Grado en Ingeniería Informática

Trabajo de Final de Grado

Sistema de Business Intelligence para monedero electrónico

Autor:
Víctor Manuel Coral Narváez

Supervisor:
Fernando Gregori Pérez
Tutor académico:
Ricardo Chalmeta Rosaleñ

Fecha de lectura: 13 de Julio de 2022
Curso académico 2021/2022

Resumen

En el presente documento se presenta la memoria del trabajo final de grado, donde se explica en detalle el desarrollo de un sistema de *Business Intelligence* para la empresa Paynopain. El propósito de dicho sistema es el de facilitar una herramienta de análisis y visualización interactiva de datos estadísticos para el departamento de soporte y análisis de la empresa.

El desarrollo de este sistema consta de una primera parte de *backend* realizada en *Java* con varias librerías para optimizar la escritura de código. La segunda parte del sistema, el *frontend*, se realiza con Angular, un *framework* basado en TypeScript

A lo largo de este documento se explicará la planificación inicial, el desglose de las tareas realizadas y sus fases. Además se ejemplificará el funcionamiento con un vídeo al que se podrá acceder desde el anexo A de este documento.

Palabras clave

Business Intelligence, Angular, TypeScript, Java, *dashboard*, análisis estadístico

Keywords

Business Intelligence, Angular, TypeScript, Java, *dashboard*, statistic analysis

Índice general

Índice de cuadros

Índice de figuras

Índice de extractos de código

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

Paynopain Solutions S.L., es una empresa española con sede en Castellón de la Plana, especializada en el *fintech*, es decir en la tecnología financiera. Paynopain (PNP) se dedica a la investigación y el desarrollo tecnológico especializado en medios de pago. En concreto se compone de tres equipos de desarrollo diferenciados: ChangeIt, PayLands y FintechLabs. Dichos equipos trabajan en monedero electrónico, pasarelas de pago y soluciones digitales a medida, respectivamente.

El proyecto de Sistema de *Business Intelligence* surge a raíz de la necesidad de crear una vista rápida e interactiva de los datos estadísticos dentro del panel de soporte y análisis de la compañía, para poder manejar conscientemente la llegada masiva de datos de nuevos clientes cada vez maś grandes en los últimos años. Desde dicho panel, el equipo de soporte es capaz de añadir, consultar y gestionar las cuentas de los clientes, además de obtener datos relevantes de análisis que se usarán para desarrollar el sistema y darle una utilidad más tangible. Por otro lado, desde otra vista más limitada del panel los clientes pueden ver sus cuentas y consultar los datos.

1.2. Objetivos del proyecto

El objetivo principal de este proyecto es crear un sistema de *Business Intelligence* para monedero electrónico, ofreciendo así una herramienta de análisis para los administradores, los clientes y el equipo de soporte , dando uso de varias herramientas de desarrollo. Este objetivo se puede subdividir en dos principales tipos de objetivos más específicos como se definen a continuación:

Objetivos tecnológicos

- Construir nuevos casos de uso en la implementación del software de monedero electróni-

co, para así obtener los datos estadísticos que se usaran en implementaciones *Business Intelligence* de PNP.

- Analizar los casos de uso estratégicos con el fin de mejorar su eficiencia y definir su utilidad, además de añadir filtros para ampliar su funcionalidad.
- A través de utilidades *frontend* formar una herramienta visual en el que se retraten los análisis, desarrollados previamente, de forma ordenada y conveniente.
- Implementar la herramienta visual mencionada anteriormente en el panel estadístico de PNP.

Objetivos empresariales

- Dar utilidad a los datos recibidos y procesados para mejorar el rendimiento empresarial.
- Analizar de un vistazo el estado actual de la compañía.
- Obtener predicciones fácilmente en base a lo observado en las gráficas.
- Desarrollar un plan de actuación con los análisis recabados.

1.3. Descripción del proyecto

1.3.1. Descripción

El *Business Intelligence* o la inteligencia de negocios son un conjunto de teorías, metodologías, arquitecturas y tecnologías que transforman datos sin procesar en información significativa y útil para fines comerciales[1]

El proyecto que se documenta a continuación es en esencia el proceso de creación de un sistema de *Business Intelligence* diseñado particularmente para la empresa Paynopain. Se elaborará un diseño previo del sistema, se valorarán diferentes maneras de implementarlo y finalmente se construirá en distintas fases, usando diferentes herramientas. Dicho sistema mostrará el análisis de los datos de interés del software de monedero electrónico, del equipo ChangeIt de PNP, es decir se visualizarán funciones analíticas construidas a partir de casos de uso específicos del software.

1.3.2. Alcance

En cuanto al alcance del sistema podemos enfocarlo desde tres perspectivas distintas. Primeramente desde el punto de vista **funcional** que se divide en:

- Un desarrollo *backend* de nuevos casos de uso reutilizables y específicos para el software de la empresa, que realizan una conexión a la base de datos mediante consultas para obtener los datos estadísticos de relevancia.
- Una interfaz gráfica interactiva, para que el departamento de soporte y análisis de la empresa pueda desplegar y observar datos estadísticos. Así como clientes que darán uso a la interfaz con funciones más limitadas.
- Una implementación embebida de la interfaz gráfica, o de uno de sus desplegables, en diferentes páginas web internas de la empresa y páginas específicas de clientes.

Respecto al alcance **organizativo**, dentro de la organización interna del proyecto se pueden identificar tres grupos de acción:

- El equipo de desarrollo del sistema *Business Intelligence* (BI), que se encarga de su ideación hasta su desarrollo final.
- El equipo de ChangeIt, encargado de gestionar, supervisar y solucionar los problemas recurrentes en el proyecto.
- El equipo de soporte y análisis de la empresa que dará uso de dicha herramienta analítica.

Finalmente, el alcance **informático** ha de permitir la conexión con la base de datos en mysql a través de varias librerías, además de facilitar la construcción del *backend* con java y del *frontend* a través de Angular basado en TypeScript. En esta sección se puede aportar información adicional sobre el proyecto y las tecnologías que se van a usar en el proyecto, así como por ejemplo cual es la situación inicial a partir de la cual se va a desarrollar el proyecto o la mejora que se espera conseguir.

1.3.3. Riesgos

Para el desarrollo del proyecto se han de documentar los riesgos que puedan afectar a su correcta ejecución, y así evitarlos o disminuir su impacto.

Existen varios riesgos en el proyecto, en concreto relacionados con la inestabilidad de la base de datos, ya que el sistema ha de crearse a través de casos de uso en el Core del software de monedero electrónico de ChangeIt, por lo que si un nuevo cliente de la empresa realiza muchas peticiones puede sobrecargar el sistema, quedando su base de datos saturada durante minutos.

Ya que, como se ha mencionado anteriormente, este sistema se desarrolla en el mismo software en el que está trabajando todo el equipo de ChangeIt, cualquier error humano en el código afectaría al desarrollo de nuestro proyecto. Aunque gracias al control de versiones esto es más o menos evitable.

Por otro lado, también podemos tener en cuenta como riesgo las exigencias del equipo de soporte y análisis.

1.4. Estructura de la memoria

Esta memoria se estructura en cinco capítulos diferenciados. En el segundo capítulo , se expondrá la planificación del proyecto, esto es la metodología, la planificación, la estimación de recursos y costes y finalmente el seguimiento del proyecto. En el tercer capítulo, se detallará con precisión un completo análisis del sistema, empezando por las condiciones de partida, el diseño del sistema, el tratamiento de los datos, los casos de uso creados y el diseño final. En el cuarto capítulo, se profundiza en el propio desarrollo del proyecto. Se detalla la implementación así como los problemas que han surgido, las soluciones, las pruebas llevadas a cabo y por último la verificación y validación con los usuarios finales. En el último capítulo, el quinto, se expone una conclusión personal que cierra esta memoria, valorando lo aprendido en el proyecto y la utilidad del *Business Intelligence*

Capítulo 2

Planificación del proyecto

2.1. Metodología

La metodología que se va a utilizar para la planificación del proyecto será ***Scrum***. En un principio se iba a realizar con **PMBOK** pero ya que el desarrollo en la propia empresa se trabaja con procesos ágiles lo más adecuado era seguir con la estrategia del equipo.

Scrum es es un marco de trabajo que define un conjunto de eventos, prácticas y roles. Que puede tomarse como conjunto base para definir el proceso de producción que usará un equipo de trabajo dentro de un proyecto. [2]

La elección de esta metodología se basa principalmente en que el desarrollo de este sistema está sujeto continuamente a cambios, por lo que una definición de requisitos funcionales desde el principio daría pie a demasiadas modificaciones impidiendo el avance del proyecto correctamente. Por otro lado y como se ha mencionado anteriormente, la manera de trabajar del equipo se basa principalmente en teorías y procesos ágiles, lo que justifica la preferencia de esta metodología que es más flexible y adaptable frente a cualquier otra.

Esta metodología en PNP se combina con su sistema de implementación hexagonal, en el que la implementación del software y la división del trabajo se divide en seis partes diferenciadas, todas ellas comunicadas entre sí.

Para empezar a usar *Scrum* se deben crear las historias de usuario junto a las tareas a realizar, esto se conoce como el *product backlog*. Para definir correctamente el desarrollo del proyecto se han de ordenar dichas tareas por prioridad. Tras priorizar las tareas se obtienen los Sprints, que representan la ejecución de las tareas en un periodo de tiempo. Los Sprints son secuenciales y en este caso son todos de una duración de dos semanas. Los Sprints también se han de organizar y definir correctamente, por ello se realizan reuniones por cada Sprint, al inicio, durante y al final del mismo.

Para comprender esta metodología es necesario identificar los siguientes roles principales:

- *Product Owner*, dirige y controla el proceso de desarrollo, que se trabaje correctamente desde un perspectiva empresarial, y a su vez de ayuda a priorizar los elementos del *product backlog* que se quieren completar.
- *Scrum Master*, organiza el equipo y se asegura que este cumpla todos los procedimientos dentro de la metodología *Scrum*, ayudando a superar los obstáculos o problemas que surjan.
- *Scrum Team*, son los trabajadores o desarrolladores que realizan la implementación del proyecto.

Durante la estancia de prácticas en las que se desarrolla este proyecto, los roles fueron asignados como sigue:

El *Product Owner* fue asignado a dos personas, el CTO y el CEO de la empresa. Aunque quién ejerció más este rol fue el CTO quién a su vez es el supervisor de las prácticas. En el caso del *Scrum Master*, el rol también se dividió en dos personas, una durante la realización del *backend* y otra para el *frontend*. Estas dos personas fueron el supervisor y un integrante del equipo ChangeIt, respectivamente. Finalmente el rol de *Scrum Team* se designó al estudiante de prácticas.

En relación a las reuniones que se realizan para concretar los Sprints, se distinguen de la siguiente manera:

- *Sprint planning*, las reuniones de planificación se realizan antes de comenzar cada Sprint, en estas reuniones asistía el *Product Owner* y el estudiante en prácticas.
- *Daily meeting*, estas son las reuniones diarias durante el desarrollo del Sprint. Estas reuniones durante la estadía se realizaban a través de Discord, una plataforma de comuniación online, ya que eran breves y consistía a detallar las tareas del día para continuar con el Sprint.
- *Sprint review*, las reuniones de cierre se realizaban al terminar con un Sprint, estas tenían lugar la reunión semanal de todo el equipo ChangeIt, en la que participaba el *Product Owner*, el *Scrum Master* y el *Scrum Team*

2.2. Planificación

Considerando la metodología escogida para este proyecto, en esta sección se presenta con detalle una planificación temporal basada en tareas que se asocia directamente con una definición de requisitos apoyada en la creación del *product backlog* o pila de producto y las historias que la componen. Este *product backlog* se verá más en detalle en secciones posteriores.

Antes de iniciar la planificación se hizo una investigación sobre las tecnologías que se iban a usar y sobre las primeras ideas de posibles implementaciones *Business Intelligence*. Con ello se pudo obtener un especificación de requisitos más clara y una estimación de las historias de usuario mejor definida.

La planificación temporal de tareas mencionada se desarrolla en los propios Sprints, que son en total cinco Sprints de una duración de dos semanas cada uno, salvo el último que es de semana y media. Cada tarea consta de una estimación en horas que suman en total las 300 horas del proyecto. Esta estimación se basa primeramente en la carga de trabajo de dichas tareas, que se componen de subtareas, y en los puntos asignados de las historias de usuario en la pila de producto que se encuentra en la definición de requisitos. Esta puntuación está asociada a la dificultad de cada historia de usuario, que a su vez se interrelaciona con la dificultad de las tareas que aquí se exponen.

Si bien las tareas han sufrido modificaciones o sustituciones a lo largo del proyecto, la tabla 2.1 recoge la versión final de esta planificación temporal o pila de tareas.

