TRABAJO DE GRADO
Opción Seminario-Diplomado.

CashPilot

Corporación Universitaria Remington.
Facultad de Ingeniería
Ingeniería de sistemas

Karen Restrepo, Daniel Felipe Ruiz y Maria Fernanda Bueno Vargas
Ingeniero Guillermo Tobón
Seminario de aplicaciones móviles
2026

Dedicatoria

Este trabajo va dedicado a Dios, por estar con nosotros en este recorrido, por darnos la fuerza cuando las cosas se volvían difíciles y la sabiduría para continuar adelante sin rendirnos.

A nuestras familias, porque siempre estuvieron a nuestro lado, apoyándonos, creyendo en nosotros aun cuando nosotros mismos teníamos dudas. Gracias por la paciencia, por las palabras de ánimo y por ser ese apoyo incondicional para que podamos llegar a este punto.

También a nosotros mismos, porque detrás de este proyecto hay un esfuerzo real, noches largas, momentos de estrés, pero también mucho aprendizaje. Este resultado es todo lo que fuimos capaces de afrontar y superar para poder conseguirlo.

Agradecimientos

Queremos reconocer a la Corporación Universitaria Remington y a la Facultad de Ingeniería por el hecho de darnos la oportunidad de hacernos profesionales y de brindarnos las bases para llegar hasta aquí.

Al ingeniero Guillermo Tobón, por su permanente acompañamiento durante el seminario, por su amplia disposición al guiarnos en el desarrollo de este proyecto y por poder incluirse en cada uno de los pasos extraordinarios, intentando que esto, finalmente, pudiera salir adelante.

A nuestro equipo de trabajo, porque si no hubiera sido así, esto no hubiese sido posible. Hubo retos, momentos difíciles, pero también hubo apoyo y trabajo en equipo que hizo que todo valiese la pena.

Y, por último, a todas las personas que de una forma u otra también han estado presente en este logro, ya sea dicho con un simple consejo, una palabra de apoyo o simplemente estando ahí, porque también tienen su logro aquí presente en todos ustedes.

Tabla de Contenidos

1. Resumen

Cashpilot es una app móvil que nació dentro del seminario de desarrollo de aplicaciones móviles, con una idea bastante concreta: hacer más fácil el manejo del dinero en el día a día. No es un secreto que la mayoría de las personas tiene problemas para saber en qué gasta, cuánto ahorra o por qué siempre llega corta a fin de mes. Esa situación fue la que nos llevó a construir algo que de verdad le sirviera a la gente.

La app está hecha para que cualquier persona, sin importar si sabe o no de finanzas, pueda anotar sus ingresos, gastos y ahorros sin ningún problema. Todo está montado bajo el patrón MVC, que parte el proyecto en tres piezas: el modelo que guarda los datos, la vista que es lo que uno ve en pantalla, y el controlador que une cada pieza. Esa manera de armar el código fue lo que nos permitió repartir el trabajo sin ningún problema.

Para armarla usamos Ionic con Angular en la parte visual, y Node.js con Express para el servidor. Esas dos partes se hablan por una API REST, que es básicamente el puente por donde va y viene la información. Por eso la app corre bien desde el celular y también desde el navegador sin necesidad de hacer dos versiones distintas.

Lo que se puede hacer en Cashpilot es registrar cualquier movimiento de plata y meterlo en una categoría, ya sea mercado, transporte, salud, etc. También muestra cómo estuvo el mes con unas gráficas que dejan ver rápido en qué se fue más el dinero y dónde se podría ahorrar algo. Tiene además un filtro por fechas para buscar movimientos puntuales, y los datos quedan guardados tanto en el celular como en el servidor para que no se pierda nada.

La seguridad también fue algo que se tuvo en cuenta. Las claves quedan protegidas y hay recuperación de contraseña por correo para cuando uno la olvida, que es algo que cualquier app debería tener.

Este documento tiene todo lo del proceso, desde cómo surgió la idea hasta cómo quedó el sistema. Están los requerimientos, las decisiones que se tomaron, cómo se trabajó y qué se aprendió al llevar a la práctica lo que se vio en el seminario.

1.1 Palabras clave:

Cashpilot, finanzas personales, aplicación móvil, Ionic, Angular, Node.js, Express, MVC, API REST, control de gastos, SCRUM

2. Objetivos

2.1 Objetivo general

Armar una app móvil de finanzas personales llamada Cashpilot, que le dé a cualquier persona una forma sencilla y visual de anotar y revisar en qué gasta y cuánto gana, construida con el patrón MVC usando Ionic y Angular por delante y Node.js con Express por detrás, todo dentro del seminario de aplicaciones móviles.

2.2 Objetivos específicos

1. Organizar el proyecto bajo el patrón MVC, dejando bien separado lo que maneja los datos, lo que tiene la lógica del sistema y lo que el usuario ve en pantalla.
2. Montar una API REST con Node.js y Express que aguante todo lo necesario para registrar, editar, consultar y borrar los movimientos de plata.
3. Realizar las pantallas con Ionic y Angular para que el usuario pueda anotar sus ingresos y gastos, ver las gráficas y buscar por fechas sin que sea un dolor de cabeza.
4. Incluir un módulo de resumen mensual que le muestre al usuario cómo le fue en el mes y le ayude a entender en qué se le va más el dinero.
5. Probar que todo funcione bien, que la app sea estable, que los datos estén seguros y que cualquier persona pueda usarla sin necesitar un manual.

