





**[AppWeb AhorroLuz]**

**(DAS) Documento Arquitectura Sistema**

**Versión 1.0**



**Identificación de Documento**

| **Identificación** |  |
| --- | --- |
| **Proyecto** | Ahorra Luz AppWeb |
| **Versión** | 1.0 |

| **Documento mantenido por** | Williams Diaz Santander |
| --- | --- |
| **Fecha de última revisión** | 30-09-2025 |
| **Fecha de próxima revisión** | 15-10-2025 |

| **Documento aprobado por** | Alexander Palama |
| --- | --- |
| **Fecha de última aprobación** | 30-09-2025 |

**Historia de Revisiones**

| **Fecha** | **Versión** | **Descripción** | **Autor** |
| --- | --- | --- | --- |
| 30-09-2025 | 1.0 |  | Williams Diaz |
| 15-10-2025 |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Tabla de Contenidos**

[**1**](#_heading=h.gjdgxs) **Introducción 3**

[1.1](#_heading=h.30j0zll) Contexto del Problema 3

[1.2](#_heading=h.1fob9te) Propósito 3

[1.3](#_heading=h.3znysh7) Ámbito 3

[1.4](#_heading=h.2et92p0) Definiciones, acrónimos y abreviaciones 3

[1.5](#_heading=h.tyjcwt) Referencias 3

[1.6](#_heading=h.3dy6vkm) Resumen ejecutivo 3

[1.7](#_heading=h.1t3h5sf) Representación 4

[**2**](#_heading=h.4d34og8) **Metas y Restricciones de la Arquitectura 5**

[2.1](#_heading=h.2s8eyo1) Metas de la arquitectura 5

[2.2](#_heading=h.17dp8vu) Restricciones de la Arquitectura 5

[2.3](#_heading=h.3rdcrjn) Otros antecedentes y consideraciones 5

[**3**](#_heading=h.26in1rg) **Vista de Escenarios 6**

[3.1](#_heading=h.lnxbz9) Modelo de Casos de Uso 6

[3.2](#_heading=h.35nkun2) Casos de Usos Extendidos 6

[3.3](#_heading=h.1ksv4uv) Especificación de los Escenarios de Calidad Relevantes 7

[**4**](#_heading=h.44sinio) **Vista de Procesos 8**

[**5**](#_heading=h.2jxsxqh) **Vista Lógica 9**

[5.1](#_heading=h.z337ya) Parte Estructural ( Diagrama de Clases y Diagrama Relacional) 9

[*5.1.1*](#_heading=h.3j2qqm3) *Descripción de Clases 9*

[*5.1.2*](#_heading=h.1y810tw) *Descripción de Tablas 10*

[5.2](#_heading=h.4i7ojhp) Parte Dinámica (Diagrama de Secuencias) 11

[**6**](#_heading=h.2xcytpi) **Vista de Desarrollo o Despliegue 12**

[**7**](#_heading=h.1ci93xb) **Vista Fisica 13**

[**8**](#_heading=h.3whwml4) **Decisiones de Diseño y Selección de Alternativas 14**

[**9**](#_heading=h.2bn6wsx) **Análisis de Reutilización 15**

1. **Introducción**
2. 1. **Contexto del Problema**

**Alzas repetidas de luz en Chile, todo empieza con el congelamiento de precios de la electricidad activado en 2019, se acumulo una deuda de $6.000.000 usd ,la medida se implementó luego del estallido social ante una prevista alza de las tarifas a pagar, determinada a través de la Ley 21.185, este contexto-país conllevo al alza en las tarifas de los hogares chilenos y la necesidad del ahorro energético**

* 1. **Propósito**

**Definir la arquitectura de la aplicación web AhorraLuz, que permitirá a los hogares chilenos registrar, visualizar y predecir su consumo eléctrico, con métricas claras de ahorro económico y reducción de huella de carbono.**

* 1. **Ámbito**

**El sistema será accesible vía navegador web desde computadores y dispositivos móviles. La versión inicial (MVP v1.0) cubrirá funcionalidades de autenticación, registro de consumo, dashboard con KPIs, historial, predicción de consumo y alertas.**

* 1. **Definiciones, acrónimos y abreviaciones**

| **ACRONIMO** | **DESCRIPCION** |
| --- | --- |
| *RBAC* | Role-Based Access Control |
| *KPI* | Key Performance Indicator |
| *ML* | Machine Learning |
| *CO2eq* | Dióxido de Carbono equivalente |

* 1. **Referencias**

A continuación, se listan las referencias a otros documentos :

* **ERS**
* **Acta de Constitucion de Proyecto**
* **Documento DAS - Plantilla Base**
* **RoadMap**
* **Requerimientos Funcionales / No funcionales** 
  1. **Resumen ejecutivo**

**La arquitectura del sistema AhorraLuz se fundamenta en el enfoque 4+1, asegurando una visión integral que combina perspectivas funcionales y no funcionales. El diseño prioriza atributos clave como seguridad, escalabilidad, rendimiento y facilidad de uso, garantizando una solución confiable y adaptable a las necesidades de los hogares.**

**La implementación se basará en Django (Python) como framework principal, con PostgreSQL como motor de base de datos relacional. La interfaz web se desarrollará con tecnologías estándar (HTML5, CSS3, JavaScript y Bootstrap) para asegurar una experiencia de usuario responsiva e intuitiva.**

