

SISTEMA DE CONTROL Y ANÁLISIS FINANCIERO

FINTRACK CON STT y TTS

Tabla de contenido

| | |
|--|----|
| Introducción..... | |
| 4 | |
| Planificación..... | |
|5 | |
| Historias de usuario..... | 5 |
| Backlog..... | |
|7 | |
| Release Plan..... | |
| 9 | |
| BPM..... | 12 |
| BPM HU - 1:..... | 15 |
| BPM HU - 2:..... | 15 |
| BPM HU - 3:..... | 15 |
| BPM HU - 4:..... | 16 |
| Librería de Voz..... | 16 |
| Desarrollo..... | |
| ...17 | |
| Generalidades..... | |
| 17 | |
| HU- | |
| 1:..... | |
|21 | |
| Tarea 1: Interacción por voz con IA financiera..... | 21 |
| Desarrollo..... | 21 |
| Testing..... | |
|23 | |
| Documentación..... | 23 |
| Tarea 2: integrar STT con entrada de voz..... | 23 |
| Desarrollo..... | 23 |
| Testing..... | |
|24 | |
| Documentación..... | 24 |
| Tarea 3: Desarrollar integración con LLM para respuestas personalizadas..... | 24 |
| Desarrollo..... | 24 |

SISTEMA DE CONTROL Y ANÁLISIS FINANCIERO

| | |
|---|----|
| Documentación..... | 25 |
| Tarea 5: Diseñar la interfaz de chat que muestre la transcripción de pregunta y respuesta.... | 25 |
| Desarrollo..... | 25 |
| Documentación..... | 25 |
| Testing..... | 25 |
| HU- 2:.....25 | |
| Tarea 1: Diseñar pestaña con opción para seleccionar estilo de voz..... | 25 |
| Desarrollo..... | 25 |
| Testing..... | 26 |
| Documentación..... | 26 |
| Tarea 2: Integrar TTS..... | 26 |
| Desarrollo..... | 26 |
| Testing..... | 26 |
| Documentación..... | 27 |
| Tarea 3: Reproducción de audio por medio de botón..... | 27 |
| Desarrollo..... | 27 |
| Testing..... | 27 |
| Documentación..... | 27 |
| HU- 3:.....27 | |
| Tarea 1: Agregar sección específica para activar registro por voz..... | 27 |
| Desarrollo..... | 27 |
| Testing..... | 28 |
| Documentación..... | 28 |
| Tarea 2: Desarrollar lógica para identificar múltiples transacciones en una entrada y extraer tipo, monto, fecha, categoría, esencialidad y descripción de cada transacción..... | 28 |
| Desarrollo..... | 28 |
| Testing..... | 28 |
| Documentación..... | 28 |
| Tarea 3: Implementar manejo de errores con mensajes de corrección específicos..... | 29 |
| Desarrollo..... | 29 |

SISTEMA DE CONTROL Y ANÁLISIS FINANCIERO

| | |
|---|----|
| Testing..... | 29 |
| Documentación..... | 29 |
| Tarea 4: Mostrar resumen de transacciones..... | 29 |
| Desarrollo..... | 29 |
| Testing..... | |
|31 | |
| Documentación..... | 32 |
| Tarea 5: Implementar aprobación, edición o cancelación de transacciones..... | 33 |
| Desarrollo..... | 33 |
| Testing..... | 35 |
| Documentación..... | 36 |
| HU- 4:..... | |
| .37 | |
| Tarea 1: Implementar interfaz de usuario para el asistente virtual..... | 37 |
| Desarrollo..... | 37 |
| Documentación..... | 37 |
| Testing..... | 37 |
| Tarea 2: Desarrollar funcionalidad de entrada por voz para el asistente..... | 37 |
| Desarrollo..... | 37 |
| Documentación..... | 37 |
| Testing..... | 38 |
| Tarea 3: Configurar comportamiento de voz del asistente..... | 38 |
| Desarrollo..... | 38 |
| Documentación..... | 38 |
| Testing..... | 38 |
| Tarea 4, 5 y 6: Programación del modelo (Implementación de respuestas útiles y limitación de respuestas)..... | 38 |
| Desarrollo..... | 38 |
| Documentación..... | 39 |
| Sprints..... | |
| ...39 | |

SISTEMA DE CONTROL Y ANÁLISIS FINANCIERO

| | |
|--|-----------|
| | Sprint |
| 1..... | 39 |
| | Sprint |
| 2..... | 41 |
| Control de versiones..... | 42 |
| Despliegue..... | 43 |
| Documentación..... | 44 |
| Documentación de código..... | 44 |
| Manuales..... | 46 |
| Manual de Usuario..... | 46 |
| Manual de Despliegue y Mantenimiento..... | 48 |
| Mock-ups..... | 51 |

Introducción

Muchas personas enfrentan dificultades para llevar un control claro y constante de sus finanzas personales. Aunque todos registran ingresos y realizan gastos a diario, existe una falta de herramientas prácticas, accesibles y comprensibles que les permitan visualizar el flujo de su dinero sin complicaciones. Las soluciones tradicionales suelen ser complejas, enfocadas en contabilidad técnica o funciones bancarias avanzadas, lo que las hace poco amigables para quienes solo desean un seguimiento sencillo y efectivo de sus transacciones. Esta carencia se traduce en desorganización financiera, malos hábitos de gasto y dificultades para ahorrar o tomar decisiones informadas. **Fintrack nace como respuesta a esta problemática, ofreciendo una solución centrada en la simplicidad y en las necesidades reales del usuario común.**

SISTEMA DE CONTROL Y ANÁLISIS FINANCIERO

Fintrack es una aplicación web diseñada para simplificar la gestión financiera personal mediante una experiencia intuitiva, accesible y ahora más interactiva gracias a nuevas funcionalidades basadas en inteligencia artificial y reconocimiento de voz. Anteriormente, se habían implementado las opciones de registro de transacciones (ingresos o gastos), ajuste de presupuesto mensual y meta de ahorro mensual, reportes de períodos personalizados y análisis de transacciones por período y con otros filtros. Ahora, usando el registro manual y por voz de ingresos y gastos, la aplicación permite al usuario mantener un historial organizado, generar reportes claros y recibir recomendaciones útiles de forma más natural.

Entre sus mejoras recientes destacan:

1. Un sistema de conversación por voz donde el usuario puede interactuar exclusivamente hablando con la IA, tanto para recibir guía sobre el uso de la aplicación como para resolver dudas sobre sus transacciones.
2. La posibilidad de registrar múltiples transacciones contando simplemente lo que ha hecho (por ejemplo: "Ayer gasté en transporte y hoy me pagaron");
3. Una narración del resumen financiero, lo cual mejora la accesibilidad.

Esta propuesta busca ser original, escalable y también adaptarse a las tendencias actuales de usabilidad, fortaleciendo su valor como herramienta para la toma de decisiones financieras.

Planificación

Historias de usuario

Para la generación de las historias de usuario se utilizó ChatGPT, empleando un único prompt específico que permitió crear y refinar las historias de forma iterativa. Este enfoque facilitó la obtención de resultados consistentes y adaptados a los requerimientos del proyecto, agilizando el proceso y optimizando la planificación de tareas y asignaciones.

Unset

Actúa como un experto en gestión de proyectos ágiles y Jira, y realiza la siguiente tarea: Entrada:

Recibirás el enunciado o descripción general de un proyecto.

Enunciado:

Contexto de la aplicación

Nuestra aplicación, un sistema de registro y análisis financiero diseñado para ayudar a los usuarios a comprender sus hábitos de gasto de forma clara y sencilla. Esta aplicación no es un sistema de contabilidad avanzada ni lleva un control del balance general del usuario, tampoco maneja cuentas bancarias ni carteras virtuales. Su enfoque principal es proporcionar un registro histórico de transacciones y ofrecer un análisis básico de los hábitos financieros del usuario.

Objetivo Integrar funcionalidades que empleen reconocimiento del habla.

Funcionalidades nuevas

Implementar un Chat de Voz Personalizado con IA: Integrar un sistema de inteligencia artificial conversacional que, utilizando TTS y teniendo acceso a los datos financieros del usuario, pueda responder preguntas formuladas por voz, ofreciendo consejos y perspectivas específicas sobre su situación financiera actual y estrategias para mejorarla.

Desarrollar un Resumen Financiero Narrado por Voz: Crear una funcionalidad que genere automáticamente un resumen hablado de los ingresos, gastos, metas y alertas financieras para un periodo determinado, utilizando TTS. Esto permitirá a los usuarios obtener una visión general rápida y auditiva de su economía sin necesidad de interacción visual.

Habilitar el Registro de Transacciones mediante Voz: Implementar un sistema STT que permita a los usuarios dictar sus transacciones (ingresos o gastos) al sistema. La IA deberá interpretar la información proporcionada por voz y crear automáticamente los registros correspondientes con los detalles necesarios.

Implementar un chat de manual de usuario por voz: Implementar un chat inteligente que, utilizando TTS y STT, es capaz de escuchar al usuario y responder sus dudas referentes al funcionamiento de la aplicación. Objetivo:

Generar un conjunto de historias de usuario (HU) bien redactadas, en el formato Connextra, que representen las incidencias del proyecto.

Formato de salida requerido: 1. Archivo JSON con la siguiente estructura para cada historia de usuario:

```
[  
  {  
    "id": "HU01",  
    "title": "Título breve y claro",  
    "description": "Como [tipo de usuario] quiero [funcionalidad] para [beneficio]",  
    "acceptance_criteria": [  
      "Criterio 1",  
      "Criterio 2"  
    ],  
    "dependencies": ["HU00", "HU02"],
```

SISTEMA DE CONTROL Y ANÁLISIS FINANCIERO

```
"estimation": "3 puntos",
"hours_estimation": "4",
"difficulty": "Media"
},
...
]
2. Texto plano copiable
```

Instrucciones específicas:

Asegúrate de que las historias estén completas, claras y viables.

Los criterios de aceptación deben ser medibles y verificables.

Si una historia depende de otra, indica esa dependencia correctamente.

La prioridad debe asignarse con base en el impacto y urgencia.

Las historias deben ser adecuadas para su uso en Jira, por lo tanto, evita texto redundante o ambiguo.

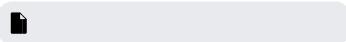
Retorna tanto el JSON como el texto plano listos para ser copiados o descargados.

El prompt se generó manualmente y se refinó con IA en una sola iteración, luego se generaron las Historias de Usuario en un proceso de dos iteraciones:

- **(4:28):** Inicialmente, se partió de un prompt sencillo, el cual fue ampliado y optimizado por IA, luego se refinó este prompt acordé a las necesidades (removiendo parámetros innecesarios para el alcance de proyecto como las 'Epic' y se añadieron parámetros como Horas Estimadas), y se tomó ese prompt para obtener las HUs.
- **(6:19):** Al obtener las Historias de Usuario, la primera iteración generó las Historias de Usuario correctas dados los objetivos planteados, para la segunda y última iteración se ajustaron los criterios de aceptación de algunas HUs para que fueran acordé al alcance del proyecto.

Se inserta las historias de usuario en texto plano y formato JSON:

 Historias de Usuario para MVP: Registro y análisis financiero con TTS y STT



historias_usuario.json

Backlog

Para la generación del Product Backlog, se utilizó ChatGPT con un único prompt específico, lo que permitió identificar características importantes para cada historia de usuario. A cada historia se le asignó una prioridad (Alta, Media o Baja) y se identificaron sus prerequisitos, lo que facilitó la planificación estructurada de los Sprints. Este enfoque optimizó la distribución de tareas y asignaciones, agilizando el proceso y asegurando una planificación alineada con los requerimientos del proyecto.

SISTEMA DE CONTROL Y ANÁLISIS FINANCIERO

Unset

Actúa como un Product Owner. Tienes un archivo JSON que contiene múltiples historias de usuario para un proyecto. Tu tarea es analizar cada historia de usuario y descomponerla en subtareas necesarias para su implementación.

