GESTHOR

GesThor Git

Nome Alumno/a: **Laureano De Sousa Dias**

Curso: ***2º DAM***

Módulo: ***Proxecto Final Ciclo***

Contido

1. Introducción

GesThor é unha aplicación de xestión de gastos persoais e empresariais que facilita o rexistro, análise e optimización financeira. Destaca polo seu uso de OCR (Reconocimiento Óptico de Caracteres) para extraer datos de tickets e facturas automaticamente, ademais da integración dunha IA que proporciona recomendacións de aforro e comparación de prezos. Tamén inclúe funcionalidades para a xestión de subscricións e a xeración de informes en PDF.

2. Obxectivos

Obxectivo iniciais:

- Desenvolver unha plataforma que permita aos usuarios rexistrar os seus gastos de forma automática ou manual.
- Implementar un sistema de escaneo de tickets mediante OCR.
- Integrar unha IA que analice os hábitos de gasto e propoña recomendacións.
- Ofrecer unha interface cómoda e intuitiva, compatible con múltiples dispositivos.
- Garantir a seguridade dos datos mediante autenticación segura e cifrado.
- Permitir a descarga e compartición de informes en PDF ou CSV.

Obxetivos acadados (11/04/2015):

- Rexistro e xestión de gastos mediante escaneo de tickets físicos, Tickets dixitais ou entrada manual.
- Clasificación manual dos gastos por categorías (No incluido).
- Chatbot IA para suxestións e asesoramento financeiro (Limitado).
- Posibilidade de sincronización multi-dispositivo mediante API segura (Seguridad operativa).
- Compatibilidade con sistemas de autenticación externa como Google (Funcionando)

3. Situación previa

GesThor nace para resolver las dificultades que enfrentan muchos usuarios al organizar sus gastos debido a la falta de tiempo. El proceso manual de registro de facturas y tickets es no solo tedioso, sino también propenso a errores. Aunque existen aplicaciones de contabilidad, muchas de ellas carecen de funcionalidades avanzadas como el OCR o asistentes basados en inteligencia artificial.

GesThor ofrece una plataforma intuitiva, segura y escalable que facilita el registro de gastos mediante el escaneo de tickets, la carga de tickets digitales o la entrada manual de datos. Además, proporciona gráficos detallados e informes interactivos para que los usuarios puedan comparar y analizar sus gastos, filtrarlos y realizar un seguimiento eficaz.

En el futuro, se incorporarán funcionalidades adicionales como el escaneo de facturas electrónicas, lo que permitirá una mayor automatización del proceso, mejorando aún más la experiencia del usuario y la eficiencia del sistema.

4. Tecnoloxías empregadas

Backend (Servidor Java - Spring Boot)

- **Spring Boot::**

  Crea a API REST, implementa a lóxica de negocio e xestiona as dependencias do proxecto.

- **Spring Security:**

  Proporciona autenticación e autorización para protexer a aplicación.

- **JWT (JSON Web Token):**

  Xestiona a autenticación dos usuarios mediante tokens seguros e sen estado.

- **OAuth2 (Google y GitHub):**

  Permite o inicio de sesión usando contas de Google.

- **MySQL:**

  Base de datos relacionaL para almacenar a información da aplicación.

- **OpenAPI (Swagger):**

  Documenta a API REST e facilita probas e integración con outras ferramentas.

- **Cliente de conexión Java-Python:**

  Permite a comunicación entre o backend Java e servizos Python.

- **BCrypt:**

  Encripta datos sensibles (como contrasinais) para almacenamento/transmisión segura.

- **Lombok:**

  Evita escribir código repetitivo como constructores, getters e setters.

- **PlantUML:**

  Convierte archivos .puml .al graficos de documentación

- **WiX Toolset:**

  Convertir .jar a ejecutables instalables .exe

Backend (Servidor Python - OCR)

- **FlaskAPI:**

  Framework para crear APIs REST de alto rendemento, especialmente útil para procesos como o OCR.

- **Tesseract OCR:**

  Extrae texto de imaxes de tickets e facturas, permitindo a extracción automática de datos.

- **pdf2image:**

  Converte arquivos PDF a imaxes para facilitar o seu procesamento mediante OCR.

- **OpenCV:**

  Mellora e procesa imaxes para optimizar a extracción de datos usando técnicas de procesamento dixital.

- **NumPy:**

  Facilita o procesado de datos numéricos e imaxes de forma eficiente, fundamental para operacións con arrays e cálculos matemáticos.

- **Requests:**

  Realiza chamadas HTTP para comunicarse con outras APIs (como a de Spring Boot), integrando servizos de forma fluída.

- **flasgger(Eliminado):**

  Documentación de endpoints (Disponibles en readmi y documentación tecnica.)

- **Sphinx:**

  Ferramenta para documentar a API REST, xerando documentación automaticamente a partir dos docstrings, o que facilita as probas e a integración.

- **openai (Compatible 0.28.0):**

  Libreia que enlaza con servidores de opienAI para uso de IA

- **python-dotenv:**

  Carga de variables de sistema o .env de forma dinamica.

Frontend (Angular - Express)

- **Express:**

  Lanza el servidor usando el proyecto compilado.

- **Angular:**

  Desenvolve a interface web dinámica e interactiva para os usuarios.

- **Bootstrap:**

  Proporciona un deseño responsivo e atractivo para a UI, mellorando a experiencia visual.

- **TypeScript:**

  Versión tipada de JavaScript que mellora a robustez, mantenibilidade e escalabilidade do código en Angular.

- **Chart.js / ng2-charts:**

  Empregado para a xeración de gráficos interactivos e visualizacións de datos.

- **Angular CLI:**

  Facilita a creación de proxectos, desenvolvemento, xeración de compoñentes e tarefas comúns en Angular.

- **jspdf:**

  Permite generar pdf aprtir de datos de la página.

Almacenamento

- **Almacenamento interno:**

  A base de datos principal será MySQL, garantindo estabilidade e compatibilidade coa aplicación. Existe a opción de migración futura a PostgreSQL segundo necesidades ou melloras no rendemento (Compatible) .

- **Almacenamento na nube (opcional):**

  Segundo o tempo dispoñible no desenvolvemento, poderá integrarse con servizos na nube (como Amazon S3) para almacenar arquivos e documentos de forma externa e segura.

Control de versións, Tests e implementación

- **GitHub:**

  Xestión do código fonte, control de versións e colaboración entre desenvolvedores.

- **Postman:**

  Proba e documenta os endpoints da API de forma interactiva, facilitando a integración e o desenvolvemento backend-frontend.

IDEs

- **InteliJ (Servidor Spring):**

  IDE que integra ferramentas apra traballar con entornos java.

- **VS-Code (Servidor angular):**

  IDE versatil e ampliable , rapido elixeiro compatible con node e extensions de angualr e bootstrap.

- **Trae (Servidor Python):**

  IDE como VS-Code , con integración de IA perfecto apra axuda na xeración de codigo.

- **Phenyx(Archivos HTML e Mardown):**

  IDE moderno parente de Brackets.

Herramientas adicionais:

- **ChatGPT:**

  Empregado para a xeración de documentación, explicacións técnicas e apoio ao desenvolvemento.

- **DeepSeek:**

  Utilizado para xerar datos de proba e comentar código de forma automática.

- **ContentCore:**

  Aplicación para creación de maquillaxes ou modificacións visuais en imaxes mediante IA.

- **Canvas:**

  Ferramenta para a creación de gráficos e visualización de datos de forma sinxela.

- **Figma:**

  Plataforma colaborativa para deseño de interfaces e creación de prototipos interactivos.

- **Responsively.app:**

  Aplicación para deseño e previsualización de interfaces responsivas en múltiples dispositivos.

- **ASCII Art Generator (ascii-art-generator.org)**

  Xera arte en texto a partir de imaxes ou palabras, útil para elementos creativos en documentación ou presentacións.

- **Lucid: https://help.lucid.co/**

  Ferramenta para desenar graficos.

- **Google Drive**

  Almacenamiento de arquivos de instalación e medios .

5. Solución proposta

Solución proposta GesThor proporciona unha plataforma de xestión de gastos con tecnoloxías avanzadas de recoñecemento de texto e intelixencia artificial. As funcionalidades principais inclúen:

- Escaneo de tickets dixitais e físicos con OCR para a extracción automática de datos.
- Entrada manual de gastos para correccións ou rexistros personalizados.
- Chatbot integrado que ofrece recomendacións financeiras.
- Xestión de subscricións e gastos recorrentes.
- Xeración de informes en PDF con detalles financeiros.
- Autenticación segura mediante JWT e compatibilidade con Google.
- Escalabilidade grazas a Spring Boot no backend e Python para o análise avanzado de datos.

Fluxo de funcionamento O cliente inicia sesión na páxina ou aplicación, obtendo un token de acceso. Con este token pode realizar peticións á API en Spring Boot, que se encarga de servir e gardar os datos. Ademais, haberá un servidor Flask/FastAPI que utilizará unha libraría OCR optimizada en Python.