**El proyecto seguirá la metodología Agile/Scrum, favoreciendo la iteración continua y la entrega incremental de valor. Asimismo, se contempla la integración futura con herramientas de Inteligencia de Negocios (Power BI) y procesos de ETL, lo que permitirá enriquecer el análisis de consumo y habilitar capacidades de reporting avanzado.**

* 1. **Representación**

La arquitectura del sistema AppWeb AhorroLuz está representada siguiendo el enfoque de del framework 4+1 y las recomendaciones del proceso unificado. Las vistas incluidas en esta versión del documento son:

* **Vista de Escenarios**: Describe los casos de uso más significativos, presenta los actores y una descripción de sus casos de uso asociados. De igual forma describe los escenarios de calidad más relevantes para la arquitectura.
* **Vista de Procesos**: Describe los procesos involucrados para darle sentido a la ejecución del sistema, así como sus relaciones de comunicación y sincronización.
* **Vista Lógica**: Describe la arquitectura del sistema presentando varios niveles de refinamiento. Indica los módulos lógicos principales, sus responsabilidades y dependencias.
* **Vista de Desarrollo o Despliegue**: Describe los componentes de deployment construidos y sus dependencias.
* **Vista Fisica**: Describe restricciones tecnológicas, normativas, estándares, etc., los cuales influyen sobre las decisiones arquitectónicas, del producto y del proceso de desarrollo.

1. **Metas y Restricciones de la Arquitectura**

A continuación, se revisan las metas y restricciones de la arquitectura.

* 1. **Metas de la arquitectura**

De acuerdo a las reuniones y al análisis de los requerimientos, se listan los principales conductores iniciales de la arquitectura los cuales corresponden a las metas arquitectónicas iniciales ( atributos de calidad)

* Seguridad: Protección contra XSS, CSRF, SQL Injection; cifrado de contraseñas; validación de entradas.
* Desempeño: Tiempo de respuesta < 3s en login; carga de dashboard < 5s.
* Escalabilidad: Integración futura con ETL/BI y servicios cloud.
* Disponibilidad: 99% de uptime en entornos de prueba.
* Usabilidad: Interfaz responsiva e intuitiva (HTML5 + Bootstrap).
* Mantenibilidad: Modularización con Django Apps, control de versiones con GitHub.
  1. **Restricciones de la Arquitectura**

Existen restricciones que han sido levantadas con los stakeholders, las cuales se presentan a continuación:

* **Tiempo de construcción: 15 semanas (según planificación).**
* **Presupuesto máximo: $66.208.000 CLP (teórico).**
* **Infraestructura: despliegue en Render o servidor similar con PostgreSQL.**
* **Repositorio: público en GitHub**
  1. **Otros antecedentes y consideraciones**

La empresa desarrolladora cuenta con un framework que considera los siguientes componentes que permiten satisfacer los requerimientos arquitectónicos:

* Se empleará framework Django (backend) y Bootstrap (frontend). Librerías de ML: Scikit-learn, Pandas, NumPy.