3. Marco conceptual y contextual

Tener un buen manejo de dinero es algo que todo el mundo necesita, pero en su gran mayoría se puede evidenciar dificultades para hacerlo. No importa la edad ni cuánto

gane, el problema es el mismo: a fin de mes no se sabe en que invirtió todo. Y lo curioso es que ahora se vive en una época donde la tecnología está en todos lados, pero mucha gente sigue apuntando los gastos en un cuaderno, en una hoja de Excel o simplemente confiando en la memoria, que como todos sabemos es lo menos confiable del mundo para estas cosas.

El problema con esos métodos es que no solo son incómodos, sino que además no sirven de mucho ya que cuando se quiere mirar hacia atrás y entender qué pasó con su dinero. No hay gráficas, no hay resúmenes, no hay nada que le diga que evidencie claramente en qué está gastando más o dónde podría ahorrar algo. La información está ahí, pero dispersa y sin ningún orden útil.

Ahí fue donde se pudo ver la oportunidad de hacer algo diferente. Cashpilot nació para resolver justo ese problema, de una forma que cualquier persona pudiera usar desde el celular sin necesitar saber de finanzas ni de tecnología. La idea era simple: que uno pudiera anotar lo que gana y lo que gasta, verlo organizado por categorías, revisar cómo estuvo cada mes y entender sus hábitos de consumo mirando unas gráficas que muestran todo de golpe y sin rodeos.

3.1 Contexto del Proyecto

Cashpilot se realizó como parte del seminario de aplicaciones móviles de la Corporación Universitaria Remington. La idea es tomar todo lo que se aprendió durante el programa y aplicarlo en algo real.

Como equipo se quiso demostrar que, si se podía de armar algo desde cero, que no fuera solo una pantalla bonita sino una app que de verdad tuviera su propósito. Eso implicó tomar decisiones técnicas, organizarnos bien y resolver los problemas que fueron apareciendo en el camino, que no fueron pocos.

La app está pensada para cualquier persona, desde un estudiante que tiene que hacer rendir el dinero del mes hasta alguien que trabaja y quiere saber en qué invierte su dinero. No hace falta saber de finanzas ni de tecnología para usarla, eso fue algo que tuvimos claro desde el principio.

3.2 Tecnologías Utilizadas

Para las pantallas se optó por Ionic con Angular, que es una combinación que permitió hacer una sola app que funciona tanto en celular como en navegador sin tener que hacerla dos veces. Ionic trae un montón de componentes ya listos para pantallas táctiles, y Angular ayuda a organizar todo en piezas reutilizables que hacen más fácil ir armando la app sin que todo se convierta en un desorden.

Para el servidor se usó Node.js con Express, que son herramientas que se usan mucho en el mundo real porque son rápidas, flexibles y no dan tanto dolor de cabeza para

conectarlas con otras cosas. Con eso se armó la API REST, que viene siendo el puente entre lo que el usuario ve en pantalla y donde se guardan los datos. Esa separación fue importante porque hace que cada parte del sistema pueda cambiar o crecer sin afectar a la otra. La información va y viene entre las dos capas por peticiones HTTP y en formato JSON, que es básicamente el idioma que hablan entre sí.

3.3 Patrón de Arquitectura MVC

Para organizar el código se usó el patrón MVC, que significa Modelo Vista Controlador. Es uno de los más usados en desarrollo de software y la razón es simple: mantiene todo ordenado y hace que trabajar en equipo sea mucho menos caótico.

El patrón divide el proyecto en tres partes:

- Modelo: Es donde viven los datos y la lógica del sistema. En Cashpilot ahí es donde están las cosas como los usuarios, los ingresos, los gastos y las categorías, junto con las reglas de cómo se guardan, se buscan y se actualizan esos datos por la API.

- Vista: Es lo que el usuario ve en pantalla. En nuestro caso se realizó con Ionic y Angular, tratando de que todo fuera claro y fácil de entender, especialmente porque se trata de información de plata que se necesita leer de un vistazo sin confundirse.

- Controlador: Es la capa del medio, la que conecta la vista con el modelo. Cuando el usuario hace algo en la app, el controlador recibe esa acción, le pregunta al modelo lo que necesita y le devuelve la respuesta a la vista. Es como el intermediario que hace que todo fluya.

Tener esa separación ayudo mucho durante el desarrollo. Cada uno podía trabajar en su parte sin dañar lo que el otro estaba haciendo, los errores eran más fáciles de encontrar porque uno sabía exactamente en qué capa buscar, y el proyecto quedó armado de una forma que si en algún momento se quiere agregar algo nuevo no toca rehacer todo desde cero.

3.4 Modalidad y Escala del Proyecto

Cashpilot es una app híbrida, lo que significa que corre tanto en el celular como en el navegador del computador sin tener que hacer una versión diferente para cada uno. Eso fue una decisión práctica desde el principio porque hacer dos apps separadas con el mismo equipo y en el tiempo que teníamos no era viable.

El frontend y el backend están separados y se comunican por una API REST, lo que hace que si en algún momento hay que cambiar algo en uno de los dos lados no hay problema que alguno este afectado con el otro. Eso le da al proyecto una estructura más limpia y deja la puerta abierta para agregarle cosas nuevas más adelante sin tener que rehacer todo.

