WalletMoneyFamily: Aplicación web para control de gastos familiares

Tabla de contenido

Contenido **Pág.**

Tabla de Tablas

Tablas **Pág.**

Lista de Figuras

Figuras **Pág.**

Introducción

El proyecto WalletMoneyFamily responde a una necesidad crítica identificada en la familia Aldana Gómez: la falta de un sistema estructurado y centralizado para la gestión de las finanzas del hogar. Actualmente, la familia utiliza métodos informales y desorganizados, como anotaciones dispersas en cuadernos, hojas de cálculo sin formato definido o, en muchos casos, la ausencia total de registros. Esta situación genera dificultades para rastrear los ingresos y gastos, lo que frecuentemente resulta en desequilibrios financieros, donde los gastos superan los ingresos disponibles, limitando la planificación económica y causando

tensiones en la toma de decisiones familiares. Este problema es común en muchos hogares que carecen de herramientas centralizadas para la gestión financiera. Para abordar esta necesidad, WalletMoneyFamily es una aplicación web diseñada para permitir a las familias registrar, monitorear y analizar sus ingresos y gastos de manera sistemática. Desarrollada por un desarrollador junior en un plazo de 6 a 8 semanas, con un costo para el cliente de aproximadamente 1859 USD (113 horas a 10-20 USD/hora, con 10% de contingencia), basado en tarifas junior y un presupuesto del desarrollador cercano de 25 USD utilizando herramientas gratuitas y de bajo costo, la aplicación emplea Flask para el backend, PostgreSQL para el almacenamiento de datos, Bootstrap para una interfaz responsiva y Chart.js para reportes visuales. Gestionado mediante un tablero Kanban en Trello con las columnas To Do, In Progress y Done, el proyecto asegura un desarrollo ágil alineado con las fases del Ciclo de Vida del Desarrollo de Software (SDLC). La aplicación consta de dos módulos principales: autenticación (registro e inicio de sesión con Flask-Login) y registro/seguimiento de ingresos y gastos (visualización de datos financieros por miembro de la familia). Al ofrecer una plataforma segura, accesible y fácil de usar, WalletMoneyFamily busca prevenir desequilibrios financieros, reducir conflictos y mejorar la toma de decisiones informadas, contribuyendo a la estabilidad financiera y la calidad de vida de la familia Aldana Gómez.

Contextualización de la necesidad

En la familia Aldana Gómez se evidencia una falta de registro y control estructurado de sus gastos. Actualmente, el manejo de las finanzas se realiza de forma informal y desorganizada, ya sea mediante anotaciones aisladas en cuadernos, registros en hojas de cálculo sin un formato definido o, en muchos casos, sin ningún tipo de registro. Esta situación genera dificultades para llevar un seguimiento claro de los ingresos y egresos del hogar, lo que con frecuencia provoca que los gastos superen los ingresos disponibles, generando

desequilibrios financieros, limitaciones en la planeación económica, y tensiones en la toma de decisiones dentro del núcleo familiar. Este problema es común en muchas familias, donde la falta de un sistema centralizado para monitorear ingresos y gastos lleva a decisiones poco informadas. De esta necesidad surge WalletMoneyFamily, una aplicación web diseñada para que las familias puedan registrar, monitorear, y analizar sus ingresos y gastos de forma estructurada. WalletMoneyFamily permitirá a los usuarios registrar ingresos y gastos asociados a cada miembro y visualizar tendencias financieras mediante tablas y gráficos, todo desde una interfaz web accesible y segura, contribuyendo al equilibrio financiero del hogar y mejorando la calidad de vida del todo el núcleo familiar.

Planteamiento del problema

¿Cómo una aplicación web puede organizar y centralizar el registro y análisis de los ingresos y gastos familiares, permitiendo identificar quién genera cada ingreso y quién realiza cada gasto, con el propósito de prevenir desequilibrios financieros, reducir conflictos y mejorar la toma de decisiones, fomentando así un equilibrio económico en el hogar?

Fase de planificación del ciclo de vida del desarrollo de software (Fase de Análisis de Proyectos)

Alcance

El proyecto WalletMoneyFamily consiste en desarrollar una aplicación web sencilla, segura y accesible desde cualquier navegador para que la familia Aldana Gómez organice, centralice y analice sus ingresos y gastos, identificando quién genera cada ingreso y quién realiza cada gasto, con el fin de evitar desequilibrios financieros, conflictos familiares y decisiones poco informadas, promoviendo un equilibrio financiero en el hogar. La aplicación será desarrollada por un desarrollador junior individual en 6-8 semanas, con un presupuesto real para el desarrollador de aproximadamente 25 USD (usando herramientas gratuitas y de

bajo costo) y un costo para el cliente de aproximadamente 1859 USD (113 horas a 10-20 USD/hora, con 10% de contingencia), basado en tarifas junior. Se utilizará la metodología ágil Kanban, gestionada en un tablero Trello con columnas To Do, In Progress y Done, actualizado diariamente para el informe del proyecto. El entorno de desarrollo será Visual Studio Code con extensiones como Python, Pylance, GitLens y Live Server para codificar, depurar y gestionar Flask, HTML/CSS y JavaScript. Las tecnologías incluyen Flask para el backend, PostgreSQL para la base de datos (tablas: usuarios, ingresos, gastos), Bootstrap para una interfaz responsiva, y Chart.js para gráficos de barras y pastel. El control de versiones se gestionará en un repositorio GitHub con rama main y ramas por funcionalidad, y la aplicación se desplegará en Render con integración CI/CD. El alcance abarca 2 módulos principales; el módulo 1 es para la autenticación (registro e inicio de sesión con Flask-Login con usuario/correo y contraseña), y el módulo 2 es para registro de ingresos (monto, categoría, titulo, descripción, fecha) y gastos (monto, categoría, titulo, descripción, fecha), y visualización mediante tablas y gráficos de tendencias financieras. Los entregables incluyen la aplicación desplegada, formularios funcionales, reportes visuales, un repositorio GitHub con README.md, un tablero Trello aplicando la metodología ágil Kanban, un informe de proyecto de software en formato APA (PDF) que detalle contextualización, planteamiento del problema, objetivos (general: mejorar la gestión financiera; específicos: implementar autenticación, registro, reportes, despliegue, y documentación), alcance, metodología, matriz de riesgos, mapa de stakeholders (gráfico de dispersión), requerimientos funcionales (RQF1: autenticación, RQF2: registro, RQF3: reportes), requerimientos no funcionales (RQNF1: respuesta <2s, RQNF2: interfaz responsiva, RQNF3: uptime >99%, RQNF4: contraseñas con hash), historias de usuario (HU1-HU3), diagrama de flujo (Draw.io), arquitectura (UML), prototipos de interfaz (Figma/papel), resultados de pruebas, y evidencias de despliegue. Las pruebas incluyen unitarias, integrales, E2E y de aceptación (pytest y Selenium para lógica del

correcto funcionamiento de la autenticación, registros, eliminación y cierre de sesión) y funcionales y no funcionales (para validar el correcto funcionamiento en autenticación, registro, visualización, rendimiento y seguridad, posterior al despliegue de la aplicación en Render). El proyecto no incluye integraciones con bancos, notificaciones en tiempo real, pruebas de carga extensivas ni aplicaciones móviles, debido a limitaciones de tiempo, presupuestales y técnicas. Las restricciones son un presupuesto real para el desarrollador junior de aproximadamente 25 USD, un desarrollador junior con 10-15 horas semanales, hardware estándar (CPU 2 GHz, 8 GB RAM, 256 GB almacenamiento), y pruebas en entornos local (PostgreSQL) y Render. Los criterios de aceptación incluyen una aplicación funcional con los módulos mencionados, interfaz responsiva, tiempo de respuesta <=2s, uptime >99%, informe APA completo, y tablero Trello actualizado. Un formulario en Google Forms se usará para el levantamiento de información con la familia Aldana Gómez.

Estructura del Desglose del Trabajo (EDT)

Planeación

- **1.1.** Redactar contextualización de la necesidad
- **1.2.** Definir planteamiento del problema
- **1.3.** Establecer objetivo general y específicos
- **1.4.** Redactar alcance, restricciones y criterios de aceptación
- **1.5.** Definir metodología ágil Kanban y configurar tablero Trello
- **1.6.** Crear matriz de riesgos
- **1.7.** Estimar presupuesto

Análisis

- **2.1.** Realizar levantamiento de información (herramientas y conclusiones)
- **2.2.** Mapear stakeholders y clasificarlos
- **2.3.** Definir requerimientos funcionales (RQF) y no funcionales (RQNF)

- **2.4.** Redactar historias de usuario
- **2.5.** Diseñar diagrama de flujo de solución
- **2.6.** Configurar repositorio GitHub (rama main, ramas por funcionalidad)

Diseño

- **3.1.** Diseñar arquitectura del sistema (backend, frontend, base de datos)
- **3.2.** Diseñar base de datos (tablas en PostgreSQL)
- **3.3.** Diseñar interfaz de usuario (wireframes en Figma o papel)
- **3.4.** Modelar sistema (diagramas UML)

Desarrollo

- **4.1.** Configurar entorno en Visual Studio Code (extensiones Python, Pylance, GitLens, Live Server)
- **4.2.** Desarrollar módulo de autenticación (Flask-Login)
- **4.3.** Desarrollar módulo de registro de ingresos y gastos y de visualización (tablas y gráficos con Chart.js)

Pruebas

- **5.1.** Realizar pruebas unitarias, de integración (pytest) y E2E (selenium) y de aceptación
- **5.2.** Realizar pruebas funcionales y no funcióneles (manuales, Selenium, Lighthouse)
- **5.4.** Documentar resultados de pruebas

Despliegue

- **6.1.** Configurar despliegue en Render (integración con GitHub)
- **6.2.** Verificar accesibilidad y seguridad de la aplicación desplegada

Documentación

- **7.1.** Redactar informe académico en PDF (incluyendo todos los entregables)

- **7.2.** Actualizar README.md en GitHub
- **7.3.** Capturar evidencias de Trello y pruebas

Objetivos

General

Diseñar e implementar una aplicación web que centralice el registro, monitoreo y análisis de los ingresos y gastos familiares, identificando a los responsables y ofreciendo herramientas de visualización para prevenir desequilibrios financieros y apoyar una toma de decisiones informada en el hogar.

Específicos

- Analizar la situación actual de manejo de gastos familiares y definir los requerimientos funcionales y técnicos de la aplicación web.
- Diseñar la arquitectura, la base de datos y la interfaz de usuario de la aplicación web, asegurando claridad, usabilidad y seguridad.
- Desarrollar e integrar módulos principales del sistema, el acceso al integrante de la familia, registro de ingresos, gastos y reportes gráficos de patrones de consumo.
- Validar y desplegar la aplicación en Render mediante pruebas unitarias, integrales, E2E, de aceptación, de funcionamiento y no funcionamiento, documentando el proceso y garantizando una buena accesibilidad en el entorno web.

Metodología ágil seleccionada

El proyecto WalletMoneyFamily implementará la metodología ágil Kanban en Trello para gestionar el desarrollo de una aplicación web que permita a una familia organizar y centralizar el registro, monitoreo y análisis de sus ingresos y gastos, identificando quién genera cada ingreso y quién realiza cada gasto. El tablero Trello organiza las tareas en tres listas principales: To Do, In Progress, y Done, alineadas con las fases del SDLC (Planeación,

Análisis, Diseño, Desarrollo, Pruebas, Despliegue, Documentación). Cada tarjeta representa una tarea específica, con descripción, responsable (estudiante), periodo estimado. La metodología ágil Kanban en Trello permite priorizar tareas, visualizar el progreso, y ajustar el flujo, garantizando la entrega en 6-8 semanas. Enlace del tablero Trello: **https://trello.com/b/hUP9ODRf/walletmoneyfamily** .

Justificación del Alcance

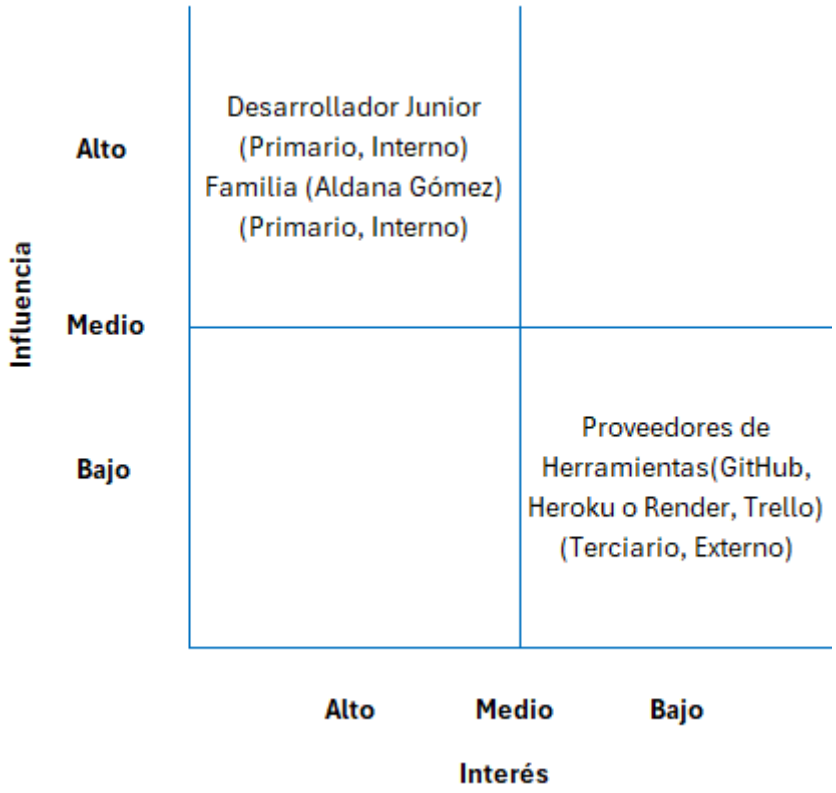
Corto Plazo (MVP, 6-8 semanas): El alcance se centra en un prototipo funcional con módulos básicos de autenticación, registro de ingresos/gastos, y visualización de reportes, resolviendo el problema principal de desorganización financiera en la familia Aldana Gómez. Esta solución inicial permite registrar y analizar datos para evitar desequilibrios inmediatos, respondiendo a stakeholders.