Identificador	**Tareas**	**Estimación (en horas)**
T01	Introducción al proyecto: Reunión inicial con el CTO de la empresa. Comprender la estructura del proyecto interno empresarial. Investigación sobre las herramientas y el Business Intelligence.	6,5
T02	Creación del primer caso de uso "Summary Transactions Amount": Conocer la estructura necesaria para el caso de uso observando los ya implementados. Crear e implementar la clase Java en el Core del proyecto de la empresa. Crear e implementar la clase Java en la pasarela de base de datos, el SQL Gateways. Crear los tests unitarios pertinentes, tanto en Core como en Gateways Crear las ramas de trabajo pertinentes en GitLab y asignarlas al supervisor.	32,5
T03	Creación del endpoint para el primer caso de uso en API: Conocer la estructura necesaria para el endpoint observando los ya implementados. Crear la clase Java que servirá como Controller, para este endpoint y los siguientes. Crear e implementar las clases Java para el Command, Command Handler y Response. Crear los tests unitarios pertinentes, para el Command Handler y para el Controller. Crear las rama de trabajo para API en GitLab y asignarla al supervisor.	19,5

T04	Creación de nuevo endpoint en API sin caso de uso en Core "Get Total Balances": Implementar toda la lógica del endpoint en el Controller. Crear el test para el nuevo método del Controller.	32,5
T05	Creación de nuevo endpoint en API sin caso de uso en Core "Get Total Fees": Implementar toda la lógica del endpoint en el Command Handler. Crear e implementar las clases Java para el Command, Command Handler y Response. Crear los tests unitarios pertinentes, para el Command Handler y para el Controller.	26
T06	Primeros pasos en el frontend del proyecto: Investigar sobre el uso de Angular y el uso de ReactJS, sus ventajas y desventajas. Investigar librerías para las representaciones gráficas, ajustado a la versión de Angular del panel de soporte. Arreglar errores en Core, API y MySQL Gateways para preparar un merge request final en GitLab. Entender la estructura del frontend del panel de soporte. Crear un boceto para toda la interfaz a representar	19,5
T07	Creación de la primera representación gráfica: Ajustar la jerarquía de ficheros en el frontend. Diseñar en papel un boceto de la primera herramienta visual a crear en el panel. Construir la primera representación gráfica para el caso de uso "Transactions Amount Summary" implementando lo necesario en el componente TypeScript y ajustando la representación visual con HTML y SCSS.	32,5
T08	Creación de las representaciones gráficas de los últimos dos endpoints construidos: Diseñar en papel un boceto de la herramienta visual a crear en el panel. Construir la representación gráfica para los endpoints "Get Total Fees y Get Total Balances" implementando lo necesario en el componente TypeScript y ajustando la representación visual con HTML y SCSS.	32,5

T09	Creación de las representaciones gráficas fijas: Diseñar en papel un boceto de las gráficas a crear en el panel. Construir la representación gráfica para el análisis de las comisiones utilizando el endpoint y caso de uso "Get All Transactions.[ei]mplementando lo necesario en el componente con TypeScript para interpretar los datos y formatearlos adecuadamente. Ajustar la representación visual con HTML y SCSS.	32,5
T10	Creación de las representaciones gráficas variables: Diseñar en papel un boceto de las gráficas a crear en el panel. Construir la representación gráfica para el análisis de las comisiones utilizando los endpoints y casos de uso "Get All Transactions y Get Reports", "Get Summaries.[ei]mplementando lo necesario en el componente con TypeScript para interpretar los datos y formatearlos adecuadamente. Ajustar la representación visual con HTML y SCSS. Ajustes finales de toda la interfaz.	65

Cuadro 2.1: Pila de tareas

2.3. Gestión de riesgos

En la siguiente sección del documento se presenta la gestión de riesgos, esto es la identificación y clasificación de los riesgos. Esta sección es clave para el correcto desarrollo de la planificación, pues su identificación y clasificación ayudará a su prevención y a crear un plan de acción. Con el objetivo de su clasificación, los riesgos han de analizarse en profundidad a través de tres fases de análisis:

- Descripción, donde se identifica adecuadamente al riesgo, asignando un identificador y un nombre. En esta fase también se clasifican los riesgos por tipo, que puede ser de producto o de proyecto. Los riesgos de producto son aquellos relacionados estrechamente con las tecnologías utilizadas en el proyecto, mientras que los riesgos de proyecto se relacionan intrínsecamente con la gestión y desarrollo del mismo. Esto es observable en el cuadro 2.2.
- Análisis, donde se realiza una descripción exhaustiva de los riesgos ya identificados. El propio análisis se compone de la definición de su magnitud, de una descripción, de su impacto y de los indicadores que nos dirán si el riesgo se está produciendo. Esto se puede observar en el cuadro 2.3
- Plan de prevención y contingencia, donde para cada riesgo se diseña un programa que se compone primeramente de un plan de prevención, para construir ciertas normas para evitar que acontezca dicho riesgo, y de un plan de contingencia que define los pasos a seguir en caso de que el riesgo ocurra. Esto se recoge en el cuadro 2.4

ID	Descripción del riesgo	Tipo de riesgo
R01	Mala comprensión de los requisitos por parte del equipo	Riesgo de proyecto
R02	Falta de experiencia en las tecnologías usadas por el equipo	Riesgo de producto
R03	Diferencias entre el diseño actual y el previsto	Riesgo de proyecto
R04	Administración del tiempo errónea	Riesgo de proyecto
R05	Ausencia de personal / baja por riesgo vírico	Riesgo del Proyecto
R06	Recursos tecnológicos inestables o falta de permisos	Riesgo del Producto

Cuadro 2.2: Identificación de riesgos

ID	Análisis del riesgo
R01	*Magnitud* Su magnitud es variable según su fase, siendo baja en sus primeras fases y muy alta en su última fase. *Descripción* Sobre los requisitos se forman las historias de usuario que sirven de guía para el desarrollo del proyecto. Su incorrecta identificación y documentación afectará a toda la calidad del proyecto. *Impacto* La nueva incorporación o modificación de requisitos durante el proyecto desencadenará en cambios en una gran parte de la documentación y la ejecución de todo el proyecto. Las modificaciones serán menos costosas en sus primeras fases y muy costosas en su última fase. *Indicadores* Durante las reuniones con el cliente, no es capaz de indicar correctamente los servicios que espera obtener de la aplicación.
R02	*Magnitud* Variable, siendo baja en sus primeras fases y alta en las últimas. *Descripción* El equipo presenta dificultades en la consecución de los objetivos dada su inexperiencia, tanto profesional como con las herramientas utilizadas. *Impacto* Estas dificultades se expresan como retrasos en el proyecto o como deficiencias de calidad. *Indicadores* El equipo trabaja más lentamente.

R03	*Magnitud* Baja. *Descripción* Entre el producto final y las expectativas del usuario final se hallan discrepancias. *Impacto* Las discrepancias o el incumplimiento de las expectativas del usuario final puede implicar que no utilice el producto final o que trabaje con él a disgusto. *Indicadores* El usuario se siente insatisfecho con los avances en el proyecto tras cada reunión.
R04	*Magnitud* Variable entre media o alta, dependiendo de su fase. *Descripción* El coste temporal, necesario para el desarrollo del producto final, no está correctamente estimado. *Impacto* La subestimación de los costes temporales provocará en el equipo una sobrecarga de trabajo, mientras que la sobrestimación provocará periodos de ausencia del equipo. En cualquier caso esto afectará a la calidad directamente. *Indicadores* El *Scrum Master* recibirá quejas por parte del equipo sobre las dimensiones y carga del trabajo. O el equipo trabajará más o menos de las horas estimadas.

R05	*Magnitud* Media. *Descripción* El personal de la oficina puede presentar varios casos de infección vírica, durante el proyecto, por lo que debería ausentarse de la oficina y necesitar baja laboral. *Impacto* Puede provocar retrasos en el proyecto por bajas del personal si los síntomas son muy graves.Requiere retrasar, añadir horas extras o transferir el trabajo del afectado a situación remota. *Indicadores* El número de contagios en la oficina aumenta.
R06	*Magnitud* Media. *Descripción* El equipo no puede trabajar correctamente debido a la inestabilidad de los recursos tecnológicos, como pueden ser caídas de red, ordenadores anticuados, etc. Las restricciones a ciertos recursos, por la confidencialidad del sistema de la empresa. *Impacto* Puede provocar retrasos en el proyecto o durante ciertas fases a lo que se traduce en un producto final deficiente o con costes incrementados *Indicadores* Problemas continuos con los aparatos de la oficina o cambio recurrente de tecnologías debido a las restricciones de confidencialidad

Cuadro 2.3: Análisis del riesgo

ID	Plan de Prevención/Acción	Plan de Corrección/Contingencia
R01	Realizar más reuniones con el *Product Owner* y el usuario final. Documentar correctamente las reuniones para conocer los puntos en los que hay que trabajar correctamente. Cuestionarios tras cada reunión para aclarar dicha documentación.	Puesto que en las primeras fases la magnitud del riesgo es más baja se podrán realizar los cambios pertinentes para que se cumplan los nuevos requisitos. En las últimas fases se sopesará la importancia de las modificaciones implicadas para adaptarse a los nuevos requisitos, pues cualquier modificación implica una cantidad de tiempo necesario para realizarla, siendo en estas fases mucho mayor. En cualquier caso, si se aceptan los nuevos requisitos se revisará y se incorporarán los cambios necesarios en la documentación y el código.
R02	Durante el desarrollo del proyecto el equipo podrá dedicar parte de su tiempo de trabajo en el aprendizaje de las herramientas con las que encuentre dificultades.	En caso de que el aprendizaje del equipo inexperto no de resultados y tampoco funcione consultar fuentes externas como profesores, artículos o foros web, etc. Se realizará una redistribución de las tareas.
R03	Realizar cuestionarios, encuestas, y otras técnicas como el brainstorming durante las reuniones con el usuario final para aclarar cuáles son sus expectativas y que es lo que se puede ofrecer.	Aligerar las expectativas del cliente en primer lugar, si es posible. Valorar u ofrecer modificaciones que no supongan demasiado costo.
R04	Hacer las estimaciones temporales dando uso de varias herramientas y otros recursos. Tras la estimación revisar en las primeras reuniones si es correcta y posible.	El equipo tratará de adaptarse a posibles cambios en las dimensiones del proyecto, impliquen más o menos tiempo.
R05	Tratar de cumplir las metas y objetivos antes de lo estimado en la planificación siempre que sea posible. Adecuar las instalaciones a la situación y seguir todas las medidas preventivas	Cambiar el flujo de trabajo asignado a cada participante o intentar implementar el teletrabajo.
R06	Tener un ordenador portátil con el proyecto actualizado en caso de no poder usar el de la oficina. Conocer los permisos de acceso a los recursos tecnológicos.	Usar el portátil de repuesto o solucionar, dentro de todo lo posible los problemas con los recursos. Si no hay acceso a los recursos, buscar alternativas.

Cuadro 2.4: Plan de prevención

2.4. Estimación de recursos y costes del proyecto

Para la realización de este proyecto se han de definir una estimación de los costes que se presenta a continuación en esta sección. Estos costes incluyen toda la totalidad del proyecto por lo que se han de mostrar los costes en función del *hardware*, *software* y recursos humanos.

Aunque este proyecto se compone tanto por *backend* y *frontend*, el cálculo de los costes es el mismo pues se utilizan los mismos recursos por el único miembro del equipo, el estudiante de prácticas, a excepción del software en el que si se puede dividir los activos en estas dos implementaciones. Todos los costes del proyecto estimados se realizan en euros (€).

2.4.1. Hardware

En primer lugar, para iniciar la estimación de los recursos empezamos por el coste de los activos *hardware*. Este coste proporcional se refiere al uso de cada activo durante el proyecto, en el tiempo que ha sido necesario su uso, esto también será útil para sopesar si la adquisición del activo esta justificada.