Para trabajar como equipo se optó por las metodologías ágiles, específicamente SCRUM. Eso permitió ir avanzando por partes, revisar cómo iba quedando todo y ajustar lo que no estaba

funcionando sin esperar a que el proyecto estuviera terminado para darse cuenta. En un seminario donde los tiempos son cortos y hay muchas cosas que hacer al mismo tiempo, esa forma de trabajar fue lo que nos mantuvo organizados.

4. Desarrollo e implementación del aprendizaje

Para desarrollar Cashpilot se eligió trabajar con SCRUM, que es una forma de organizar el trabajo en equipo que se usa mucho en proyectos de software porque permite ir avanzando poco a poco, revisar cómo va quedando todo y cambiar lo que no está funcionando sin esperar a que sea demasiado tarde. En un proyecto como este, donde el tiempo es corto y hay muchas cosas que resolver al mismo tiempo, esa flexibilidad fue clave.

4.1 Justificación

La decisión de usar SCRUM no fue al azar. Se tomo después de revisar cómo era como equipo, cuánto tiempo teníamos y qué tan claro teníamos todo al inicio, que honestamente no era tanto. Estas fueron las razones principales:

- **Adaptabilidad ante cambios en los requerimientos**

En el desarrollo de una aplicación de finanzas personales, es natural que durante el proceso surjan nuevas ideas, ajustes en las funcionalidades o cambios en la prioridad de los módulos a desarrollar. SCRUM permite gestionar este tipo de situaciones de forma

ordenada, ya que al finalizar cada sprint el equipo tiene la oportunidad de revisar los avances, incorporar retroalimentación y reorientar el trabajo según las necesidades que vayan emergiendo. Esto es especialmente valioso en un contexto académico donde los tiempos son ajustados y cada decisión debe tomarse con agilidad.

Entrega continua de funcionalidades

Una cosa que se evidencio y llamo la atencion fue trabajar con SCRUM, no había que esperar a tener todo listo para ver algo funcionando. Desde los primeros sprints ya se tenia partes de la app andando, como el registro de gastos o el login, y era un buen avance porque se podia evidenciar que el proyecto iba tomando forma de verdad.

Eso también ayudó a encontrar problemas antes. Si algo estaba mal en el registro de ingresos, se descubria en ese sprint y se arreglaba en ese instante. Ir construyendo por partes hizo que el proyecto creciera de forma más ordenada y que llegáramos al final con menos inconvenientes.

Organización y trabajo colaborativo del equipo

Trabajar en equipo suena fácil pero es un gran reto. Al inicio del proyecto se presentaron dificultades en la organización, ya que no había claridad en la asignación de responsabilidades, lo que generaba que algunas tareas quedaran sin ejecutar.. Cuando se definieron los roles con SCRUM eso cambió bastante.

La definición de roles, utilizando la metodología SCRUM. Establecer los roles del Product Owner, Scrum Master y el equipo de desarrollo, ayudó mucho; cada integrante

entendía sus deberes y el alcance de su participación en el proyecto. Este detalle, en particular, emergió cómo esencial, dados los equipos divididos frontend, usaron Ionic y Angular, mientras el backend usó Node.js y Express. Una buena sincronización entre ambas secciones, era crítica para la perfecta unión y operación del sistema.

Una estructura bien definida realmente impulso el progreso técnico y además solidificó la comunicación en el grupo. Los malos entendidos disminuyeron al igual que los reprocesos, mientras tanto se alcanzó una mayor cohesión entre los miembros, todos enfocados en las metas del proyecto.

Estructura de eventos bien definida

Un aspecto clave fue la estructura de eventos de SCRUM que ayudó a ordenar el trabajo semanal, evitando un caos comunicacional en el equipo.
El Sprint Planning era el lugar donde se definia qué hacer cada semana, priorizando tareas y repartiendo responsabilidades.
En cambio, el Daily Scrum se convirtió en esa breve reunión matutina donde cada uno hablaba de sus avances, trabajos en curso, y problemas, sí. Aunque al principio nos costó un poco adaptarnos, luego vimos lo crucial que era para mantenernos coordinados y prevenir errores.
La Sprint Review sirvió para mostrar el progreso, revisar resultados y obtener opiniones.
La Sprint Retrospective, sin embargo, se enfocaba en cómo lo estábamos haciendo,

encontrando aciertos y oportunidades. Aunque al principio no estábamos muy acostumbrados, luego se convirtió en una herramienta importantísima.

4.2 Definición de roles del equipo SCRUM

Para garantizar el buen funcionamento de la metodología SCRUM, fué indispensable fijar, dende el inicio, una asignación clara de roles y responsabilidades. La ausencia de esta definición habria provocado desorganización en el equipo, dificultando la toma de decisiones y la eficaz ejecución de las tareas.
Contar con una estructura bien definida permitío delimitar funciones, previniendo la duplicidad de esfuerzos, garantizando que cada integrante tomara responsabilidades específicas en el proyecto.

4.2.1. Product Owner (P.O.)

El miembro con la visión más clara de CashPilot tomó este rol crucial. Definió las funcionalidades más importantes, así como tambien estableció el orden de desarrollo y garantizó que las entregas satisfacían las necesidades del usuario. Además, cada sprint, revisaba los logros alcanzados, comprobando si se cumplían los objetivos y realizando correcciones si fuera necesario.