Medio Plazo (3-6 meses post-desarrollo): Se podría expandir con funcionalidades como exportación de reportes a PDF o filtros avanzados para patrones de gasto, mejorando la toma de decisiones informada. Esto respondería a stakeholders al proporcionar mayor transparencia, pero se limita al MVP actual por restricciones de tiempo y presupuesto.

Largo Plazo (6-12 meses): Se podría integrar con apps de productividad o agregar usuarios múltiples para familias extendidas, promoviendo escalabilidad. Esta descripción de soluciones aborda el problema de la gestión desorganizada, ofreciendo a stakeholders (familia, estudiante, profesores) una herramienta que centraliza finanzas reduce conflictos, y facilita decisiones informadas, contribuyendo a un equilibrio financiero sostenido.

Figura 1

Mapa de Stakeholders (Influencia vs Interés)



Fuente: Elaboración propia**.**

Tabla 1

Matriz de Riesgos

Riesgo	Probabilidad	Impacto	Mitigación
Retrasos por imprevistos (e.g., problemas técnicos)	Media	Alto	Hitos semanales en Kanban, en Trello para mantener foco
Errores en lógica de datos (e.g., guardado de ingresos)	Media	Alto	Pruebas unitarias e integrales, E2E y de aceptación exhaustivas
Problemas en despliegue (e.g., configuración en Render)	Baja	Medio	Pruebas previas locales y CI/CD con GitHub
Limitaciones de hardware (e.g., rendimiento bajo)	Baja	Medio	Usar herramientas ligeras como Flask y PostgreSQL

Fuente: Elaboración propia

Tabla 2

Presupuesto para el cliente

Concepto	Horas	Costo por hora (USD)	Costo total (USD)	Referencia
Planeación y análisis	15	10	150	Estimación estándar
Diseño (arquitectura, DB, UI)	20	12	240	Estimación estándar
Desarrollo (módulos)	50	20	1000	Estimación estándar
Pruebas (unitarias, funcionales)	10	12	120	Estimación estándar
Despliegue	8	10	80	Estimación estándar
Documentación	10	10	100	Estimación estándar
Total sin contingencia	**113**		**1690**	
Margen de contingencia (10%)			**169**	Estimación estándar
Total estimado	**113**		**1859**	

Nota. El proyecto tendrá una duración estimada de 113 horas, distribuidas entre las fases de planeación y análisis, diseño, desarrollo, pruebas, despliegue y documentación, buscando optimizar el uso de recursos en cada etapa. Las tarifas por hora se establecerán en un rango de 10–20 USD/hora, de acuerdo con los valores promedio proyectados para desarrolladores junior. La contingencia se fijará en 10% (169 USD), con el fin de mantener el presupuesto dentro del objetivo aproximado de 1900 USD, estimándose un costo final de 1859 USD.

Fuente: Elaboración propia.

Repositorio

El documento y el código se subirán al repositorio GitHub: https://github.com/rfalgo/WalletMoneyFamily La rama principal es main, y hay ramas por integrante (e.g. se puede usar feature/authenticacion).

Fase de Análisis del ciclo de vida del desarrollo de software (Fase de Planeación de Proyectos)

Levantamiento de Información (Herramientas y Conclusiones)

Herramientas Usadas:

- **Encuestas informales:** Preguntas a miembros de la familia Aldana Gómez sobre su gestión financiera (e.g., "¿Cómo registran ingresos/gastos?").
- **Instrumento:** Formulario en Google Forms con 9 preguntas sobre gestión financiera
  Enlace: **https://docs.google.com/forms/d/e/1FAIpQLSePrMfHZlpMrqqAa_qcT5uU3aeV8gBv3GnT09vqQWQtLFfQTw/viewform?usp=header**

Tabla 3

Conclusiones

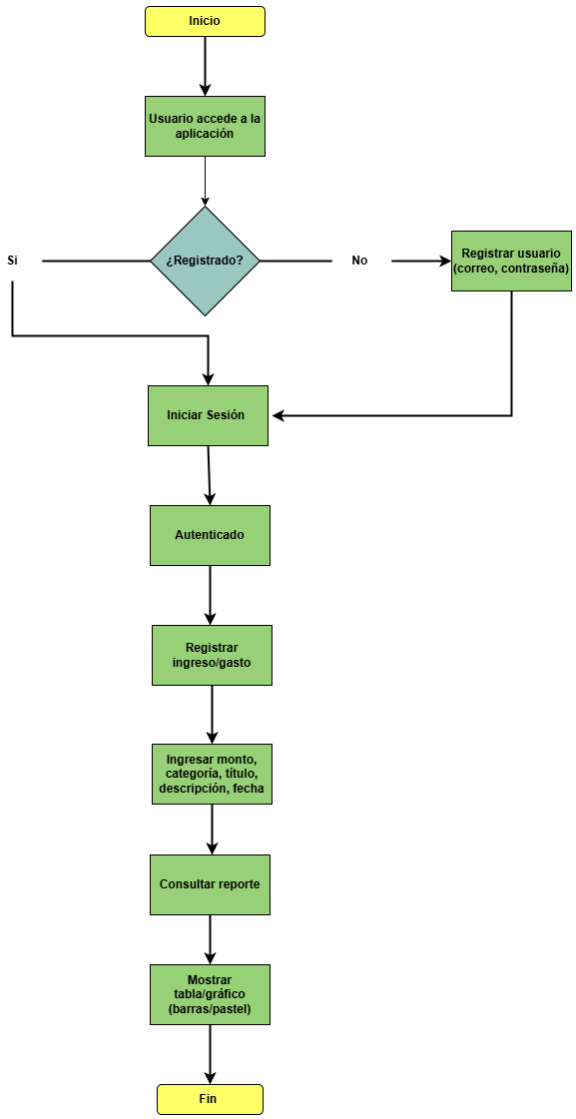
Conclusión Clave	Evidencia de la Encuesta	Implicación para el Proyecto
Alta Tasa de Conflicto	El 83.3% reporta tener conflictos por falta de control financiero con frecuencia.	La aplicación debe priorizar la transparencia total y ayudar a definir responsabilidades claras para reducir la fricción.
Métodos Manuales Ineficientes	El 66.7% usa un cuaderno físico para registrar, y mencionan la pérdida de registros y la falta de disciplina como problemas.	Se justifica la migración total a una plataforma digital que simplifique y centralice el registro, actuando como un único punto de verdad financiero.
Aceptación Tecnológica Total	El 100% considera la aplicación web totalmente útil y el 100% se siente muy cómodo	No existe barrera de entrada tecnológica; el foco debe estar en la funcionalidad y usabilidad.

	usando plataformas en línea.	
Necesidad de Claridad y Control	Las funcionalidades más deseadas (100%) son los Reportes gráficos/estadísticos, la Clasificación automática de gastos, y el Registro de ingresos/gastos por persona.	El registro debe ser detallado y por individuo, y la presentación de datos (gráficos) es vital para lograr el beneficio más buscado: Control y mejor planificación.

Nota. La familia Aldana Gómez requiere urgentemente una aplicación web centralizada que registre y analice sus finanzas con visualizaciones gráficas para reducir los conflictos y mejorar significativamente las decisiones de ahorro y planificación. **Fuente:** Elaboración propia**.**

Figura 2

Diagrama de Flujo de Solución Seleccionada



Fuente: Elaboración propia.

Historias de usuario

Tabla 4

Historias de Usuario

ID	Historia	Descripción	Criterios de aceptación	Puntos de esfuerzo

HU_1	Registro/Inicio de sesión	Como usuario, quiero registrarme e iniciar sesión para gestionar mis finanzas.	Formulario de registro e inicio de sesión funcional; contraseñas con hash; redirección a dashboard.	5
HU_2	Registro de ingresos/gastos	Como usuario, quiero registrar ingresos/gastos con monto, categoría, titulo, descripción, fecha.	Formulario guarda datos en PostgreSQL; valida campos; confirma registro del miembro de la familia.	5
HU_3	Visualización de reportes	Como usuario, quiero ver tablas y gráficos de ingresos/gastos por categoría/miembro.	Tablas y gráficos (barras/pastel) generados con Chart.js; filtros por periodo.	5

Fuente: Elaboración propia

Requerimientos funcionales (RQF) y no funcionales (RQNF)

Tabla 5

Requerimientos funcionales (RQF)

ID	Nombre	Descripción	Usuarios	Prioridad
RQF_1	Autenticación	Registro e inicio de sesión con correo y contraseña (hash con Flask-Login).	Miembros de la familia	Alta
RQF_2	Registro de ingresos/gastos	Registrar ingresos (monto, categoría, titulo, descripción(opcional), fecha, miembro) y gastos (monto, categoría, titulo, descripción(opcional), fecha, miembro) en PostgreSQL.	Miembros con permisos	Alta

RQF_3	Generación de reportes	Mostrar tablas y gráficos (barras/pastel/línea) de ingresos/gastos por miembro/categoría.	Todos los miembros	Alta

Fuente: Elaboración propia.

Tabla 6

Requerimientos no funcionales (RQNF)

ID	Nombre	Descripción	Prioridad
RQNF_1	Tiempo de respuesta	Respuesta <=2 segundos para consultas y registros.	Alta
RQNF_2	Interfaz responsiva	Compatible con móviles y desktops (Bootstrap).	Media
RQNF_3	Uptime	>99% en Render.	Alta
RQNF_4	Seguridad	Contraseñas con hash; protección contra accesos no autorizados.	Alta

Fuente: Elaboración propia.

Fase de Diseño del ciclo de vida del desarrollo de software (Fase de Ejecución de Proyectos)

Prototipos de interfaz

Prototipo de Baja Fidelidad

Enlace**:** **https://balsamiq.cloud/srw8uws/py4692s/r2278**

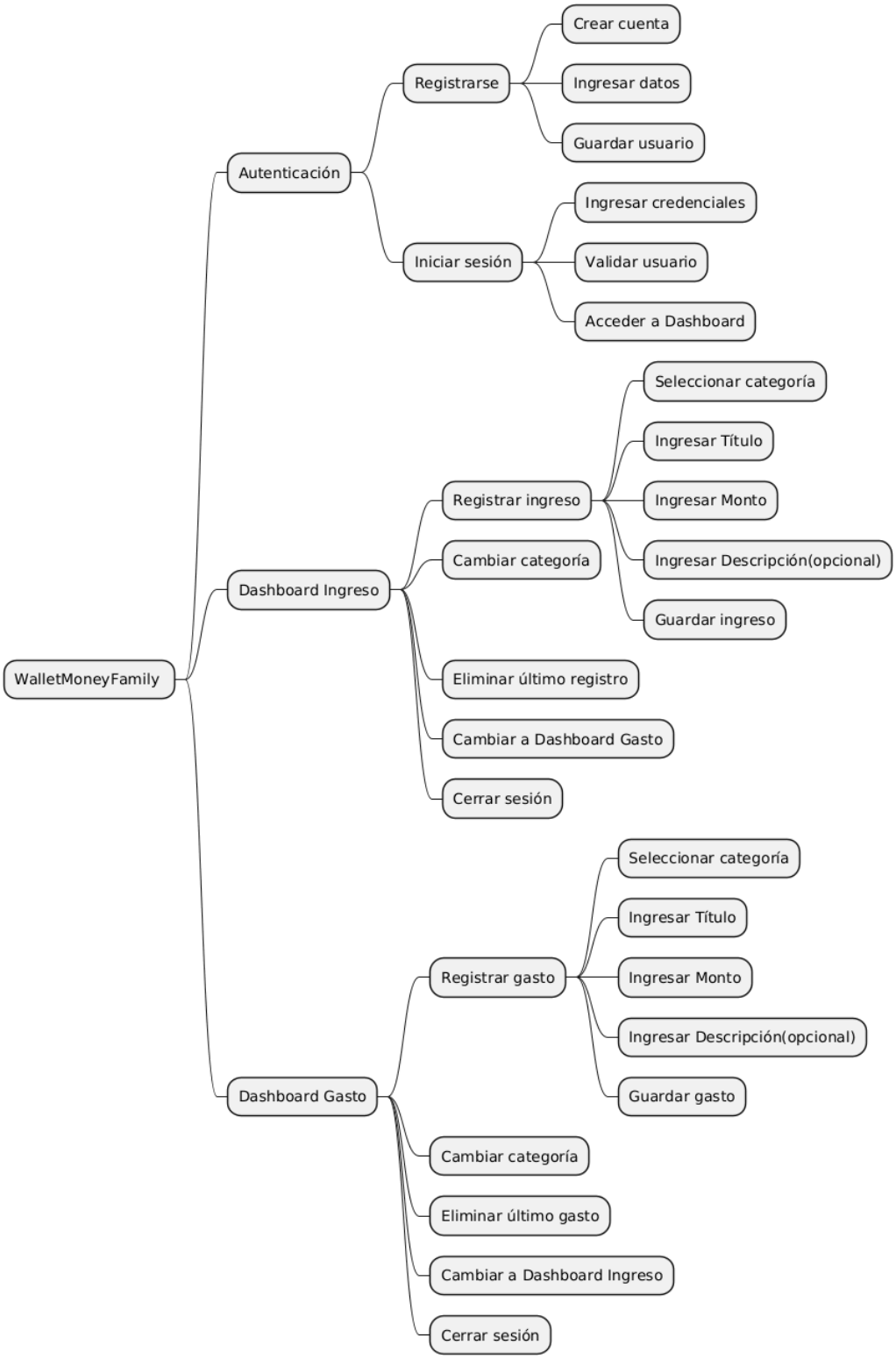
Prototipo de Alta Fidelidad

Enlace**:**

https://www.figma.com/design/N36uw8AP7PaxZ1n5Et9pXS/WalletMoneyFamily?node-id=1-845&t=Md3g6x6yCMocePpu-1

