UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de Ingeniería de Sistemas Informáticos

Máster Universitario en Ingeniería Web

Trabajo Fin de Máster

Aplicación Web para la Gestión de finanzas personales

Autor
Bate Ye

Tutor
Santiago Alonso Villaverde

Julio de 2024

AGRADECIMIENTOS

A mis padres, por apoyarme siempre en mis objetivos y brindarme toda la ayuda que necesito para lograrlos.

A mi hermano, por confiar siempre en mí y animarme en todos los momentos.

A Bea, por estar siempre a mi lado en todos los momentos difíciles y ayudándome a superar días y noches de estrés.

A mis compañeros Juan Carlos Guillen Soto y David Yareth Macaya Albericio, por hacer mucho más ameno este año del máster y ayudarme en todo lo que he necesitado.

RESUMEN

En el mundo actual, la estabilidad financiera es esencial para el bienestar personal y social. Este trabajo presenta una aplicación diseñada para ayudar a los usuarios a controlar sus ingresos y gastos de manera eficiente.

La aplicación ofrece funcionalidades clave como la creación de categorías de gastos, el registro detallado de ingresos y gastos, y la gestión de cuentas bancarias y tarjetas de crédito. Además, incluye gráficas que proporcionan una visión clara de los patrones de gastos e ingresos, facilitando decisiones financieras informadas.

El desarrollo de este proyecto se detalla a través de varios capítulos.

En el capítulo de **Introducción**, se definen los objetivos y se introduce brevemente el problema de la gestión financiera personal que se pretende solucionar con este proyecto, junto con un modelo del dominio que representa el vocabulario y los conceptos claves del dominio del proyecto.

En el capítulo de **Requisitos**, se presentan los casos de uso, un diagrama de contexto y las especificaciones de los casos de uso, junto con un prototipo de la interfaz de usuario.

El **Análisis** extiende un paso más los artefactos obtenidos del capítulo de **Requisitos**, analizando la arquitectura, los casos de uso, las clases y los paquetes utilizando un lenguaje más técnico para facilitar el entendimiento a los desarrolladores.

El capítulo de **Diseño** detalla el diseño de la arquitectura y de los casos de uso.

La **Implementación** cubre el ecosistema de desarrollo, las herramientas utilizadas y las medidas para asegurar la calidad tanto interna como externa del software.

El capítulo de **Pruebas** documenta las estrategias y resultados de las pruebas realizadas para asegurar la funcionalidad y confiabilidad del sistema.

Finalmente, el capítulo de **Gestión** aborda la planificación y administración del proyecto.

Esta aplicación está diseñada para mejorar la planificación financiera y fomentar el ahorro, contribuyendo así a una mayor estabilidad económica para sus usuarios.

ABSTRACT

In today's world, financial stability is essential for personal and social well-being. This project presents an application designed to help users efficiently manage their income and expenses.

The application offers key functionalities such as creating expense categories, detailed recording of income and expenses, and managing bank accounts and credit cards. Additionally, it includes graphs that provide a clear view of spending and income patterns, facilitating informed financial decisions.

The development of this project is detailed through several chapters.

In the **Introduction** chapter, the objectives are defined, and the problem of personal financial management that this project wants to solve is briefly introduced, along with a domain model that represents the vocabulary and key concepts of the project's domain.

In the **Requirements** chapter, use cases, a context diagram, and detailed specifications of the use cases are presented, along with a prototype of the user interface.

The **Analysis** chapter extends the artifacts obtained from the **Requirements** chapter, analyzing the architecture, use cases, classes and packages using more technical language to facilitate understanding for developers.

The **Design** chapter details the architecture design and use cases.

The **Implementation** chapter covers the development ecosystem, the tools used, and measures to ensure both internal and external software quality.

The **Testing** chapter documents the strategies and results of the tests conducted to ensure the system's functionality and reliability.

Finally, the **Management** chapter addresses the planning and administration of the project.

This application is designed to improve financial planning and promote savings, thereby contributing to greater economic stability for its users.

ÍNDICE

CAPÍTULO 1. INTRODUCCIÓN

1. Objetivos

El principal objetivo de este proyecto es aplicar los diferentes conocimientos adquiridos a lo largo del Máster de Ingeniería Web con el desarrollo de una aplicación real, destacando sobre todo, las siguientes asignaturas:

- Ingeniería Web Visión General.
- Arquitectura y Patrones para Aplicaciones Web.
- Front-end para Navegadores Web.
- Back-end con Tecnologías de Código Abierto.
- Metodologías de desarrollo Web

La aplicación que se desarrollará se denomina *MaBills*, una aplicación web para la gestión de finanzas personales, cuyo objetivo es ayudar a los usuarios a tener un mejor control sobre sus gastos e ingresos de forma centralizada, personalizada y simplificada.

La metodología que se seguirá para el desarrollo de este proyecto es *RUP* (*Rational Uniform Process*), se utilizará *Spring* para el desarrollo de la capa de negocio, *Angular* para la capa de presentación y *MySQL* para la capa de datos.

El proyecto contiene 2 actores y 21 casos de usos. La decisión de realizar este proyecto es debido a su reducido tamaño, ya que sólamente será desarrollado por un alumno, por lo cual, el tiempo de desarrollo es limitado.

2. Introducción al problema

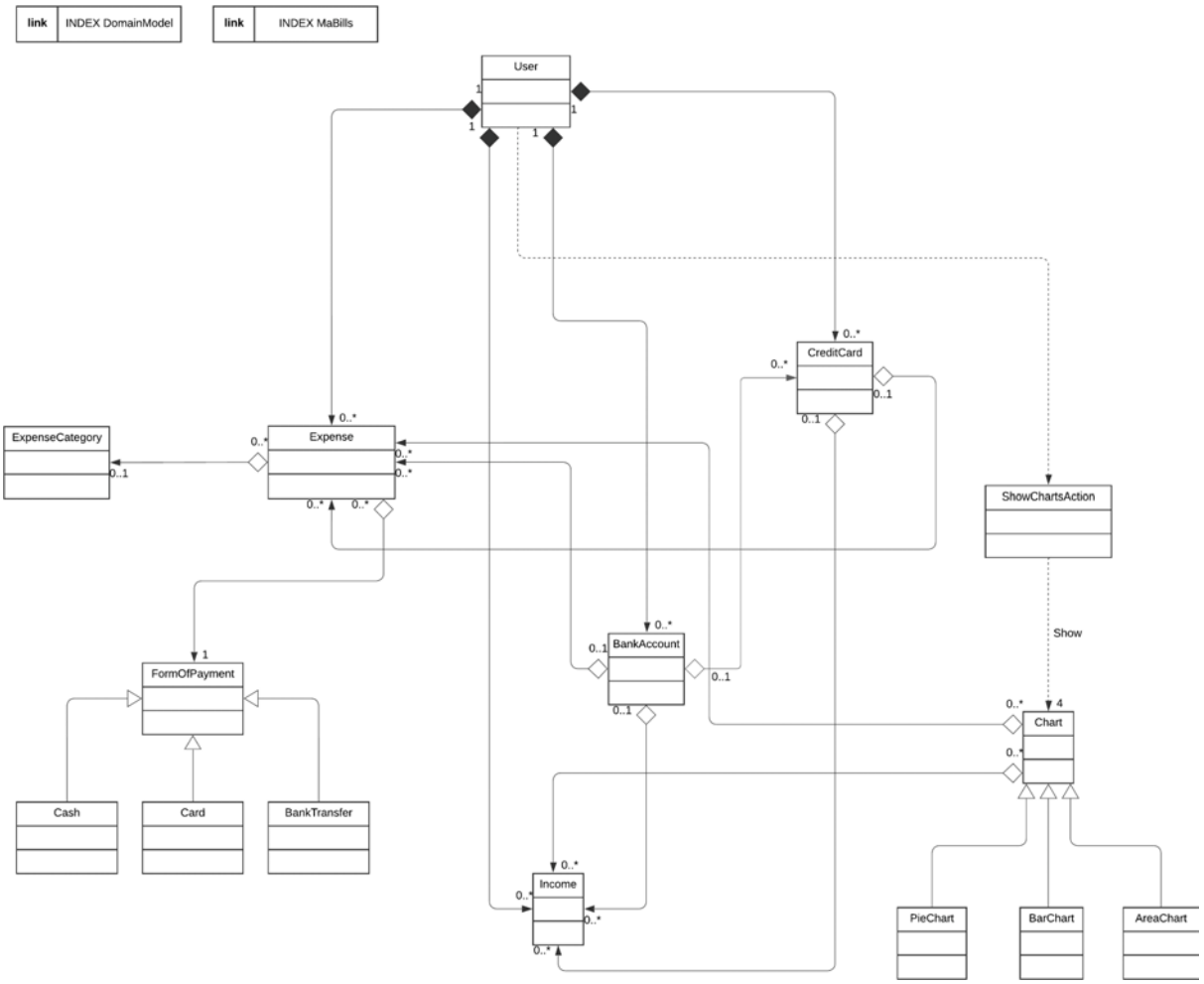
En el mundo actual, donde la estabilidad financiera es vital, la gestión efectiva de las finanzas personales se vuelve imprescindible. Este trabajo presenta una aplicación diseñada para facilitar a los usuarios el control de sus ingresos y gastos de manera eficiente.

La aplicación proporciona funcionalidades clave, como la creación de categorías de gastos, el registro detallado de ingresos y gastos, así como la gestión de cuentas bancarias y tarjetas de crédito. Su enfoque se centra en la adaptabilidad, permitiendo a los usuarios gestionar tanto gastos regulares como excepcionales, permitiendo asociar cada gasto con una categoría, la aplicación también permite asociar los gastos e ingresos con tarjetas de crédito y cuentas bancarias del usuario. Además la inclusión de gráficas ofrece una visión clara y concisa de los patrones de gastos e ingresos, facilitando la toma de decisiones financieras informadas.

En resumen, esta herramienta busca proporcionar una solución integral para el control financiero, facilitando la planificación y el ahorro, con el fin de mejorar la estabilidad económica de los usuarios.

3. Modelo del dominio

Este apartado presenta el *modelo del dominio* de la aplicación, con este modelo, se pretende representar el vocabulario y los conceptos claves del dominio del proyecto. Este *modelo del dominio*, además, será la base de todo el proyecto, desde el análisis hasta finalizar la implementación del proyecto.



Observen que en el diagrama se muestran los gastos como *Expense,* los ingresos como *Income,* las tarjetas de crédito como *CreditCard,* las cuentas bancarias como *BankAccount* y las gráficas como *Chart.*

También, hay una entidad llamada FormOfPayment, que está relacionada con *Expense.* Esta entidad representa de qué manera se hizo el gasto, si se hizo por metálico (*Cash*), por tarjeta de crédito (*Card*) o por transferencia bancaria (*BankTransfer*).

CAPÍTULO 2. REQUISITOS

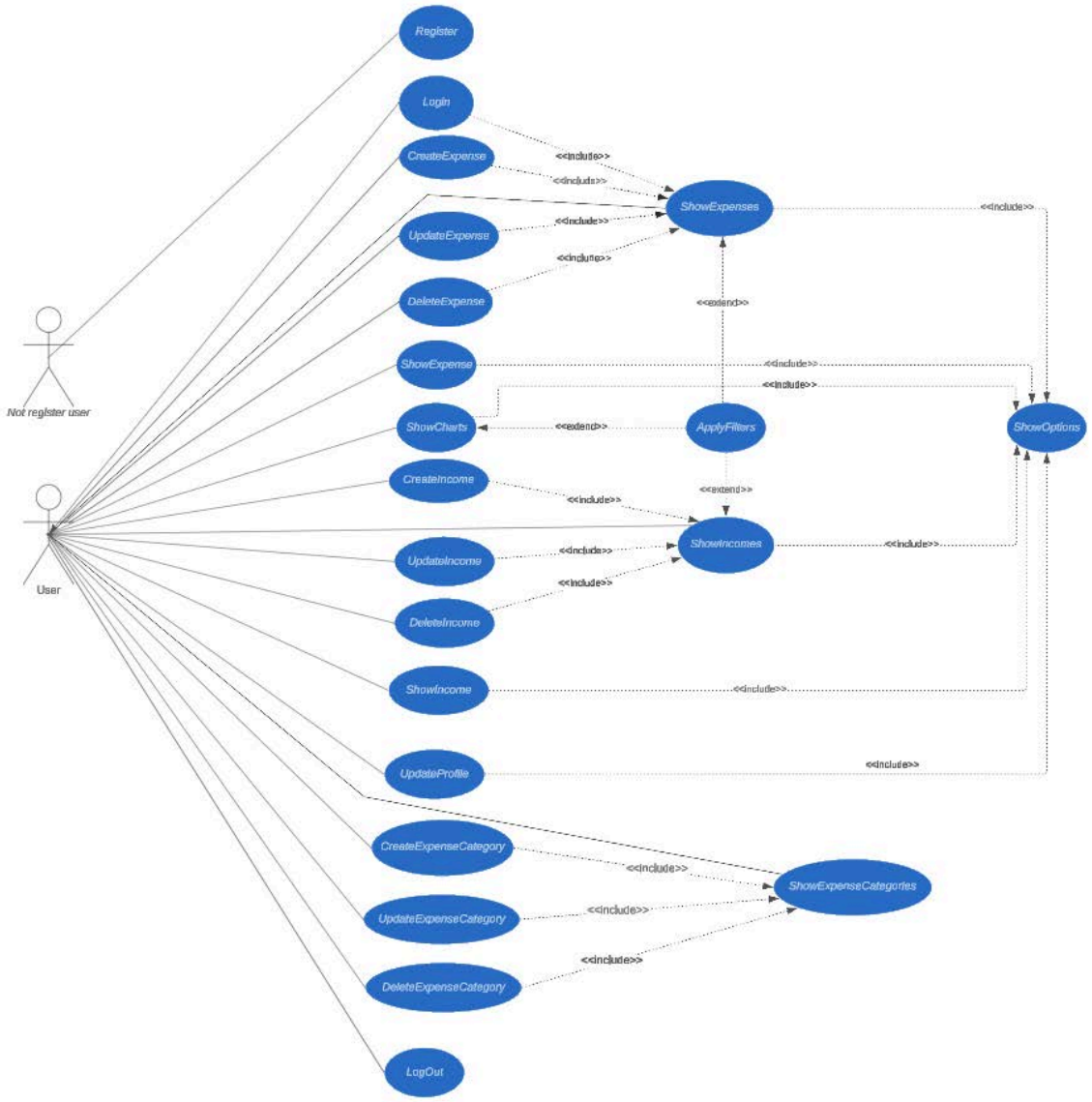
En este capítulo, se recogen todos los requisitos que guiará el correcto desarrollo de la aplicación, en los próximos capítulos se realizarán el análisis y el diseño de dichos requisitos.

1. Casos de uso

En el próximo diagrama, se muestran los dos actores y los casos de uso que dichos actores pueden ejercer en el sistema (los casos de uso definen las acciones que el actor puede ejercer sobre el sistema).

link	INDEX UseCaseView

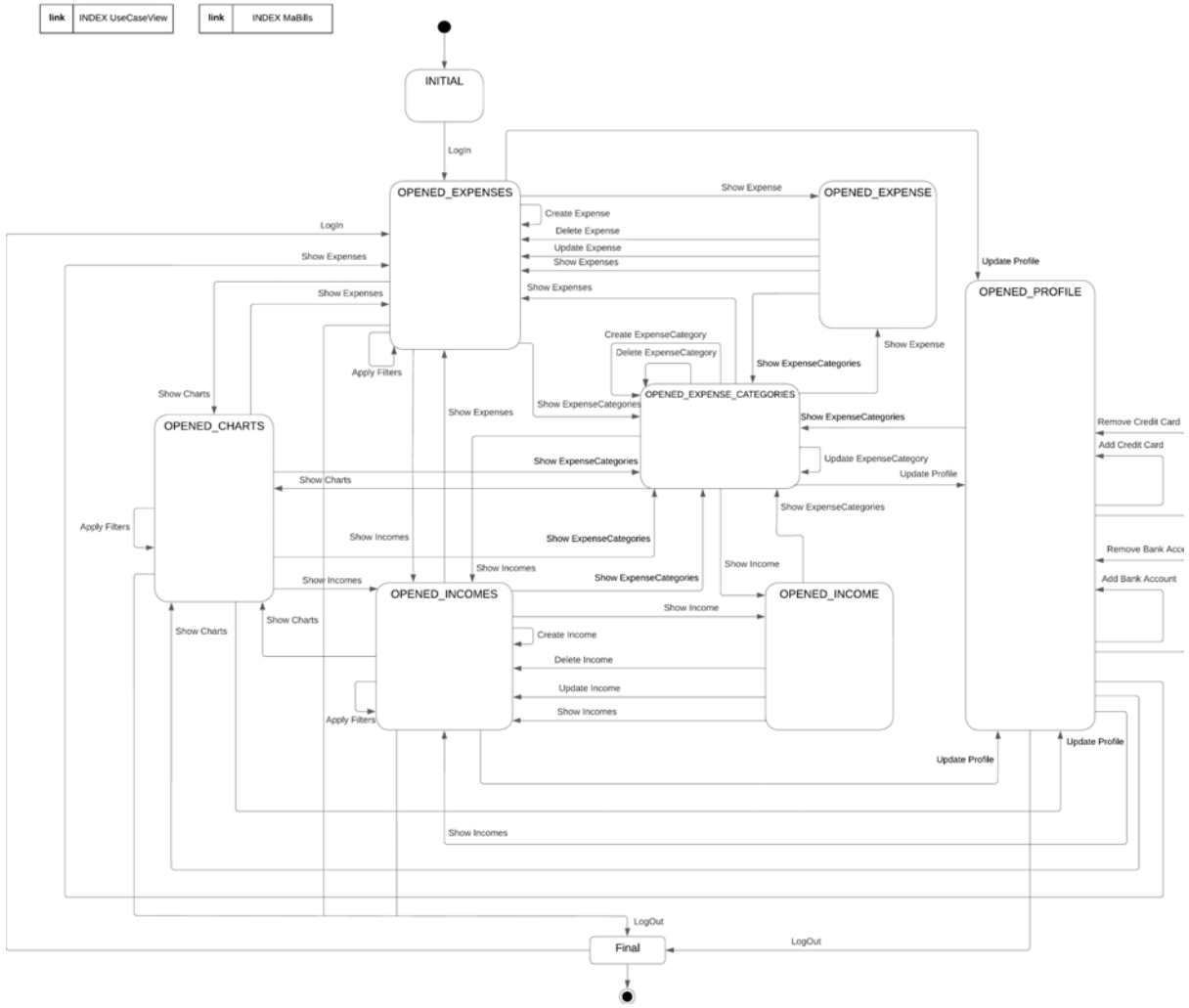
link	INDEX MaBills



Observen que, un usuario no registrado sólamente puede registrarse en el sistema para poder realizar los demás casos de uso.

2. Diagrama de contexto

En este apartado, se presenta un diagrama de contexto que muestra los diferentes estados del sistema y las acciones que lleva al usuario a ellas. Cada acción es un caso de uso, que inicia en un estado y termina en otro estado o en el mismo estado desde que se inició la acción.



Fíjense cómo desde el estado inicial, mediante la acción *Login,* el usuario pasa a estar en el estado *OPENED_EXPENSES,* esto es porque el caso de uso *Login* incluye el caso de uso *ShowExpenses.*

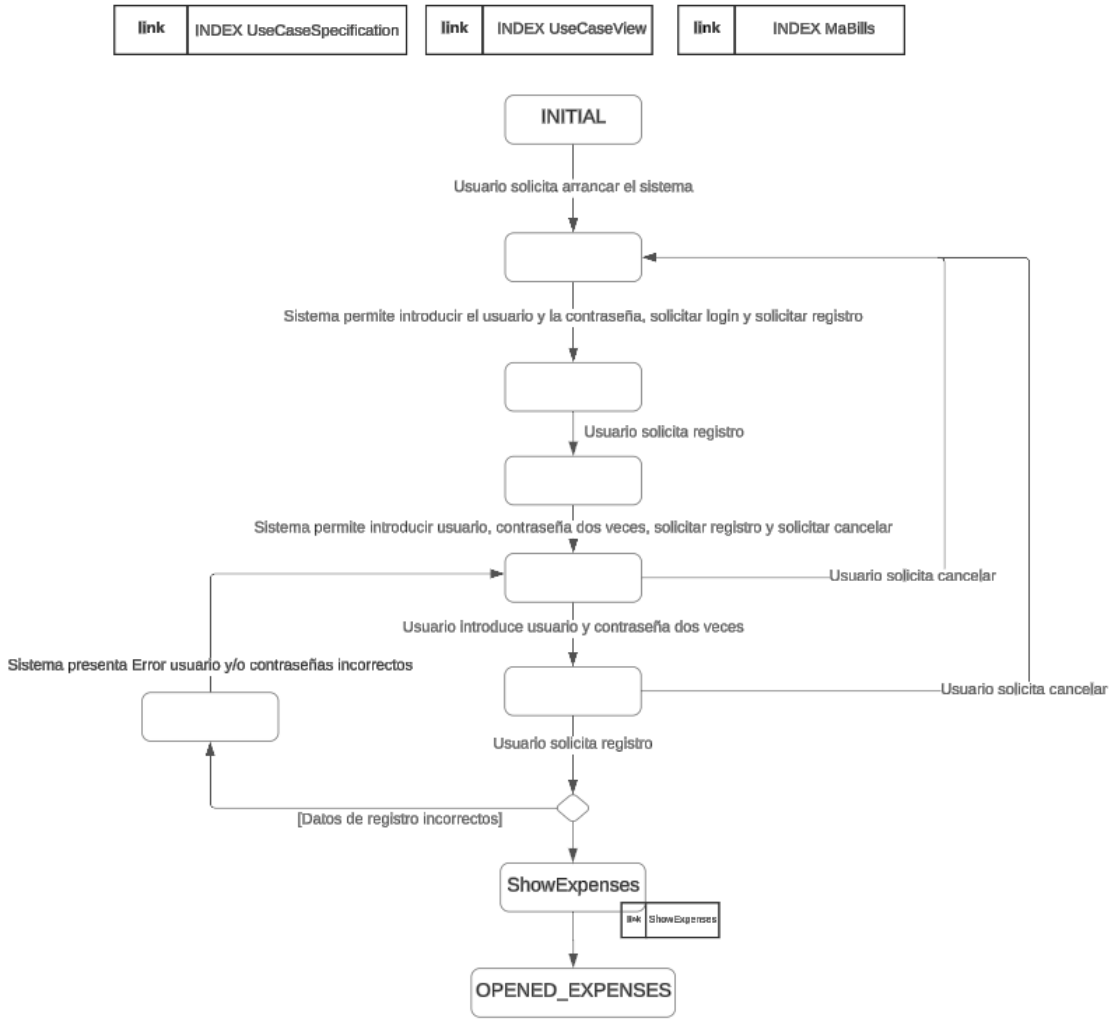
Además, desde un estado como *OPENED_EXPENSES*, el actor puede llegar a otros estados a través de diferentes acciones, por ejemplo, realizando la acción *ShowExpenseCategories*, se pasaría del estado *OPENED_EXPENSES* al estado *OPENED_EXPENSE_CATEGORIES*.

3. Especificaciones de casos de uso

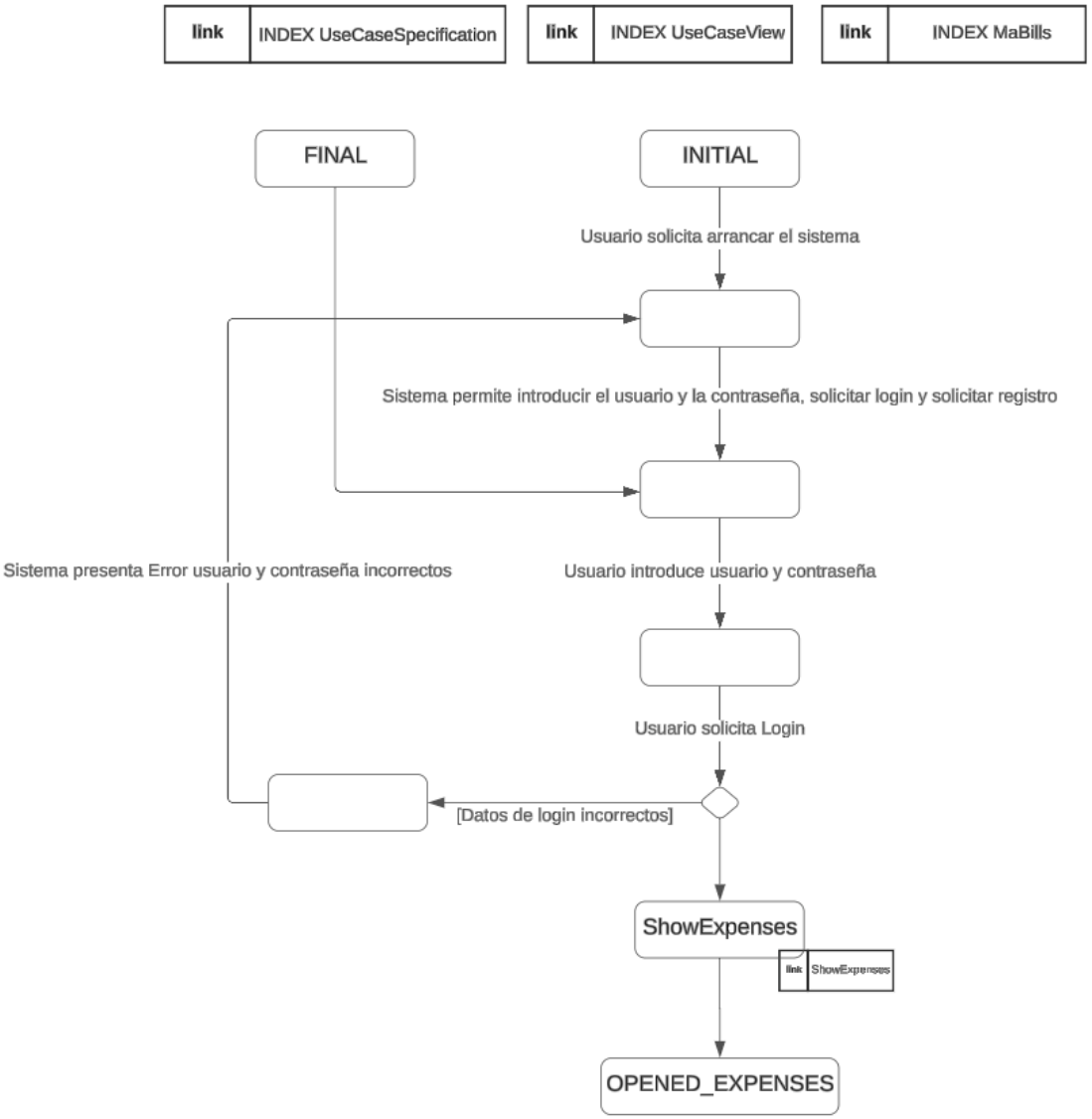
En esta sección, se presenta la secuencia de acciones de los casos de uso, explicando los inicios de cada caso de uso y su "fin", que es el resultado que se espera obtener al completar la ejecución de un caso de uso.

Por ejemplo en el caso de uso *ShowExpenses,* se puede observar que el usuario empezaría en un estado x, interactúa con el sistema realizando diferentes acciones y finalmente pasaría al estado final *OPENED_EXPENSES.*

Register

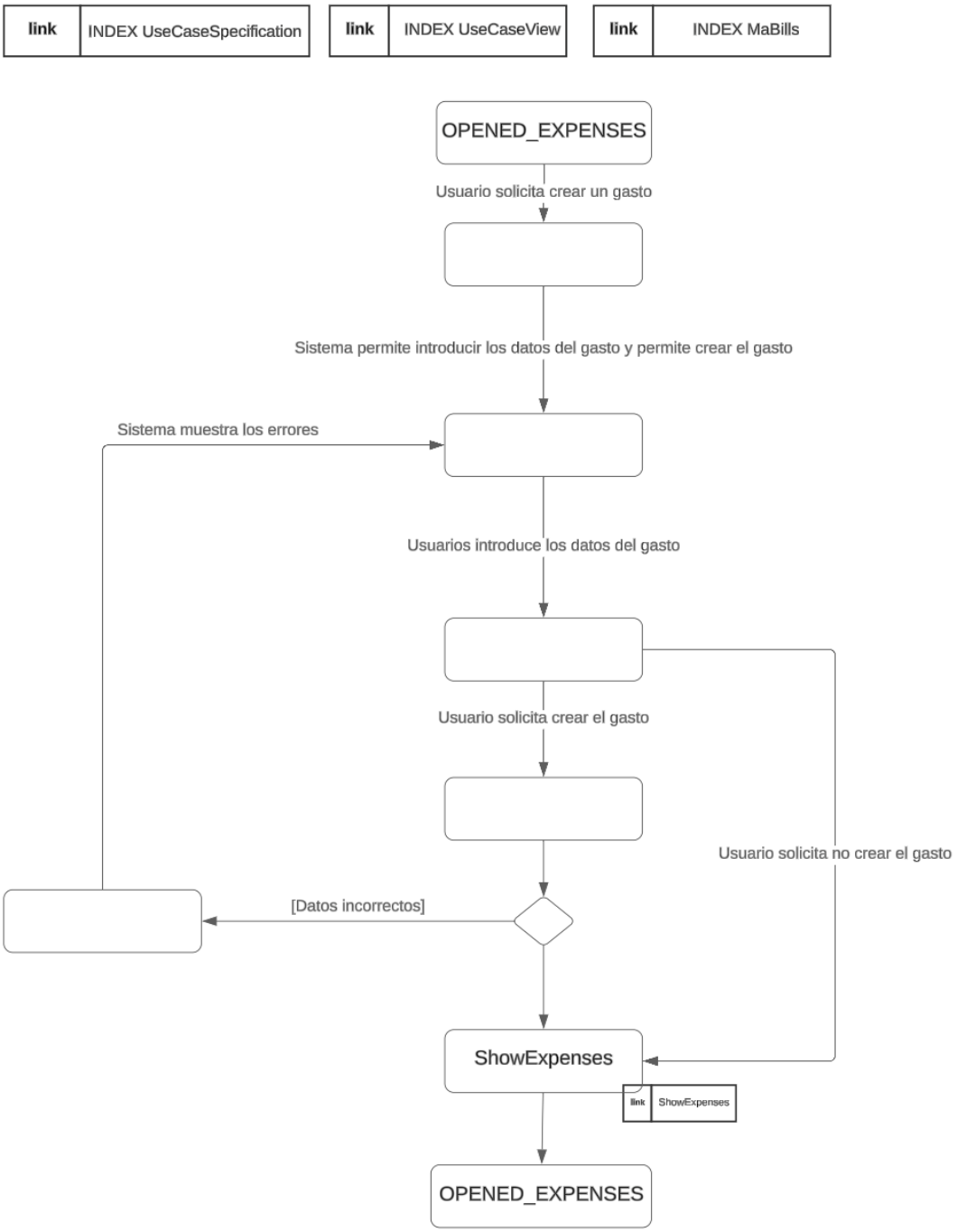


Login

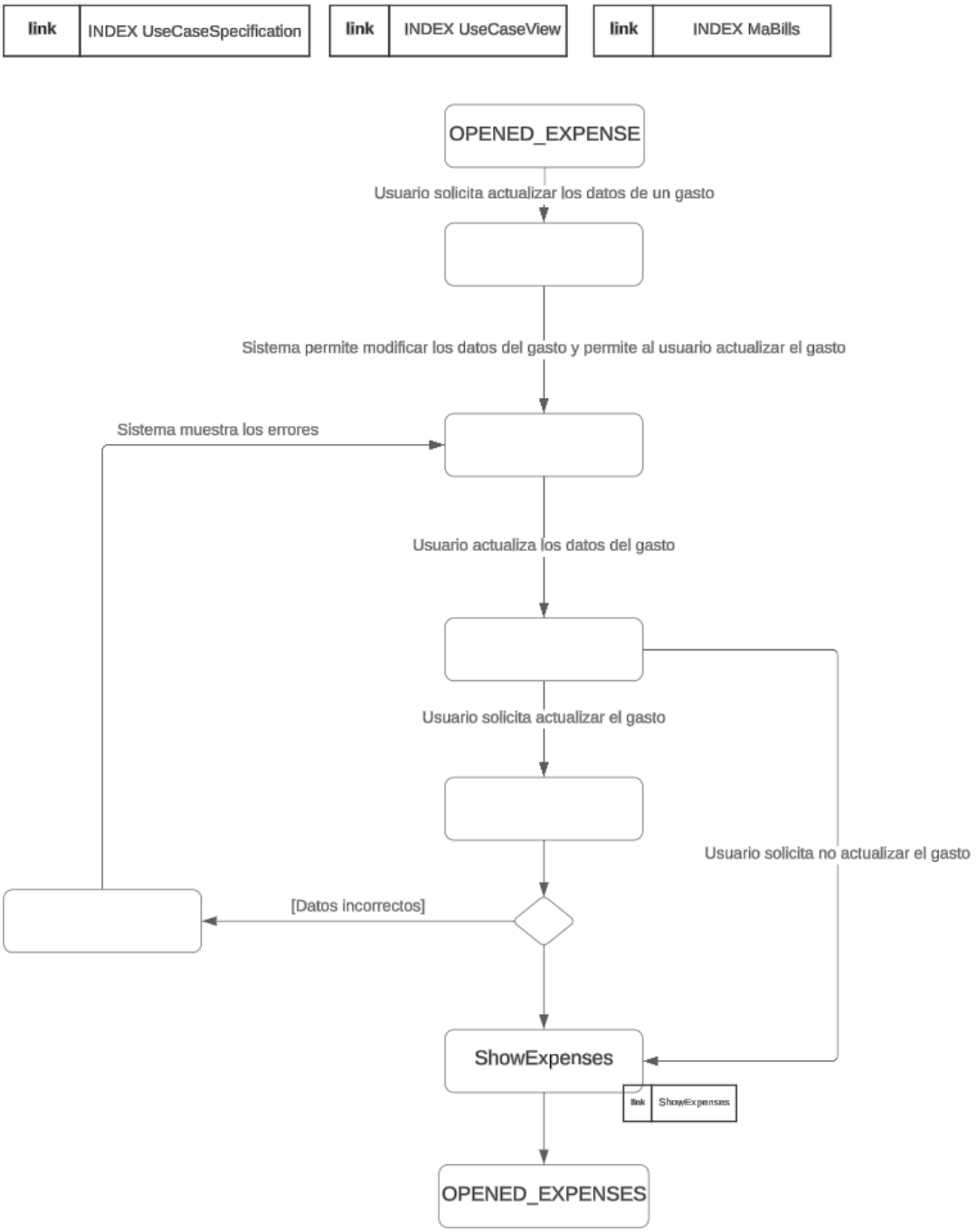

link
INDEX UseCaseSpecification
link
INDEX UseCaseView
link
INDEX MaBills
FINAL
INITIAL
Usuario solicita arrancar el sistema
Sistema permite introducir el usuario y la contraseña, solicitar login y solicitar registro
Sistema presenta Error usuario y contraseña incorrectos
Usuario introduce usuario y contraseña
Usuario solicita Login
[Datos de login incorrectos]
ShowExpenses
link
ShowExpenses
OPENED_EXPENSES