4.2.2 Scrum Master

El Scrum Master era quien se aseguraba de que todo fluyera. Si algo frenaba al equipo, ya fuera un problema técnico, una confusión o una tarea que nadie había recogido, era su trabajo desbloquearlo. También fue quien más pendiente estuvo de que el frontend y el backend estuvieran hablando bien entre sí, porque cuando esas dos partes no están coordinadas todo se cae.

4.2.3. Equipo de desarrollo

El grupo de desarrollo, se estructuró así:

1. **El desarrollador frontend:** diseñó e implementó las interfaces, usando Ionic y Angular, oye, se aseguro una presentación clara, funcional, además de adecuada para el uso en móviles.
2. **Desarrollador backend:** construyó la API REST, con Node.js y Express; manejando la lógica del sistema y garantizando el flujo de datos seguro
3. **Analista de calidad (QA):** verificó todos los entregables, antes de aprobarlos, e identificando fallos o comportamientos raros para corregirlos.

4.3 Gestión de requerimientos

Antes de empezar el desarrollo, primero hubo que analizar y fijar los requerimientos de la aplicación eh. No fue fácil esta tarea, porque al principio habían opiniones variadas en el equipo, sobre el alcance y qué funciones tendría CashPilot; por eso, se necesitó concretar acuerdos antes de seguir.

Se identificaron las acciones principales que el usuario podría hacer, como lo son, registrar gastos, mirar resúmenes mensuales y usar filtros de fechas. Además, tambien se miraron

cosas del sistema, tipo seguridad y rendimiento.
Toda esta información se metió en el backlog del producto, ordenando cada requerimiento por importancia. Con esto se planificaron los sprints, para que el equipo supiera que hacer en cada fase del proyecto.

4.4 Diseño del sistema

4.4.1 Flujo de navegación del sistema

Después de establecer los casos de uso, se creó un diagrama de flujo. Para entender la interacción del usuario dentro de la aplicacion de forma bien clarita, desde entrar hasta el uso de funciones.
Este diagrama ayudo a averiguar como el sistema tendría que reaccionar en cada situación, incluyendo la verificación de datos al logearse, gestionar entradas, modificar y borrar movimientos, uso de filtros por tipo y fecha y también el cierre de sesión. Tener esta representación antes de desarrollar hizo que fuera más fácil encontrar errores, de antemano y ayudo a detallar las comprobaciones necesarias en cada sección. Esto mejoró la calidad del sistema incluso antes de su ejecución.

4.5 Implementación del backend

Para el desarrollo del servidor, Node js y Express fueron la base, allì se construyó la lógica interna del sistema. La recuperación de contraseñas por email, un proceso que parece simple, conllevo una configuracion bien detallada.

Para hacerlo funcionar, fue necesario acceder a las herramientas de Google, crear una cuenta Gmail, y generar una clave especial para la aplicación, así el backend podía mandar emails seguro, sin mostrar las credenciales principales. Entender la seguridad y la configuración, fueron vitales, aunque no parecían a primera vista, para que todo anduviera bien.

Las imagenes que verán, muestran lo que hicimos en Google, desde la creación de la clave hasta su uso en el sistema.

Al seleccionar el item permite crear una contraseña personalizada que requerirá nuestro backend para acceder de manera a Gmail y generar la petición de recuperación de contraseña.