Objetivo: Para cada historia de usuario, genera un conjunto de subtareas claras y ejecutables que representen los pasos necesarios para su desarrollo.

Requisitos de salida:

```
[  
  {  
    "id": "HUXX",  
    "title": "[Título de la historia]",  
    "description": "[Breve descripción de la historia de usuario]",  
    "acceptance_criteria": [  
      "Criterio 1",  
      "Criterio 2"  
    ],  
    "dependencies": [ "HU00", "HU02" ],  
    "estimation": [número de puntos],  
    "hours_estimation": [número de horas],  
    "priority": "[Alta/Media/Baja]",  
    "difficulty": "[Alta/Media/Baja]",  
    "subtasks": [  
      "Subtask #1",  
      "Subtask #3",  
      "Subtask #3"  
    ]  
  }]
```

```
    ],
},
...
]
```

Además, genera un archivo CSV con las siguientes columnas, una por fila:
`id,title,description,acceptance_criteria,dependencies,estimation,hours_estimation,priority,difficulty,subtasks`

Instrucciones adicionales:

Las subtareas deben ser lo suficientemente específicas para poder estimarse individualmente.

Asigna una prioridad adecuada a cada historia según su impacto y urgencia.

Evalúa la dificultad de cada historia como baja, media o alta en función del esfuerzo técnico.

Identifica correctamente las dependencias entre historias si existen.

Asegúrate de que tanto el JSON como el CSV estén completos y bien formateados.

El prompt se generó con ayuda de IA, para llegar al prompt se realizaron dos iteraciones. Con el prompt ya definido, se realizó una única iteración:

- **(4:30):** Se le solicita un prompt adecuado para la tarea, en la primera iteración se generó el prompt, se le pidió que realizara ciertos refinamientos para que dentro del formato de salida, omita información que ya estaba definida dentro de archivo JSON con las historias de usuario.
- **(6:38):** En la primera iteración se generó el prompt y se obtuvo la respuesta de ChatGPT.

Se inserta el product backlog en formato JSON y CSV:

 `product_backlog.csv`  `product_backlog.json`  `product_backlog`

Release Plan

Para el desarrollo del Release Plan, se recurrió de manera activa a herramientas de Inteligencia Artificial (Chat GPT). El objetivo principal era agilizar y optimizar la planificación, ahorrando tiempo y asegurando la consistencia en la organización de tareas, asignaciones y fechas de entrega.

Para la realización del Release Plan se empleó el prompt final utilizado en la entrega anterior, adaptándolo a los requerimientos específicos de esta ocasión para generar directamente el Release Plan completo. Este prompt contiene todas las instrucciones y parámetros necesarios para su creación.

Unset

"Actúa como un experto en planificación de lanzamientos, metodologías ágiles y asignación de recursos en proyectos de desarrollo de software. Tu tarea es generar un release plan completo para la adición de nuevas funcionalidades a un MVP de una app financiera, utilizando las n historias de usuario contenidas en el archivo [product_backlog.json] y la información de los desarrolladores del archivo [desarrolladores.json]. La salida del release plan debe estar disponible en dos formatos: -Un texto en formato markdown.

-Un archivo csv.

Información y Requerimientos Clave:

- Descripción de la App: Esta aplicación no es un sistema de contabilidad avanzada ni lleva un control del balance general del usuario, tampoco maneja cuentas bancarias ni carteras virtuales. Su enfoque principal es proporcionar un registro histórico de transacciones y ofrecer un análisis básico de los hábitos financieros del usuario.*
- Problema que Resuelve: Todos conseguimos y gastamos dinero, muchas personas sin importar su nivel de conocimiento financiero necesitan una herramienta práctica para monitorear sus ingresos y gastos sin la complejidad de una contabilidad formal. Las aplicaciones de finanzas más avanzadas suelen incluir funciones complejas como la gestión de inversiones, balances y múltiples cuentas, lo que puede ser innecesario para quienes solo buscan un registro simple de sus movimientos y un análisis claro de su dinero. Nuestra aplicación llena ese vacío, permitiendo a los usuarios visualizar su historial financiero y recibir información útil sobre sus hábitos de gasto de forma accesible.*

- *Falta de visibilidad y control financiero: Muchos usuarios tienen dificultad para conocer en detalle a dónde se dirige su dinero, lo que puede llevar a malos hábitos de gasto y dificultades para ahorrar. Falta de aplicaciones con un enfoque simplista: Sin una herramienta adecuada, es complicado llevar un registro actualizado y estructurado de todas las transacciones, la mayoría de aplicaciones tienen funciones muy sofisticadas que permiten esta tarea pero la convierten en un proceso tedioso que requiere esfuerzo para llevar el control de transacciones.*
- *Dificultad para diferenciar entre gastos esenciales y opcionales: Los usuarios no siempre identifican cuáles gastos son realmente necesarios y cuáles pueden ajustarse o eliminarse.*
- *Ausencia de alertas y análisis predictivos: La falta de notificaciones y sugerencias para ajustar el comportamiento financiero impide una mejor toma de decisiones.* -
Inicio del Proyecto: 8 de abril de 2025.
- *Plazo: 2 semanas para el desarrollo del MVP.*
- *Sprints: El proyecto se organizará en sprints semanales (un sprint por semana), totalizando 2 sprints.* - *Priorización: MoSCoW*
- *Asignación de subtareas: Utiliza la información contenida en 'desarrolladores.json' para distribuir las tareas de forma equitativa y evitar que la carga recaiga únicamente en los desarrolladores con mejores calificaciones. Es fundamental considerar las habilidades y la experiencia de cada miembro para asignar subtareas de manera justa.*
- *Contenido del Release Plan: Organización de los sprints con fechas correspondientes. Listado de las historias de usuario con las subtareas (no todas las subtareas de una hu se deben completar en un mismo sprint, entonces podrías empezar una HU y seguirla en el otro sprint) a implementar en cada sprint, junto con las tareas principales a desarrollar.* - *Identificación de hitos y entregables clave en cada sprint.*
- *Asignación de subtareas a desarrolladores, basada en sus habilidades y experiencia, garantizando una distribución equilibrada de la carga de trabajo.* - *Instrucciones sobre revisiones y ajustes al final de cada sprint.*
- *El release plan se generará a partir de la información de [historias_usuario.json] y [desarrolladores.json] y deberá entregarse en los siguientes formatos: Texto en formato markdown. Tabla en formato Csv, con columnas que incluyan: Sprint, Fechas, Subtarea,*

Historia de Usuario a la que pertenece la Subtarea, Hitos/Entregables, Duración Estimada y Responsable(s).

- *Genera un release plan integral y detallado que sirva como guía para el equipo de desarrollo del MVP de la aplicación financiera.*

Se adjunta el JSON con las historias de usuario y la información de los desarrolladores:

 product_backlog.json  desarrolladores.json

El proceso completo tomó en su totalidad alrededor de 15 minutos, en donde se dividió principalmente en dos partes:

- **(5:22):** La primera parte se dedicó este tiempo a editar el prompt utilizado en la entrega anterior, incorporando los requisitos específicos de la presente ocasión. De este modo, se obtuvo una primera respuesta por parte de la IA la cual se adjunta abajo.
- **(10:46):** La segunda parte se centró en revisar detalladamente la respuesta generada por la IA, con el objetivo de lograr un equilibrio más justo en la distribución de subtareas entre los desarrolladores, asegurando que todos tuvieran una carga de trabajo equitativa.

Se adjunta el resultado del Release Plan en texto plano y en formato CSV:

 Release Plan  Release Plan

BPM

Para la creación de los Diagramas de Procesos de Negocio (BPMs) a partir de las historias de usuario y el manual de usuario, se siguió un enfoque estructurado con la asistencia de Inteligencia Artificial (IA) para agilizar y refinar el proceso.

1. **Generación de Descripciones de BPMs:** El primer paso consistió en generar una descripción detallada de cada BPM basándose en las historias de usuario y el manual del usuario. Para ello, utilizamos un prompt inicial, cuyo tiempo de generación fue aproximadamente de 10 minutos. La IA jugó un papel crucial en la mejora y refinamiento del prompt para asegurar que la descripción fuera clara y precisa.

Unset

Contexto:

Tenemos un archivo llamado "hus.json" con información detallada sobre diversos procesos BPM.

Objetivo:

Utilizando la información contenida en "hus.json", necesito que redactes la descripción de cada uno de los procesos BPM que allí se encuentran. Indicaciones específicas:

Identifica y enumera cada proceso BPM que aparezca en "hus.json" (por su nombre o identificador). Para cada proceso BPM, describe:

El propósito u objetivo principal.

Los roles o actores involucrados.

Las etapas o pasos clave que conforman el proceso.

Las reglas de negocio o decisiones importantes que apliquen.

Cualquier recomendación adicional útil para la comprensión o ejecución del proceso.

Mantén un estilo narrativo claro y accesible, de modo que la información sea comprensible para cualquier persona, sin necesidad de conocimientos técnicos sobre diagramas o modelado de procesos.

Estructura tu respuesta de forma que cada proceso BPM tenga su propia sección o apartado bien definido.

Resultado esperado:

Un texto organizado, comprensible y bien redactado que describa de forma clara y detallada cada uno de los procesos BPM presentes en "hus.json", sin incluir diagramas ni referencias visuales.

2. Conversión a Mermaid: Una vez obtenidas las descripciones de los BPMs en lenguaje natural, utilizamos otro prompt para convertir estas descripciones al lenguaje de diagramas Mermaid. Este lenguaje es compatible con generadores de diagramas que permiten su fácil copia y pegado en herramientas de diagramación. La generación de los diagramas mediante Mermaid tomó alrededor de 15 minutos, debido a que, en algunos casos, las respuestas generadas por la IA incluían información innecesaria o la orientación de los diagramas no era la correcta. Esto requería ajustes manuales para asegurar que el formato y la estructura fueran adecuados.

Unset

Contexto:

Se cuenta con una descripción detallada de varios procesos BPM, redactados en texto plano (objetivos, roles involucrados, etapas, reglas de negocio, etc.). Queremos transformar dicha descripción en diagramas de flujo utilizando el lenguaje de diagramas Mermaid.

Objetivo:

Generar, a partir del texto base, uno o varios diagramas en formato Mermaid que representen la secuencia de pasos de cada proceso BPM de forma clara y comprensible, con orientación horizontal. **Instrucciones:**

1. Analiza cada proceso BPM descrito en el texto (identifica nombre o identificador, objetivo, etapas o pasos principales y participantes).
2. Con toda esa información, crea un diagrama de flujo en lenguaje Mermaid que refleje la secuencia de las actividades o etapas mencionadas.
3. Para cada proceso BPM, presenta el diagrama en un bloque de código Mermaid independiente. Cada bloque de código debe mostrar:
 - El inicio del proceso (Start).
 - Las principales etapas o pasos, conectados por flechas que indiquen el flujo.
 - Decisiones o bifurcaciones relevantes, si se especifican reglas o puntos de decisión.
 - El fin del proceso (End).
4. No incluyas información ajena a los pasos de cada proceso; cántrate en la secuencia lógica y los actores principales que intervienen.
5. Asegúrate de usar la sintaxis de Mermaid (por ejemplo, 'flowchart TB' o 'flowchart LR') y respetar la indentación para que los diagramas puedan generarse correctamente.

Resultado

esperado:

- Bloques de código Mermaid (uno por cada proceso BPM), que representen el flujo de actividades según la descripción original.
- Cada bloque debe ser fácil de copiar y pegar, de modo que puedan visualizarse los diagramas en un visor compatible con Mermaid.

- 3. Intervención Manual y Ajustes:** Durante el proceso de generación de los diagramas, fue necesario intervenir manualmente en ciertas ocasiones, principalmente para corregir la orientación y la inclusión de elementos irrelevantes. Esto alargó el tiempo de generación, pero permitió obtener diagramas precisos y bien estructurados.