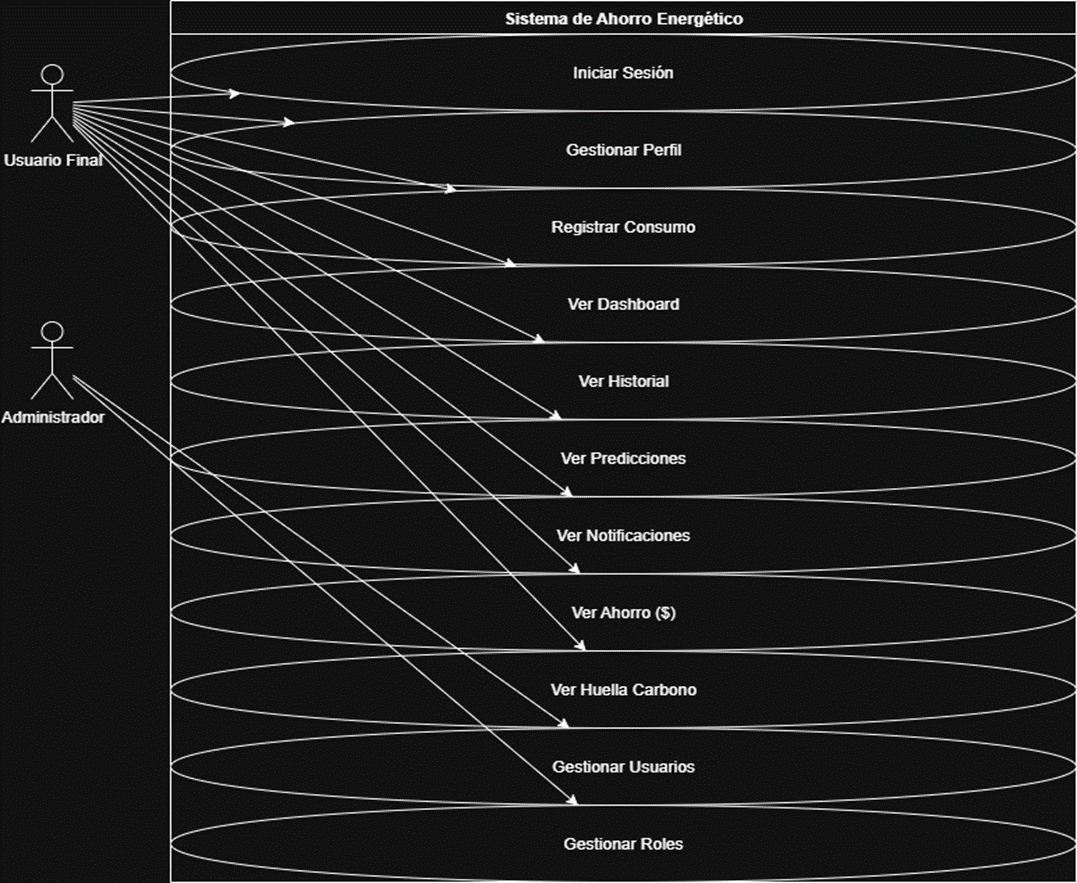
1. **Vista de Escenarios**

Esta sección describe en detalle el conjunto de escenarios funcionales y no funcionales que obtuvieron la mayor prioridad en el análisis. Para esto se presenta y describe el diagrama de casos de uso y los casos de uso prioritarios, así como los escenarios en que uno o más atributos de calidad se ven involucrados de manera significativa.

* 1. **Modelo de Casos de Uso**

Agregar el modelo de caso uso general del sistema

**Ilustración 1: Diagrama de Caso Uso General del Sistema**



* 1. **Casos de Usos Extendidos**

Los casos de uso considerados son los más relevantes para el desarrollo de la arquitectura. Se adjunta el documento o planilla caso uso.

**Adjuntar la planilla caso uso extendido**

A continuación, se listan los casos de uso relevantes, los cuales pueden ser encontrados con su especificación detallada en el documento “Casos de Uso Extendido”.

| **Código** | **Nombre** | **Actores** | **Prioridad** |
| --- | --- | --- | --- |
| CU-001 | Registrar Usuario | Administradores | Alta |
| CU-002 | Autenticar Usuario | Todos los usuarios | Alta |
| CU-003 | Gestionar Perifl | Todo los usuarios reg | Media |
| CU-004 | Registrar consumo eléctrico | Usuarios | Media |
| CU-005 | Visualizar Dashboard | Usuario Final | Alta |
| CU-006 | Consultar Historial de consumo/predicciones | Usuario Final | Alta |
| CU-007 | Prediccion de Consumo | Usuario Final | Alta |
| CU-008 | Gestionar Alertas | Sistema | Alta |
| CU.009 | Calcular Huella de Carbono | Sistema | Alta |
| CU-010 | Administración de Usuarios | Administradores | Alta |
| CU-011 | Gestión de Roles y Permisos | Administradores | Alta |

* 1. **Especificación de los Escenarios de Calidad Relevantes**

Después de un análisis en conjunto con los stakeholders, los escenarios de calidad se expresan a continuación:

| Identificador: EQ-01 | | |
| --- | --- | --- |
| Escenario(s): | | Intento de inyección SQL en el formulario de login |
| Atributos de Calidad relevantes: | | Seguridad |
| Componentes del Escenario | Estímulos: | El usuario malicioso ingresa credenciales manipuladas con código SQL. |
| Fuente del estimulo | Actor externo (atacante) |
| Ambiente: | Sistema en producción (Render.com). |
| Artefacto: | Módulo de autenticación (Django Auth + PostgreSQL). |
| Respuesta: | El sistema rechaza la solicitud, no ejecuta código malicioso, registra el incidente en logs. |
| Medida de Respuesta | Ninguna alteración de datos, respuesta HTTP 400 o 403 en < 1 segundo |