Show expenses

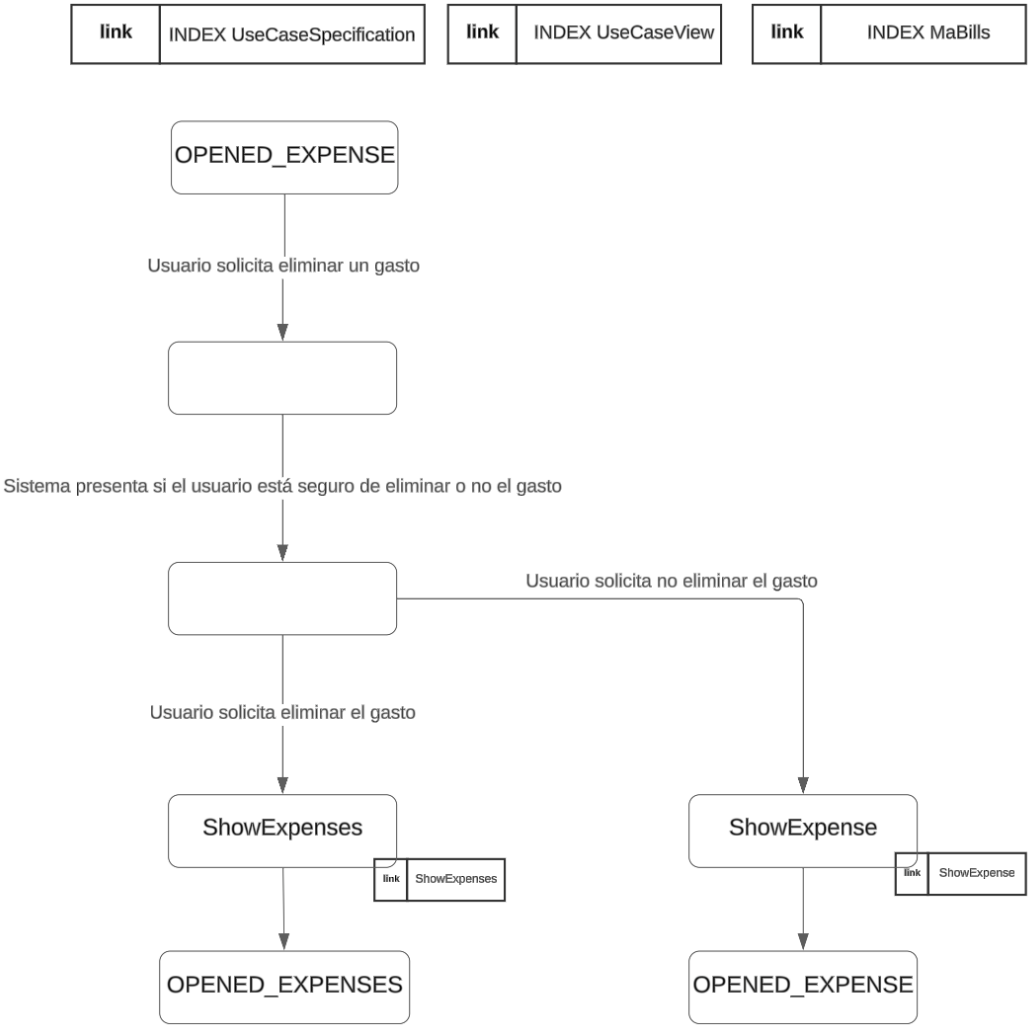
Create expense



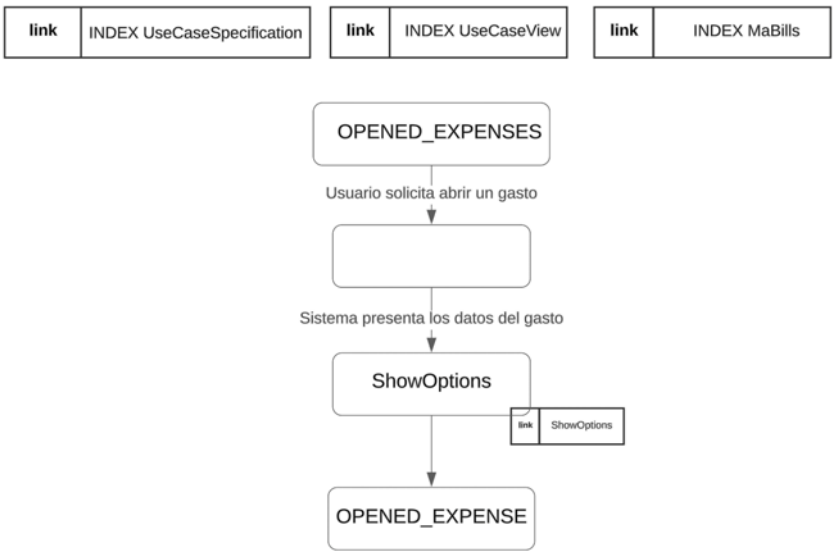
Update expense



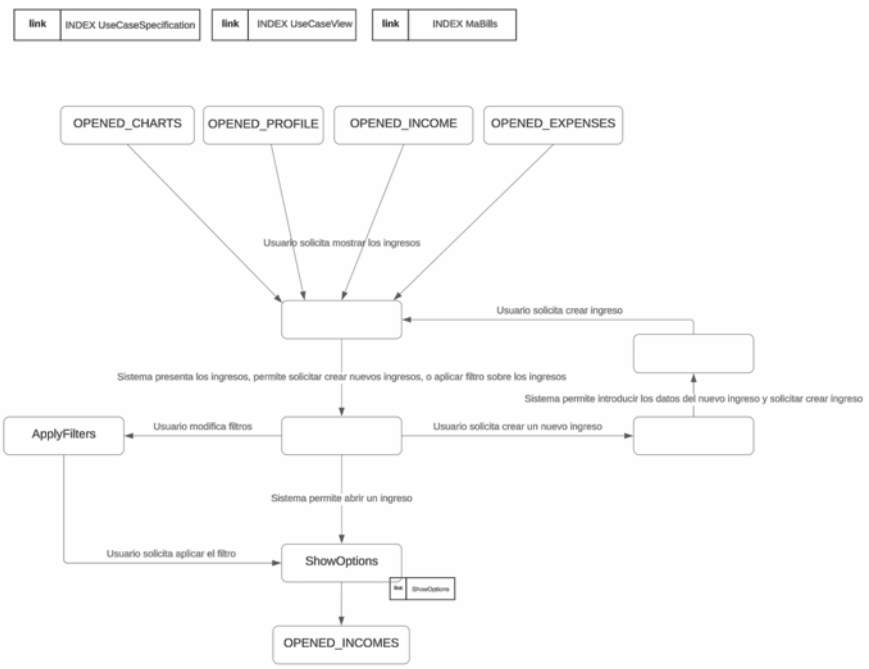
Delete expense



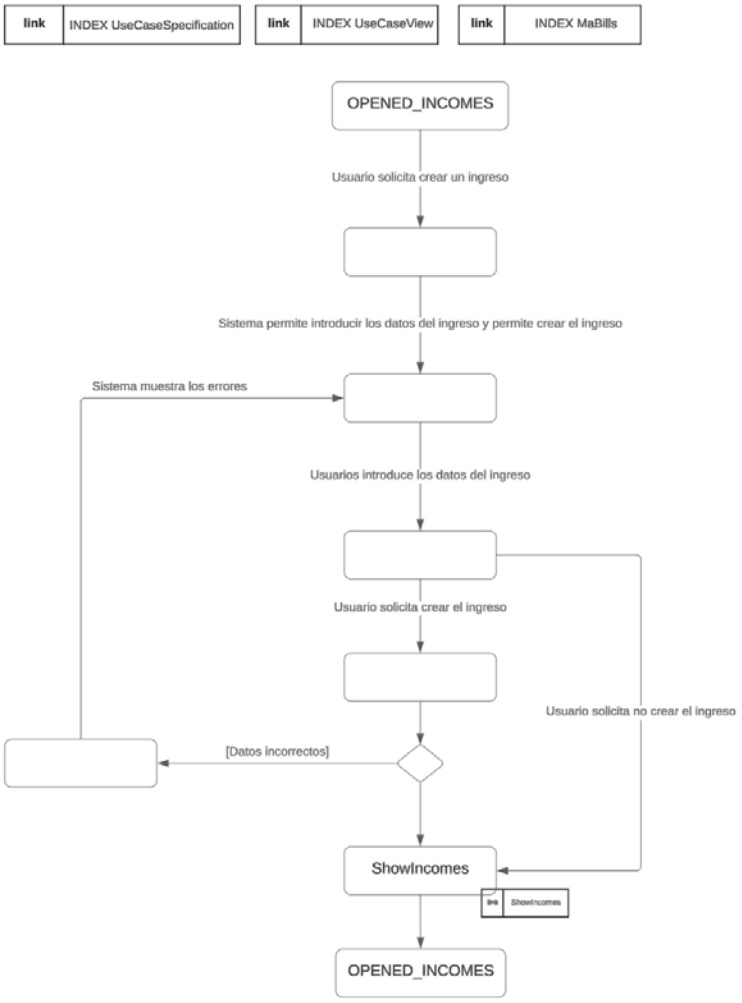
Show expense



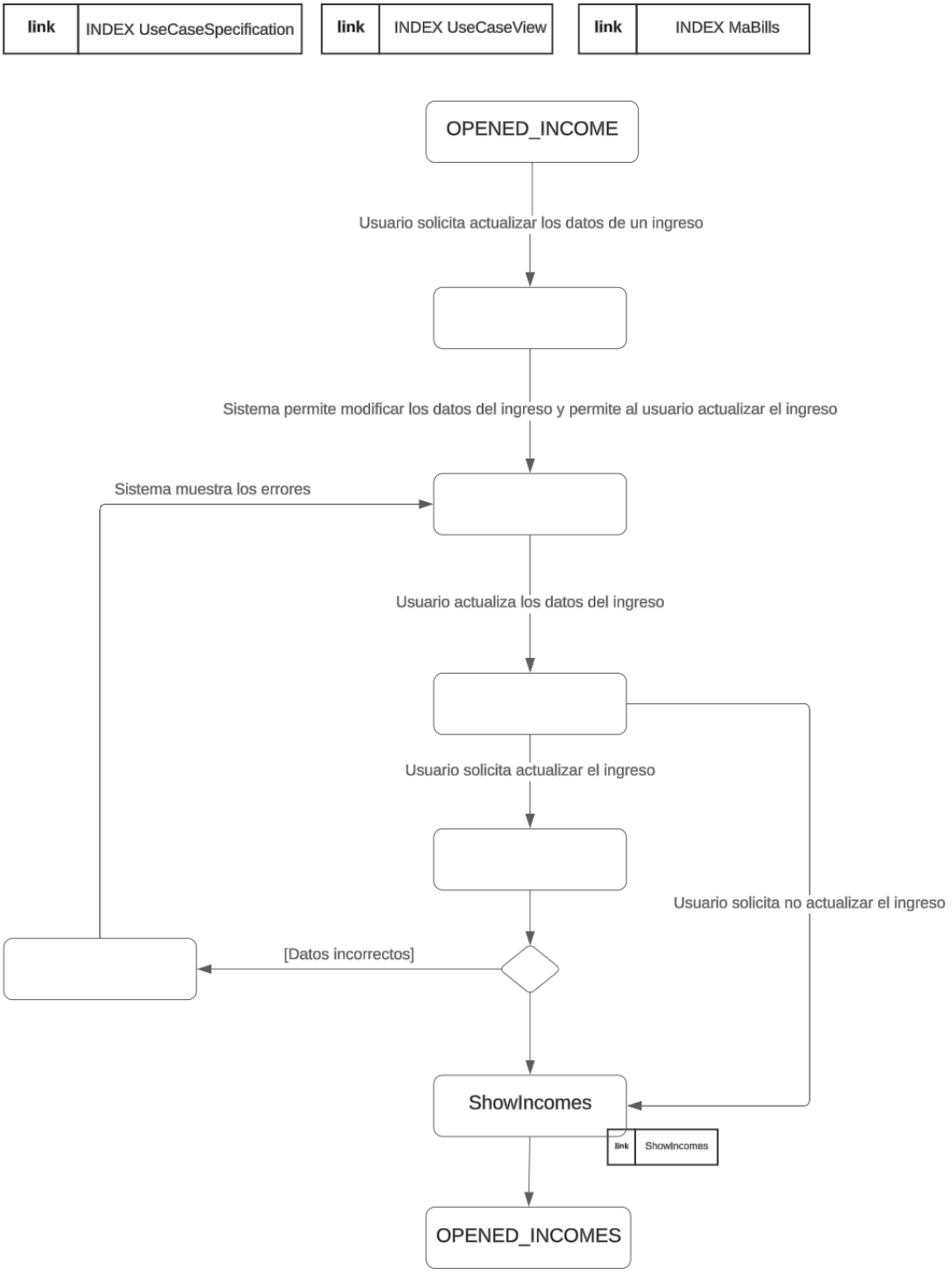
Show incomes



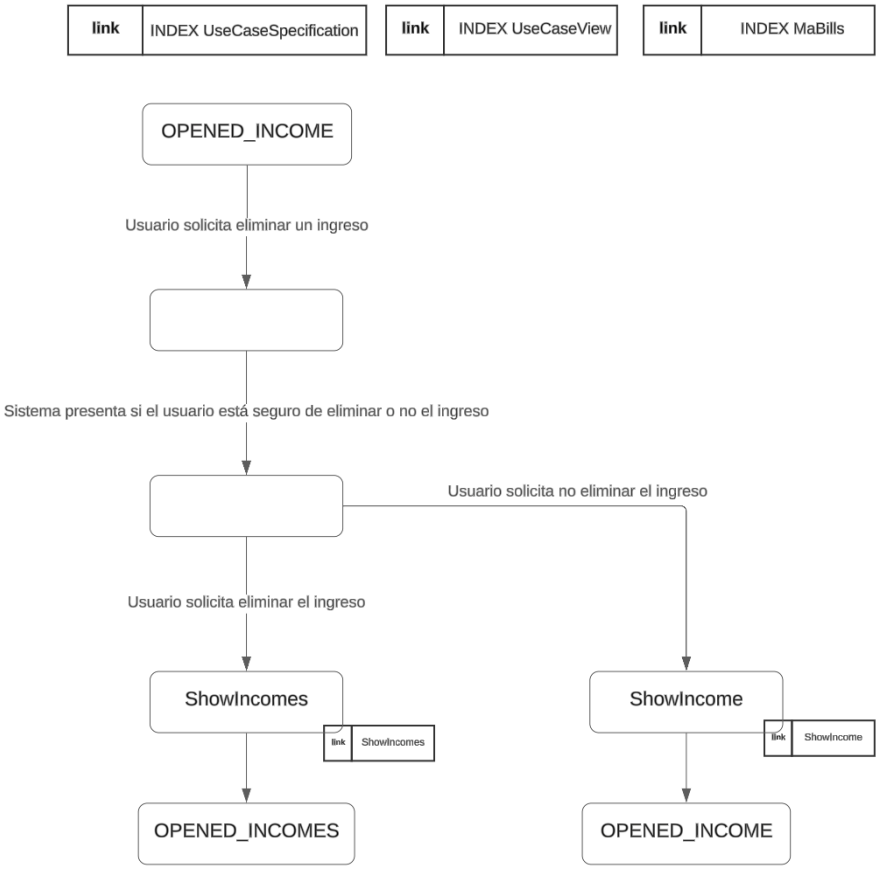
Create income



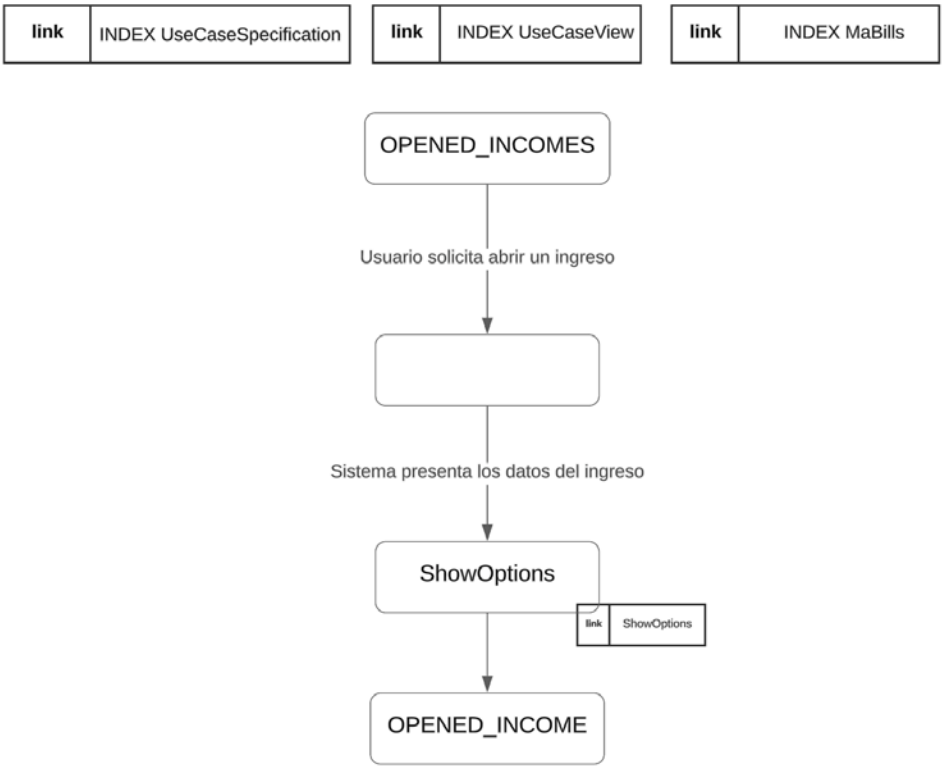
Update income



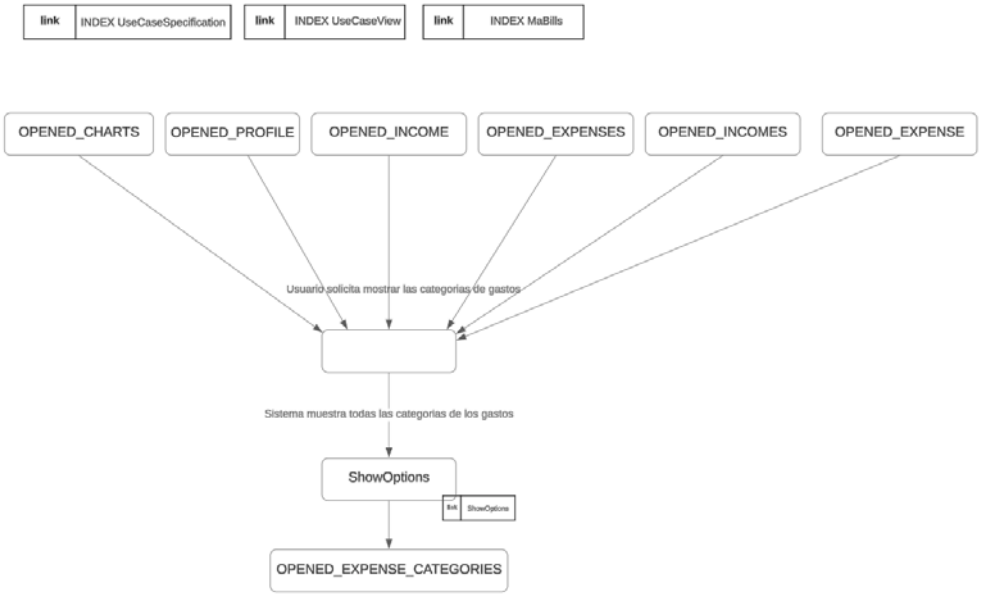
Delete income



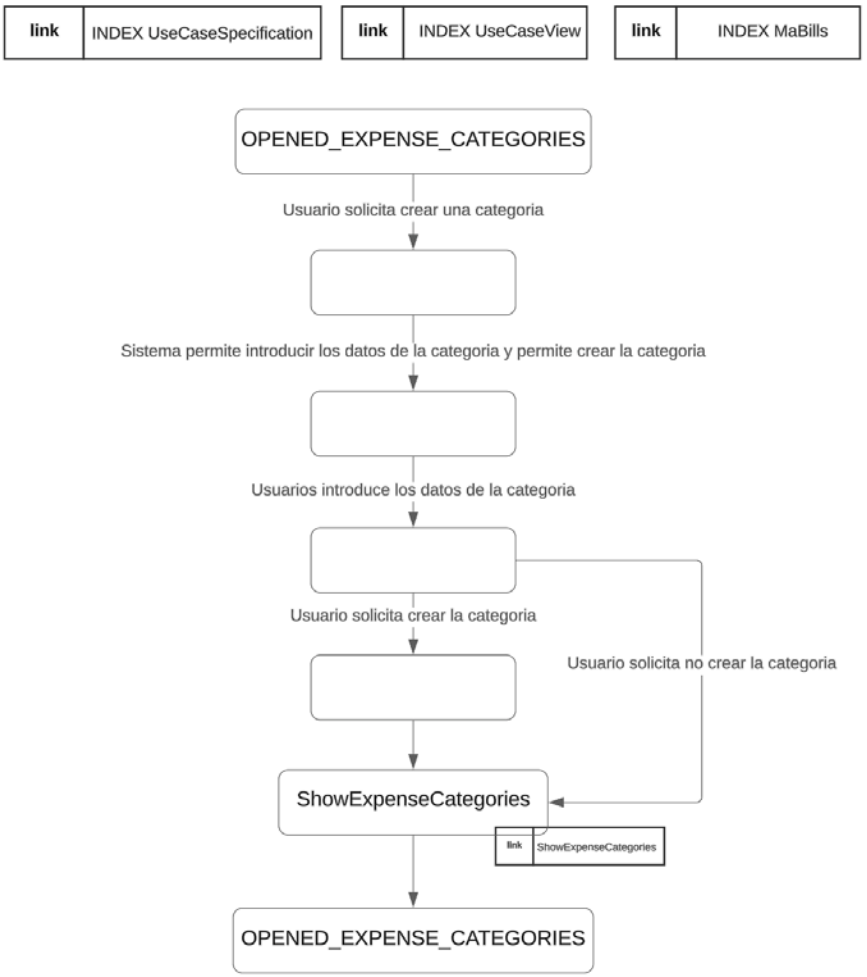
Show income



Show expense categories



Create expense category

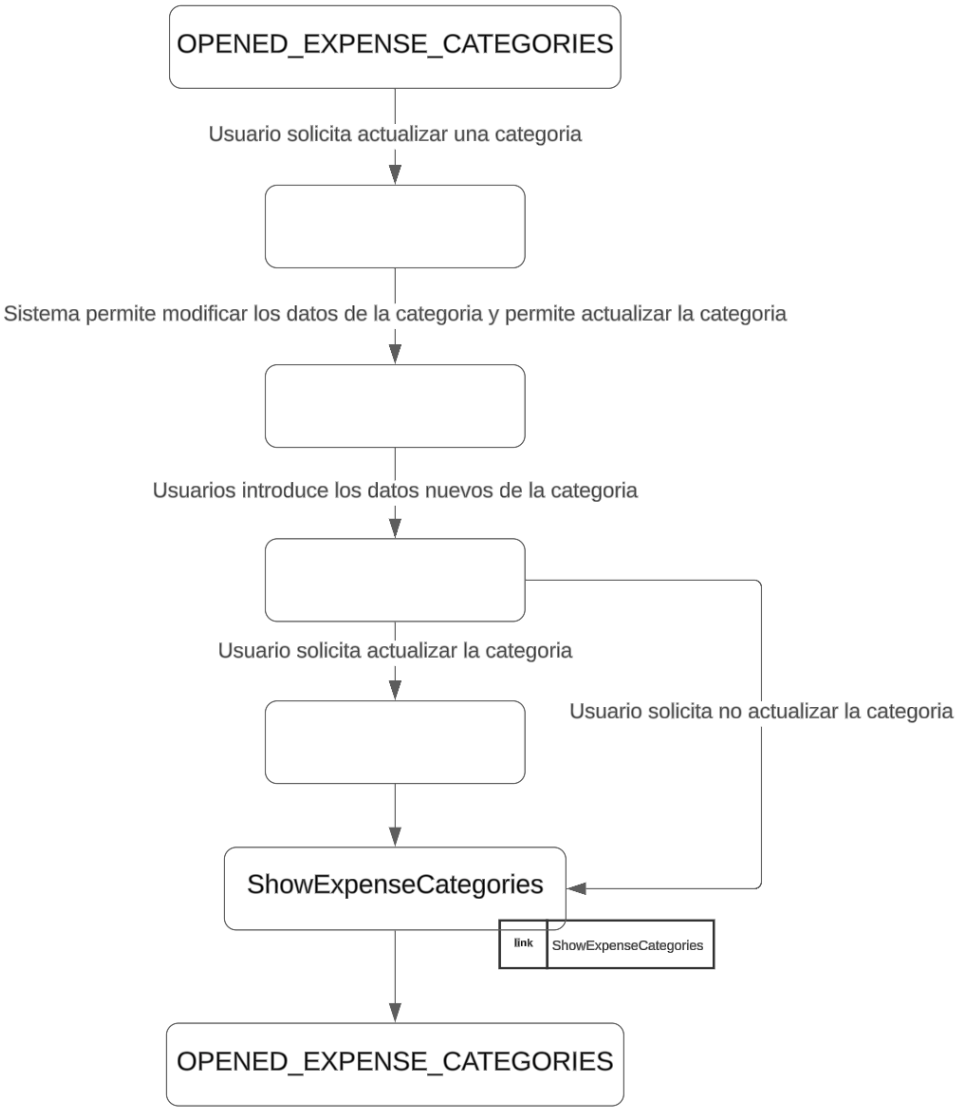


Update expense category

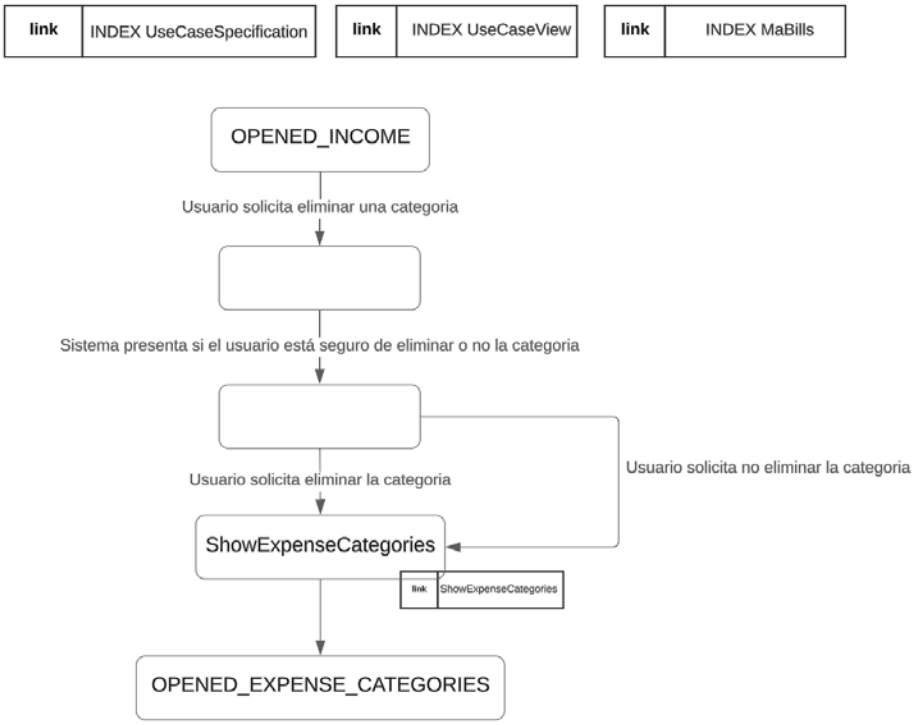
link	INDEX UseCaseSpecification

link	INDEX UseCaseView

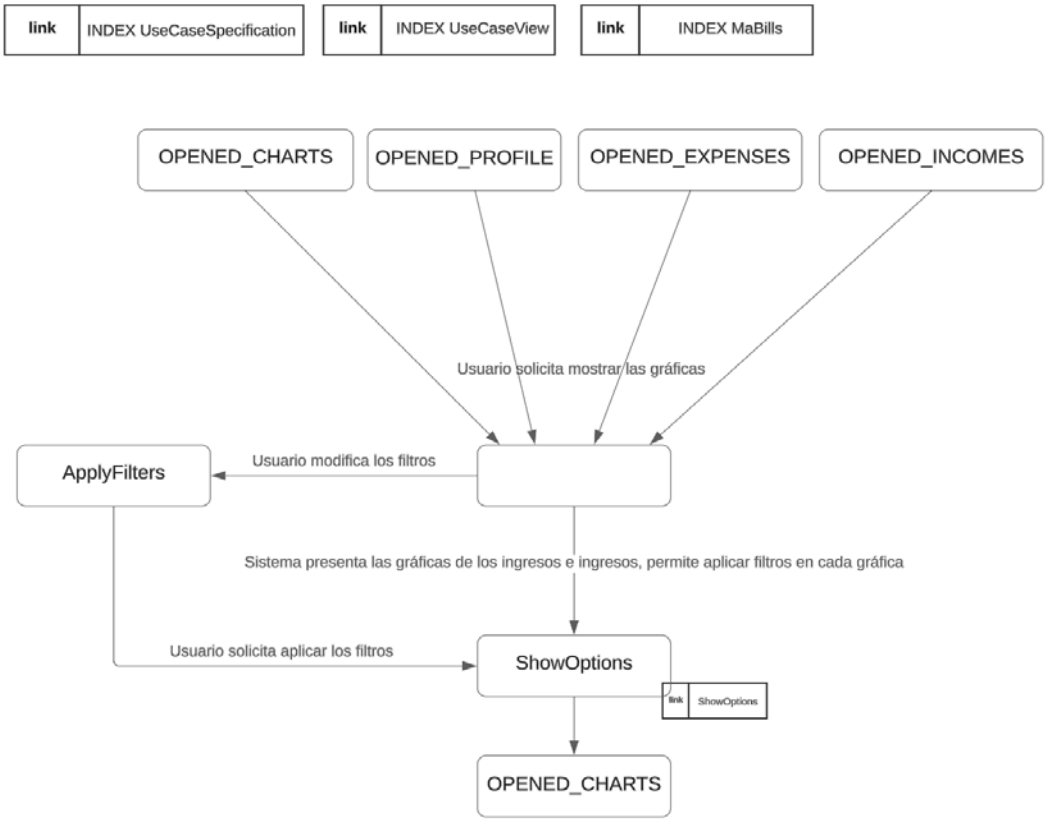
link	INDEX MaBills



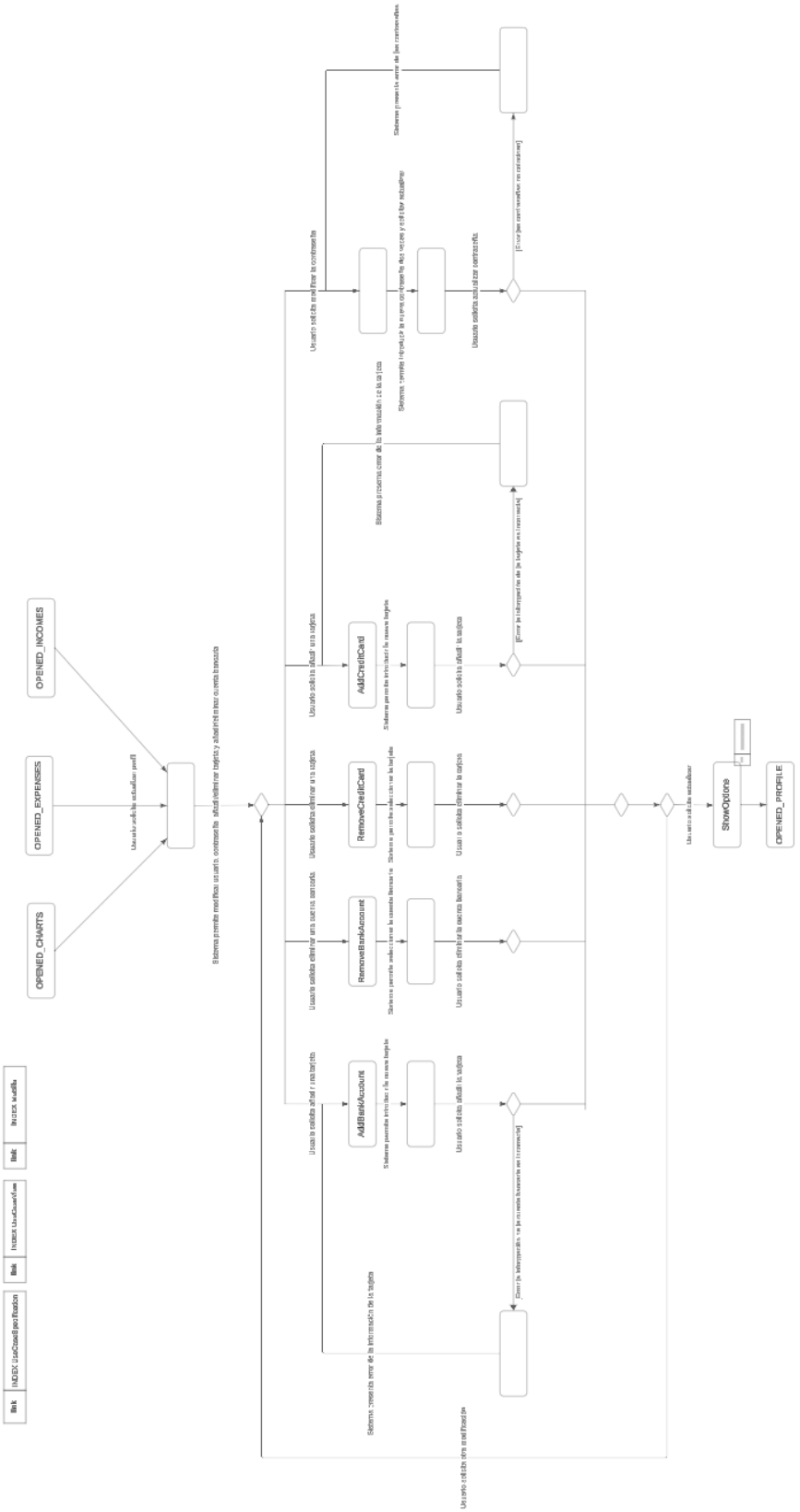
Delete expense category



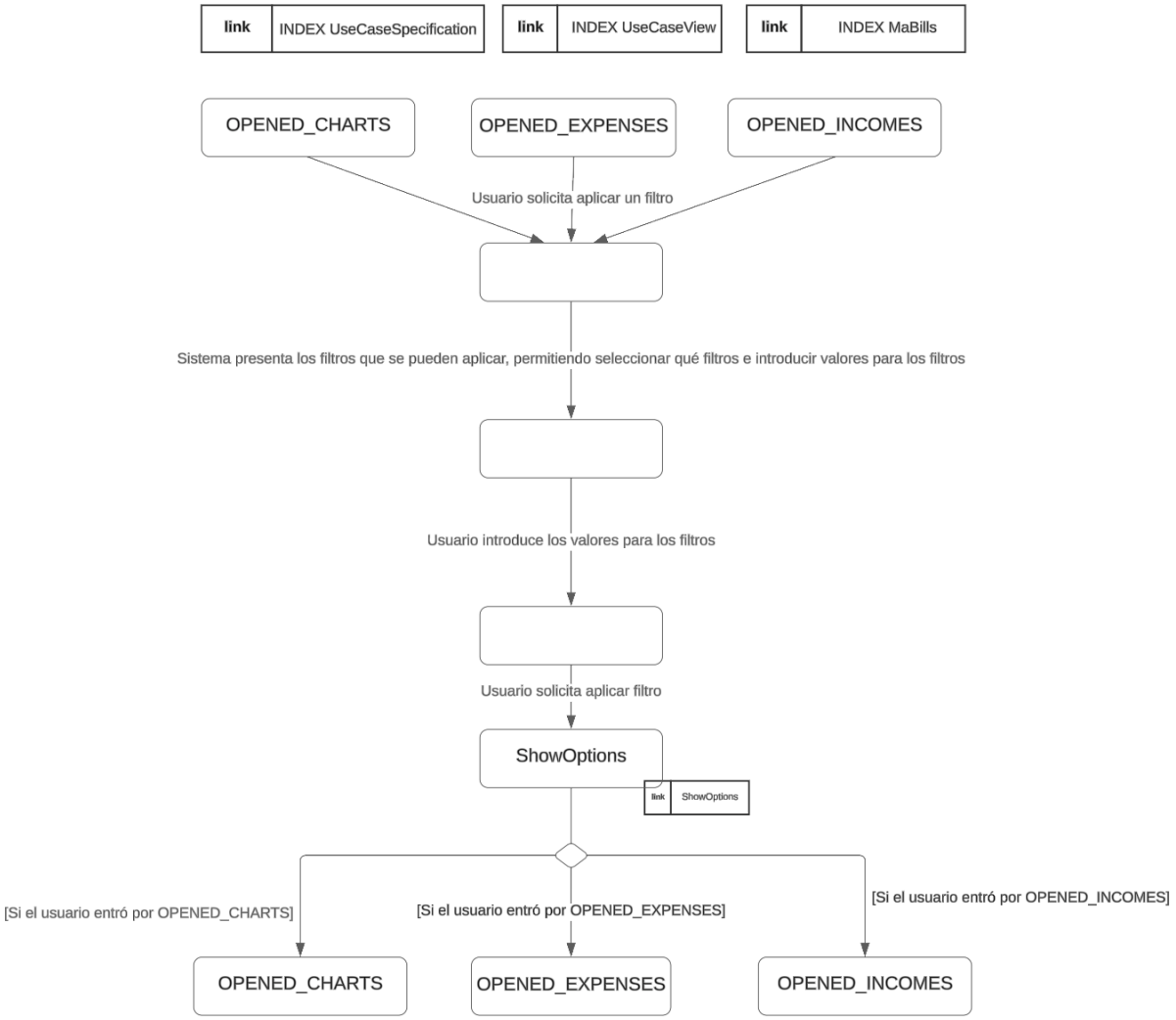
Show charts



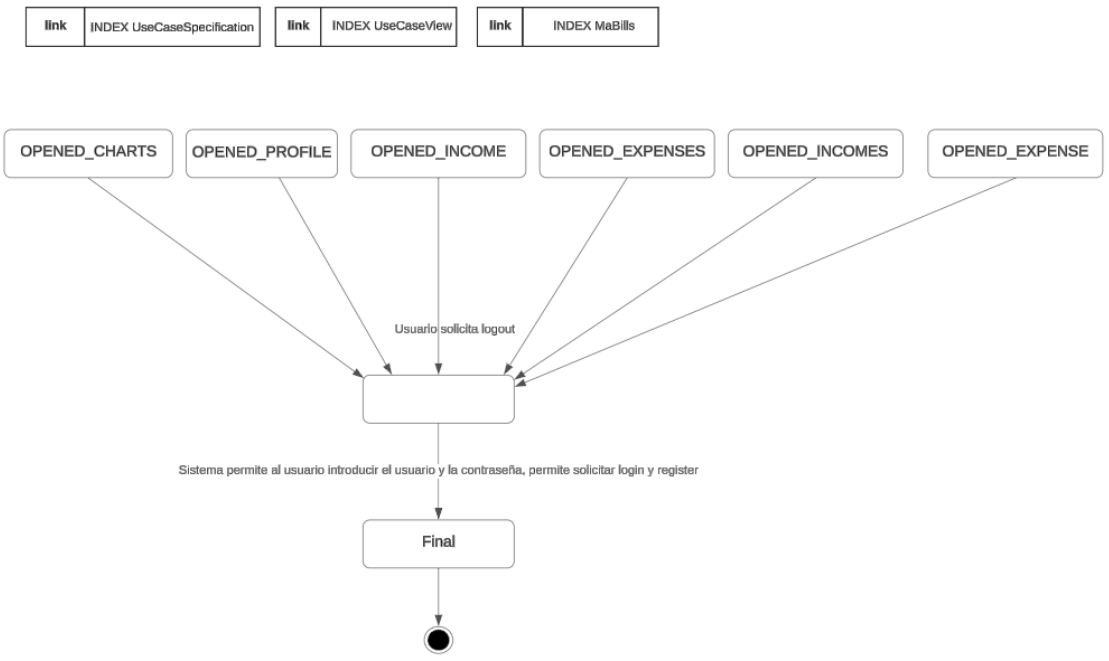
Update profile



Apply filters



Log out

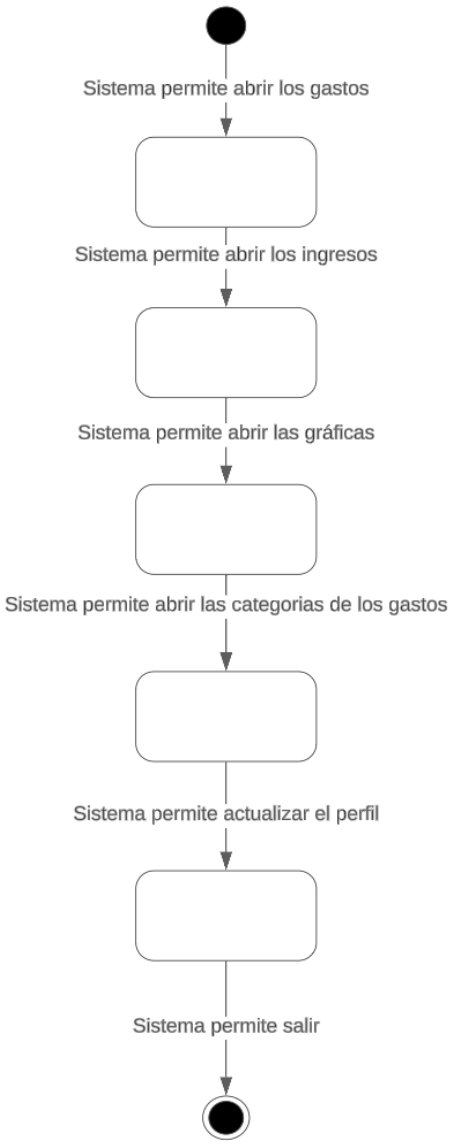


Show options

link	INDEX UseCaseSpecification

link	INDEX UseCaseView

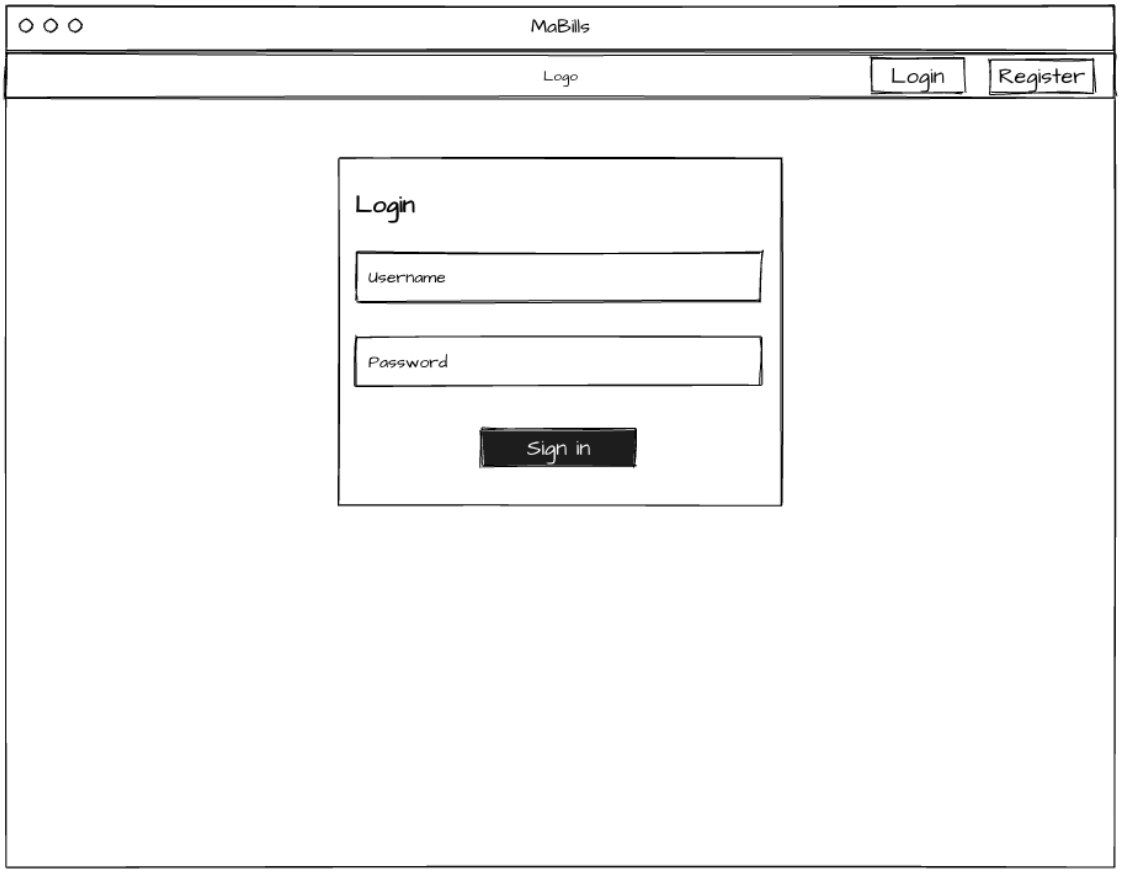
link	INDEX MaBills



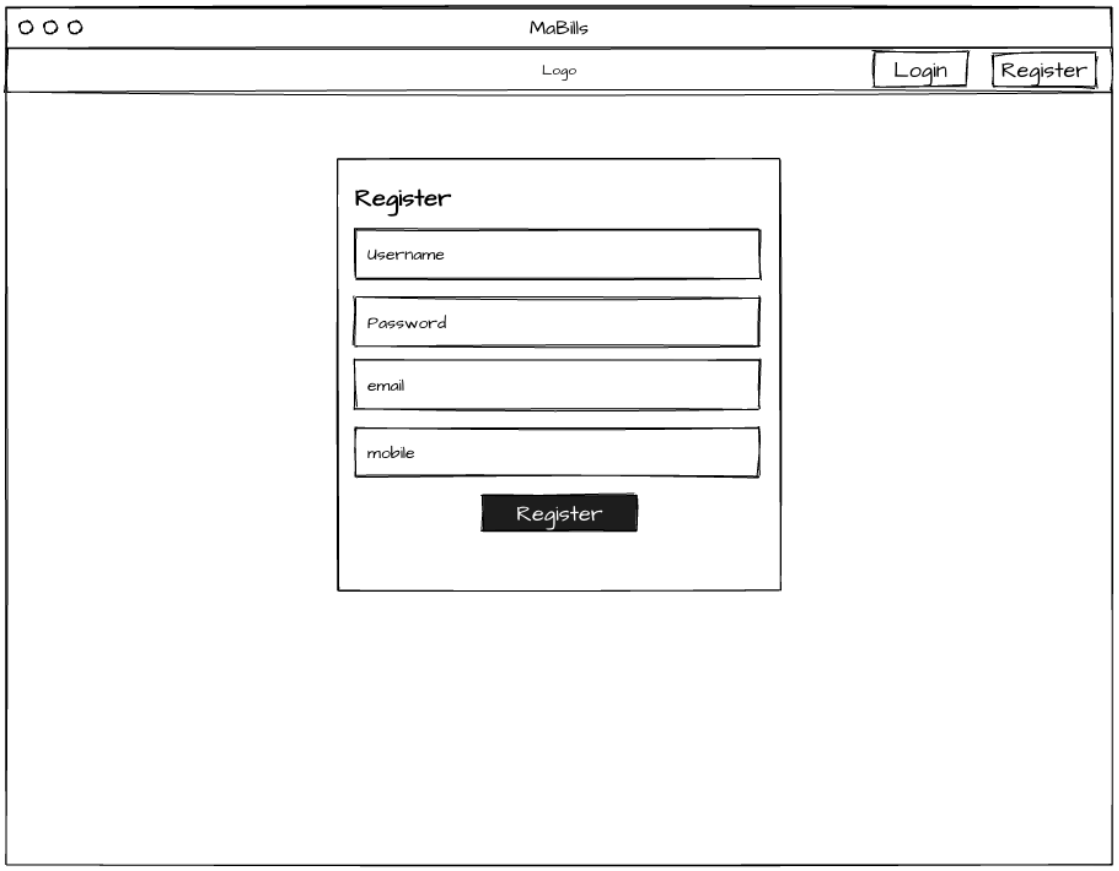
4. Prototipo interfaz de usuario

A continuación se presentan los prototipos de interfaz de usuario. Estos prototipos son simplemente guías para el desarrollo, no son las versiones finales, por lo cual, la aplicación real tiene algunas diferencias al respecto.