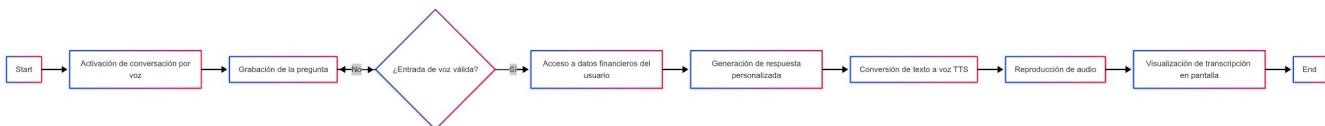
- 4. Resultados Finales:** A continuación, se presentan ejemplos de los diagramas generados, los cuales reflejan los BPMs basados en las historias de usuario. También se incluye el enlace a la carpeta con los BPMs generados para cada Historia de Usuario (HU), así como el documento con las respuestas obtenidas de la IA.

Prompts BPM

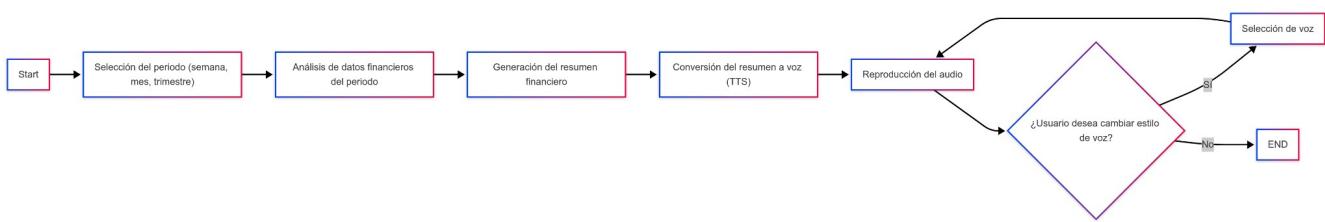
Este proceso fue realizado utilizando ChatGPT para la generación de descripciones y ajustes, y Mermaid para la creación de los diagramas de flujo.

La siguiente imagen es uno de los BPMs generados que representa el flujo de ingresar una nueva transacción:

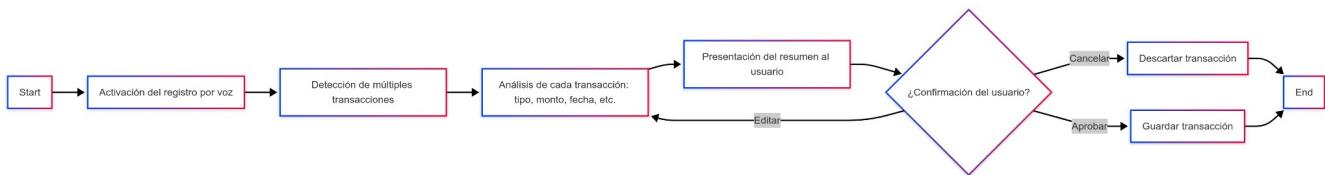
BPM HU - 1:



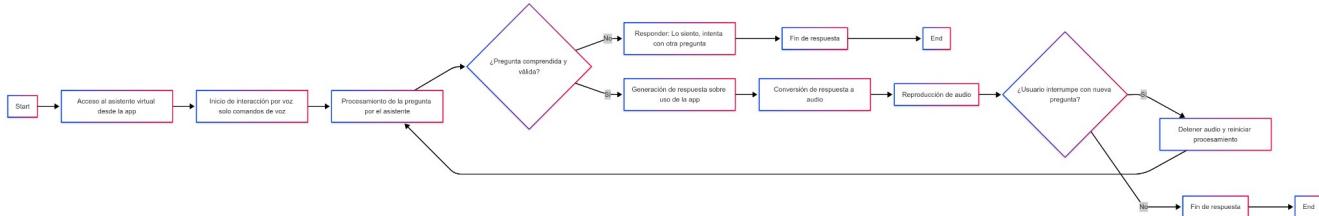
BPM HU - 2:



BPM HU - 3:



BPM HU - 4:



Librería de Voz

ElevenLabs es una plataforma avanzada de inteligencia artificial especializada en síntesis de voz (Text-to-Speech o TTS), clonación de voz realista y también ofrece capacidades de reconocimiento del habla (Speech-to-Text o STT), lo que permite convertir audio en texto con alta precisión. Su tecnología permite convertir texto en audio de alta calidad con voces que suenan naturales, expresivas y adaptables a distintos idiomas, incluyendo el español. Es ampliamente utilizada en productos digitales que buscan ofrecer experiencias más humanas a través del audio, como asistentes virtuales, narración de contenido o interfaces conversacionales.

Para una aplicación como Fintrack, que busca hacer de la gestión financiera personal una experiencia más intuitiva, accesible y centrada en el usuario, incorporar funcionalidades de voz representa un avance natural hacia una interacción más humana y directa. En este contexto, ElevenLabs se presenta como una opción ideal para integrar capacidades de síntesis de voz (TTS) y reconocimiento del habla (STT) en nuestra solución.

A diferencia de otras herramientas más genéricas o complejas, ElevenLabs destaca por su facilidad de uso, calidad natural del audio generado y flexibilidad de integración. Su API es accesible para proyectos de pequeño y mediano tamaño, lo cual es perfecto para el enfoque del MVP de Fintrack.

Además, ElevenLabs ofrece voces realistas en español y otros idiomas, lo que es fundamental para asegurar una experiencia inclusiva y empática para nuestros usuarios. En un sistema que pretende ayudar a las personas a tomar decisiones sobre su dinero, la calidad de la voz y el tono emocional son clave para generar confianza.

Otro punto a favor es su escalabilidad. Si el proyecto evoluciona más allá del MVP, ElevenLabs permite pasar de pruebas gratuitas a planes pagos de forma progresiva, sin requerir reestructuración técnica significativa. Esto es especialmente útil para Fintrack, dado que uno de sus pilares es precisamente la posibilidad de crecer sin abandonar su simplicidad.

Desarrollo

Generalidades

La aplicación fue desarrollada en Next.js, utilizando IndexedDB como base de datos local. Al tratarse de un MVP, se priorizó la demostración de viabilidad. Esta decisión facilitó el despliegue y la configuración del alojamiento.

Para garantizar la calidad del software y verificar las funcionalidades, se emplearon Jest en conjunto con React Testing Library. La planificación y el seguimiento de las tareas se realizaron con Jira, mientras que el control de versiones se llevó a cabo mediante GitHub, siguiendo el flujo de trabajo GitHub Flow.

Además, se configuró un pipeline de GitHub Actions que ejecutaba los tests y validaba el proceso de construcción del proyecto de forma automática. Para la documentación, se utilizó el estándar JSDoc, generando la misma de manera continua y alojándola en GitHub Pages para su consulta en línea.

Prompt para Generación de Pruebas Unitarias:

Este prompt permite cubrir al menos el 80% de la funcionalidad a través de pruebas con Jest y React Testing Library.

Unset

Actúa como un experto en Next.js, Jest y React Testing Library.

Voy a pasarte fragmentos de código de mi proyecto Next.js (Componentes, Hooks, utilidades, API Routes, etc.). Quiero que me generes archivos de prueba (.test.jsx o *.spec.tjs que cumplan con estas pautas:*

Para componentes React:

Usa @testing-library/react para renderizar y hacer assertions sobre DOM.

Mockea dependencias (contextos, hooks personalizados, Next Router) con Jest.

Cubre estados "happy path", interacción de usuario (fireEvent), y manejo de errores/estados vacíos.

Incluye tests de accesibilidad básicos (getByRole, getByLabelText, etc.).

Para Hooks:

Usa `@testing-library/react-hooks` o `RTL renderHook`.

Simula llamadas a `fetch/axios` usando `jest.mock('axios')` o `msw`.

Testea `loading`, éxito, error, actualización de estado.

Para funciones utilitarias:

Tests puros con Jest (`describe/it/expect`).

Cubre todos los casos borde (inputs válidos/erróneos).

Para API Routes (pages/api/*.ts):

Usa `supertest + createMocks()` de `next-test-api-route-handler`.

Mockea llamadas a bases de datos o servicios externos con Jest.

Verifica status codes, JSON responses, manejo de errores.

Ejemplo

```
import { render, screen, fireEvent } from '@testing-library/react';
import MyButton from '@/components/MyButton';

describe('MyButton', () => {
  it('renders label and calls onClick', () => {
    const handleClick = jest.fn();
    render();
    fireEvent.click(screen.getByRole('button', { name: /click me/i }));
    expect(handleClick).toHaveBeenCalled();
  });
});
```

Código:

Además de generar pruebas desde cero, surgió la necesidad de actualizar pruebas existentes cuando los componentes cambiaban.

Para ello, se creó un segundo prompt que permitiera recibir tanto el componente actualizado como el test anterior, y así devolver el archivo de prueba ya ajustado a los nuevos cambios:

Unset

Actúa como un experto en Next.js, Jest y React Testing Library. Ya no quiero que generes un test desde cero. En su lugar, te voy a pasar el fragmento de código actualizado de un componente o función de mi proyecto Next.js, y también el archivo de prueba original correspondiente (.test.jsx o .spec.tsx) que necesita ser ajustado.

Tu tarea será adaptar el test existente para que cubra correctamente los nuevos cambios introducidos (por ejemplo: props adicionales, lógica modificada, nueva interacción, etc.).

Es importante que mantengas la estructura y estilo del test actual, y solo agregues o actualices lo que sea necesario para cubrir los cambios.

Asegúrate de seguir buenas prácticas como:

Usar @testing-library/react y consultas accesibles (getByRole, getByLabelText).

Mockear dependencias externas con Jest.

Cubrir happy path, errores, vacíos, accesibilidad básica.

En tu respuesta, dame únicamente el archivo de test ya modificado, listo para reemplazar.

No expliques los cambios ni incluyas ningún otro comentario.

| File | % Stmt | % Branch | % Funcs | % Lines |
|---|--------|----------|---------|---------|
| All files | 93.82 | 81.51 | 87.39 | 93.82 |
| components/ui/features | 94.29 | 81.81 | 80 | 94.29 |
| ai-user-manual.jsx | 85.47 | 73.91 | 66.66 | 85.47 |
| boolean-input.jsx | 100 | 100 | 100 | 100 |
| date-input.jsx | 98.75 | 100 | 50 | 98.75 |
| number-input.jsx | 100 | 100 | 100 | 100 |
| select-input.jsx | 100 | 100 | 100 | 100 |
| sidebar.jsx | 100 | 100 | 100 | 100 |
| text-input.jsx | 100 | 100 | 100 | 100 |
| components/ui/features/configuracion | 99.23 | 80 | 100 | 99.23 |
| module.jsx | 100 | 100 | 100 | 100 |
| voiceSelector.jsx | 98.96 | 77.77 | 100 | 98.96 |
| components/ui/features/metas | 99.49 | 91.66 | 94.11 | 99.49 |
| form.jsx | 100 | 100 | 100 | 100 |
| goal.jsx | 100 | 92.59 | 100 | 100 |
| header.jsx | 100 | 100 | 100 | 100 |
| list.jsx | 100 | 83.33 | 100 | 100 |
| main.jsx | 100 | 100 | 100 | 100 |
| module.jsx | 98.6 | 88.88 | 83.33 | 98.6 |
| components/ui/features/presupuestos | 97.36 | 82.69 | 100 | 97.36 |
| expenses.jsx | 100 | 93.33 | 100 | 100 |
| form.jsx | 100 | 100 | 100 | 100 |
| header.jsx | 93.98 | 73.68 | 100 | 93.98 |
| module.jsx | 97.36 | 78.57 | 100 | 97.36 |
| components/ui/features/reportes | 86.63 | 78.57 | 82.35 | 86.63 |
| dashboard.jsx | 85.33 | 90.32 | 80 | 85.33 |
| export.jsx | 85.54 | 60 | 83.33 | 85.54 |
| import.jsx | 88.27 | 66.66 | 80 | 88.27 |
| module.jsx | 100 | 100 | 100 | 100 |
| components/ui/features/transacciones | 92.21 | 76.29 | 84 | 92.21 |
| ai-chat.jsx | 82.86 | 76.19 | 66.66 | 82.86 |
| ai-suggestion.jsx | 95.85 | 76 | 77.77 | 95.85 |
| ai-voice-transaction-creator.jsx | 89.31 | 57.14 | 83.33 | 89.31 |
| filters.jsx | 100 | 100 | 85.71 | 100 |
| footer.jsx | 100 | 100 | 80 | 100 |
| form.jsx | 93.58 | 75 | 100 | 93.58 |
| history.jsx | 85.42 | 67.74 | 100 | 85.42 |
| item.jsx | 100 | 78.57 | 100 | 100 |
| list.jsx | 100 | 100 | 100 | 100 |
| module.jsx | 100 | 100 | 66.66 | 100 |
| components/ui/features/transacciones/form | 100 | 100 | 75 | 100 |
| form-carrousel.jsx | 100 | 100 | 100 | 100 |
| form-wrapper.jsx | 100 | 100 | 50 | 100 |
| lib | 100 | 100 | 100 | 100 |
| utils.js | 100 | 100 | 100 | 100 |