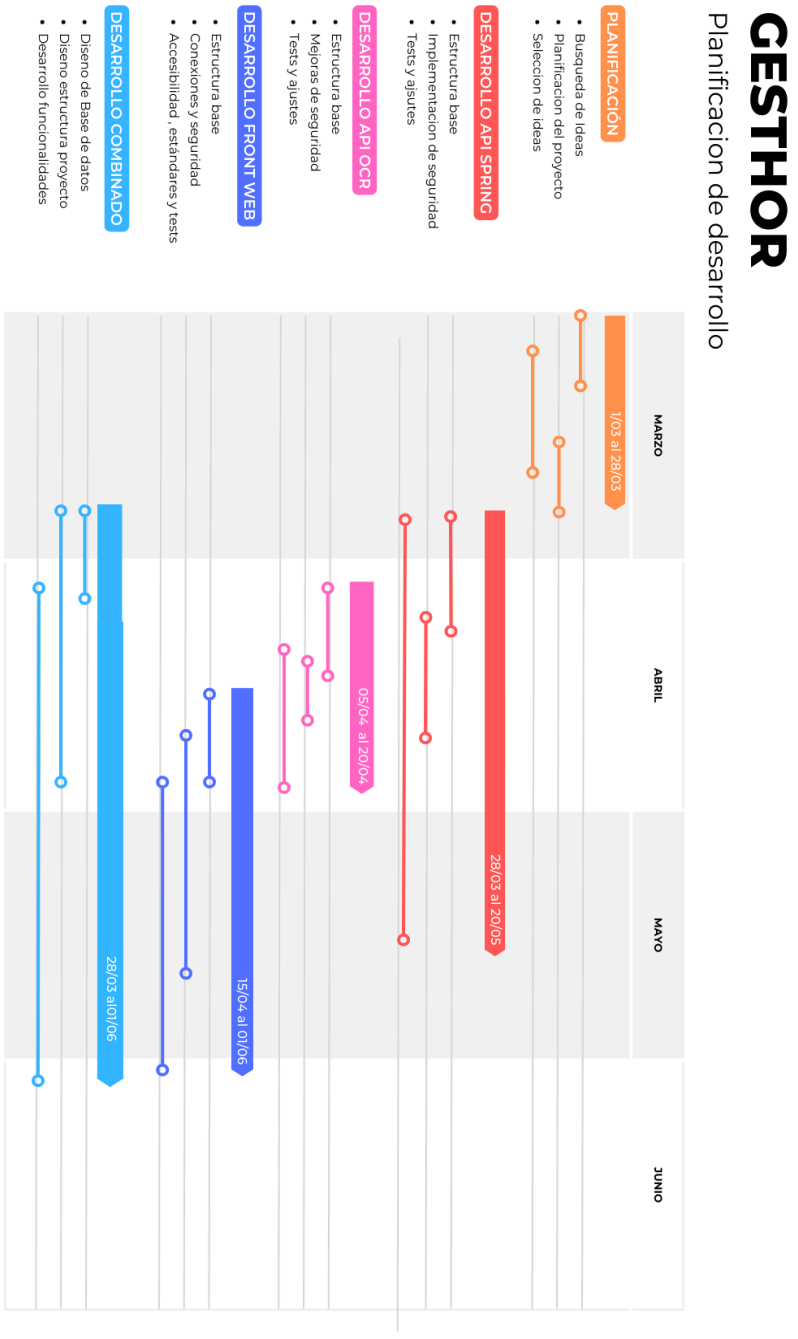
O proceso de carga de tickets funciona da seguinte maneira:

1. O cliente carga un ticket, que se envía ao servidor Spring Boot.
2. Spring Boot almacena temporalmente o ticket e o remite ao servidor Python.
3. O servidor Python realiza o proceso de OCR, extrae os datos e os devolve a Spring Boot.
4. Spring Boot preenche o ticket con estes datos e envía unha vista previa ao cliente.
5. O cliente pode aceptar, modificar ou descartar os datos extraídos.
6. Se os acepta, gárdanse a imaxe precargada e os datos definitivos; en caso contrario, descártase todo.

A comunicación entre Spring Boot e o servidor Python será mediante endpoints internos para garantir eficiencia e seguridade con claves e cifrados (Si funciona e da tempo).

6. Planificación do proxecto

Diagrama de Gantt



Planificación basada en el modelo de desarrollo en cascada modificado con enfoque incremental.

Cada bloque representa una fase o módulo funcional (OCR, backend, frontend, etc.), desarrollada de forma progresiva. Se permite el solapamiento entre fases para validar entregas parciales y facilitar la integración continua.

Descripción de modelo de desenvolvemento de software a implementar.

No desenvolvemento da aplicación **GesThor**, optarase por un **modelo en cascada modificado con enfoque incremental**. Este modelo combina a estrutura clásica e secuencial do modelo en cascada cunha metodoloxía máis flexible, permitindo realizar entregas parciais e validar módulos de forma progresiva ao longo do proxecto.

O modelo en cascada tradicional establece unha serie de fases ben definidas (análise de requisitos, deseño, implementación, probas e mantemento) que se executan de forma secuencial. Con todo, no contexto deste proxecto, aplicouse unha versión **modificada**, onde se permite o **solapamento entre fases** e a **iteración parcial**, facilitando a revisión e mellora continua.

Complementariamente, engádese un **enfoque incremental**, que consiste en dividir o sistema en diferentes módulos ou funcionalidades (incrementos), desenvolvéndoos e integrándoos por separado. No caso de GesThor, isto reflíctese no desenvolvemento individual de partes como:

- O módulo de recoñecemento óptico de caracteres (OCR),
- O backend baseado en Spring Boot,
- A interface gráfica en Angular ou Python,
- E os mecanismos de análise e almacenamento de datos.

Cada un destes compoñentes será desenvolvido, probado e validado antes de integrar o seguinte, permitindo detectar erros anticipadamente e obter versións funcionais parciais ao longo do proceso.

Xustificacion de aleccion:

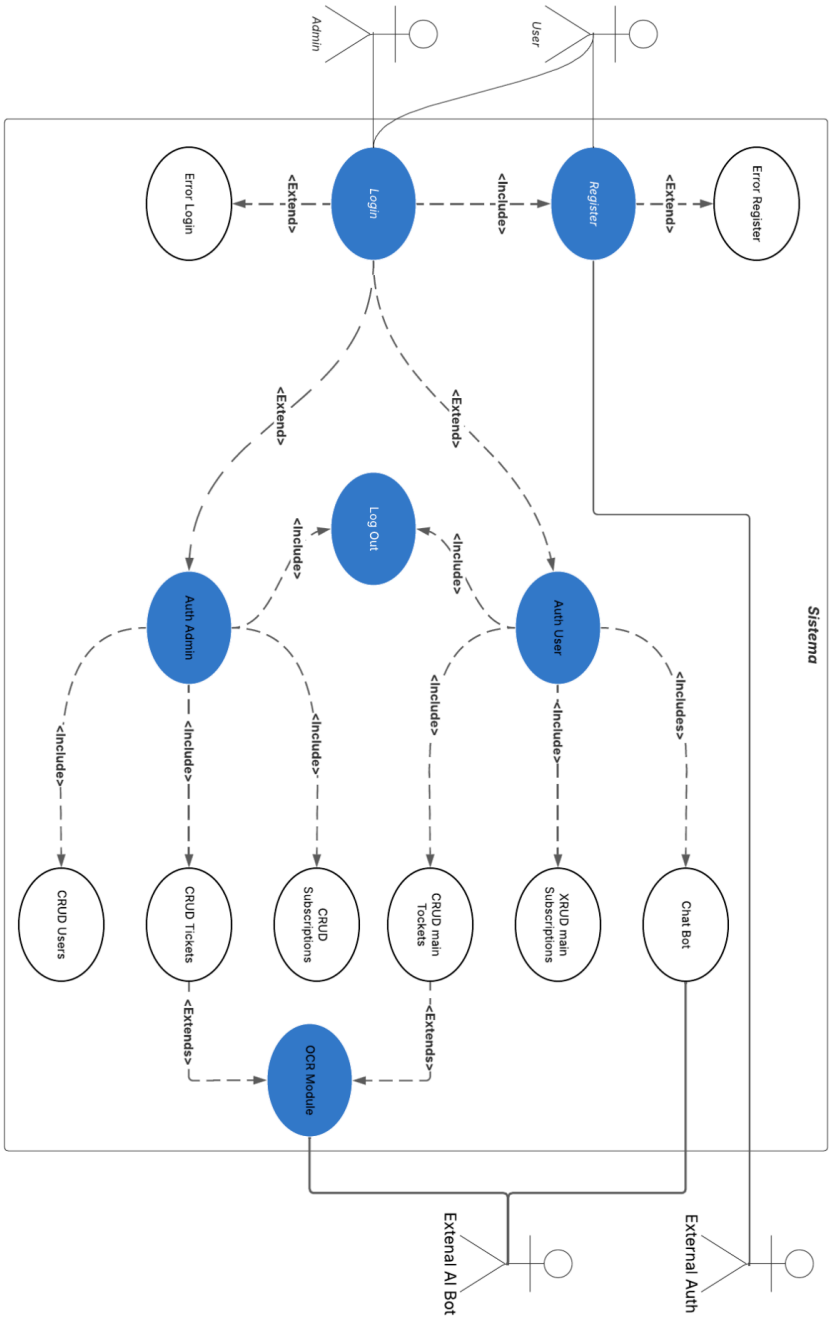
A elección deste modelo de desenvolvemento baséase nas seguintes razóns:

- **Adaptación ao contexto educativo:** O proxecto conta cun prazo delimitado e uns requisitos definidos dende o inicio, polo que un modelo estruturado axuda a organizar as tarefas.
- **Claridade e planificación:** A separación por fases facilita o seguimento do progreso e o cumprimento dos obxectivos parciais.
- **Redución de riscos:** O desenvolvemento incremental permite detectar fallos en fases temperás e realizar axustes antes de continuar coa seguinte parte.
- **Validación continua:** É posible comprobar que cada módulo cumpre os seus requisitos funcionais antes de pasar á integración xeral.
- **Facilidade de documentación:** Ao tratarse dun modelo estruturado, resulta máis sinxelo elaborar os documentos técnicos correspondentes a cada fase.