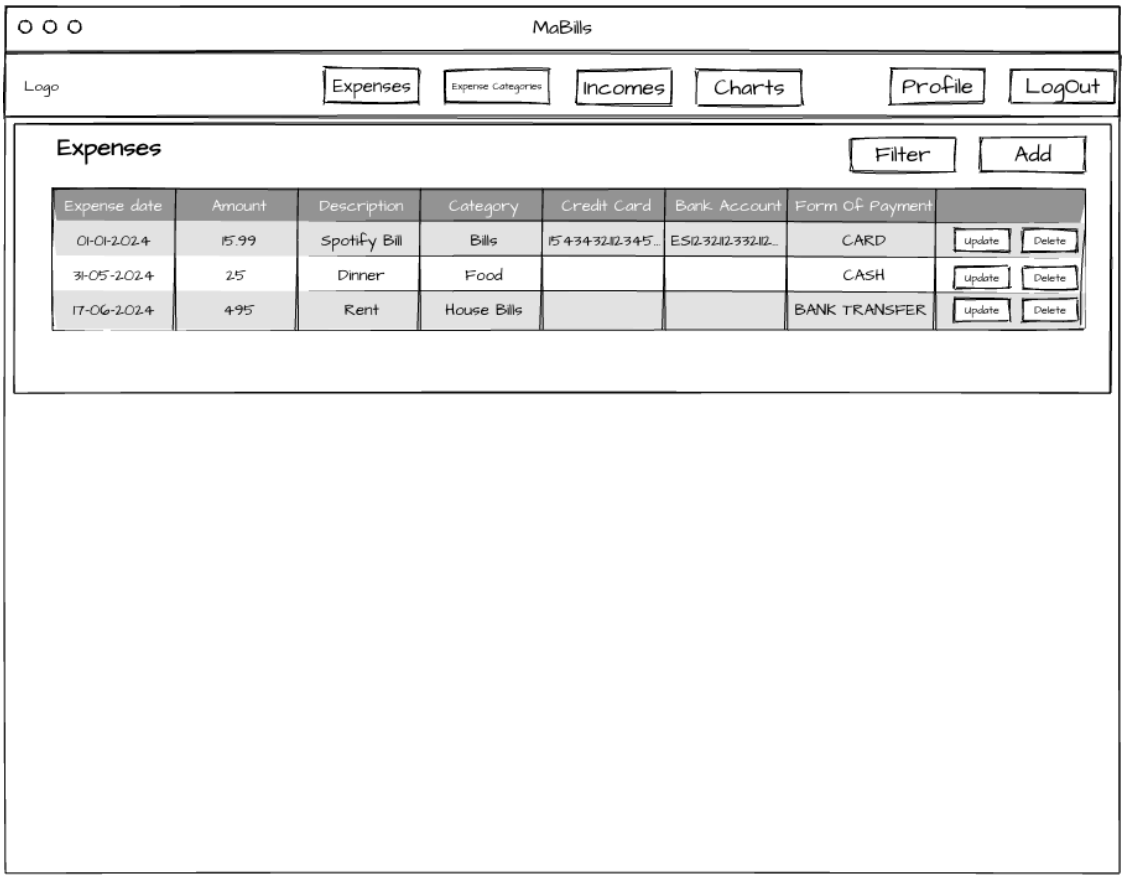
Login

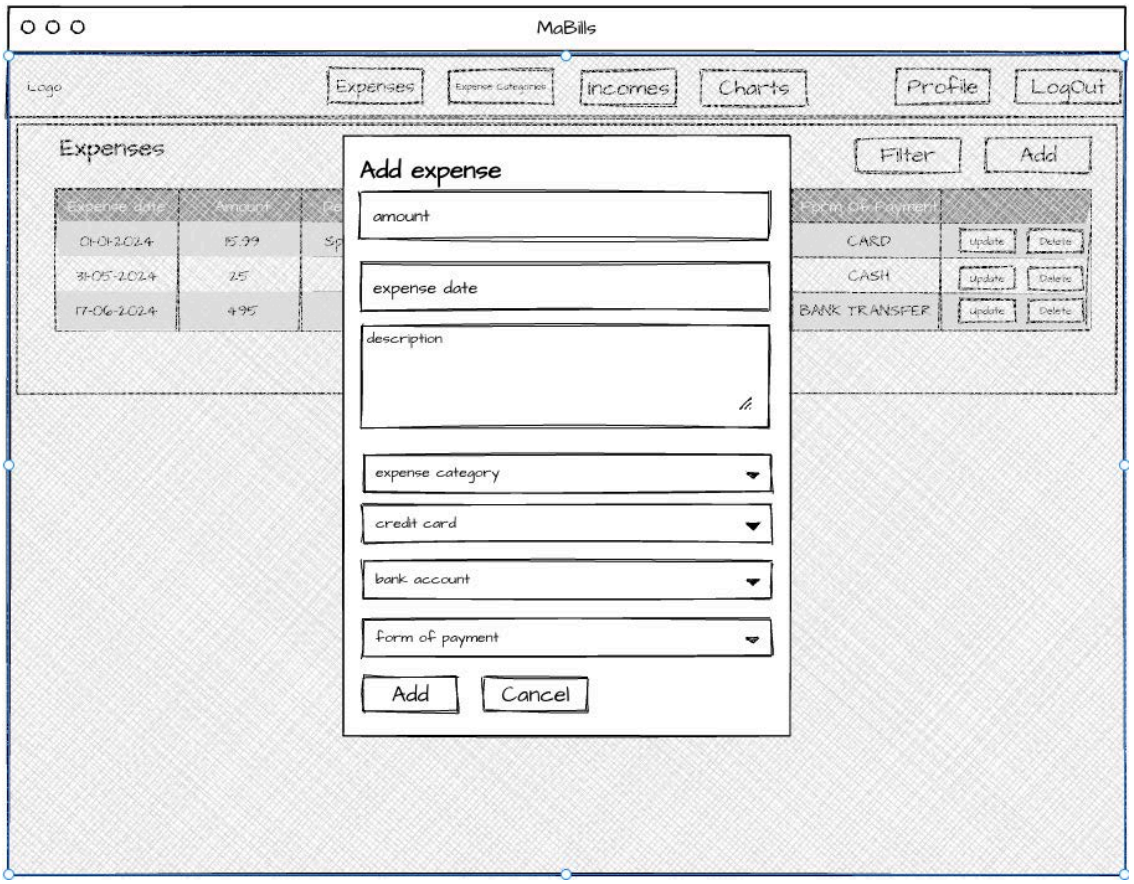


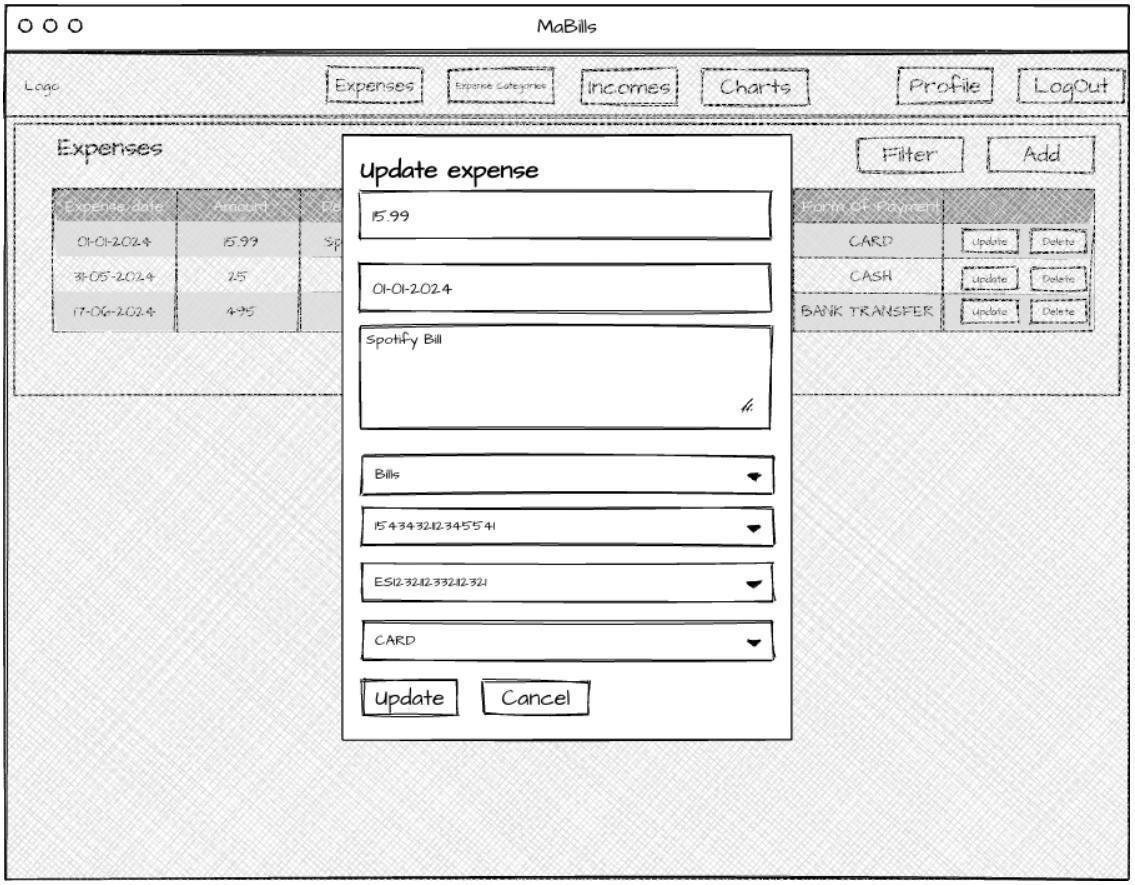
Register



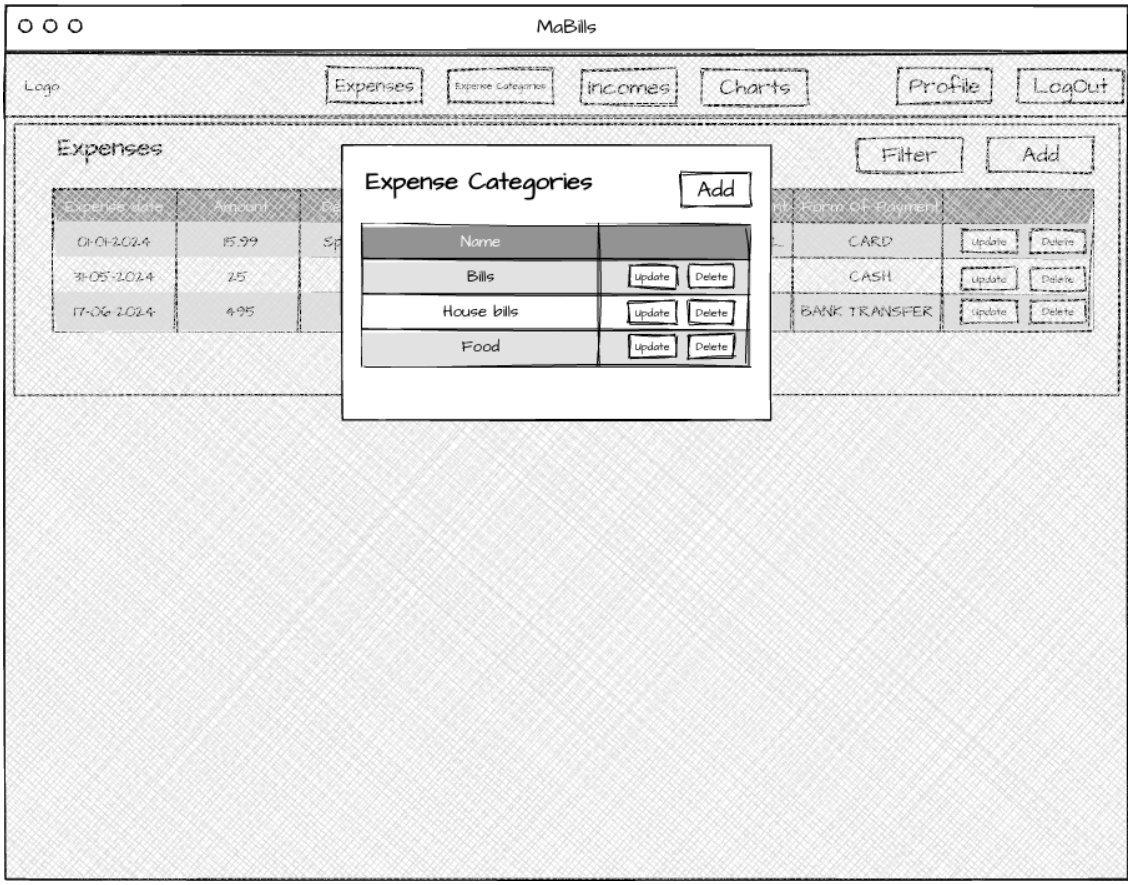
Expenses

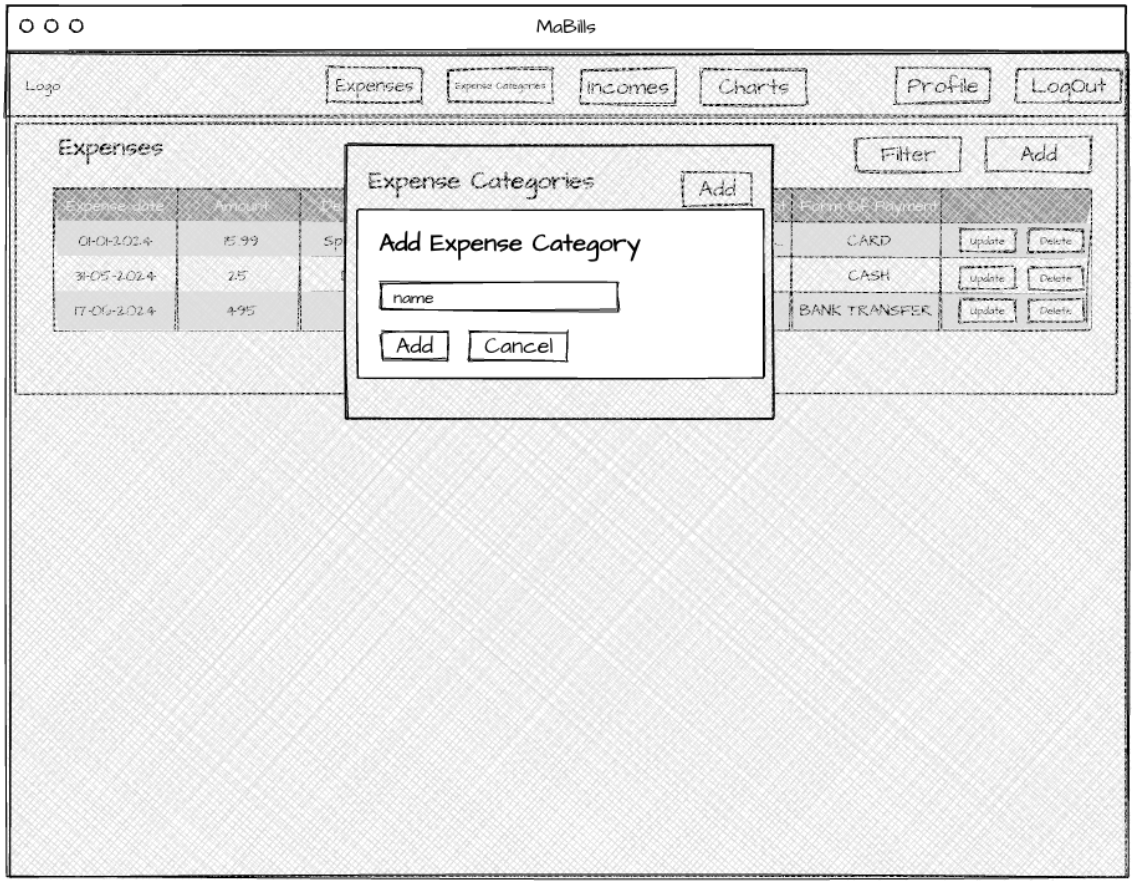


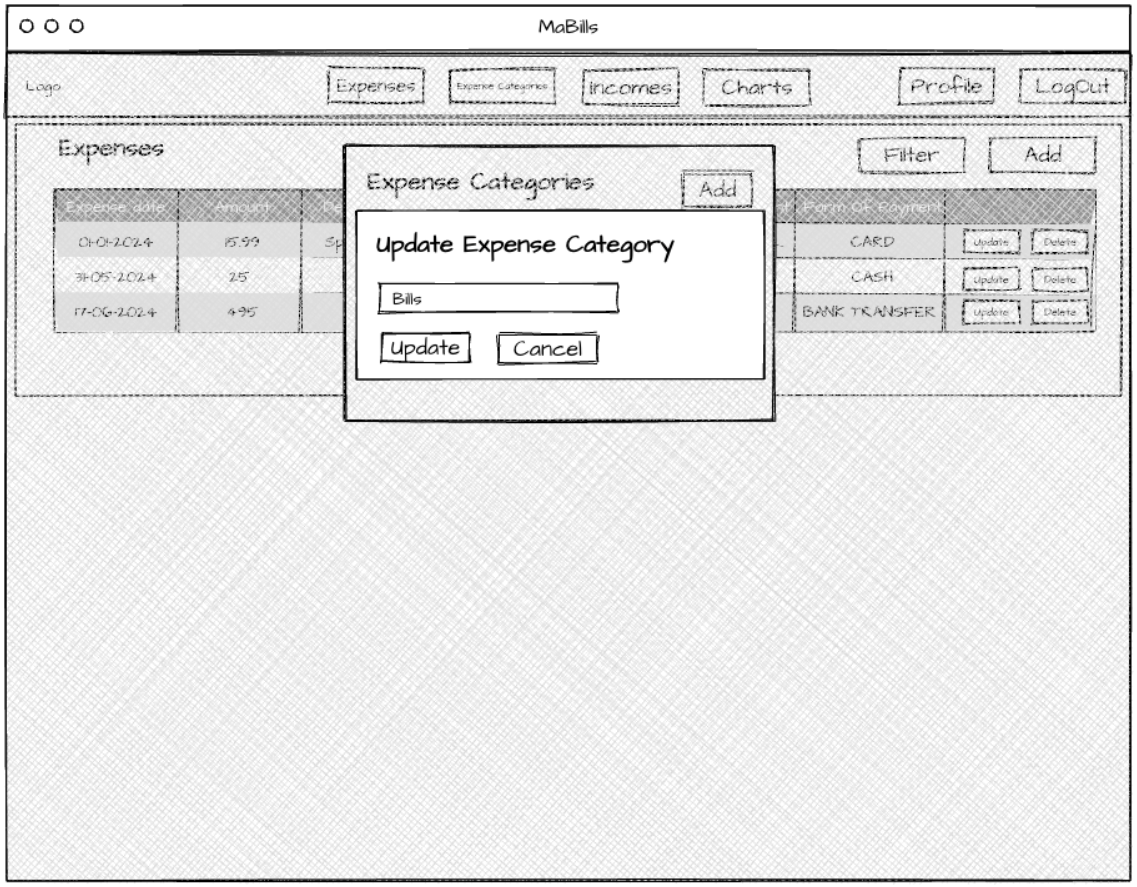

MaBills
Logo
Expenses
Expense Categories
Incomes
Charts
Profile
LogOut
Expenses
Filter
Add
Add expense
amount
expense date
description
expense category
credit card
bank account
form of payment
Add
Cancel
CARD
CASH
BANK TRANSFER
Update
Delete


MaBills
Logo
Expenses
Expense Categories
Incomes
Charts
Profile
LogOut
Expenses
Filter
Add
Update expense
15.99
01-01-2024
Spotify Bill
Bills
ES12-3212-3312-3212-3212-321
CARD
Update
Cancel
CARD
CASH
BANK TRANSFER
Update
Delete

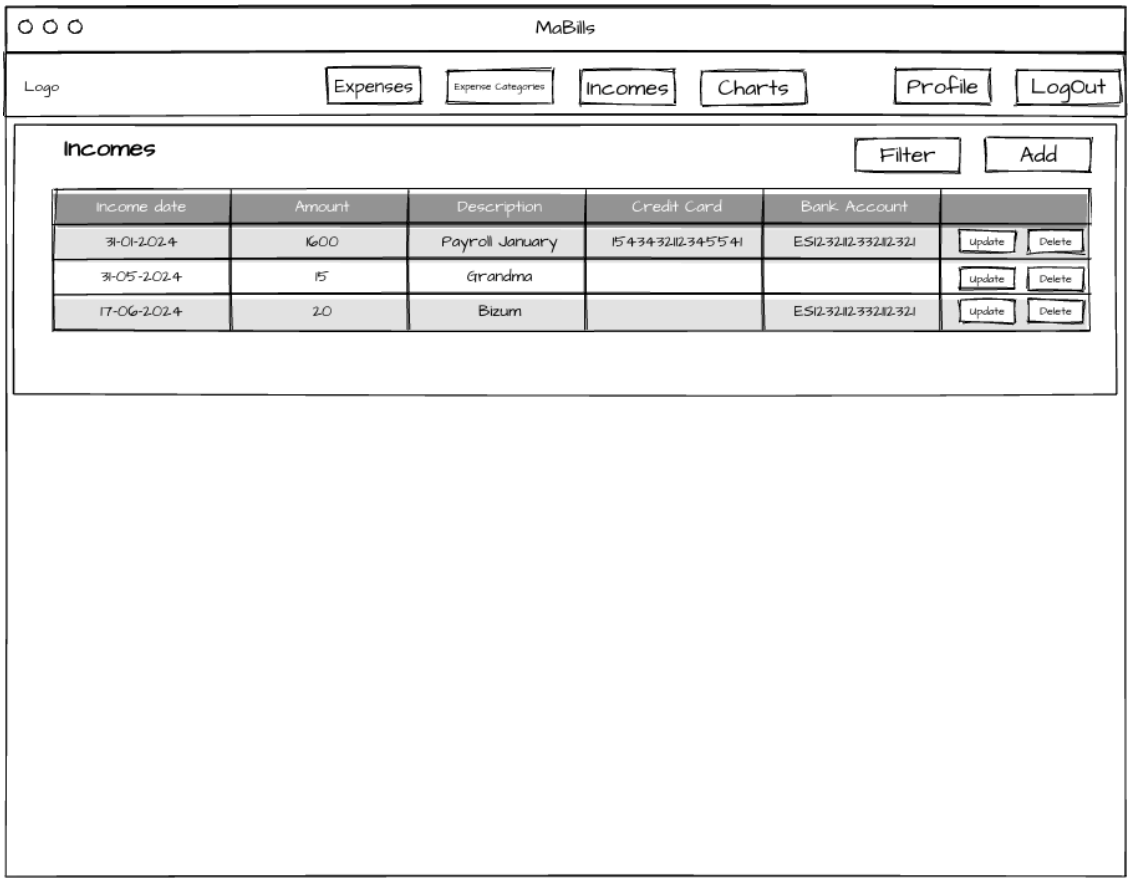
Expense Categories

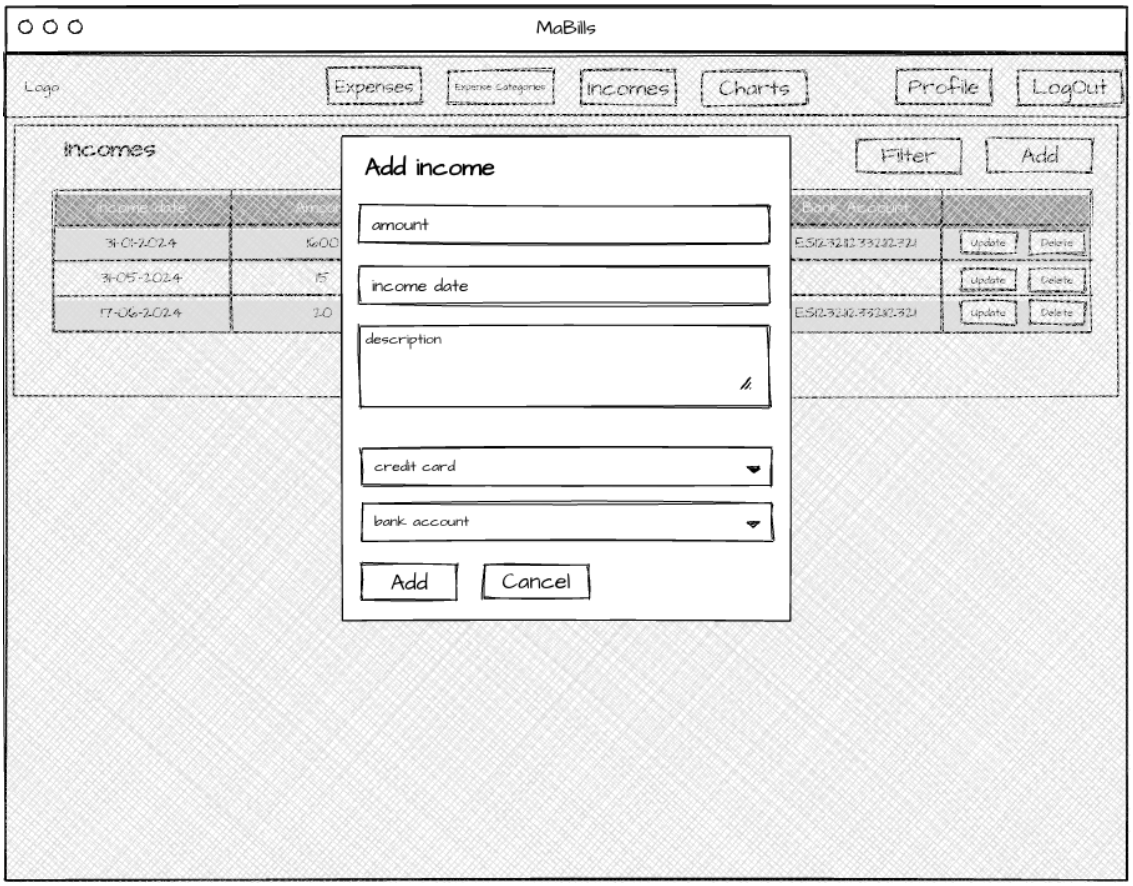


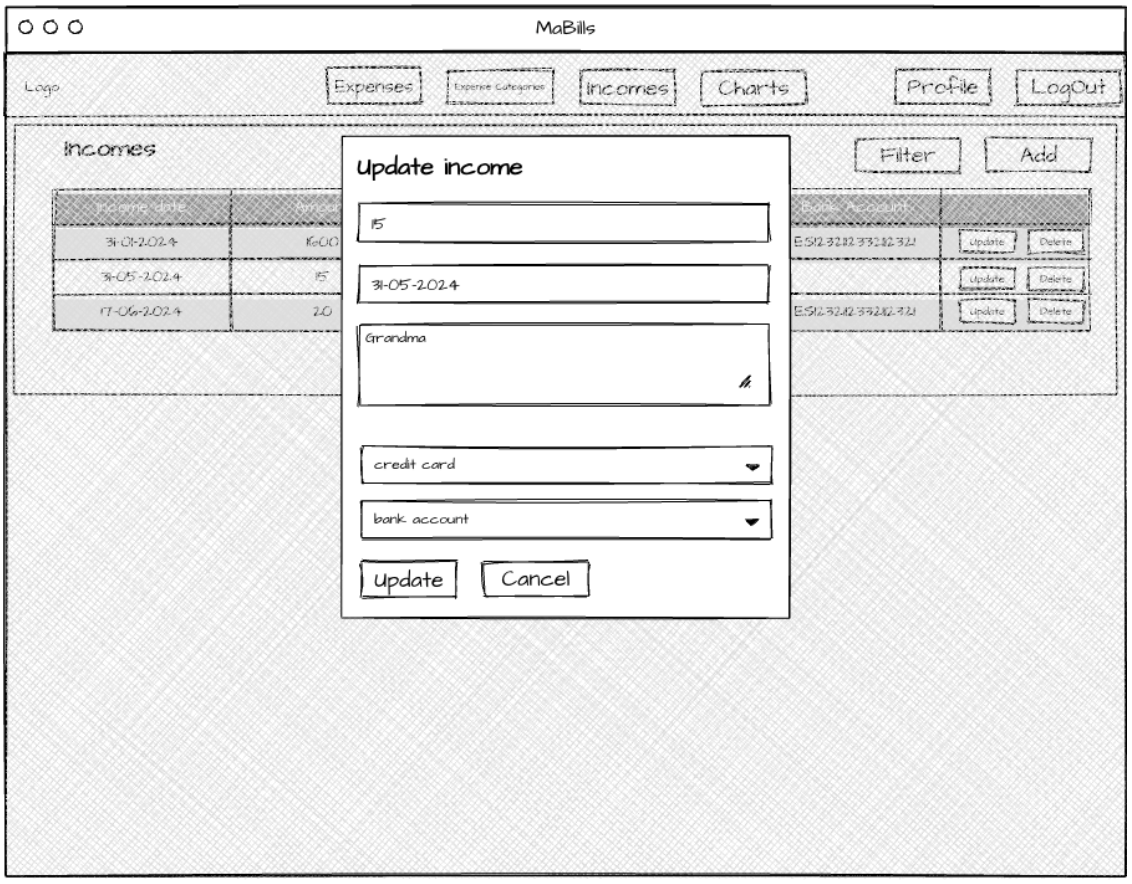



MaBills
Logo
Expenses
Expense Categories
Incomes
Charts
Profile
LogOut
Expenses
Filter
Add
Expense Categories
Add
Update Expense Category
Bills
Update
Cancel
Expense date
Amount
Form Of Payment
01-01-2024
15.99
CARD
31-05-2024
25
CASH
17-06-2024
495
BANK TRANSFER
Update
Delete

Incomes

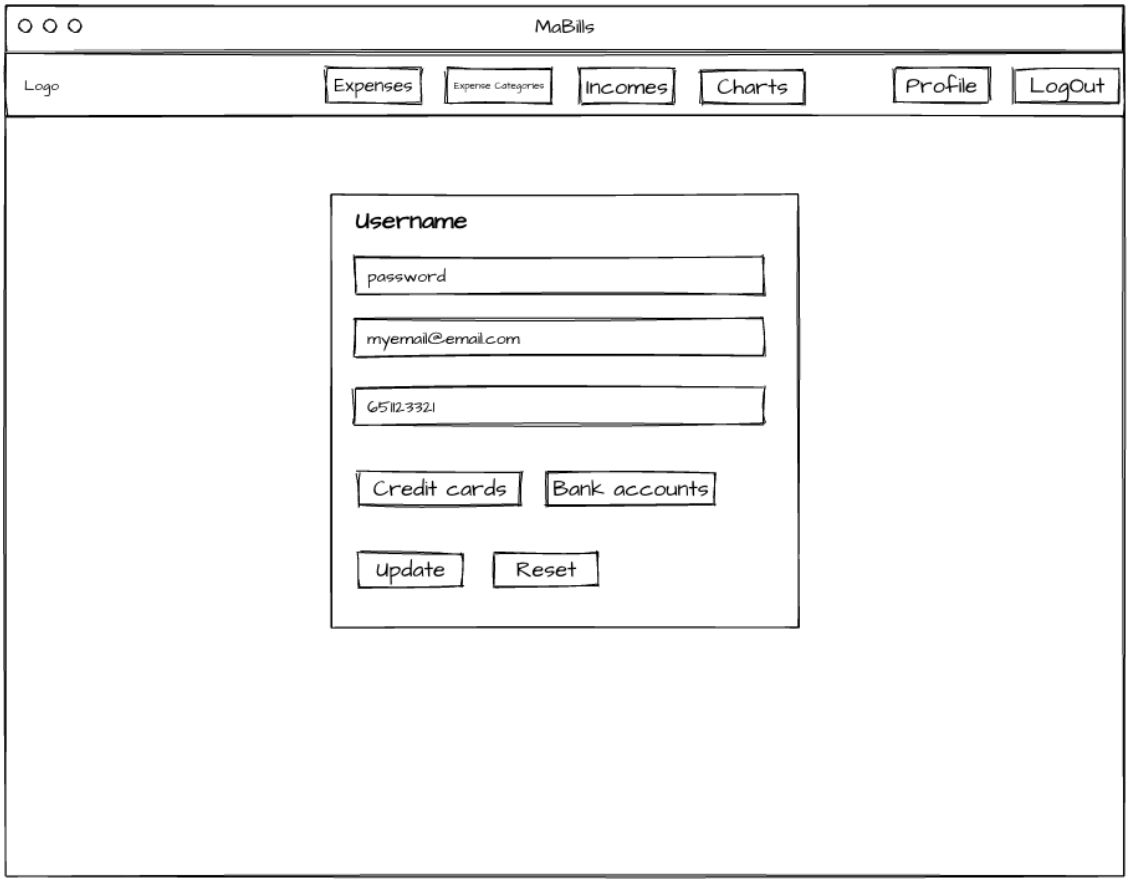

MaBills
Logo
Expenses
Expense Categories
Incomes
Charts
Profile
LogOut
Incomes
Filter
Add
Income date
Amount
Description
Credit Card
Bank Account
31-01-2024
1600
Payroll January
1543432112345541
ES1232112332112321
31-05-2024
15
Grandma
17-06-2024
20
Bizum
ES1232112332112321
Update
Delete


MaBills
Logo
Expenses
Expense Categories
Incomes
Charts
Profile
LogOut
Incomes
Filter
Add
Add income
amount
income date
description
credit card
bank account
Add
Cancel

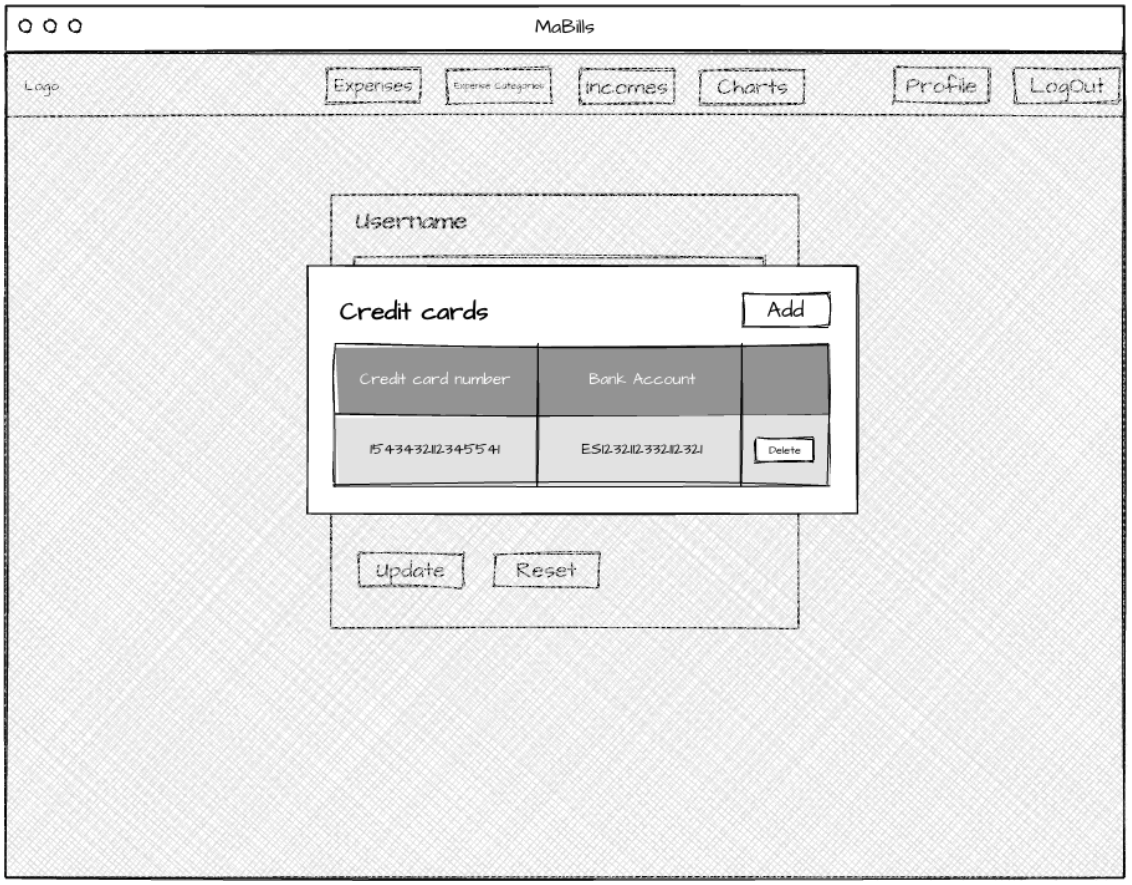

MaBills
Logo
Expenses
Expense Categories
Incomes
Charts
Profile
LogOut
Incomes
Filter
Add
Update income
15
31-05-2024
Grandma
credit card
bank account
Update
Cancel