La elaboración de estos prompts se realizó así::

El nuevo prompt de actualización fue una extensión natural del flujo de trabajo, y su incorporación fue rápida gracias a la reutilización del contexto del prompt original. Solo se requirió una breve instrucción a la IA para adaptarlo, sin mayor esfuerzo manual.

HU-1:

Tarea 1: Interacción por voz con IA financiera

Desarrollo

Tiempo estimado: ~26 minutos

Esta subtarea fue la primera implementación relacionada con la funcionalidad conversacional de la IA financiera.

El objetivo fue permitir que el usuario pudiera iniciar una interacción a través de un botón, lo que se resolvió incorporando un botón en el componente AISuggestion que abre un modal al hacer clic. Dentro del modal se implementó una esfera animada con framer-motion, la cual reacciona en tiempo real al volumen del micrófono usando un hook personalizado (useMicVolume). Aunque todavía no se procesaba voz ni se mostraban respuestas, esta base servía como simulación visual de que el sistema “escucha”. Se utilizó ChatGPT para guiar parte del desarrollo y resolver dudas puntuales, comenzando con un prompt como:

Unset

Quiero agregar una funcionalidad visual que represente la activación de una conversación con la IA financiera. El objetivo es implementar un botón en el componente [NombreDelComponentePadre] que abra un modal centrado. Esta modal no debe tener lógica real de reconocimiento de voz, solo una animación que simule una "escucha activa" utilizando la intensidad del volumen captado por el micrófono.

Requisitos:

Tecnologías: Utilizar React, motion, Tailwind CSS y componentes de shadcn/ui.

Botón: Agregar un botón en [NombreDelComponentePadre], estilo minimalista, que diga "Iniciar conversación" o

muestre un ícono de micrófono. Al hacer clic, debe abrir la modal.

Modal: Crear una modal usando el componente Dialog de shadcn/ui.

Debe tener un fondo con opacidad que cambie sutilmente según el volumen detectado.

En el centro debe haber una "bola" o círculo que aumente/disminuya su tamaño ligeramente con base en el volumen.

Hook personalizado: Crear un useVolumen() que:

Use la API Web Audio de JavaScript.

Devuelva un valor numérico actualizado del volumen del micrófono (normalizado entre 0 y 1).

Inicie la escucha al montar y se detenga al desmontar.

Animación:

El fondo del modal debe cambiar suavemente de opacidad (o color sutil) según el volumen detectado.

La bola central debe escalar ligeramente con base en ese mismo valor.

La animación debe verse fluida pero no invasiva

No implementar lógica de transcripción ni escucha real. Solo es una interfaz visual.

Para generar el prompt inicial de esta tarea, se utilizó inteligencia artificial (ChatGPT), dedicando aproximadamente 3 minutos al proceso. Se le proporcionó la historia de usuario, el contexto general del proyecto y una explicación informal sobre los objetivos específicos de la funcionalidad que se quería implementar.

A partir de ese punto, se trabajó de forma manual sobre el componente para ajustar detalles finos en la implementación, como el estilo visual, los niveles de volumen adecuados para la animación y la integración con la estructura existente del código.

The image shows two side-by-side code editors. Both have a dark theme with syntax highlighting. The left editor contains a file named 'useVolume.tsx' which defines a functional component 'useVolume'. It uses hooks like useState and useRef to manage state and references to audio context and analyser nodes. It includes an useEffect hook for initializing the audio stream and connecting the analyser. The right editor contains a file named 'VoiceModal.tsx' which defines a component 'VoiceModal' that wraps a Dialog component from '@components/ui/dialog'. It includes an interface 'VoiceModalProps' with properties 'open' and 'onOpenChange'. The component itself returns a Dialog component with aDialogContent and a styled div for the modal content.

```

tsx
import { useEffect, useRef, useState } from "react";

export function useVolume() {
  const [volume, setVolume] = useState(0);
  const audioContextRef = useRef<AudioContext | null>(null);
  const analyserRef = useRef<AnalyserNode | null>(null);
  const dataArrayRef = useRef<Int8Array | null>(null);
  const sourceRef = useRef<MediaStreamAudioSourceNode | null>(null);
  const rafId = useRef<number>();

  useEffect(() => {
    const init = async () => {
      try {
        const stream = await navigator.mediaDevices.getUserMedia({ audio: true });
        const audioContext = new (window.AudioContext || (window as any).webkitAudioContext);
        const analyser = audioContext.createAnalyser();
        const source = audioContext.createMediaStreamSource(stream);

        source.connect(analyser);
        analyser.fftSize = 256;

        const bufferLength = analyser.frequencyBinCount;
      } catch (err) {
        console.error("Error initializing audio context: ", err);
      }
    };
    init();
  }, []);

  return {
    volume,
    setVolume,
    audioContextRef,
    analyserRef,
    dataArrayRef,
    sourceRef,
    rafId,
  };
}

tsx
import { Dialog, DialogContent } from "@components/ui/dialog";
import { useVolume } from "@hooks/useVolume";

interface VoiceModalProps {
  open: boolean;
  onOpenChange: (value: boolean) => void;
}

export const VoiceModal = ({ open, onOpenChange }: VoiceModalProps) => {
  const volume = useVolume();

  return (
    <Dialog open={open} onOpenChange={onOpenChange}>
      <DialogContent
        className="flex items-center justify-center h-[300px] bg-black/70 transition-all"
        style={{
          backgroundColor: `rgba(0,0,0,${0.6 + volume * 0.4})`, // Opacidad variable
        }}
      >
        <div
          className="w-20 h-20 bg-blue-500 rounded-full transition-transform duration-200"
          style={{
            transform: `scale(${volume})`,
          }}
        >
        </div>
      </DialogContent>
    </Dialog>
  );
}

```

Testing

Tiempo estimado: ~14 minutos

Dado que se creó un nuevo componente para esta funcionalidad, se generaron tests desde cero utilizando un prompt general de creación de pruebas con ChatGPT. Como aún no existía lógica de voz ni respuestas, no fue necesario testear flujos complejos. No se requirió actualizar pruebas previas ya que no se modificaron componentes existentes.

Documentación

Tiempo estimado: ~8 minutos

La documentación se generó utilizando el prompt genérico ya definido anteriormente, con descripciones enfocadas en props del nuevo componente, comportamiento esperado, y ejemplo de uso.

Tarea 2: integrar STT con entrada de voz

Desarrollo

Tiempo estimado: ~ 40 minutos

Para la implementación de esta historia de usuario (HU), se invirtieron alrededor de 40 minutos. Se utilizó IA principalmente a través de GitHub Copilot y ChatGPT para generar gran

parte de código y resolver dudas respecto a la estructura, se le explicó a ChatGPT el contexto de la app, tecnologías y objetivos.

El prompt utilizado fue el siguiente: Eres un desarrollador que va a trabajar en un proyecto financiero, el cual usa Next.js, React e indexDB, debes realizar la siguiente tarea [H.U], se estará utilizando elevenLabs.

Testing

Tiempo estimado: ~0 minutos

El componente utilizado ya estaba probado así que no hubo necesidad.

Documentación

Tiempo estimado: ~10 minutos

Para la documentación, se recurrió al *prompt* general propuesto, principalmente enfocado en la creación de bloques JSDoc. El mayor esfuerzo se concentró en describir cada componente y función, así como en revisar la exactitud de lo generado por la IA, asegurando que se cumplieran los estándares de estilo y contenido requeridos.

Tarea 3: Desarrollar integración con LLM para respuestas personalizadas

Desarrollo

Tiempo estimado: ~4 horas

Se configuró y afinó iterativamente una instancia de Gemini, definiendo su rol, instrucciones de sistema, manejo de historial y formato de entrada/salida para generar respuestas conversacionales basadas en las transacciones del usuario. La mayor parte del tiempo se dedicó a ajustar el prompt y refactorizar código previo para no romper funcionalidades existentes. No se usaron prompts externos; se apoyó en un copiloto para esbozar funciones y solucionar errores de sintaxis.

Documentación

Tiempo estimado: ~30 minutos

Se añadió JSDoc al módulo de Gemini y se documentó la función de chat, describiendo parámetros y formato de respuesta.

Tarea 5: Diseñar la interfaz de chat que muestre la transcripción de pregunta y respuesta

Desarrollo

Tiempo estimado: ~7 horas

Con apoyo en Chat GPT, se diseñó e implementó una interfaz de chat capaz de mostrar en pantalla las transcripciones de las interacciones entre el usuario y el asistente. Para ello, se desarrolló la lógica necesaria que garantiza el funcionamiento adecuado de la aplicación, incluyendo el control de estados del botón de micrófono y la capacidad de silenciar al asistente automáticamente cuando el usuario comienza a hablar. Documentación

Tiempo estimado: ~15 minutos

La documentación se generó utilizando el prompt genérico ya definido anteriormente, con descripciones enfocadas en las nuevas funcionalidades, comportamiento esperado, y ejemplo de uso.

Testing

Tiempo estimado: ~30 minutos

Dado que el componente ya existía, se aplicó el prompt general enfocado únicamente en testear la nueva funcionalidad incorporada. Las pruebas se realizaron con el apoyo de ChatGPT, centrándose en la validación del comportamiento agregado.

HU-2:

Tarea 1: Diseñar pestaña con opción para seleccionar estilo de voz

Desarrollo

Tiempo estimado: ~ 3 horas y 37 minutos

Para la implementación de esta historia de usuario (HU), se invirtieron alrededor de 3 horas y 37 minutos. Se utilizó IA principalmente a través de GitHub Copilot y ChatGPT para generar gran parte de código y resolver dudas respecto a la estructura, se le explicó a ChatGPT el contexto

de la app, tecnologías y objetivos. Fue algo difícil ya que dio muchos problemas al momento de pasar la variable de voz, la cual no cambiaba, pero gracias a la IA se pudo debugear.

El prompt utilizado fue el siguiente: Eres un desarrollador que va a trabajar en un proyecto financiero, el cual usa Next.js, React e indexDB, debes realizar la siguiente tarea [H.U] Testing

Tiempo estimado: ~ 1 hora y 2 minutos

En la fase de pruebas, se empleó el *prompt* general definido junto con ChatGPT para la generación de tests unitarios. Fue bastante fácil de probar, aunque daba algunos errores con el mock de ElevenLabs, se pudo debugear fácilmente.