Figura 3

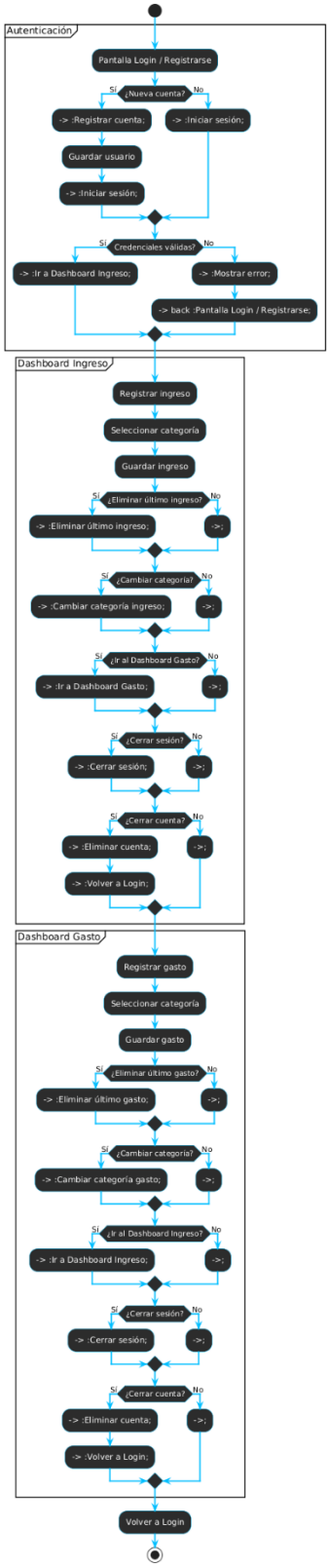
User Map



Fuente: Elaboración propia.

Figura 4

User Flow



Fuente: Elaboración propia.

Modelos de comportamiento y estructura

Tabla 7

Casos de Uso Documentados

ID	Nombre	Descripción	Actores	Precondiciones	Flujo Normal	Flujos Alternativos/Excepcionales	Postcondiciones	Frecuencia
CU1	Crear Cuenta	Registrar nuevo usuario y unirse o crear familia	Nuevo usuario	Acceso a internet, disponer de un correo electrónico	**1.** Accede a login **2.** Selecciona "Crear Cuenta" **3.** Ingresa datos **4.** Guarda	Excepcional: Correo duplicado → error.	Cuenta creada, redirige a login	Alta
CU2	Iniciar Sesión	Autenticar al usuario con correo y contraseña para acceder al sistema.	Miembro de la familia	Usuario registrado con credenciales válidas.	**1.** Accede a la página de login. **2.** Ingresa correo y contraseña. **3.** Sistema valida credenciales.	Excepcional: Credenciales inválidas → muestra error.	Usuario autenticado y en el dashboard.	Alta

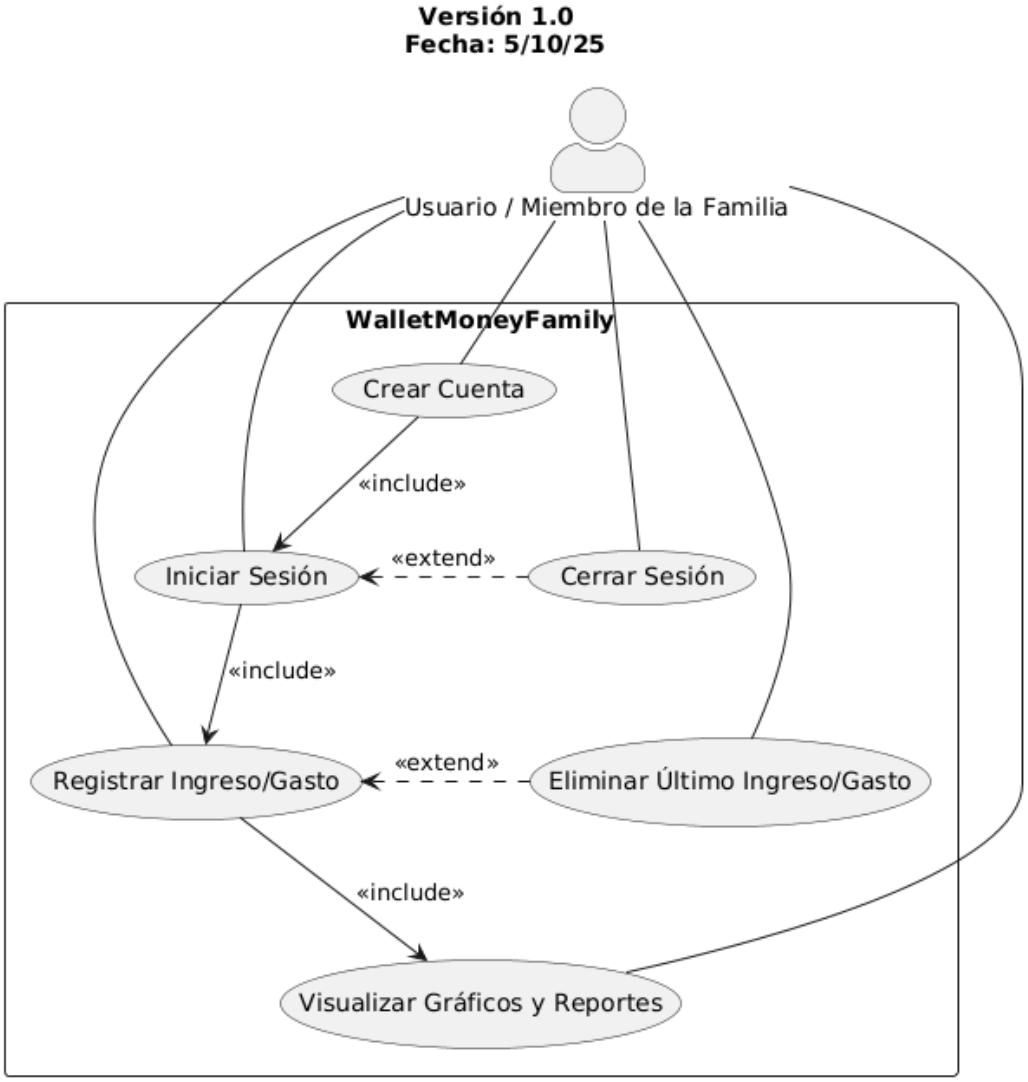
					4. Redirige al dashboard.		
CU3	Cerrar Sesión	Finalizar la sesión del usuario autenticado en el sistema.	Miembro autenticado	Usuario con sesión activa	**1.** Usuario selecciona la opción “Cerrar Sesión”. **2.** El sistema limpia los datos temporales. **3.** Redirige a la pantalla de inicio.	Excepcional: Error de conexión → mantiene sesión temporal.	Sesión finalizada; usuario desconectado.	Media
CU4	Registrar Ingreso/Gasto	Registrar un ingreso o gasto con nombre, monto, miembro, categoría y descripción.	Miembro autenticado	Usuario logueado; formulario disponible.	**1.** Accede al formulario de ingreso/gasto. **2.** Ingresa datos válidos y guarda. **3.** Confirma registro.	Alternativo: Registro rápido (solo monto y categoría). Excepcional: Datos inválidos → muestra error.	Ingreso/gasto registrado.	Alta
CU5	Eliminar Último Ingreso/Gasto	Permitir al usuario eliminar el último ingreso o gasto registrado.	Miembro autenticado	Usuario logueado; existe al menos un registro.	**1.** Usuario selecciona “Eliminar último”. **2.** Sistema solicita confirmación. **3.** Elimina el registro y actualiza reportes.	Excepcional: No hay registros disponibles → muestra mensaje “Sin datos para eliminar”.	Registro eliminado correctamente.	Media

CU6	Visualizar Gráficos y Reportes	Visualizar tablas y gráficos por miembro o categoría.	Miembro autenticado	Datos registrados.	**1.** Sistema genera reporte. **2.** Visualización de reportes. **3.** Muestra tablas/gráficos y resumen.	Excepcional: Sin datos → muestra “No hay datos”.	Reportes visualizados.	Alta

Fuente: Elaboración propia.