Ilustración 1: Crear contraseña personalizada

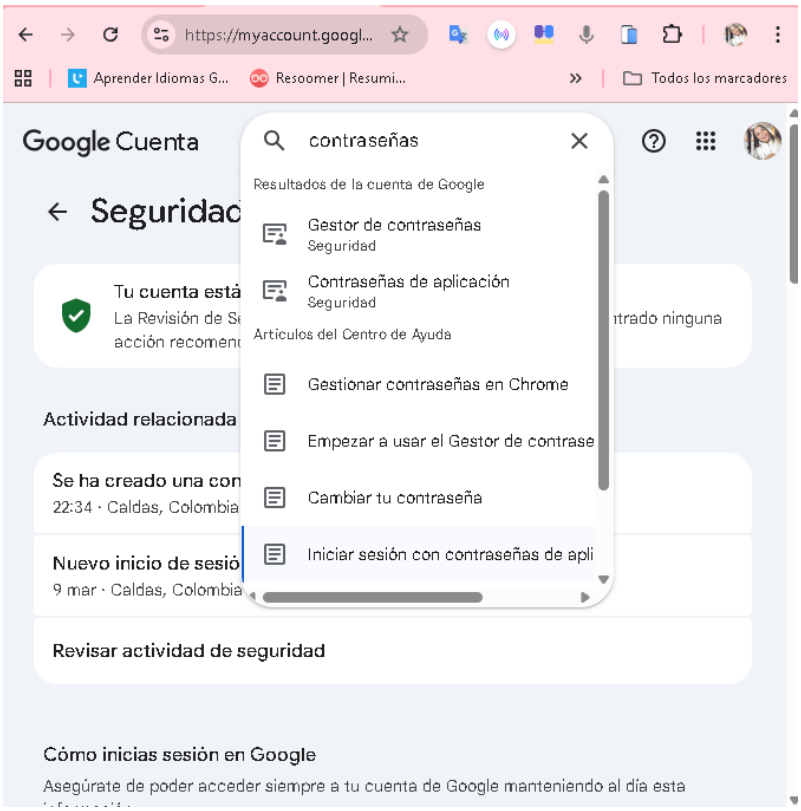
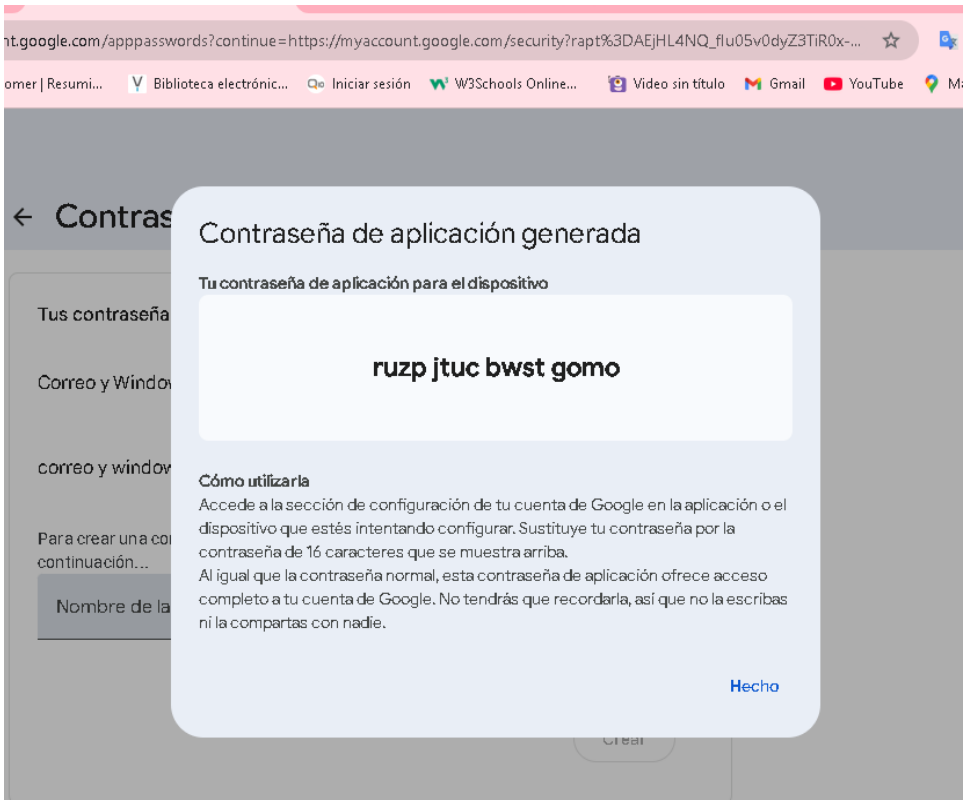
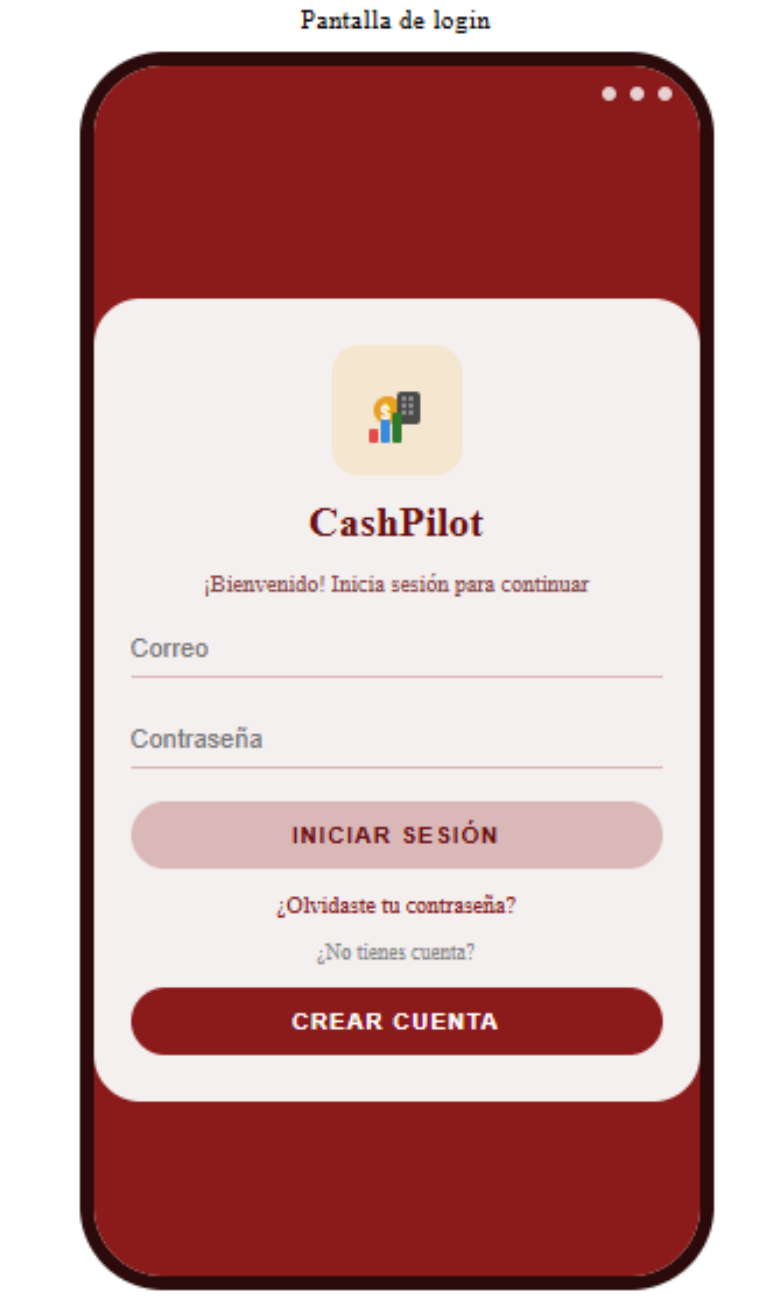