Para calcular el coste *hardware*, se listan cada uno de los activos junto a su coste total (CT), que es su valor en el mercado. Seguidamente calculamos el coste proporcional dentro del proyecto, para esto hace falta conocer la vida útil o el tiempo de amortización del activo. En este caso se calcula en función del tiempo de vida útil (TVU), para el cual se considera de 5 años para ordenadores de sobremesa y de 4 años para ordenadores portátiles. Así pues, para realizar el cálculo se obtiene el coste total del activo entre la vida útil en horas y se multiplica por las horas del proyecto, 300 horas, quedando así la ecuación 2.1 que se muestra a continuación:

$$CP = (CT \div (TVU \times 12) \div 30 \div 24) \times 300 \tag{2.1}$$

Hardware	**Tiempo de Vida Útil**	**Coste Total (€)**	**Coste Proporcional (€)**
Ordenador de sobremesa	5 años	1.200,00	8,33
Monitor	6 años	140,00	0,81 €
Teclado y ratón	6 años	50,00	0,29
Ordenador portátil	4 años	900,00	7,81
Total		2.290,00	17,24

Cuadro 2.5: Costes hardware del proyecto

2.4.2. Software

El siguiente apartado en el que se ha de proseguir durante la estimación de costes, son los costes del *software*, situados en el cuadro 2.6. Esto es el conjunto de herramientas utilizadas en el desarrollo del proyecto. Como en el apartado anterior, se empieza listando los activos junto a su coste total o coste de mercado, pero en este caso el coste total es mensual pues, todos los

activos software son servicios de pago mensual. Además se diferencian los activos *software* que forman parte del *frontend* o *backend*. Ya que el estudiante en prácticas, posee licencias gratuitas de uso universitario el coste del *software* es reducido.

Implementación	**Software**	**Coste Total / mes (€)**
Común	Discord	0,00
	GitLab Premium	16,82
Backend	IntelliJ IDEA Community	0,00
	Docker	0,00
	AWS	1,00
Frontend	WebStorm	0,00
	Visual Studio Code	0,00
Total		17,82

Cuadro 2.6: Coste software del proyecto

2.4.3. Recursos Humanos

Por último, tras haber evaluado tanto los costes *hardware* y *software* el último paso es estimar los recursos humanos, que se recogen el cuadro 2.7. El sueldo de un programador *junior* o recién graduado, que en este caso sería el rol que desempeña el alumno en prácticas dentro de un *Scrum Team* profesional, sería de 16.000,00 € brutos anuales (SA), que vendrían a ser 1333,33 € al mes. Dicho todo lo cual, para obtener el coste de los recursos humanos el sueldo se calcula en función de las horas invertidas, para ello se calcula primeramente el salario por hora (SH) como se muestra en la ecuación 2.2.

$$SH = SA \div (52 \times 40) \tag{2.2}$$

Rol	**Sueldo Anual (€)**	**Sueldo por Hora (€)**	**Horas**	**Coste Total (€)**
Programador recién graduado	16.000,00	7,69	300	2.307,69

Cuadro 2.7: Costes de recursos humanos

Tras obtener los costes de recursos humanos se han de valorar los costes de contratación (CC). Estos son las cotizaciones de la seguridad social que paga la empresa, entre otros pagos. Pueden ser de hasta el 40 % de los costes de recursos humanos (CRRHH), por lo tanto se calcula como sigue en la ecuación 2.3

$$CC = 0,40 \times CRRHH \tag{2.3}$$

2.4.4. Costes Totales

Una vez se han obtenido los costes hardware, software y de recursos humanos por separado, finalmente se puede obtener el coste total del proyecto (CT) como se observa en la ecuación 2.4.

$$CT = CH + CS + CRRHH \tag{2.4}$$

A este coste total habría que añadirle los costes indirectos y de contratación, para obtener el coste total del proyecto completo (CTP).

Los costes indirectos (CI) son los que corresponden al lugar de trabajo, mesas, sillas, la limpieza, la electricidad y demás gastos. Suponen un 20 % del coste total del proyecto (CT), como se observa en la ecuación 2.5.

$$CI = CT \times 0,2 \tag{2.5}$$

Así pues, el coste total del proyecto completo (CTP) corresponde al cálculo de la ecuación 2.6.

$$CTP = CT + CI + CC \tag{2.6}$$

El resultado obtenido tras realizar las operaciones sería de 3.734,38 €. Los datos de la operación se pueden ver el cuadro 2.8

Recurso	**Costes individuales (€)**
Hardware	17,24
Software	17,82
Recursos Humanos	2.307,69
Coste Total	2.342,75
Costes Indirectos	468,55
Costes Contratación	923,08
Coste Total del Proyecto	**3.734,38**

Cuadro 2.8: Costes totales del proyecto

2.5. Seguimiento del proyecto

En esta sección del documento se detalla el control a lo largo del proyecto basado en la metodología descrita en secciones anteriores, el *Scrum*. Este control se basa en los *Sprints* desarrollados en el proyecto, por lo que por cada *Sprint* aquí se expone todo lo relacionado con este, es decir las reuniones de planificación, de seguimiento y las de cierre, además de las tareas que se llevaron a cabo y los desvíos o problemas en la planificación que se dieron si fue el caso.

2.5.1. Sprint 1

Reunión de planificación

En esta reunión de planificación se definió el objetivo del primer Sprint, que sería la primera toma de contacto con el proyecto. Este objetivo consistía en comprender la estructura del proyecto y la creación del primer caso de uso junto a la implementación de su *endpoint* en API.

Descripción y reunión de seguimiento

Este Sprint se centra en crear un nuevo caso de uso con el que realizar un sumatorio de todos los montos de las transacciones del sistema, a este nuevo caso de uso se le dará el nombre de *Transactions Amount Summatory* para ello se hizo un primer estudio de la estructura del sistema interno y de los ejemplos y casos de uso ya creados en el Core. En primer lugar se hizo la implementación en el Core de ChangeIt, para más tarde enlazarla con una nueva implementación en la Gateway de la base de datos. Para ambas implementaciones se crearon tests para comprobar su correcto funcionamiento. Por último se hizo una implementación del endpoint en la API del nuevo caso de uso, para este también fue necesario la creación de tests para el Controller y Command Handler.

Durante las reuniones de seguimiento se dieron a conocer errores y detalles a corregir. Como por ejemplo el ajuste de los decimales, pues en el Core del proyecto se trabaja en céntimos y en API en euros.

Tareas

- T01
- T02
- T03

Observaciones y reunión de cierre

Tras la reunión de cierre se hizo una optimización de la consulta mysql en la Gateway de la base de datos, para eliminar las sentencias *JOIN* innecesarias.

2.5.2. Sprint 2

Reunión de planificación

En esta reunión de planificación se establecieron los objetivos a cumplir en este segundo Sprint, que es el de la creación de dos nuevos *endpoints* dedicado el primero a obtener el balance total de los puntos de venta, por cada tipo de monedero y moneda. Y un segundo para obtener comisiones por cada monedero asociado a un monedero "padre".

Descripción y reunión de seguimiento

En este Sprint se desarrolla un nuevo *endpoint* para la API de ChangeIt, llamado *Get Total Balances* en la que se ha de hacer uso de métodos y estructuras ya creadas en el sistema, como son otros *endpoints* para obtener los tipos de monedero y divisa. El objetivo de este nuevo *endpoint* es poder obtener el balance total de los puntos de venta, teniendo en cuenta su tipo de monedero y su tipo de moneda. Toda la lógica de este *endpoint* se incluye en el Controller

En la segunda mitad del Sprint se trabaja en un nuevo *endpoint* para obtener las comisiones totales por cada monedero dentro del sistema que este sujeto a un monedero "padre"que se obtendrá con la ayuda de otros *endpoints* o casos de uso en Core.

De la misma manera que en el Sprint anterior, se han de crear los tests necesarios para comprobar su funcionamiento.

Durante la reuniones de seguimiento se localizaron errores de implementación en los tests.

Tareas

- T04
- T05

Observaciones y reunión de cierre

Este Sprint fue interrumpido por festivos, de todas formas se consiguieron los objetivos dentro del tiempo establecido. Tras la reunión de cierre se modificaron nombres de clases para conseguir que fueran más aclaratorios.

2.5.3. Sprint 3

Reunión de planificación

Durante la reunión de planificación se estableció el objetivo de este Sprint, que fue el de construir la primera herramienta visual en el panel de soporte, previamente habiendo investigado sobre el uso de Angular.

Descripción y reunión de seguimiento

En este Sprint se inició en primer lugar investigando sobre como utilizar correctamente Angular y las librerías disponibles para implementar gráficas. Tras comprender la estructura del *frontend* del proyecto interno se pudo empezar a construir los archivos necesarios para empezar a trabajar en la interfaz. Esta interfaz es básicamente un panel web de gran tamaño con herramientas interactivas y gráficas en función a los datos recogidos a través de caso de uso creados en Core, para el cuál se utiliza TypeScript para la comunicación con los endpoints de API y HTML y SCSS para todo el diseño en general.

En las reuniones de seguimiento se discutió por qué escoger Angular frente a ReactJS, y la

conclusión final fue que encajaría directamente en el panel de soporte sin tener que embeber una posible implementación con ReactJS en dicho panel.

Tareas

- T06
- T07

Observaciones y reunión de cierre

Las implementaciones con Angular quedan algo limitadas por la antigüedad de la versión utilizada, que además supone una investigación más costosa sobre material de aprendizaje y librerías a usar. Tras la reunión de cierre, el rol de *Scrum Master* se transfirió a otro integrante del equipo ChangeIt con más experiencia en el *frontend*.

2.5.4. Sprint 4

Reunión de planificación

Durante la reunión de planificación se estableció el objetivo de este Sprint, que fue el de construir las gráficas fijas en torno a las comisiones, que a la postre son el tipo de transacción más relevante para la empresa pues en estas se ve reflejado el beneficio económico que se va obteniendo.

Descripción y reunión de seguimiento

En este Sprint se ha trabajado en la adición de la representación visual de los últimos *endpoints* creados, en el *frontend* usando Angular. Se han pulido detalles de diseño y funcionalidad. En la segunda mitad del Sprint se ha trabajado en el diseño de gráficas fijas , es decir sin estar ligadas a ningún cambio de fecha manual, usando *frameworks* de gráficas estadísticas para Angular, como es *ngx-charts*. Además para dichas gráficas es necesario recoger los datos a través de las llamadas a la API, usando TypeScript. Una vez los datos han sido recogidos se deben formatear para que las gráficas puedan utilizar los datos correctamente, usando diferentes estructuras de datos para crear un JSON que pueda procesar el framework

En las reuniones de seguimiento, se barajó la posibilidad de utilizar otras librerías para las gráficas, pero al final se eligió la más compatible con el sistema.

Tareas

- T08
- T09

Observaciones y reunión de cierre

La elección de la librería de gráficas acabó resultando una mala decisión, pues retrasó el trabajo por su costosa curva de aprendizaje y uso. Además resultaron representaciones muy limitadas.

Tras la reunión de cierre, se ajustaron detalles de diseño como colores y tipos de gráficas. El cambio de tipos de gráficas equivalía a empezar de cero la construcción del JSON adecuado.

2.5.5. Sprint 5

Reunión de planificación

En esta última reunión de planificación se repaso el contenido de las gráficas fijas para conseguir el objetivo de las nuevas gráficas variables que se habrían de implementar en este Sprint. Estas gráficas son llamadas variables por que están sujetas a una fecha seleccionada o un lapso de tiempo ofrecido.

Descripción y reunión de seguimiento

Durante este Sprint se ha empezado trabajando en perfeccionar los datos representados en las gráficas, y añadir más opciones en las mismas y así obtener distintos tipos de datos. Es decir se han creado botones, para cambiar de nº de operaciones a importes totales y viceversa, selectores de tipo de transacción, botones de fechas y tooltips personalizados para las gráficas.

Seguidamente se desarrollaron las gráficas variables, una gráfica para mostrar las comisiones en los últimos siete, treinta y noventa días, una gráfica circular para mostrar las transacciones externas exitosas y una gráfica de barras para comparar las transacciones externas fracasadas con las exitosas.

Durante las reuniones de seguimiento se presentó la interfaz entera al equipo de ChangeIt con el objetivo de recibir retroalimentación para mejorar el diseño y las implementaciones

Tareas

- T10

Observaciones y reunión de cierre

Tras la reunión de cierre se concluyó que sería necesario implementar toda la interfaz en función de no solo las empresas, sino de los puntos de venta tanto los asignados a una única empresa como todos los que se hayan en la base de datos de Paynopain. Esta mejora se contempló desde el principio, pues todo el *backend* posee los filtros necesarios para ello pero habría que ajustar la implementación del *frontend*.

Capítulo 3

Análisis y diseño del sistema

3.1. Análisis del sistema

Durante esta sección se realiza el análisis del sistema, en primer lugar a través de la definición de requisitos basado en las historias de usuario y los requisitos de datos identificados. Y finalmente se expone el flujo organizativo de la empresa, la participación del sistema de *Buisness Intelligence* dentro de él y el diseño de la interfaz.

Si bien en este capítulo habría que explicar el diseño de la base de datos, el acceso a esta está restringido pues la empresa Paynopain trabaja con datos sensibles que no pueden ser públicos.

3.1.1. Definición de requisitos

Historias de usuario

Como se mencionó durante en secciones anteriores, para crear la pila de tareas que determinaron la planificación temporal se definieron las historias de usuario. Esta definición tuvo lugar durante una primera reunión en la que participaron el *Product Owner*, el *Scrum Master* y el *Scrum Team*. Las historias de usuario deben estimarse mediante una puntuación para poder clasificarlas correctamente, esta puntuación se decide tomando la historia que previsiblemente es más compleja y asignándole la puntuación más alta, en este caso sería de 10 puntos. Tras contemplar la historia más compleja, se puntúan las demás historias en relación a esta.