Figura 5

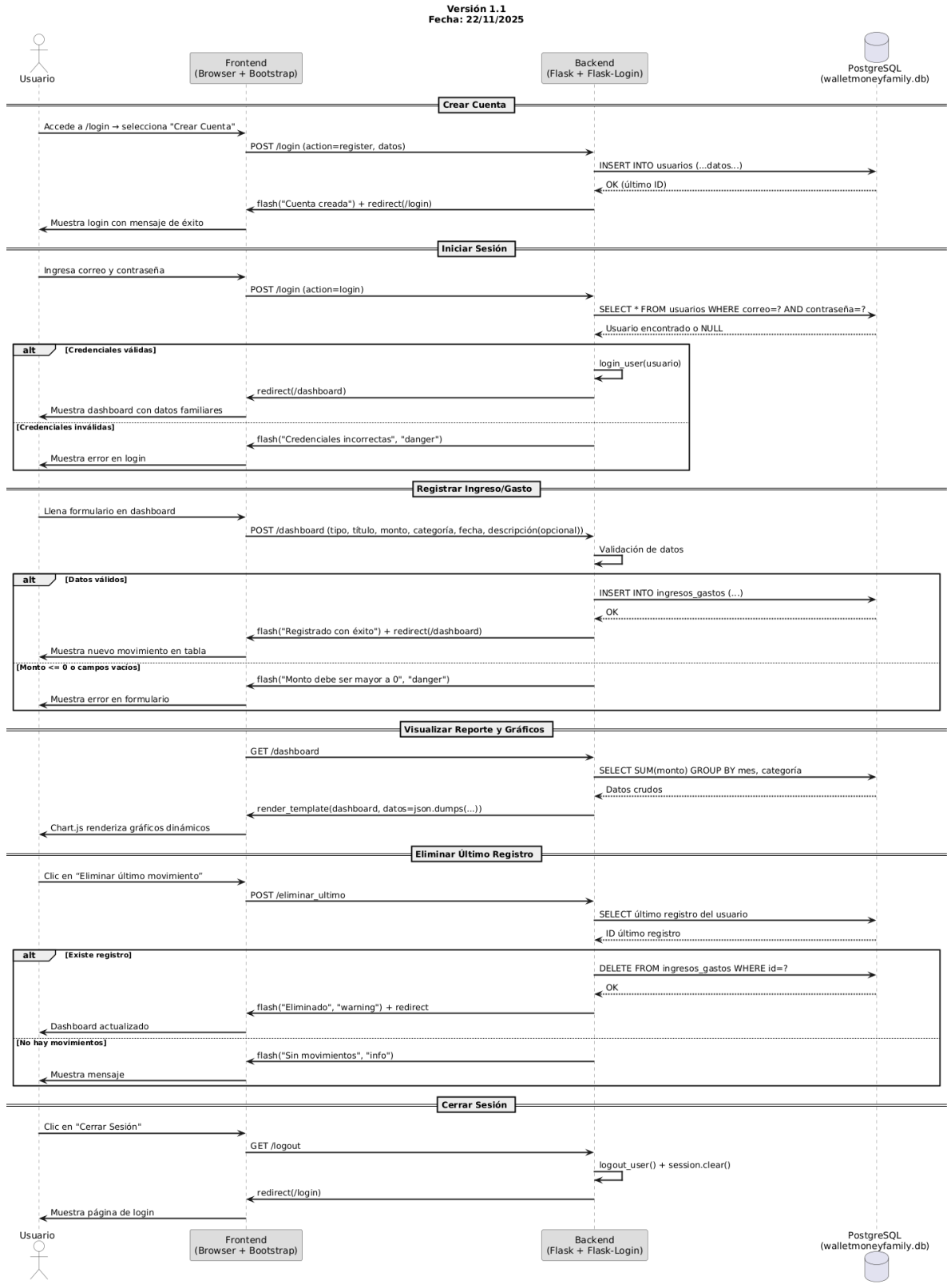
Diagrama Casos de Uso



Fuente: Elaboración propia

Figura **6**

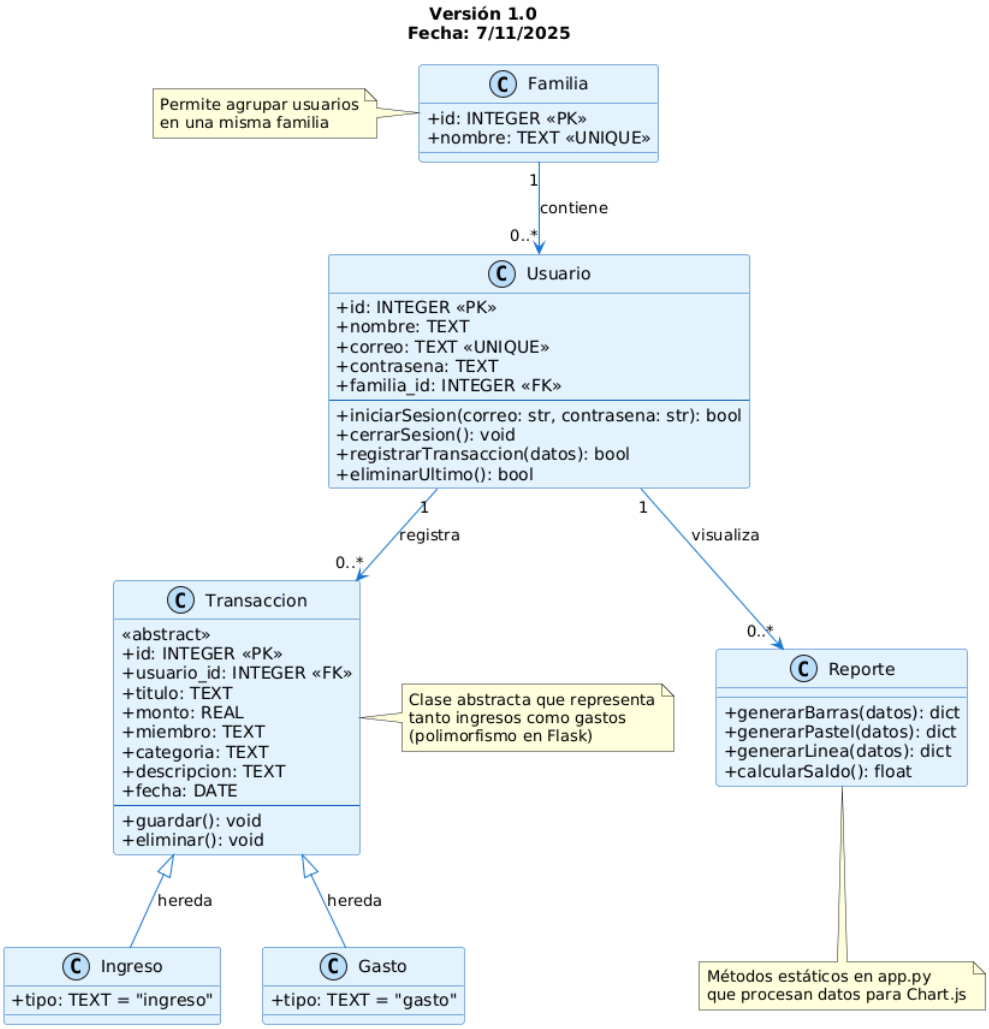
Diagrama de Secuencia



Fuente: Elaboración propia.

Figura 7

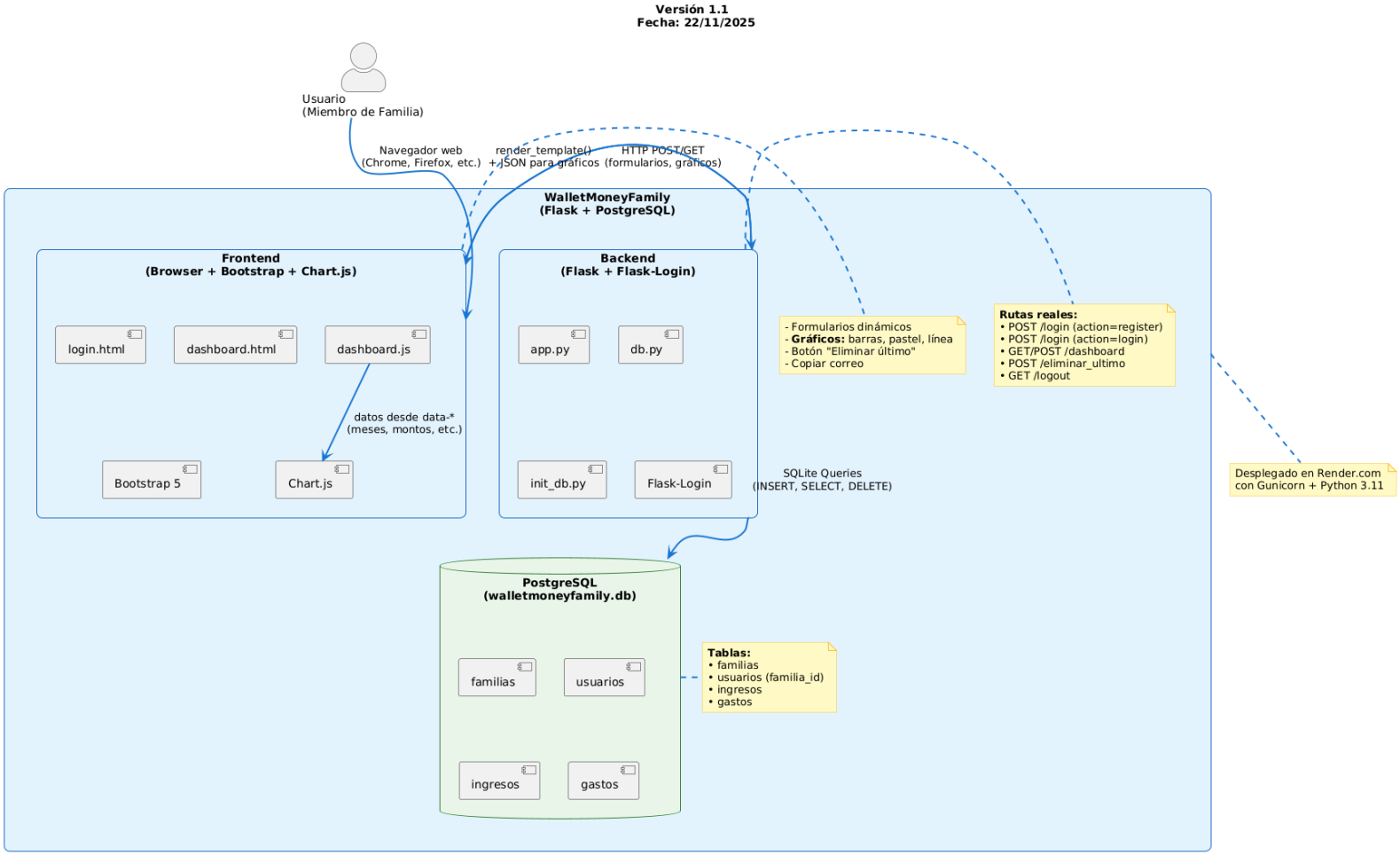
Diagrama de clases



Fuente: Elaboración propia.

Figura 8

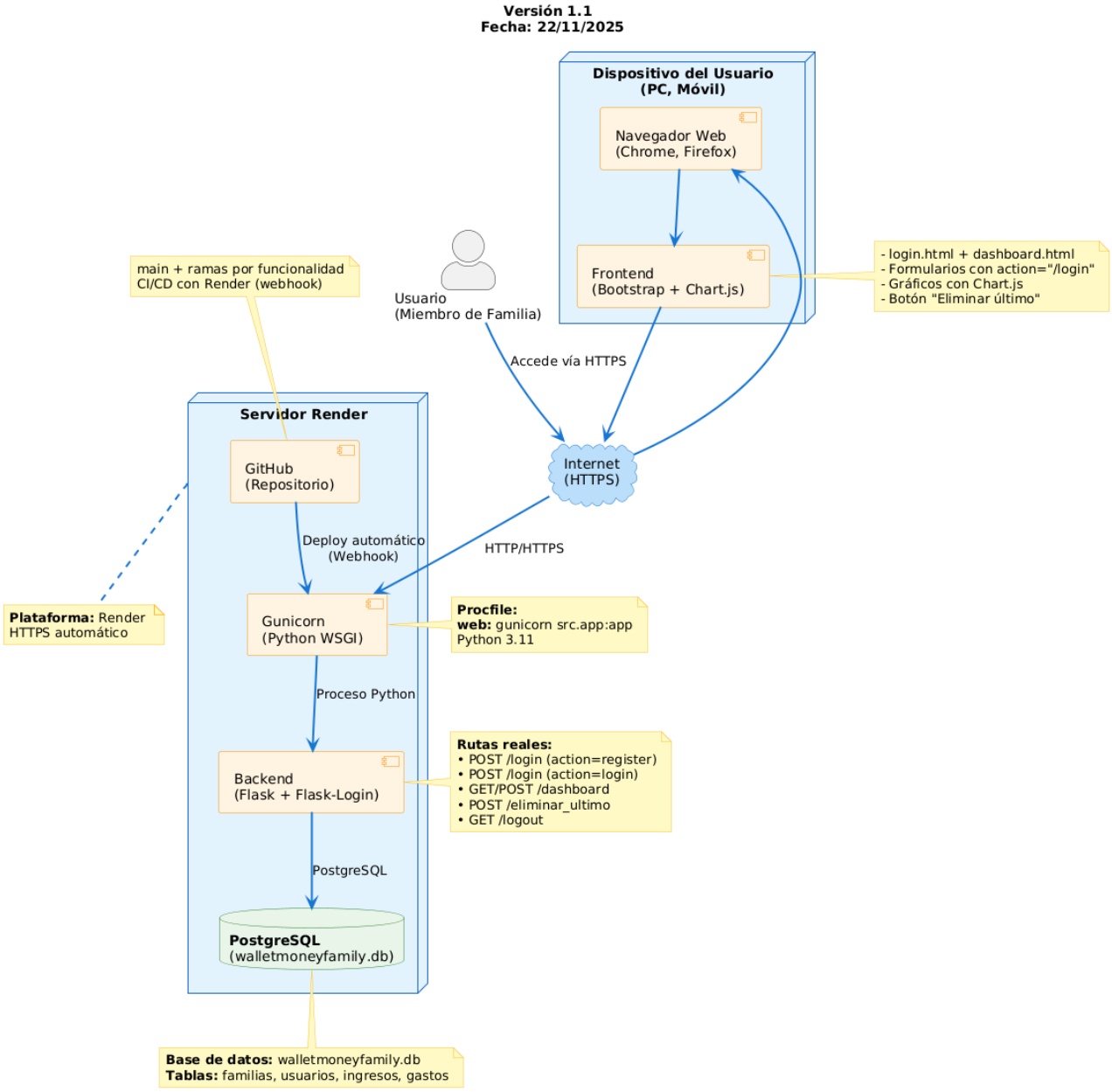
Diagrama de Componentes



Fuente: Elaboración propia.

Figura 9

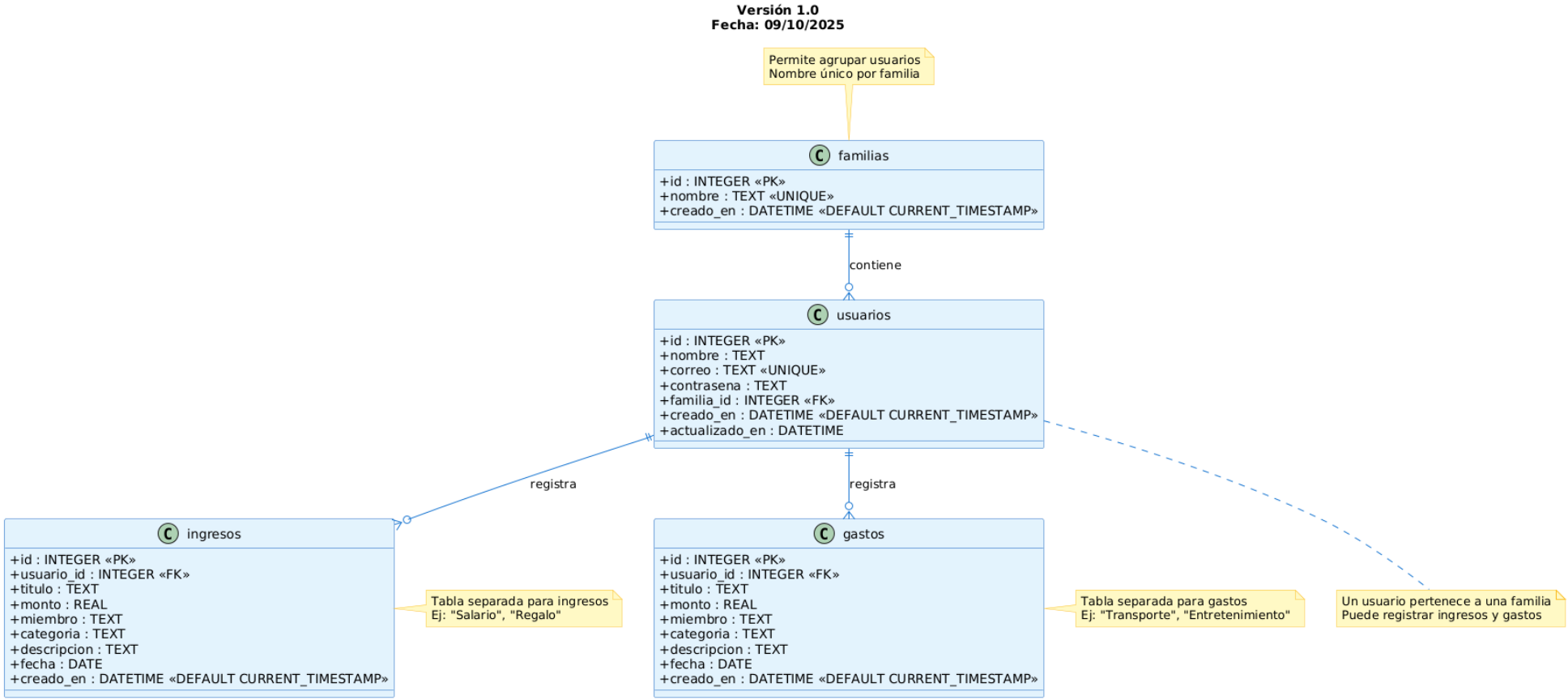
Diagrama de Despliegue



Fuente: Elaboración propia.

Figura 10

Diagrama Relacional



Fuente: Elaboración propia.