Documentación

Tiempo estimado: ~10 minutos

Para la documentación, se recurrió al *prompt* general propuesto, principalmente enfocado en la creación de bloques JSDoc. **Tarea 2: Integrar TTS**

Desarrollo

Tiempo estimado: ~30 minutos

Para el desarrollo de esta HU se destinó aproximadamente 30 minutos y no se hizo uso de la IA.

Testing

Tiempo estimado: ~30 minutos

Para las pruebas de esta HU se destinó aproximadamente 30 minutos y se hizo uso de inteligencia artificial específicamente Chat GPT junto con el prompt general de testing para realizar los test, se iteró unas cuantas veces para solucionar errores al mockear en los test.

Documentación

Tiempo estimado: ~10 minutos

Para documentar los cambios realizados se hizo uso de Chat GPT con el prompt general de documentación y esto tomó alrededor de 10 minutos.

Tarea 3: Reproducción de audio por medio de botón

Desarrollo

Tiempo estimado: ~2 horas

Para el desarrollo de esta HU se destinó aproximadamente 2 horas y no se hizo uso de la IA para generar código, sin embargo, se hizo uso de Chat GPT para hacer debugging.

Testing

Tiempo estimado: ~20 minutos

Para las pruebas de esta HU se destinó aproximadamente 20 minutos y se hizo uso de inteligencia artificial específicamente Chat GPT junto con el prompt general de testing para realizar los test, se iteró una vez para solucionar errores al mockear en los test.

Documentación

Tiempo estimado: ~10 minutos

Para documentar los cambios realizados se hizo uso de Chat GPT con el prompt general de documentación y esto tomó alrededor de 10 minutos.

HU-3:

Tarea 1: Agregar sección específica para activar registro por voz

Desarrollo

Tiempo estimado: ~1 hora

Se modificó el footer para incluir un nuevo botón “Crear con voz” que abre un modal de voz. Se incorporó un componente base `AIVoiceTransactionCreator.jsx` con el botón y el área de mensajes. El copiloto aceleró la creación de la estructura inicial.

Testing

Tiempo estimado: ~30 minutos

Se usó el prompt general para hacer las pruebas unitarias y verificar que el botón aparece y abre el modal. Se simuló el estado de escucha/stt con mocks.

Documentación

Tiempo estimado: ~15 minutos

Se añadió JSDoc al nuevo componente y se actualizó la documentación de la interfaz de usuario para reflejar la opción de registro por voz.

Tarea 2: Desarrollar lógica para identificar múltiples transacciones en una entrada y extraer tipo, monto, fecha, categoría, esencialidad y descripción de cada transacción

Desarrollo

Tiempo estimado: ~5 horas

Se integró STT para transcribir el texto hablado y se implementó la utilidad de Gemini para convertir la transcripción en un JSON de transacciones, con reglas para descartar o marcar registros incompletos. El formato y la lógica de exclusión (montos faltantes, fechas erróneas) se definieron en el prompt de sistema. GitHub Copilot facilitó la evolución del código, generando bocetos de funciones y el bucle de parseo.

Testing

Tiempo estimado: ~1 hora

Se escribieron tests unitarios para `interpretTransactions` y el componente de voz, verificando con distintos inputs (válidos, vacíos o erróneos) que se llamaran correctamente las funciones y se actualizaran los mensajes de usuario.

Documentación

Tiempo estimado: ~30 minutos

Se documentó la utilidad de Gemini con ejemplos de entrada/salida y se explicó en JSDoc las reglas de transformación y descarte de transacciones.

Tarea 3: Implementar manejo de errores con mensajes de corrección específicos

Desarrollo

Tiempo estimado: ~2 horas

Se creó un validador con Zod para el esquema de transacción que devuelve el error para identificar el campo conflictivo. El componente de la interfaz llama a esta validación para verificar la correctitud de la respuesta de Gemini.

Testing

Tiempo estimado: ~30 minutos

Tests para comprobar que `isValidTransactionArray` retorna errores esperados y que el componente renderiza los mensajes de validación sin romper el flujo de voz.

Documentación

Tiempo estimado: ~30 minutos

Se añadió JSDoc a las funciones de validación y se documentó en el README las posibles rutas de error y cómo interpretarlas en la UI.

Tarea 4: Mostrar resumen de transacciones

Desarrollo

Tiempo estimado: ~1 Hora y 30 Minutos

Para la implementación de esta historia de usuario (HU), se invirtieron aproximadamente 1 hora y media. Se utilizó IA de forma intensiva, principalmente a través de GitHub Copilot y ChatGPT, para generar parte del código y resolver dudas relacionadas con estructura, arquitectura y mejores prácticas.

Al inicio, se proporcionó el siguiente prompt a ChatGPT:

Unset

Quiero crear un componente en react que me permita visualizar y editar múltiples transacciones de forma secuencial, utilizando un solo formulario (TransactionForm) que ya tengo desarrollado y en uso en otras partes del proyecto.

La idea es mostrar cada transacción individualmente, como si fuera un carrusel, con botones de navegación hacia la izquierda y derecha que permitan recorrerlas una por una.

Es importante que este nuevo componente no modifique la lógica interna del TransactionForm, ya que se reutiliza en otras partes del sistema. Solo necesito una forma de envolverlo para que pueda navegar entre distintas instancias del formulario sin romper su estructura ni validaciones.

En resumen, busco una experiencia tipo carrusel (no scroll horizontal libre), donde se muestre un solo formulario a la vez y se pueda navegar mediante botones, manteniendo todo sincronizado con el estado local.

Para generar el prompt inicial de esta tarea, se utilizó inteligencia artificial (ChatGPT), dedicando aproximadamente 3 minutos al proceso. Se le proporcionó la historia de usuario, el contexto general del proyecto y una explicación informal sobre los objetivos específicos de la funcionalidad que se quería implementar.

Junto con este prompt, se compartieron los componentes TransactionForm y AIVoiceTransactionCreator. A partir de eso, la IA generó la implementación inicial de los componentes FormCarousel y FormWrapper.

```

jsx
import { useForm } from "react-hook-form";
import { zodResolver } from "@hookform/resolvers/zod";
import { transaccionSchema } from "@/components/schemas/transaccion";
import { useEffect } from "react";
import TransactionForm from "./TransactionForm";

export default function FormWrapper({ transaction, index, onUpdate }) {
  const form = useForm({
    resolver: zodResolver(transaccionSchema),
    defaultValues: transaction,
  });

  useEffect(() => {
    const subscription = form.watch((value) => {
      onUpdate(index, value);
    });
    return () => subscription.unsubscribe();
  }, [form.watch]);

  useEffect(() => {
    form.reset(transaction);
  }, [transaction]);

  return (
    <TransactionForm
      transaction={transaction}
      setIsCreateOpen={() => {}}
      formInstances={form}
    />
  );
}

tsx
import {
  Carousel,
  CarouselContent,
  CarouselItem,
  CarouselPrevious,
  CarouselNext,
} from "@components/ui/carousel";
import { useState } from "react";
import FormWrapper from "./form-wrapper";

export default function FormCarousel({ transactions }) {
  const [localTransactions, setLocalTransactions] = useState([...transactions]);

  const handleUpdate = (index, updatedData) => {
    const updated = [...localTransactions];
    updated[index] = updatedData;
    setLocalTransactions(updated);
  };

  return (
    <div className="flex flex-col items-center gap-6">
      <Carousel className="w-[380px]">
        <CarouselContent>
          {localTransactions.map((transaction, index) => (
            <CarouselItem key={index} className="p-1">
              <div className="border p-4 rounded shadow bg-white">
                <FormWrapper
                  index={index}
                  transaction={transaction}
                  onUpdate={handleUpdate}
                />
              </div>
            </CarouselItem>
          ))}
        </CarouselContent>
      </Carousel>
    </div>
  );
}

```

Durante la fase de integración y ajustes, se siguió iterando con ChatGPT para refinar la solución, depurar errores y resolver detalles de funcionalidad o UX. Además, se realizaron intervenciones manuales con conocimiento propio cuando fue necesario.

Posteriormente, se buscó mejorar la experiencia de usuario del carrusel integrando animaciones con Framer Motion, para lograr una transición suave entre formularios al navegar. Se implementó un efecto de entrada y salida con deslizamiento en el eje X y cambios de opacidad, sin alterar la lógica de navegación existente. Testing

Tiempo estimado: ~40 Minutos

Durante esta fase, se reutilizó el prompt general para generación de pruebas unitarias con ChatGPT. Parte del esfuerzo se centró en modificar tests ya existentes, debido a los nuevos props y funcionalidades agregadas en componentes previamente desarrollados, como TransactionForm, TransactionModule y AIVoiceTransactionCreator.

En particular, la parte del carrusel (FormCarousel) presentó retos al momento de mockear correctamente su comportamiento. Se tuvieron que realizar varias iteraciones de mocks para embla-carousel, enviando errores detallados a ChatGPT hasta obtener un mock funcional que permitiera testear sin romper la suite.

También se adaptaron pruebas para reflejar correctamente las nuevas capacidades de actualizar y eliminar formularios de forma individual dentro del carrusel

```
✓ SOLUCIÓN COMPLETA: mock de embla-carousel-react con API válida
Reemplázá tu mock por esto:

js
jest.mock("embla-carousel-react", () => {
  const mockEmblaApi = {
    canScrollPrev: jest.fn(() => false),
    canScrollNext: jest.fn(() => true),
    scrollPrev: jest.fn(),
    scrollNext: jest.fn(),
    on: jest.fn(),
    off: jest.fn(),
    slideNodes: [],
  };

  const useEmblaCarousel = () => [jest.fn(), mockEmblaApi];

  return {
    __esModule: true,
    default: useEmblaCarousel,
    useEmblaCarousel,
  };
});

const mockTransactions = [
  {
    id: 1,
    amount: 100,
    category: "Food",
    date: "2025-04-17",
    description: "Empanadas",
    type: "Expense",
    essential: true,
  },
  {
    id: 2,
    amount: 2000,
    category: "Salary",
    date: "2025-04-18",
    description: "Pago mensual",
  }
];
```

Documentación**Tiempo estimado:** ~15 Minutos

La documentación se generó usando el prompt base de JSDoc junto con Copilot. Se documentaron todas las funciones, props y estructuras clave, asegurando que cada componente tuviera su bloque JSDoc correctamente definido.

El mayor tiempo fue dedicado a revisar y ajustar manualmente el contenido generado por la IA, con foco en mantener consistencia de estilo, claridad técnica, y precisión sobre el propósito y comportamiento de cada parte del código.

Tarea 5: Implementar aprobación, edición o cancelación de transacciones

Desarrollo

Tiempo estimado: ~53 minutos

Para la implementación de esta historia de usuario (HU), se invirtieron aproximadamente 53 minutos. Se utilizó IA de forma activa, principalmente a través de **GitHub Copilot** y **ChatGPT**, para generar gran parte del código y resolver dudas sobre estructura y comportamiento esperado.

El prompt inicial fue:

Unset

Quiero extender un componente ya existente que muestra transacciones individuales en un carrusel utilizando TransactionForm, FormWrapper y FormCarrouse, y AIVoiceTransactionCreate.

Ahora necesito que el carrusel permita no solo visualizar, sino también crear, editar en tiempo real y eliminar transacciones una por una. Los nuevos requerimientos son:

Editar en tiempo real: Cuando el usuario modifica cualquier campo de una transacción, los cambios deben reflejarse inmediatamente en el estado local (localTransactions) sin necesidad de guardar manualmente. Para esto, el formulario se controla desde fuera usando formInstance, y form.watch() es utilizado con protección de render inicial para evitar loops infinitos.

Eliminar transacción: Cada slide del carrusel debe incluir un botón para “ Descartar” que permita eliminar esa transacción específica del listado.