Para finalizar, se priorizaron y ordenaron de tal manera que se formó el *product backlog* observable en la figura 3.1, donde confluyen tanto las historias para el *backend* como las del *frontend*. La prioridad y el orden de las historias se decidió en base a su dificultad, puesto que las primeras historias servirían como una inicialización para el estudiante en prácticas.

Identificador	Descripción	Estimación
HU01	Como soporte. Quiero poder observar el estado actual de las empresas del sistema. Para actualizar, eliminar o aceptar la situación de cada empresa.	2 pts
HU02	Como soporte o administrador. Quiero poder obtener la suma de los importes de cada tipo de transacción o de todas a la misma vez, filtrado por fecha. Para conocer la situación de todas las transacciones y tomar las medidas que sean necesarias.	4 pts
HU03	Como soporte o administrador. Quiero poder observar el número de transacciones externas realizadas con éxito o la suma de los importes de cada tipo de transacción externa o de todas a la misma vez, filtrado por fecha. Para controlar la situación de todas las transacciones externas y tomar las medidas que sean necesarias.	5 pts
HU04	Como soporte o administrador. Quiero poder observar una comparativa entre el número de transacciones externas realizadas con éxito y las fracasadas. O una comparativa entre suma de los importes de cada tipo de transacción externa con éxito y fracasadas o de todas a la misma vez, filtrado por fecha. Para obtener un balance real entre las transacciones externas que se completan y las que no.	7 pts
HU05	Como soporte. Quiero poder observar el estado actual de las comisiones totales obtenidas en los distintos monederos, y en detalle las comisiones reales y las devoluciones. Para conocer la situación de los monederos del sistema interno de la empresa	3 pts
HU06	Como soporte o administrador. Quiero obtener una comparativa gráfica de todos los tipos de comisiones en los últimos siete, treinta y noventa días en función al número de operaciones o el importe total. Para conocer el balance de cada tipo de comisiones en varios periodos de tiempo.	6 pts

HU07	Como soporte o administrador. Quiero obtener una representación gráfica de las comisiones de todo el último año por cada mes en función al número de operaciones o su importe total. Para obtener una sutil predicción del año venidero o comparar entre ambos.	6 pts
HU08	Como soporte o administrador. Quiero obtener una comparativa gráfica de las comisiones del último año y este, por mes en función al número de operaciones o el importe total. Para comparar el estado actual de las comisiones y el del año pasado de una manera más concisa	4 pts
HU09	Como soporte o administrador. Quiero obtener una comparativa gráfica circular de las comisiones de los últimos cuatro años, en función al número de operaciones o el importe total. Para comparar el estado actual de las comisiones y el de años anteriores.	4 pts
HU10	Como soporte. Quiero obtener los balances totales por divisa y tipo de monedero de los puntos de venta de todo el sistema. Para conocer el balance de todos los monederos de los puntos de venta del sistema interno.	5 pts

Cuadro 3.1: Pila de producto con historias estimadas

Cabe mencionar, como se ha visto en la tabla anterior 3.1, que existen dos roles diferenciados, el rol de administrador que corresponde al administrador de una empresa punto de venta y el rol de miembro del equipo de soporte y análisis. En el primer caso las funciones dentro del sistema están algo limitadas pues a este solo le concierne observar los datos respectivos a su empresa, mientras que el miembro de soporte tiene acceso a todos los datos posibles.

Requisitos de datos

En cuanto a los requisitos de datos, en las siguientes tablas de 3.2 a 3.7 se hallan todos los requisitos identificados teniendo en cuenta las limitaciones de acceso a la base de datos o la visualización real de un diagrama de clases completo. En esta identificación se muestra el código o ID asignado, el nombre ,los datos específicos y los comentarios si proceden. Hay que tener en cuenta que esta identificación de datos se da en lo que respecta a la implementación *frontend*, donde se descartan ciertos datos que a priori no son necesarios para la creación de las representaciones visuales o gráficas de la interfaz.

Identificador	RD01
Nombre	Estado de empresas
Datos específicos	Identificación de empresa, tipo de empresa
Comentarios	En la representación del estado de las empresas se muestran las diferentes situaciones por las que se encuentran las empresas que están en ese momento en el sistema. Estos estados son: aprobado, en revisión, rechazado pendiente y de baja.

Cuadro 3.2: Requisito de datos RD01

Identificador	RD02
Nombre	Suma de importes transaccionales
Datos específicos	Fecha de inicio, fecha de fin, fecha contable, tipos de transacción, subtipos de transacción.
Comentarios	En la suma de importes transaccionales, se realiza dicha operación obteniendo los montos de las transacciones entre una fecha determinada y filtrando por los tipos y subtipos de transacción. La fecha pude ser regular o contable.

Cuadro 3.3: Requisito de datos RD02

Identificador	RD03
Nombre	Comisiones totales
Datos específicos	Fecha de inicio, fecha de fin, fecha contable
Comentarios	En la representación de las comisiones totales de los monederos del sistema se observa el importe total de todos los tipos de comisiones dentro de una fecha dada, que puede ser regular o contable

Cuadro 3.4: Requisito de datos RD03

Identificador	RD04
Nombre	Balances totales
Datos específicos	Estado
Comentarios	En la representación de los balances totales se muestra el importe total de los monederos de todos los puntos de venta que se encuentran en estado activo en la base de datos.

Cuadro 3.5: Requisito de datos RD04

Identificador	RD05
Nombre	Gráficas de transacciones normales
Datos específicos	Fecha inicio, fecha fin, fecha contable, tipo de transacción, subtipo de transacción
Comentarios	En las gráficas se representan diferentes datos sobre ciertos tipos de transacciones, por lo general comisiones, dentro de una fecha determinada que puede ser regular o contable. Sin embargo la fecha puede ser opcional, puesto que hay gráficas fijas que obtienen una fecha a través de la fecha real actual.

Cuadro 3.6: Requisito de datos RD05

Identificador	RD06
Nombre	Gráficas de transacciones externas
Datos específicos	Fecha inicio, fecha fin, fecha contable, tipo de reporte
Comentarios	En las gráficas se representan diferentes datos sobre ciertos tipos de transacciones externas dentro de una fecha determinada que puede ser regular o contable. Las transacciones externas dependerán del tipo de reporte. Este indica su origen, es decir, el departamento del que provienen.

Cuadro 3.7: Requisito de datos RD06

Los requisitos de datos anteriores 3.6 y 3.7 están relacionados con todos los tipos de representaciones gráficas de las transacciones normales y transacciones externas respectivamente.

En lo que respecta a la creación de diagramas de caso de uso o de actividades, se ha de mencionar que no se han realizado puesto que el flujo del sistema es sencillo y con la definición de las historias de usuario se sobrentiende su funcionamiento. De cualquier manera, en las siguientes secciones se refuerza todavía más la mecánica del sistema.

3.1.2. Análisis de requisitos

Para comprender en profundidad el sistema donde se da la anterior especificación de requisitos, se muestra el diagrama de clases de la figura 3.1.

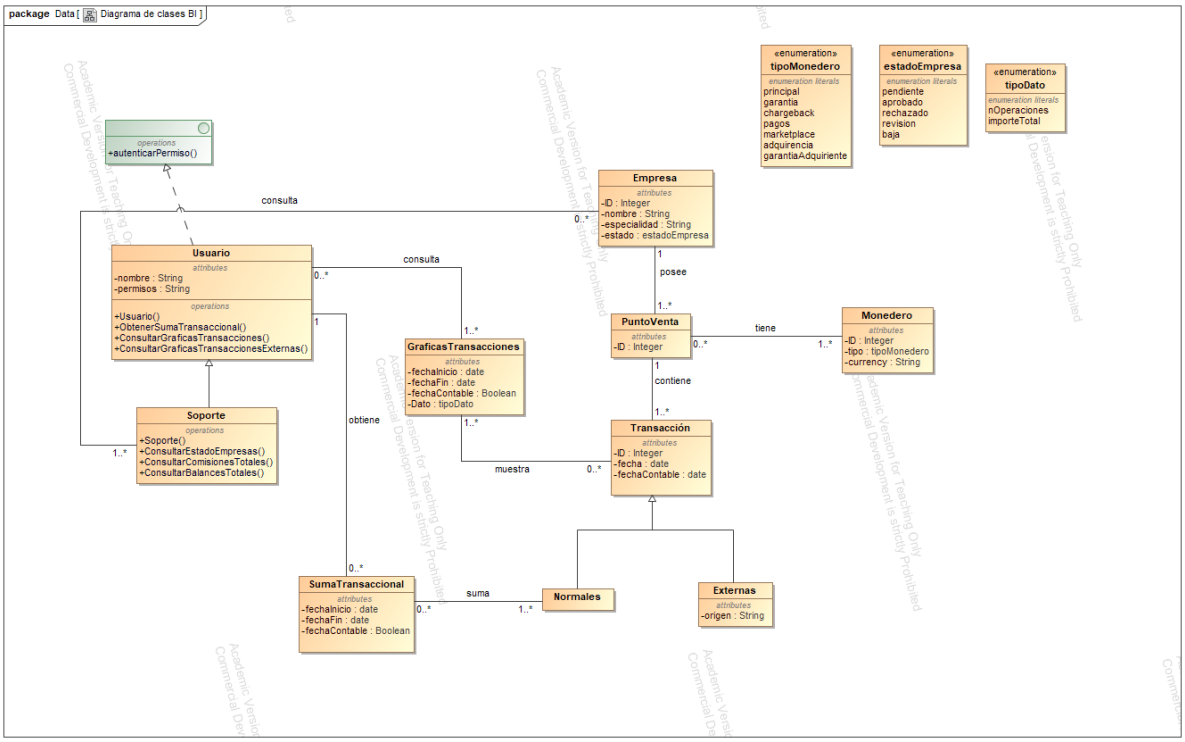

Figura 3.1: Diagrama de clases del sistema BI

Si bien la complejidad de los requisitos de las tablas 3.2, 3.4, 3.5 se resumen en una simple asociación entre el usuario y la empresa, el resto del diagrama muestra lo que sería un correcto análisis de los requisitos del sistema.

3.2. Diseño

3.2.1. Diseño del sistema

El sistema interno de la empresa, como se mencionó anteriormente, tiene una estructura de desarrollo hexagonal apoyado en tecnologías *RESTFUL*. Esta estructuración observable en el diagrama 3.2 representa una arquitectura que recoge tanto los componentes *backend* como *frontend*.

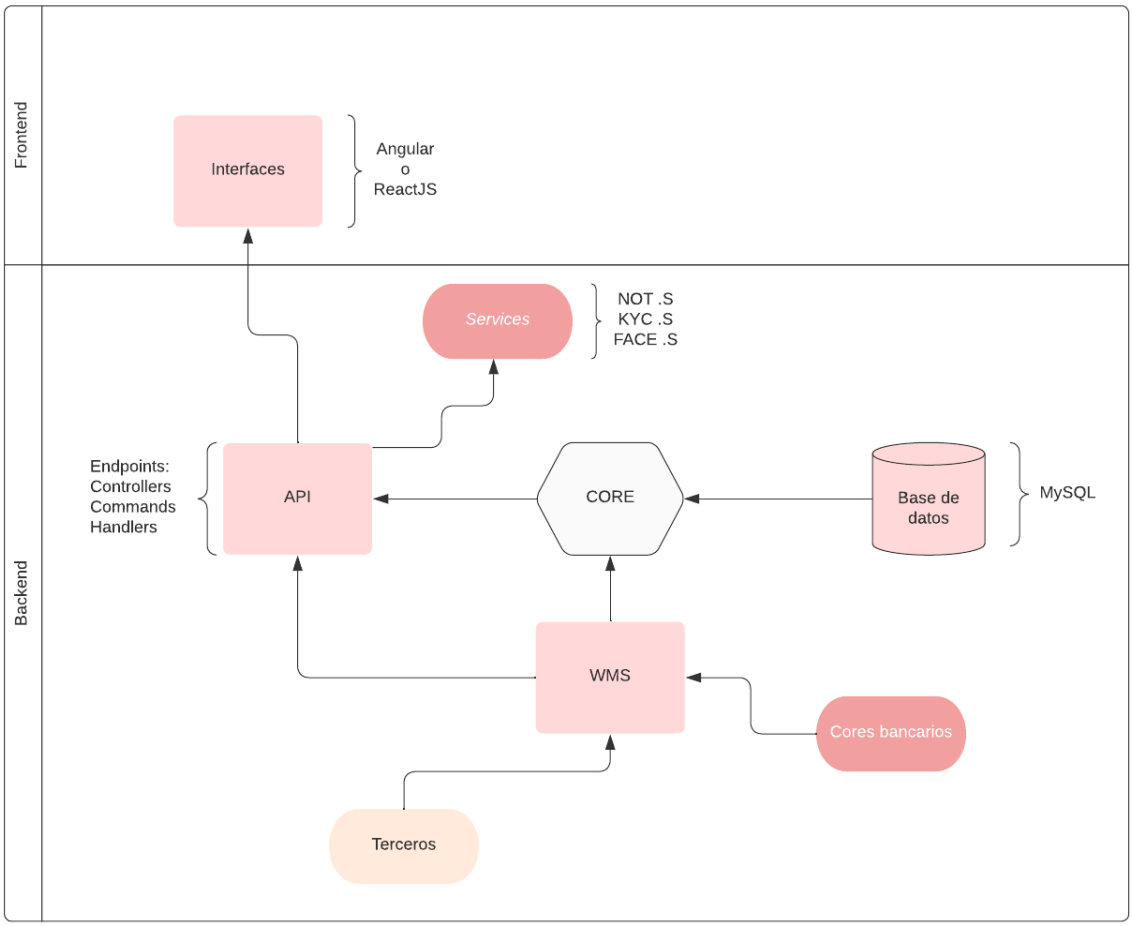

Figura 3.2: Estructura hexagonal del sistema en PNP

Como se observa en el diagrama anterior, todo el sistema tiene como punto central el llamado **Core** donde se implementa la lógica de los casos de uso, este mismo recibe desde el componente de Base de Datos todo lo necesario para construir los casos de uso. La **Base de Datos** se compone de implementaciones MySQL que recogen todas las consultas y acciones de inserción, actualización o borrado necesarias para los casos de uso.