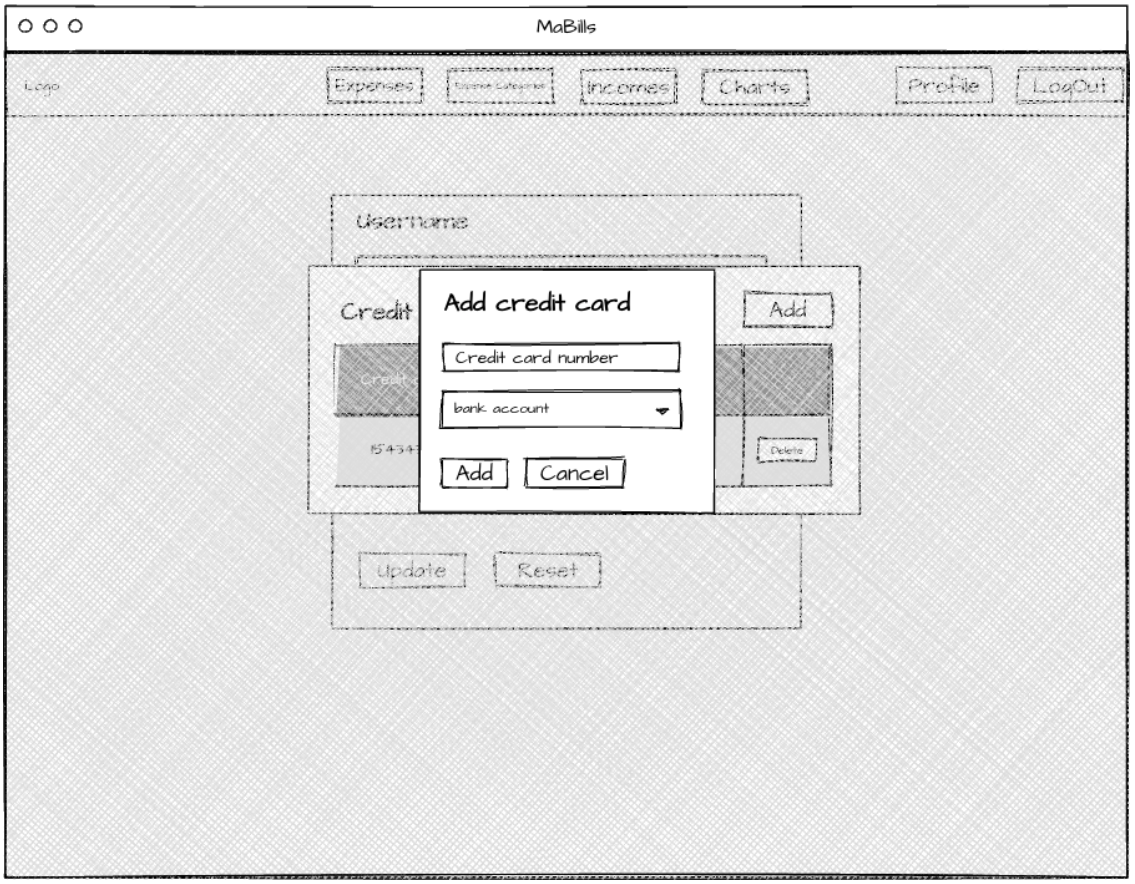
Charts

User Profile

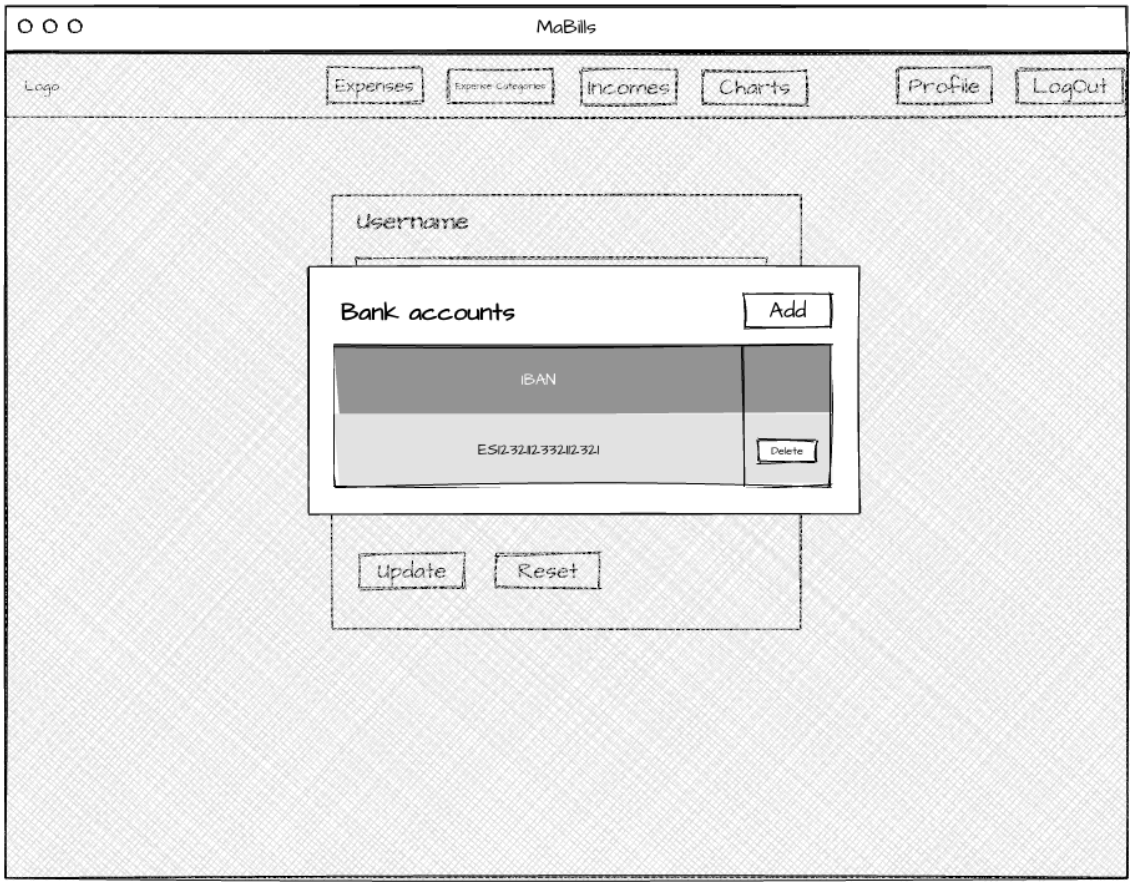


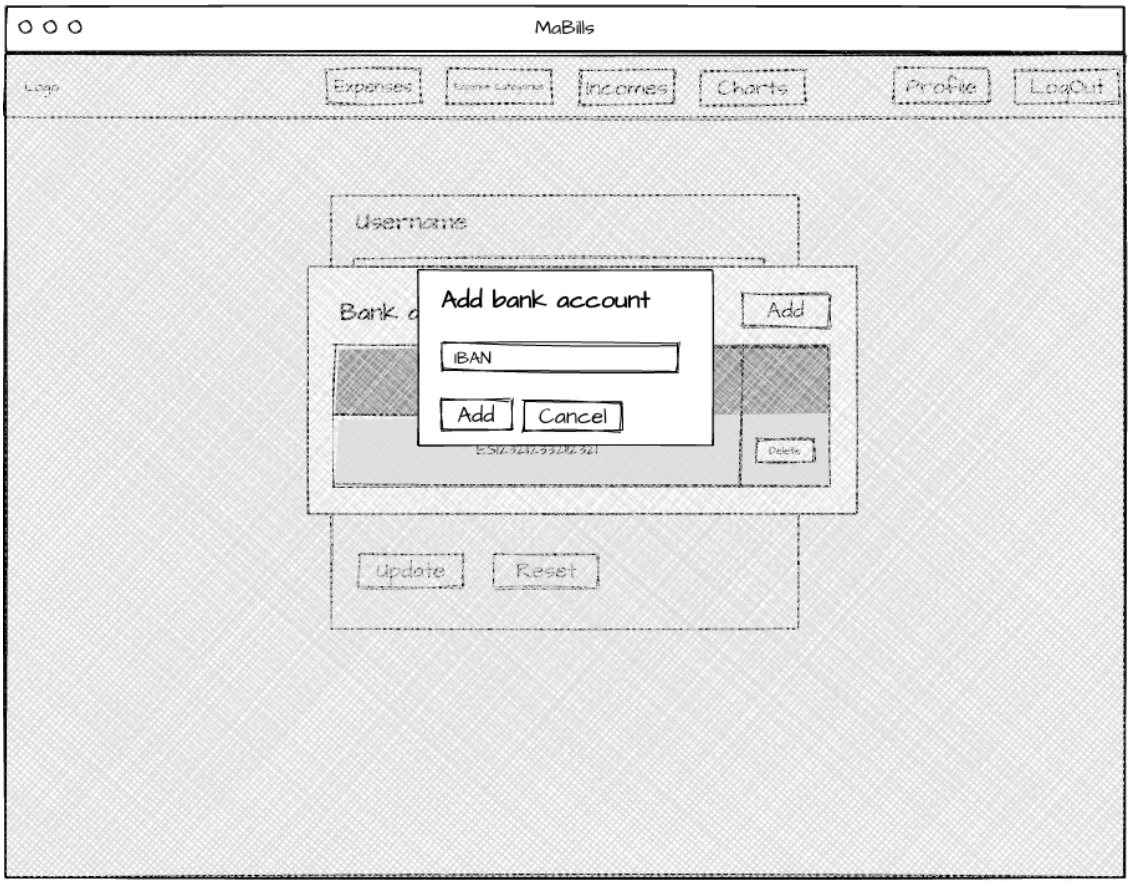
User Profile - credit cards



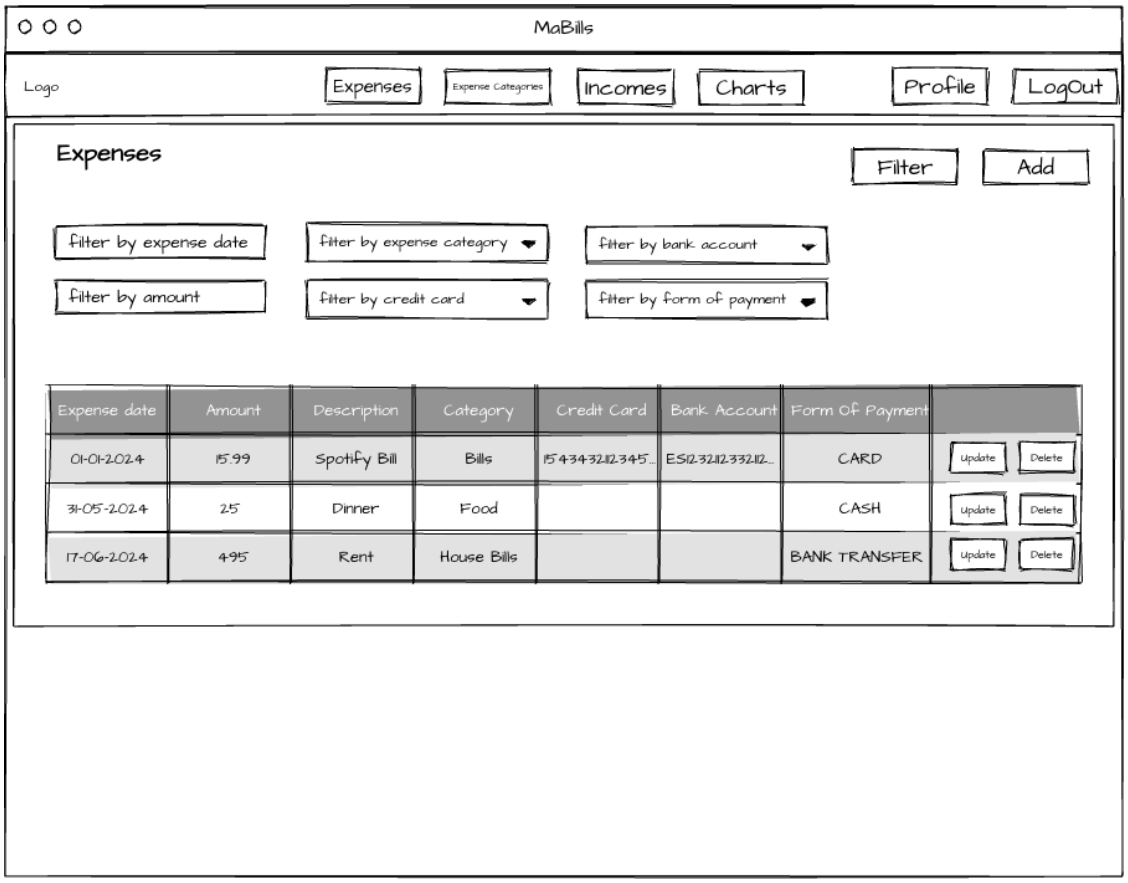


User Profile - bank accounts





Apply Filters



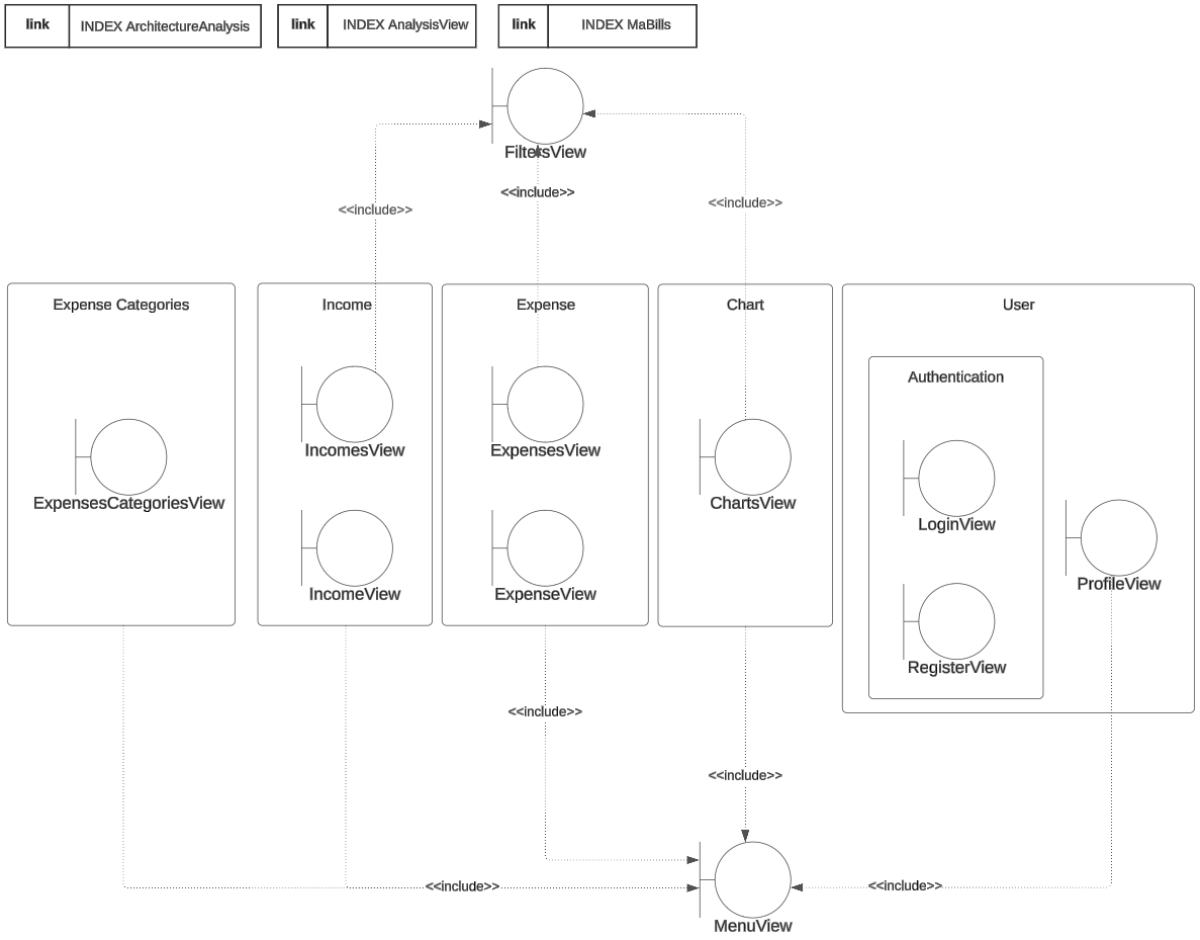
Con estos prototipos, aunque no representan la versión final de la aplicación, se puede apreciar que la interfaz de usuario es bastante simple, esto también es debido a la falta de tiempo para la realización del proyecto, pero en un futuro, se pretende darle un *"lavado de cara"* a la aplicación y mejorar la interfaz de usuario, junto con otras propuestas que se comentarán en el **capítulo 8**.

CAPÍTULO 3. ANÁLISIS

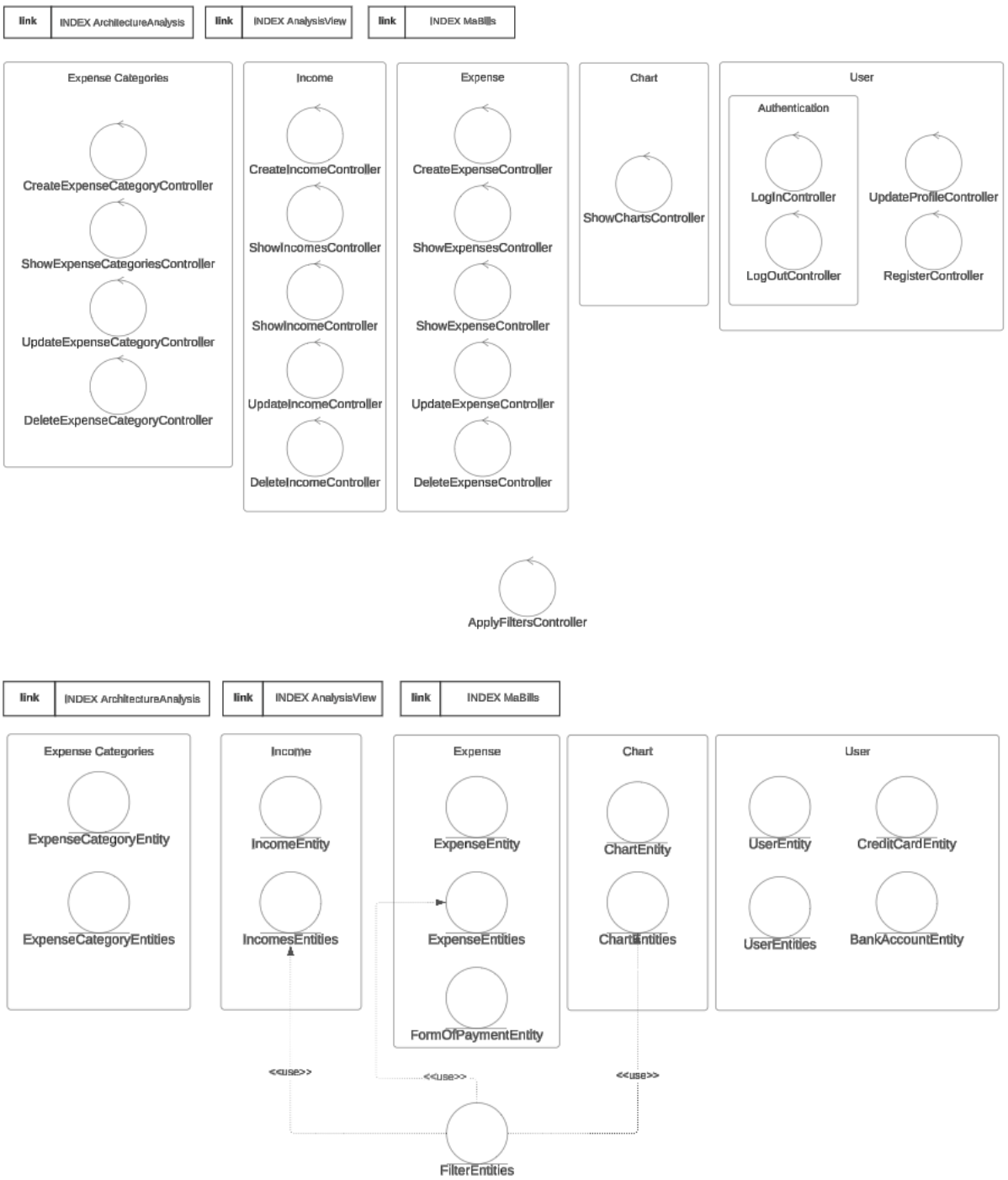
Con el análisis se pretende refinar más aún los requisitos encontrados en el capítulo anterior y así, lograr una mejor comprensión de dichos requisitos.
Durante este capítulo se estructuraran los casos de uso, describiendolos usando el lenguaje del desarrollador y acortando el modelo de diseño para dar entrada al próximo paso, el diseño.

1. Análisis de la arquitectura

En este apartado se pretende identificar los componentes (clases, paquetes o requisitos especiales comunes) que forman el sistema, Se han agrupado los componentes por tipo (gastos, ingresos…).



En las vistas, se puede apreciar que algunos componentes incluyen a otros, esto es debido a que algunas vistas se encuentran dentro de otras.



En este último diagrama, se puede observar que *FilterEntities usa IncomeEntities, ExpenseEntities y ChartEntities,* esto es debido a que *FilterEntities* se apoya en las otras entidades para poder realizar su función, que es el filtrado de datos.

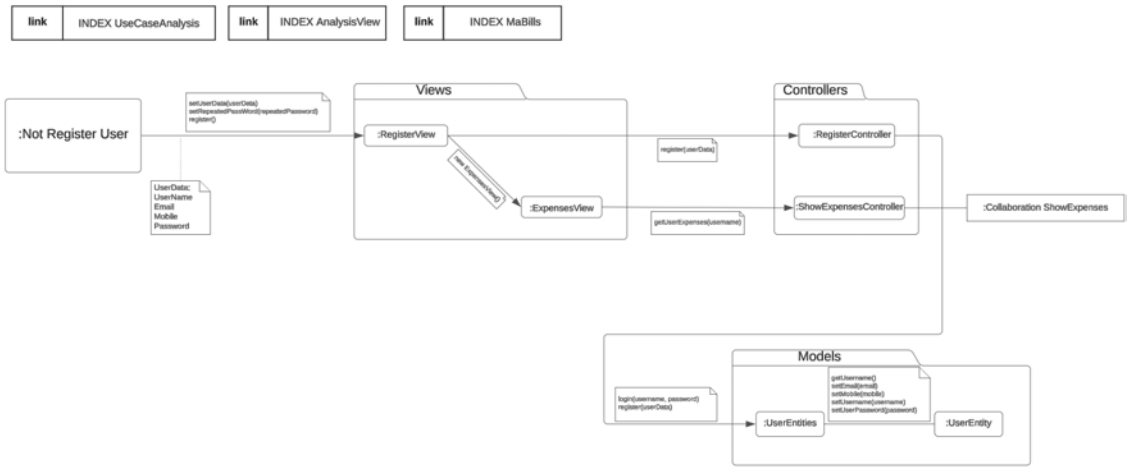
2. Análisis de casos de uso

Con este análisis se pretende identificar clases de control, entidad y vistas necesarias para realizar el caso de uso, representandolos en diagramas de colaboración para describir cómo interactúan entre ellos.

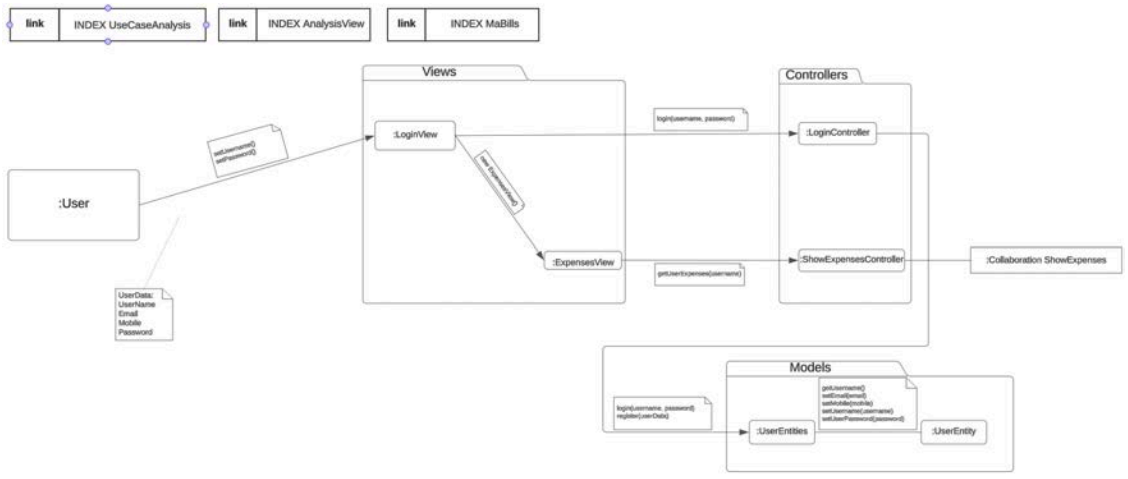
De forma genérica, se puede apreciar que todos los casos de uso empiezan con la interacción del actor con una vista, luego la vista interactúa con un controlador y este, si fuese necesario, interactúa con un modelo.

En esta sección, hay que tener en cuenta que para abreviar, *User* se refiere a los usuarios registrados, los usuarios no registrados sólamente pueden realizar el caso de uso *Register.*