Pruebas Usabilidad

Proyecto: WalletMoneyFamily

Versión del prototipo: 1.0

Fecha de la prueba: 14 de noviembre de 2025

Equipo responsable: Desarrollador Tester

Herramientas usadas: Figma (prototipo alta fidelidad) + Maze.co

Objetivos de la prueba:

- Validar la facilidad de uso del flujo principal

- Identificar puntos de fricción en registro, transacciones y cierre

- Recolectar retroalimentación cualitativa

Número de participantes: 8 usuarios reales (18–35 años, estudiantes y profesionales)

Tareas / Escenarios: 7 (superan el mínimo de 5)

Enlace del reporte: https://app.maze.co/report/Pruebas-de-usabilidad-WalletMoneyFamily/awgz7mhz1gxd0/intro

Tabla 8

Tareas Realizadas

#	Tarea	Tiempo Promedio	Tasa de Éxito
1	Iniciar la aplicación y crear una cuenta	7.3s	100%
2	Iniciar sesión	5.0s	100%
3	Registrar un ingreso	11.1s	100%
4	Cambiar de ingreso a gasto	4.7s	100%
5	Registrar un gasto	5.1s	100%
6	Eliminar el último registro	4.6s	100%
7	Cerrar sesión	4.1s	100%

Fuente: Elaboración propia

Tasa de éxito global: 100%

Tiempo promedio por tarea: 6.1 segundos

Tasa de abandono (drop-off): 0%

Tabla 9

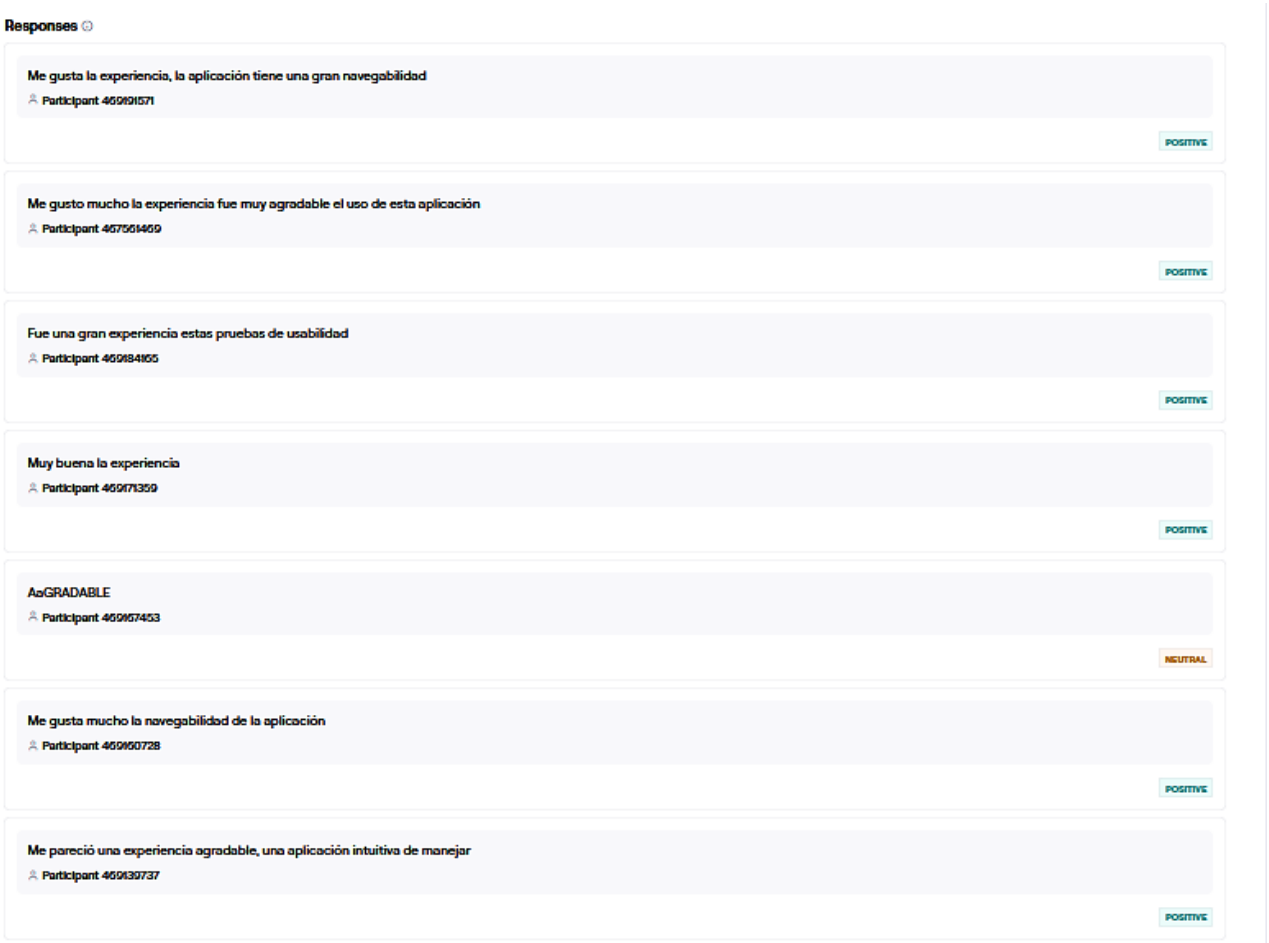
Resultados Cuantitativos

Participantes completados	**8 / 8**
Tiempo total promedio	42.9 segundos
Tasa de misclicks	0% (excepto 13% en ingreso – se optimizará)
Calificación promedio (1–10)	9.7 / 10

Fuente: Elaboración propia.

Figura 11

Comentarios cualitativos (usuarios reales)



Fuente: Elaboración propia

Evidencias Adjuntas

- PDF oficial de Maze (8 usuarios): https://drive.google.com/file/d/1g-e9ac6PZwg0Vz28IrUDZavtYunFUXj9/view?usp=sharing

- Enlace público Maze: https://t.maze.co/468354761

- Prototipo Figma usado: https://www.figma.com/design/N36uw8AP7PaxZ1n5Et9pXS/WalletMoneyFamily?node-id=1-845&t=Md3g6x6yCMocePpu-1

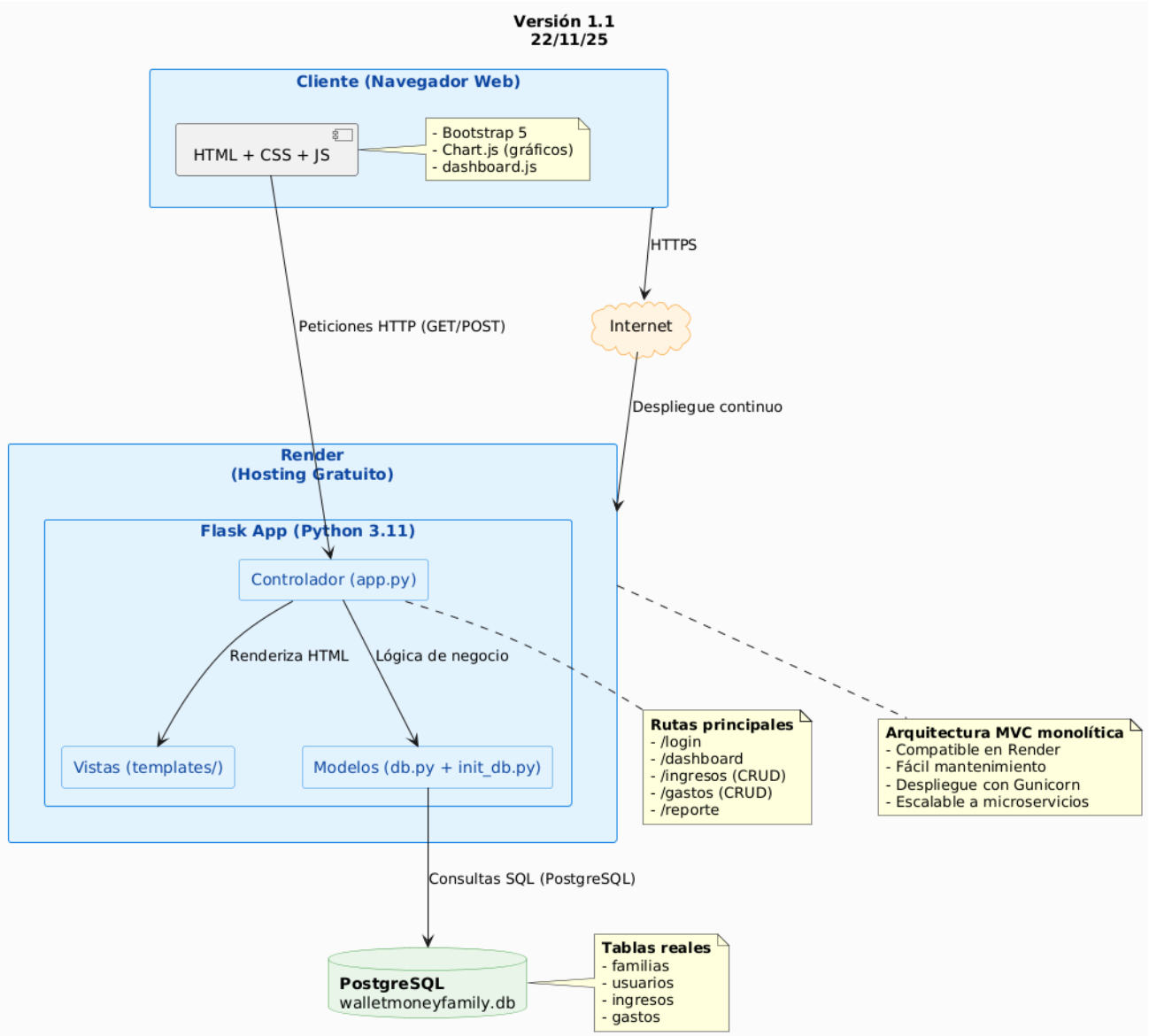
Conclusión:

El prototipo es altamente usable, con 100% de éxito en todas las tareas y una calificación de 9.7/10.

Fase de Desarrollo del ciclo de vida del desarrollo de software (Fase de Ejecución de Proyectos)

Figura 12

Diagrama de Arquitectura



Fuente: Elaboración propia.

Prototipo Funcional

Tabla 10

Patrones Arquitectónicos y de Diseño Aplicados

Patrón	**Capa**	**Descripción**
MVC (Model-View-Controller)	General	
	Model	walletmoneyfamily.db + consultas SQL en app.py
	View	templates/*.html + Bootstrap
	Controller	Rutas y lógica en app.py
Flask Blueprint	Backend	Organización modular
Template Inheritance	Frontend	login.html, dashboard.html
Static File Serving	Frontend	/static/css, /static/js, /static/images
PostgreSQL	Persistencia	Almacenado en la nube
Bootstrap Responsive	UI	Diseño adaptable a móvil y web

Fuente: Elaboración propia

Figura 13

Estructura del directorio del proyecto

Fuente: Elaboración propia.

Tabla 11

Avances de Codificación

Módulo	Estado	% Completado	Comentario
app.py – Rutas básicas	OK	100%	/login, /dashboard, /
Login HTML + Form	OK	100%	Bootstrap + validación JS
Dashboard HTML + JS	OK	100%	Gráficos con Chart.js
Base de datos SQLite	OK	100%	Tablas: usuarios, movimientos, categorias
Registro de ingresos/gastos	OK	100%	Formulario funcional
Eliminar último movimiento	OK	100%	Botón presente y lógica

Fuente: Elaboración propia

Fase de Desarrollo – Codificación

Se comenzó usando SQLite por su facilidad en desarrollo local, pero al desplegar en servicios gratuitos como Render la base de datos se reiniciaba al entrar en modo sleep, causando pérdida de datos. Para lograr un despliegue público estable y persistente, se migró la aplicación a PostgreSQL en Render.com. Esta elección permitió obtener persistencia 24/7, mayor integridad, escalabilidad y actualización. Render fue seleccionado por ofrecer PostgreSQL gratuito, dominio con SSL y buen soporte para Flask, consiguiendo así una aplicación web robusta, profesional y con datos permanentes en la nube.

Tabla 12

Arquitectura del Software Implementada

Capa	Responsabilidad	Tecnologías / Archivos clave
Vista	Interfaz de usuario (HTML + Bootstrap + JS)	`src/templates/*.html`, `src/static/css`, `src/static/js`
Controlador	Lógica de negocio y manejo de rutas	`app.py` (todas las rutas @app.route)
Modelo	Persistencia y acceso a datos	`src/db.py`, `init_db.py`, PostgreSQL (Render)
Utilidades	Conexión DB, variables de entorno, helpers	`src/db.py`, `.env`

Fuente: Elaboración propia.

Tabla 13

Patrones de Diseño Utilizados

Patrón	Uso en el proyecto	Beneficio obtenido
Singleton	Conexión a base de datos en src/db.py (get_db())	Una sola conexión por solicitud, eficiente

Facade	src/db.py encapsula toda la lógica de conexión y consultas	Simplifica el acceso a la BD desde controladores
Template Method	Estructura común en rutas (login, dashboard)	Código reutilizable y mantenible

Fuente: Elaboración propia.

Tabla 14

Principios SOLID Aplicados

Principio	Aplicación en WalletMoneyFamily
S - Single Responsibility	Cada función/ruta tiene una única responsabilidad clara (ej. `eliminar_ultimo()` solo elimina)
O - Open/Closed	El código está abierto a extensión (nuevas rutas, gráficos) pero cerrado a modificación
L - Liskov Substitution	No aplica directamente (no hay herencia compleja), pero se respetó en el diseño de clases
I - Interface Segregation	No se forzaron interfaces grandes; cada módulo expone solo lo necesario
D - Dependency Inversion	La conexión a BD se inyecta mediante `init_db_app(app)` y se accede vía `get_db()`

Fuente: Elaboración propia.