Guardar todas: Se agrega un botón global “Guardar Todos” que recorre el array actualizado de transacciones y:

Llama a addTransaction() si la transacción no tiene id

Llama a updateTransaction() si la transacción ya existe (tiene id)

El formulario TransactionForm no debe modificarse internamente. Solo se le inyecta un formInstance y opcionalmente un onCustomSubmit, pero no debe manejar lógica externa directamente.

Se evita el loop infinito que se generaba al usar form.watch() junto con form.reset() usando una ref isInitialRender.

En resumen, ya tengo un carrusel funcional de formularios. Ahora lo quiero extender para permitir una experiencia completa de CRUD local antes de persistir los datos: edición instantánea, eliminación uno a uno, y guardado masivo.

Para generar el prompt inicial de esta tarea, se utilizó inteligencia artificial (ChatGPT), dedicando aproximadamente 3 minutos al proceso. Se le proporcionó la historia de usuario, el contexto general del proyecto y una explicación informal sobre los objetivos específicos de la funcionalidad que se quería implementar.

```
FormCarousel.jsx
jsx Copiar Editar
import {
  Carousel,
  CarouselContent,
  CarouselItem,
  CarouselPrevious,
  CarouselNext,
} from "@components/ui/carousel";
import { Button } from "@components/ui/button";
import { addTransaction, updateTransaction } from "@db/db";
import { useState } from "react";
import FormWrapper from "./form-wrapper";

export default function FormCarousel({ transactions }) {
  const [localTransactions, setLocalTransactions] = useState([...transactions]);

  const handleUpdate = (index, updatedData) => {
    const updated = [...localTransactions];
    updated[index] = updatedData;
    setLocalTransactions(updated);
  };

  const handleDelete = (index) => {
    setLocalTransactions((prev) => prev.filter((_, i) => i !== index));
  };

  const handleSaveAll = async () => {
    try {
      await addTransaction(localTransactions);
    } catch (error) {
      console.error(error);
    }
  };
}

FormWrapper.jsx
jsx Copiar Editar
import { useForm } from "react-hook-form";
import { zodResolver } from "zodform/resolvers/zod";
import { transaccionSchema } from "@components/schemas/transaccion";
import { useEffect, useRef } from "react";
import TransactionForm from "./form";
import { Button } from "@components/ui/button";
import { Trash2 } from "lucide-react";

export default function FormWrapper({ transaction, index, onUpdate, onDelete }) {
  const form = useForm({
    resolver: zodResolver(transaccionSchema),
    defaultValues: transaction,
  });

  const isInitialRender = useRef(true);

  useEffect(() => {
    const subscription = form.watch((data) => {
      if (!isInitialRender.current) {
        onUpdate(index, data);
      }
    });
    return () => subscription.unsubscribe();
  }, [form, index, onUpdate]);
}

TransactionForm.jsx
jsx Copiar Editar
export default function TransactionForm({
  setIsCreateOpen,
  transaction = null,
  isSaveAvailable = true,
  formInstance,
}) {
  const { notifyTransactionUpdate } = useTransactionContext();
  const form = formInstance ?? useForm({
    resolver: zodResolver(transaccionSchema),
    defaultValues: transaction || defaultTransaccion,
  });

  useEffect(() => {
    form.reset(transaction || defaultTransaccion);
  }, [transaction]);
}
```

Además del prompt, se le enviaron los componentes mencionados para que propusiera las modificaciones necesarias. Posteriormente, se fue iterando con ChatGPT para depurar problemas frecuentes como múltiples renderizados o comportamientos inesperados debido a malinterpretaciones del contexto por parte del modelo. También se realizaron intervenciones manuales para ajustar comportamientos o detalles más específicos. Testing

Tiempo estimado: ~47 Minutos

Durante esta fase se usó el prompt de actualización de tests ya definido previamente con ChatGPT. El mayor esfuerzo se debió a la necesidad de actualizar tests previamente creados para reflejar nuevas funcionalidades, como los botones de “guardar todos” o “eliminar transacción”.

En algunos casos, la IA actualizaba correctamente los tests, pero en otros generaba errores nuevos debido a malentendidos en el código base o al no simular correctamente mocks como los del carrusel. Se tuvo que depurar en conjunto con ChatGPT, ajustando los mocks y casos de prueba. En situaciones más complejas, fue necesario intervenir manualmente, analizar el código y realizar los cambios por cuenta propia para lograr una cobertura confiable y funcional.

The image shows two side-by-side screenshots of a code editor displaying Jest test files. Both files are named 'test.js' and are written in JSX syntax.

Left Screenshot (TransactionForm Component Test):

```

jsx
import React from "react";
import { render, screen, fireEvent } from "@testing-library/react";
import TransactionForm from "@/components/ui/features/transacciones/form";
import { useTransactionContext } from "@/context/TransactionContext";
import { useForm } from "react-hook-form";

// Mock de useTransactionContext
jest.mock("@/context/TransactionContext", () => ({
  useTransactionContext: jest.fn(() => ({
    notifyTransactionUpdate: jest.fn(),
  })),
});

describe("TransactionForm Component", () => {
  // ... otros tests arriba

  it("usa formInstance si se proporciona", () => {
    function Wrapper() {
      const formInstance = useForm();
      const handleSubmitSpy = jest.spyOn(formInstance, "handleSubmit");

      return (
        <TransactionForm
          formInstance={formInstance}
          setIsCreateOpen={() => {}}
        />
      );
    }

    render();
    expect(handleSubmitSpy).not.toHaveBeenCalled();
  });
});

```

Right Screenshot (FormCarousel Component Test):

```

jsx
import { render, screen, fireEvent } from "@testing-library/react";
import FormCarousel from "@/components/ui/features/transacciones/form/form-carousel";

// * Mock necesario para que embla no explote
jest.mock("embla-carousel-react", () => {
  const mockEmblaApi = {
    canScrollPrev: jest.fn(() => false),
    canScrollNext: jest.fn(() => true),
    scrollPrev: jest.fn(),
    scrollNext: jest.fn(),
    on: jest.fn(),
    off: jest.fn(),
    slideNodes: [],
  };

  const useEmblaCarousel = () => [jest.fn(), mockEmblaApi];

  return {
    __esModule: true,
    default: useEmblaCarousel,
    useEmblaCarousel,
  };
});

```

Documentación

Tiempo estimado: ~ 8 Minutos

Para documentar los cambios, se utilizó nuevamente el prompt general para generación de JSDoc, enfocado en describir props, comportamiento, ejemplos de uso y lógica interna de cada componente y función.

Se prestó especial atención a los componentes modificados (FormCarousel, FormWrapper y TransactionForm), asegurando que los nuevos props (como formInstance, onDelete, onUpdate) estuvieran correctamente documentados.

También fue necesario actualizar documentación ya existente sobre componentes previamente creados. En este punto, GitHub Copilot resultó útil para detectar comentarios desactualizados y sugerir descripciones más adecuadas según los nuevos cambios funcionales.

HU-4:

Tarea 1: Implementar interfaz de usuario para el asistente virtual

Desarrollo

Tiempo estimado: ~20 minutos

Se utilizó el apoyo de Chat GPT, por medio de una prompt que tomó como base un componente ya creado (HU-1: Tarea 5) para diseñar una interfaz de chat capaz de mostrar en pantalla las transcripciones de las interacciones entre el usuario y el asistente.

Documentación

Detalles en Tarea 3

Testing

Detalles en Tarea 3

Tarea 2: Desarrollar funcionalidad de entrada por voz para el asistente

Desarrollo

Tiempo estimado: ~20 minutos

Se utilizó el apoyo de Chat GPT para implementar la lógica de entrada de voz en el componente, fue útil la experiencia obtenida en la creación de componentes anteriores, lo cual redujo significativamente el tiempo de desarrollo.

Documentación

Detalles en Tarea 3

Testing

Detalles en Tarea 3

Tarea 3: Configurar comportamiento de voz del asistente

Desarrollo

Tiempo estimado: ~10 minutos

Se utilizó el apoyo de Chat GPT para implementar las configuraciones de voz necesarias para la interfaz (el asistente debe silenciarse cuando el usuario está hablando), se aprovechó la experiencia obtenida en el desarrollo de otros componentes, reduciendo el tiempo de desarrollo.

Documentación

Tiempo estimado: ~15 minutos

La documentación se generó utilizando el prompt genérico ya definido anteriormente, con descripciones enfocadas en las nuevas funcionalidades, comportamiento esperado, y ejemplo de uso. **Esta abarca las 3 tareas anteriores. Testing**

Tiempo estimado: ~1 hora

Para las pruebas de estas 3 tareas, se destinó aproximadamente 1 hora, se hizo uso de Chat GPT junto con el prompt general de testing para realizar los test, se iteró unas cuantas veces para solucionar errores al mockear en los test.

Tarea 4, 5 y 6: Programación del modelo (Implementación de respuestas útiles y limitación de respuestas) Desarrollo

Tiempo estimado: ~30 minutos

Para crear el prompt que guía al modelo, se partió de una función ya implementada en sprints previos y se contó con la asistencia de ChatGPT; este prompt se diseñó a partir de los contenidos del manual de usuario de la app.

Documentación

Tiempo estimado: ~5 minutos

Se documentó con JSDoc la función **askUsageHelp**, que envía una consulta de uso de Fintrack al modelo.

Sprints

Sprint 1

SISTEMA DE CONTROL Y ANÁLISIS FINANCIERO

Informe escrito

Aplicación de Gestión / Proyecto de software

Backlog

Proyectos / Aplicación de Gestión Básica de Finanzas

Backlog

Q Buscar H +3 Epic ▾

Sprint 3: 4 abr - 15 abr (1 issue)

FINAN-10: Interacción por voz con IA financiera FINAN-10 FINALIZADA 8

+ Crear incidencia

Sprint 4: 15 abr - 22 abr (2 incidencias)

FINAN-15: Registrar transacciones usando la voz FINAN-15 TAREAS POR HACER 8

FINAN-14: Resumen narrado del estado financiero FINAN-14 TAREAS POR HACER 5

+ Crear incidencia

Backlog (0 incidencias) 2 incidencias Estimación: 13

Añadir epic / FINAN-13

Actividades secundarias

100 % hecho

| Tipo | Clave | Resumen | Prioridad | Persona asign... | Estado | ... |
|----------|--|---------|------------------|------------------|--------|-----|
| FINAN-16 | Implementar activación del chat por botón | Medium | H herra... | FINALIZADA | | |
| FINAN-17 | Integrar STT con validación de entrada de voz | Medium | SC SAMUEL G... | FINALIZADA | | |
| FINAN-20 | Desarrollar integración con LLM para respuestas... | Medium | Juan David... | FINALIZADA | | |
| FINAN-21 | Integrar TTS para convertir respuestas a voz y... | Medium | JR Julian Ren... | FINALIZADA | | |
| FINAN-22 | Diseñar la interfaz de chat que muestre la transcripción de... | Medium | J juan.muno... | FINALIZADA | | |

Backlog

Información del backlog

Utiliza esta información para planificar tu próximo sprint.

Sprint: Sprint 3

Compromiso con el sprint

8 puntos Under target: 18-21 points

Desglose de tipos de incidencias

Tu principal tipo de incidencia en la que centrarse en este sprint.