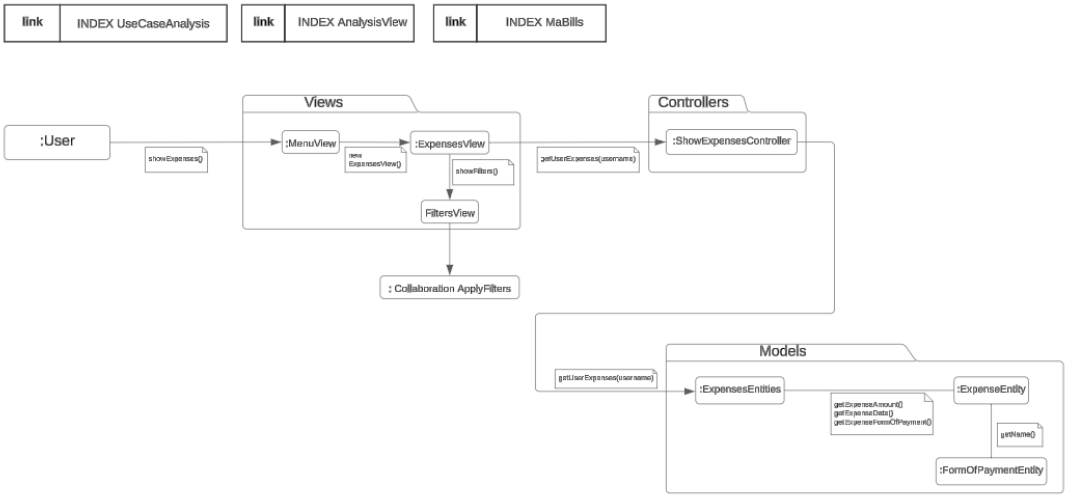
Register



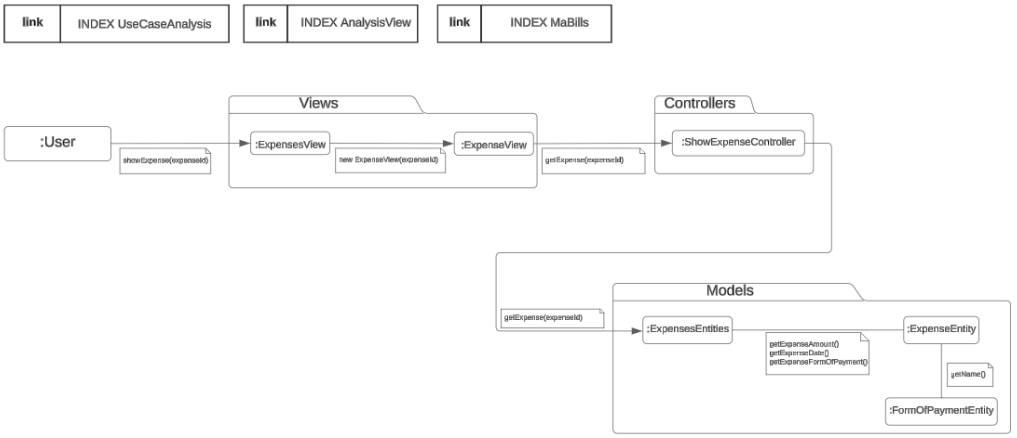
Login



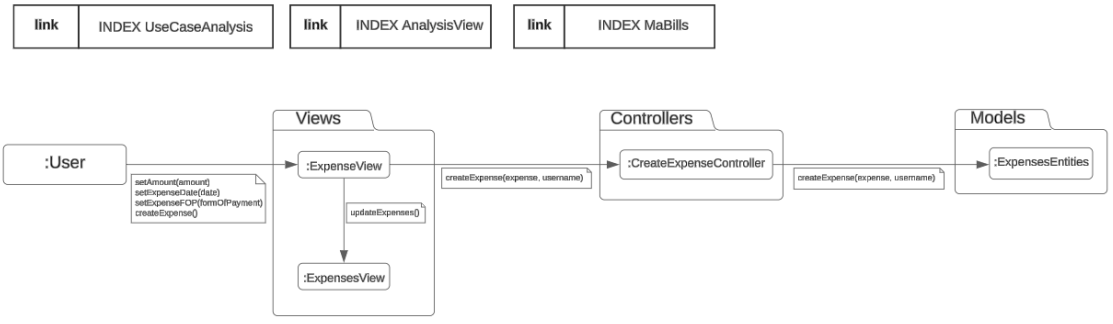
Show expenses



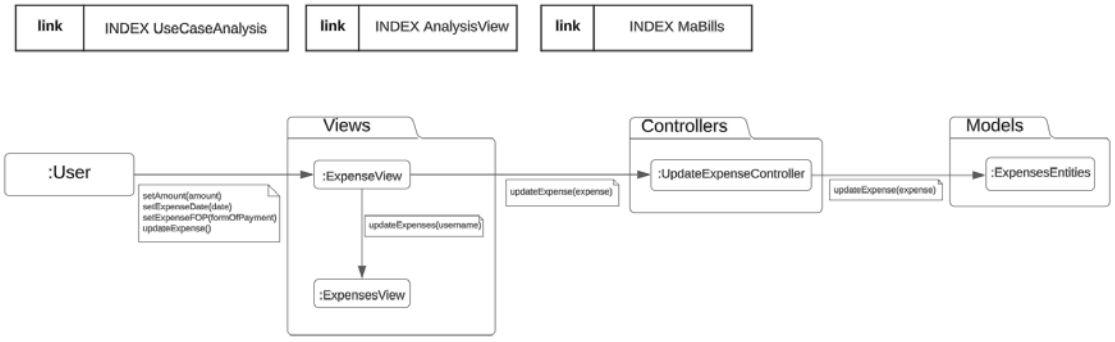
Show expense



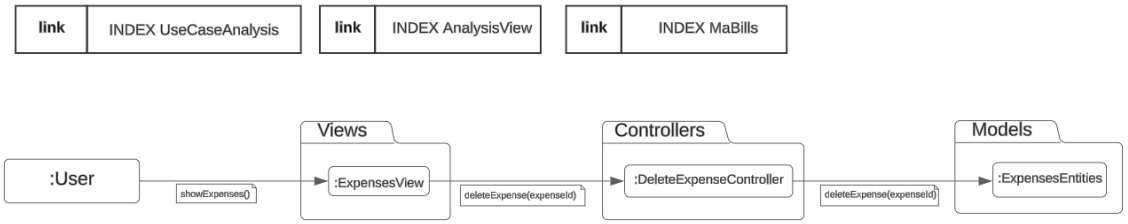
Create expense



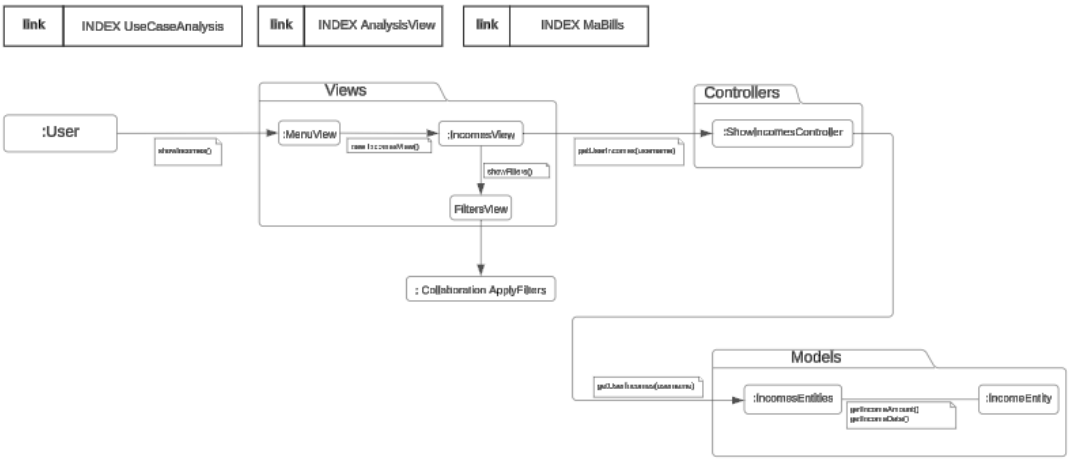
Update expense



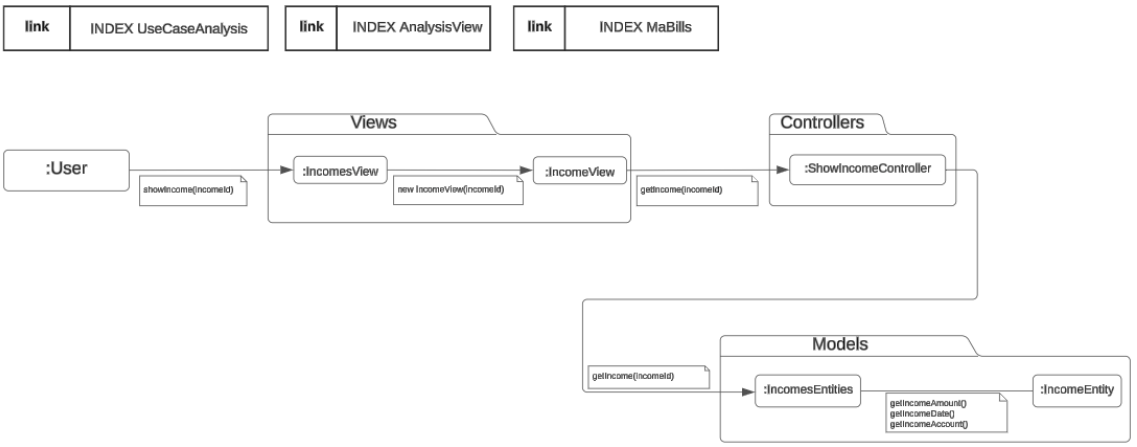
Delete expense



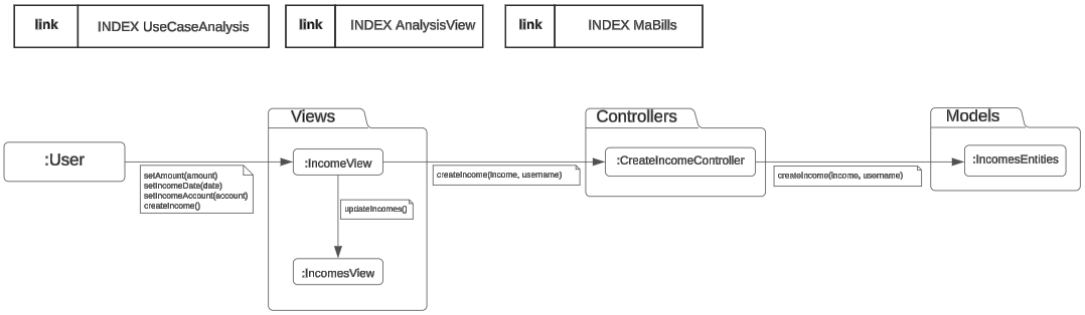
Show incomes



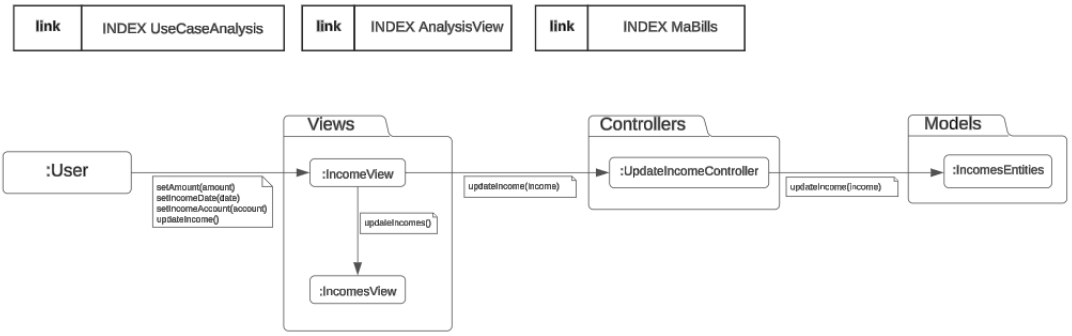
Show income



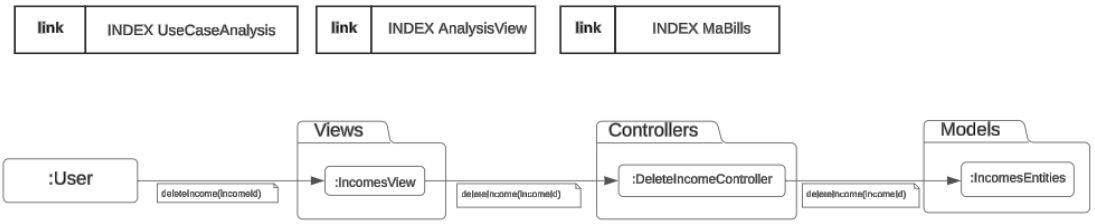
Create income



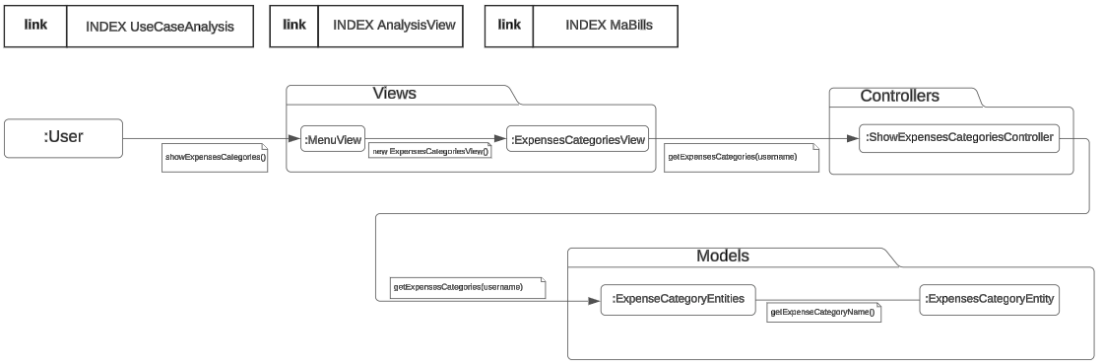
Update income



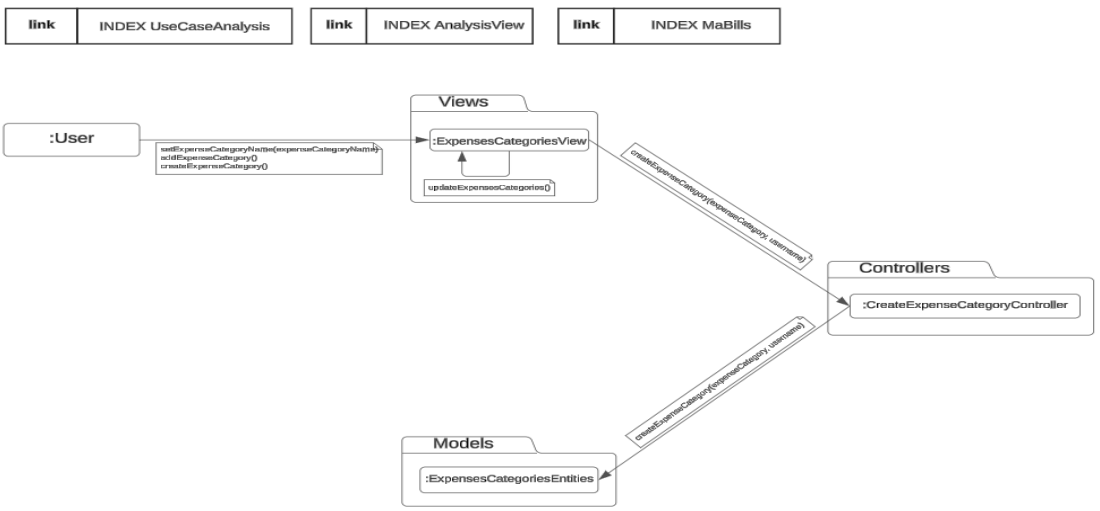
Delete income



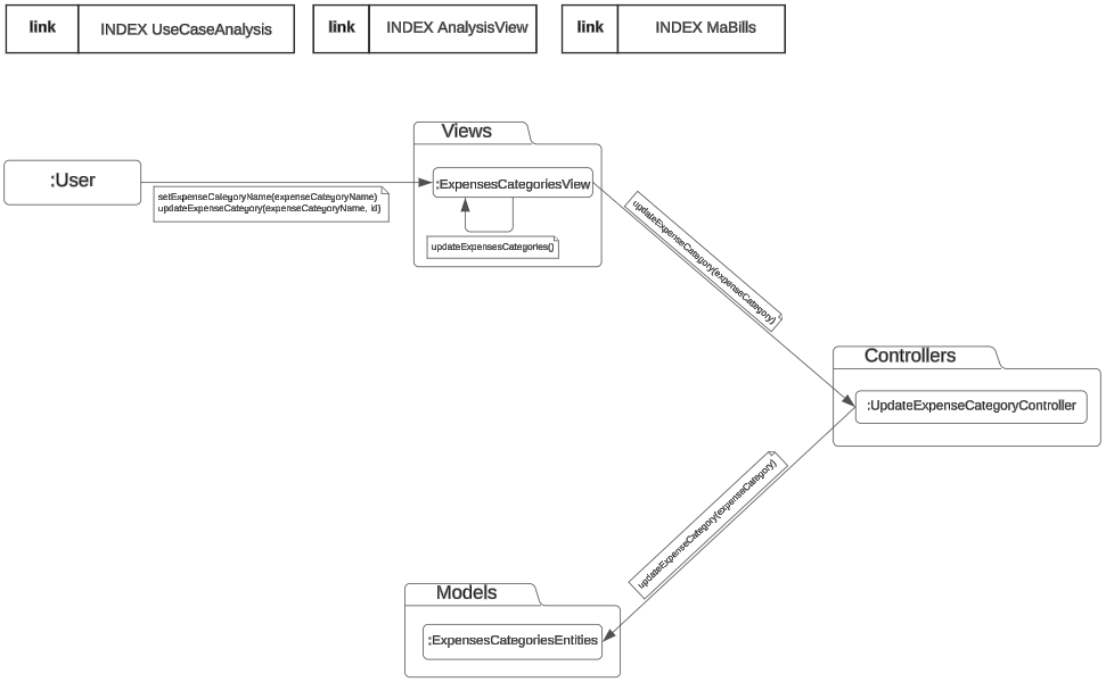
Show expenses categories



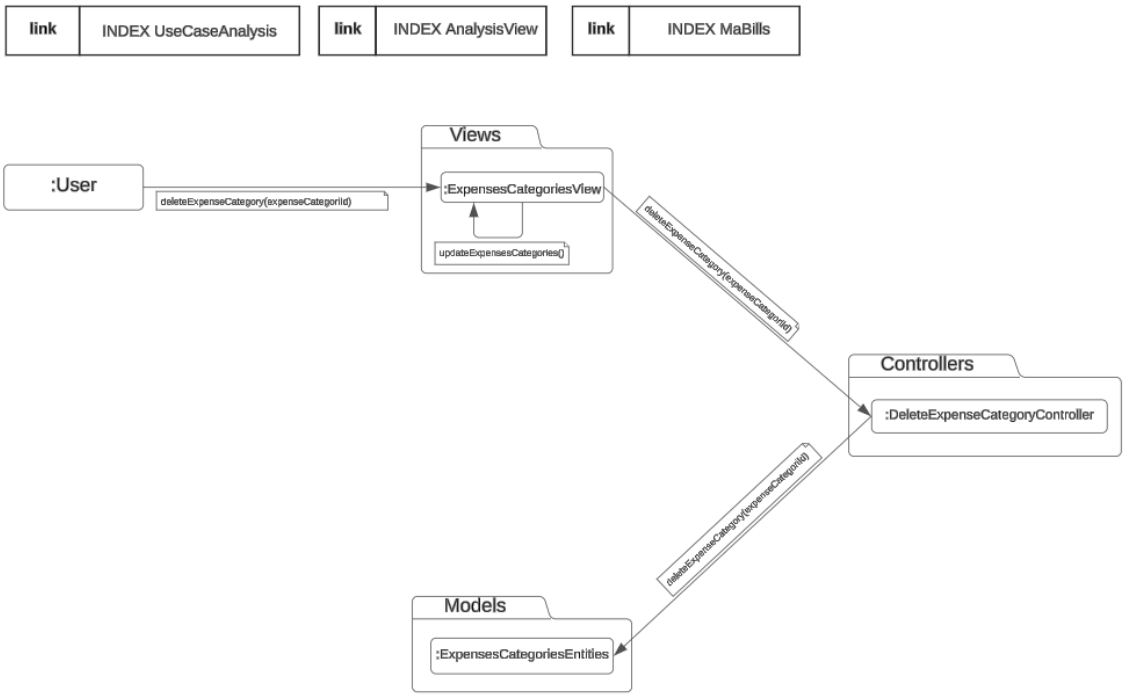
Create expense category



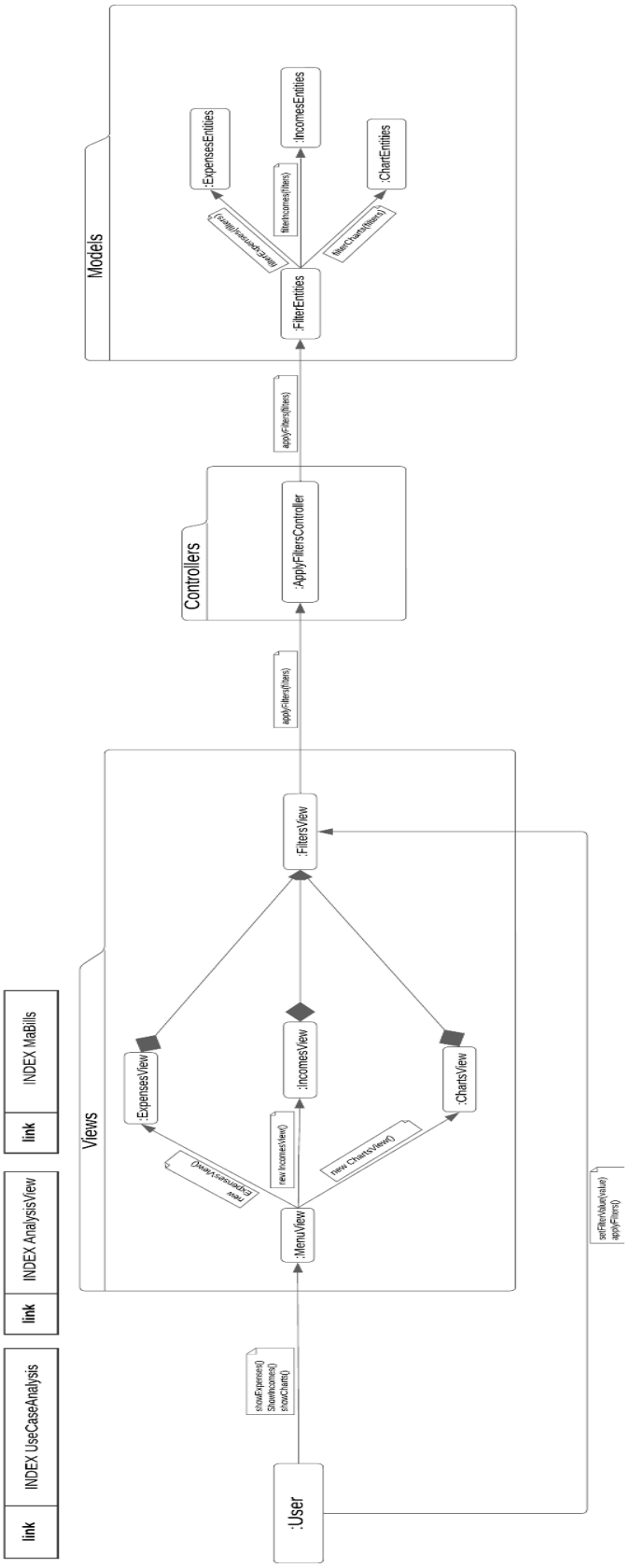
Update expense category



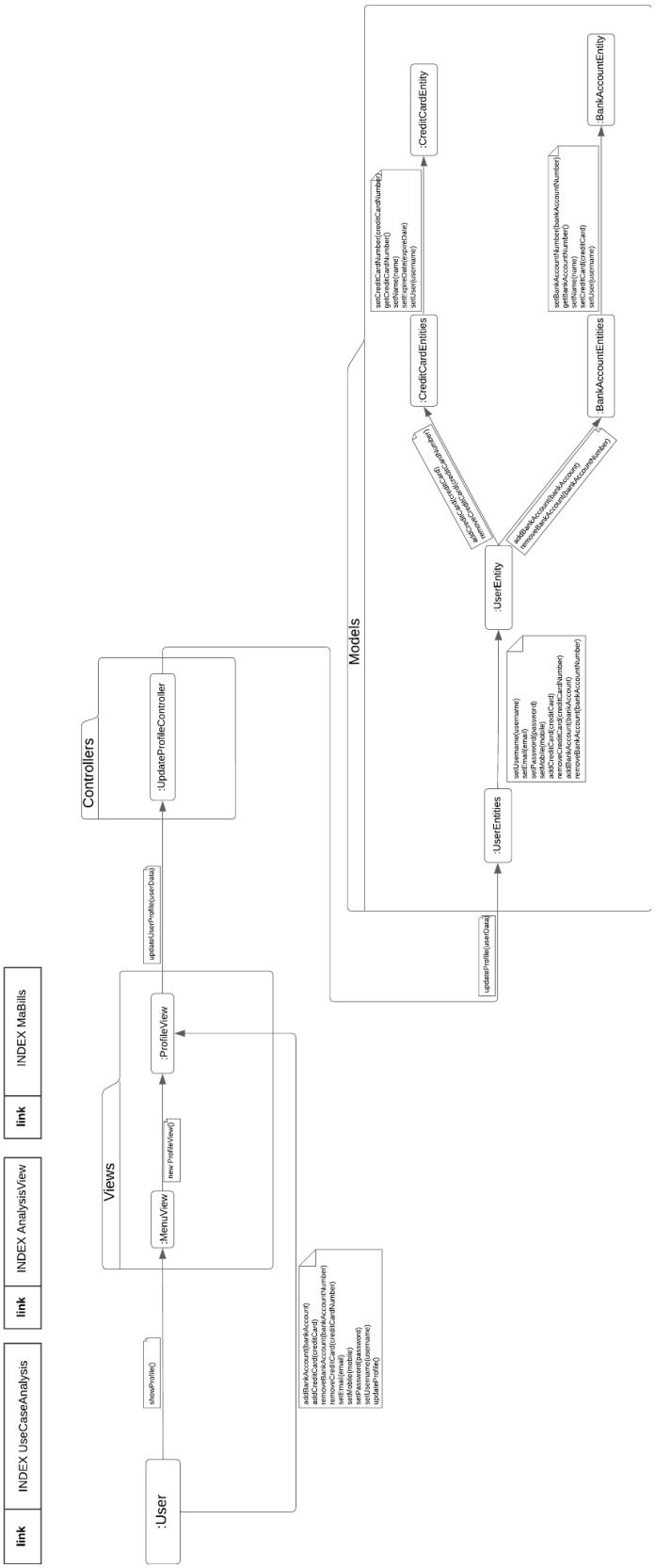
Delete expense category



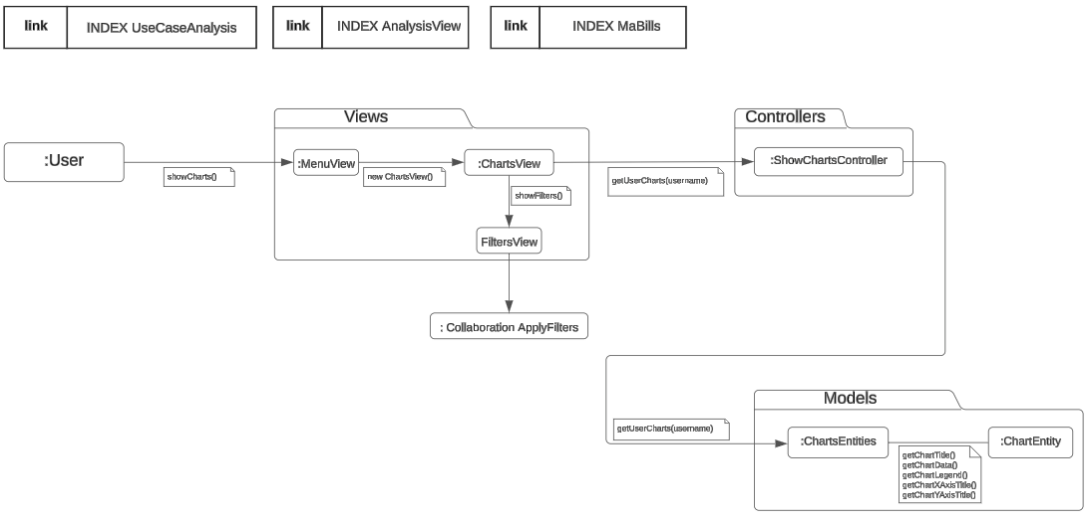
Apply filters



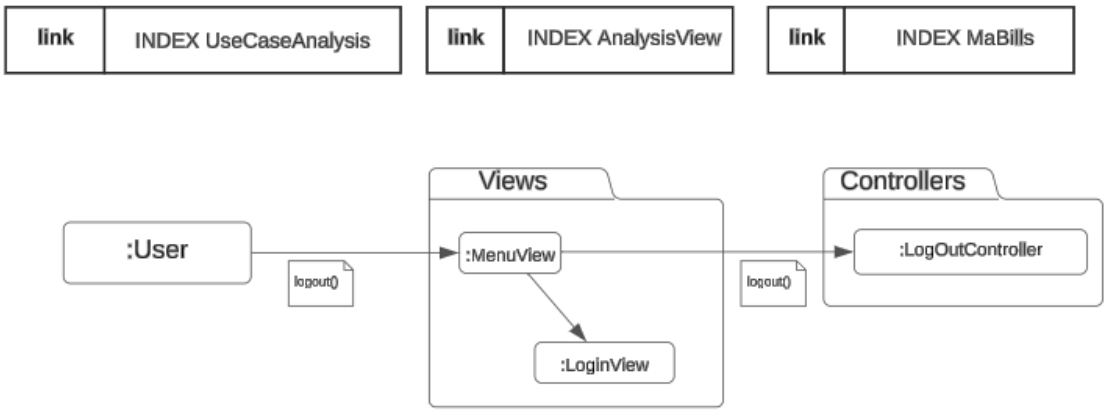
Update profile



Show charts



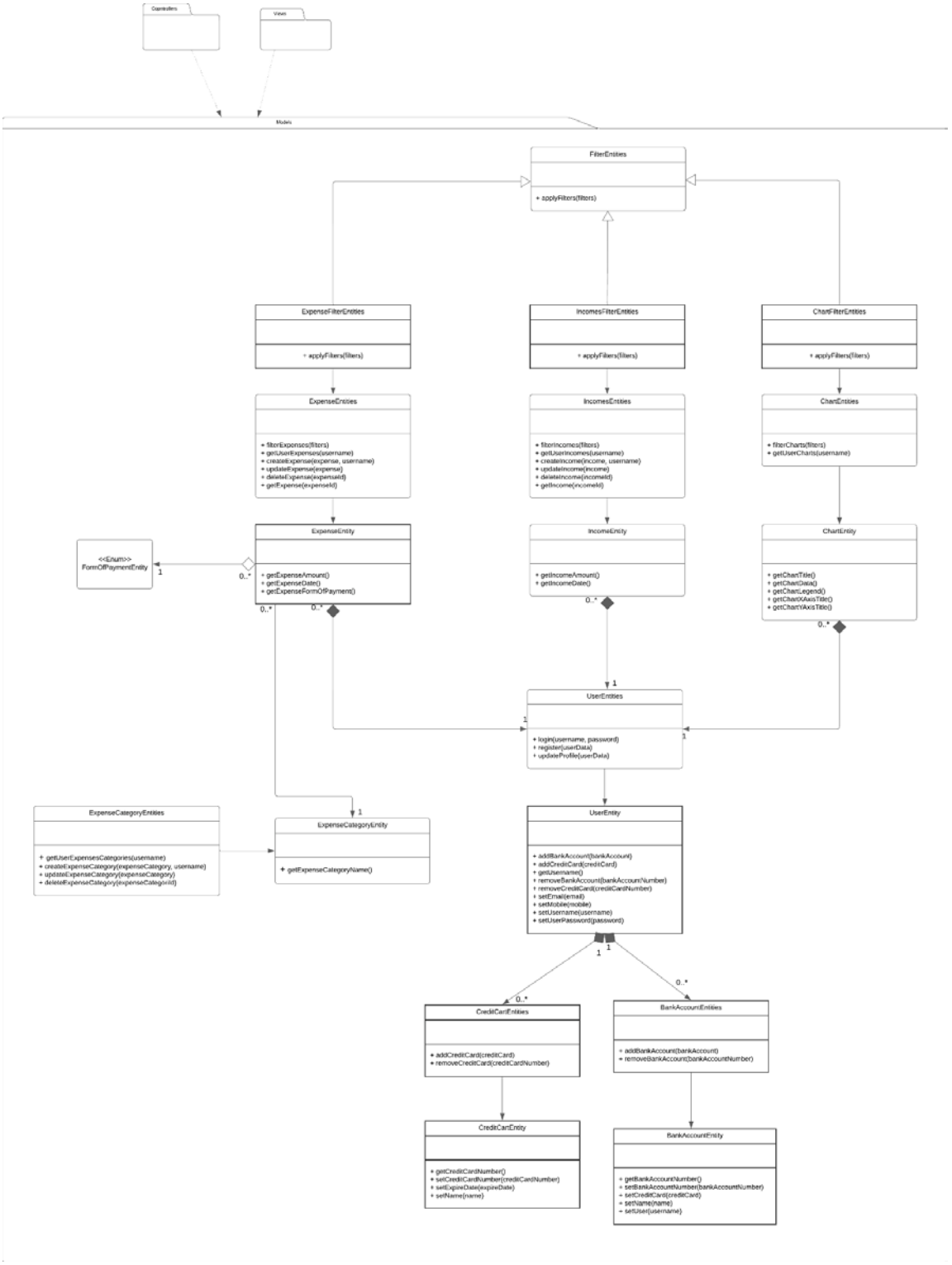
Log out

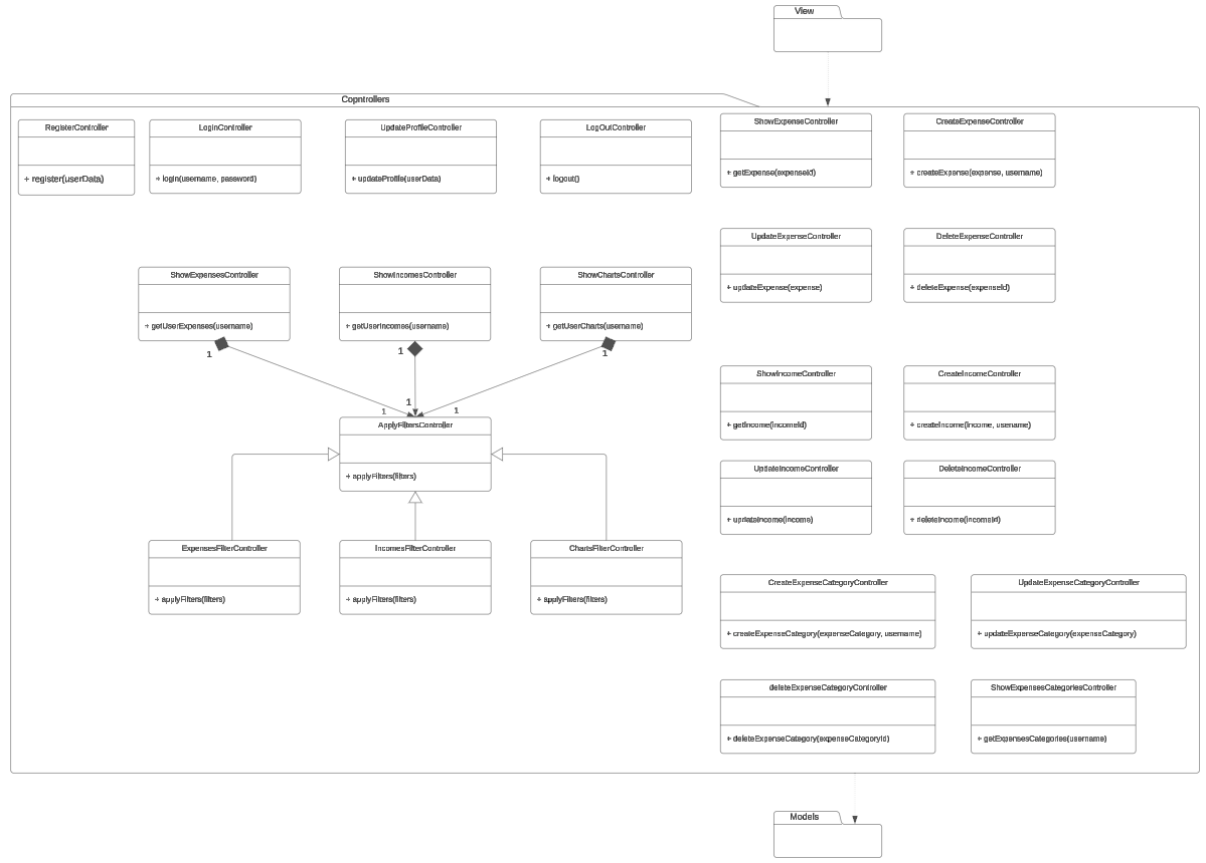


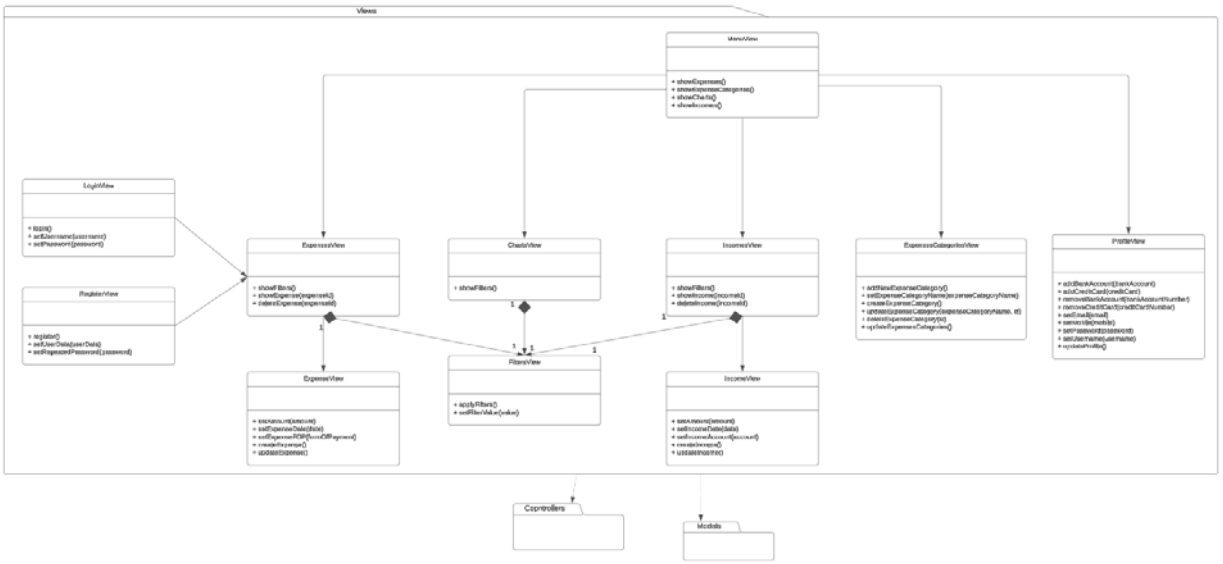
3. Análisis de clases y de paquetes

Se ha decidido realizar estas dos actividades en una para evitar representar repetitivamente los mismos diagramas.

Con esta sección se pretende identificar las responsabilidades que una clase puede tener durante la realización de los casos de uso, además, se identificarán las relaciones entre las clases. Como parte de análisis de paquetes, se definirán y se mantendrán las dependencias entre los paquetes y se asegurará que las clases pertenecen al paquete correspondiente.




View
Copntrollers
RegisterController
+ register(userData)
LoginController
+ login(username, password)
UpdateProfileController
+ updateProfile(userData)
LogOutController
+ logout()
ShowExpenseController
+ getExpense(expenseId)
CreateExpenseController
+ createExpense(expense, username)
UpdateExpenseController
+ updateExpense(expense)
DeleteExpenseController
+ deleteExpense(expenseId)
ShowExpensesController
+ getUserExpenses(username)
ShowIncomesController
+ getUserIncomes(username)
ShowChartsController
+ getUserCharts(username)
ShowIncomeController
+ getIncome(incomeId)
CreateIncomeController
+ createIncome(income, username)
UpdateIncomeController
+ upretaIncome(income)
DeleteIncomeController
+ deleteIncome(incomeId)
ApplyFiltersController
+ applyFilters(filters)
ExpensesFilterController
+ applyFilters(filters)
IncomesFilterController
+ applyFilters(filters)
ChartsFilterController
+ applyFilters(filters)
CreateExpenseCategoryController
+ createExpenseCategory(expenseCategory, username)
UpdateExpenseCategoryController
+ updateExpenseCategory(expenssCategory)
deleteExpenseCategoryController
+ deleteExpenseCategory(expenseCategoryId)
ShowExpensesCategoriesController
+ getExpensesCategories(username)
Models


Views
LoginView
RegisterView
ExpensesView
ChartsView
IncomesView
ExpensesCategoriesView
PrinterView
FiltersView
ExpenseView
IncomeView
Copntrollers
Models

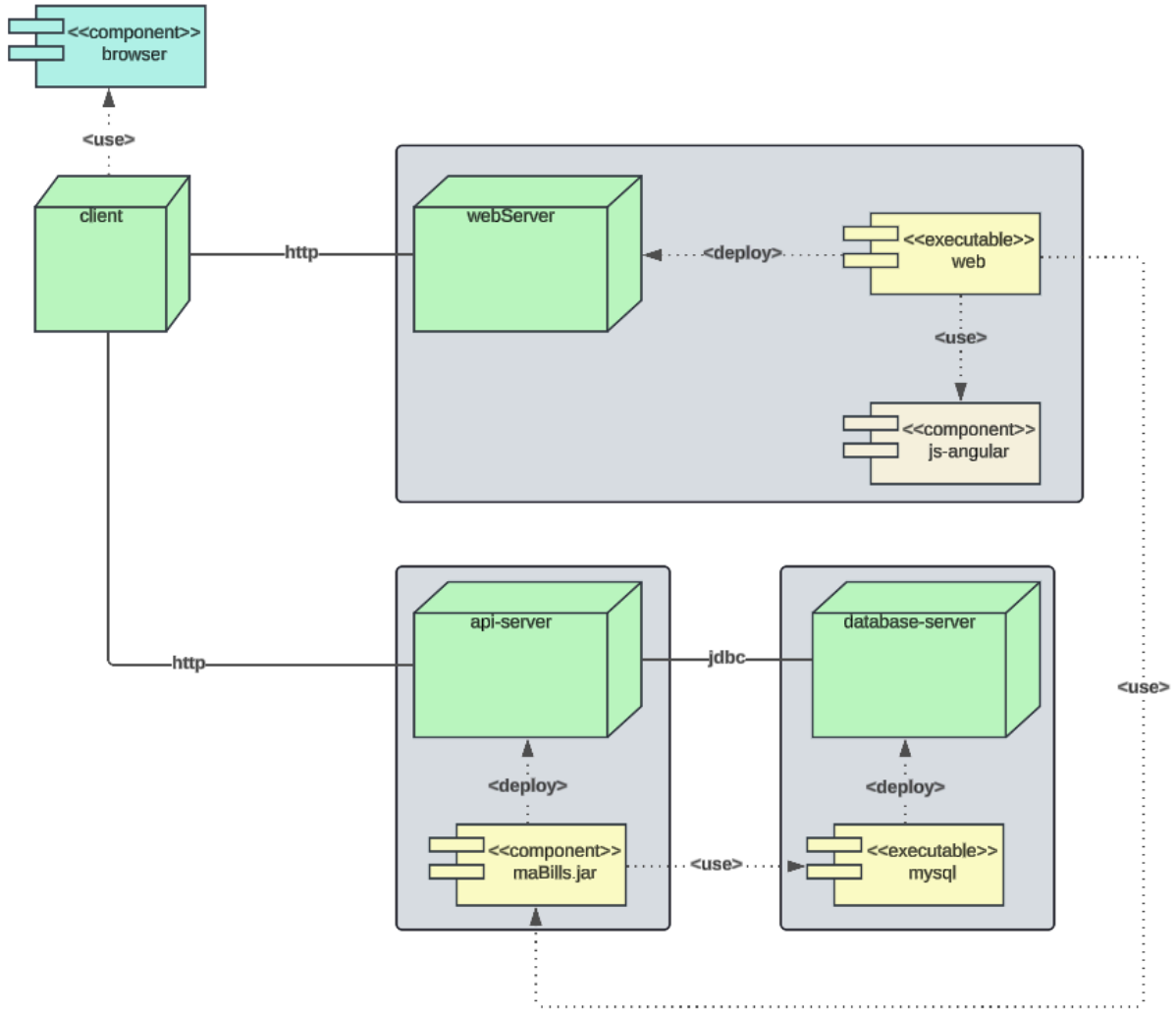
CAPÍTULO 4. DISEÑO

Este capítulo se centra en el desarrollo de modelos enfocados sobre los requisitos **no funcionales** y en el dominio de la solución, para poder dar paso a la implementación y pruebas del sistema.

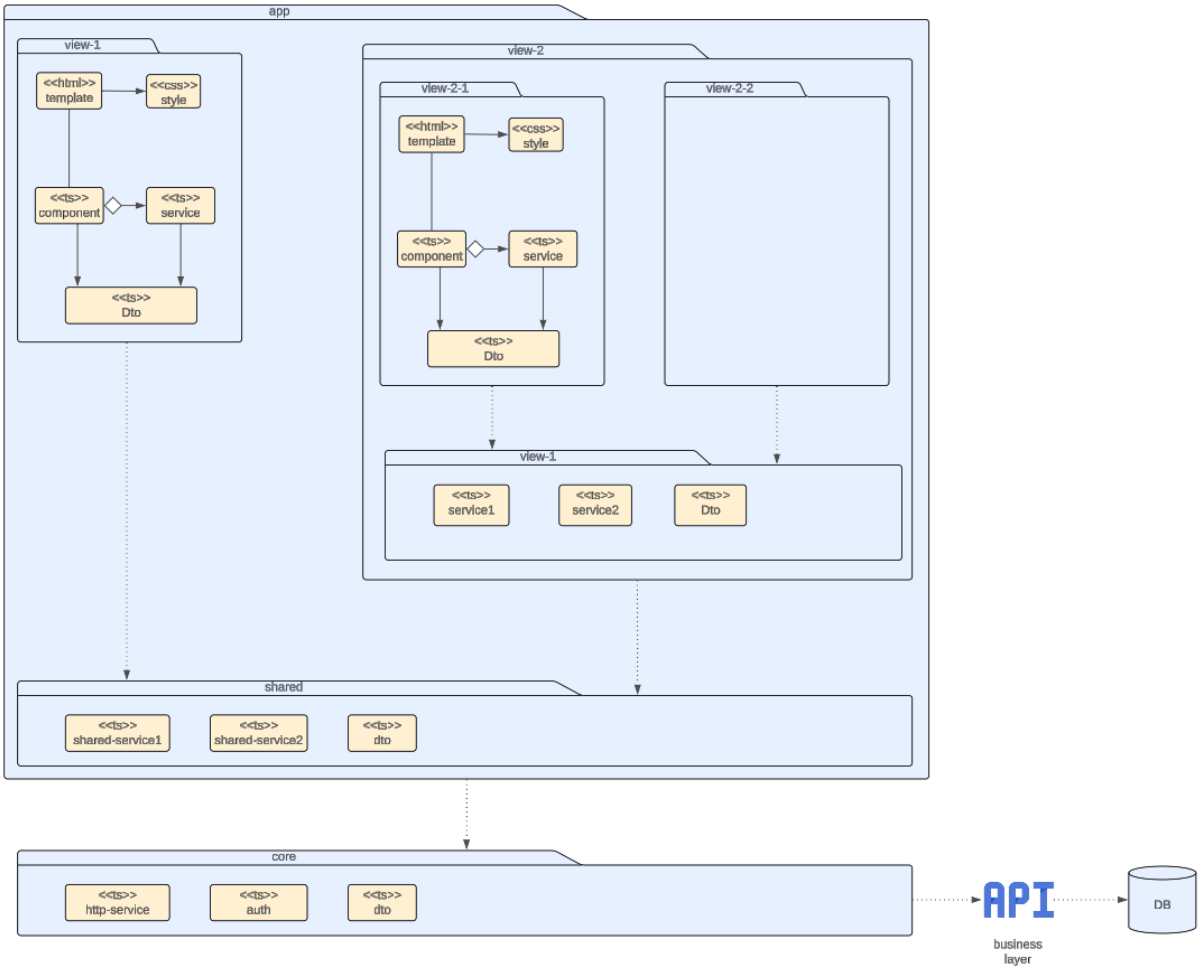
1. Diseño de la arquitectura

Esta aplicación está dividida por capas:

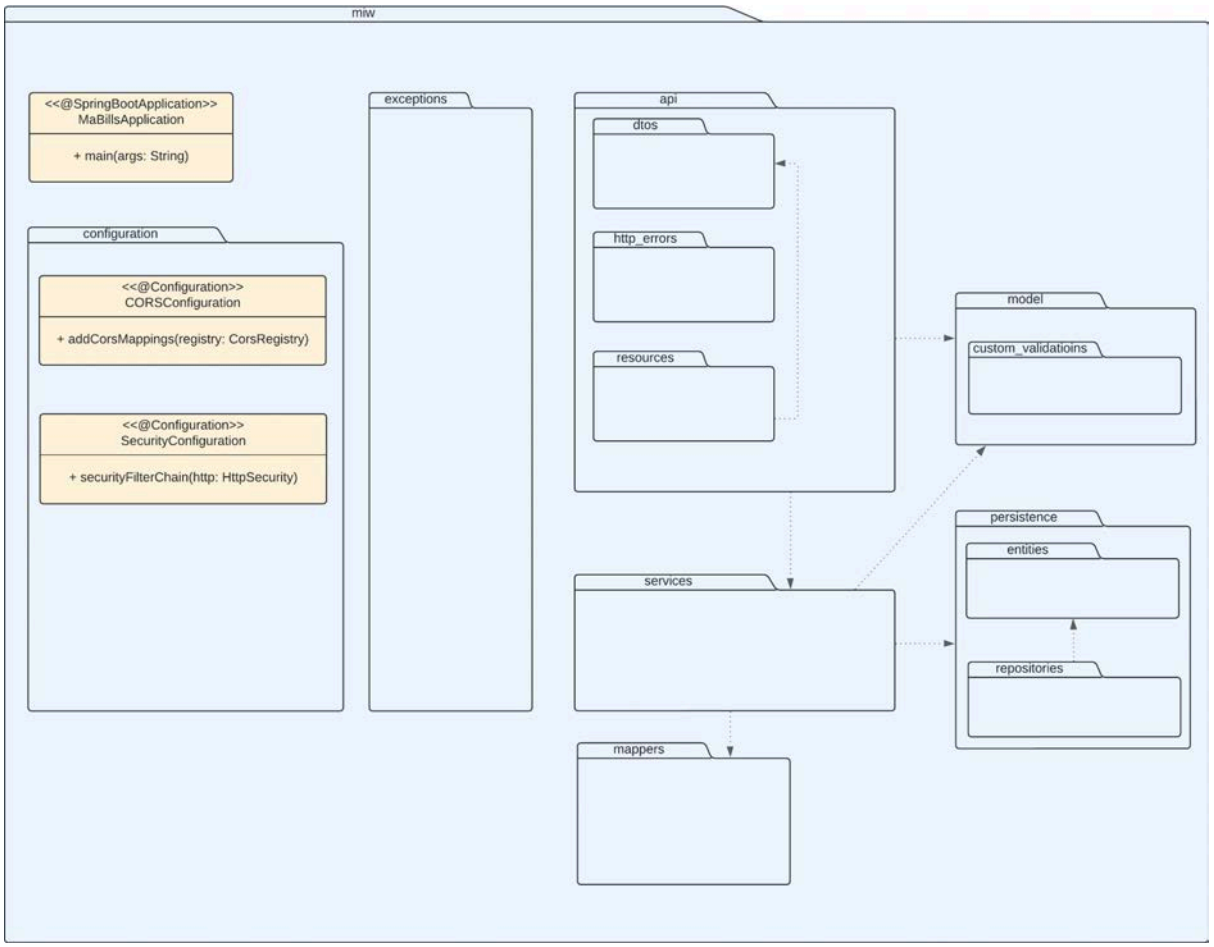
- Capa de presentación: esta capa está formada por un servidor web que permite al cliente realizar peticiones *http* para obtener un contenido que está hecha con *Angular,* este contenido será ejecutada en el cliente (navegador web) y será el encargado de realizar las llamadas *http* a la capa de negocio.
- Capa de negocio: en esta capa tenemos un servidor que contiene la api de la aplicación creada con el framework *Spring,* este será consumido por la capa de presentación y se comunicará también con la capa de datos mediante *jdbc*.
- Capa de datos: en esta capa tenemos un servidor de datos con *MySQL*.



La capa de presentación, utiliza la arquitectura basada en componentes, esta es una arquitectura que organiza los paquetes por componentes, cada componente tiene sus propios estilos, templates y lógica encapsulada.



La capa de negocio utiliza la arquitectura por capas, esta arquitectura permite organizar y estructurar el código de manera modular y mantenible, cada capa tiene una responsabilidad específica, sea acceso a datos, o lógica de negocio. Entre las capas se comunican a través de interfaces bien definidas.



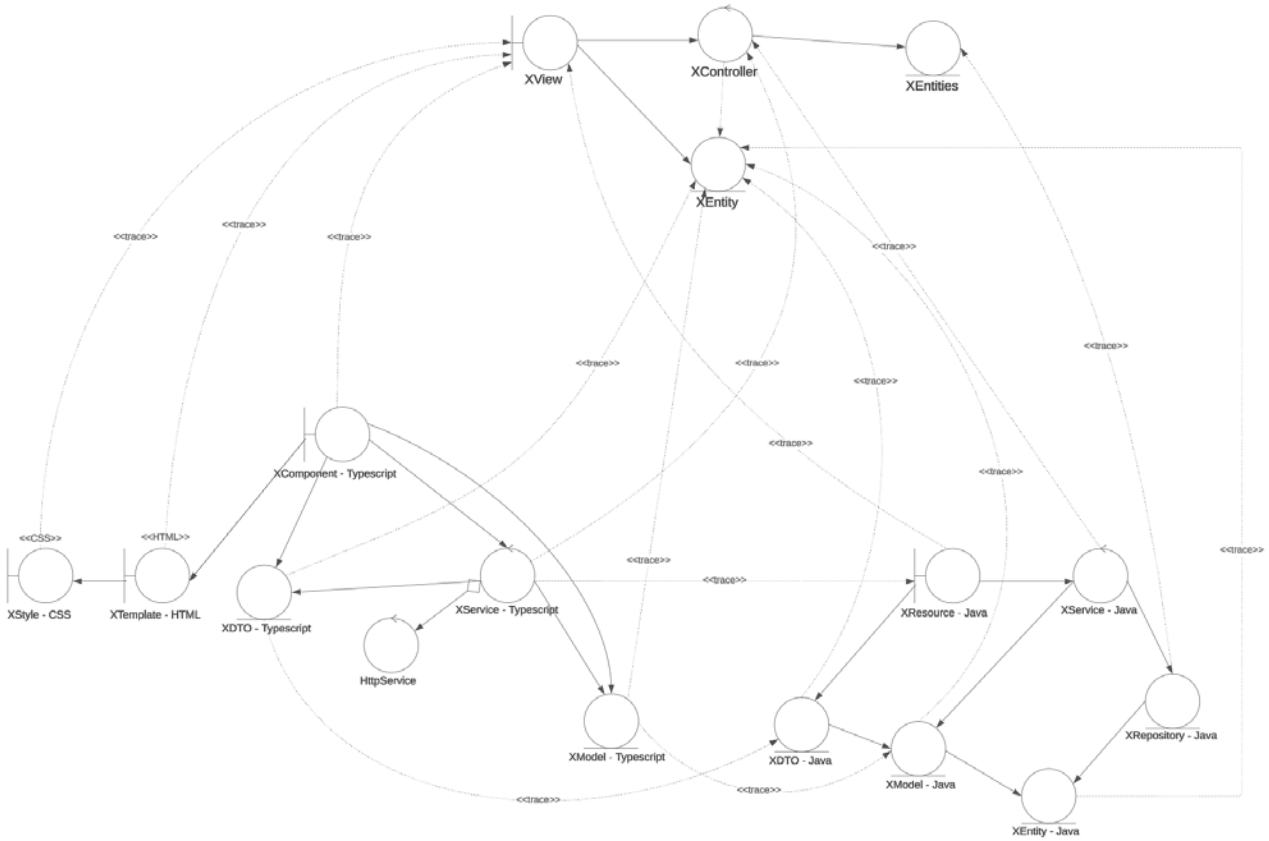
2. Diseño de caso de uso

El diagrama que se presenta a continuación tiene como objetivo explicar de forma genérica, las interacciones entre las distintas clases durante la realización de un caso de uso.

a. En la capa de presentación:
   - Cada *View* se corresponde a un *template* con su correspondiente *css* y su *component.*
   - Cada *Controller* se corresponde a un *Service* de la capa de presentación.
   - Cada *Entity* se corresponde a un *DTO* o a un *modelo*.

b. En la capa de negocio y datos:
   - Cada *Entity* se corresponde a una entidad de la base de datos, un DTO o un modelo.
   - Cada *View* se corresponde a un *Resource,* cuya función es presentar datos a los clientes.
   - Cada *Controller* se corresponde a un *Service*, que maneja la lógica de negocio.
   - Los *Resources* utilizan los *Services* para presentar los datos al cliente.

- Cada *Entities* se corresponde a un *Repository,* cuya responsabilidad es manejar operaciones con la base de datos.



CAPÍTULO 5. IMPLEMENTACIÓN

1. Ecosistema del desarrollo

Para la realización del proyecto, se ha definido un ecosistema del desarrollo compuesto por las siguientes herramientas:

- Gestión de registros: *Apache Log4j*
  Se ha decidido utilizar esta librería para la gestión de registros por su popularidad en la comunidad *Java*, aunque en algunas versiones antiguas han encontrado vulnerabilidades, en las últimas versiones se han corregido todas las vulnerabilidades y es seguro frente a ataques de inyección de registros.
- Sistema de control de versiones: *Git*
- Herramienta de análisis de código estático: *SonarCloud*
- Pruebas unitarias y de integración con *JUnit5, Mockito* y *H2 Database*
- Integración continua con *GitHub Actions*, detecta todos los cambios en la rama máster, dispara las pruebas y ejecuta la herramienta de análisis estático de código para cada cambio en la rama.

2. Herramientas del desarrollo

Se han utilizado las siguientes herramientas para desarrollar el proyecto:

1. JetBrains IntelliJ: *IDE* para el desarrollo de proyectos con lenguaje de programación *Java* o *Gradle*.
   Se ha utilizado esta herramienta para desarrollar la capa de negocio del proyecto, el motivo por el que se ha decidido utilizar esta herramienta es por su popularidad en la comunidad, ya que es considerada la mejor herramienta para el desarrollo de aplicaciones en *Java*, además, el *IDE* te ofrece múltiples opciones para que lo puedas personalizar a tu gusto, contando también con *plugins* que ayudan a detectar antes fallas en el código.
2. JetBrains WebStorm: *IDE* para el desarrollo de proyectos web.
   Se ha utilizado esta herramienta para desarrollar la capa de presentación del proyecto.
3. Google Chrome: navegador web.
4. GitHub: portal para alojar el proyecto, basado en *Git*.
5. DBeaver: herramienta de administración de base de datos.

3. Calidad interna del software

Durante la realización de la asignatura *Ingeniería Web Visión General,* se ha podido adquirir conocimientos sobre los estándares de calidad en el software. Dicho estándares han sido respetados durante todo el desarrollo del proyecto, manteniendo las 3 reglas de hierro: alta cohesión, bajo acoplamiento y de tamaño pequeño. Como resultado, se ha desarrollado un software de alta calidad.

Además, gracias a la asignatura de *Arquitectura y Patrones para Aplicaciones Web,* se ha adquirido conocimientos sobre diferentes arquitecturas y patrones de desarrollo que han sido de gran ayuda para mejorar la calidad del software, resolviendo problemas de cohesión y acoplamiento y mejorando la legibilidad del código.

A continuación, se va a listar una serie de estándares que se ha respetado durante el desarrollo del proyecto:

a. Estándares de legibilidad del código:
   - Elegir nombres descriptivos
   - Elegir nombres al nivel de abstracción apropiado
   - Utilizar nomenclatura estándar siempre que sea posible
   - Evitar ambigüedades en los nombres
   - Evitar los comentarios en el código, ya que el código debe ser autoexplicativo
   - Formatear el código según los estándares de la comunidad, sin romper nunca las reglas del sangrado.
   - Seguir las convenciones estándares
   - Mantener un sólo estilo de programación para mejorar la consistencia
   - Una línea como máximo tiene entre 80 y 120 caracteres

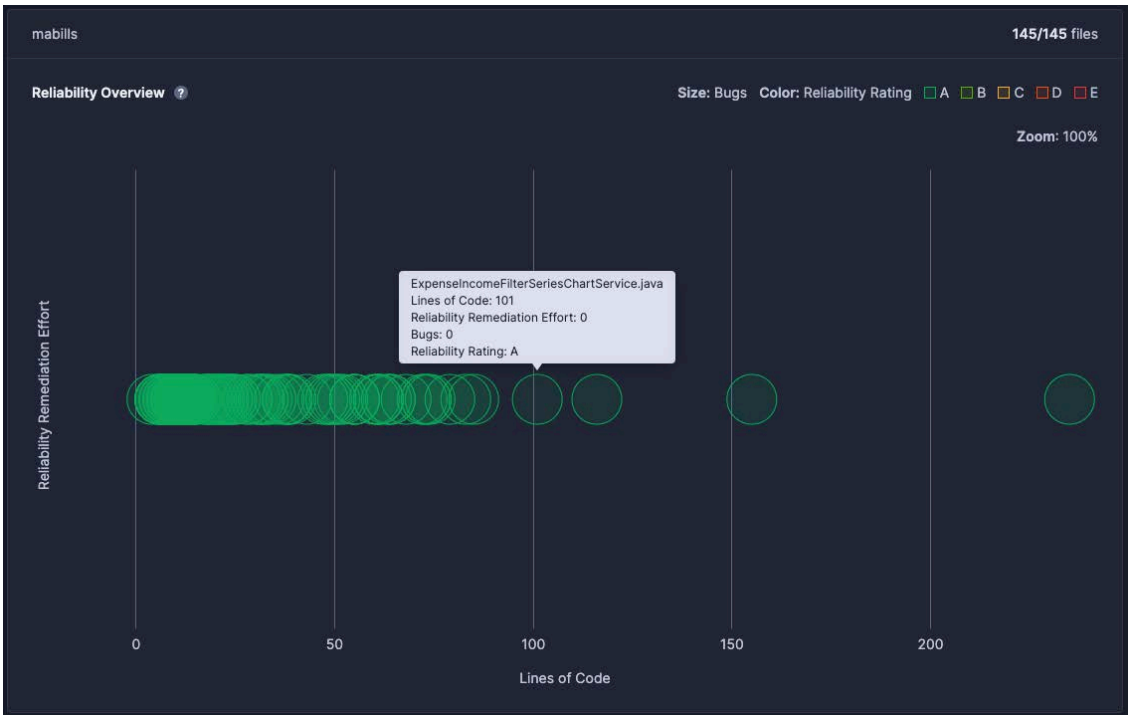
b. Estándares de calidad del código:
   - Evitar códigos malolientes (*Smell Codes*) siguiendo estas reglas: *DRY*, *YAGNI*, *KISS*, eliminar código muerto etc.

c. Estándares diseño de métodos:
   - Los nombres de los métodos explican la responsabilidad del método
   - Cohesión de los métodos: cada método tiene una única responsabilidad (debe hacer una sola cosa)
   - Lista de parámetros:
     - De media, tener 1 o 2 parámetros por cada método.
     - Como máximo, tener 3 parámetros en un método.
   - Número de líneas de códigos:
     - Máximo: 15-25 líneas de código por cada método
   - Como máximo, tener 3 sentencias anidadas en un método.
   - Complejidad ciclomática de McCabe, como máximo entre 10 y 25.

d. Estándares de diseño de clases:
   - De media cada clase tiene 3 atributos
   - Como máximo una clase tiene 5 atributos
   - Una clase tiene como máximo entre 20 y 25 métodos
   - Una clase tiene como máximo entre 200 y 500 líneas de código
   - Evitar código sucio por clases perezosas
   - Evitar código sucio por obsesión por tipos primitivos
   - Principio de responsabilidad única: una clase debe tener sólamente un motivo de cambio

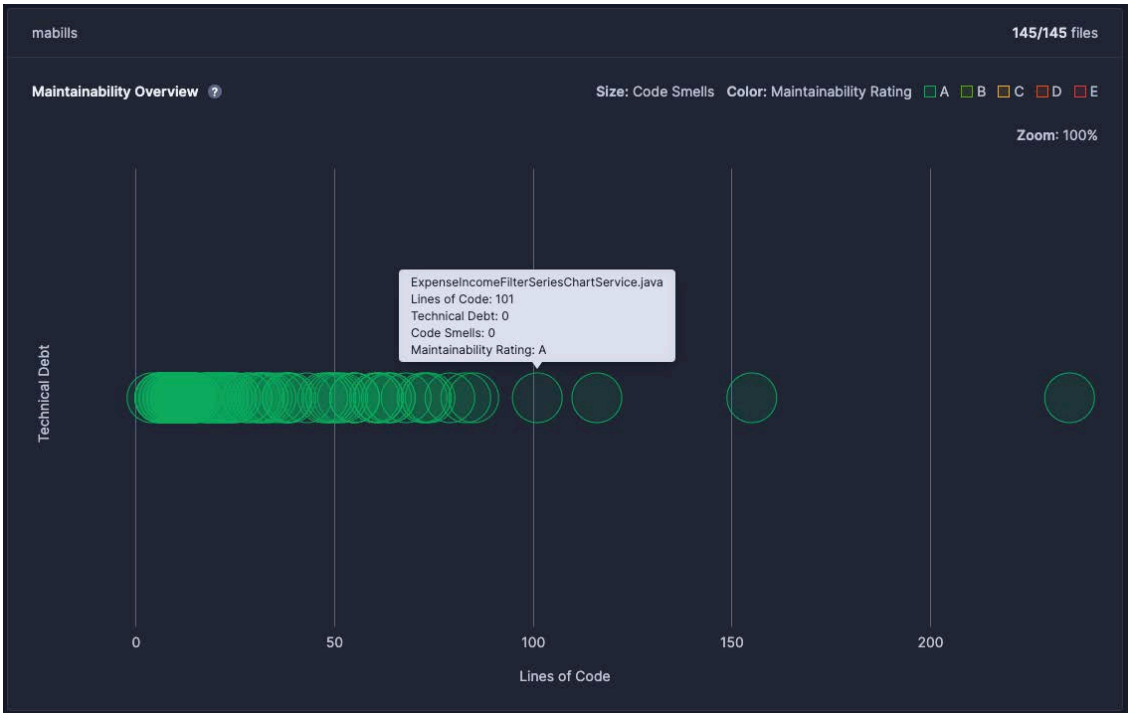
Siguiendo los estándares anteriores, se ha obtenido las siguientes gráficas en *SonarCloud* que pueden servir como prueba de que el código es de calidad:

Fiabilidad



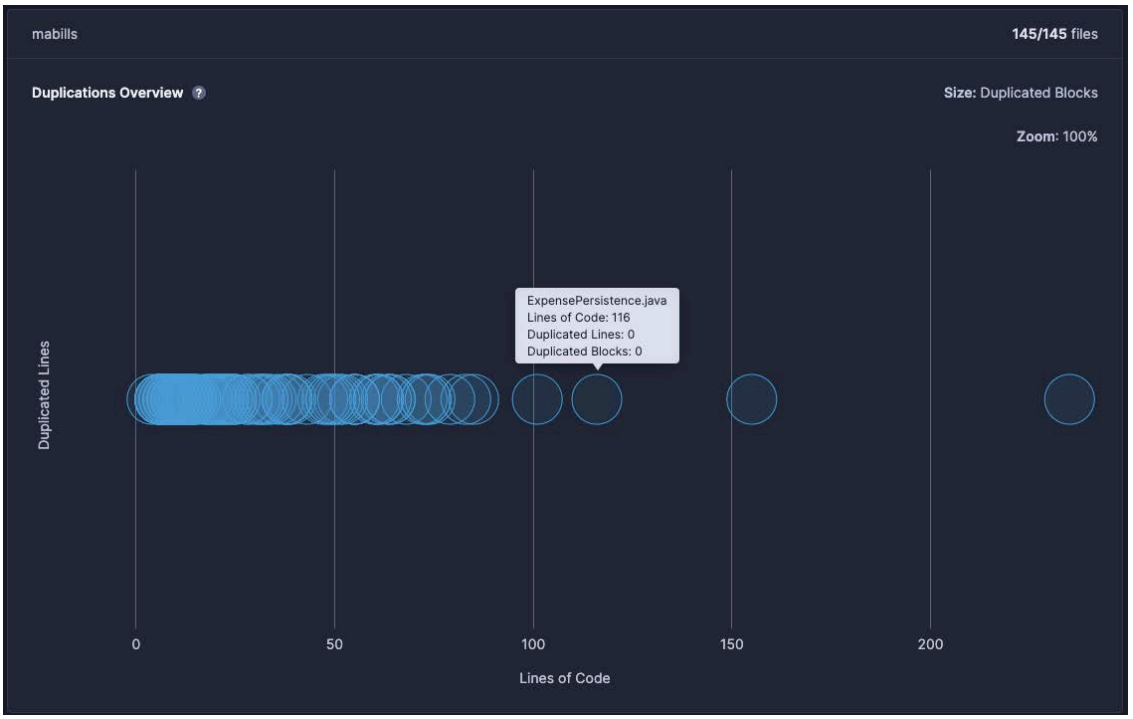
Esta gráfica muestra la fiabilidad de las clases del proyecto, una alta fiabilidad indica que el código tiene **menos probabilidad** de tener errores, como se puede observar, todas las clases tienen una fiabilidad de nivel A en *SonarCloud.*

Mantenibilidad



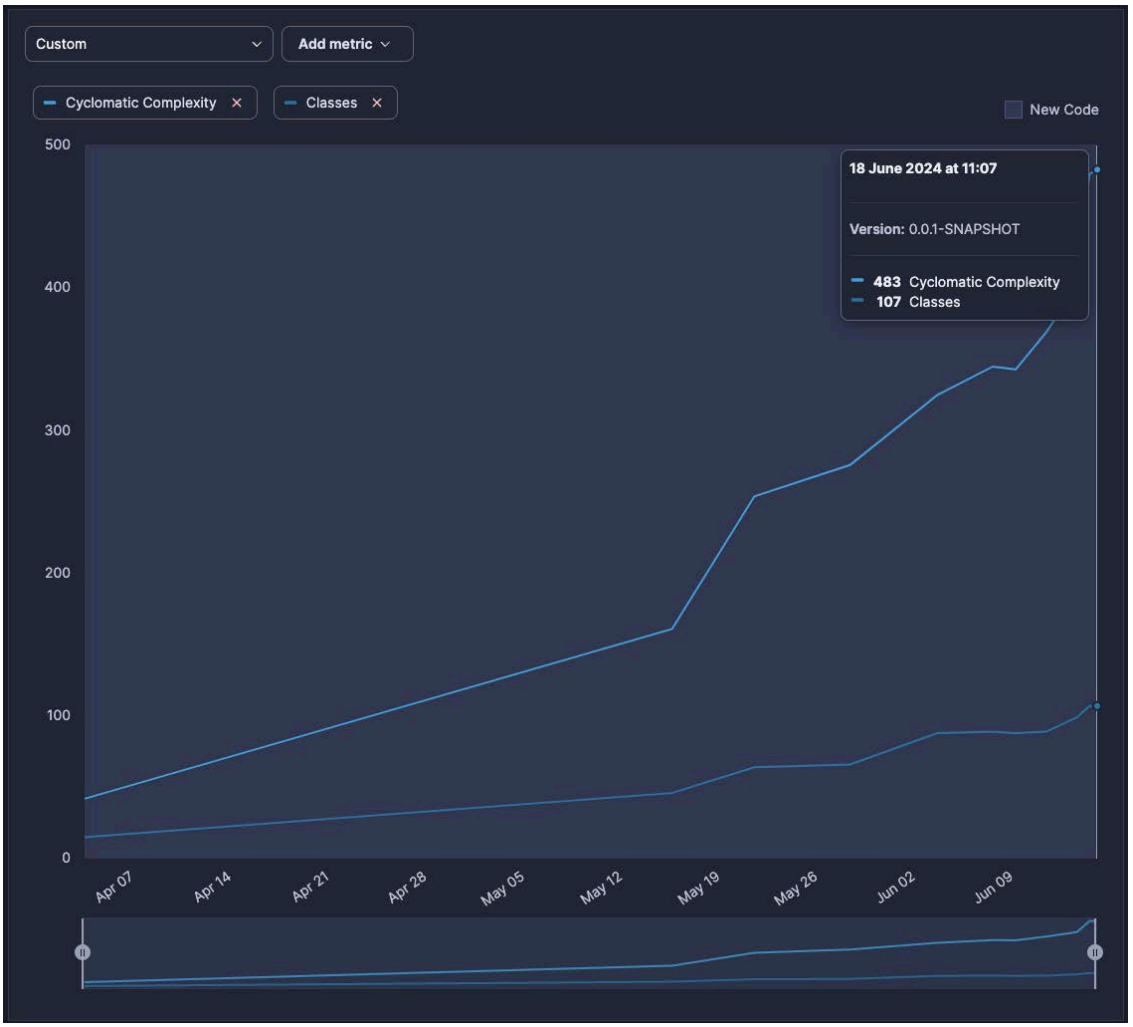
Esta segunda gráfica muestra la mantenibilidad (facilidad de modificar, mejorar o corregir el código) del proyecto, al igual que la fiabilidad. Se puede observar que todas las clases del proyecto tienen una calificación de nivel A.