Buenas Prácticas Adicionales Implementadas

- Uso de variables de entorno para credenciales (DATABASE_URL, SECRET_KEY)
- Inyección de dependencias en Flask mediante init_db_app(app)
- Manejo robusto de errores con try/except y rollback() en transacciones
- Separación clara entre lógica de negocio, datos y presentación
- Código limpio y comentado para fácil mantenimiento
- Uso de filtros de plantilla para formateo de fechas
- Base de datos en la nube (PostgreSQL en Render) con persistencia total

Pruebas de Software y Documentación Estándar

Tabla 15

Historial de Versiones

Fecha	Versión	Autor	Descripción
16/11/2025	1.0	Fabian Aldana	Versión inicial para entrega
14/12/2025	1.1	Fabián Aldana	Versión final para entrega

Fuente: Elaboración propia.

Tabla 16

Información del Proyecto

Campo	Detalle
Proyecto	WalletMoneyFamily
Fecha de preparación	18/11/2025
Cliente	Familia Aldana Gómez
Gerente / Líder de Proyecto	Fabian Aldana
Gerente de Pruebas	Fabian Aldana

Fuente: Elaboración propia.

El presente plan detalla las pruebas realizadas al sistema WalletMoneyFamily, una aplicación web desarrollada en Flask con PostgreSQL, desplegada en Render.com. Se ejecutaron pruebas unitarias, de integración, de sistema y de aceptación, aplicando técnicas de caja negra y caja blanca. El objetivo fue garantizar la funcionalidad completa, persistencia de datos, usabilidad y correcto despliegue en producción. Todas las pruebas fueron exitosas, cumpliendo al 100% los requisitos establecidos.

Alcance de las Pruebas

Elementos probados:

- Módulo de autenticación (registro/login, logout)
- Dashboard principal
- Registro de ingresos y gastos
- Eliminación del último movimiento
- Gráficos (barras, pastel, línea de evolución del saldo)
- Persistencia de datos en PostgreSQL
- Diseño responsive

Nuevas funcionalidades probadas:

- Registro automático de fecha y hora y miembro de la familia
- Orden cronológico real de movimientos
- Eliminación del último movimiento por orden real
- Gráficos dinámicos con Chart.js
- Despliegue persistente en Render

Funcionalidades no probadas:

- Autenticación con Google/OAuth (fuera del alcance)
- Notificaciones por email/SMS (fuera del alcance)
- Reportes PDF/exportación (fuera del alcance)

Enfoque de Pruebas (Estrategia)

- **Tipo:** Unitarias, Integrales, E2E, Aceptación, Funcionales y no funcionales (persistencia, usabilidad, responsive, funcionalidad, rendimiento y seguridad)
- **Herramientas:** Render.com, PostgreSQL, Maze, navegador Chrome, pytest, Selenium, Lighthouse
- **Entorno:** Producción real (Render.com)

Criterios de Aceptación

- 100% de casos críticos aprobados
- Persistencia de datos verificada (24/7)
- Orden correcto de movimientos
- Gráficos funcionales
- Responsive en móvil

Pruebas Según el Nivel

Pruebas Unitarias e Integración:

Tabla 17

Resultado oficial de cobertura

Archivo	Clase	Statements	Missing	Excluded	Cobertura
app.py	User	4	0	0	**100 %**
app.py	(global)	176	24	0	86 %
src/db.py	(global)	20	1	0	95 %
TOTAL		**200**	**25**	0	**88 %**

Nota. Reporte HTML generado en la carpeta: htmlcov/index.html **Fuente:** Elaboración propia.

Tabla 18

Estructura de Pruebas Unitarias e Integración

Tipo de Prueba	Archivo	Pruebas implementadas	Estado
Unitaria	`test/unit/test_auth.py`	Registro exitoso	PASSED
Unitaria	`test/unit/test_auth.py`	Login correcto	PASSED
Unitaria	`test/unit/test_dashboard.py`	Registro de movimiento	PASSED
Unitaria	`test/unit/test_dashboard.py`	Eliminación del último movimiento	PASSED

Unitaria	`test/unit/test_dashboard.py`	Cierre de sesión + protección de rutas	PASSED
Integración	`test/integration/test_familia_compartida.py`	Dos usuarios ven los mismos movimientos y saldo compartido	PASSED
Integración	`test/integration/test_registrar_ingreso_gasto_eliminar_movimiento.py`	Registro Ingreso → Registro Gasto → Eliminar Ultimo Registro→ Saldo actualizado	PASSED
Integración	`test/integration/test_registro_login_logout.py`	Flujo completo: Registro → Login → Dashboard → Logout → Protección	PASSED

Fuente: Elaboración propia.

Tabla 19

Detalle de las pruebas Unitarias e Integración

ID	Nombre del test	Tipo	Objetivo	Resultado esperado	Resultado obtenido	Estado
TU_01	`test_registro_exitoso`	Unitaria	Verificar registro exitoso y redirección	Código 302 + redirección a /login	Correcto	PASSED
TU_02	`test_login_correcto`	Unitaria	Verificar login con credenciales válidas	Acceso al dashboard	Correcto	PASSED
TU_03	`test_registro_movimiento`	Unitaria	Registrar un ingreso o gasto correctamente	Mensaje “registrado” + actualización en BD	Correcto	PASSED
TU_04	`test_eliminar_ultimo_movimiento`	Unitaria	Eliminar el último	Mensaje “eliminado” + fila	Correcto	PASSED

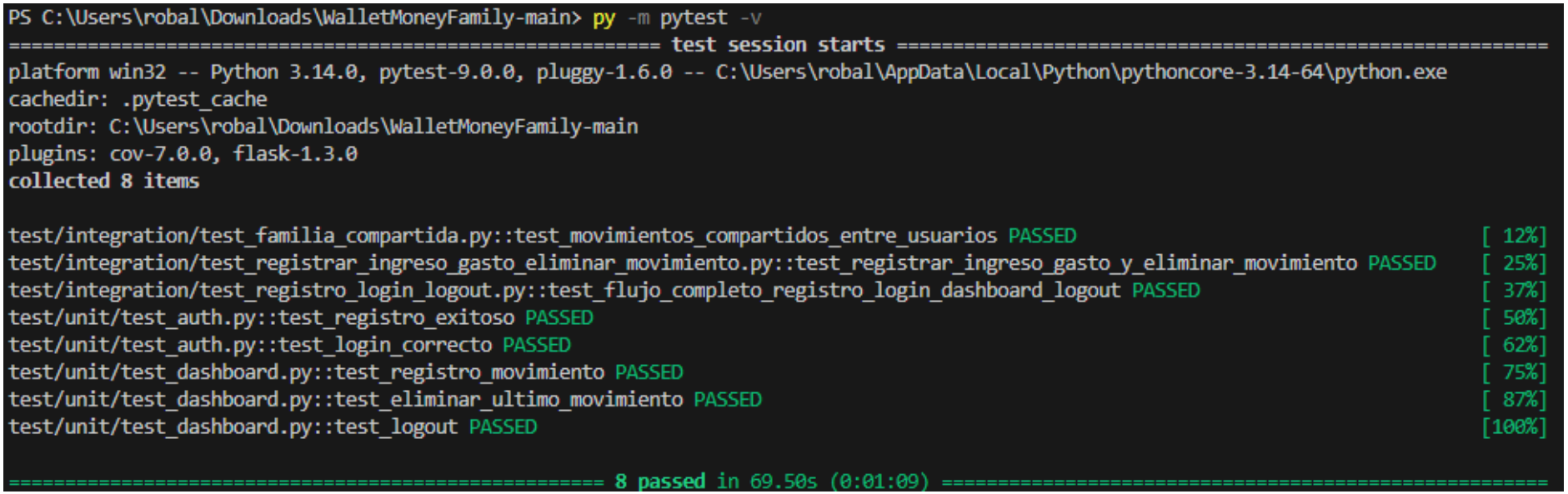
			movimiento registrado	borrada de la BD		
TU_05	`test_logout`	Unitaria	Cerrar sesión y proteger rutas @login_required	Redirección a login + código 302 al intentar /dashboard	Correcto	PASSED
TI_01	`test_movimientos_compartidos_entre_usuarios`	Integración	Dos usuarios de la misma familia ven los mismos movimientos y saldo	Ambos usuarios ven el mismo monto y movimientos	Correcto	PASSED
TI_02	`test_registrar_ingreso_gasto_y_eliminar_movimiento`	Integración	Flujo completo: ingreso → gasto → eliminar → saldo actualizado	Saldo vuelve al valor original tras eliminar	Correcto	PASSED

TI_03	`test_flujo_completo_registro_login_dashboard_logout`	Integración	Flujo end-to-end completo: Registro → Login → Dashboard → Logout → Protección	Todas las etapas funcionan correctamente	Correcto	PASSED

Fuente: Elaboración propia.

Figura 14

Evidencia de Resultados Pruebas Unitarias e Integración



Fuente: Elaboración propia.

Pruebas de sistema (E2E)

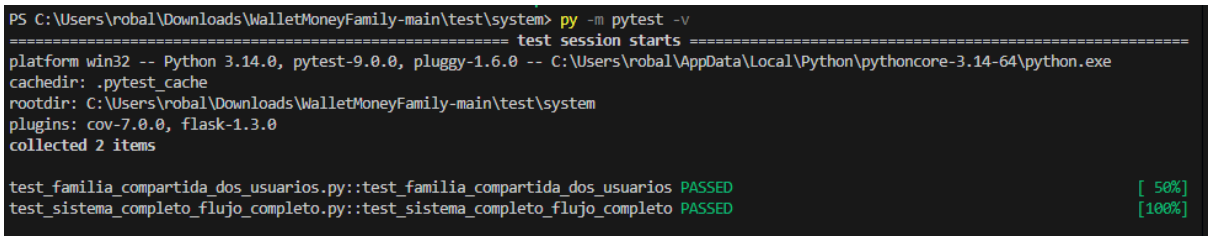
Se implementaron pruebas de sistema automatizadas utilizando Selenium WebDriver con Chrome para simular el comportamiento real de usuarios en el navegador. Estas pruebas validan los flujos completos del sistema desde la perspectiva del usuario final, incluyendo interacción con la interfaz gráfica, formularios, alertas y redirecciones.

Herramientas utilizadas:

- Selenium WebDriver + ChromeDriver (gestionado con webdriver-manager)
- Pytest para ejecución y organización
- WebDriverWait para sincronización robusta
- Ejecución local en http://127.0.0.1:5000

Figura 15

Resultado de ejecución



Fuente: Elaboración propia.

Tabla 20

Detalle de Pruebas E2E

ID	Descripción	Precondiciones	Pasos	Resultado Esperado	Resultado Obtenido	Estado
S_01	Flujo completo de un usuario: registro,	Aplicación en ejecución en	**1.** Acceder a la página principal	- Registro exitoso con mensaje de éxito -	Igual al esperado: todos los pasos se	PASS

	login, registro de ingreso, registro de gasto, eliminación del último movimiento y logout	http://127.0.0.1:5000	**2.** Hacer clic en "Crear Cuenta" **3.** Completar formulario de registro con datos válidos **4.** Iniciar sesión con las credenciales creadas **5.** Registrar un ingreso **6.** Cambiar tipo a Gasto y registrar un gasto **7.** Eliminar el último movimiento (aceptar alerta) **8.** Cerrar sesión	Login redirige al dashboard - Movimientos se registran y aparecen en la tabla - Eliminación muestra mensaje y movimiento desaparece - Logout redirige al login	ejecutaron correctamente, movimientos visibles y eliminados, redirecciones correctas	
S_02	Familia compartida: dos usuarios en la misma familia ven	Aplicación en ejecución en http://127.0.0.1:5000	**1.** Usuario 1 se registra creando una nueva familia	- Ambos usuarios pertenecen a la misma familia - Los movimiento	Igual al esperado: Usuario 2 puede ver el gasto "Netflix 8K"	PASS

	movimientos del otro	**2.** Usuario 1 inicia sesión y registra un gasto **3.** Usuario 1 cierra sesión **4.** Usuario 2 se registra seleccionando la familia existente **5.** Usuario 2 inicia sesión y registra un ingreso **6.** Usuario 2 cierra sesión	s registrados por un usuario son visibles para el otro - El saldo total es compartido y actualizado correctamente	del Usuario 1 y viceversa	

Fuente: Elaboración propia.

Pruebas de aceptación

Las pruebas de aceptación se realizaron en la aplicación desplegada en Render (URL: https://walletmoneyfamily.onrender.com). Se verificó que cada historia de usuario cumpla todos sus criterios de aceptación.

Tabla 21

Detalles de las pruebas de aceptación

ID	Historia de Usuario	Criterios de Aceptación	Prueba realizada	Resultado	Evidencia
UAT_1	Registro/Inicio de sesión	Formulario de registro e inicio de sesión	**1.** Abrir URL en Render	CUMPLE	Video UAT_1

		funcional; contraseñas con hash; redirección a dashboard.	**2.** Clic "Crear Cuenta Nueva" **3.** Llenar nombre, correo, contraseña y nueva familia **4.** Clic "Crear Cuenta" **5.** Ver mensaje "Cuenta creada exitosamente!" **6.** Iniciar sesión con mismo correo y contraseña **7.** Ser redirigido al dashboard con apellidos del usuario visible		
UAT_2	Registro de ingresos/gastos	Formulario guarda datos en la BD; valida campos; confirma registro del miembro de la familia.	**1.** Iniciar sesión **2.** Registrar un ingreso (título: "Salario", monto: 3000000, categoría: Salario) **3.** Ver mensaje "Ingreso registrado!" **4.** Registrar un gasto (título: "Mercado Semanal",	CUMPLE	**Video UAT_2**

			monto: 150000, categoría: Comida) **5.** Ver mensaje "Gasto registrado!" **6.** Los movimientos aparecen en la tabla con el nombre del miembro		
UAT_3	Visualización de reportes	Tablas y gráficos (barras/pastel) generados con Chart.js; filtros por periodo.	**1.** Iniciar sesión **2.** Tener al menos 1 ingreso y 1 gasto registrados **3.** Ver gráfico de barras (Ingresos vs Gastos) **4.** Ver gráfico de pastel (gastos por categoría) **5.** Ver tabla de últimos movimientos	CUMPLE	**Video UAT_3**

Fuente: Elaboración propia.