En definitiva, este modelo proporciona un **equilibrio entre estrutura, control e flexibilidade**, o que o converte na opción máis axeitada para garantir a calidade e a viabilidade do proxecto **GesThor**.

Analise do proxecto

Diagrama de casos de usos.



Descripción de Caso de Uso (Login)

UC-001	Autenticación de Usuario mediante OAuth2 o Login		
Descripción	Permite a un usuario iniciar sesión en el sistema utilizando un proveedor externo de OAuth2 (Google, Facebook, etc.). El sistema validará las credenciales y otorgará acceso si la autenticación es exitosa.		
Actores	Usuario no autenticado Proveedor OAuth2 (Google, Facebook, etc.) Sistema		
Pre condiciones	El usuario tiene una cuenta válida en el proveedor OAuth2. El sistema está configurado para aceptar autenticación con el proveedor seleccionado. La conexión a internet está activa.		
Post condiciones	El usuario queda autenticado en el sistema. Se crea una sesión activa asociada al usuario. Se registra el intento de login (éxito/fracaso) en logs.		
Secuencia Normal	#	Acción	Reacción
	1	Usuario hace clic en "Login con OAuth2"	Muestra lista de proveedores disponibles (Google,GitHub, etc.).
	2	Usuario selecciona un proveedor (ej: Google)	Redirige al usuario a la página de autenticación del proveedor.
	3	Proveedor valida credenciales y devuelve token OAuth2	Recibe el token, lo valida y extrae datos del usuario (email, nombre, etc.).
	.4		Verifica si el usuario existe en la base de datos. Si no existe, lo registra.
	5		Inicia sesión y redirige al usuario a la página principal.
Excepciones	#	Acción	Reacción
	p	El proveedor OAuth2 no está disponible	Mostrar mensaje: "Servicio temporalmente no disponible. Use otro método de login."
	q	El usuario rechaza permisos solicitados	Cancelar flujo y redirigir a página de login con mensaje: "Se requieren permisos para continuar."
Rendimiento	El sistema deberá realizar las acciones descritas en los pasos 1 al 5 en un máximo de 5 segundos.		
Frecuencia	Este caso de uso se espera que se lleve a cabo una media de 500 veces al día.		
Importancia	{Vital (el sistema no permite interacción sin autenticación).		
Urgencia	Inmediatamente (requerimiento central del proyecto).		
Comentarios	Se recomienda implementar CAPTCHA en caso de múltiples intentos fallidos. Los tokens OAuth2 deben tener un tiempo de expiración configurado (ej: 1 hora).		

Descripción de Caso de Uso (Registro)

[UC-002]	Registro de Usuario (Formulario u OAuth2)		
Descripción	Permite a un nuevo usuario crear una cuenta en el sistema, ya sea mediante formulario con email/contraseña o mediante autenticación externa (OAuth2). El sistema almacena los datos básicos y habilita el acceso.		
Actores	Usuario no registrado Proveedor OAuth2 (opcional) Sistema		
Pre condiciones	El usuario no tiene una cuenta activa en el sistema. Si usa OAuth2: el proveedor seleccionado está configurado en el sistema.		
Post condiciones	Se crea un nuevo perfil de usuario en la base de datos. El usuario recibe un email de verificación). El sistema redirige al usuario a la página de inicio o dashboard.		
Secuencia Normal	#	Acción (actor)	Reacción (sistema)
	1	Usuario hace clic en "Registrarse"	Muestra formulario con campos: email, contraseña, nombre, teléfono (opcional)
	2	Usuario ingresa datos y envía formulario	salida formato de email y fortaleza de contraseña.
	3		Verifica que el email no esté registrado previamente.
	.4		Almacena datos (contraseña encriptada) y redirige a página de "Cuenta creada".
Excepciones	#	Acción (actor)	Reacción (sistema)
	p	Error al enviar email de verificación	Registrar fallo en logs, pero permitir acceso (el usuario podrá reenviar el email).
	q	Datos de OAuth2 incompletos	Solicitar manualmente los campos faltantes (ej: teléfono para verificación).
Rendimiento	Registro con formulario: ≤ 3 segundos (pasos 1-5). Registro con OAuth2: ≤ 5 segundos (pasos 1-4).		
Frecuencia	200 veces al día		
Importancia	Vital (sin registro, no hay usuarios activos).		
Urgencia	Hay presión (requerido para la primera versión funcional).		
Comentarios	Seguridad: - Para OAuth2: validar tokens con el proveedor. UX: - Permitir registro con solo email + contraseña (el resto de datos se pueden completar luego).		

Descripción de Caso de Uso (Cargar Gasto)

[UC-003]	Registro de Gasto Automatizado mediante OCR		
Descripción	Permite al usuario subir una imagen/foto de un ticket o factura para extraer automáticamente los datos del gasto (monto, fecha, categoría, comercio) usando tecnología OCR. El sistema valida y almacena la información.		
Actores	Usuario autenticado Sistema OCR (ej: Google Vision, Tesseract) Sistema		
Pre condiciones	El usuario tiene una sesión activa. La imagen cumple requisitos (formato JPG/PNG, tamaño ≤5MB). El servicio OCR está disponible.		
Post condiciones	El gasto se registra en la base de datos con los datos extraídos. La imagen se almacena en storage (opcional). Se notifica al usuario si hubo errores en la extracción.		
Secuencia Normal	#	Acción (actor)	Reacción (sistema)
	1	Usuario selecciona "Cargar gasto con foto"	Abre cámara/dispositivo para capturar o subir imagen.
	2	Usuario sube imagen del ticket/factura	Valida formato y tamaño de la imagen.
	3		Envía imagen al servicio OCR y procesa texto crudo.
	4		Extrae datos clave (ej: monto, fecha, RFC emisor) usando expresiones regulares.
	5		Muestra previsualización con datos detectados y permite edición manual.
	6	Usuario confirma o corrige datos	Guarda el gasto en la base de datos y asocia la imagen (si aplica).
	7		Muestra confirmación: "Gasto de $[monto] en [comercio] registrado."
Excepciones	#	Acción (actor)	Reacción (sistema)
	p	Servicio OCR no responde (timeout) sistema deberá {<acción a realizar>, realizar el caso de uso [caso de uso]}	Notificar: "Servicio ocupado. Sube la imagen más tarde o ingresa los datos manualmente."
	q	Imagen con formato inválido	Mostrar: "Formato no soportado. Usa JPG o PNG (máx. 5MB)."

Rendimiento	Procesamiento OCR + extracción de datos: ≤ 10 segundos (pasos 3-5).
Frecuencia	100-300 veces al día .
Importancia	Importante (reduce fricción en registro manual de gastos).
Urgencia	Puede esperar (prioridad media, depende de recursos para integrar OCR).
Comentarios	Mejoras de precisión: - Entrenar modelo personalizado con tickets locales (ej: formatos de CFDI en México). - Usar geolocalización para inferir comercio si el OCR no lo detecta. Privacidad: - Borrar imágenes procesadas después de 30 días (solo guardar datos estructurados).

Universo de discurso da base de datos.

A aplicación “GesThor” permite aos usuarios rexistrar, xestionar e analizar os seus gastos persoais ou familiares. Os datos almacenados permiten realizar seguimento por categorías, identificar patróns, controlar subscricións e gardar tickets escaneados ou introducidos manualmente.

Entidades e atributos principais:

Usuario (usuarios)

- id: Identificador único.
- username: Nome de usuario (único).
- email: Correo electrónico (único).
- password: Contrasinal cifrada.
- name: Nome completo.
- image_url: URL da imaxe do perfil.
- active: Se o usuario está activo.
- provider: Tipo de login (LOCAL, GOOGLE, GITHUB).
- provider_id: ID do provedor externo.
- created_at, updated_at: Fechas de rexistro e modificación.

Gasto (gastos)

- spent_id: Identificador único do gasto.
- name, description: Nome e descrición do gasto.
- total: Importe total.
- iva: IVE aplicado.
- expense_date: Data do gasto.
- icon: Icona asociada (se se mostra nunha interface).
- tipo: Tipo de gasto: FACTURA, GASTO_GENERICO, SUBSCRIPCION, TICKET, TRANSFERENCIA
- categoria_id: FK → Categoria asociada.
- user_id: FK → Usuario propietario.
- created_at, updated_at: Fechas de rexistro e modificación.

Categoría (categorias)

- id: Identificador único.
- name: Nome da categoría (ex. Alimentación, Transporte...).
- description: Descrición opcional.
- iva: Porcentaxe de IVE aplicada por defecto.
- created_at, updated_at: Control de tempo.

Subscrición (subscripciones)

- spent_id: FK e PK → Referência ao gasto asociado.
- activa: Se está activa ou pausada.
- accumulate: Cantidade acumulada ao longo do tempo.
- start, end: Datas de inicio e fin.
- interval_time: Intervalo de tempo (ex. 30 días).
- restart_day: Día no que se renova.

Ticket (tickets)

- spent_id: FK e PK → Referência ao gasto asociado.
- store: Nome da tenda.
- productsjson: Lista de produtos do ticket en formato JSON.

