21.305.

- [Ministerio de Vivienda y Urbanismo, 2024] Ministerio de Vivienda y Urbanismo (2024). Nueva reglamentación térmica: Actualización de la ordenanza general de urbanismo y construcciones (oguc). Accedido el 16 de abril de 2024.
- [Ministerio de Vivienda y Urbanismo, 2025] Ministerio de Vivienda y Urbanismo (2025). *Manual de Procedimientos: Calificación Energética de Viviendas en Chile (CEV)*. Gobierno de Chile, Santiago, Chile. Accedido: 17 de abril de 2026.
- [Ministerio Secretaría General de la Presidencia, 1999] Ministerio Secretaría General de la Presidencia (1999). Ley 19.628: Sobre protección de la vida privada. Biblioteca del Congreso Nacional de Chile. Última modificación: 23 de julio de 2011 mediante Ley 20.521.
- [Ministerio Secretaría General de la Presidencia, 2024] Ministerio Secretaría General de la Presidencia (2024). Ley 21.719: Regula la protección y el tratamiento de los datos personales y crea la agencia de protección de datos personales. Biblioteca del Congreso Nacional de Chile. Publicada en el Diario Oficial el 13 de diciembre de 2024. Entra en vigencia el 1 de diciembre de 2026.
- [Organización de las Naciones Unidas,] Organización de las Naciones Unidas. Objetivos de desarrollo sostenible.
- [SARA Working Group, 2002] SARA Working Group (2002). Software Architecture Review and Assessment (SARA) Report. Technical Report Version 1.0, SARA W.G. Contributors: P. Kruchten, R. Kazman, A. Ran, H. Obbink, R. Hilliard, T. Mikkonen, W. Kozaczynski, E. Kahan, J. Mueller, L. Dominick, N. Werner, F. Paulisch, W. Tracz, H. Postema, S. Heise.
- [van Heesch *et al.*, 2014] van Heesch, U., Eloranta, V.-P., Avgeriou, P., Koskimies, K., y Harrison, N. (2014). Decision-centric architecture reviews. *IEEE Software*, 31(1):69–76.