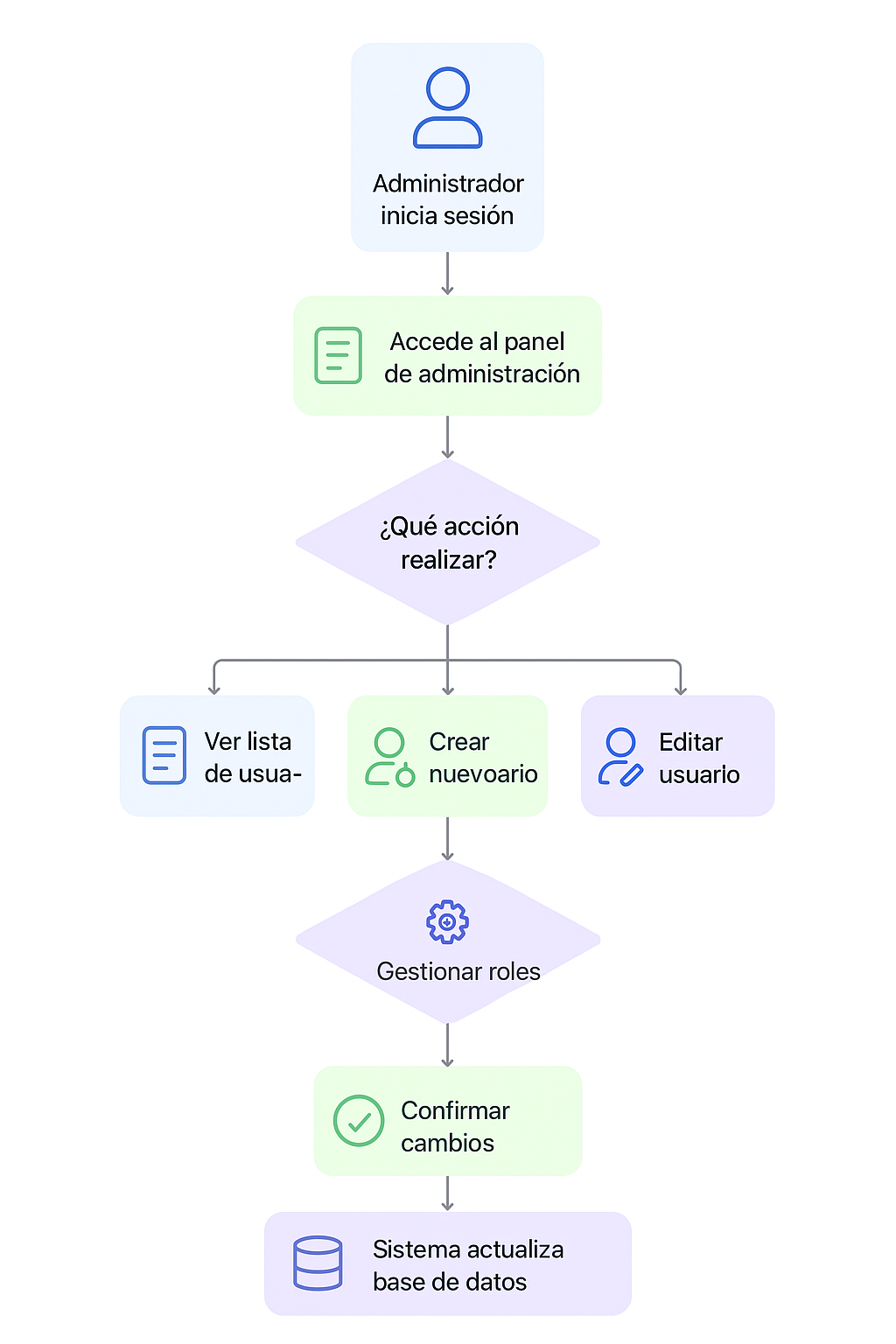
| Identificador: EQ-02 | | |
| --- | --- | --- |
| Escenario(s): | | Ninguna alteración de datos, respuesta HTTP 400 o 403 en < 1 segundo |
| Atributos de Calidad relevantes: | | Rendimiento, Usabilidad |
| Componentes del Escenario | Estímulos: | Solicitud de carga de dashboard después de login exitoso |
| Fuente del estimulo | Usuario Final. |
| Ambiente: | Navegador web moderno con conexión estable |
| Artefacto: | Backend Django + consultas PostgreSQL + librerías gráficas. |
| Respuesta: | El sistema consulta datos, calcula KPIs, genera gráficos y los muestra en interfaz web |
| Medida de Respuesta | Dashboard completo cargado en < 5 segundos |

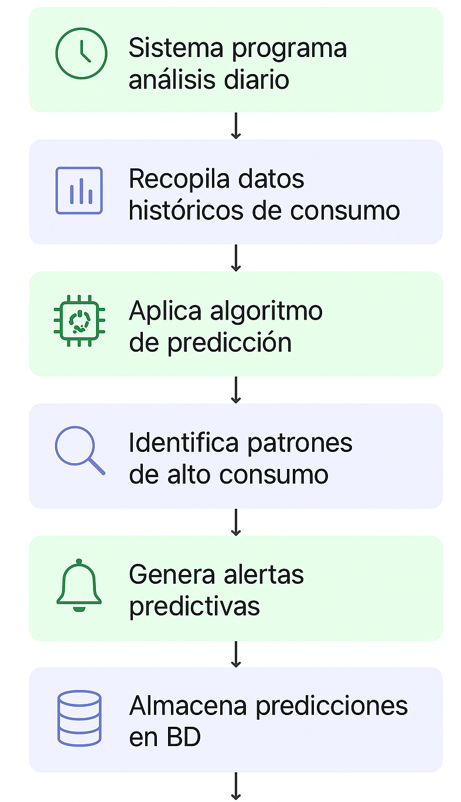
| Identificador: EQ-03 | | |
| --- | --- | --- |
| Escenario(s): | | Falla temporal en la base de datos. |
| Atributos de Calidad relevantes: | | Disponibilidad, Tolerancia a fallos |
| Componentes del Escenario | Estímulos: | La base de datos deja de responder durante una transacción. |
| Fuente del estimulo | Infraestructura (PostgreSQL) |
| Ambiente: | Entorno de producción en Render. |
| Artefacto: | Módulo de persistencia de Django (ORM) |
| Respuesta: | El sistema muestra un mensaje de error controlado al usuario, reintenta conexión en segundo plano. |
| Medida de Respuesta | Recuperación automática en < 30 segundos; no pérdida de datos críticos. |