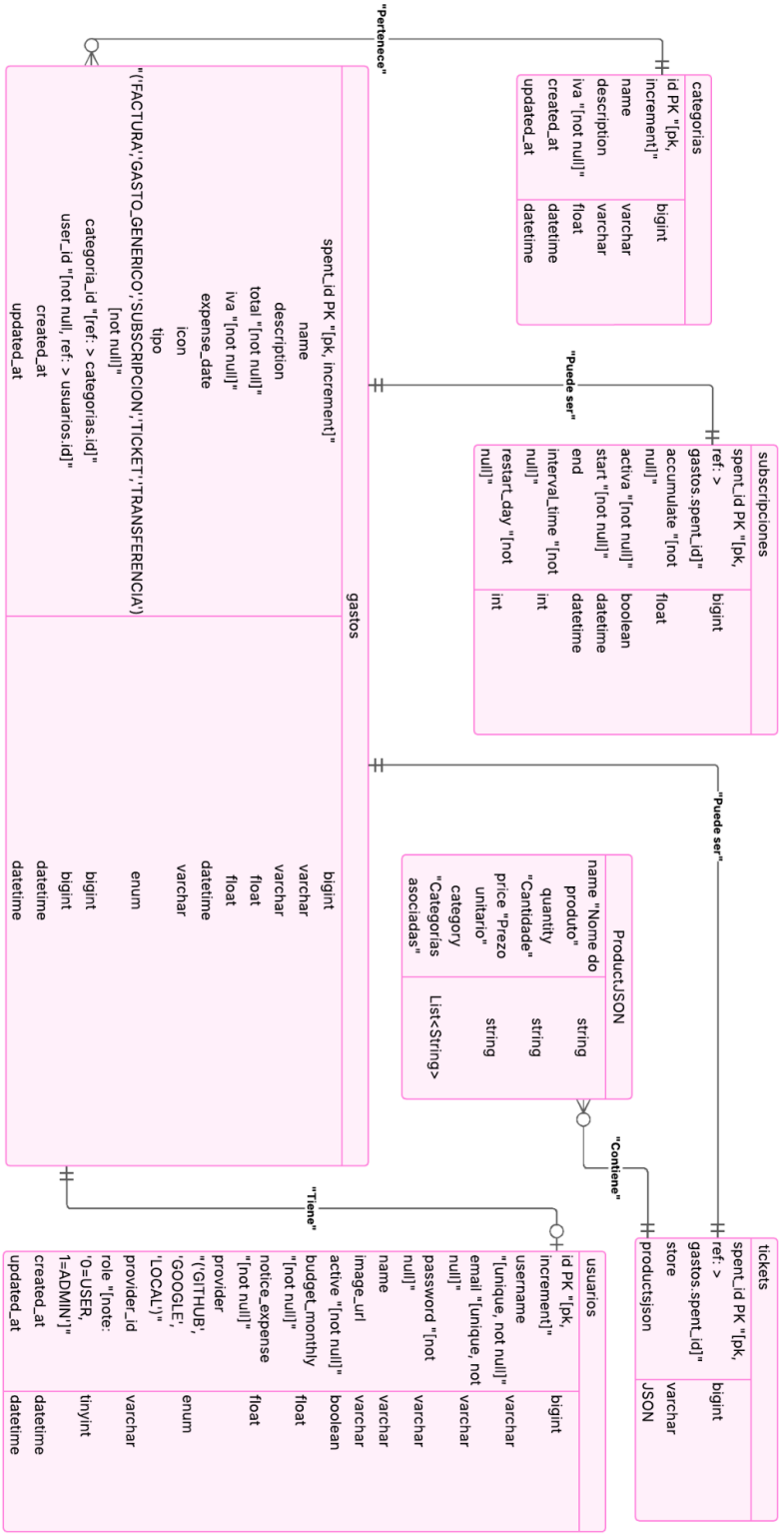
productsjson (productos):

- private String name.
- private String quantity.
- private String price.
- private List<String> category.

Relacións:

- `usuarios (1) ⟶ (N) gastos`
- `gastos (N) ⟶ (1) categorias`
- `gastos (1) ⟶ (1) subscripciones (relación 1:1 por PK compartida)`
- `gastos (1) ⟶ (1) tickets (relación 1:1 por PK compartida)`

Diagrama de Entidad-Relación.



Deseño do proxecto:

Modelo relacional da base de datos.

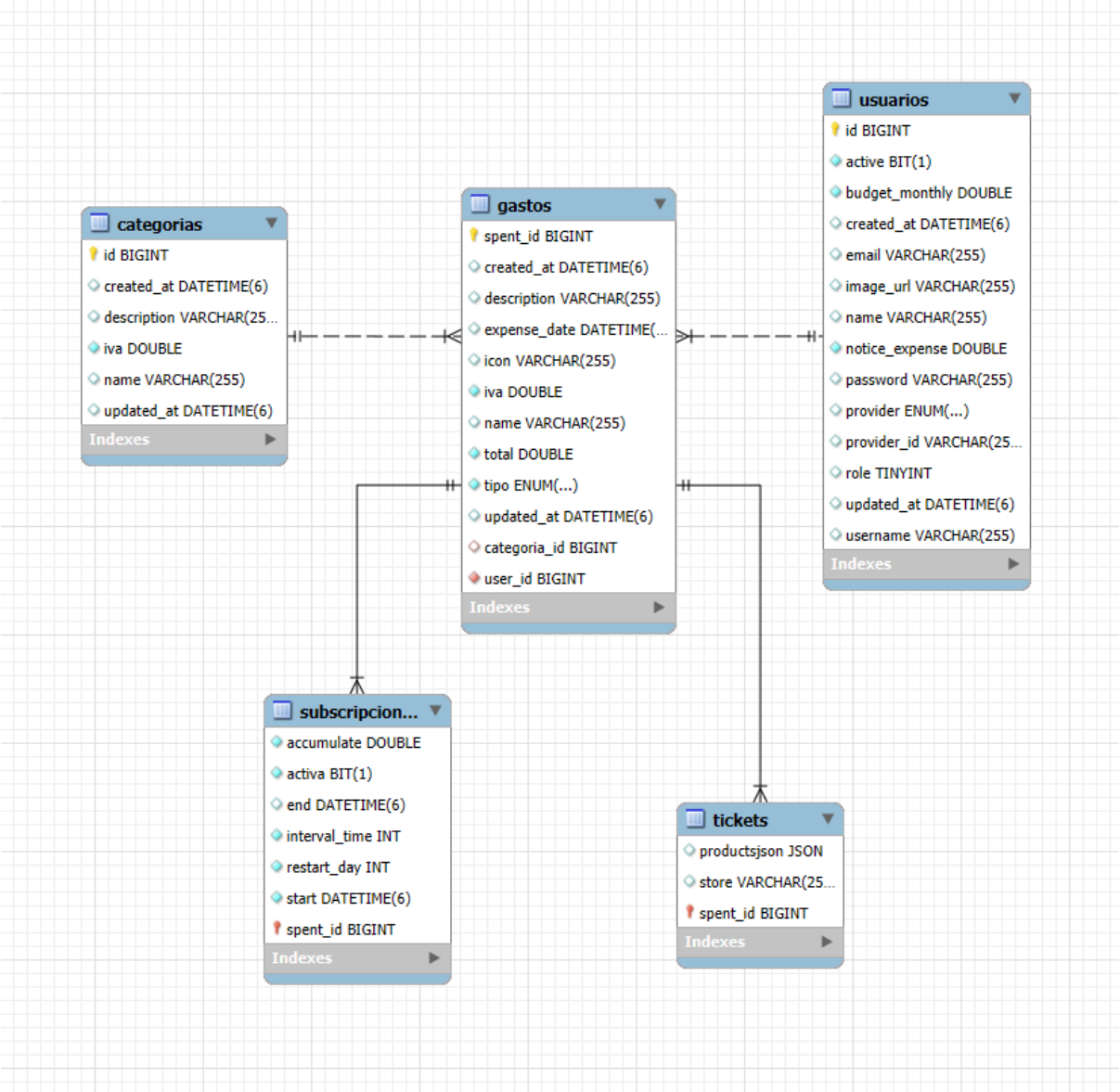
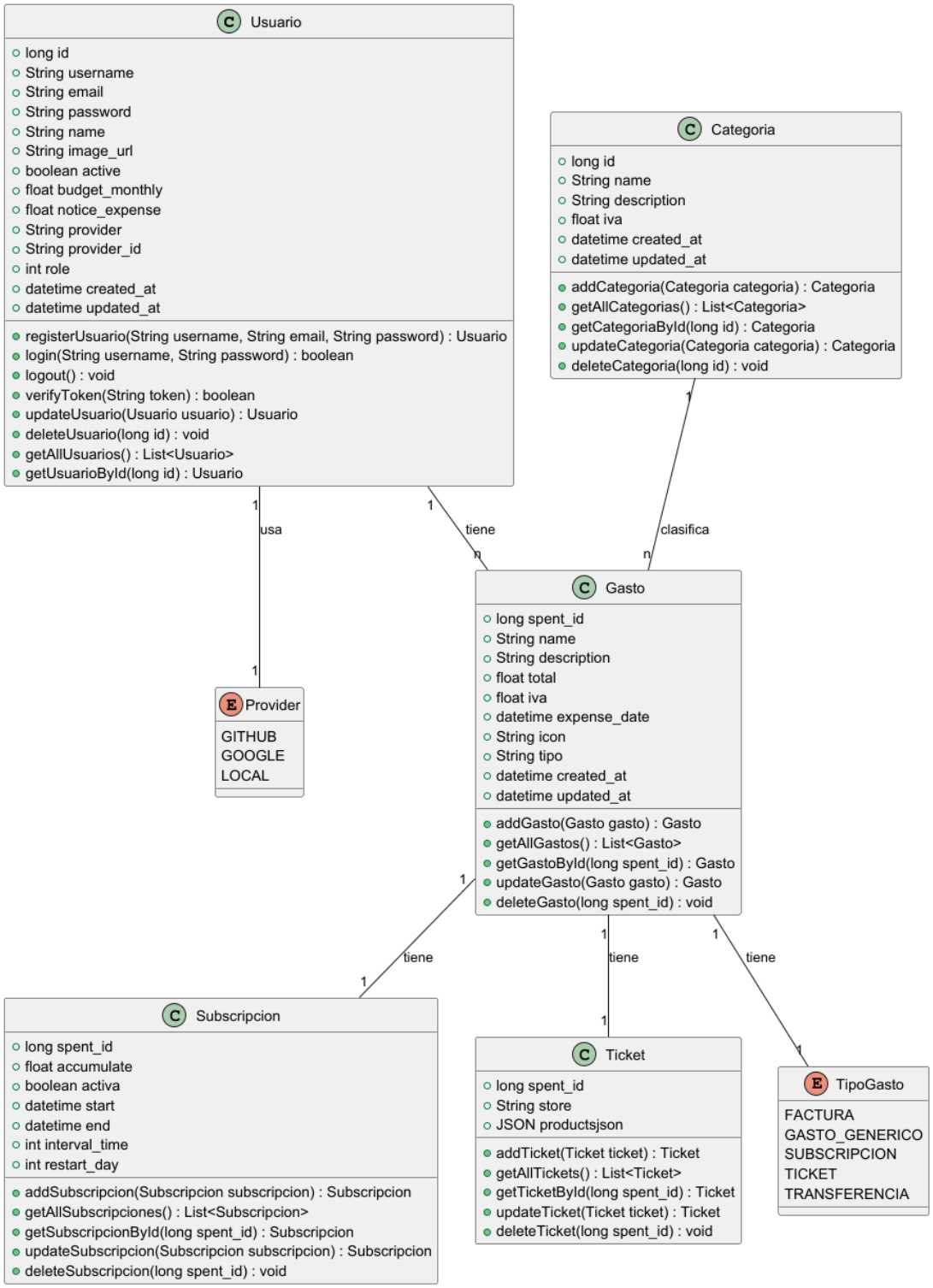
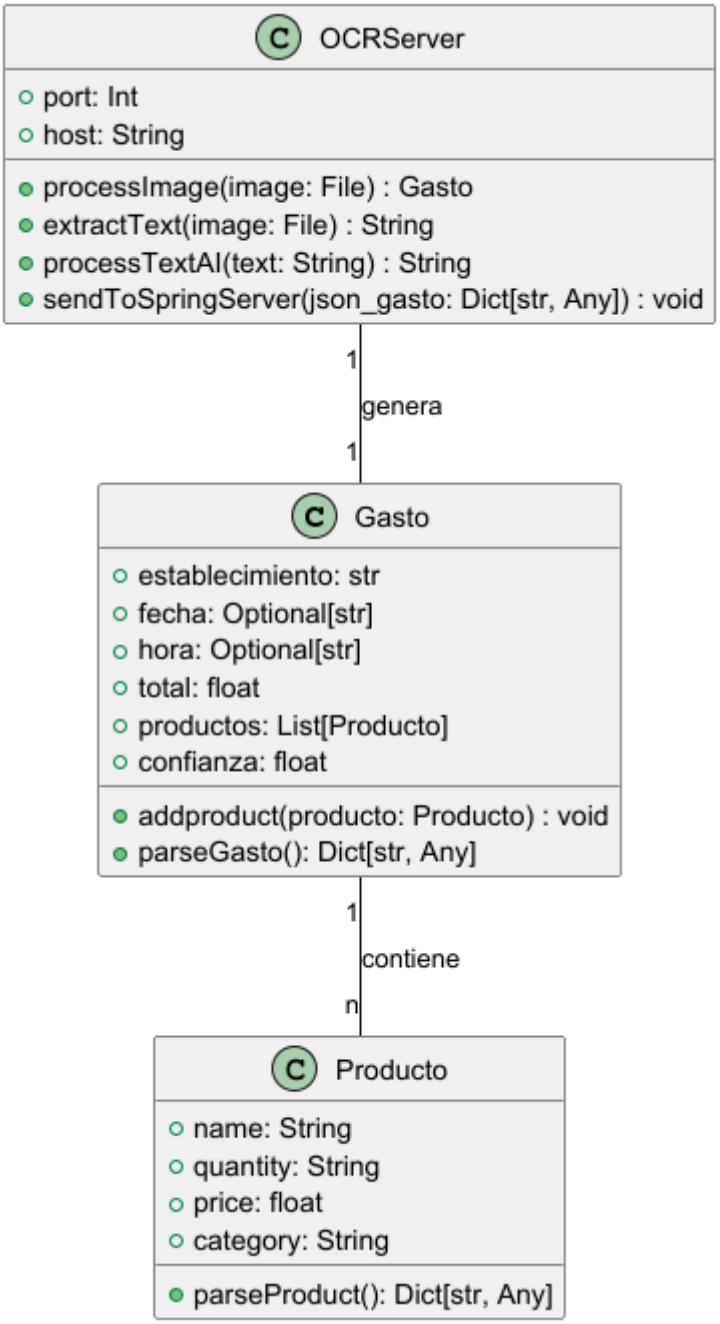