Ilustración 2: Contraseña requerida por Gmail



5. Figuras

Imagen 1. Login para inicio de sesión



Crear cuenta

Imagen 2. Crear cuenta

Imagen 3. *Menú o página principal*

Imagen 4. Ventana de nueva transacción

Imagen 5. Historial de movimientos

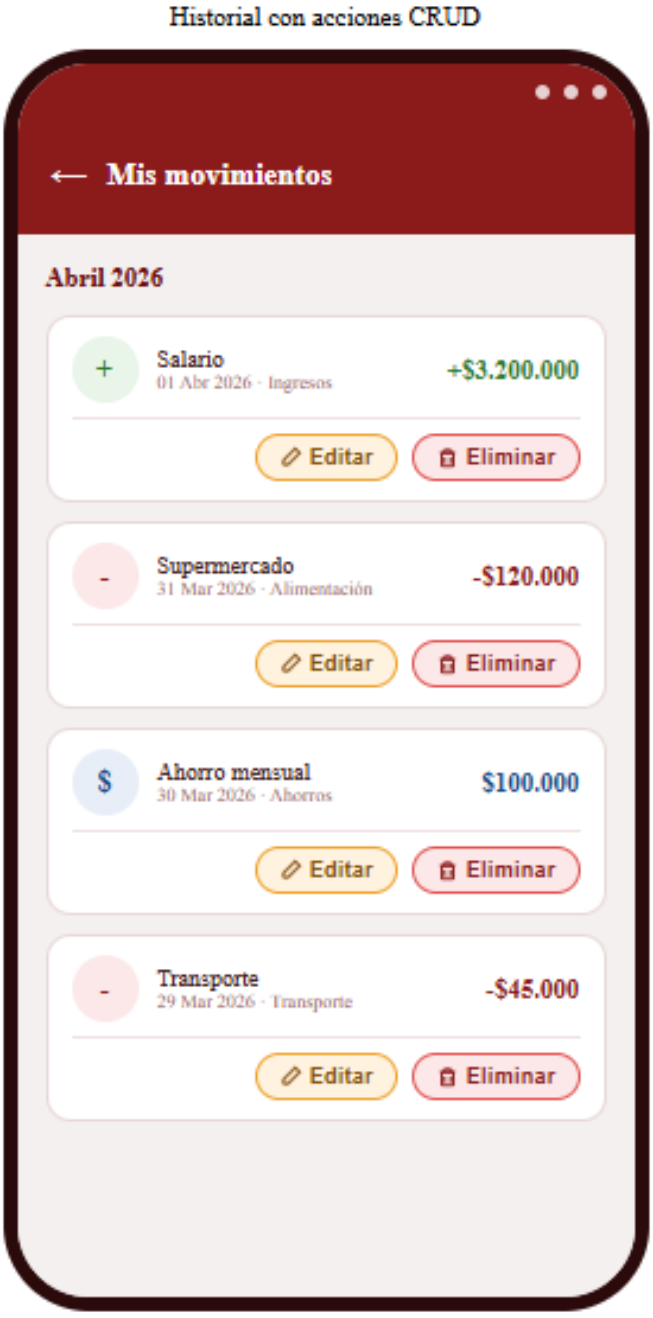


Imagen 6. Filtros de movimiento

Imagen 7. Recuperar contraseña

Imagen 8. Correo recibido con código

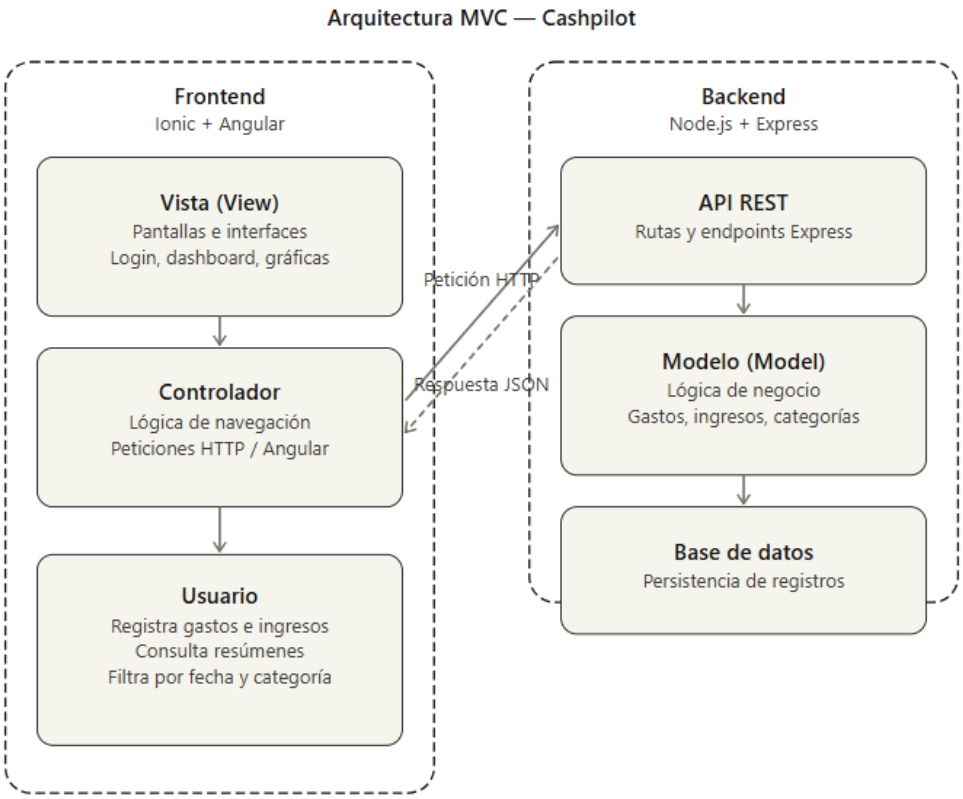