Duplicaciones de código



Se puede apreciar en esta gráfica que el proyecto no tiene ninguna duplicación de código.

Complejidad ciclomática



En esta gráfica se muestra el número de clases y la complejidad ciclomática de todo el proyecto durante el tiempo, se puede observar que a medida que se aumenta el número de clases en el proyecto, se aumenta también la complejidad ciclomática, esto no quiere decir que el proyecto es de mala calidad, ya que este indicador de complejidad ciclomática es la **suma** de la complejidad ciclomática de todas las clases.
Dividiendo el indicador de complejidad ciclomática entre el número de clases, se obtiene como resultado una complejidad ciclomática media de 4,51 por clase.

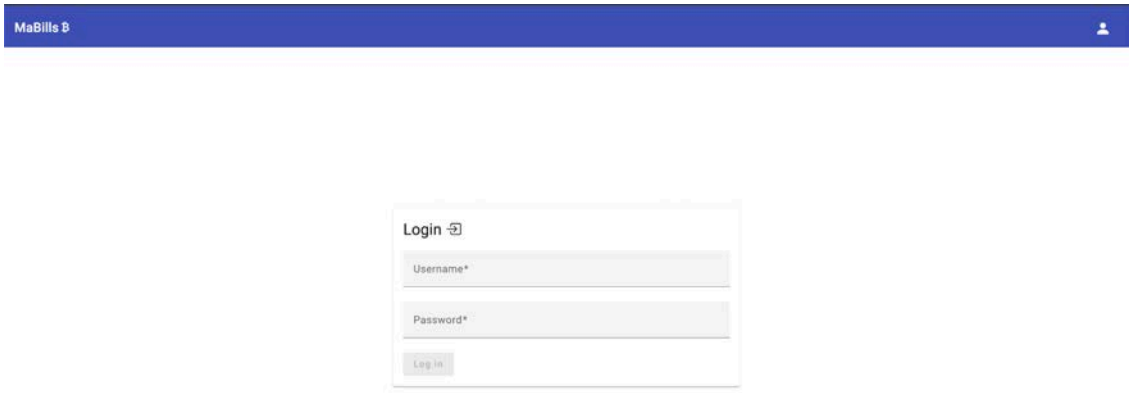
Smell codes

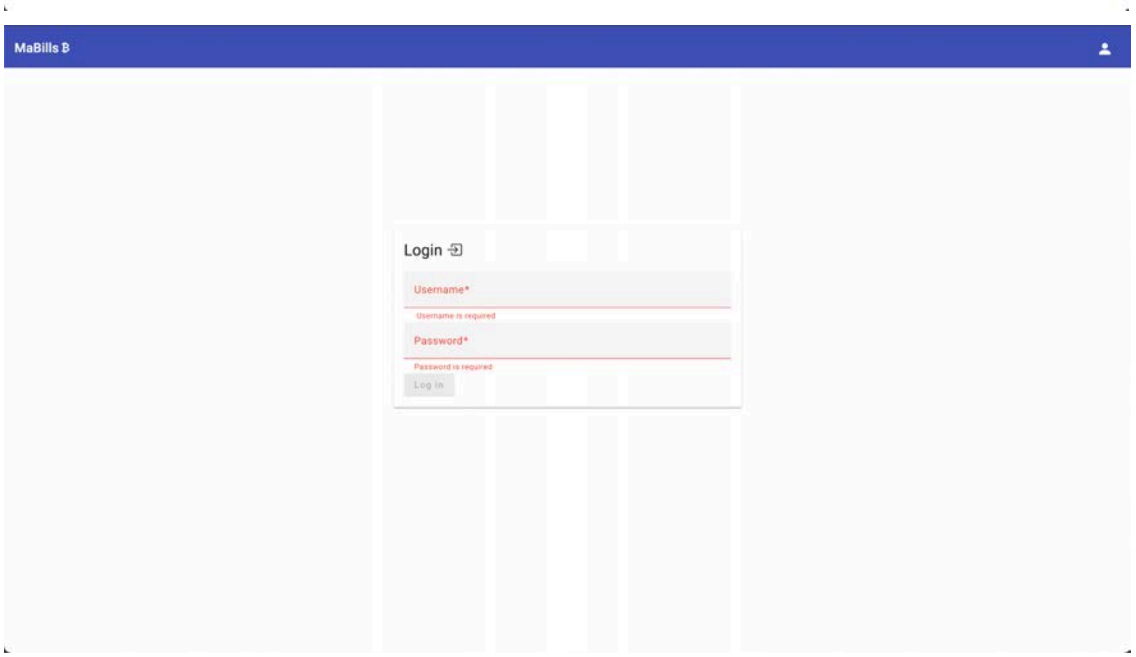
Con esta gráfica se puede observar que el número de *smell codes* ha ido variando en el tiempo, alcanzando su pico (4 *smell codes*) entre el 5 y el 12 de Mayo, posteriormente, hasta Junio se ha mantenido el número de *smell codes* (3) y finalmente, se han corregido **todos** los *smell codes*, lo que indica que actualmente el proyecto tiene una mejor calidad.

4. Calidad externa del software

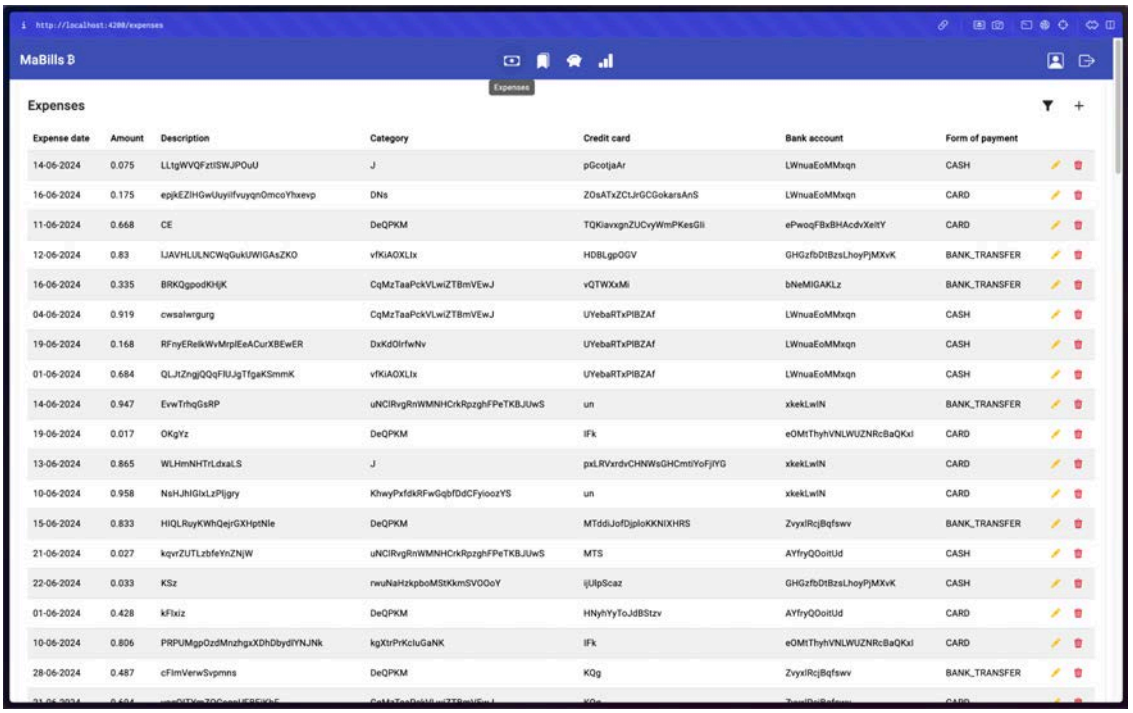
Se han respetado los estándares de calidad externa del software, como prueba de ello, se presenta el resultado final de la aplicación.
Tengan en cuenta que, los datos que aparecen en estas capturas de pantalla son datos de prueba generados directamente sobre la base de datos, es por eso que hay algunos datos que no tienen el valor o formato correcto:

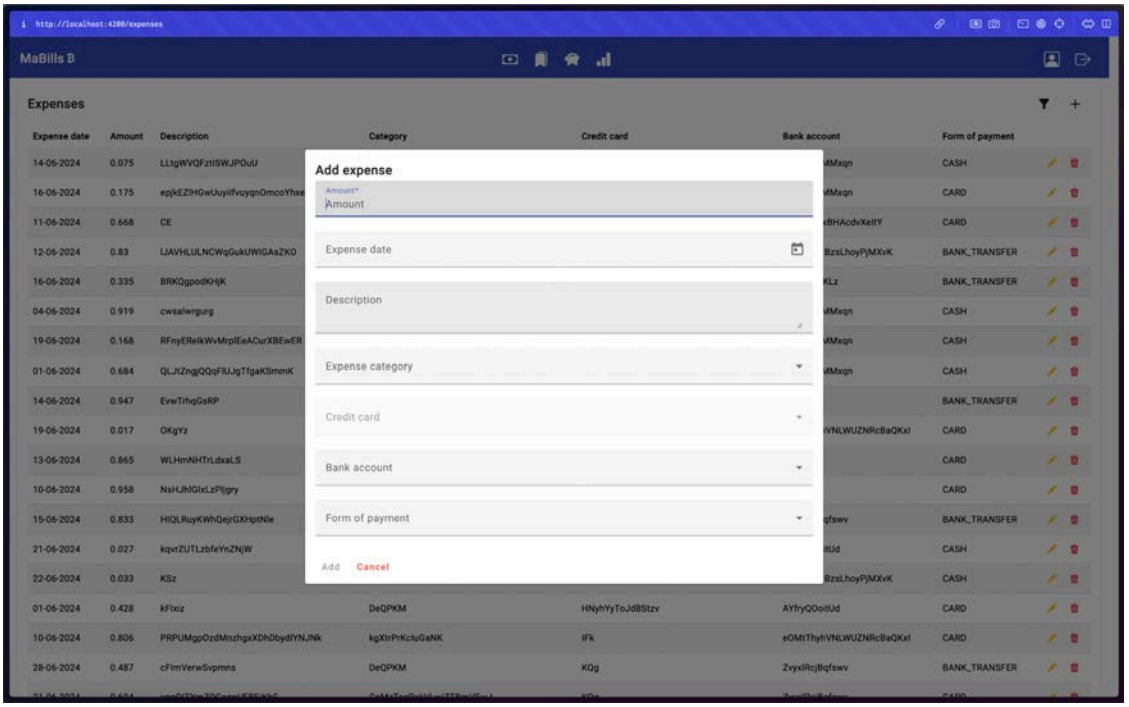
Login

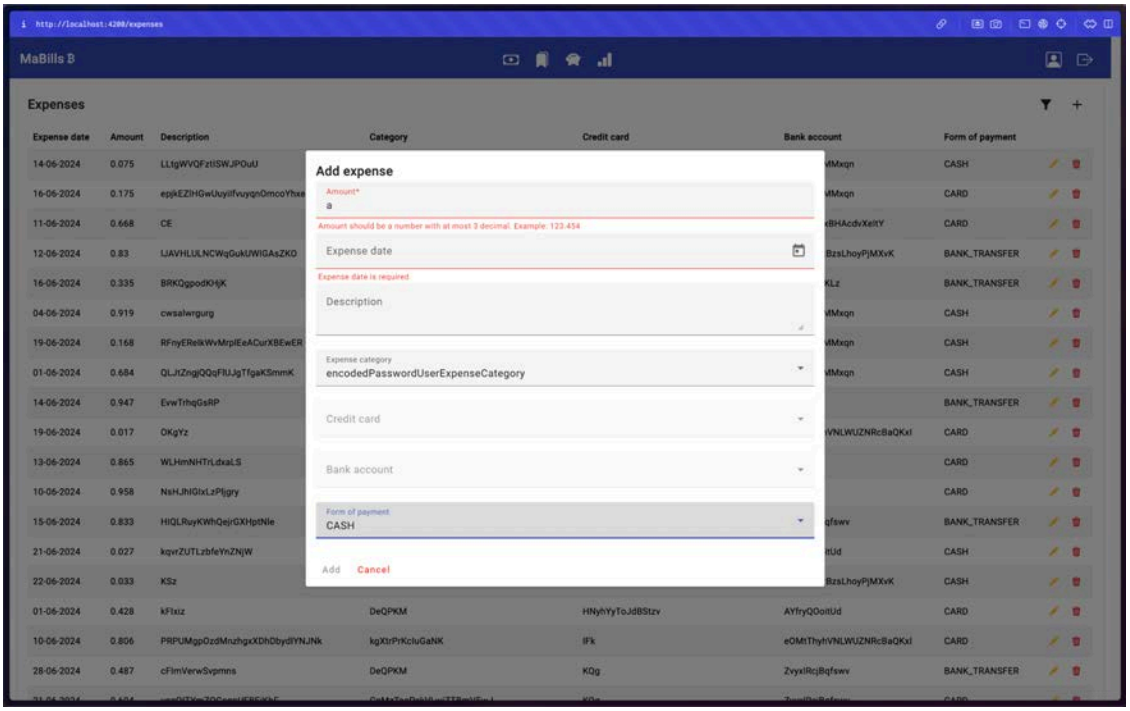


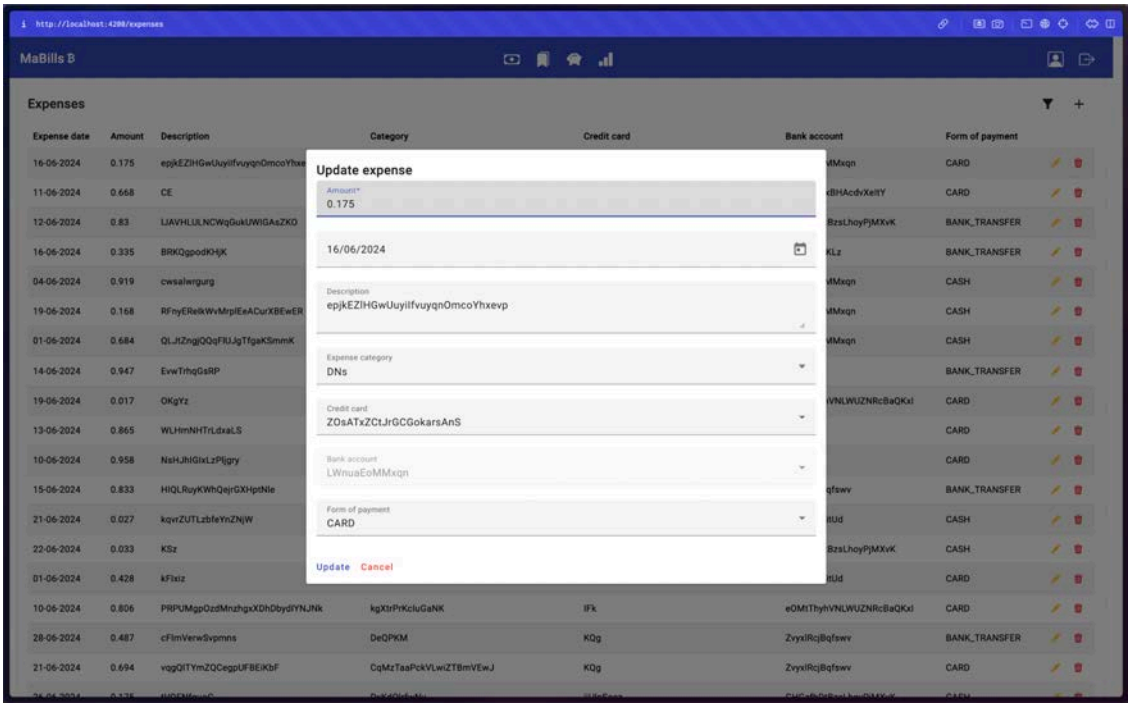


Expenses

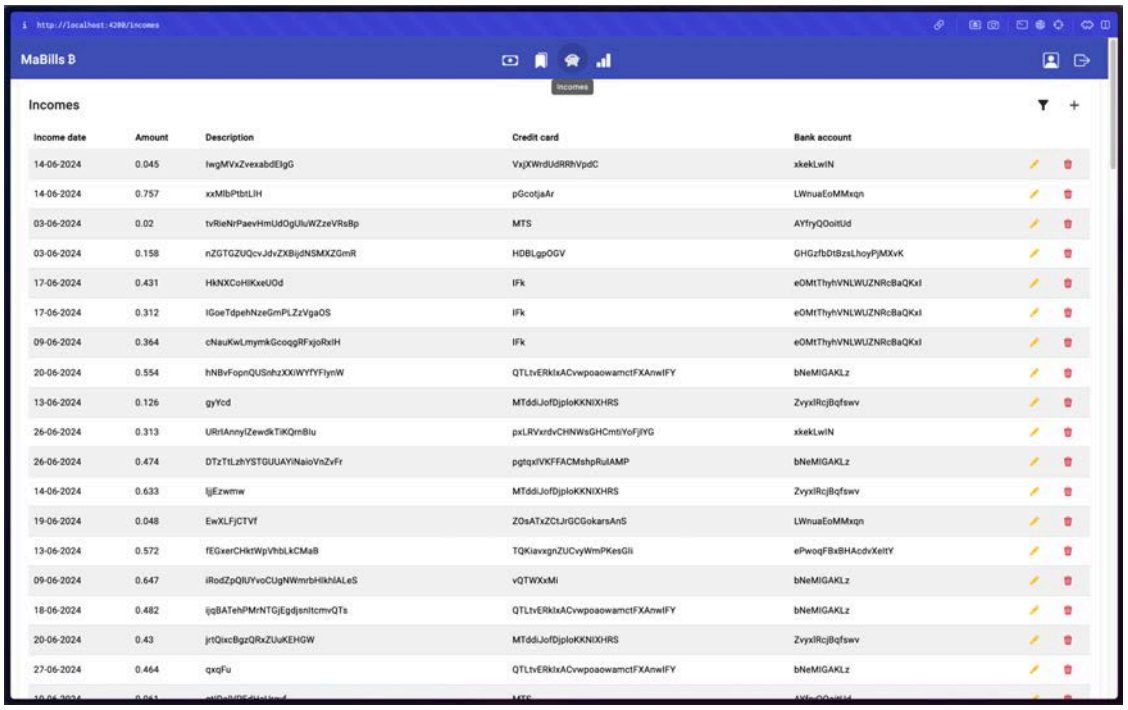

http://localhost:4200/expenses
MaBills B
Expenses
Expenses
Expense date
Amount
Description
Category
Credit card
Bank account
Form of payment
14-06-2024 0.075 LLtgWVQFztlSWJPOuU J pGcotjaAr LWnuaEoMMxqn CASH
16-06-2024 0.175 epjkEZlHGwUuyiIfvuyqnOmcoYhxevp DNs ZOsATxZCtJrGCGokarsAnS LWnuaEoMMxqn CARD
11-06-2024 0.668 CE DeQPKM TQKiavxgnZUCvyWmPKesGli ePwoqFBxBHAcdvXeltY CARD
12-06-2024 0.83 IJAVHLULNCWqGukUWlGAsZKO vfKiAOXLIx HDBLgpOGV GHGzfbDtBzsLhoyPjMXvK BANK_TRANSFER
16-06-2024 0.335 BRKQgpodKHjK CqMzTaaPckVLwiZTBmVEwJ vQTWXxMi bNeMlGAKLz BANK_TRANSFER
04-06-2024 0.919 cwsalwrgurg CqMzTaaPckVLwiZTBmVEwJ UYebaRTxPIBZAf LWnuaEoMMxqn CASH
19-06-2024 0.168 RFnyERelkWvMrplEeACurXBEwER DxKdOlrfwNv UYebaRTxPIBZAf LWnuaEoMMxqn CASH
01-06-2024 0.684 QLJtZngjQQqFlUJgTfgaKSmmK vfKiAOXLIx UYebaRTxPIBZAf LWnuaEoMMxqn CASH
14-06-2024 0.947 EvwTrhqGsRP uNClRvgRnWMNHCrkRpzghFPeTKBJUwS un xkekLwIN BANK_TRANSFER
19-06-2024 0.017 OKgYz DeQPKM IFk eOMtThyhVNLWUZNRcBaQKxl CARD
13-06-2024 0.865 WLHmNHTrLdxaLS J pxLRVxrdvCHNWsGHCmtiYoFjlYG xkekLwIN CARD
10-06-2024 0.958 NsHJhIGlxLzPljgry KhwyPxfdkRFwGqbfDdCFyiioozYS un xkekLwIN CARD
15-06-2024 0.833 HIQLRuyKWhQejrGXHptNle DeQPKM MTddiJofDjploKKNIXHRS ZvyxlRcjBqfswv BANK_TRANSFER
21-06-2024 0.027 kqvrZUTLzbfeYnZNjW uNClRvgRnWMNHCrkRpzghFPeTKBJUwS MTS AYfryQOoitUd CASH
22-06-2024 0.033 KSz rwuNaHzkpboMStKkmSVOOoY ijUlpScaz GHGzfbDtBzsLhoyPjMXvK CASH
01-06-2024 0.428 kFlxiz DeQPKM HNyhYyToJdBStzv AYfryQOoitUd CARD
10-06-2024 0.806 PRPUMgpOzdMnzhgxXDhDbydIYNJNk kgXtrPrKcluGaNK IFk eOMtThyhVNLWUZNRcBaQKxl CARD
28-06-2024 0.487 cFlmVerwSvpmns DeQPKM KQg ZvyxlRcjBqfswv BANK_TRANSFER

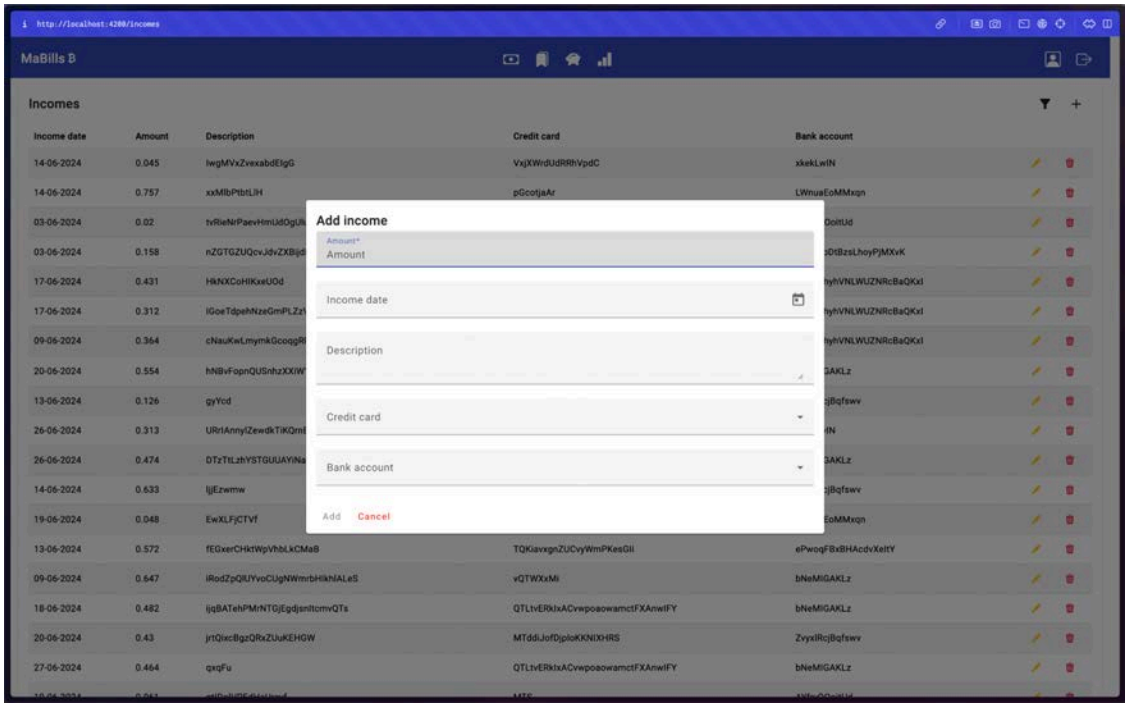

http://localhost:4200/expenses
MaBills B
Expenses
Add expense
Amount*
Amount
Expense date
Description
Expense category
Credit card
Bank account
Form of payment
Add
Cancel

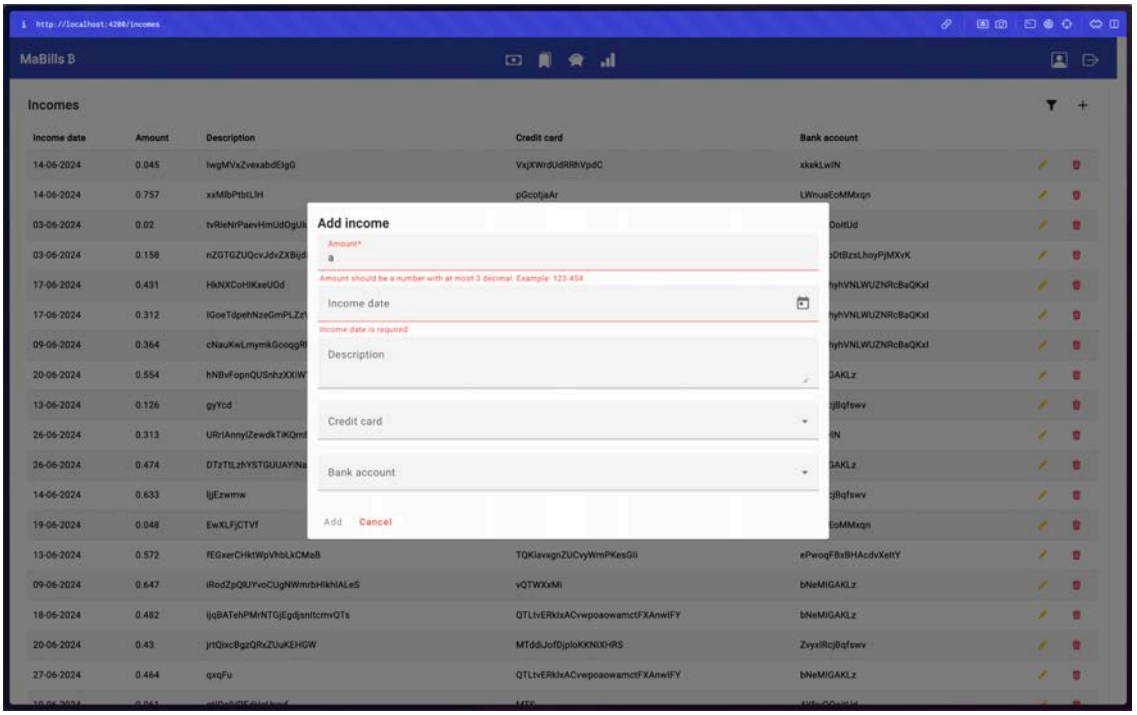

MaBills B
Expenses
Add expense
Amount*
a
Amount should be a number with at most 3 decimal. Example: 123.454
Expense date
Expense date is required
Description
Expense category
encodedPasswordUserExpenseCategory
Credit card
Bank account
Form of payment
CASH
Add
Cancel

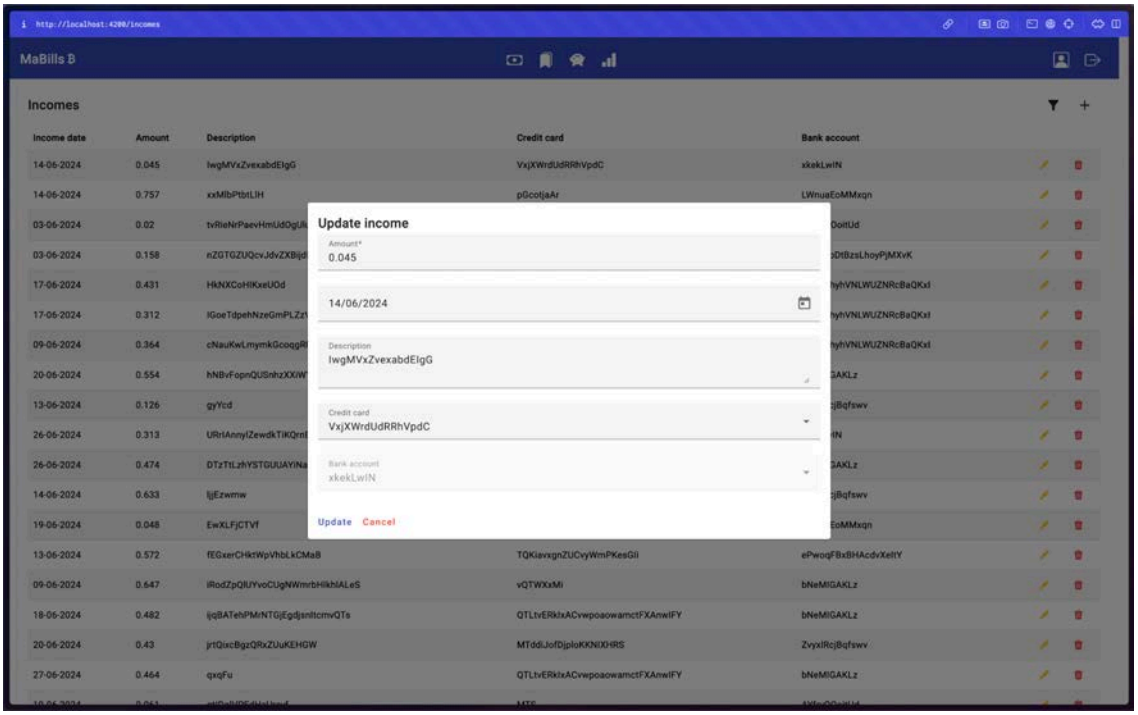

MaBills B
Expenses
Update expense
Amount*
0.175
16/06/2024
Description
epjkEZlHGwUuyilfvuyqnOmcoYhxevp
Expense category
DNs
Credit card
ZOsATxZCtJrGCGokarsAnS
Bank account
LWnuaEoMMxqn
Form of payment
CARD
Update
Cancel

Incomes

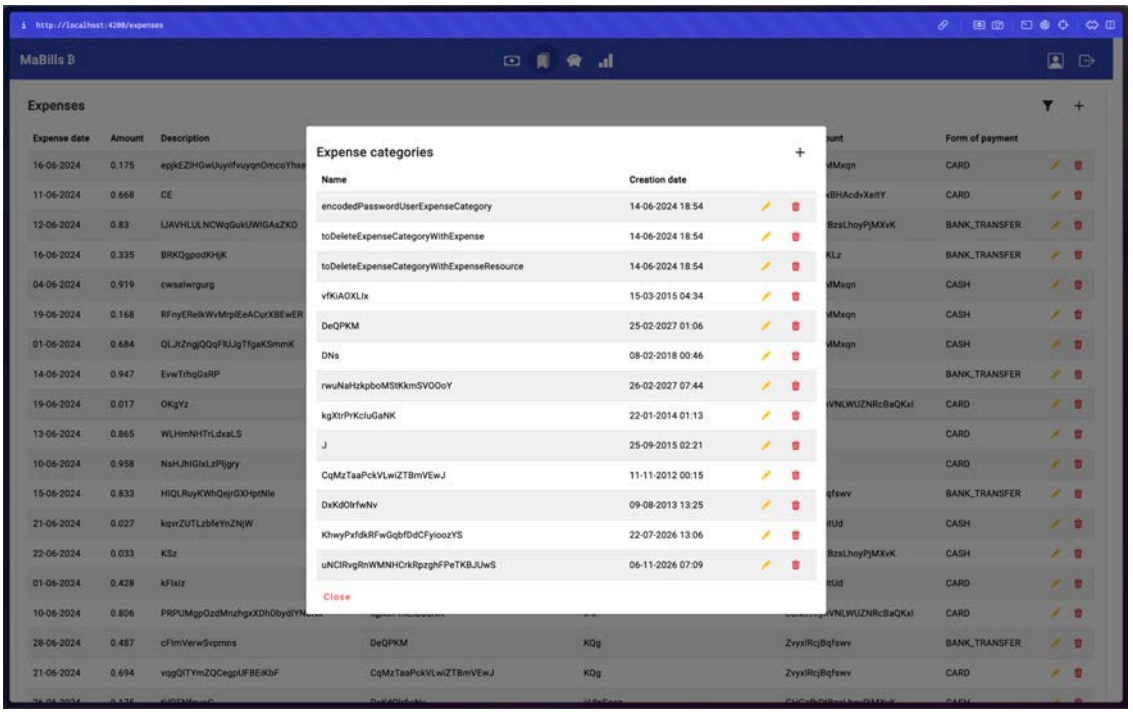

http://localhost:4200/incomes
MaBills B
Incomes
Incomes
Income date
Amount
Description
Credit card
Bank account
14-06-2024 0.045 IwgMVxZvexabdElgG VxjXWrdUdRRhVpdC xkekLwIN
14-06-2024 0.757 xxMlbPtbtLIH pGcotjaAr LWnuaEoMMxqn
03-06-2024 0.02 tvRieNrPaevHmUdOgUluWZzeVRsBp MTS AYfryQOoitUd
03-06-2024 0.158 nZGTGZUQcvJdvZXBijdNSMXZGmR HDBLgpOGV GHGzfbDtBzsLhoyPjMXvK
17-06-2024 0.431 HkNXCoHlKxeUOd IFk eOMtThyhVNLWUZNRcBaQKxI
17-06-2024 0.312 IGoeTdpehNzeGmPLZzVgaOS IFk eOMtThyhVNLWUZNRcBaQKxI
09-06-2024 0.364 cNauKwLmymkGcoqgRFxjoRxIH IFk eOMtThyhVNLWUZNRcBaQKxI
20-06-2024 0.554 hNBvFopnQUSnhzXXiWYfYFlynW QTLtvERkIxACvwpoaowamctFXAnwIFY bNeMlGAKLz
13-06-2024 0.126 gyYcd MTddiJofDjploKKNiXHRS ZvyxlRcjBqfswv
26-06-2024 0.313 URrIAnnylZewdkTiKQrnBlu pxLRVxrdvCHNWsGHCmtiYoFjlYG xkekLwIN
26-06-2024 0.474 DTzTtLzhYSTGUUAYiNaioVnZvFr pgtqxlVKFFACMshpRulAMP bNeMlGAKLz
14-06-2024 0.633 IjjEzwmw MTddiJofDjploKKNiXHRS ZvyxlRcjBqfswv
19-06-2024 0.048 EwXLFjCTVf ZOsATxZCtJrGCGokarsAnS LWnuaEoMMxqn
13-06-2024 0.572 fEGxerCHktWpVhbLkCMaB TQKiavxgnZUCvyWmPKesGli ePwoqFBxBHAcdvXeltY
09-06-2024 0.647 iRodZpQlUYvoCUgNWmrbHlkhlALeS vQTWXxMi bNeMlGAKLz
18-06-2024 0.482 ijqBATehPMrNTGjEgdjsnItcmvQTs QTLtvERkIxACvwpoaowamctFXAnwIFY bNeMlGAKLz
20-06-2024 0.43 jrtQixcBgzQRxZUuKEHGW MTddiJofDjploKKNiXHRS ZvyxlRcjBqfswv
27-06-2024 0.464 qxqFu QTLtvERkIxACvwpoaowamctFXAnwIFY bNeMlGAKLz

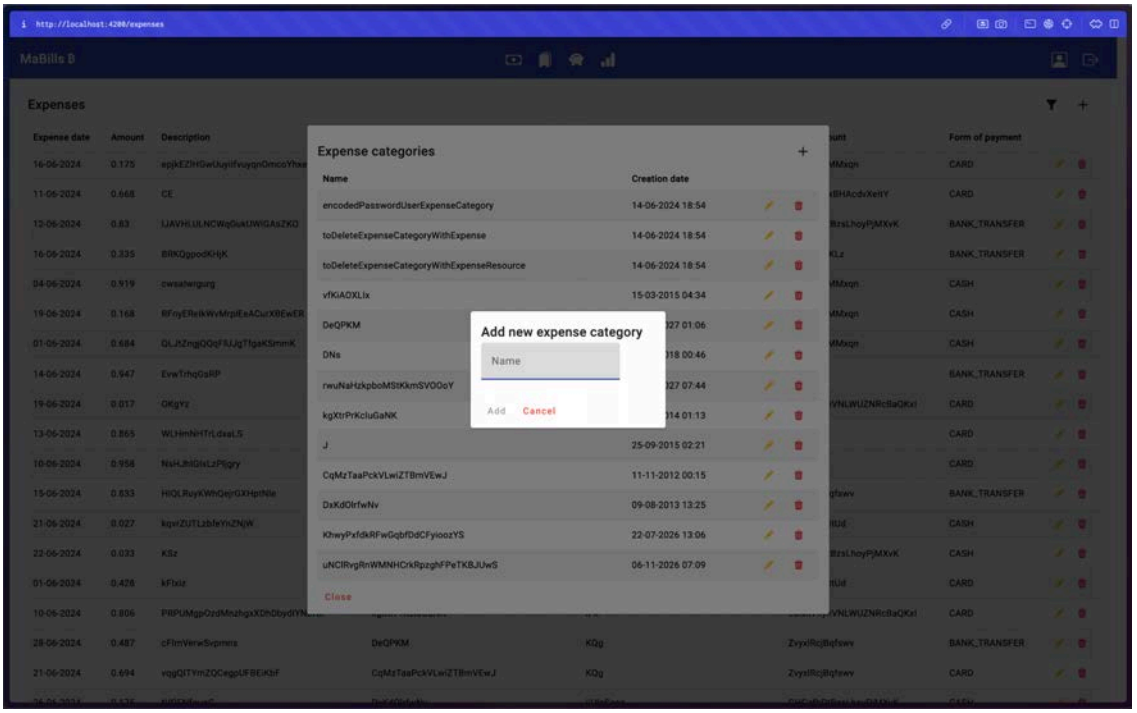

http://localhost:4200/incomes
MaBills B
Incomes
Add income
Amount*
Amount
Income date
Description
Credit card
Bank account
Add
Cancel