Diagrama de clases.

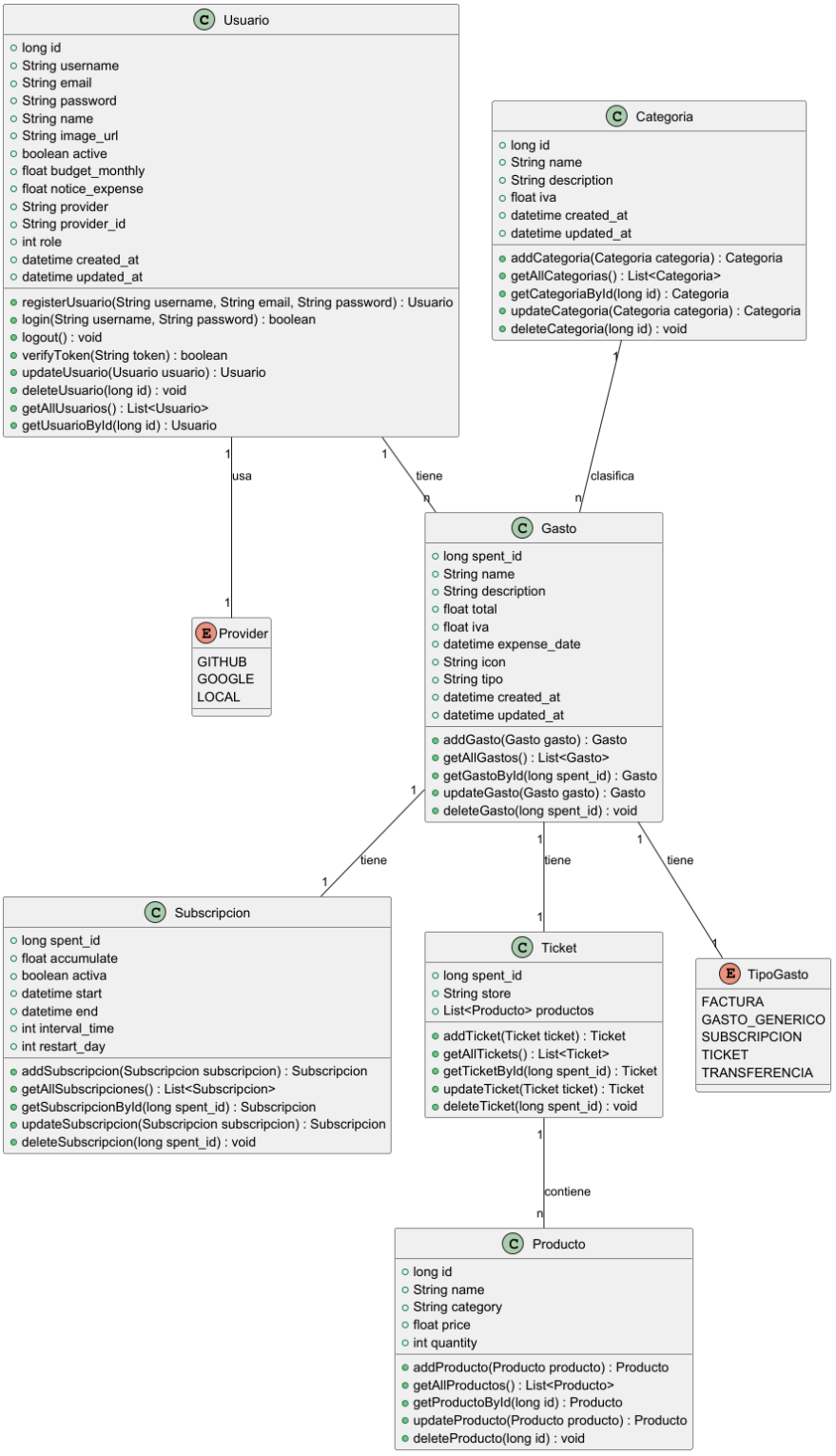
JAVA (Server Spring Boot)



Python (Server OCR APIrest)



Angular (Front Angular)



Diagramas de fluxo

Caso de uso Login y registro . (Link)