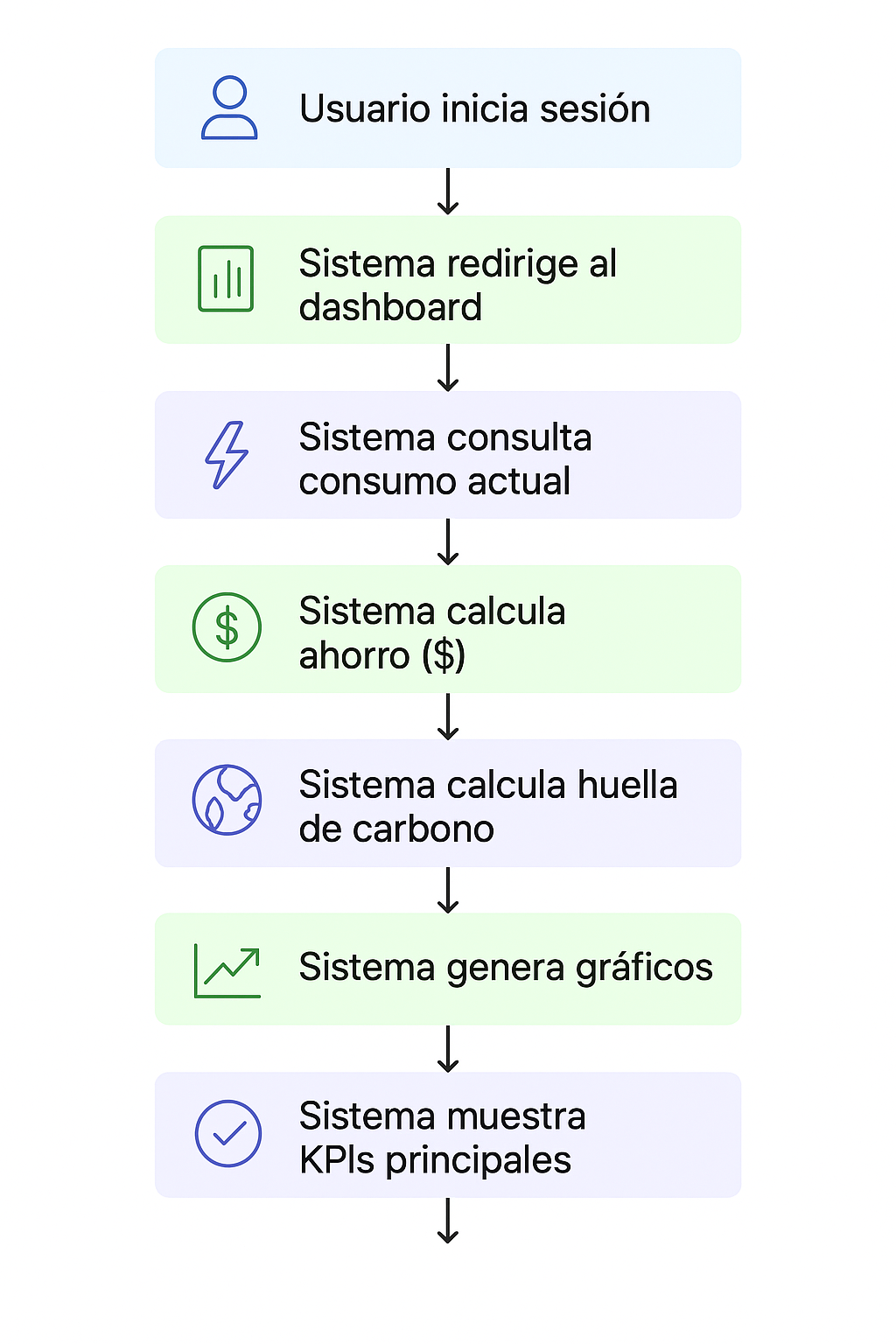
| Identificador: EQ-04 | | |
| --- | --- | --- |
| Escenario(s): | | 1.000 usuarios concurrentes acceden al sistema en hora punta |
| Atributos de Calidad relevantes: | | Escalabilidad, Desempeño |
| Componentes del Escenario | Estímulos: | Acceso simultáneo de múltiples usuarios autenticados |
| Fuente del estimulo | Usuarios Finales |
| Ambiente: | Producción (Render.com, PostgreSQL administrado) |
| Artefacto: | Servidor de aplicación (Django + Gunicorn) |
| Respuesta: | El sistema mantiene un servicio estable, balancea carga y procesa solicitudes concurrentes sin caídas. |
| Medida de Respuesta | Tiempo de respuesta promedio < 3 segundos para el 95% de las solicitudes. |

1. **Vista de Procesos**

**Ilustración 2: Diagramas de Actividades**







1. **Vista Lógica**

A continuación, se presenta una vista lógica de la aplicación expresado en tres diagramas, uno de ellos que muestra la parte estructural o estática de la aplicación (clases) y a la base de datos (modelo relacional).otra vista que representa la parte dinámica (secuencias).