Historia

Enviar comentarios

Progreso del sprint

100 % finalizado

Cerrado En curso Sin iniciar

100% 0% 0%

Trabajo pendiente en el sprint

8 puntos completados, 0 puntos restantes

Apr 8 Apr 16

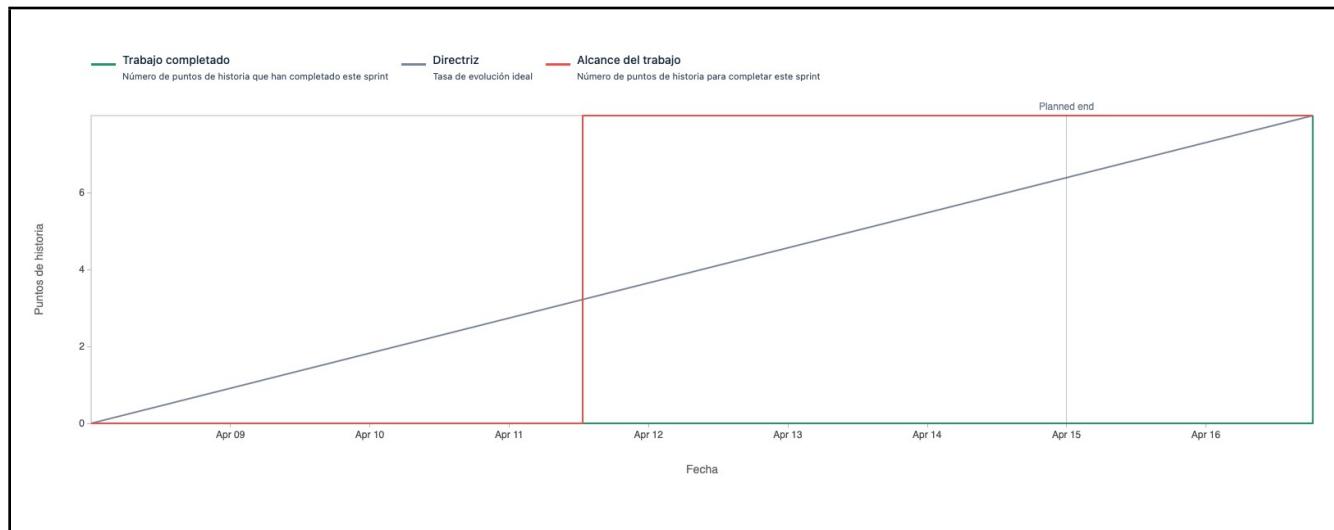
Trabajo restante Orientaciones

El alcance de tu sprint ha aumentado en 8 puntos

Añadida Eliminada Modificada

+8 puntos 0 puntos 0 puntos

6 incidencias 0 incidencias 0 incidencias



En este sprint, el Burndown Chart muestra una caída muy repentina hacia el final del periodo, lo cual puede dar la impresión de que todo el trabajo se completó en el último día. Sin embargo, esto no refleja con precisión la realidad del desarrollo.

Esto se debe a que el gráfico está basado únicamente en el seguimiento de Historias de Usuario (HU) y no considera el avance sobre las subtareas. En este caso particular, el sprint incluía una única HU dividida en varias subtareas que se fueron resolviendo progresivamente a lo largo de los días.

Sprint 2

Aplicación de Gestión ...
Proyecto de software

PLANIFICACIÓN

- Resumen
- Cronograma
- Backlog**
- Tablero
- Calendario
- Lista
- Informes
- Formularios

Proyectos / Aplicación de Gestión Básica de Finanzas

Backlog

Buscar: +4 Epic | Tipo: ...

Sprint 4: 15 abr - 22 abr (4 actividades)

| Actividad | Estado | Puntos | Orientaciones |
|---|------------|--------|---------------|
| FINAN-07 Chat de manual de usuario | FINALIZADA | 8 | 0 |
| FINAN-16 Registrar transacciones usando la voz | FINALIZADA | 8 | 0 |
| FINAN-14 Resumen narrado del estado financiero | FINALIZADA | 3 | 0 |
| FINAN-08 Errores de comportamiento en chat de voz en la conversación financiera | FINALIZADA | 2 | 0 |

+ Crear

Insights | Ver configuración

Progreso del sprint

100 % finalizado

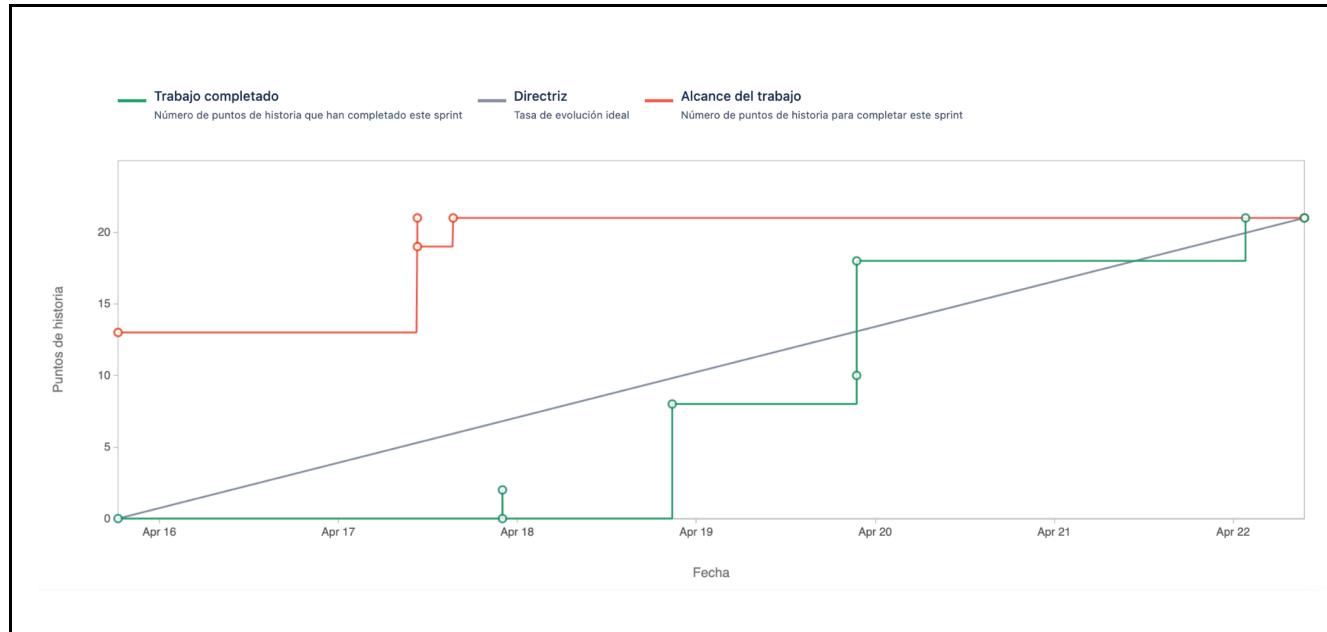
Cerrado En curso Sin iniciar
100% 0% 0%

Trabajo pendiente en el sprint

21 puntos completados, 0 puntos restantes

El alcance de tu sprint ha aumentado en 8 puntos

Añadida Eliminada Modificada
0 puntos 0 puntos +8 puntos
10 actividades 0 actividades 3 actividades



Se adjunta el PDF con el detalle de todas las incidencias registradas en Jira:

[Jira2.pdf](#)

Control de versiones

El código fuente del proyecto ha sido gestionado manualmente a través de GitHub, reutilizando y adaptando flujos de trabajo de proyectos anteriores para ahorrar tiempo y asegurar una estructura sólida. Esta estrategia permitió aplicar buenas prácticas de control de versiones, alineadas con los lineamientos de la rúbrica, facilitando así la integración continua y el trabajo colaborativo.

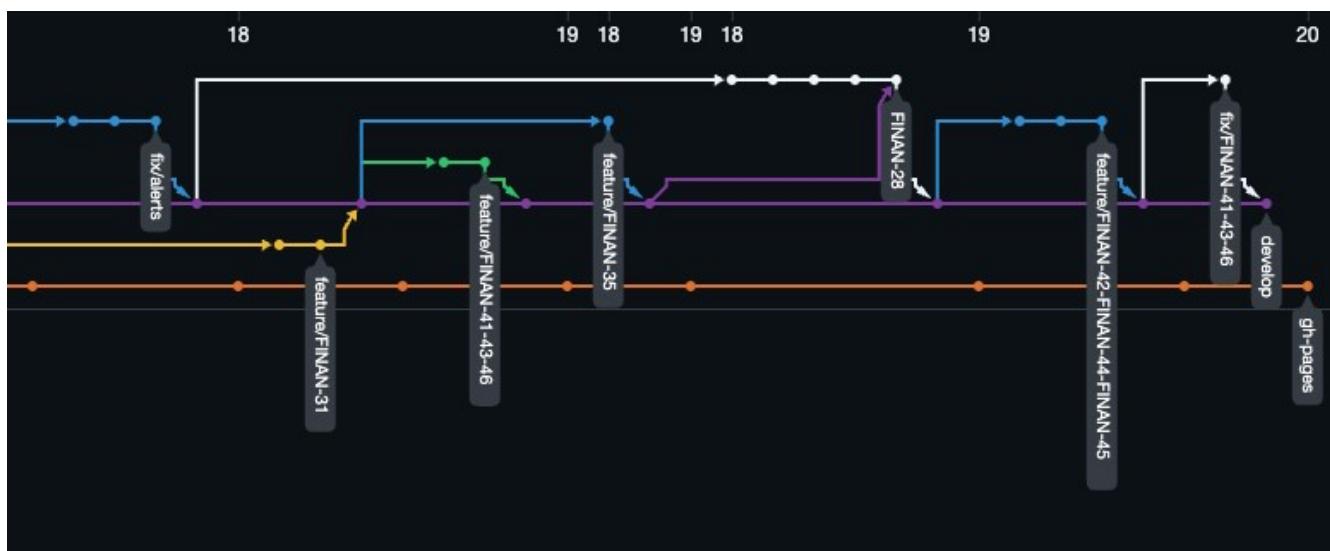
Cabe mencionar que gran parte de esta configuración ya había sido establecida desde sprints anteriores, por lo que en esta etapa no fue necesario trabajar directamente sobre la gestión del repositorio. Simplemente se continuó utilizando la estructura y los flujos previamente definidos.

Detalles y Prácticas:

- Gestión Manual:** Se configuró el repositorio utilizando la estrategia GitFlow, con ramas específicas como feature, develop y main (esta última utilizada como rama de

producción). Esta organización permite un manejo más estructurado y granular del desarrollo, separando claramente el trabajo en curso de las versiones estables.

- **Reutilización de Workflows:** Se reaprovecharon workflows exitosos de proyectos anteriores, en particular de prácticas previas donde se implementaban pipelines de integración continua. Estos flujos incluyen procesos automáticos de construcción del proyecto y ejecución de pruebas unitarias, que deben completarse satisfactoriamente antes de permitir el merge hacia la rama develop. Esto permite mantener la calidad del código y prevenir errores durante la integración.
 - **Documentación Desplegada:** Además, se conservó un workflow del sprint anterior que se encarga de actualizar automáticamente una rama separada (gh-pages), donde se publica la documentación del proyecto. Esta documentación se genera a partir de los comentarios en el código, lo que facilita su mantenimiento y garantiza que esté siempre alineada con el estado actual del repositorio.



Despliegue

El despliegue del MVP se ha automatizado mediante la integración del repositorio de GitHub con Vercel, lo cual permite mantener separados de forma clara y eficiente los entornos de desarrollo y producción.

producción y preproducción. Esta integración ya se había configurado desde el sprint anterior, por lo que en esta etapa no fue necesario realizar ajustes significativos.

Detalles Técnicos:

- **Conexión con Vercel:**

- El repositorio de GitHub se conecta a Vercel para la automatización de despliegues.
- Configuración del pipeline de CI/CD en Github para garantizar actualizaciones continuas y seguras en producción y pre-producción.

- **URLs de Despliegue:**

- **Producción:** <https://fintrack-mvp.vercel.app/>

Cabe destacar que no fue necesario utilizar inteligencia artificial en esta fase, ya que la integración con Vercel es altamente automatizada. El proceso consiste básicamente en otorgar los permisos necesarios desde GitHub, lo que hace que el despliegue sea directo y no presente margen relevante para optimización mediante IA.