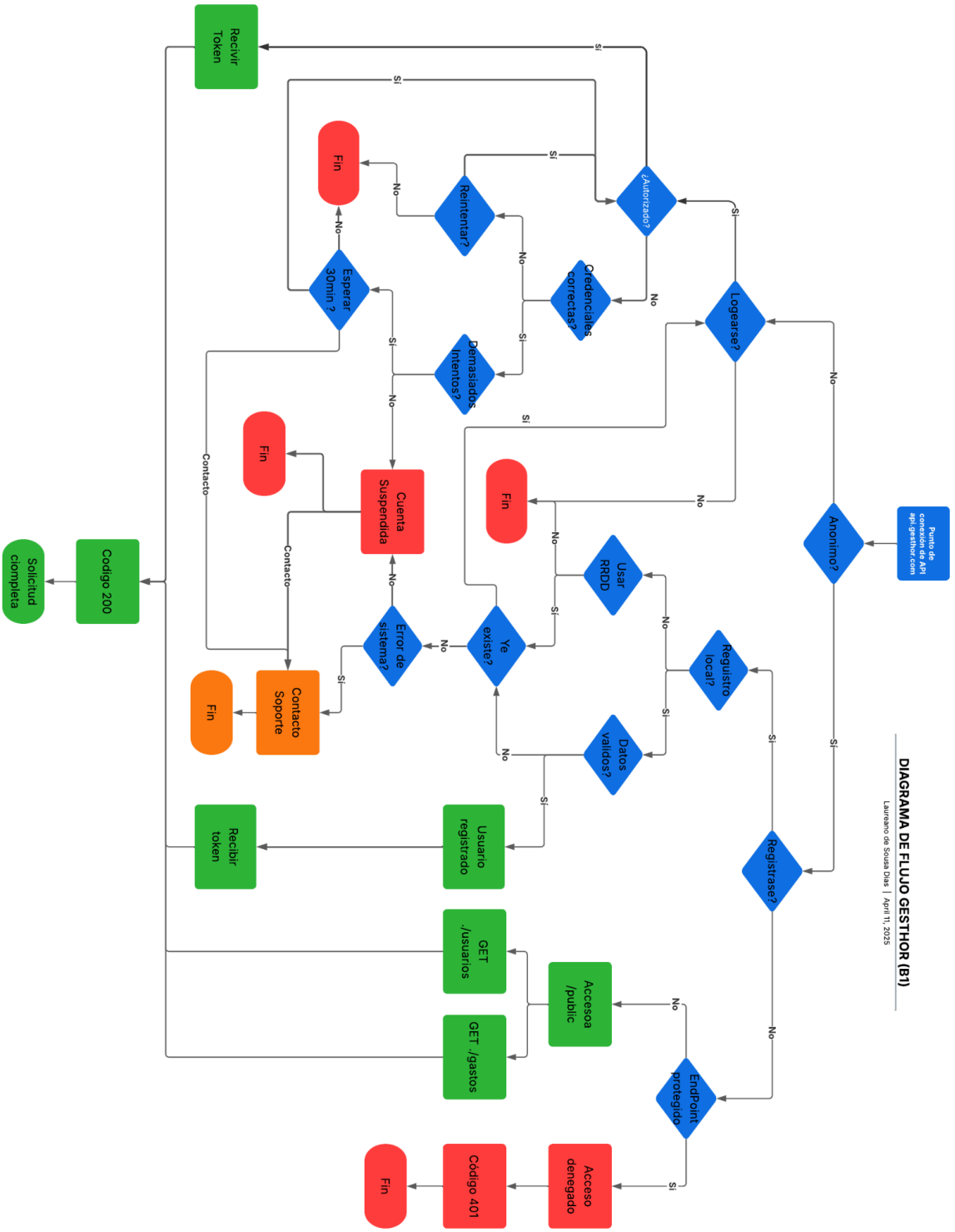
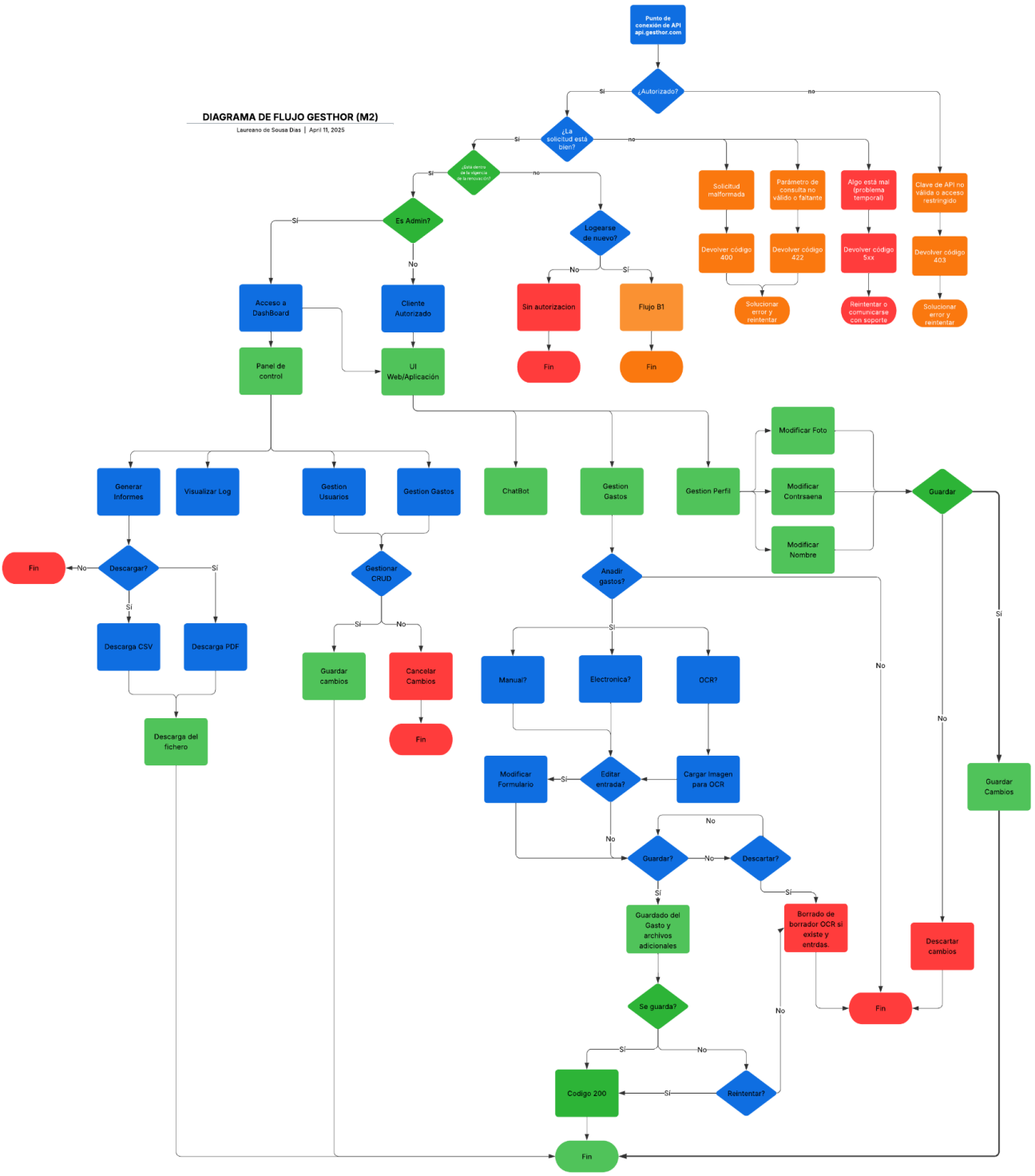
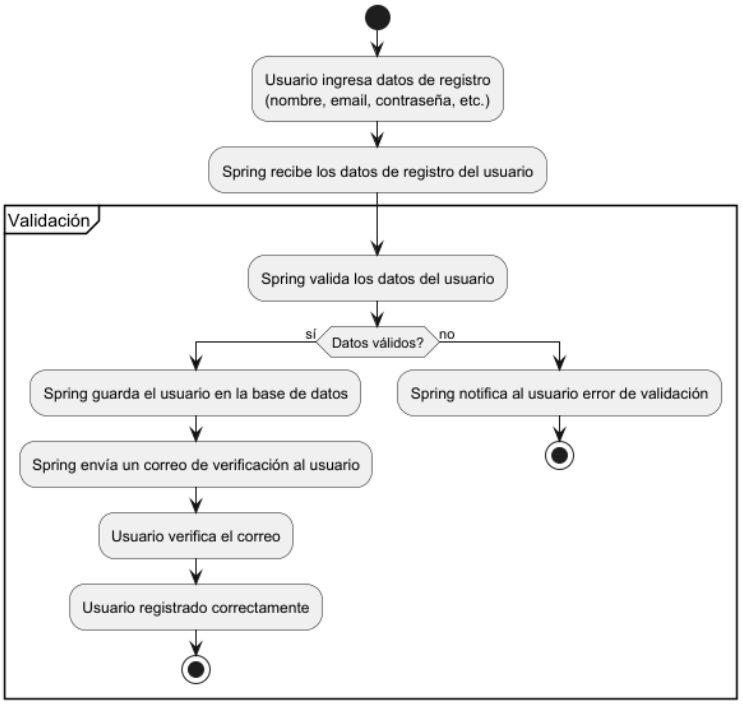


Diagrama de servidor (Link)