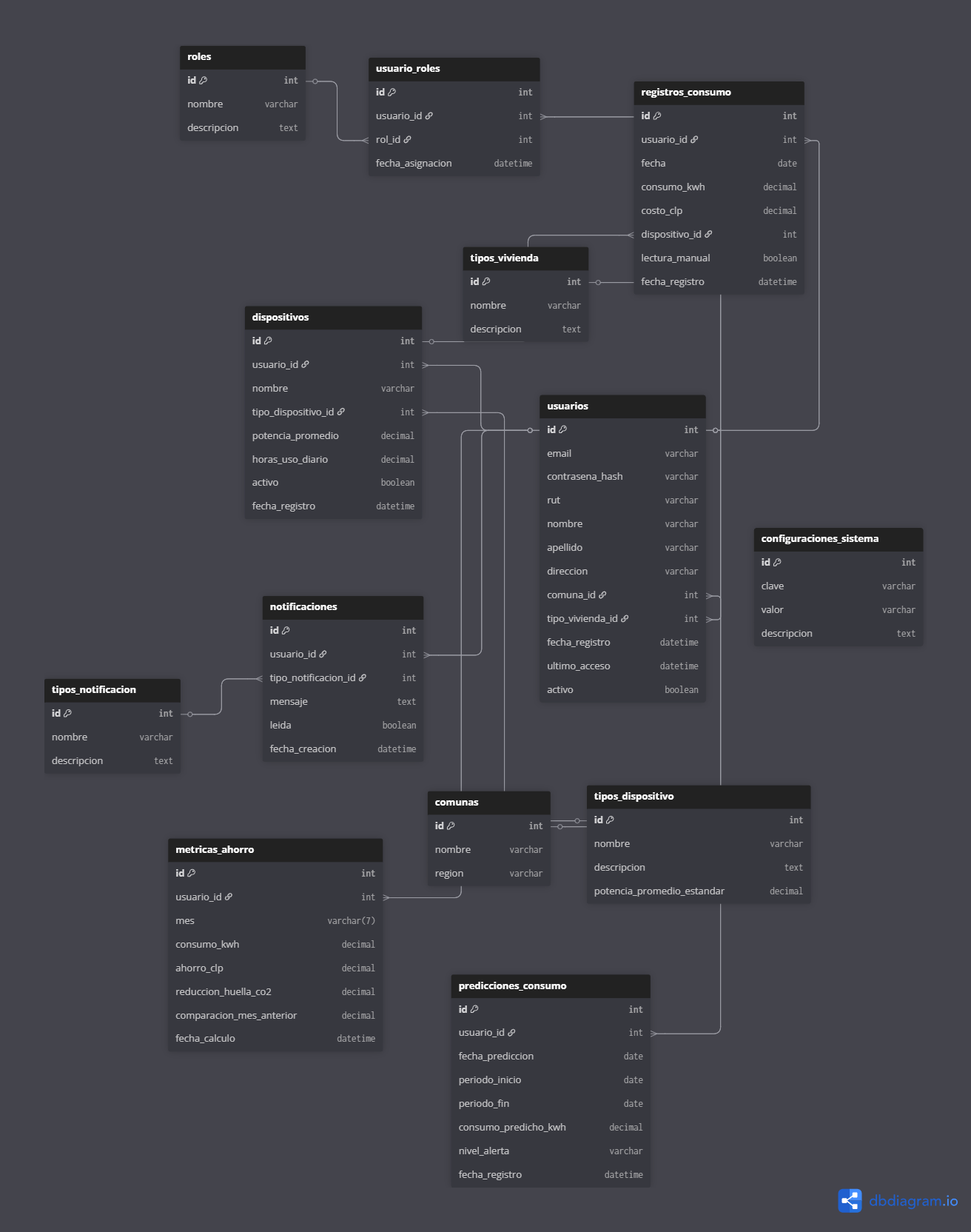
* 1. **Parte Estructural ( Diagrama de Clases y Diagrama Relacional)**

**Ilustración 3: Diagrama de Clases**

* + 1. *Descripción de Clases*

| **Código** | **Nombre** | **Descripción** |
| --- | --- | --- |
| CL-001 | Usuarios | atributos: rut, email, contraseña, rol |
| CL-002 | Perfil Hogar | dirección, tipo vivienda, habitantes, dispositivos |
| CL-003 | Consumo | fecha, kWh, costo, observaciones |
| CL-004 | Alerta | tipo, mensaje, fecha |
| CL-005 | Predicción | periodo, valor estimado, nivel confianza |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Ilustración 4: Diagrama de Base Datos (Relacional)**

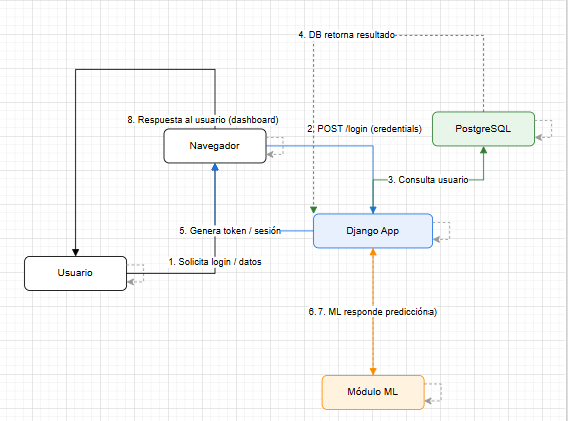


* + 1. *Descripción de Tablas*

| **Código** | **Nombre** | **Descripción** |
| --- | --- | --- |
| TB-001 | USUARIOS | Almacena la información de los usuarios del sistema de monitoreo energético |
| TB-002 | DISPOSITIVOS | Registra los dispositivos eléctricos asociados a cada usuario |
| TB-003 | REGISTROS\_CONSUMO | Contiene los registros históricos de consumo energético de los usuarios |
| TB-004 | PREDICCIONES CONSUMO | Almacena las predicciones de consumo energético futuro por usuario |
| TB-005 | NOTIFICACIONES | Gestiona las notificaciones y alertas enviadas a los usuarios |
| TB-006 | METRICAS\_AHORRO | Registra las métricas de ahorro y eficiencia energética mensual por usuario |
| TB-001 | USUARIOS | Almacena la información de los usuarios del sistema de monitoreo energético |
| TB-002 | DISPOSITIVOS | Registra los dispositivos eléctricos asociados a cada usuario |
| TB-003 | REGISTROS\_CONSUMO | Contiene los registros históricos de consumo energético de los usuarios |
| TB-004 | PREDICCIONES CONSUMO | Almacena las predicciones de consumo energético futuro por usuario |
| TB-005 | NOTIFICACIONES | Gestiona las notificaciones y alertas enviadas a los usuarios |
| TB-006 | METRICAS\_AHORRO | Registra las métricas de ahorro y eficiencia energética mensual por usuario |
| TB-001 | USUARIOS | Almacena la información de los usuarios del sistema de monitoreo energético |
| TB-002 | DISPOSITIVOS | Registra los dispositivos eléctricos asociados a cada usuario |
| TB-003 | REGISTROS\_CONSUMO | Contiene los registros históricos de consumo energético de los usuarios |