Imagen 9. *Vista del código en el E-mail*

Imagen 10. Ingreso de código para cambio de contraseña

6. Tablas

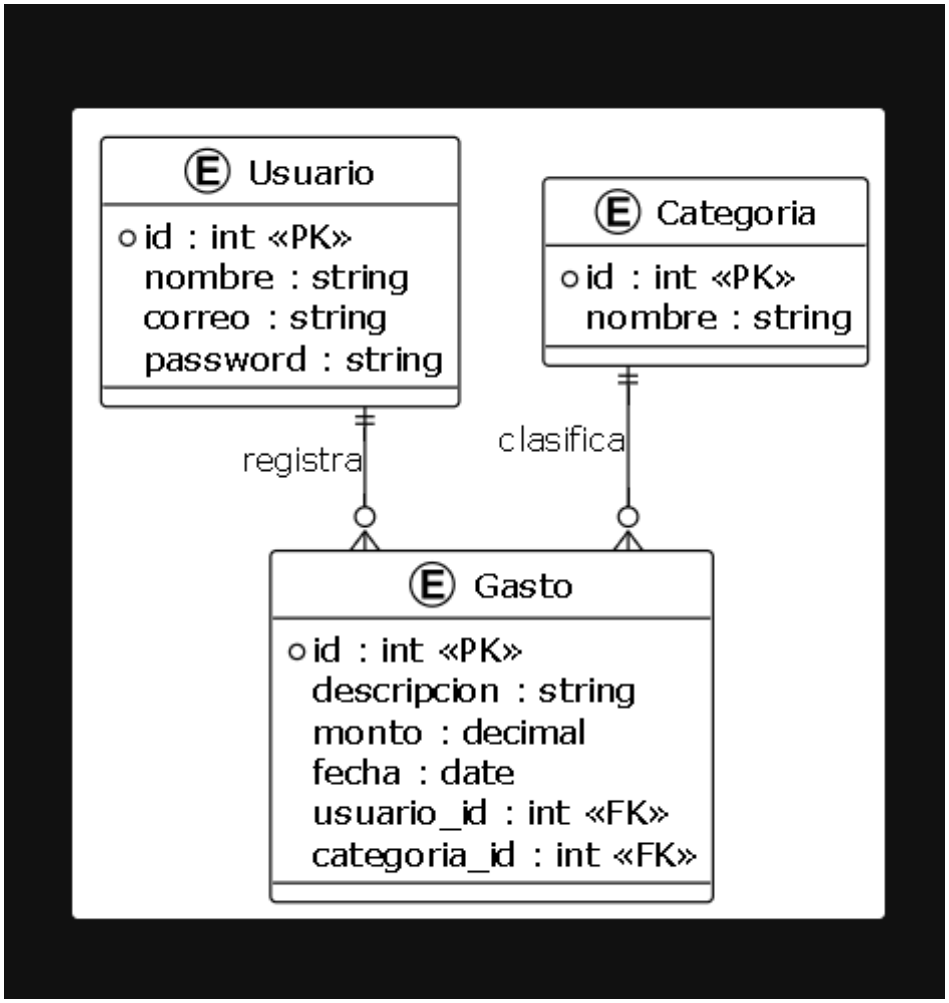
La Figura 1 ilustra la arquitectura del sistema Cashpilot bajo el patrón MVC, mostrando la distribución de responsabilidades entre el frontend en Ionic-Angular y el backend en Node.js con Express, así como la comunicación entre ambas capas mediante peticiones HTTP y respuestas JSON.