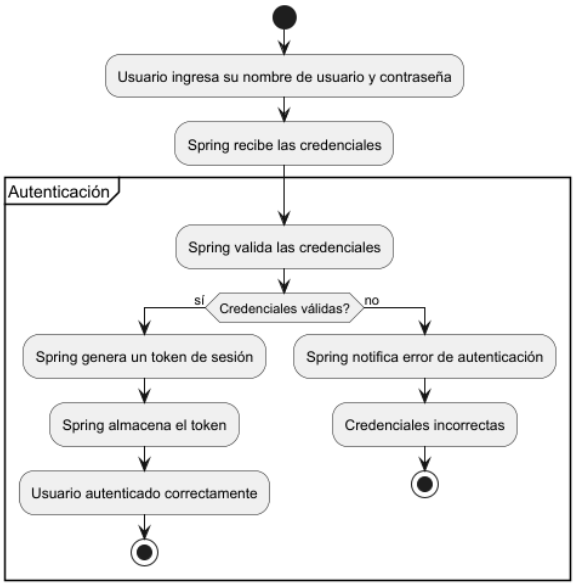


Diagrmas De fluxo complementarios

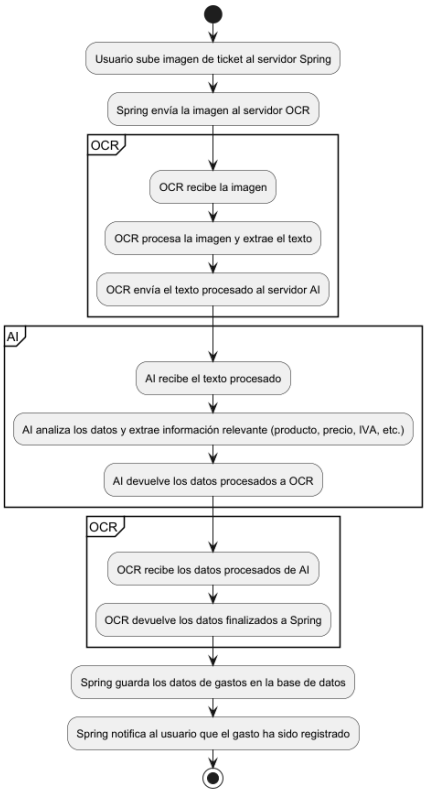
Regsitro:



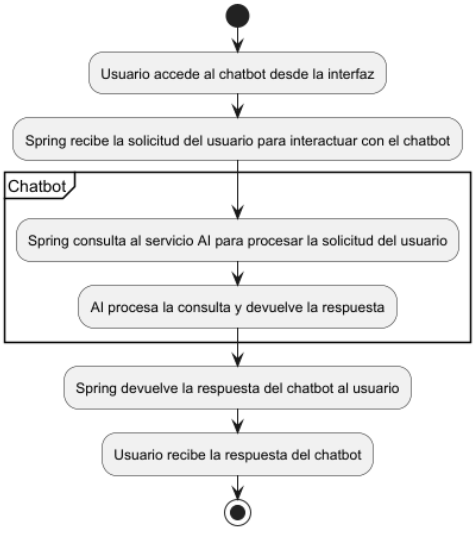
Login:



Carga de ticket:

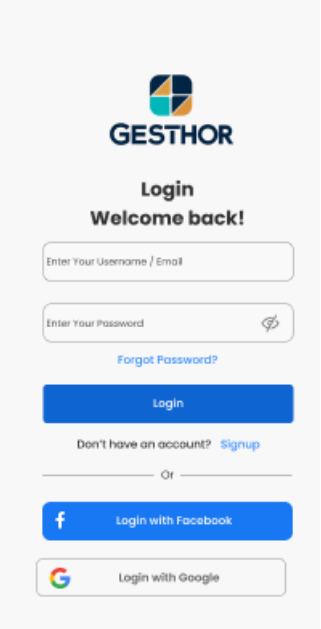


Chat Bot:

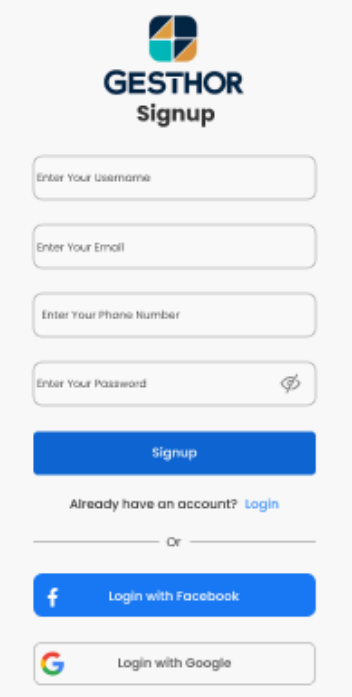


Mockups da interface (Link).

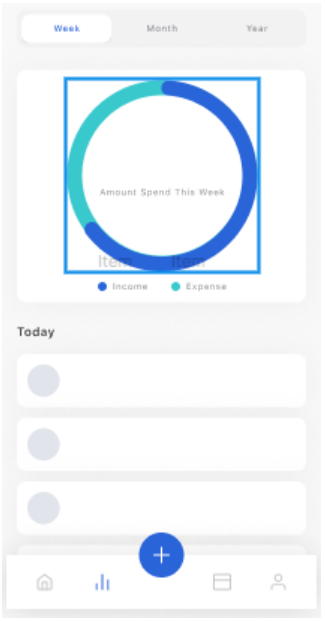
Login



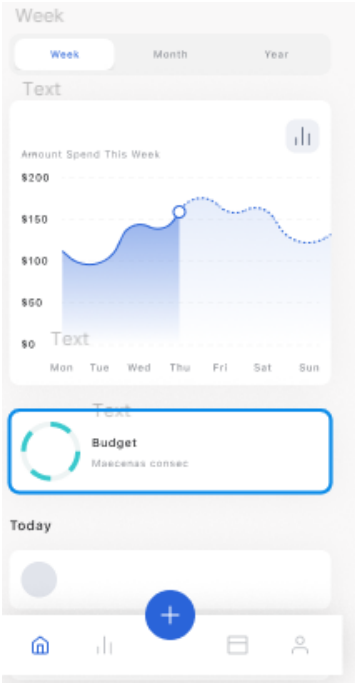
Registro



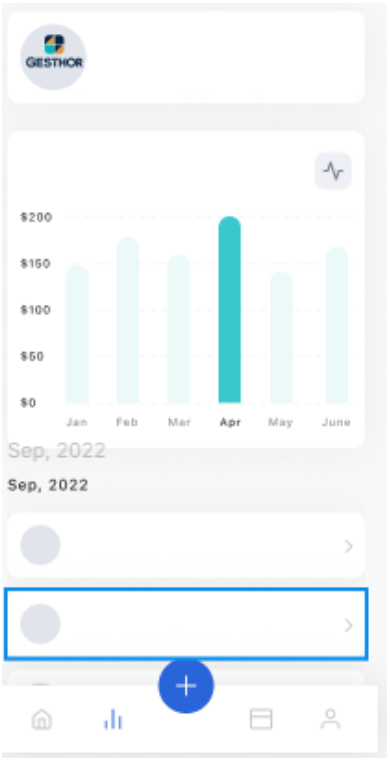
Inicio



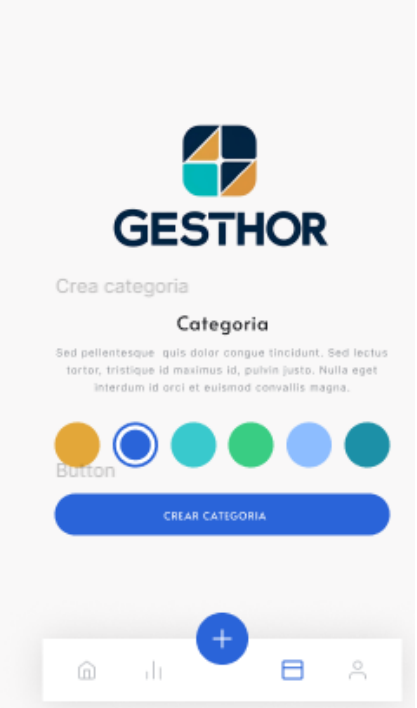
Filtros



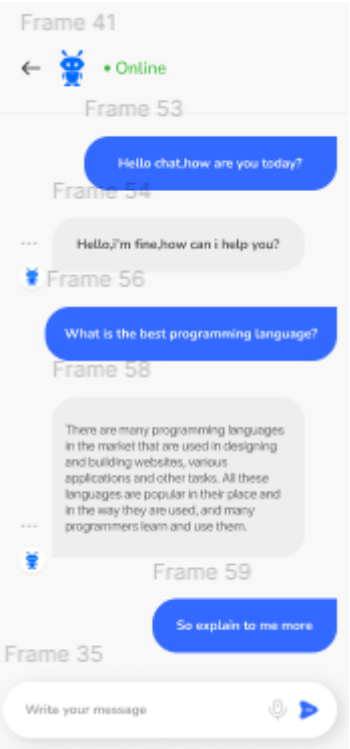
Comparaciones



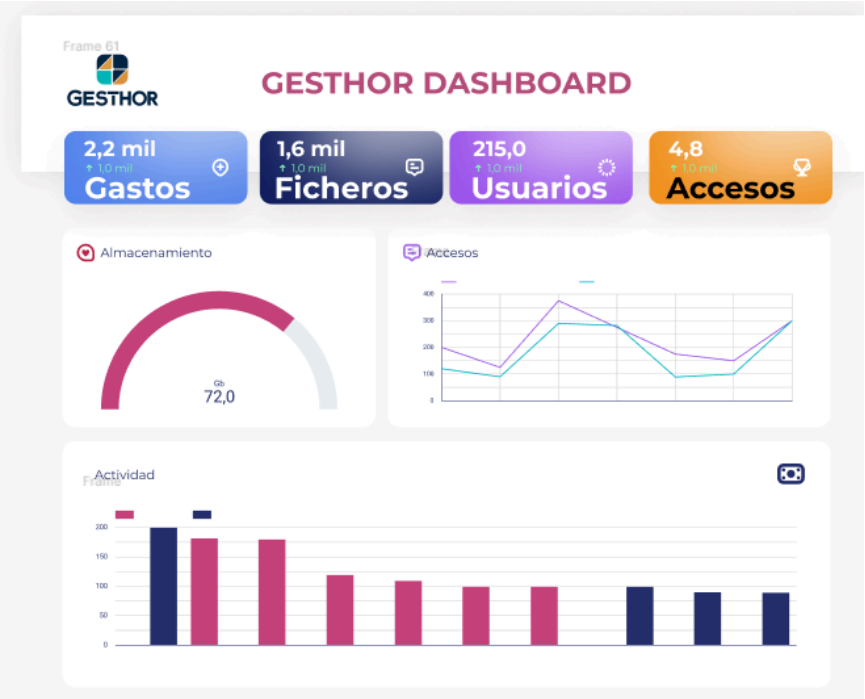
Categorias Personalizadas (Opcional).