* 1. **Parte Dinámica (Diagrama de Secuencias)**

**Ilustración 5: Diagramas de Secuencias**



Este diagrama de secuencia ilustra el flujo de interacciones clave dentro de una aplicación web, mostrando cómo un usuario se autentica y, en un caso particular, cómo se invoca un módulo de Machine Learning (ML) para generar una predicción. Los componentes principales son el Usuario, el Navegador, la aplicación Django App, un Módulo ML y la base de datos PostgreSQL.

**Descripción del Proceso**

El flujo comienza con el usuario solicitando acceso o datos a través de su navegador.

El Usuario interactúa con la interfaz web del Navegador, solicitando un inicio de sesión o ingresando datos.

El Navegador envía una solicitud POST con las credenciales al servidor, en este caso, a la Django App.

La Django App recibe la solicitud y consulta la base de datos PostgreSQL para verificar la existencia y la validez del usuario.

PostgreSQL procesa la consulta y retorna la información del usuario a la Django App.

Si las credenciales son válidas, la Django App genera un token o una sesión y la envía de vuelta al Navegador para autenticar al usuario.

En un escenario específico, como al solicitar una predicción, la Django App invoca al Módulo ML, pasando los datos necesarios para el análisis.

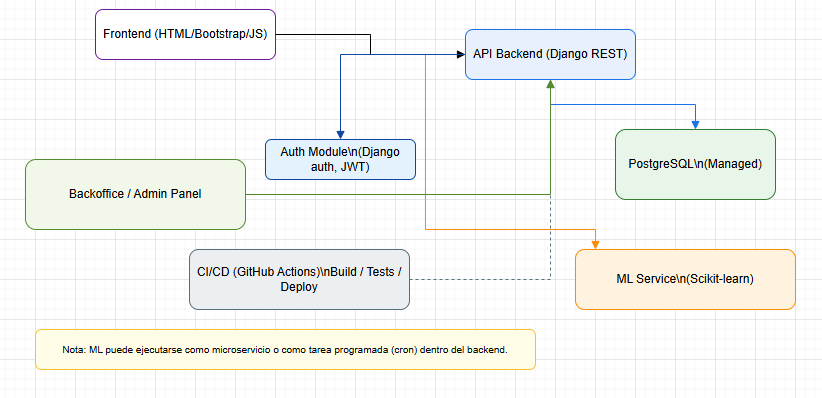
El Módulo ML procesa la solicitud y retorna el resultado de la predicción a la Django App.

Finalmente, el Navegador recibe la respuesta completa y la presenta al Usuario, mostrando el dashboard o la página con la predicción, completando así el ciclo de interacción.

1. **Vista de Desarrollo o Despliegue**

En esta vista se describen las componente o modulos en las cuales se dividirá o implementará el sistema

**Ilustración 6: Diagrama de componentes**

****

Este diagrama de componentes presenta una arquitectura modular para una aplicación web que integra un modelo de Machine Learning (ML). La estructura separa las responsabilidades en distintos bloques funcionales, lo que mejora la escalabilidad, el mantenimiento y la organización del proyecto.

Componentes y sus Interacciones

Frontend (HTML/Bootstrap/JS): Es la capa de presentación que el usuario final ve y con la que interactúa. Se encarga de la interfaz de usuario y la lógica del lado del cliente.

API Backend (Django REST): Es el cerebro de la aplicación. Gestiona la lógica de negocio, procesa las solicitudes del frontend y se comunica con los demás servicios. Construido con Django, expone los endpoints que permiten la comunicación.

Auth Module (Django auth, JWT): Un módulo de autenticación especializado que maneja el registro, el login y la gestión de sesiones de los usuarios. Utiliza JSON Web Tokens (JWT) para una autenticación segura y sin estado, lo cual es ideal para APIs.

ML Service (Scikit-learn): Este es un servicio o módulo dedicado que contiene el modelo de machine learning (probablemente desarrollado con Scikit-learn). El Backend lo llama para realizar predicciones basándose en los datos recibidos.

PostgreSQL (Managed): Es la base de datos relacional del sistema. Al ser gestionada, su mantenimiento, copias de seguridad y escalabilidad son manejados por un proveedor de servicios, lo que permite al equipo de desarrollo centrarse en la aplicación.

Backoffice / Admin Panel: Una interfaz administrativa separada, utilizada por los equipos internos para gestionar datos, usuarios, o para realizar tareas de supervisión y mantenimiento del sistema.