La siguiente parte o componente de esta estructura que se comunica con el Core es el **Wallet Manager Service (WMS)** que representa en gran medida el gestor de los monederos, ya que cada monedero posee características distintas como puede ser el tipo, la divisa, el código SWIFT y diferentes requerimientos bancarios. En este componente hallamos también Cores bancarios particulares, es decir un componente del tipo Core como aquí se representa pero que ha sido implementado por una entidad ajena a la empresa, en este caso siempre o casi siempre un banco en particular.

Como bien se sabe, la confidencialidad es de suma importancia en todas las compañías *fintech* por lo que en ciertas ocasiones la gestión de monederos viene dada directamente desde un tercero, es decir una entidad bancaria. Esta gestión es luego procesada y verificada en el WMS del sistema.

En lo que respecta a la parte de la **API**, esta representa todas las funciones necesarias de comunicación entre las implementaciones *backend* y la implementación *frontend*. En la API se presentan los *endpoints* o los puntos finales de comunicación con el Core donde se implementan los Controllers, Command y Command Handlers, que servirán para manejar las peticiones HTTP, procesarlas y devolver las respuestas. La API también se comunicará con el WMS para las implementaciones, por otro lado también se hace uso de los **Services** que son diferentes servicios ligados generalmente a herramientas *fintech* de verificación y reconocimiento de clientes, como por ejemplo el *KYC (Know Your Costumer)* que es el proceso de "Conoce a Tu Cliente" que realizan las entidades financieras para verificar la identidad de sus clientes en cumplimiento de exigencias legales [3].

Y finalmente, las **Interfaces**, que se comunican estrechamente con la API para el uso de las implementaciones *backend*. Las interfaces se realizan principalmente con Angular, donde se incluye un lógica de creación de distintos elementos visuales con TypeScript, HTML y SCSS, aunque para ciertas ocasiones se utiliza ReactJS. Se podría decir que esta es la última capa de abstracción en la que se halla un último procesado de los datos para representarlos visualmente de la manera adecuada según lo necesario y según la autorización del usuario que este haciendo uso de la interfaz.

El sistema de *Business Intelligence* se posiciona a lo largo de esta estructura, haciendo uso de los componentes principales mostrados en mayor o menor medida. Sin embargo, cabe mencionar que hay componentes que son confidenciales como los Cores bancarios y hay otros como los Services que no necesitan de implementaciones nuevas.

3.2.2. Diseño de la interfaz

En esta sección se describen los criterios de diseño elegidos para el desarrollo de la parte *frontend*, es decir la interfaz interactiva y visual del sistema de *Business Intelligence*.

Dado que este proyecto se incluye en el sistema interno de la empresa, los estándares de diseño se ajustaron a la naturaleza del mismo. En pocas palabras, el objetivo a la hora de realizar el diseño era que encajara como una sección más del panel web de soporte de la empresa. Este proyecto interno utiliza una plantilla de pago para el dicho panel de soporte, esta plantilla *Fuse Theme* [4] proporciona una guía de diseño tanto para Angular y ReactJS. En la figura 3.3 se puede ver una pantalla de ejemplo de lo que sería dicha plantilla.

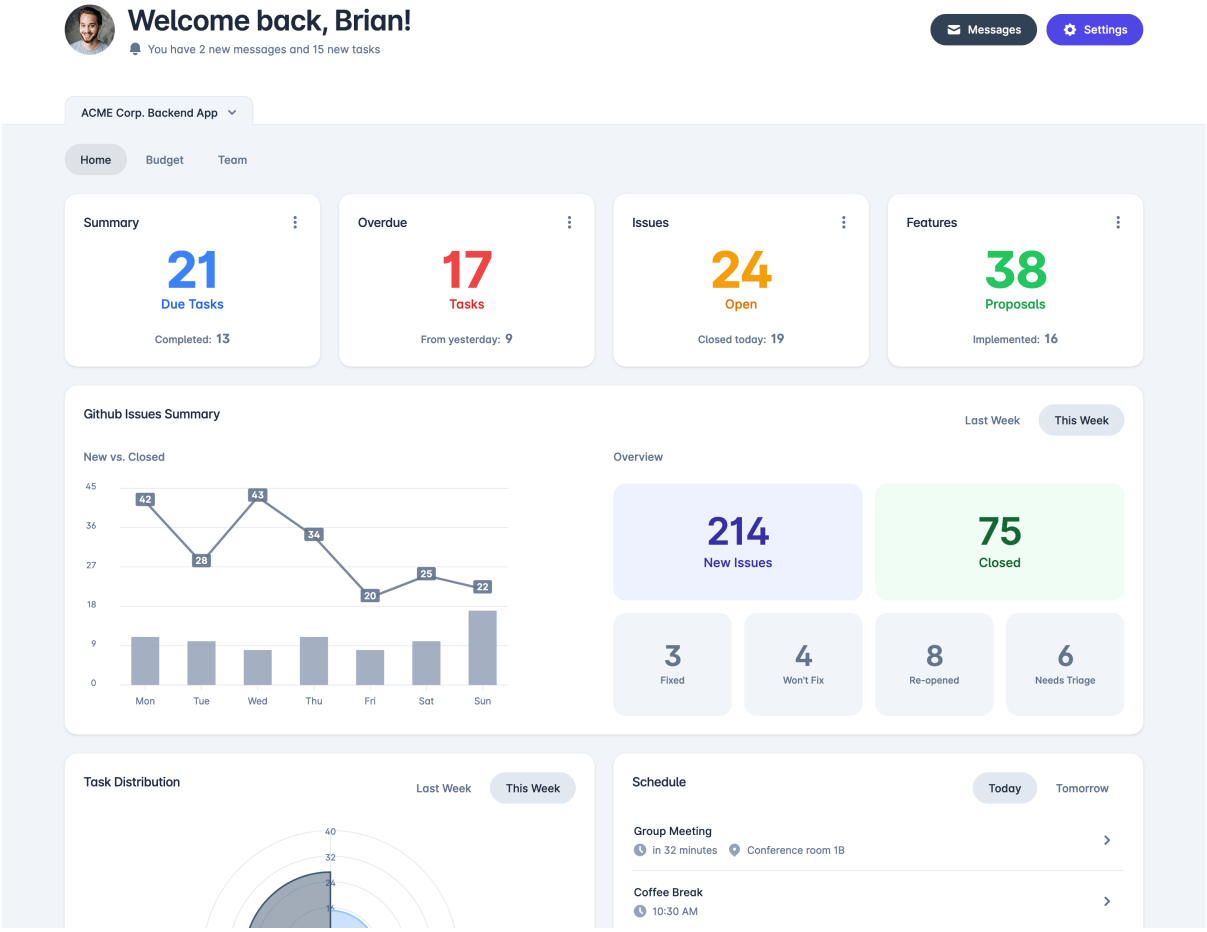

Figura 3.3: Pantalla de ejemplo de la plantilla *Fuse Theme*

En el caso de este proyecto se utilizó la plantilla mencionada con Angular, aunque con una versión diferente, pues el proyecto interno de la empresa utiliza Angular en su versión 4.0.0 par el panel de soporte. En cualquier caso esta plantilla supone una fuente de aprendizaje, pues el código se haya bien estructurado, comentado y documentado.

Antes de tomar la decisión de proceder con Angular se barajó la opción de utilizar ReactJS para la interfaz. Para ello se hizo una breve investigación sobre cuál de las dos opciones sería mejor para el desarrollo de este proyecto, así pues la decisión final tras revisar ciertos artículos se basó en la siguiente definición:

Angular es un *framework* de Javascript creado con Typescript, mientras que ReactJS es una biblioteca de Javascript creado con JSX. Angular se usa principalmente para crear aplicaciones complejas de nivel empresarial, como aplicaciones de una sola página y aplicaciones web progresivas, mientras que ReactJS se usa para crear componentes de interfaz de usuario en cualquier aplicación con datos frecuentemente variables. [5]

Puesto que la implementación de la interfaz del sistema de *Business Intelligence* se ajusta más a la definición de uso de Angular, se optó por este *framework*.

Toda la interfaz *frontend* se ha diseñado de la misma manera, como se mencionó anteriormente, siguiendo la naturaleza de las demás interfaces del panel. Por ello lo que se muestra a continuación son los detalles estéticos que se pueden recabar de todas las interfaces del sistema de la empresa Paynopain.

Paleta principal

Como se puede observar en las capturas de pantalla entre las figuras 3.4 y 3.6, la paleta de colores escogida ha de ser vistosa y estar relacionada con los colores de la empresa. Los colores predominantes son el rojo, gris y verde. Estos colores se pueden apreciar en varias tonalidades, buscando una armonía estética en su conjugación durante la creación de los elementos de la interfaz.

El color insignia de la empresa es el rojo con detalles en blanco 3.4.

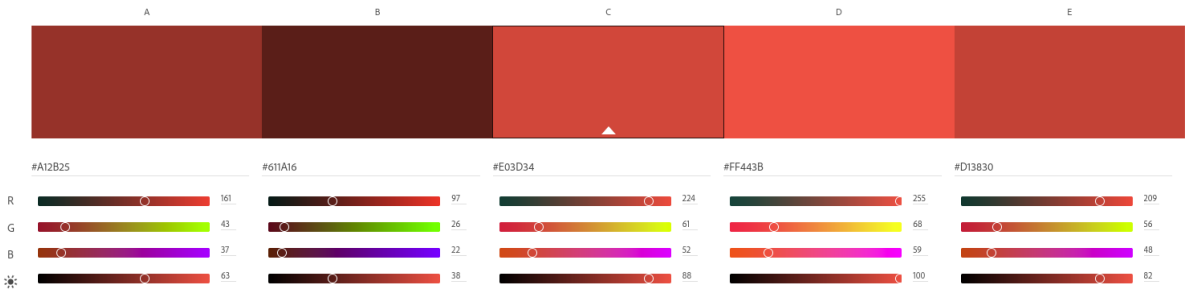

Figura 3.4: Escala de rojos similares al logotipo de Paynopain

Paleta de gráficas

La paleta de colores para las gráficas va en función de los datos que se representan. Es decir, los colores serán análogos si en la gráfica se representa el mismo dato pero en tiempos distintos o si son un mismo tipo de dato y serán colores contrarios o diferentes cuando se presenten datos que necesariamente se hayan de diferenciar unos de otros.

Por ejemplo en la captura de pantalla que se muestra a continuación 3.5 se muestra una gráfica circular diseñada con una paleta de colores tierra y colores chillones porque son tipos de transacciones que se han de diferenciar a simple vista para observar la proporción de cada uno. Mientras que en la 3.6 se aprecia una gráfica con un mismo color en diferente tonalidad porque es el mismo dato en un momento distinto en el tiempo.