Chat bot



Dashboard



Presuposto Hardware e Software

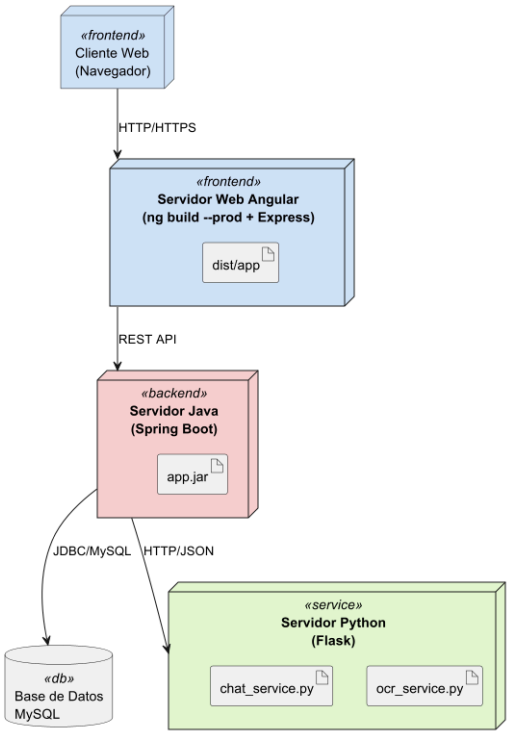
Elemento Harware	Descrición	Prezo
Procesador	Intel Core i5-9400F	105€
RAM	16GB DDR4	60€
Placa base	MPG Z390 GAMING PLUS	130€
Tarxeta gráfica	NVIDIA GeForce GTX 1660 Ti	200€
SSD	Kingston SA400M8120G (120GB)	25€
HDD	Seagate ST1000DM010-2EP102 (1TB)	40€
Caixa	NOX Hammer	30€
WiFi USB	Adaptador	2€
Periféricos	Teclado e rato HOCO	Gratis
Auriculares	Hi-Fi	Gratis
Tablet	10" para visualización	Gratis
Disco externo	WD 1TB	50€
Monitor 1	Nilox 24" reacondicionado	45€
Monitor 2	Philips 24" reacondicionado	65€
Portail MSI Pulse 15	Portátil de trabajo y desarrollo	1040€

Elemento Software	Descrición	Prezo
SO	Windows 11 Home	Gratis
Backend IDE	IntelliJ IDEA / Trae	Gratis-
Frontend IDE	Visual Studio Code	Gratis-
Contedores	Docker (Opcional)	Gratis-
Control de versións	GitHub	Gratis-
Base de datos	MySQL Server (con opción de PostgreSQL)	Gratis-
API Testing	Postman	Gratis-
Frameworks	Spring Boot, Angular CLI, FastAPI/FaskAPI	-Gratis
OCR/IA	Librarías Python/IA ChatGPT	5€ o cafe
Revisión de Código y Documentación, generación de Imágenes y animaciones,etc..	ChatGPT/DeepSeek/Pika/ContentCore/Canvas/Figma/responsively.app/	Gratis

Elemento Software	Descrición	Prezo
Desarrollador Junior Backend	Desarrollo de APIs	Gratis
Desarrollador Junior Frontend	Desarrollo de WEB/Aplicación	Gratis
Gestor DBA junior	Planificacnon y estructura de bases	Gratis

7. Desenvolvemento e execución

Diagrama de despregamento e descripción xeral do funcionamento da aplicación.



Frontend Angular: ejecutado en navegador o servidor web.

Backend Java (Spring Boot): servidor principal con lógica de negocio y autenticación.

Servidor de servicios Python: maneja tareas específicas como IA, OCR, etc.

Base de datos MySQL: almacena datos de usuarios, transacciones, etc.

Completar cos detalles técnicos que fose preciso resolver documentados ao longo do desenvolvemento.

Requiere preparación de entorno ver documentacion github:

- Variables de entorno.
- Node
- Python
- Tesseract

8. Conclusións e reflexións

Funcionalidades Pendentes:

Carga de CSV ou tickets dixitais: Está pendente que funcione cargar arquivos CSV ou tickets dixitais (sin modelos) directamente dende a interface web.

Restauración de conta e verificación en dous pasos: A funcionalidade está prevista para reforzar a seguridade do sistema, mais aínda non se implementou (Redirixida a formularios de contacto temporalmente).

Contedorización con Docker: Non se logrou integrar todo o sistema en contedores Docker por problemas de configuración e dependencias cruzadas.

Dificultades Atopadas e Solucións Propostas:

- Backend e Infraestrutura:

  Problemas coas datas entre SQL e Spring Boot: Houbo desaxustes ao interpretar formatos de data/hora entre a base de datos e o backend. Solucionouse engadindo conversores específicos e controlando o uso de @Temporal e @JsonFormat.

  Eliminación dunha segunda base de datos en PostgreSQL: Intentouse usar dúas bases, pero a configuración foi complexa e a instalación de PostgreSQL resultaba innecesaria. Decidiuse manter unha única base.

  Inicialización da base de datos: O inicializador lanzaba erros por falta de datos e estrutura. Engadíronse parámetros adicionais e validacións para estabilizalo.

  Implementación de JWT e OAuth: Foi unha das partes máis complicadas. Supuxo revisar moitos repositorios e titoriais para comprender como combinar Spring Security con JWT e OAuth.

  Uso de key compartida en API Flask: No canto dun sistema de login completo, optouse por empregar unha API key compartida para autenticar os usuarios e limitar o acceso aos endpoints, simplificando a implementación.

  Swagger en Python: Intentouse usar Flasgger para a documentación automática da API Flask, pero o resultado visual non era axeitado. Aplicouse Sphinx.

- *Frontend e Experiencia de Usuario*

  Responsive design con Bootstrap: A interface tiña que ser responsiva, pero non se adaptaba ben a móbiles. Empregáronse clases Bootstrap con trucos manuais para conseguir un comportamento máis fluído.

  Problemas co dashboard que non se actualizaba: O dashboard non mostraba datos en tempo real. Engadiuse un temporizador para refrescar a información, pero queda o problema de que realiza chamadas constantes mentres o usuario admin está conectado.

  Gráficas en Angular: Ao principio non se comprendía ben como pasar os parámetros dinámicos aos gráficos. Unha vez entendido, foi sinxelo integrar Chart.js e outras librerías para visualización.

- *Outros*

  Problemas coa IA local (OCR): O recoñecemento de texto con Tesseract non foi moi preciso en moitos casos. A solución parcial foi combinar o OCR con interpretación mediante IA.

  Errores tipográficos e solucións manuais: A IA Copilot foi útil para detectar erros tipográficos ou de sintaxe.

  Moitas cousas que me ensinaro a facer automaticas , fixenas a man. Pendiente de mellorar..

- *Estratexias e Axustes Técnicos*

  Uso de almacenamento local para ficheiros e datos temporais (Paso de AWS).

  Asistencia da IA para solución de erros, comentar de código e mellora de este.

  Simplificación do backend de Flask mediante API key no canto de autenticación complexa.

  Aplicación de boas prácticas de commits e ramas en Git tras problemas iniciais.

  Probas manuais exhaustivas e revisións de logs para mellorar a estabilidade do sistema.

9. Bibliografía e Webgrafía Formato APA

Autor: Laureano de Sousa Dias

Spring Boot Docs: https://spring.io/projects/spring-boot

Angular Docs: https://angular.io/docs

Bootstrap 5: https://getbootstrap.com/docs/5.3/getting-started/introduction/

JWT y OAuth en Spring: https://www.baeldung.com/spring-security-oauth-jwt

Tesseract OCR: https://github.com/tesseract-ocr/tesseract

Flasgger (Swagger para Flask): https://github.com/flasgger/flasgger

Chart.js con Angular: https://www.chartjs.org/docs/latest/

StackOverflow, YouTube (Amigoscode, Java Brains, etc.)

Asistencia continua con OpenAI ChatGPT

14. Anexo I – Manual técnico de instalación ou posta en marcha

https://github.com/LaureDSD/ProyectoGestorGastos?tab=readme-ov-file

15. Anexo II – Documentación de uso (manuais usuario)

https://github.com/LaureDSD/ProyectoGestorGastos?tab=readme-ov-file

16. Anexo III – Outra documentación

https://github.com/LaureDSD/ProyectoGestorGastos?tab=readme-ov-file