Tabla 1. Diagrama de arquitectura MVC



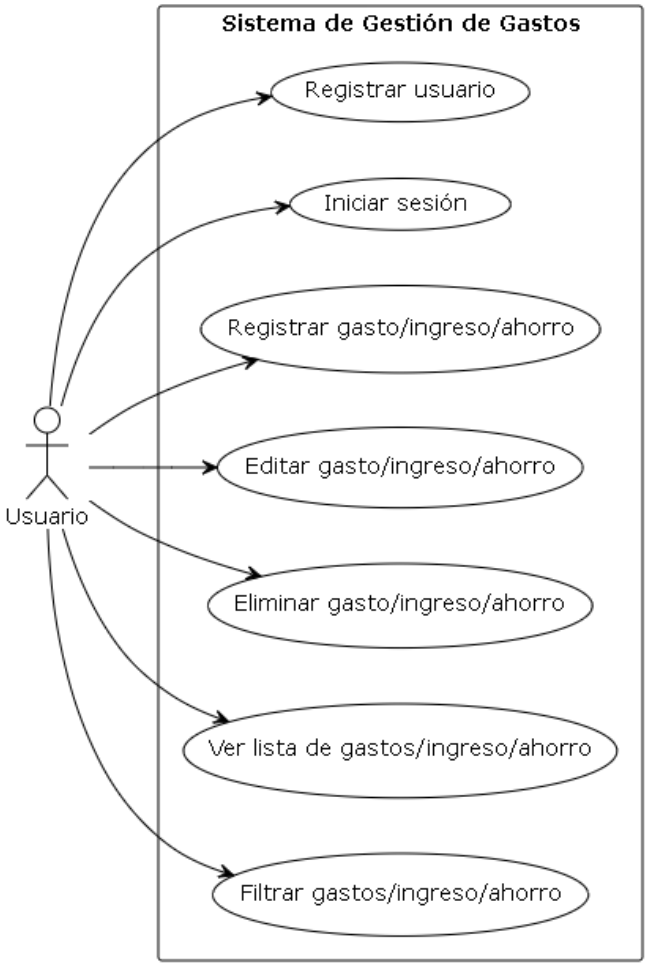
La Figura 2 muestra el modelo entidad-relación del sistema, el cual define las entidades principales que conforman la base de datos de Cashpilot, incluyendo usuarios, gastos, ingresos y categorías, así como las relaciones existentes entre ellas.

Tabla 2. Modelo entidad-relación



El diagrama de casos de uso presentado a continuación describe las interacciones principales entre el usuario y el sistema, identificando las funcionalidades que Cashpilot debe ofrecer para satisfacer las necesidades del usuario, tales como el registro de ingresos y gastos, la consulta de resúmenes mensuales, la aplicación de filtros por fecha y la visualización de gráficas.

Tabla 3. Diagrama de caso de uso



La Tabla 1 resume las tecnologías empleadas en el desarrollo de Cashpilot, especificando su función dentro del proyecto

Tabla 4. Tecnologías utilizadas

Tecnología	**Tipo**	**Función en el proyecto**
Ionic	Framework frontend	Construcción de la interfaz móvil
Angular	Framework frontend	Arquitectura de componentes y lógica de la vista
Node.js	Entorno de ejecución	Base del servidor backend
Express	Framework backend	Definición de rutas y endpoints de la API REST
HTTP Client	Librería Angular	Comunicación entre frontend y backend

7. Conclusiones:

- El desarrollo de CashPilot demostró, en realidad, que la creación de una app es diferente a lo que uno pensaría teóricamente. Durante el proceso, se notó que varios conceptos vistos en clase, aun pareciendo simples al principio, resultaron más complicados al implementarlos, presentando desafíos y mucho aprendizaje.
- El patrón MVC al comienzo fue difícil de entender, pero después de aplicarlo, ayudo a organizar el sistema bien. La separación en capas simplifico el trabajo en equipo y evito desorden, probando ser muy importante en proyectos reales.
- Usar Ionic y Angular facilito el desarrollo de las interfaces de la aplicación, poniendo en práctica los conocimientos aprendidos en el seminario. Así fue posible llevar las ideas teóricas a una solución funcional en móviles.
- Además, desarrollar el backend con Nodejs y Express significó un gran reto, en particular al construir la API REST y conectar con la base de datos.
  Aun con esas dificultades, sirvieron para robustecer el entendimiento de la comunicación frontend y backend, cierto.

- Mediante la aplicación de la metodología SCRUM, se logró organizar el proyecto a lo largo de su construcción. Definir roles, además del trabajo por Sprint, facilitaron el progreso y aseguraron la claridad en las responsabilidades.

En resumen, el seminario entrego herramientas prácticas, las cuales se usaron en el desarrollo de CashPilot mostrando el aprendizaje, que se obtuvo en el camino.

8. Referencias

- Ionic Framework. (2024). *Ionic documentation*. Ionic. https://ionicframework.com/docs
- Angular. (2024). *Angular documentation*. Google. https://angular.io/docs
- OpenJS Foundation. (2024). *Node.js documentation*. OpenJS Foundation. https://nodejs.org/en/docs
- Express.js. (2024). *Express documentation*. OpenJS Foundation. https://expressjs.com
- Schwaber, K., y Sutherland, J. (2020). *La guía de Scrum: la guía definitiva de Scrum, las reglas del juego*. Scrum.org. https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Spanish-Latin-South-American.pdf
- Fowler, M. (2002). *Patterns of enterprise application architecture*. Addison-Wesley Professional.

- Pressman, R., y Maxim, B. (2021). *Ingeniería del software: un enfoque práctico* (9.a ed.). McGraw-Hill.
- Debitoor. (2023). *¿Qué son las finanzas personales?* Debitoor. https://debitoor.es/glosario/finanzas-personales
- REST API Tutorial. (2024). *What is a REST API?* https://restfulapi.net