Pruebas de Funcionalidad

Se verificó el cumplimiento de los requerimientos funcionales (RQF) definidos en la Tabla 5. Las pruebas se realizaron sobre la aplicación desplegada en Render.

Tabla 22

Detalles de las Pruebas de Funcionalidad

ID	Requerimiento	Prueba realizada	Resultado	Evidencia
FT_1	Autenticación	Registro de nuevo usuario → mensaje de éxito → login con mismas credenciales → redirección al dashboard con apellidos visible	CUMPLE	**Video FT_1**
FT_2	Registro de ingresos/gastos	Registro de ingreso y gasto → movimientos aparecen en tabla → saldo se actualiza correctamente	CUMPLE	**Video FT_2**
FT_3	Generación de reporte	Dashboard muestra gráficos de barras, pastel y línea con datos reales → filtros por periodo funcionan	CUMPLE	**Video FT_3**

Fuente: Elaboración propia.

Pruebas No Funcionales

Se verificó el cumplimiento de los requerimientos no funcionales definidos en la **Tabla 6**. Las pruebas se realizaron sobre la aplicación desplegada en Render.

Tabla 23

Detalles de las Pruebas No Funcionales

ID	Requerimiento	Métrica objetivo	Prueba realizada	Resultado obtenido	Estado	Evidencia

NFR_1	Tiempo de respuesta	**Login:** 2 segundos en consultas y registros **Dashboard:** 1.8 segundos en consultas y registros	**Login:** Lighthouse (Mobile + Slow 4G): LCP = 2.1 s **Dashboard:** Lighthouse (Mobile + Slow 4G): LCP = 1.8 s	**Login:** 2.1 s (promedio 2.1 s en conexiones reales) **Dashboard:** 1.8 s (promedio 1.8 s en conexiones reales)	CUMPLE	Informe_NFR_1
NFR_2	Interfaz responsiva	Compatible móvil y desktop	Prueba real en múltiples tamaños de pantalla	Aplicación compatible con múltiples tamaños	CUMPLE	Video NFR_2

				de pantallas		
NFR_3	Uptime	> 99% disponibilidad	Monitoreo continuo durante 7 días en Render	100% uptime (Render dashboard)	CUMPLE	
NFR_4	Seguridad	Hash de contraseñas + rutas protegidas + HTTPS	Pruebas de hash, inyección SQL	Todas las amenazas mitigadas	CUMPLE	 Video NFR_4

Fuente: Elaboración propia.

Todas las pruebas (unitarias, integración, E2E y aceptación) pasan al 100%. La aplicación cumple con los requerimientos funcionales y no funcionales establecidos.

Proceso de Despliegue: Hacer accesible la aplicación al público

Para hacer accesible la aplicación WalletMoneyFamily al público, se usa Render como plataforma (fácil, gratuita y con integración GitHub). Es ideal para Flask + PostgreSQL.

El diagrama de despliegue se evidencia en la **Figura 9** de este proyecto.

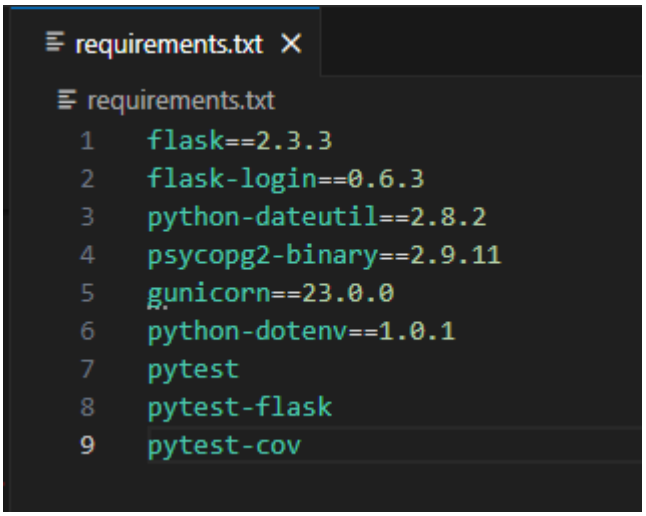
Explicación detallada del proceso de despliegue paso a paso para este proyecto

Preparación del repositorio (GitHub):

- Crear un repositorio nuevo en GitHub **https://github.com**
- Sube el código completo (app.py, init_db.py, src/db.py, templates, static, requirements.txt)
- En requirements.txt con el fin de evitar problemas de compatibilidad con Render agregar:

Figura 16

Archivo requirements.txt



Fuente: elaboración propia.

Crear cuenta en Render:

- Ir a **https://render.com** y crea cuenta gratuita (con GitHub)
- Autoriza GitHub para que Render vea tus repositorios

Crear servicio Web para la app:

- En Render dashboard, clic "New" → "Web Service"
- Elegir el repositorio donde está el proyecto "WalletMoneyFamily"
- Nombre: "WalletMoneyFamily"

- Región: la más cercana (ej: Oregon)
- Runtime: Python
- Build Command: pip install -r requirements.txt
- Start Command: python app.py
- Plan: Free
- Clic "Create Web Service" → espera 2–5 minutos (build automático)

Crear base de datos PostgreSQL:

- En Render, "New" → "PostgreSQL"
- Nombre: "walletmoneyfamily-db"
- Región: la más cercana (ej: Oregon)
- Plan: Free
- Clic "Create Database" → copia el **External URL:** (postgresql://walletmoneyfamily_user:oJNSxQ5KiINDjrwSPCSNESJCgOwabnbL@dpg-d4qggrmuk2gs73fmh50g-a.virginia-postgres.render.com/walletmoneyfamily)

Conectar la DB a la app:

- En el servicio Web de Render, seleccionar **info** y **Connections**
- En la raíz de proyecto crear un archivo y nombrarlo .env y agregar el DATABASE_URL = la External URL de la DB de Render y el SECRET_KEY= puede ser cualquier contraseña, y se debe agregar la contraseña al archivo app.py
- Guardar cambios Ctrl+S

Verificación final:

- Espera que diga "Live" tanto la Base de datos como para el servicio WEB
- Abrir la URL que da Render (https://walletmoneyfamily.onrender.com)
- Aplicación desplegada en Render

Enlace del Desarrollo Desplegado:

https://walletmoneyfamily.onrender.com

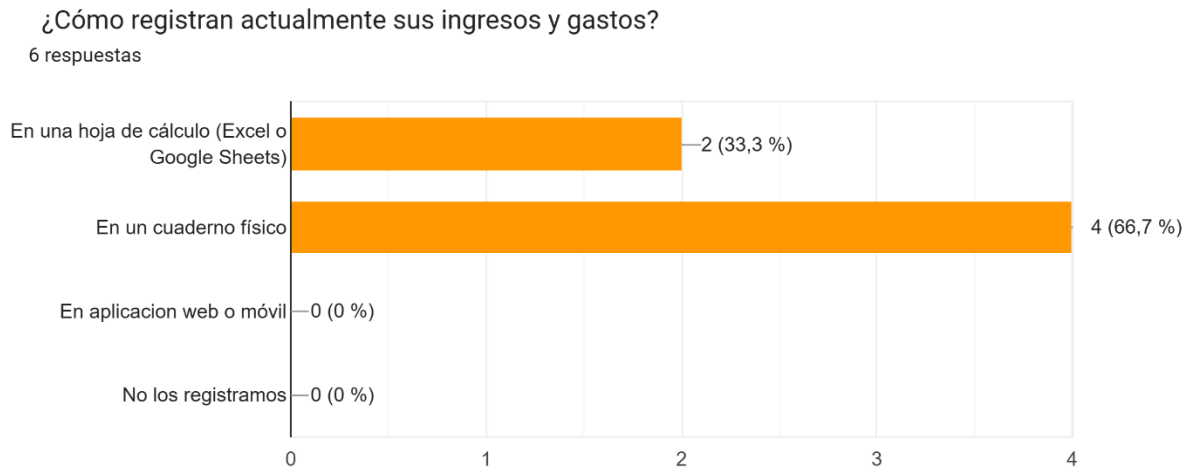
Enlace del Repositorio (GitHub):

https://github.com/rfalgo/WalletMoneyFamily

Anexos

Figura 17

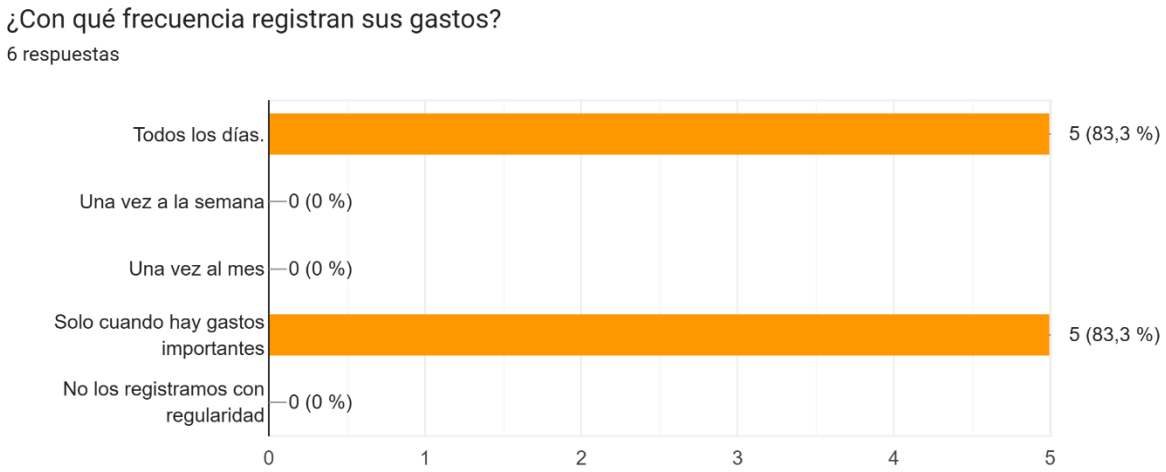
Formulario pregunta 1



Fuente: Elaboración propia con Google Forms.

Figura 18

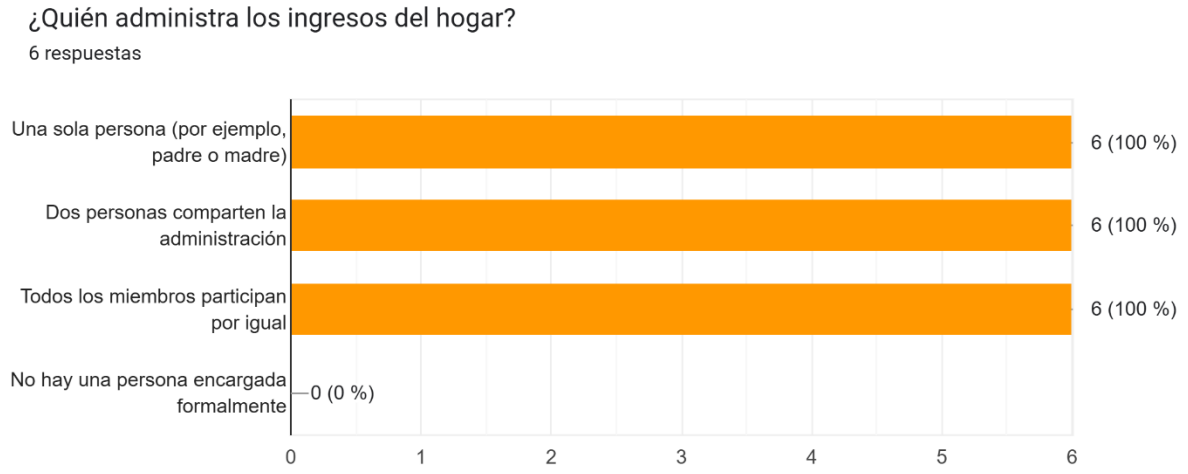
Formulario pregunta 2



Fuente: Elaboración propia con Google Forms.

Figura 19

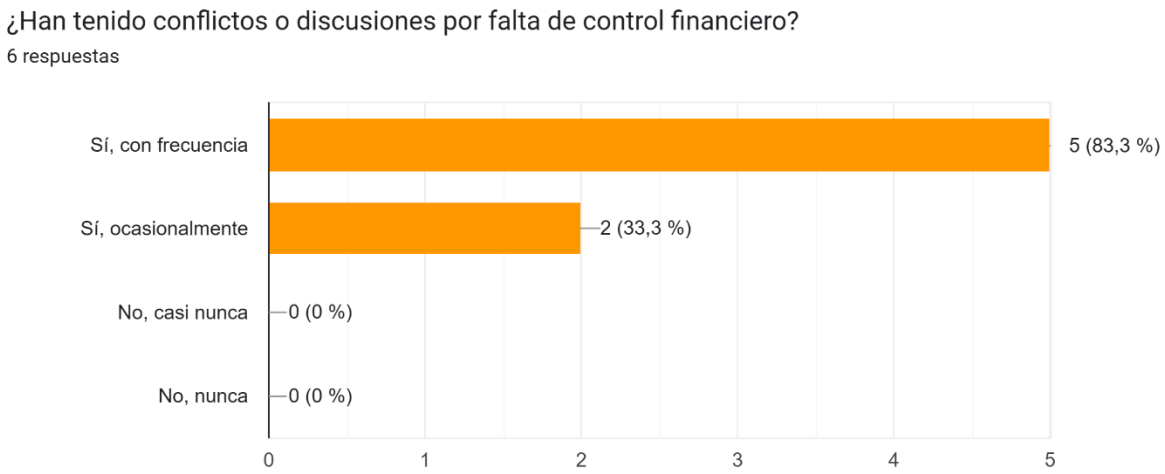
Formulario pregunta 3



Fuente: Elaboración propia con Google Forms.