Documentación

Documentación de código

La documentación del proyecto se realizó utilizando ChatGPT a través de dos prompts diferenciados: uno enfocado en los componentes visuales (JSX) y otro en los componentes orientados a la funcionalidad. Esto permitió generar documentación precisa y consistente, cubriendo tanto la interfaz como la lógica del negocio.

Métodos y Herramientas:

- **Prompt para Componentes Visuales:**

Se diseñó un prompt específico para describir la estructura, estilo y comportamiento de los componentes visuales desarrollados con Next.js, React y Tailwind CSS.

Unset

 **Prompt – Para documentar un componente React**

Eres un especialista senior en React y documentación. Recibirás un componente funcional escrito en JSX. Tu tarea es generar un bloque JSDoc completo en inglés inmediatamente encima de la definición del componente. Las instrucciones están en español, pero TODO el comentario debe escribirse en inglés. Debes:

Incluir una breve descripción (one-sentence summary) de la finalidad del componente.

Usar la etiqueta @component.

Documentar cada prop con @param {Type} props.propName, incluyendo descripción, valor por defecto (if any) y si es required/optional.

Añadir @remarks para describir cualquier efecto secundario o hook relevante.

Documentar el retorno con @returns {JSX.Element} y breve descripción de lo que renderiza.

Incluir un bloque @example con un uso realista del componente con props.

Seguir estrictamente el estilo JSDoc (orden de tags, gramática, puntuación).

Solo devolver el bloque JSDoc seguido del código original del componente – sin explicaciones adicionales.

• **Prompt para Componentes Funcionales:**

Se creó otro prompt orientado a documentar funciones, módulos y la integración con la Google Gemini API, asegurando que la lógica de negocio y el procesamiento de datos estén claramente explicados.

Unset

 **Prompt – Para documentar una función JavaScript**

Eres un generador experto de documentación JavaScript. Recibirás un bloque de código que contiene únicamente una función JavaScript. Tu tarea es generar un bloque JSDoc completo en inglés (`/** ... */`) inmediatamente encima de la declaración de la función. Las instrucciones que leerás están en español, pero TODO el comentario JSDoc que produzcas debe estar en inglés. Debes:

Incluir una descripción concisa (one-sentence summary).

Documentar cada parámetro con @param {Type} name – inferir el tipo correcto – y describir su propósito (indicar si es required).

Documentar el valor de retorno con @returns {Type} y su descripción.

Documentar posibles excepciones con @throws {ErrorType} si aplica.

Incluir un bloque @example que muestre un caso de uso realista.

Seguir las convenciones oficiales de JSDoc (orden de tags, puntuación, capitalización).

Solo devolver el bloque JSDoc seguido del código original de la función – nada más.

- **Herramientas Utilizadas:**

- **ChatGPT:** Para generar bloques de documentación y comentarios en el código.
- **Copilot:** Para asistir en algunos casos en la creación de bloques JSDoc y en la revisión de la documentación generada.

Manuales

La generación de manuales se realizó utilizando ChatGPT a través de dos prompts diferenciados: uno enfocado en el manual de usuario y otro en el manual de despliegue y mantenimiento.

Manual de Usuario

Este manual va dirigido al usuario final, explicando detalladamente las funcionalidades de la aplicación.

Métodos y Herramientas:

- **Prompt General para generación de Manual de Usuario:**

Unset

Estás encargado de generar la base de un manual operativo para usuarios finales de una aplicación, utilizando la [GUIA] y personalizando el contenido con la [INFORMACION-ESPECIFICA]. Se te proporciona un video que muestra el funcionamiento completo de una aplicación, sigue las [INSTRUCCIONES-MULTIMEDIA] para asegurarte de producir el resultado deseado. Debes seguir las [REGLAS] para asegurar precisión y coherencia en el resultado además de evitar consecuencias negativas y problemas a futuro.

[GUIA]

1. "guia.pdf": Contiene la estructura y requisitos recomendados para el manual de usuario. Analízala y extrae la estructura básica, explicando cada sección de forma objetiva y concisa.

[REGLAS]

- No intuyas ni inventes información o datos, no busques en internet; si la información proporcionada aquí es insuficiente o potencialmente erronea estas obligado a añadir el marcador para revisión manual "***!!REVISAR!!***". No tienes permitido hacer cambios sobre la información proporcionada ni inventar nueva información. Agregar información erronea, falsa o incorrecta puede llevar a problemas judiciales, daño a la reputación, daño a terceros, etc.
- Tu respuesta debe ser el manual operativo en TEXTO PLANO y con formato MARKDOWN.
- Aplica la estructura recomendada de la guía al documento generado.
- Para la sección de funcionalidades sigue las [INSTRUCCIONES-VIDEO] y desarrolla la sección de forma general.
- Si en la [INFORMACION-ESPECIFICA] algún valor es "No aplica" omítelo a menos que la guía lo requiera. Combina valores de categorías similares cuando sea permitido.

[INSTRUCCIONES-MULTIMEDIA]

1. Analiza todos los fotogramas del video (mínimo 1 por segundo) para capturar transiciones, cambios de pantalla y elementos dinámicos.
2. Enfócate en extraer:
 - * Interfaz gráfica: Distribución de menús, botones principales, barras de navegación y elementos visuales recurrentes (colores, iconos).
 - * Pestañas/secciones: Nombres y propósito de cada sección visible (ej: "Configuración", "Reportes", "Perfil").
 - * Funcionalidades clave: Enumera cada función observable (ej: "Botón 'Exportar' para descargar datos", "Formulario de registro de usuarios"). Describe cada una en una oración breve sin detalles técnicos.
 - * Flujo de navegación: Cómo se transita entre pantallas (ej: "Desde el menú principal se accede a 'Historial' mediante un deslizamiento lateral").
3. Excluye detalles específicos de configuración, pasos complejos o elementos no relevantes para la visión general.
4. Organiza la información en una lista estructurada con subtítulos claros (ej: "Interfaz", "Secciones", "Funcionalidades").

[INFORMACION-ESPECIFICA]

- [nombre-aplicacion]: Fintrack.
- [objetivo-aplicacion]: Aplicación de control y análisis financiero.
- [requisitos-hardware-aplicacion]: Mínimos.
- [sistemas-operativos-compatibles-aplicacion]: Mínimos.
- [navegadores-compatibles-aplicacion]: Cualquier navegador moderno, tanto en web como móvil.
- [acceso-aplicacion]: <https://fintrack-mvp.vercel.app/>
- [contacto-empresa]: <correos>

Se realizaron varias iteraciones sobre este prompt, modificándolo para generar paso a paso cada sección del manual, se utilizaron pistas de videos, archivos de imágenes y ejemplos de manuales ya existentes.

- **Herramientas Utilizadas:**

- **ChatGPT:** Para analizar archivos PDF y vídeos frame por frame.

Tiempo estimado

- Generación y adecuación de Prompt: ~20 minutos
- Desarrollo del Manual con Herramientas de IA e iteraciones, incluye tiempos de refinamiento y carga de documentos: ~4 horas
- Actualización del Manual de Usuario con las nuevas funcionalidades de STT y TTS: se realizó manualmente en ~1 hora,

Se inserta enlace al manual:  [Manual de Usuario - Fintrack](#)

Manual de Despliegue y Mantenimiento

Este manual está dirigido a desarrolladores y proporciona una guía detallada sobre el proceso de despliegue y mantenimiento de la aplicación, asegurando su correcto funcionamiento y estabilidad.

Métodos y Herramientas:

- **Prompt para Analizar un Manual de Despliegue de ejemplo**

El objetivo de este analizar un archivo pdf que contiene un ejemplo de un Manual de Despliegue, con el objetivo de entender los temas que se deben incluir:

Unset

Actúa como un ingeniero de documentación senior especializado en manuales de despliegue técnico. Tu tarea es analizar el instructivo que recibirás y:

1. **Desglosar su estructura**: Identificar las secciones principales y subsecciones críticas.
2. **Explicar las partes más importantes**:
 - Objetivo de cada sección.
 - Elementos técnicos clave (ej: requisitos, dependencias, flujos de trabajo).
 - Riesgos o puntos críticos a resaltar.
3. **Generar una lista completa de temas** que debe cubrir el manual final, organizada jerárquicamente (ej: 1.1, 1.2, etc.).

Formato de salida esperado:

- **Análisis**: Breve explicación de la estructura y relevancia del instructivo.
- **Lista de temas**: En formato markdown, con prioridad lógica (ej: desde requisitos previos hasta validación post-despliegue).

Instructivo a analizar: [manual_ejemplo.pdf]

- **Prompt General para generación de Manual de Despliegue y Mantenimiento:**

Dada la lista de temas necesarios para el manual y adaptándolo al alcance y tecnologías utilizadas en el MVP, se utiliza este prompt para generar el manual de despliegue y mantenimiento.

Unset

Estás encargado de generar un manual de despliegue técnico para un sistema/plataforma, utilizando la [GUIA] y personalizando el contenido con la [INFORMACION-ESPECIFICA]. Sigue las [REGLAS] para asegurar precisión y evitar consecuencias negativas.

[GUIA]

1. "plantilla-despliegue.pdf": Estructura estándar para manuales de despliegue (ITIL/DevOps). Extrae y explica cada sección de forma concisa.

[REGLAS]

- **Prohibido inventar información**: Si falta algún dato crítico, añade "!!!REVISAR!!!".
- **Formato**: Texto plano en Markdown.
- **Precisión técnica**: Usa solo la información proporcionada. No generes comandos, configuraciones o requisitos no verificados.
- **Estructura obligatoria**: Sigue la guía al pie de la letra. Omite secciones marcadas como "No aplica".

[INFORMACION-ESPECIFICA]

- [nombre-sistema]: "FinTrack".
- [objetivo-despliegue]: "Desplegar la aplicación en entornos de desarrollo y producción".
- [entornos]: Desarrollo (dev), Producción (prod).
- [requisitos-hardware]: "Mínimo".
- [dependencias]: Next JS, Gemini AI
- [seguridad]: No cuenta
- [contacto]: <correos>

Estructura Esperada del Manual (Markdown):

```markdown

### \*\*1. Introducción\*\*

- \*\*1.1 Objetivo\*\*

- [objetivo-depsliegue].

- \*\*1.2 Alcance\*\*

- Entornos cubiertos: dev, prod.

### \*\*2. Requisitos\*\*

- \*\*2.1 Hardware\*\*

- [requisitos-hardware]

- \*\*2.2 Software\*\*

- [dependencias]

### \*\*3. Procedimiento de Despliegue\*\*

- \*\*3.1 Pre-despliegue\*\*

-

- \*\*3.2 Implementación\*\*

-

### \*\*4. Validación\*\*

-

### \*\*5. Documentación Adjunta\*\*

Se realizaron varias iteraciones sobre este prompt, agregando y corrigiendo la información para su refinamiento y exactitud de acuerdo al MVP realizado. Se utilizaron ejemplos de manuales ya existentes para obtener las secciones necesarias del manual.

- **Herramientas Utilizadas:**

- **ChatGPT:** Para analizar archivos PDF.

### Tiempo estimado

- Generación y adecuación de Prompt: ~15 minutos
- Desarrollo del Manual con Herramientas de IA e iteraciones, incluye tiempos de refinamiento y carga de documentos: ~2 horas
- Actualización del Manual: se realizó manualmente en 2 minutos, el tiempo tan corto corresponde a que la única actualización técnica relevante es la adición de una nueva API key, así como el enlace a su respectiva documentación.

El Manual de Despliegue y Mantenimiento se desarrolló después del Manual de Usuario, lo que permitió aprovechar la experiencia adquirida para optimizar y simplificar gran parte de las tareas. Además, al ser un documento menos extenso, su elaboración resultó más eficiente.

Se inserta enlace al manual:  [Manual de Despliegue y Mantenimiento Técnico - FinTrack](#)

### Mock-ups

La creación de los mockups restantes del MVP tomó aproximadamente 30 minutos.

Acceso a los mockups: [Mockups Fintrack](#)