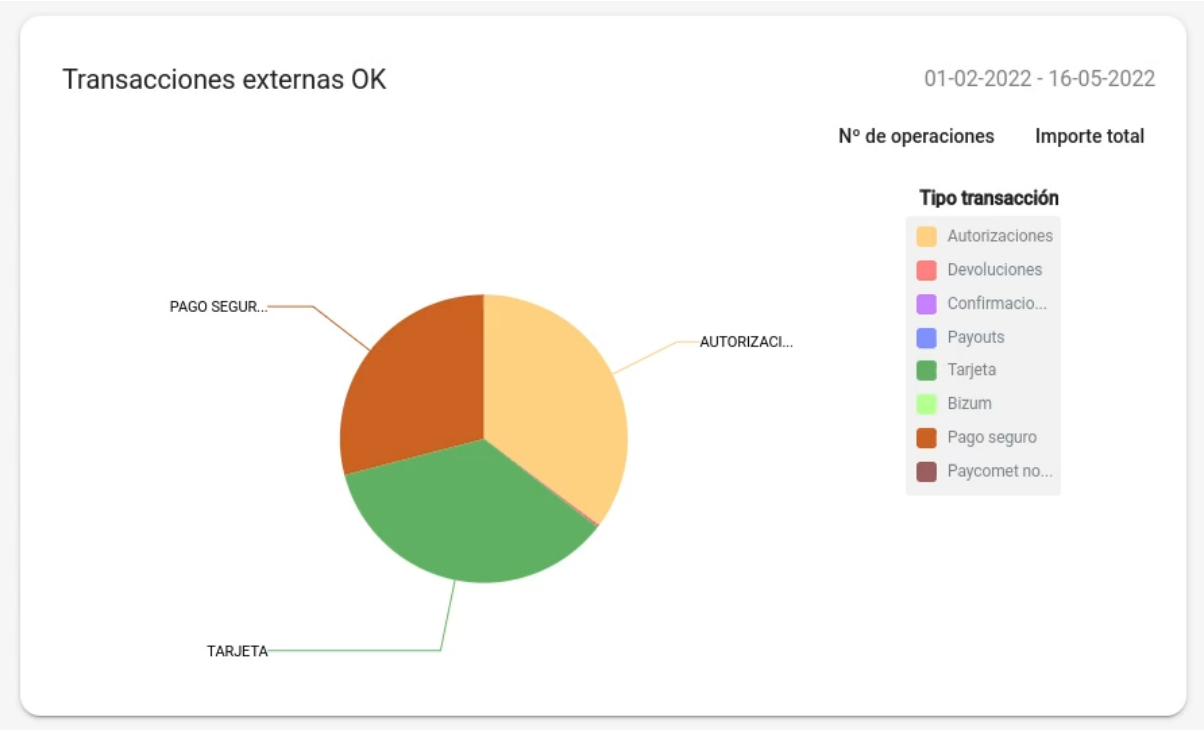

Figura 3.5: Gráfica circular de transacciones externas

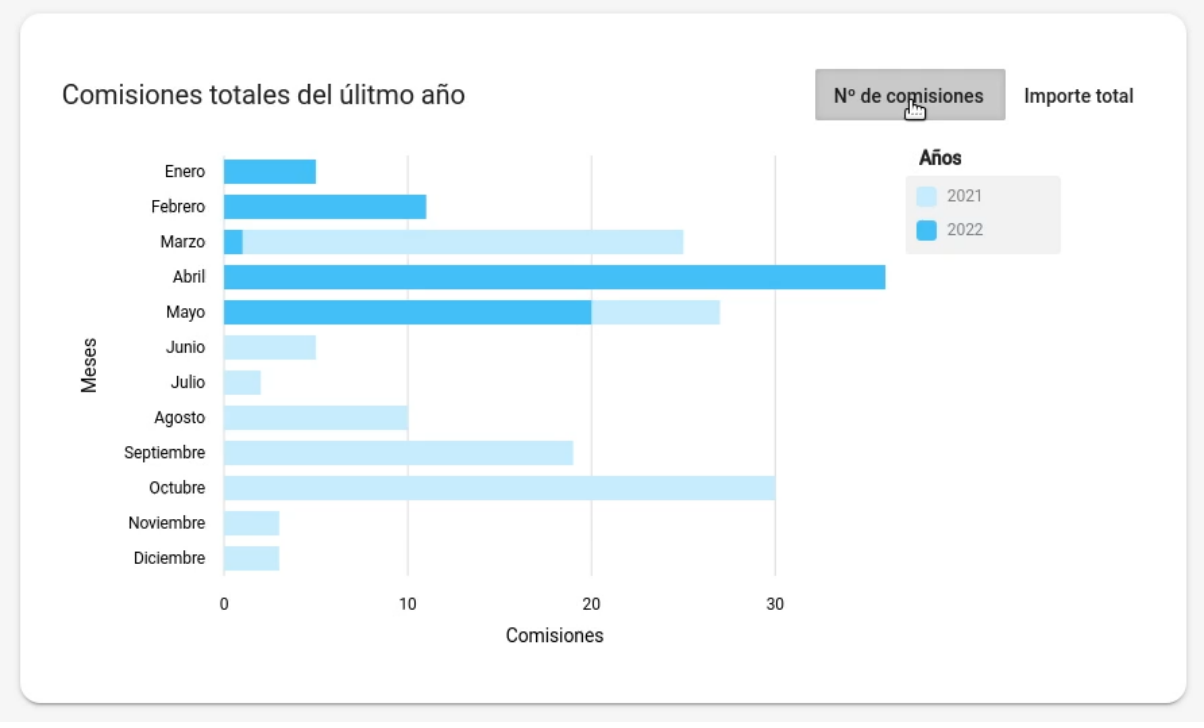

Figura 3.6: Gráfica comparativa entre comisiones de este año y el anterior

Mensajes de alerta

Los mensajes de alerta que se muestran cuando se da un error dentro del componente, generalmente un fallo en la llamada a la función de Core, son de color rojo siguiendo la estética de las interfaces.

Iconografía y tipografía

Tanto la iconografía como la tipografía se exportaron directamente de la plantilla en la que se basan todas las interfaces en el panel de soporte de la empresa 3.3. En las capturas de pantalla de dicha plantilla se observan la tipografía 3.7 e iconografía 3.8 respectivamente.

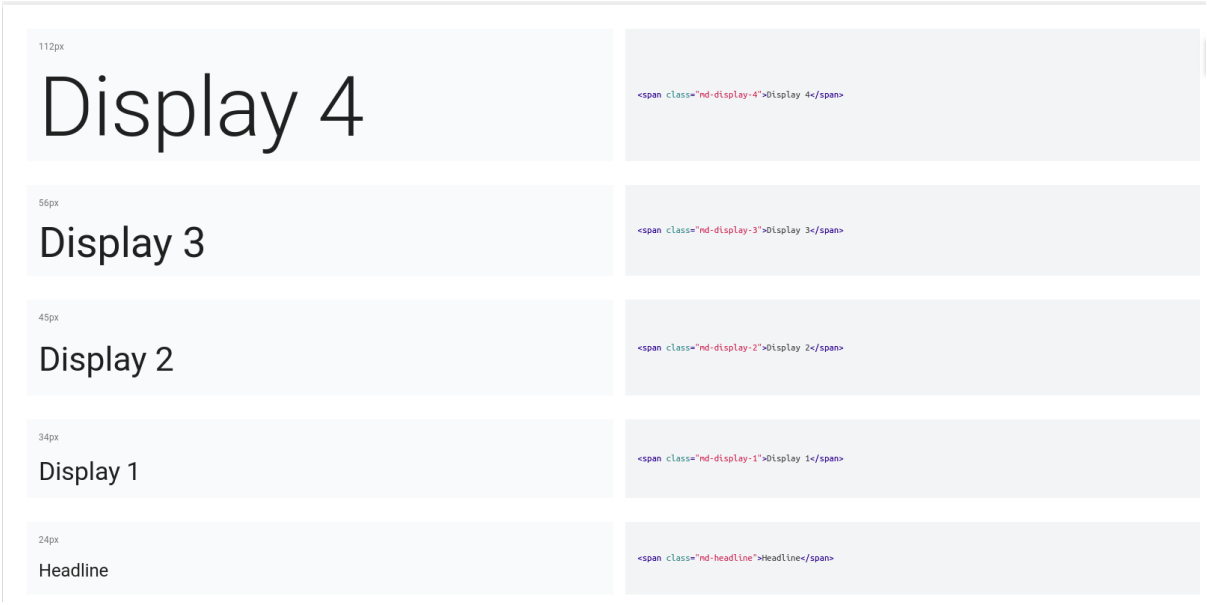

Figura 3.7: Tipografía de *Fuse Theme*

Figura 3.8: Iconografía de *Fuse Theme*

Capítulo 4

Implementación y pruebas

En este capítulo se muestran las implementaciones tanto en *frontend* como en *backend* del proyecto, además de una explicación en detalle de las pruebas realizadas.

4.1. Detalles de implementación

En esta sección se exponen los caminos tomados en la implementación del proyecto, desde una aclaración de los patrones de diseño usados pasando por las estrategias y decisiones realizadas durante el desarrollo del sistema.

Así pues se expone en primer lugar la implementación *frontend* y seguidamente la implementación *backend*. Y al final de esta sección se detallarán los resultados finales para ambas implementaciones, con muestras de código relevante que en el caso del *backend* serán los *endpoints* creados y casos de uso. Para apreciar con mejor claridad el resultado del *frontend* se incluirá una vídeo-demostración y capturas en el anexo A de esta memoria.

4.1.1. Implementación frontend

Como bien se ha mencionado a lo largo de todo el documento, la implementación *frontend* se ha realizado con el *framework* para aplicaciones web basado en JavaScript, Angular, en su versión 4.0.0. Esto significa que se ha hecho uso del lenguaje de programación TypeScript, que es un superconjunto de JavaScript, que esencialmente añade tipos estáticos y objetos basados en clases [6].

A su vez Angular hace uso de HTML y CSS, sin embargo en este proyecto se utilizará SCSS que es también un superconjunto, en este caso de CSS. SCSS es una versión más avanzada de CSS, a menudo llamado Sassy.

Para la creación de las gráficas se hizo uso de la librería *ngx-charts* que son adecuadas a la versión de Angular que se utiliza en el proyecto. Cabe mencionar que se hizo una investigación

al concluir el proyecto sobre como mejoraría el sistema si se sustituyera dicha librería por *chart js*. Se concluyó con que se apreciaba un mejor rendimiento y mejores resultados estéticos.

Estructura de la implementación

En cuanto a la disposición de las carpetas que forman la estructura del *frontend* no se puede mostrar ninguna captura de pantalla puesto que la implementación se encuentra dentro del proyecto interno de la empresa y esta se reserva el derecho a mostrar su estructura.

De cualquier forma, la estructura de un proyecto en Angular se puede generar a través de la gestión de paquetes *npm* y sus comandos *CLI*. Esto generará los archivos y directorios necesarios para cualquier proyecto, entre los que distinguimos los archivos *app* que son los módulos, archivos de rutas y archivos de datos. Además de los directorios *node modules*, donde se encontrar instaladas las librerías definidas por el archivo *package.json* y *src*, donde se hallan los componentes de la aplicación web.

A continuación se describe brevemente en qué consisten los archivos *app* mencionados:

- *app.modules*, cuando se crea un proyecto Angular se crea automáticamente en este fichero el módulo AppModule que es comúnmente conocido como módulo raíz. Dentro de este fichero se encuentran ciertas propiedades características, estas son: *declarations, exports, imports, providers, boostrap*. Así pues en este fichero sirve para importar y exportar módulos, componentes, servicios y demás elementos para el desarrollo *frontend*.
- *app.routes*, en este fichero se establecen las rutas de navegación en que permitirán a los usuarios, a medida que realicen tareas dentro de la aplicación web, moverse entre las diferentes vistas definidas [7].
- *app.data*, en este fichero se encuentran implementaciones auxiliares para los componentes de la aplicación web.

Por otro lado, los archivos que conciernen al *frontend* se incluyeron en una carpeta nombrada *statistics* dentro de otra llamada *panel* en *src* donde se encuentran todas las interfaces del panel de soporte. Cada una de estas interfaces representa un componente en Angular ligados a los *services*.

Un componente está formado siempre por tres tipos de archivo: el *.html* que se encarga de la representación visual del *frontend* junto al *.css*, o *.scss* en este caso, y el *.ts* que es el archivo *component* escrito en TypeScript donde se halla la lógica de la implementación. Aquí es donde se interpretan y tratan los datos para las gráficas y las herramientas visuales.

Los *components* hacen uso del decorador *@Component* que marca una clase como un componente Angular y proporciona metadatos de configuración que determinan cómo se debe procesar, crear instancias y usar el componente. En la figura 4.1 se puede ver el uso de dicho decorador.