Figura 20

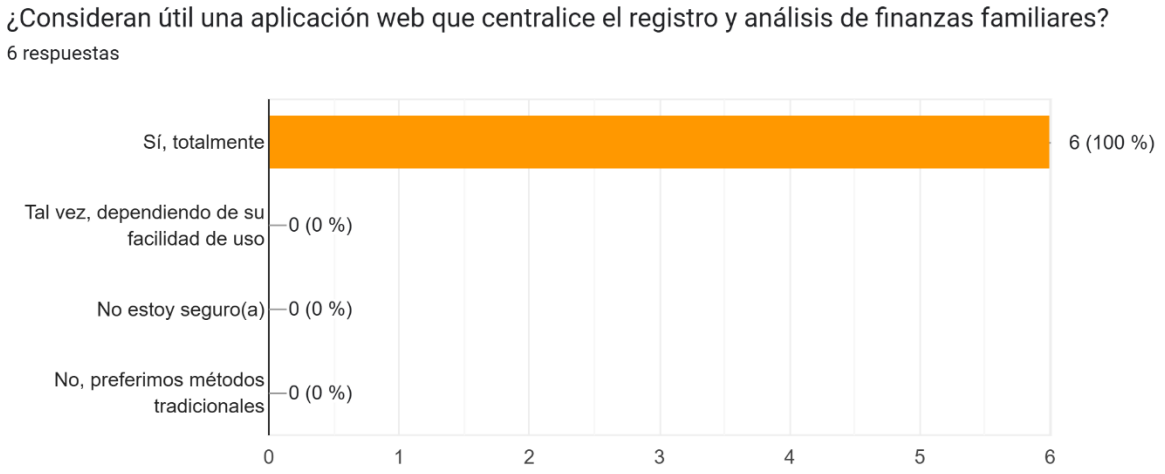
Formulario pregunta 4



Fuente: Elaboración propia con Google Forms.

Figura 21

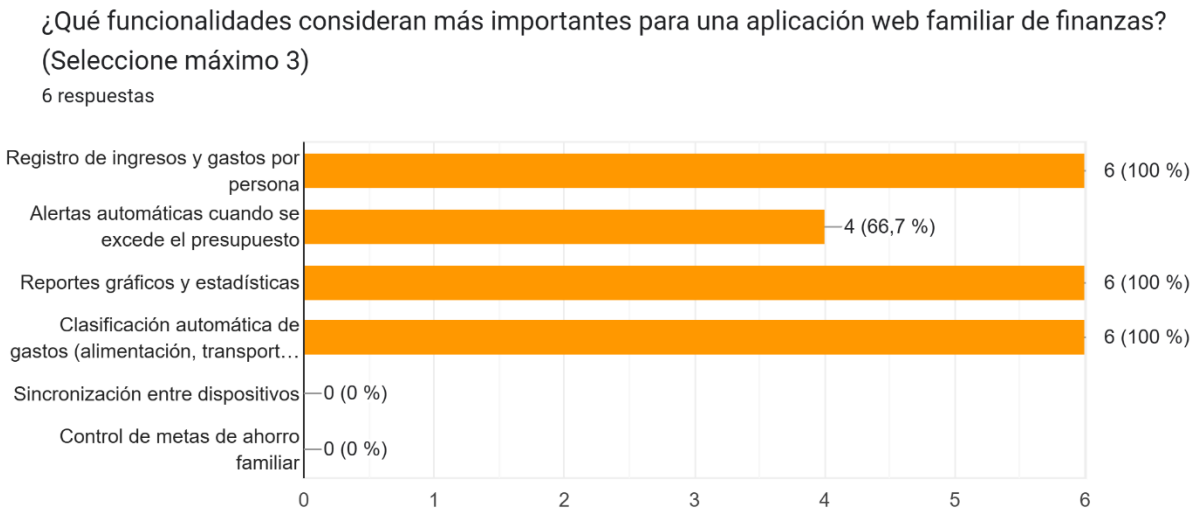
Formulario pregunta 5



Fuente: Elaboración propia con Google Forms.

Figura 22

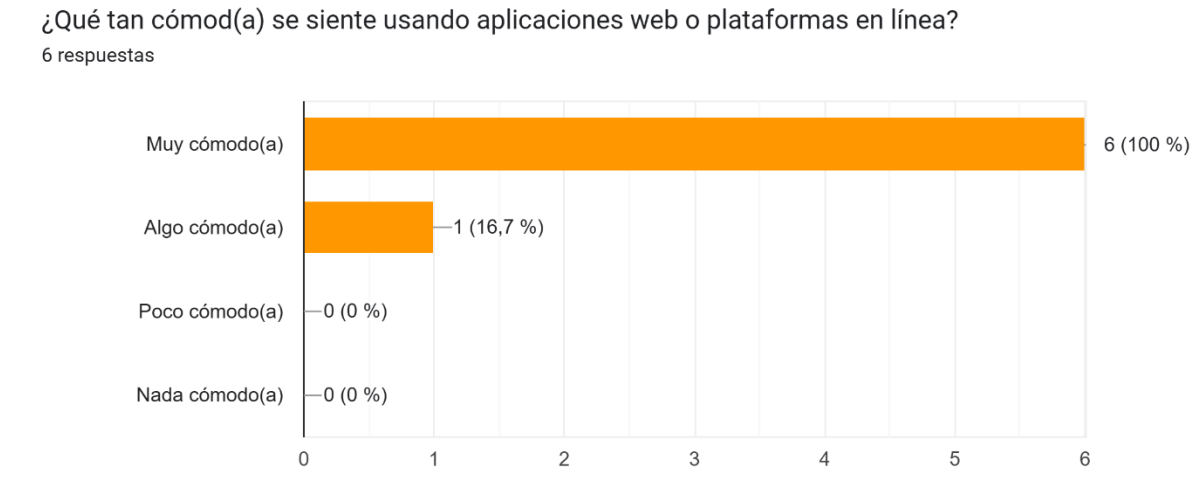
Formulario pregunta 6



Fuente: Elaboración propia con Google Forms.

Figura 23

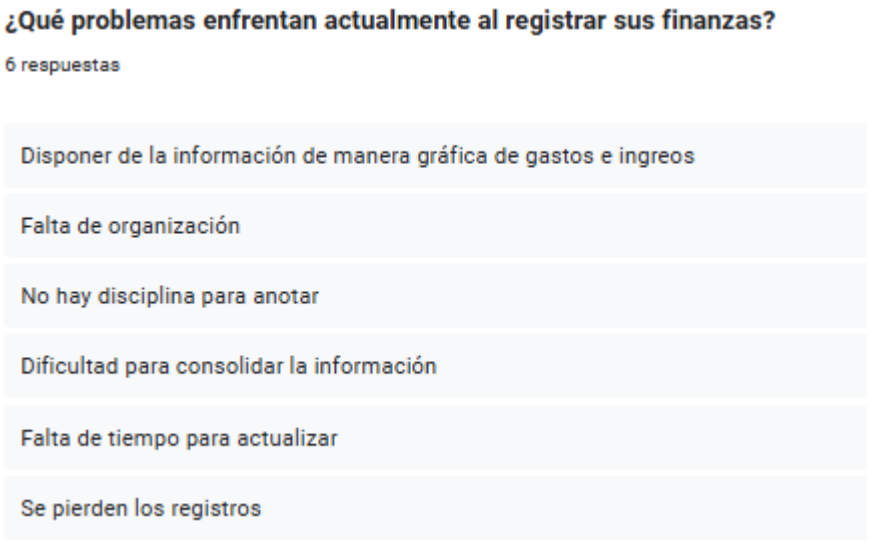
Formulario pregunta 7



Fuente: Elaboración propia con Google Forms.

Figura 24

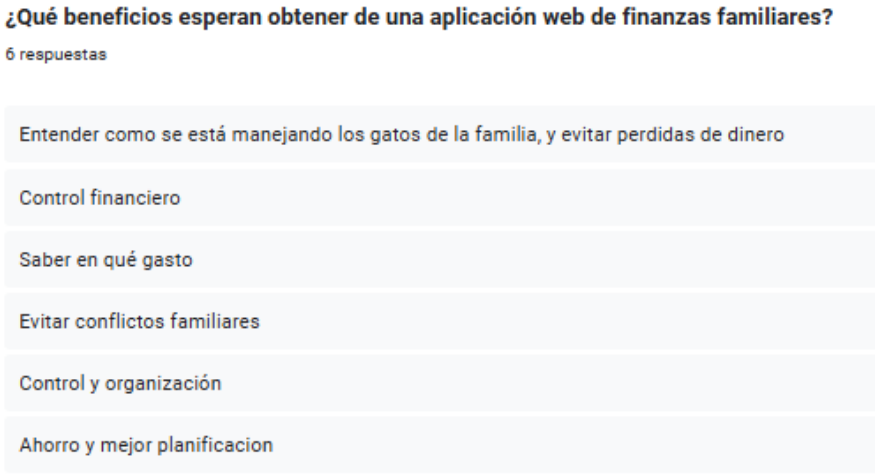
Formulario pregunta 8



Fuente: Elaboración propia con Google Forms.

Figura 25

Formulario pregunta 9



Fuente: Elaboración propia con Google Forms.

Conclusión

Se espera que el proyecto WalletMoneyFamily entregue un Producto Mínimo Viable (MVP) funcional que abordará la necesidad de la familia Aldana Gómez de organizar la gestión financiera del hogar. Mediante la implementación de una aplicación web con módulos de autenticación y registro/seguimiento de ingresos y gastos, el proyecto cumplirá con su objetivo general de centralizar el registro, monitoreo y análisis de las finanzas

familiares. Los objetivos específicos; analizar la situación financiera actual, diseñar la arquitectura y la interfaz del sistema, desarrollar los módulos principales y validar mediante pruebas y despliegue, se lograrán dentro del plazo de 6 a 8 semanas. La metodología ágil Kanban, gestionada en un tablero Trello con columnas To Do, In Progress y Done, asegurará una gestión eficiente de tareas y un seguimiento claro del progreso, alineado con las fases del Ciclo de Vida del Desarrollo de Software (SDLC): Planeación, Análisis, Diseño, Desarrollo, Pruebas, Despliegue y Documentación. La aplicación, que se desarrollará con Flask para el backend, PostgreSQL para la base de datos, Bootstrap para una interfaz responsiva y Chart.js para reportes gráficos, cumplirá con los requerimientos funcionales (autenticación, registro de datos y generación de reportes) y no funcionales (tiempo de respuesta <=2 segundos, interfaz responsiva, uptime >99%, contraseñas con hash), los cuales se validarán mediante pruebas unitarias, integrales, E2E, aceptación , pruebas funcionales y no funcionales. El presupuesto para la familia debe ser de 1859 USD, mientras que el presupuesto para el desarrollador junior se mantendrá en ~25 USD gracias al uso de herramientas gratuitas o de bajo precio como Flask, PostgreSQL, Trello, GitHub y Render. La encuesta en Google Forms orientará el diseño de la aplicación, priorizando la transparencia y los reportes gráficos para garantizar usabilidad y claridad, según las necesidades identificadas de la familia Aldana Gómez. A corto plazo, el MVP resolverá la desorganización financiera inmediata; a mediano plazo (3-6 meses tras el desarrollo), se espera incorporar funcionalidades como exportación de reportes a PDF o filtros avanzados para mejorar la toma de decisiones; a largo plazo (6-12 meses), la integración con aplicaciones de productividad o el soporte para familias extendidas podría potenciar la escalabilidad. WalletMoneyFamily no solo proporcionará a la familia Aldana Gómez una herramienta para reducir conflictos y mejorar la planificación financiera, respaldado por el tablero Trello, el repositorio GitHub y un informe en formato APA, demostrando una gestión efectiva del proyecto y una ejecución técnica sólida.

Referencias Bibliográficas

Aldana Gómez, R. F. (2025). *WalletMoneyFamily: Aplicación web para control de gastos familiares* (Proyecto de software). Corporación Universitaria Iberoamericana, Facultad de Ingeniería.

Báez Pérez, C. I., & Suárez Zarabanda, M. I. (2013). *Proceso de desarrollo de software: basado en la articulación de RUP y CMMI priorizando su calidad* (Cap. 1, pp. 11–16). Universidad de Boyacá.

Gual Ortí, J. (2016). *Fundamentos del modelado y prototipado virtual en el diseño de productos* (Caps. 2 y 3). Universitat Jaume I.

Omaña, M. (2012). *Manufactura esbelta: una contribución para el desarrollo de software con calidad* (pp. 14–18). Red Enlace.

Pressman, R. S. (2021). *Ingeniería de software* (Caps. 24–26, pp. 490–548). McGraw-Hill Interamericana.

Sommerville, I. (2005). *Ingeniería del software*. Pearson Educación.

Suárez, E. C. (2017). *Prototipo, contexto e ingeniería del software* (pp. 2–20). Estudios de Postgrado en Sistemas de Información.