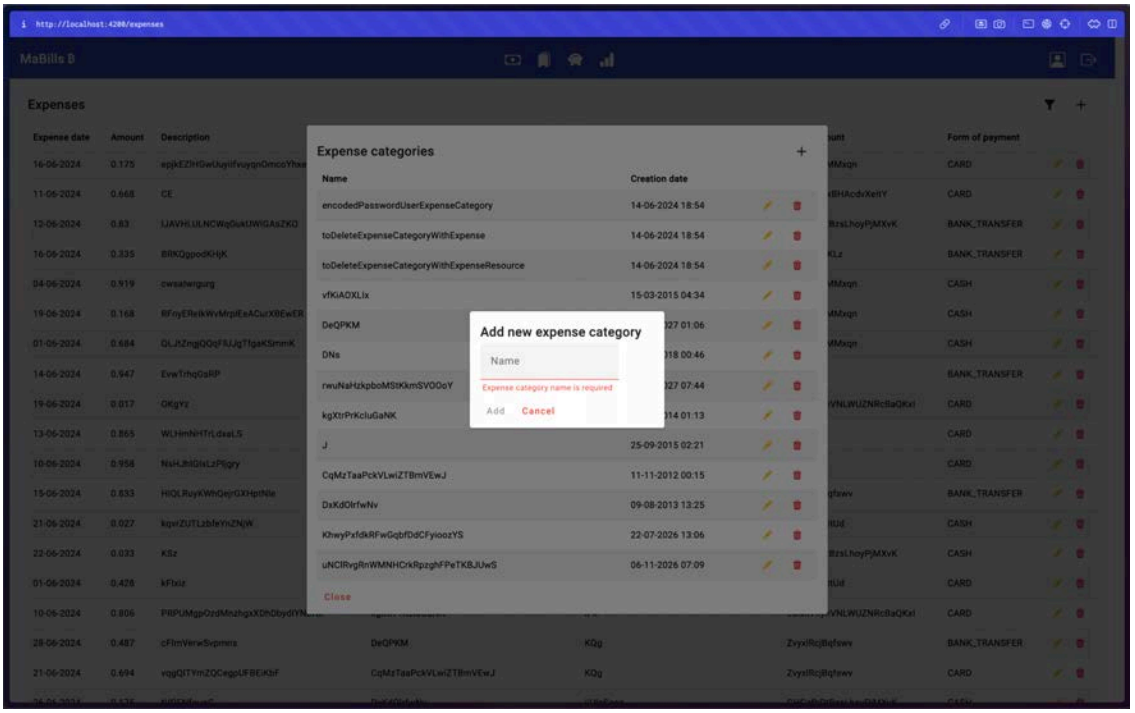

MaBills B
Incomes
Income date
Amount
Description
Credit card
Bank account
Add income
Amount*
a
Amount should be a number with at most 3 decimal. Example: 123.454
Income date
Income date is required
Description
Credit card
Bank account
Add
Cancel

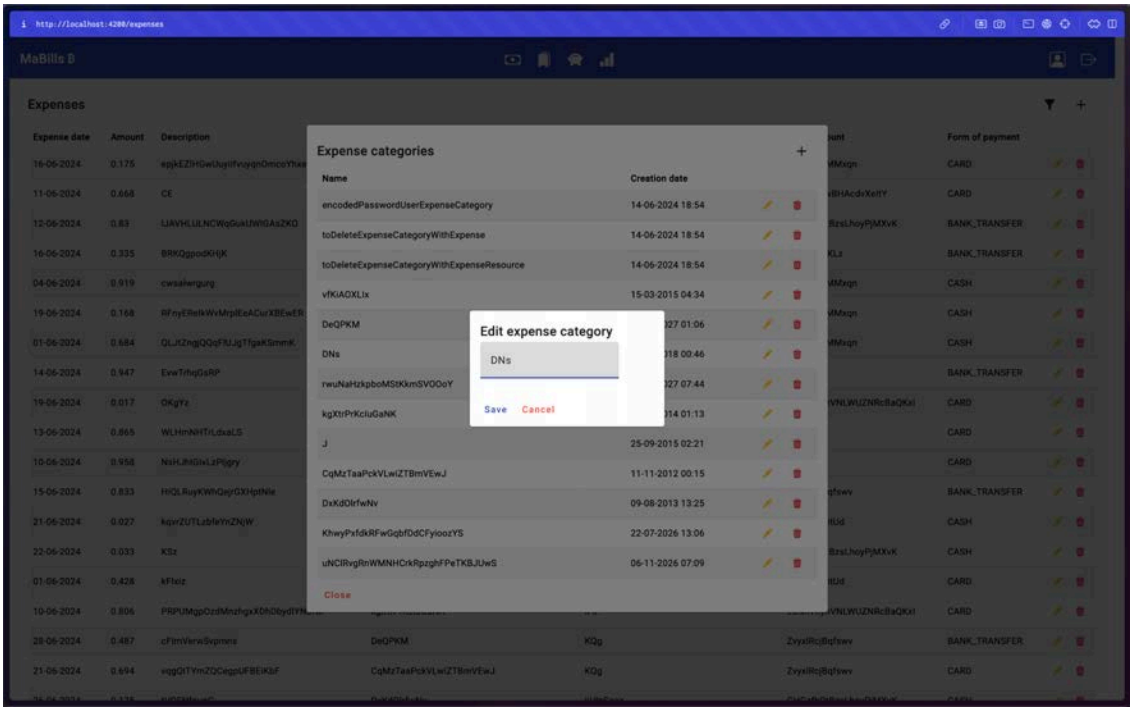

MaBills B
Incomes
Income date
Amount
Description
Credit card
Bank account
Update income
Amount*
0.045
14/06/2024
Description
lwgMVxZvexabdElgG
Credit card
VxjXWrdUdRRhVpdC
Bank account
xkekLwIN
Update
Cancel

Expense Categories

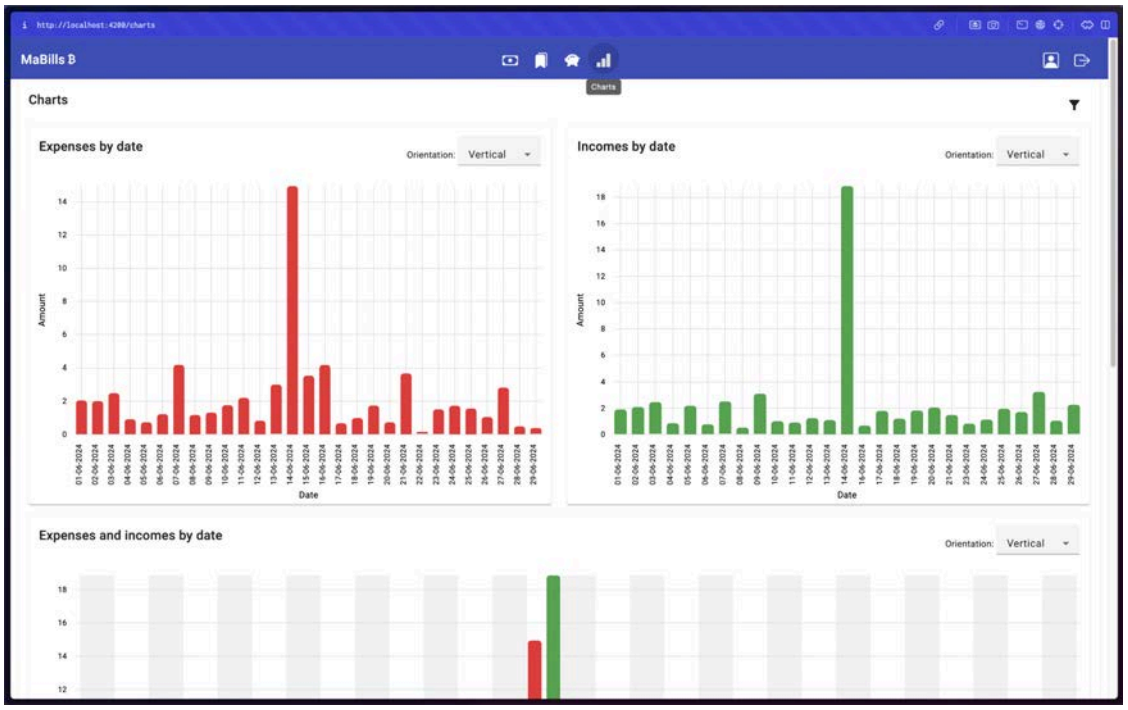


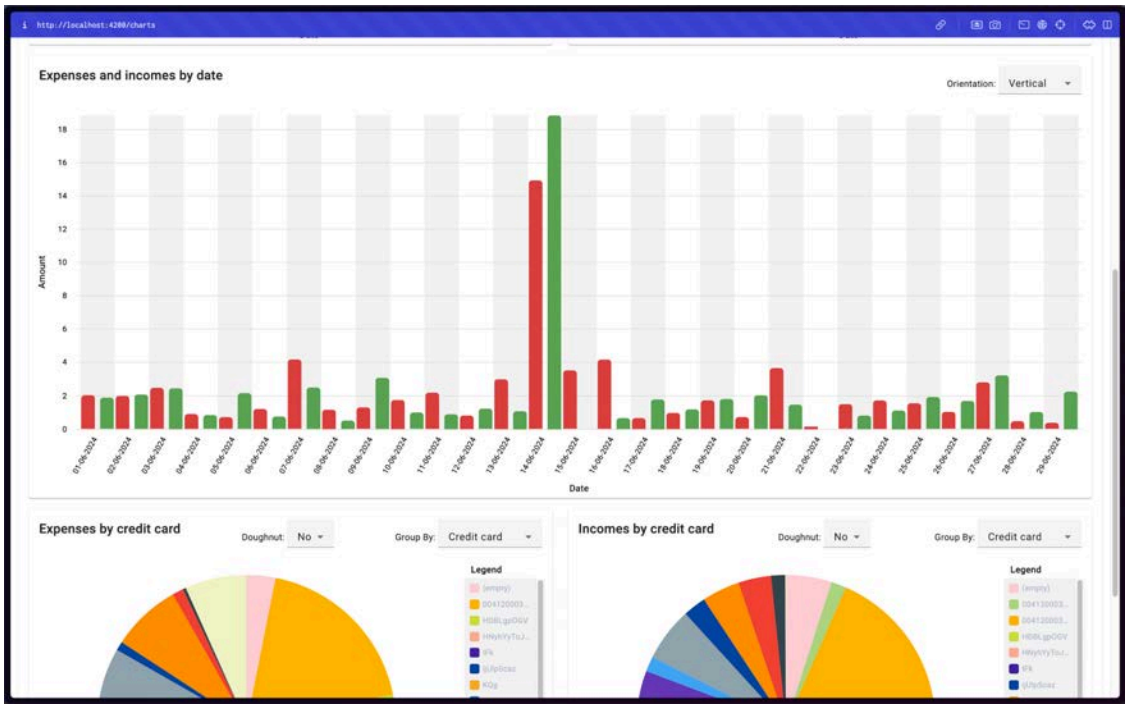



Expenses
Expense categories
Name
Creation date
encodedPasswordUserExpenseCategory
14-06-2024 18:54
toDeleteExpenseCategoryWithExpense
14-06-2024 18:54
toDeleteExpenseCategoryWithExpenseResource
14-06-2024 18:54
vfKiAOXLIx
15-03-2015 04:34
CqMzTaaPckVLwiZTBmVEwJ
11-11-2012 00:15
DxKdOlrfwNv
09-08-2013 13:25
KhwyPxfdkRFwGqbfDdCFyioozYS
22-07-2026 13:06
uNCIRvgRnWMNHCrkRpzghFPeTKBJUwS
06-11-2026 07:09
Close
Add new expense category
Name
Expense category name is required
Add
Cancel

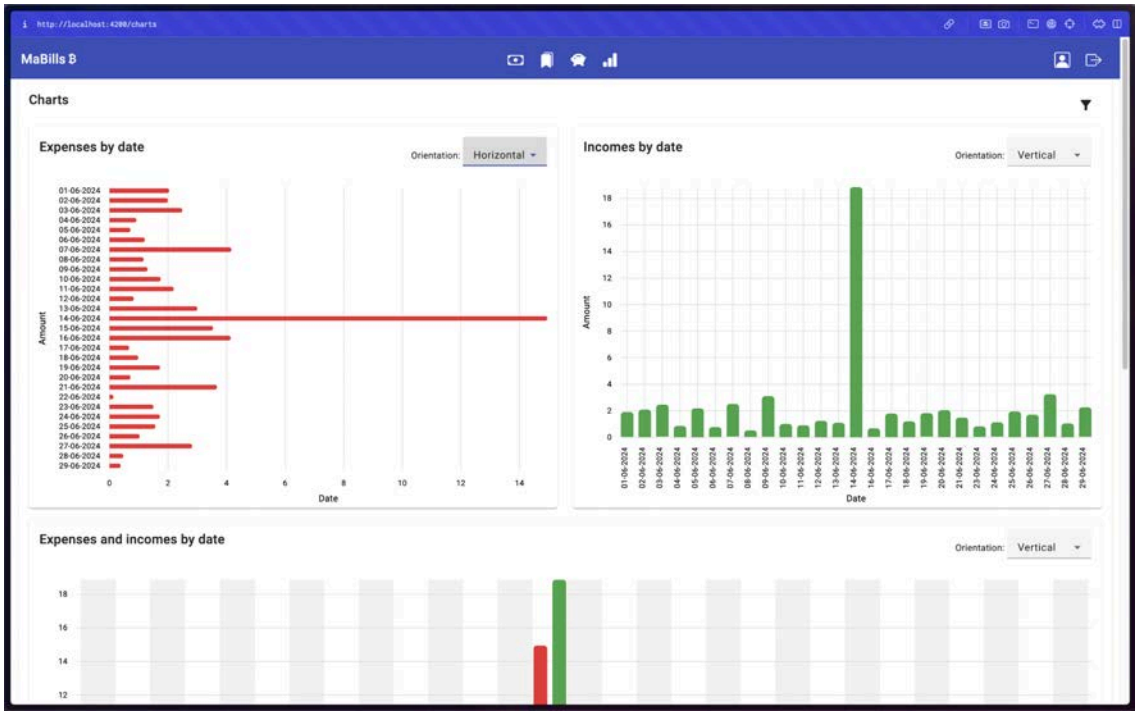

Expenses
Expense categories
Name
Creation date
encodedPasswordUserExpenseCategory
14-06-2024 18:54
toDeleteExpenseCategoryWithExpense
14-06-2024 18:54
toDeleteExpenseCategoryWithExpenseResource
14-06-2024 18:54
vfKiAOXLIx
15-03-2015 04:34
CqMzTaaPckVLwiZTBmVEwJ
11-11-2012 00:15
DxKdOlrfwNv
09-08-2013 13:25
KhwyPxfdkRFwGqbfDdCFyioozYS
22-07-2026 13:06
uNCIRvgRnWMNHCrkRpzghFPeTKBJUwS
06-11-2026 07:09
Close
Edit expense category
DNs
Save
Cancel

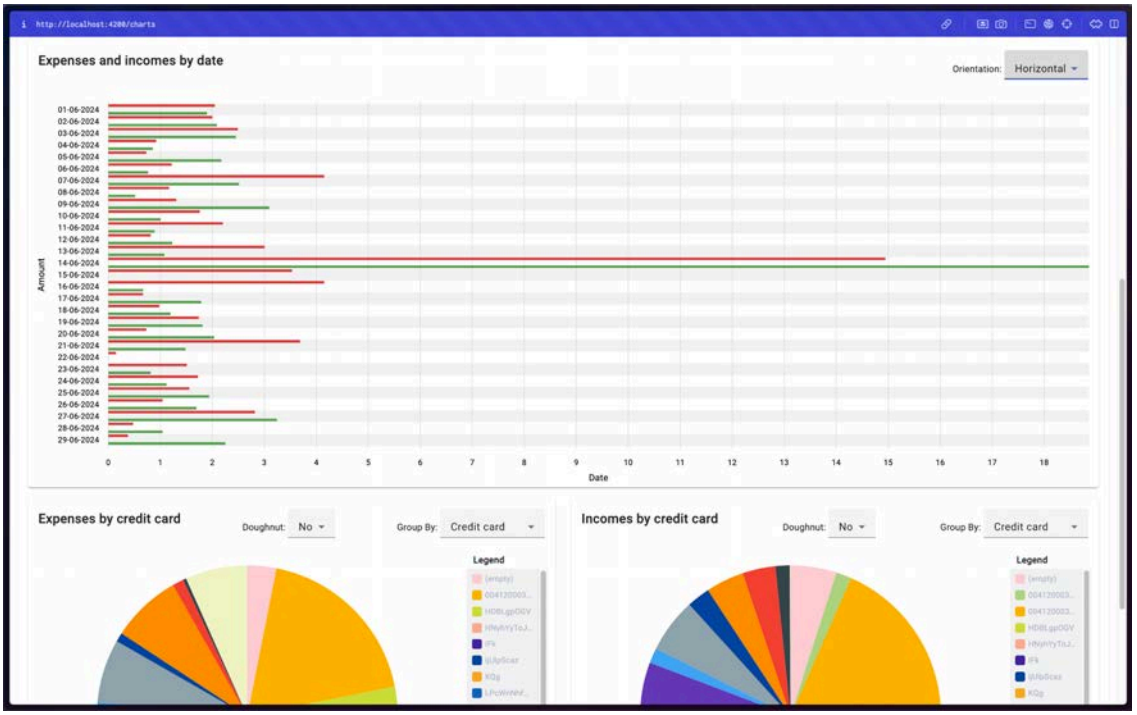
Charts

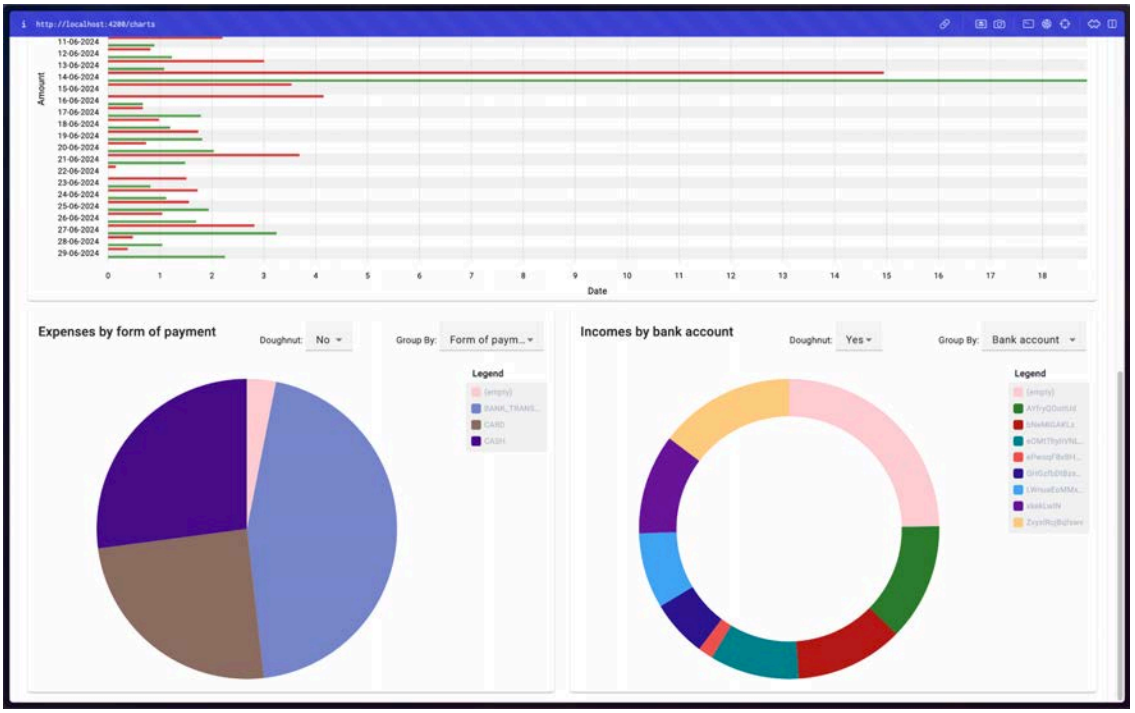

MaBills B
Charts
Expenses by date
Orientation: Vertical
Amount
Date
Incomes by date
Orientation: Vertical
Amount
Date
Expenses and incomes by date
Orientation: Vertical


Expenses and incomes by date
Orientation: Vertical
Amount
Date
Expenses by credit card
Doughnut: No
Group By: Credit card
Legend
Incomes by credit card
Doughnut: No
Group By: Credit card
Legend

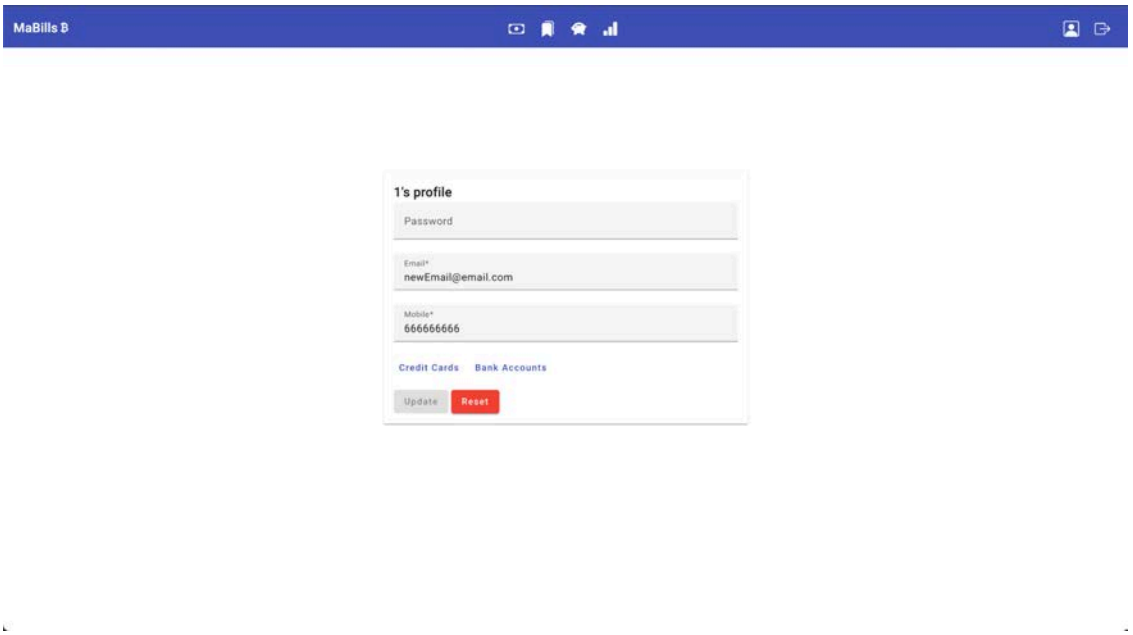

Amount
Date
Expenses by credit card
Doughnut: No
Group By: Credit card
Legend
Incomes by credit card
Doughnut: No
Group By: Credit card
Legend

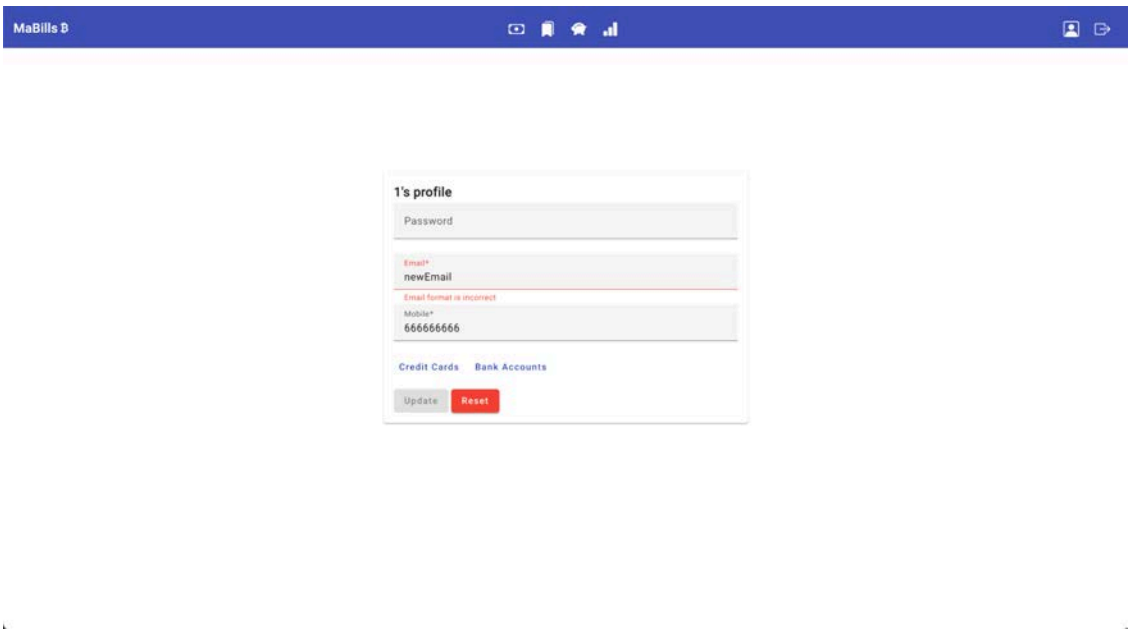

MaBills B
Charts
Expenses by date
Orientation: Horizontal
Amount
Date
Incomes by date
Orientation: Vertical
Amount
Date
Expenses and incomes by date
Orientation: Vertical


Expenses and incomes by date
Orientation: Horizontal
Amount
Date
Expenses by credit card
Doughnut: No
Group By: Credit card
Legend
Incomes by credit card
Doughnut: No
Group By: Credit card
Legend

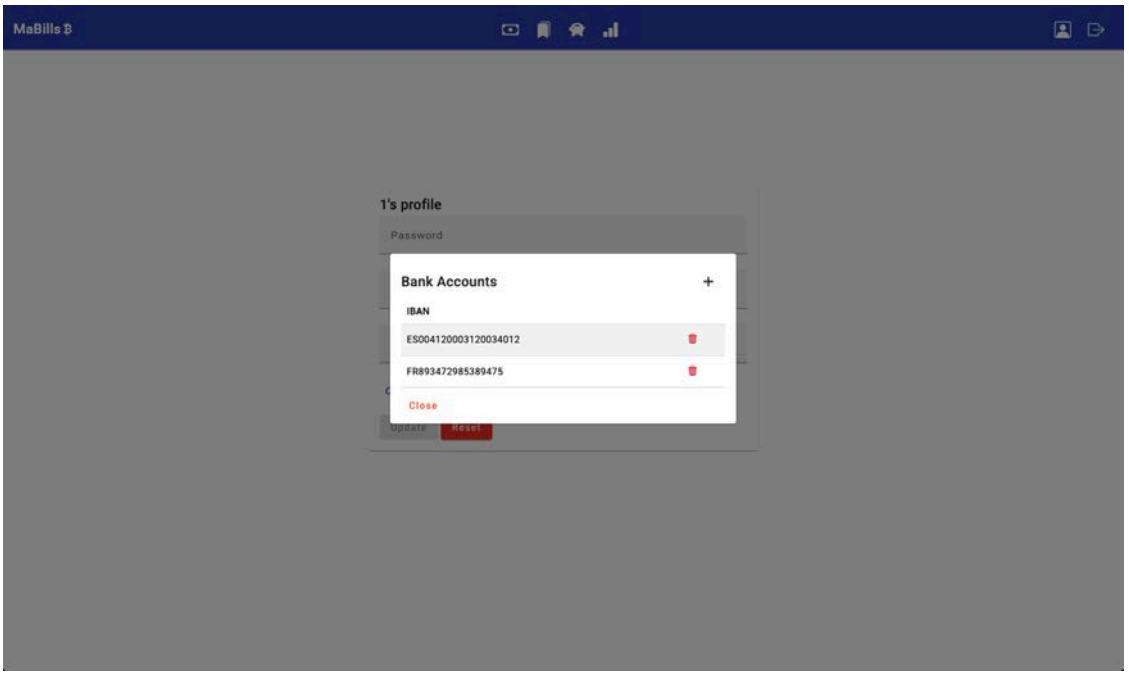

Amount
Date
Expenses by form of payment
Doughnut: No
Group By: Form of paym...
Legend
Incomes by bank account
Doughnut: Yes
Group By: Bank account
Legend

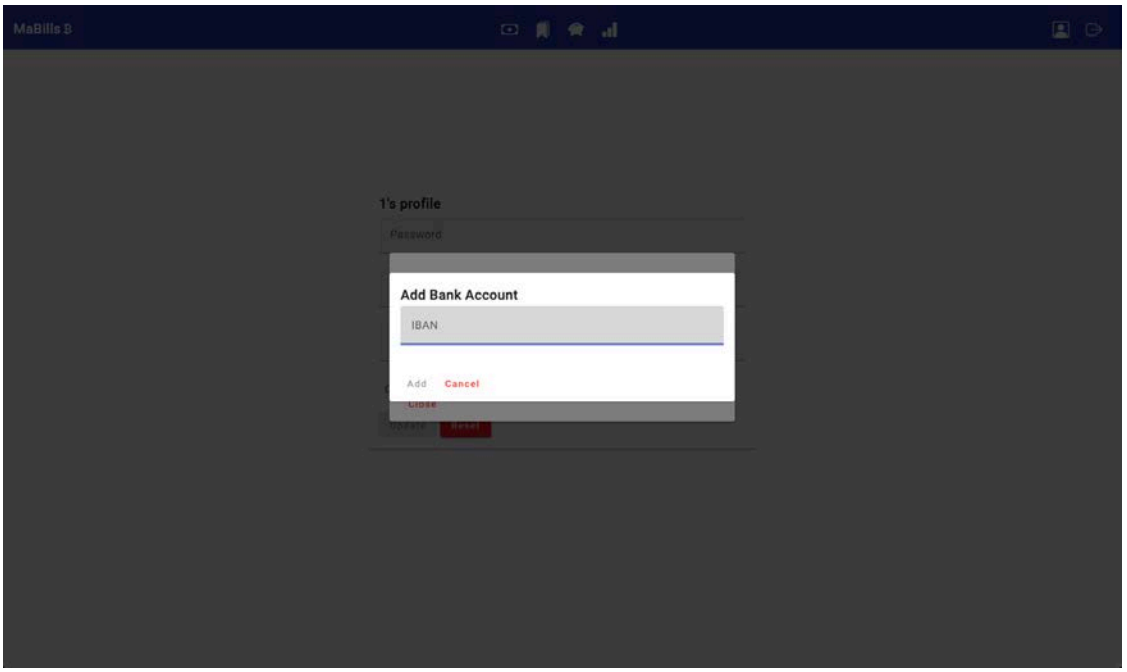
Profile

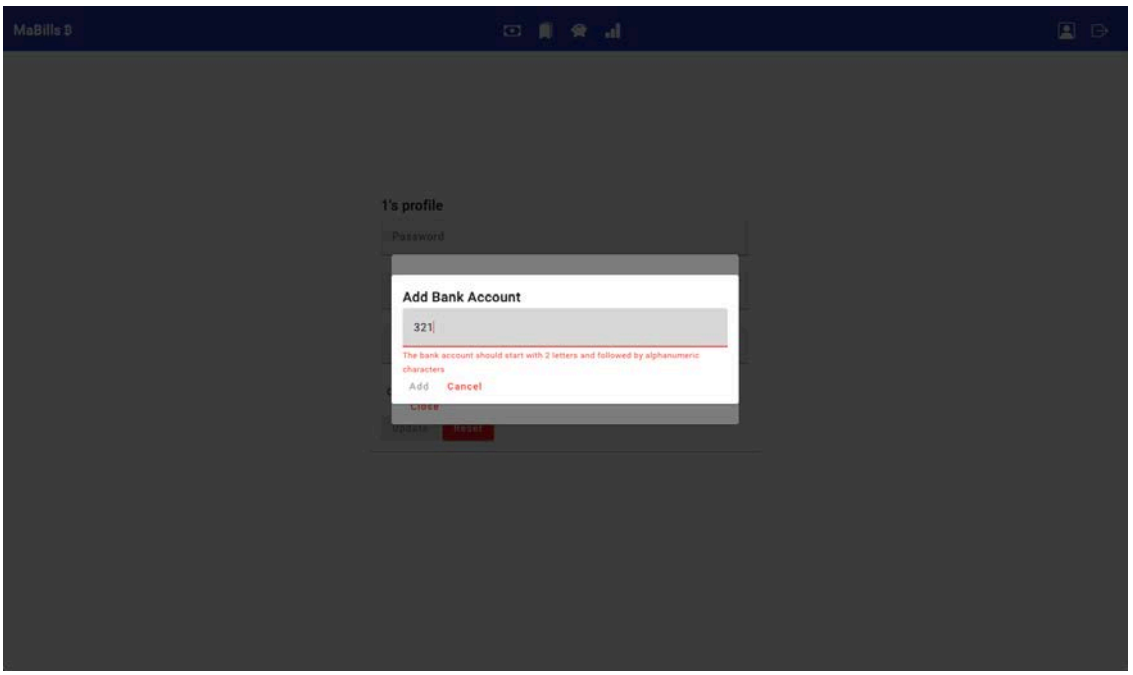

MaBills
1's profile
Password
Email*
newEmail@email.com
Mobile*
666666666
Credit Cards
Bank Accounts
Update
Reset


MaBills
1's profile
Password
Email*
newEmail
Email format is incorrect
Mobile*
666666666
Credit Cards
Bank Accounts
Update
Reset

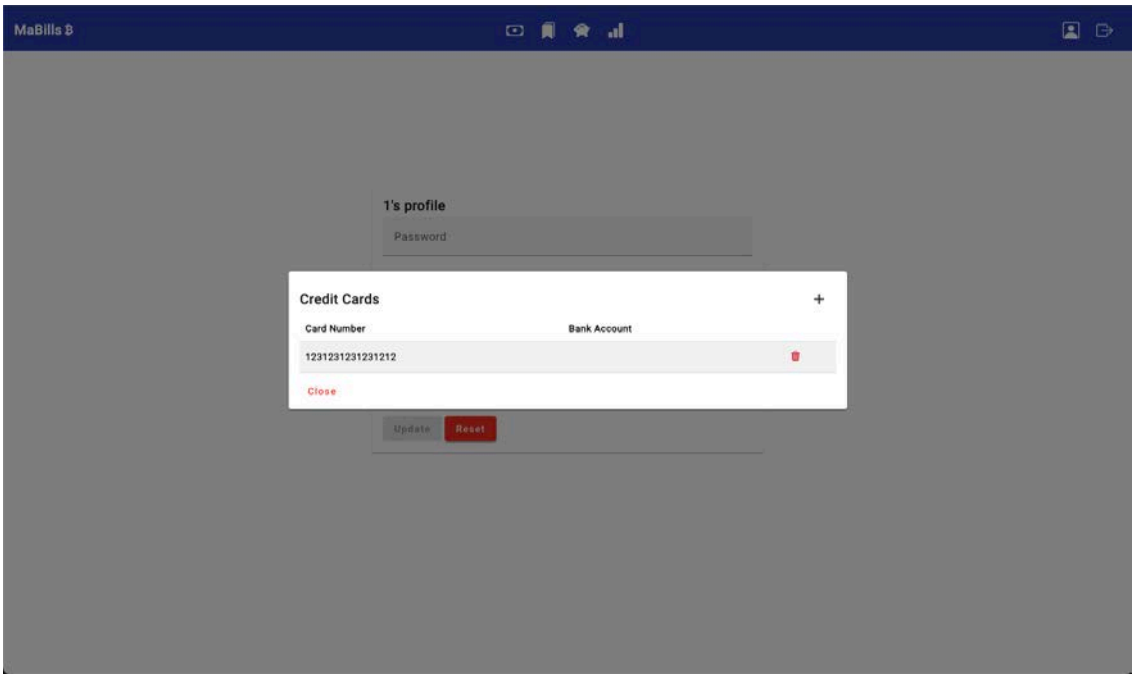
Bank accounts management

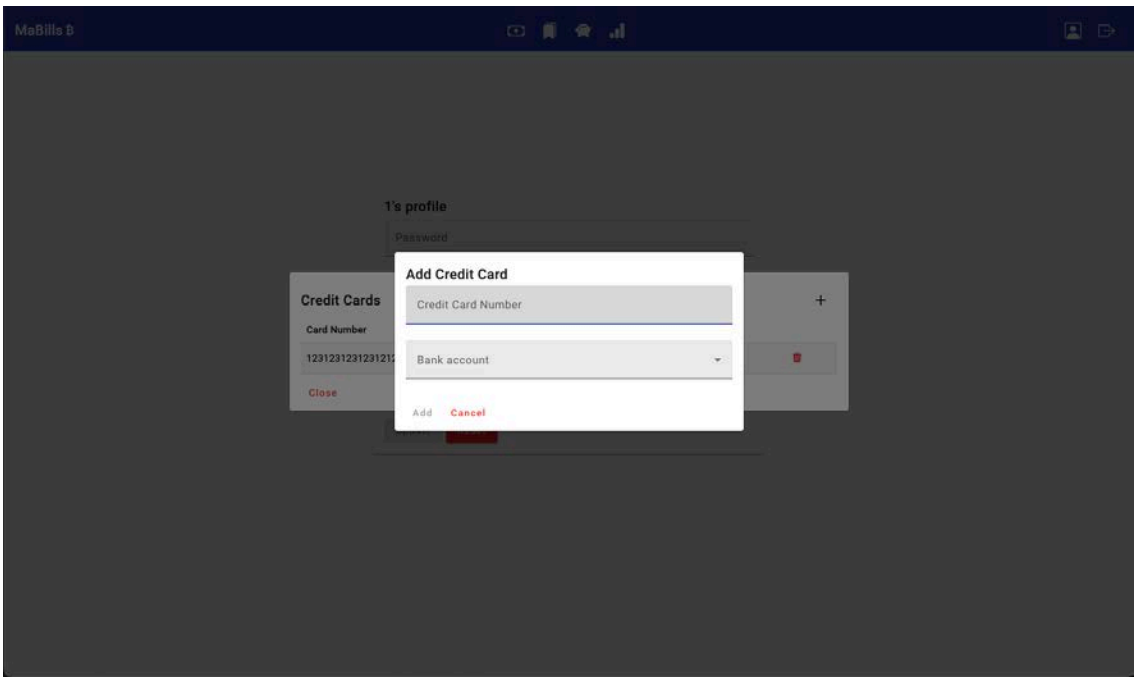


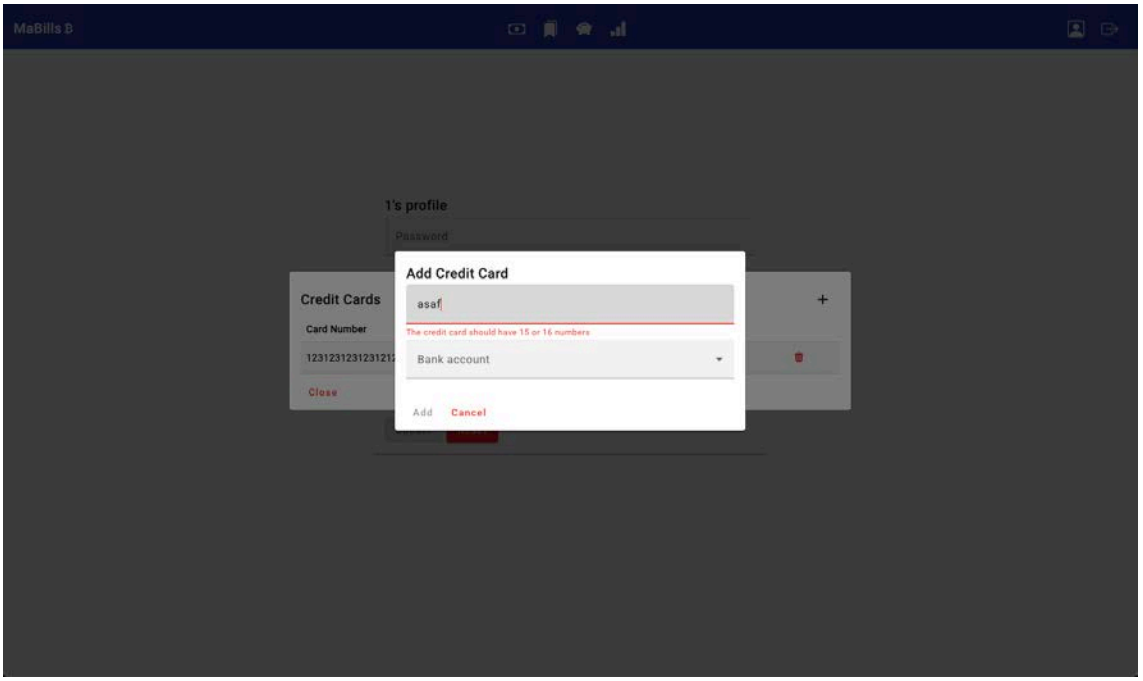




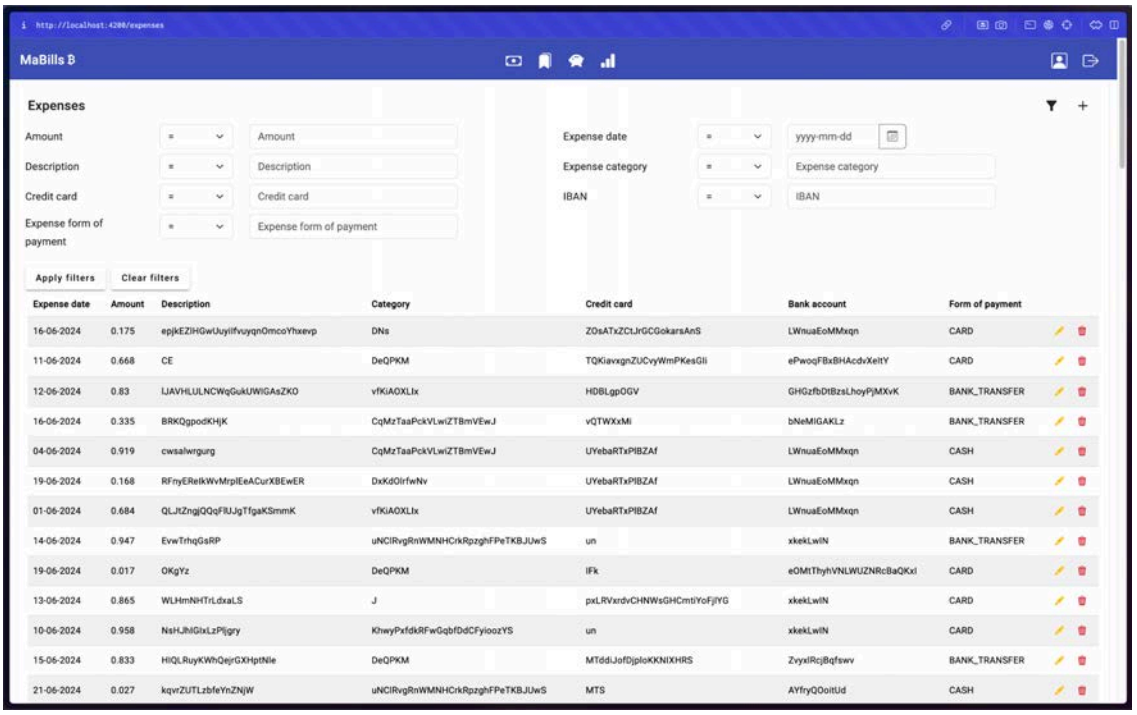
Credit cards management

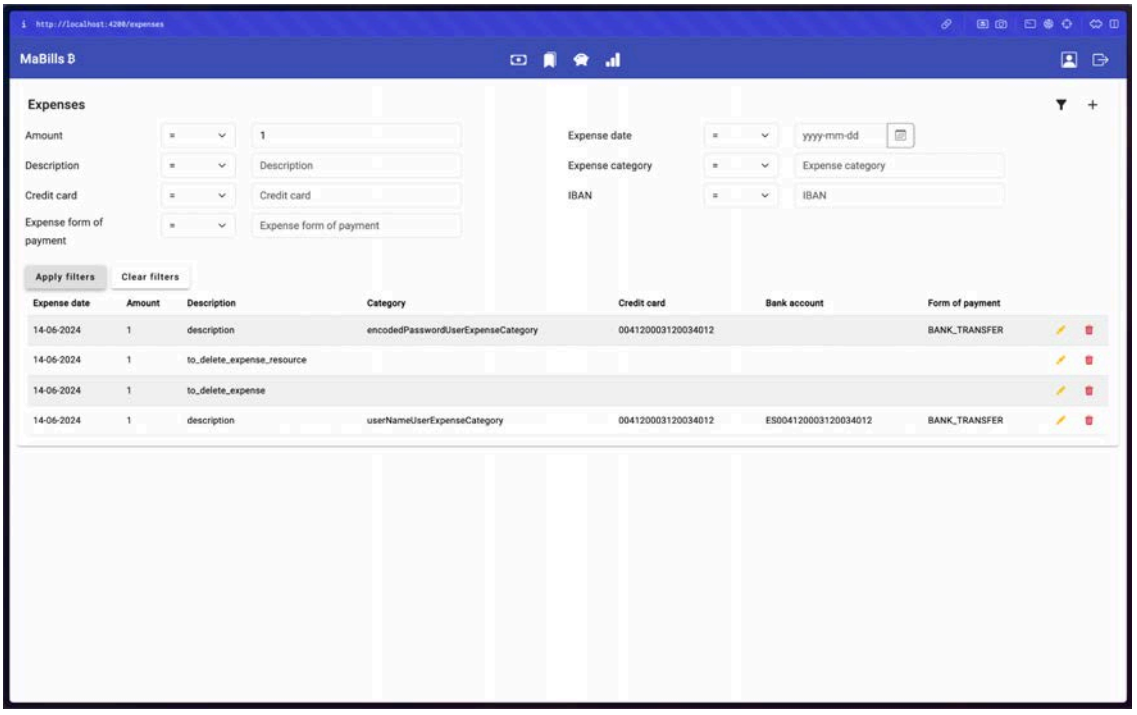


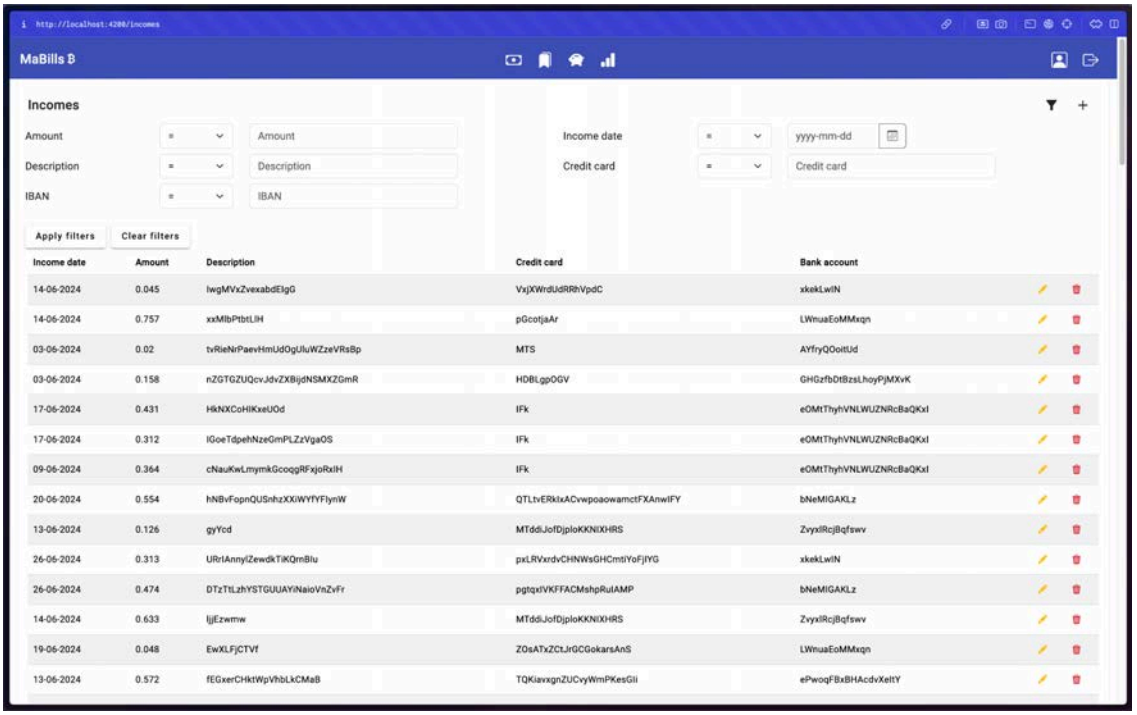

MaBills
1's profile
Password
Add Credit Card
Credit Card Number
Bank account
Add
Cancel
Credit Cards
Card Number
Close