Y como se ha mencionado, cada componente necesita de uno o varios *services*. Estos son archivos *.ts*, es decir TypeScript, que se encargan de comunicar,a través de llamadas HTTP, el *component* con los *endpoints* en API y así poder utilizar sus métodos en el *frontend*.

```
@Component({
  selector: 'fuse-statistics',
  templateUrl: './statistics.html',
  styleUrls: ['./statistics.scss']
})
```

Figura 4.1: Decorador *@Component* en el archivo *statistics.component.ts*

En la figura 4.2 se puede observar un extracto de código del *service* para que el *component* pueda dar uso del *endpoint "Transactions Amount Summatory"*.

```
@Injectable()
export class StatsService {

  constructor(private oauthService: OAuthService, private credentials: AppConfig, private http: Http, private app: AppData) {
  }
  /*Sum transactions amount with all filters*/
  public SumAmountWithAllFilters(startDate, endDate, accountingDateFilter, typeFilter: any[], subTypeFilter: any[]) {
    const headers = new Headers();
    headers.append('Content-Type', 'application/json');
    headers.append('Authorization', 'Bearer ' + this.oauthService.getAccessToken());
    let request = '';
    if (startDate != null && endDate != null) {
      if (accountingDateFilter){
        request = request + '?start_accounting_date=' + encodeURIComponent(startDate) + '&end_accounting_date=' + encodeURIComponent(endDate);
      }else {
        request = request + '?start_date=' + encodeURIComponent(startDate) + '&end_date=' + encodeURIComponent(endDate);
      }
    }
    if (!isNullOrUndefined(typeFilter) && typeFilter.length > 0){
      request = request + '&transaction_types=' + typeFilter;
    }
    if (!isNullOrUndefined(subTypeFilter) && subTypeFilter.length > 0){
      request = request + '&transaction_sub_types=' + subTypeFilter;
    }
    const url = this.credentials.getApiUrl() + 'stats/total_amount' + request;
    return this.http.get(url, {headers}).map(res => (<any>res)._body === '' ? {} : res.json());

  }
```

Figura 4.2: Llamada en *stats.service.ts* al *endpoint "Transactions Amount Summatory"*

Tras la creación del método de llamada en el *service* se ha de implementar en el *component* un método que reciba e interprete la respuesta adecuadamente. En la figura 4.3 se puede ver el método creado para el *endpoint* anterior.

En el caso de las gráficas creadas la lógica en el *component* es algo distinta, pues se necesita que los datos reciban un formato especial en JSON para que la librería pueda interpretarlos y crear las gráficas. Cabe tener en cuenta que cada tipo de gráfica exigía un formateado diferente. En la figura 4.4 se aprecia la lógica de una de las gráficas.

```
private SumAmount(startDate, endDate) {
  this.showLoader();
  this.statsService
    .SumAmountWithAllFilters(startDate, endDate, this.accountingDateFilter, this.listTypesToSend, this.listSubTypesToSend)
    .toPromise()
    .then(response => {
        this.totalAmount = response.amount;
        this.totalTransactionsProcessed = response.count;
        this.hideLoader();
      },
      (error) => {
        reject(error);
        this.hideLoader();
        this.panelComponent.checkError(error, this.translate.instant('ERROR.STATISTICS.TRANSACTIONS'),
          () => {
            this.SumAmount(startDate, endDate);
          }
        );
      }
    );
}
```

Figura 4.3: Método en *statistics.component.ts* para interpretar llamada desde *stats.service.ts*

```
private formatReportsTransactionsOkVsKo(reportsList: any, externalTypesSelected: any) {
  const data = [];
  const matrix: number[][] = [[], []];
  const ok = 0;
  const ko = 1;

  for (const report of reportsList) {
    const dateRaw = new Date(report.summary.date);
    const day = moment(dateRaw).toDate().getDate();
    const transactionsList = report.summary.summary_transactions;

    for (const transaction of transactionsList) {
      const valueOk = this.barsOkVsKo ? transaction.transactions_ok : transaction.amount_ok;
      const valueKo = this.barsOkVsKo ? transaction.transactions_ko : transaction.amount_ko;
      const amountOk = isNullOrUndefined(valueOk) ? 0 : valueOk;
      const amountKo = isNullOrUndefined(valueKo) ? 0 : valueKo;

      if (externalTypesSelected.indexOf(transaction.summary_type) > -1 || (externalTypesSelected.length === 0  && transaction.summary_type !== 'TOTAL')) {
        if (isNullOrUndefined(matrix[ok][day])) {
          matrix[ok][day] = amountOk;
        } else {
          matrix[ok][day] = matrix[ok][day] + amountOk;
        }
        if (isNullOrUndefined(matrix[ko][day])) {
          matrix[ko][day] = amountKo;
        } else {
          matrix[ko][day] = matrix[ko][day] + amountKo;
        }
      }
    }
  }
  console.log(matrix.length);
  if (!isNullOrUndefined(matrix)) {
    for (let i = matrix.length - 1; i >= 0; i--) {
      for (let j = 0; j < matrix[i].length; j++) {
        if (!isNullOrUndefined(matrix[i][j])) {
          const series = [];
          const serie = {
            'name': i === 0 ? 'OK' : 'KO',
            'value': matrix[i][j]
          };
          series.push(serie);
          const element = {
            'name': j.toString(),
            'series': series
          };
          data.push(element);
        }
      }
    }
  }
  return data;
}
```

Figura 4.4: Formateo de datos para la gráfica de Transacciones externas OK vs KO

Por supuesto, toda esta lógica creada no tendría sentido sin la creación de un HTML en concordancia. Puesto que el HTML es más largo y tedioso de entender, no se mostrarán extractos de código en esta sección, tampoco del SCSS.

Así pues, todos los archivos que se crearon o editaron específicamente para la creación del *frontend* se muestran en la figura 4.5.

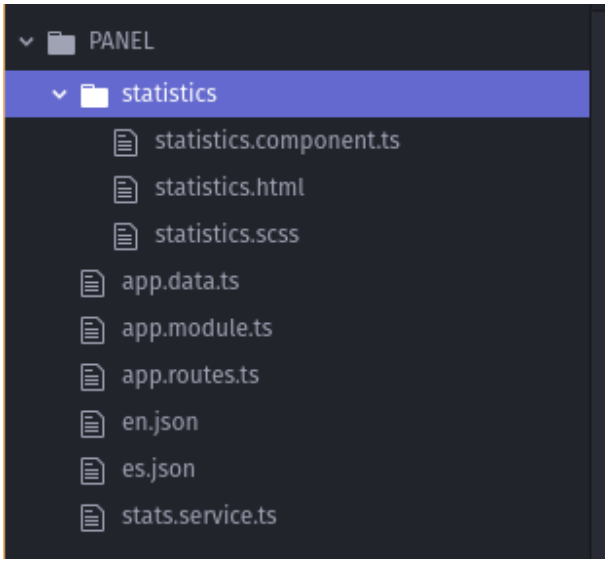

Figura 4.5: Archivos utilizados en la implementación *frontend*

Como se puede observar también existen un par de archivos JSON, estos son los archivos de traducción que se utilizan en el componente con el objetivo de intercambiar entre el idioma español y el inglés, en la interfaz. Esto se realiza a través de la librería *ngx-translate* que deja definir traducciones de contenido en diferentes idiomas y cambiar entre ellos fácilmente [8].

4.1.2. Implementación backend

La implementación *backend* de este proyecto, también referida como la implementación del lado del servidor (pues el proyecto se basa en la arquitectura clásica cliente-servidor) está dividida en tres implementaciones diferenciadas: Core, MySQL Gateways y API.

Como se podrá observar en esta mitad del capítulo, la implementación del lado del cliente fue más extensa que en la del lado del servidor ya que para esta última existieron más recursos formativos y ejemplos en los que basarse.

En la implementación del *backend* se hizo uso de la tecnología *Spring Boot* con la cual se elimina un gran parte del trabajo de configuración para las aplicaciones basadas en el *framework* de *Spring*, como sería este caso.

En otras palabras, *Spring Boot* permite enfocarnos únicamente en el desarrollo, pues este se encargará de tareas de configuración de dependencias, despliegue de servicio o aplicación a

un servidor de aplicaciones, etc [9] Para ello utiliza internamente un servidor de aplicaciones embebido, *Apache Tomcat* en este proyecto.

Conjuntamente se ha hecho uso de la anotación de *@GetMapping*, perteneciente al *framework Spring*, que nos permite simplificar el uso de los métodos de *Spring MVC @RequestMapping*.

También se ha hecho uso de la librería *Lombok* para eliminar código repetitivo y generar un código más limpio mediante anotaciones especiales como pueden ser *@EqualsAndHashCode @ToString*.

Core

El Core es el punto central de unión de la implementación *backend*. Para este proyecto se crearon dos clases relacionadas con las transacciones existentes en la base de datos, uno para contar el número de transacciones realizadas en una fecha determinada y otro para sumar los importes transaccionales en una fecha determinada, ambos siendo filtrados por el tipo de transacción. Estas dos clases representan los llamados internamente en la empresa, casos de uso.

Las clases aquí creadas deben de ser declaradas en el CoreDIC, un clase que funciona como puente entre las clases del Core y las del MySQL Gateways. En la figura 4.1 se observa la ejecución del método de suma de montos de MySQL Gateways en Core con los parámetros de la petición.

```
1  public class SumTransactionsAmount implements UseCase<SumTransactionsAmount.Request,
       Long> {
2
3      private TransactionGateway transactionGateway;
4
5      public SumTransactionsAmount(TransactionGateway transactionGateway) {
6          this.transactionGateway = transactionGateway;
7      }
8
9      @Override
10     public Long execute(Request request) throws UnavailableGatewayException {
11
12         Long sumTransactionsAmount = transactionGateway.sumAll(
13             request.eventId,
14             request.businessId,
15             request.startDate,
16             request.endDate,
17             request.sourceWalletId,
18             request.destinationWalletId,
19             request.sourceOrDestinationWalletId,
20             request.transactionTypes,
21             request.startAccountingDate,
22             request.endAccountingDate,
23             request.transactionSubTypes,
24             request.terminalId
25         );
26         return sumTransactionsAmount;
27     }
```

Código 4.1: El método *SumTransactionsAmount* en Core

MySQL Gateways

En esta parte de la implementación es donde se realiza la conexión con la base de datos MySQL. Lo que aquí se crea es una puerta de enlace entre el Core y la base de datos mediante métodos que incluyen consultas MySQL. El resultado de la consulta o consultas se interpreta y formatea de la manera adecuada para construir una respuesta que recibirá el Core.

Siguiendo el ejemplo anterior, en la extracción del código 4.2 se puede ver el formateo de ciertos datos obtenidos en la consulta y la ejecución de la consulta, que ha sido omitida por motivos de confidencialidad.

```
try (
            Connection connection = pool.get();
            PreparedStatement statement = connection.prepareStatement(
                "CONSULTA" + where
            )
        ) {

            int i = 1;

            //Formateo de consulta

            ResultSet resultSet = statement.executeQuery();
            resultSet.next();
            Long sum = new Long(resultSet.getInt("Resultado Consulta"));
            return sum;

        }
```

Código 4.2: El método de *SumAll* en MySQL

API

En la API los datos se reciben del Core y los comunica a la interfaz, es decir a al *frontend*. En la API se distinguen varios ficheros relevantes, entre ellos: el *Controller*, el *Command* y el *Command Handler*. Dicho todo lo cual, un *endpoint* se declara en el *Controller* y si la lógica de la implementación es muy particular, simple o corta se añade a este *Controller*. Si el *endpoint* puede ser relevante en otros se implementará la lógica en el *Command Handler* con una clase auxiliar llamada *Command*.

El ***Controller***, una clase Java que cumple la función de controlador, en este se definen los *endpoints* implementados en base a los casos de uso de Core y se encarga de recibir las peticiones HTTP de la interfaz. En 4.3 se observa un extracto de código del método *"Get Total Fees"*.

```

@RestController
@CrossOrigin("*")
```

```
@RequestMapping(value = "/stats", produces = MediaType.APPLICATION_JSON_VALUE)
@Validated

public class StatsController extends BaseController {

    @Autowired
    private SumTransactionsAmountCommandHandler sumTransactionsAmountCommandHandler;
    @Autowired
    private ListMerchantLocationsCommandHandler listMerchantLocationsCommandHandler;
    @Autowired
    private GetTotalFeesCommandHandler getTotalFeesCommandHandler;

        @GetMapping(value = "/total_fees")
    public ResponseEntity getTotalFees(
        HttpServletRequest request,
        @Valid GetTotalFeesRequestDTO getTotalFeesRequestDTO) throws Throwable {

        OAuthDTO oAuthDTO = getOAuthAttributes(request);

        GetTotalFeesResponse response = getTotalFeesCommandHandler.handle(
            new GetTotalFeesCommand(
                oAuthDTO.realmId,
                null == getTotalFeesRequestDTO.start_date ? null :
                    DateFormatter.parseDateFromString(getTotalFeesRequestDTO.start_date),
                null == getTotalFeesRequestDTO.end_date ? null :
                    DateFormatter.parseDateFromString(getTotalFeesRequestDTO.end_date),
                null == getTotalFeesRequestDTO.start_accounting_date ? null :
                    DateFormatter.parseDateFromString(
                    getTotalFeesRequestDTO.start_accounting_date),
                null == getTotalFeesRequestDTO.end_accounting_date ? null :
                    DateFormatter.parseDateFromString(
                    getTotalFeesRequestDTO.end_accounting_date)
            )
        );

        return new ResponseEntity<>(response, HttpStatus.OK);
    }
```

Código 4.3: Extracto de código del método *"Get Total Fees"*

Como se puede apreciar en el código, se utilizan anotaciones propias de *Spring* como el *@GetMapping* mencionado al principio de esta subsección. También anotaciones como *@RestController*, que indica la creación de un servicio REST, y *@RequestMapping*, que mapea las peticiones HTTP que se han de manejar.