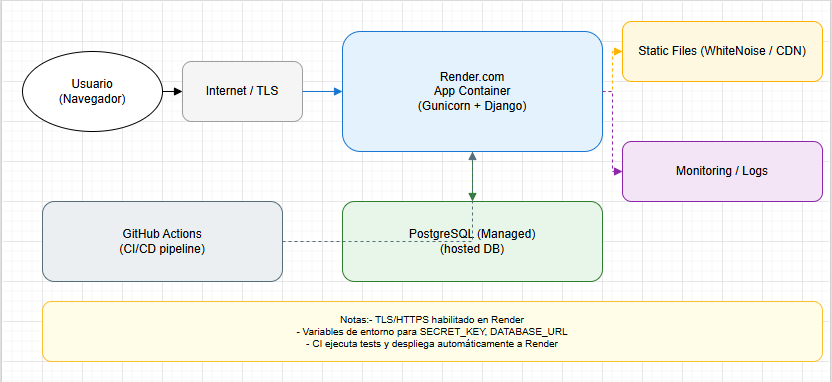
CI/CD (GitHub Actions): Un sistema de Integración Continua y Despliegue Continuo. Automatiza el proceso de construcción, prueba y despliegue del código cada vez que se realizan cambios, garantizando que el sistema esté siempre actualizado y funcionando correctamente.

El flujo principal es iniciado por el Frontend que se comunica con la API Backend. El Backend, a su vez, delega tareas específicas a los módulos de autenticación y ML, mientras que utiliza la base de datos PostgreSQL para persistir los datos. El sistema de CI/CD automatiza el proceso de actualización de la API Backend, y el Backoffice proporciona una forma de gestión interna del sistema.

1. **Vista Fisica**

En esta vista se despliegan los nodos que participan con el sistema.

**Ilustración 7: Diagrama de Despliegue**



Este diagrama de despliegue detalla la arquitectura de la aplicación en el entorno de producción, mostrando cómo los diferentes componentes de la aplicación se mapean a nodos físicos o lógicos y cómo interactúan entre sí. Esta vista de alto nivel es esencial para comprender la infraestructura subyacente que soporta la aplicación.

Componentes de la Arquitectura de Despliegue

* **Usuario (Navegador)**: Representa al usuario final que accede a la aplicación a través de su navegador web.

* **Internet / TLS:** Actúa como el medio de transporte para las solicitudes del usuario. La comunicación se realiza a través del protocolo TLS (Transport Layer Security) para garantizar que las peticiones y respuestas estén cifradas y sean seguras.

* **Render.com (Gunicorn + Django)**: Este es el contenedor principal que aloja la aplicación web. Render.com es la plataforma de despliegue donde se ejecuta la aplicación, usando Gunicorn como servidor WSGI para manejar las peticiones web y Django como el framework de la aplicación.

* **PostgreSQL (Managed)**: Representa la base de datos de la aplicación. El uso de una base de datos gestionada por un proveedor externo simplifica las tareas de administración, como la escalabilidad, las copias de seguridad y la seguridad, sin necesidad de que el equipo de desarrollo las gestione directamente.

* **Static Files (WhiteNoise / CDN)**: Los archivos estáticos, como CSS, JavaScript e imágenes, son servidos de manera eficiente. WhiteNoise es una librería que permite a Django servir sus propios archivos estáticos, mientras que el uso de un CDN (Content Delivery Network) optimiza la entrega de estos archivos al usuario, reduciendo la latencia.

* **GitHub Actions (CI/CD pipeline)**: Se trata de un sistema de Integración Continua y Despliegue Continuo. Cada vez que se actualiza el código en el repositorio de GitHub, GitHub Actions ejecuta automáticamente una serie de pasos, como pruebas, y luego despliega la nueva versión de la aplicación en Render.com.

* **Monitoring / Logs**: Este componente es crucial para la observabilidad del sistema. Permite a los desarrolladores y administradores supervisar el rendimiento de la aplicación, rastrear errores, y auditar eventos, lo que es vital para la resolución de problemas y el mantenimiento.

**Flujo de Despliegue y Operación**

El flujo de operación comienza con el Usuario enviando una petición que viaja de forma segura por Internet y llega al contenedor de la aplicación en Render.com. El contenedor procesa la solicitud, interactuando con la base de datos PostgreSQL para obtener o guardar información. Los archivos estáticos se cargan directamente desde un CDN para mejorar la velocidad.

Para el despliegue, el pipeline de CI/CD en GitHub Actions se encarga de automatizar el proceso de subir el código a Render, mientras que un sistema de monitoreo se mantiene activo para registrar la salud y los logs de la aplicación en todo momento.

1. **Decisiones de Diseño y Selección de Alternativas**

Las principales decisiones arquitectónicas se tomaron en consideración de la restricción Tiempo de Construcción. Dado que el proyecto debe implementarse en un tiempo ajustado y sin holguras, se privilegió la adopción de una arquitectura conocida y que presente un bajo riesgo en su implementación.

Asimismo, la arquitectura se modularizó y cada módulo serán testeables unitariamente automatizadamente con pipeline GitHub Actions de forma de asegurar que cada pieza tenga una baja tasa de fallas.

Un segundo proposito que fue considerado en la selección de alternativa son :

Uso de Django por rapidez de desarrollo y seguridad integrada.

Uso de PostgreSQL por compatibilidad con Django y soporte a operaciones complejas.

Modularización en apps Django para separar autenticación, perfil, consumo, reportes y administración.

Integración futura con Power BI para reporting avanzado.

1. **Análisis de Reutilización**

* Se utilizará el código del mantenedor de productos en clientes
* Se utilizarán plantillas gráficas con los mismos colores y logotipos en el mantenedor de clientes, productos, ventas.
* Uso de librerías estándar Django para autenticación.
* Uso de librerías de Python (Scikit-learn, Pandas, NumPy) para ML y análisis.
* Plantillas Bootstrap reutilizadas para UI.
* Configuración de logging y validación en settings.py será estándar y replicable.