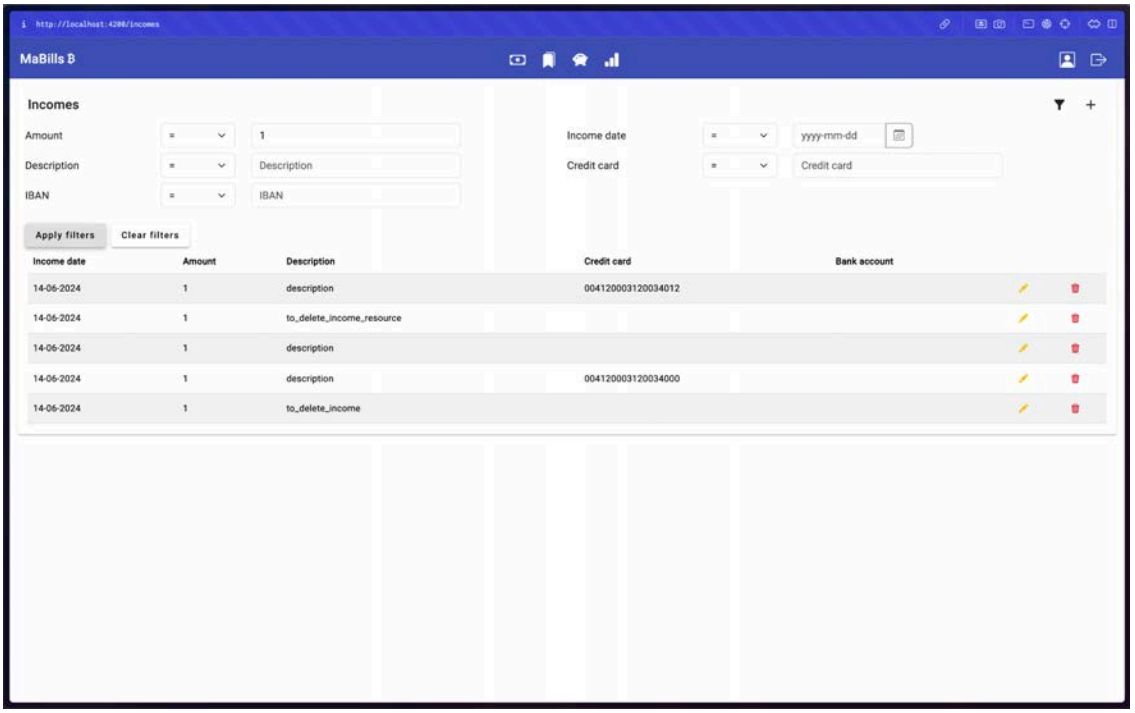

MaBills
1's profile
Password
Add Credit Card
asaf
The credit card should have 15 or 16 numbers
Bank account
Add
Cancel
Credit Cards
Card Number
Close

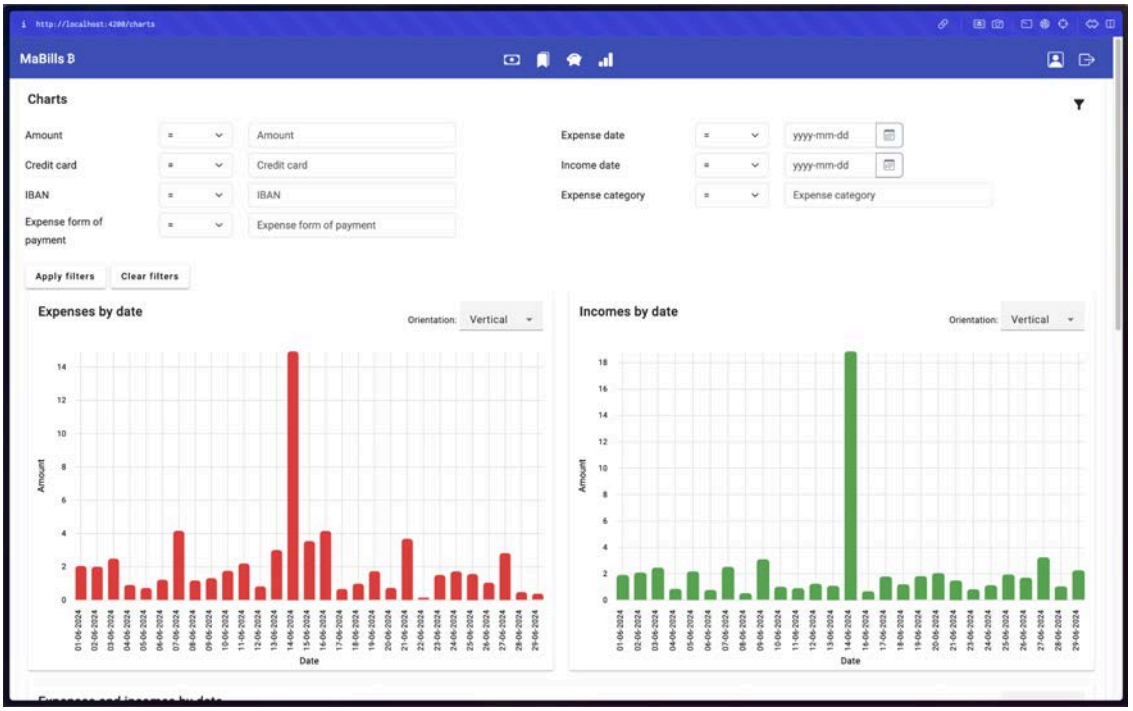
Filters

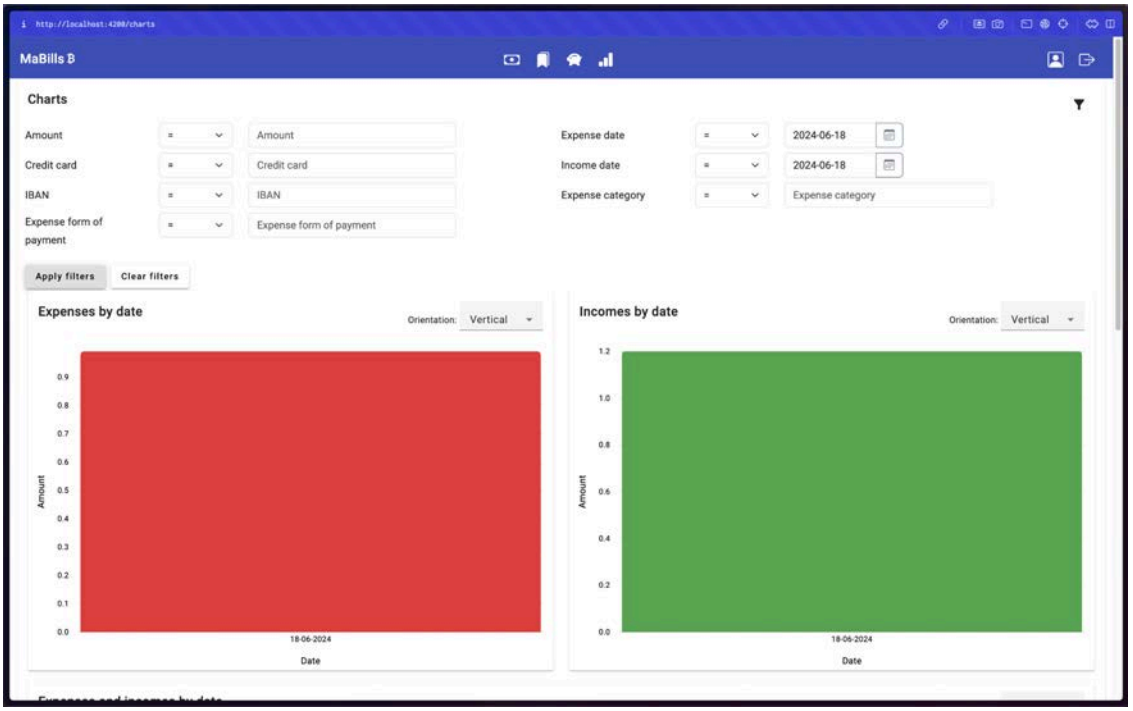

http://localhost:4200/expenses
MaBills B
Expenses
Amount
Description
Credit card
Expense form of payment
Expense date
Expense category
IBAN
yyyy-mm-dd
Apply filters
Clear filters
Expense date | Amount | Description | Category | Credit card | Bank account | Form of payment
16-06-2024 | 0.175 | epjkEZlHGwUuyiIfvuyqnOmcoYhxevp | DNs | ZOsATxZCtJrGCGokarsAnS | LWnuaEoMMxqn | CARD
11-06-2024 | 0.668 | CE | DeQPKM | TQKiavxgnZUCvyWmPKesGli | ePwoqFBxBHAcdvXeltY | CARD
12-06-2024 | 0.83 | IJAVHLULNCWqGukUWlGAsZKO | vfKiAOXLlx | HDBLgpOGV | GHGzfbDtBzsLhoyPjMXvK | BANK_TRANSFER
16-06-2024 | 0.335 | BRKQgpodKHjK | CqMzTaaPckVLwiZTBmVEwJ | vQTWXxMi | bNeMIGAKLz | BANK_TRANSFER
04-06-2024 | 0.919 | cwsalwrgurg | CqMzTaaPckVLwiZTBmVEwJ | UYebaRTxPIBZAf | LWnuaEoMMxqn | CASH
19-06-2024 | 0.168 | RFnyERelkWvMrpIEeACurXBEwER | DxKdOlrfwNv | UYebaRTxPIBZAf | LWnuaEoMMxqn | CASH
01-06-2024 | 0.684 | QLJtZngjQQqFlUJgTfgaKSmmK | vfKiAOXLlx | UYebaRTxPIBZAf | LWnuaEoMMxqn | CASH
14-06-2024 | 0.947 | EvwTrhqGsRP | uNClRvgRnWMNHCrkRpzghFPeTKBJUwS | un | xkekLwIN | BANK_TRANSFER
19-06-2024 | 0.017 | OKgYz | DeQPKM | IFk | eOMtThyhVNLWUZNRcBaQKxl | CARD
13-06-2024 | 0.865 | WLHmNHTrLdxaLS | J | pxLRVxrdvCHNWsGHCmtiYoFjlYG | xkekLwIN | CARD
10-06-2024 | 0.958 | NsHJlnGlxLzPljgry | KhwyPxfdkRFwGqbfDdCFyioozYS | un | xkekLwIN | CARD
15-06-2024 | 0.833 | HlQLRuyKWhQejrGXHptNle | DeQPKM | MTddiJofDjploKKNlXHRS | ZvyxlRcjBqfswv | BANK_TRANSFER
21-06-2024 | 0.027 | kqvrZUTLzbfeYnZNjW | uNClRvgRnWMNHCrkRpzghFPeTKBJUwS | MTS | AYfryQOoitUd | CASH

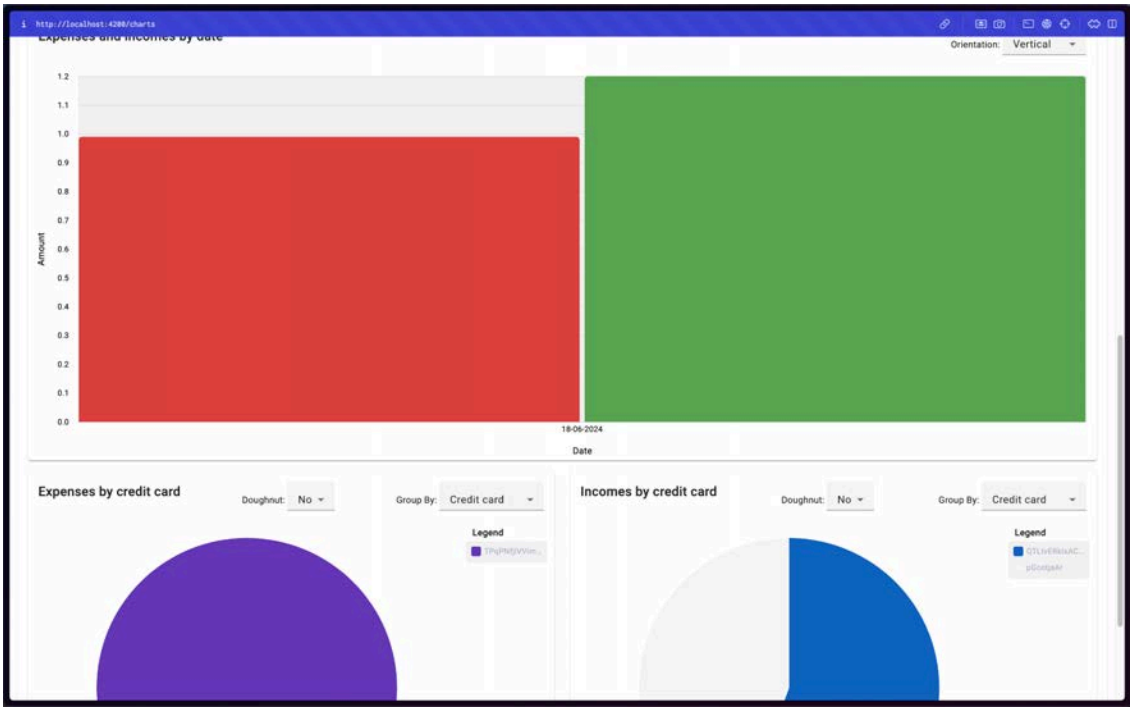

http://localhost:4200/expenses
MaBills B
Expenses
Amount
1
Description
Credit card
Expense form of payment
Expense date
Expense category
IBAN
yyyy-mm-dd
Apply filters
Clear filters
Expense date | Amount | Description | Category | Credit card | Bank account | Form of payment
14-06-2024 | 1 | description | encodedPasswordUserExpenseCategory | 004120003120034012 | | BANK_TRANSFER
14-06-2024 | 1 | to_delete_expense_resource
14-06-2024 | 1 | to_delete_expense
14-06-2024 | 1 | description | userNameUserExpenseCategory | 004120003120034012 | ES0041200031200340 12 | BANK_TRANSFER

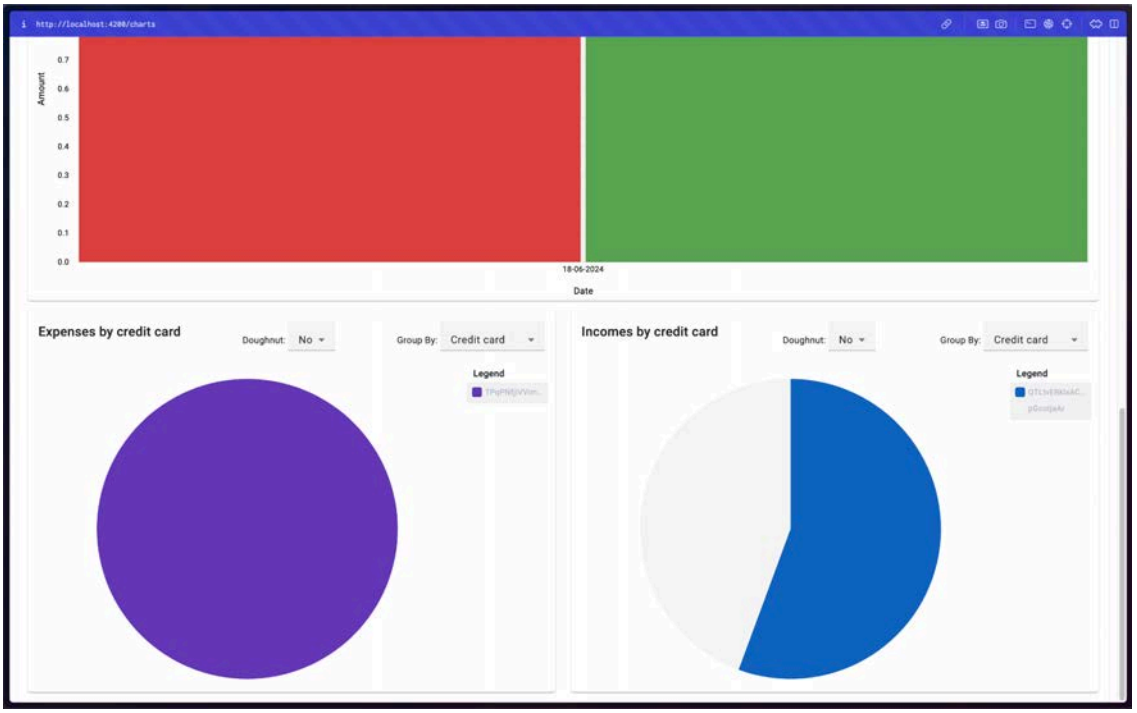

http://localhost:4200/incomes
MaBills B
Incomes
Amount
Description
IBAN
Income date
Credit card
yyyy-mm-dd
Apply filters
Clear filters
Income date
Amount
Description
Credit card
Bank account
14-06-2024 0.045 lwgMVxZvexabdElgG VxjXWrdUdRRhVpdC xkekLwIN
14-06-2024 0.757 xxMlbPtbtLIH pGcotjaAr LWnuaEoMMxqn
03-06-2024 0.02 tvRieNrPaevHmUdOgUluWZzeVRsBp MTS AYfryQOoitUd
03-06-2024 0.158 nZGTGZUQcvJdvZXBijdNSMXZGmR HDBLgpOGV GHGzfbDtBzsLhoyPjMXvK
17-06-2024 0.431 HkNXCoHIKxeUOd IFk eOMtThyhVNLWUZNRcBaQKxI
17-06-2024 0.312 IGoeTdpehNzeGmPLZzVgaOS IFk eOMtThyhVNLWUZNRcBaQKxI
09-06-2024 0.364 cNauKwLmymkGcoqgRFxjoRxIH IFk eOMtThyhVNLWUZNRcBaQKxI
20-06-2024 0.554 hNBvFopnQUSnhzXXiWYfYFlynW QTLtvERkIxACvwpoaowamctFXAnwIFY bNeMIGAKLz
13-06-2024 0.126 gyYcd MTddiJofDjploKKNiXHRS ZvyxlRcjBqfswv
26-06-2024 0.313 URrIAnnylZewdkTiKQrnBlu pxLRVxrdvCHNWsGHCmtiYoFjIYG xkekLwIN
26-06-2024 0.474 DTzTtLzhYSTGUUAYiNaioVnZvFr pgtqxfVKFFACMshpRulAMP bNeMIGAKLz
14-06-2024 0.633 IjjEzwmw MTddiJofDjploKKNiXHRS ZvyxlRcjBqfswv
19-06-2024 0.048 EwXLFjCTVf ZOsATxZCtJrGCGokarsAnS LWnuaEoMMxqn
13-06-2024 0.572 fEGxerCHktWpVhbLkCMaB TQKiavxgnZUCvyWmPKesGli ePwoqFBxBHAcdvXeltY


http://localhost:4200/incomes
MaBills B
Incomes
Amount
1
Description
IBAN
Income date
Credit card
yyyy-mm-dd
Apply filters
Clear filters
Income date
Amount
Description
Credit card
Bank account
14-06-2024 1 description 0041200031200340l2
14-06-2024 1 to_delete_income_resource
14-06-2024 1 description
14-06-2024 1 description 004120003120034000
14-06-2024 1 to_delete_income


MaBills B
Charts
Amount
Credit card
IBAN
Expense form of payment
Expense date
Income date
Expense category
yyyy-mm-dd
Apply filters
Clear filters
Expenses by date
Orientation: Vertical
Incomes by date
Orientation: Vertical
Amount
Date


MaBills B
Charts
Amount
Credit card
IBAN
Expense form of payment
Expense date
Income date
Expense category
2024-06-18
Apply filters
Clear filters
Expenses by date
Orientation: Vertical
Incomes by date
Orientation: Vertical
Amount
18-06-2024
Date





Como se pueden apreciar en las capturas de pantalla, los datos que aparecen en algunas capturas parecen ser incoherentes, esto es porque son creados directamente en la base de datos, pero tanto en la capa de presentación como en la capa de negocio, se comprueba siempre que los datos sean coherentes utilizando expresiones regulares para las cadenas de caracteres, valores positivos para los campos numérico (no tiene sentido un ingreso io un gasto negativo), y también comprobando que las fechas tengan un formato correcto.

CAPÍTULO 6. PRUEBAS

Como bien dijo **Kent Beck:** *"Un software sin pruebas, es un software defectuoso",* un software siempre debe tener pruebas ya que estas son una de las claves de la calidad del software.

Las pruebas en el software nos ayuda en diferentes aspectos, como por ejemplo:

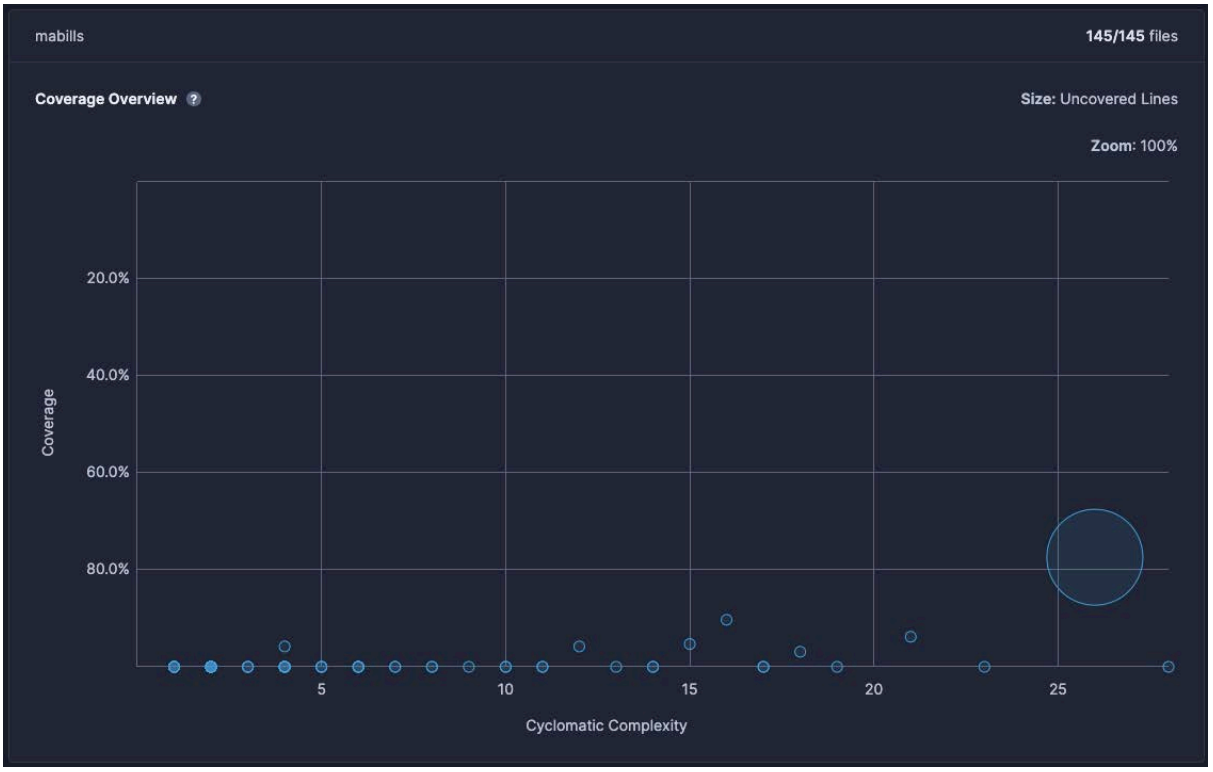
- Detección temprana de errores
- Aseguramiento de la calidad
- Facilitan el refactorización y mantenimiento
- Prevención de regresiones
- Mejorar el diseño del software

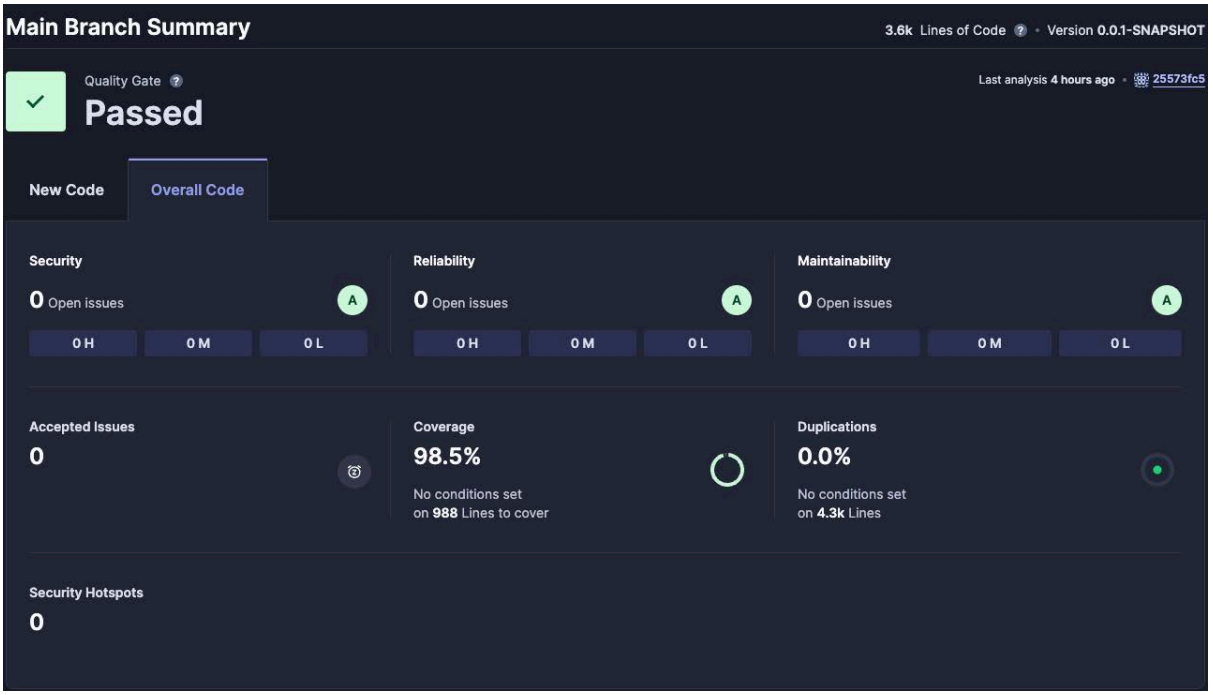
Es por eso que durante el desarrollo de este proyecto, se ha guiado siempre de la técnica *TFD* (Test First Development), escribiendo las pruebas en fases tempranas y desarrollando la aplicación en base a ellas. No se ha seguido *TDD* (Test Driven Development) por la falta de tiempo y el gran número de cambios de contextos que implica *TDD*.

Se han realizado pruebas de esta manera:

- En la capa del negocio, se ha realizado pruebas de distintos tipos:
    - Pruebas unitarias para asegurar el correcto funcionamiento de la lógica de negocio.
    - Pruebas de integración para asegurar que la comunicación entre componentes es correcta y además la integración con la base de datos sea correcta.
    - Pruebas end-to-end para asegurar que la capa de negocio funcione correctamente desde el inicio hasta el final.
- En la capa de presentación, no se ha realizado ningún tipo de pruebas automatizadas, ya que no se impartieron conocimientos sobre ellas durante el máster y al ser una interfaz sencilla, no se ha tenido la necesidad de tener pruebas automatizadas aquí.
- Se ha utilizado *JUnit5* y Mockito para las pruebas unitarias y de integración, y se ha utilizado *Spring-WebFlux* para realizar las pruebas end-to-end.
- Para medir la cobertura del código, se ha utilizado *SonarCloud*.

Utilizando la herramienta *SonarCloud,* se han obtenido las siguientes gráficas que mostrarán el nivel de cobertura del código.





Con estas dos gráficas se puede observar que la mayoría de las clases tienen una cobertura del 100% en el código con una complejidad ciclomática menor de 6 (se puede apreciar en la primera gráfica que los puntos azules se concentran en las zonas de menor complejidad ciclomática).

En la segunda gráfica, se puede observar que el proyecto ha alcanzado un **98,5%** de cobertura del código en todo el proyecto, esto asegura que la gran mayoría de las partes del código son probadas.

Con esta gráfica se puede apreciar que durante el desarrollo del proyecto, las líneas de códigos cubiertas siempre han estado muy cerca del número de líneas de código a cubrir, lo que indica que el proyecto siempre ha estado desarrollándose con la cantidad de pruebas necesarias.

Se puede apreciar en esta última gráfica que, el número de pruebas unitarias han ido creciendo a lo largo del tiempo, y no han fallado estas pruebas unitarias ni tampoco han tenido ningún error.

CAPÍTULO 7. GESTIÓN

Según RUP, un proyecto se divide en 4 fases, *Inicio, Elaboración, Construcción* y *Transición*. A continuación, se presenta una gráfica donde se puede apreciar la distribución del esfuerzo y duración entre las diferentes fases, y dentro de cada fase, entre las distintas disciplinas.

Distribución de Disciplinas por Fase		Absoluto				Relativo			
	Proyecto	Inicio	Elaboración	Construcción	Transición	Inicio	Elaboración	Construcción	Transición
Duración	100%	100%	100%	100%	100%	10%	30%	50%	10%
Esfuerzo	100%	100%	100%	100%	100%	5%	20%	65%	10%
Gestión	11%	14%	12%	10%	14%	0.7%	2.4%	6.5%	1.4%
Requisitos	11%	38%	18%	8%	4%	1.9%	3.6%	5.2%	0.4%
Análisis/Diseño	19%	19%	36%	16%	4%	1.0%	7.2%	10.4%	0.4%
Implementación	27%	8%	13%	34%	19%	0.4%	2.6%	22.1%	1.9%
Pruebas	20%	8%	10%	24%	24%	0.4%	2.0%	15.6%	2.4%
Despliegue	6%	3%	3%	3%	30%	0.2%	0.6%	2.0%	3.0%
Entorno/Versiones	6%	10%	8%	5%	5%	0.5%	1.6%	3.3%	0.5%

Este proyecto al no tener despliegue, la fase de *Transición* no se ha tenido en cuenta durante la gestión, por lo cual, la distribución del esfuerzo y duración es ligeramente diferente:

	Absoluto			Relativo		
	Inicio	Elaboración	Construcción	Inicio	Elaboración	Construcción
Duracion	100%	100%	100%	11%	34%	55%
Esfuerzo	100.00%	100%	100.00%	5.50%	22%	72.50%
Gestión	16.00%	14.00%	11%	0.88%	3.08%	8%
Requisitos	43%	20%	8.64%	2.37%	4.4%	6.0%
Análisis/Diseño	22.00%	40%	17.28%	1.21%	8.8%	12.5%
Implementación	9.50%	15.00%	36.72%	0.52%	3.3%	27%
Pruebas	9.50%	11.00%	25.92%	0.52%	2.42%	19%

Este proyecto tiene una duración total de 75 días, con una media de 4 horas por día y un solo alumno, el tiempo total dedicado al proyecto es de 300 horas. En el tiempo total, también está incluido el esfuerzo aplicado por parte del alumno en buscar documentación, investigar las diferentes tecnologías y solucionar errores del entorno al inicio del desarrollo, toda esta parte ha llevado un esfuerzo cerca de 80 horas.

A continuación, se presenta una tabla con el esfuerzo **estimado** (en horas) del alumno en cada fase y cada disciplina dentro de las fases, aplicando los porcentajes de la distribución del esfuerzo de RUP:

	Inicio	Elaboración	Construcción	Total
Gestión	2.6	9.2	24.0	35.9
Requisitos	7.1	13.2	18.0	38.3
Análisis/Diseño	3.6	26.4	37.5	67.5
Implementación	1.6	9.9	81.0	92.5
Pruebas	1.6	7.3	57.0	65.8
Total	16.5	66.0	217.5	300.0

Ahora, se presenta los datos reales sobre el esfuerzo realizado por el alumnado, al ser un proyecto de tamaño reducido, sólamente se ha realizado una iteración durante el desarrollo del proyecto:

Inicio

Gestión	0.5	Organizar la fase
Requisitos	2	Encontrar actores y casos de uso
	1	Priorizar casos de uso
	7	Especificar casos de uso: - Register - Login - Logout - ShowExpenses - ShowIncomes - ShowExpenseCategories
	3	Prototipar interfaz de usuario
Análisis/Diseño	2	Analizar arquitectura
Implementación	2	Crear los proyectos de la capa de presentación (*Angular*), capa de negocio (*Spring)* y capa de datos (*MySQL*)
Test	0	
Total	17.5	

Elaboración

Gestión	2	Organizar la fase
Requisitos	15	Especificar casos de uso: - ShowExpense - CreateExpense - UpdateExpense

		- DeleteExpense - ShowIncome
	4	Prototipar interfaz de usuario
Análisis/Diseño	3	Analizar arquitectura
	15	Analizar casos de uso
	8	Analizar clases
	3	Analizar paquetes
	8	Diseño de la arquitectura
	5	Diseño de casos de uso
Implementación	5	Implementar los casos de uso: - Register - Login - Logout
Pruebas	3	Diseñar e implementar pruebas para los casos de uso: - Register - Login - Logout Crear la integración continua con *GitHub Actions*
Total	71	

Construcción

Gestión	4	Organizar la fase
Requisitos	16	Especificar casos de uso: - ShowCharts - UpdateProfile - CreateIncome - UpdateIncome - DeleteIncome - ApplyFilters - ShowOptions - CreateExpenseCategory - UpdateExpenseCategory - DeleteExpenseCategory
Análisis/Diseño	3	Analizar arquitectura
	20.5	Analizar casos de uso
	2	Analizar clases
	1	Analizar paquetes

	3	Diseño de la arquitectura
	2	Diseño de casos de uso
Implementación	100	Implementar los casos de uso: - ShowExpenses - ShowExpense - CreateExpense - UpdateExpense - DeleteExpense - ShowIncomes - ShowIncome - CreateIncome - UpdateIncome - DeleteIncome - ApplyFilters - ShowOptions - ShowExpenseCategories - CreateExpenseCategory - UpdateExpenseCategory - DeleteExpenseCategory - ShowCharts - UpdateProfile
Pruebas	60	Diseñar e implementar pruebas para los casos de uso: - ShowExpenses - ShowExpense - CreateExpense - UpdateExpense - DeleteExpense - ShowIncomes - ShowIncome - CreateIncome - UpdateIncome - DeleteIncome - ApplyFilters - ShowOptions - ShowExpenseCategories - CreateExpenseCategory - UpdateExpenseCategory - DeleteExpenseCategory - ShowCharts - UpdateProfile
Total	211.5	

	Inicio	Elaboración	Construcción	Total
Gestión	0.5	2.0	4.0	6.5
Requisitos	13.0	19.0	16.0	48.0
Análisis/Diseño	2.0	42.0	31.5	75.5
Implementación	2.0	5.0	100.0	107.0
Pruebas	0.0	3.0	60.0	63.0
Total	17.5	71.0	211.5	300.0

Como se puede observar, el esfuerzo **real** varía en comparación con el esfuerzo **estimado.** Además, la variación es mayor en las fases iniciales, sobre todo en las disciplinas de **requisitos, análisis y diseño** debido a la falta de experiencia en RUP.

CAPÍTULO 8. CONCLUSIÓN Y DESARROLLOS FUTUROS

1. Conclusión

El principal objetivo de la realización de este proyecto es aplicar los conocimientos adquiridos durante el máster y aplicarlos en un desarrollo real, se ha alcanzado este objetivo exitosamente ya que durante el desarrollo, se ha podido aplicar varios conocimientos como:

1. RUP: se ha mostrado que se ha aplicado la metodología RUP durante todo el desarrollo del proyecto, siguiendo las disciplinas en su orden y repartiendo correctamente el tiempo total entre las diferentes disciplinas. Gracias a la aplicación de esta metodología, se ha podido disminuir el tiempo de implementación, aumentar la calidad del software y además, aumentar la capacidad del alumnado sobre las estimaciones de tiempo.
2. Calidad del software: como bien se muestra en el capítulo 5 y 6, la calidad del código tanto a niveles estáticos como a niveles de pruebas, han sido de alta calidad, calificado con una nota de ***A*** según SonarCloud.
3. *Spring:* se ha desarrollado la capa de negocio con el framework *Spring.* Este framework es ideal para desarrollar un api REST con Java, ya que proporciona interfaces de alto nivel y el usuario puede despreocuparse de los detalles a bajo nivel.
4. *Angular:* la capa de presentación está realizada con *Angular*.

Además de alcanzar el objetivo principal, gracias a la realización de este proyecto, el alumno ha podido fortalecer sus conocimientos en las tecnologías como *Spring, Angular, GitHub Actions, CI/CD* y también en *Docker.*

2. Desarrollos futuros

Debido a que el tiempo de la realización del proyecto es limitado, se propone como desarrollos futuros estos puntos:

1. Integrar un sistema de notificaciones *push,* con la intención de mostrar al usuario alertas a tiempo real de los gastos (por ejemplo, ha superado el límite de gastos mensuales), o avisarle de algún posibles futuros gastos.
2. Integrar un modelo de inteligencia artificial para que la aplicación pueda predecir ciertos gastos futuros del usuario y enviar avisos al usuario (utilizando el punto anterior) sobre dichos gastos.
3. Implementar un role *Admin,* con sus casos de uso correspondientes, para que realice la función de administrar los usuarios, un caso de uso podría ser, dar de bajo a usuarios que han estado inactivos durante un largo periodo de tiempo.
4. Mejorar la interfaz gráfica.
5. Añadir pruebas en la capa de presentación.

BIBLIOGRAFÍA

[1] *Spring [Website]*, https://spring.io/. Último acceso 12 Junio 2024.

[2] *Refactoring Guru [Website]*, https://refactoring.guru/. Último acceso 30 Mayo 2024.

[3] *Angular v16 [Website]*, https://v16.angular.io/docs. Último acceso 16 Junio 2024.

[4] *Angular Material UI component library [Website]*, https://v16.material.angular.io/. Último acceso 15 Junio 2024.

[5] *Angular powered Bootstrap [Website]*, https://ng-bootstrap.github.io/#/home. Último acceso 16 Junio 2024.

[6] *Bootstrap Icons - Official open source SVG icon library for Bootstrap [Website]*, https://icons.getbootstrap.com/. Último acceso 15 Mayo 2024.

[7] *Baeldung [Website]*, https://www.baeldung.com/. Último acceso 14 Junio 2024.

[8] *Medium – Where good ideas find you.*, https://medium.com/. Último acceso 12 Mayo 2024.

[9] *ngx-charts: Introduction [Website]*, https://swimlane.gitbooks.io/ngx-charts/content/. Último acceso 12 Junio 2024.

[10] *GitHub Actions [Website]*, https://docs.github.com/es/actions. Último acceso 12 June 2024.

[11] *Stack Overflow - Where Developers Learn, Share, & Build Careers*, https://stackoverflow.com/. Accessed 15 June 2024.

[12] Kruchten, Philippe. *The Rational Unified Process: An Introduction*. Addison-Wesley, 2004.

[13] Martin, Robert C. *Clean Code: A Handbook of Agile Software Craftsmanship*. Edited by Robert C. Martin, Prentice Hall, 2009.

[14] Beck, Kent. *Test Driven Development: By Example*. Pearson Education, 2022.

[15] Fowler, Martin, and Kent Beck. *Refactoring*. Addison-Wesley, 1999.

[16] Fowler, Martin. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Pearson, 2010

[17] Brown, William J. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. Wiley, 1998.

[18] Martin, Robert C. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall, 2018.

[19] Kroll, Per, and Philippe Kruchten. *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*. Addison-Wesley, 2003.