Respecto al *Command*, es un tipo de archivo creado como una estructura de datos a recibir por el *Command Handler*. Su relevancia reside en la simplificación de código y en la manera en la que sea estructurado el *backend* desde le principio.

Finalmente, en el *Command Handler* se halla la lógica del *backend* en API, si no ha sido ya desarrollada en el *Controller*. Aquí se se realiza una llamada a los métodos necesarios de Core y se formatea la respuesta recibida de la manera correcta para después crear una respuesta, para la que suele crearse una clase *Response* específica que contiene métodos básicos que utilizará la

interfaz para crear los elementos visuales. En el extracto de código 4.4 se puede observar como se desarrolla el *Command Handler* para el *endpoint "Transactions Amount Summatory*, con partes omitidas que son similares.

```
@Service
public class SumTransactionsAmountCommandHandler implements
    ChangeitCommandHandler<SumTransactionsAmountCommand,
    SumTransactionsAmountResponse> {

    @Autowired
    private SumTransactionsAmount sumTransactionsAmount;

    @Override
    public SumTransactionsAmountResponse handle(SumTransactionsAmountCommand command)
        throws UnavailableGatewayException {

        SumTransactionsAmount.Request sumRequest = new SumTransactionsAmount.Request(
            new EventId(command.getRealmId()),
            null == command.getBusinessId() ? null : new
                BusinessId(command.getBusinessId()),
            null == command.getStartDate() ? null : command.getStartDate(),
            null == command.getEndDate() ? null : command.getEndDate(),
            null == command.getSourceWalletId() ? null : new
                WalletId(command.getSourceWalletId()),
            null == command.getDestinationWalletId() ? null : new
                WalletId(command.getDestinationWalletId()),
            null == command.getSourceOrDestinantionWalletId() ? null : new
                WalletId(command.getSourceOrDestinantionWalletId()),
            null == command.getTransactionTypes() ? null: transactionTypes,
            null == command.getStartAccountingDate() ? null :
                command.getStartAccountingDate(),
            null == command.getEndAccountingDate() ? null :
                command.getEndAccountingDate(),
            null == command.getTransactionSubTypes() ? null: transactionSubTypes,
            null == command.getDeviceId() ? null : new
                TerminalId(command.getDeviceId())
        );

        Long sumResponse = sumTransactionsAmount.execute(sumRequest);
        Integer countResponse = countAllTransactions.execute(countRequest);

        return new SumTransactionsAmountResponse(new
            BigDecimal(sumResponse).movePointLeft(2), countResponse);
    }
}
```

Código 4.4: Extracto de código del *Command Handler "Sum Transactions Amount*

Como se aprecia en el código anterior, se utiliza la anotación de *Spring* llamada *@Service* que indica su carácter de proveedor de servicio, esto quiere decir que se encarga de obtener la información de la base de datos y comunicar con el controlador. Por otro lado también aparece la anotación de *@Autowired*, que permite la inyección de dependencias con otras en *Spring*.

4.1.3. Patrones de diseño

El patrón que más peso ha tenido durante la implementación tanto para *frontend* y *backend* es el de la inyección de dependencias, este patrón que suele ser muy común en proyectos de este tipo, consiste en que cuando una clase requiere instancias de una o más clases, no las genera dentro de su propio constructor sino que las recibe ya instanciadas por un mecanismo externo. Este mecanismo suele ser un constructor o desde los *setters*, siendo esto último menos recomendable que el primero.

Esto se traduce a que cuando se requieran servicios u objetos que alguna de nuestras clases necesita, ya sean componentes, directivas o servicios, no habrá ninguna necesidad de instanciar dichas dependencias [10].

La inyección de dependencias logra directamente cumplir uno de los principios *SOLID* de la programación orientada a objetos, que es el Principio de Inversión de Dependencias o *Dependency Inversion Principle*. Este principio consiste en poder crear implementaciones concretas acopladas entre sí y poder intercambiarlas fácilmente. Esto significa que se debe separar la lógica que pueda ser reutilizable para aligerar la responsabilidad de las implementaciones o clases en este caso.
En Angular para este patrón de diseño se utiliza el decorador *@Injectable*, y concretamente en esta implementación *frontend* se encuentra en los *services* con el objetivo de poder inyectar la clase en el *component*.

Dada la naturaleza de este decorador en Angular, pues se haya accesible para todas las clases desde una sola instancia, se puede decir que pertenece a otro patrón muy importante en la programación, el *Singleton*.

El *Singleton* es un patrón de diseño, perteneciente a la categoría de patrones creativos que consiste en evitar la creación de más de un objeto concreto por clase. Al utilizar el patrón *Singleton* para crear una instancia de una clase, este se asegura de que realmente sólo permanezca con esta instancia única y de que esta clase sea accesible globalmente[11]. Es decir que la misma instancia será reutilizada e inyectada en todas las clases que la utilicen.

Finalmente, el último patrón de diseño software destacable es el de MVVM (modelo-vista-vistaModelo) una alternativa de Angular para el clásico MVC (modelo-vista-controlador). El MVVM es un patrón de diseño que tiene como objetivo separar la parte de la interfaz de usuario (Vista) de la parte de la lógica empresarial (Modelo) para que la parte visual sea completamente independiente. El último componente es el de la vistaModelo, que actuará comunicando la Vista y el Modelo [12].

4.1.4. Estrategia de objetivos

Para la construcción de las gráficas del sistema que se ha creado, se ha tenido en cuenta el concepto de indicador. Un indicador es un métrica asociada a un objetivo. Por ejemplo: 20 transacciones de tipo comisión realizadas cada mes, 5 más de lo previsto respecto al año pasado.

Dicho esto, para la creación de las gráficas con buenos indicadores, se ha utilizado la estrategia ***SMART*** *(Specific, Measurable, Attainable, Relevant, Time-bound)*. Esta es una estrategia basada en objetivos que permite hallar la manera más efectiva posible para la consecución de metas en una empresa.

Los objetivos *SMART*, que los indicadores de las gráficas del sistema han de cumplir son:

- Specific, el indicador utilizado tiene un indicador específico para la empresa.
- Measurable, su valor puede obtenerse.
- Attainable, es un indicador realista, es decir alcanzable.
- Relevant, ha de ser importante para el equipo y la empresa, es decir relevante.
- Time-bound, se encuentra medido en un marco de tiempo.

4.2. Verificación y validación

En esta sección se detallan las pruebas de verificación y validación realizadas con la intención de comprobar si el funcionamiento de las implementaciones es correcto. Ya que este proyecto se desarrolla tanto en *backend* como en *frontend* se ha de distinguir entre las pruebas unitarias para *backend* y las pruebas simples de uso que se realizaron para *frontend*.

4.2.1. Pruebas unitarias en backend

Durante el desarrollo de todo el sistema se han realizado tests unitarios para todas sus partes, es decir en API, Core y MySQL Gateways. En el caso de la API, se realizaron tests para el *Controller* y el *Command Handler*.

Estos tests se realizaron conforme se desarrollaba el código, pues es imprescindible comprobar que lo implementado es correcto para seguir avanzando sin tener que retroceder para encontrar fallos que se den en otras partes. Los tests deben procurar probar todos las interacciones posibles de los métodos que se hallen en cada clase.

En **Core** y **API** se utilizó el *framework* de prueba de código *Mockito* que permite "falsear"las dependencias que necesita la clase para ser probada. Para crear un *mock* se ha de conocer exactamente lo que se quiere devolver para así implementar un respuesta "falsa". Cuando se pruebe la clase implementada realizará un *doReturn* que "falseará"la llamada a la clase que se quiere *mockear* y devolverá la respuesta "falsa". Esto se puede observar en el extracto de código 4.5.

```
1 public class SumTransactionsAmountTest {
2
3     TransactionGateway transactionGatewayMock;
4     SumTransactionsAmount sumTransactionsAmount;
```

```
5
6       public SumTransactionsAmountTest() {
7           transactionGatewayMock = mock(TransactionGateway.class);
8           sumTransactionsAmount = new SumTransactionsAmount(transactionGatewayMock);
9       }
10
11      @Test
12      public void givenCorrectRequest_whenExecute_shouldReturnAmount() throws
            UnavailableGatewayException {
13
14          doReturn((long) 300).when(transactionGatewayMock).sumAll(
15              new EventId(1),
16              new BusinessId(1),
17              new Date(1),
18              new Date(2),
19              new WalletId(1),
20              new WalletId(2),
21              new WalletId(1),
22              new ArrayList<TransactionType>() {{
23                  add(TransactionType.valueOf("FEE"));
24              }},
25              new Date(1),
26              new Date(2),
27              new ArrayList<String>() {{
28                  add("CASH_WITHDRAWAL");
29              }},
30              new TerminalId(1)
31          );
32  }
```

Código 4.5: Extracto de código del test de *"Sum Transactions Amount"* en Core

Si bien los tests en API requerían de más elaboración, el funcionamiento del *mock* es prácticamente idéntico salvo por anotaciones de *Mockito* como *@InjectMocks*, utilizada para inyectar un *mock* en otra clase.

Respecto a las pruebas unitarias realizadas en MySQL Gateways, se realizaron sin *Mockito*, pues estas pruebas consistían en realizar una conexión directa con la base de datos. Para saber como construir las llamadas con los parámetros adecuados se utilizaba un ejemplo de batería de datos para las transacciones en XML.

4.2.2. Pruebas simples de uso en frontend

Debido a que es una única interfaz y su uso está centrado a quienes tienen ya bastante experiencia en el panel interno web de Paynopain, se realizaron pruebas sencillas de cómo debería funcionar.

En primer lugar se comprobó el funcionamiento de los filtros de fecha y el botón de fecha contable, en una primera versión de la interfaz el filtro de fecha no funcionaba al marcar la opción de fecha contable. Este fallo procedía de la API y se arregló fácilmente.

Seguidamente se comprueba que las fechas correspondan a todas las gráficas y representaciones visuales. Para las gráficas que no dependen de los filtros de fecha sino que utilizan una fecha fija se comprobó que no se alterasen al utilizar los filtros para las demás gráficas.
Esta comprobación es esencial para que no se interpreten datos incorrectos en las gráficas. Y puesto que es una parte sensible del código del *Component* ya que depende de una librería en camino de ser deprecada, *MomentJS* .

También se comprobó que el cambio de datos en las gráficas funcione, es decir los botones que intercambian dichos datos por nº de operaciones e importe total.

Más allá de todo esto, se cotejaron los datos representados en las gráficas con los que se podrían obtener de las herramientas visuales creadas, además de otras secciones del panel de soporte de Paynopain.

Capítulo 5

Conclusiones

Para finalizar con esta memoria en este último capítulo se exponen las conclusiones en tres distintos aspectos: en el ámbito formativo, en el ámbito profesional y en el ámbito personal. Empezando por este último.

5.1. Conclusiones técnicas

En cuanto a la consecución del proyecto, se han cumplido con los objetivos del inicio pero han aparecido inconvenientes y cambios de última hora que han rebajado, en cierta medida, la calidad el producto final. Si bien el tiempo para realizar el proyecto no fue un problema, la elección de las librerías de gráficas por mi parte no fue la adecuada.

Dicho esto, la interfaz puede ganar más visibilidad y funcionamiento con ciertas mejoras. Por ejemplo una investigación en profundidad de las librerías de gráficas más completas disponibles y su sustitución por las actuales.

5.2. Conclusiones personales

Ámbito personal

Durante la estancia he podido aprender sobre como trabajar en un equipo de desarrollo de *software*, desde el flujo de trabajo hasta las actitudes y decisiones que tomar dependiendo de la situación. Aunque este proyecto pareciera a simple vista dirigido más ciertamente a solo el campo de los sistemas de información y el análisis de datos únicamente pues el *Business Intelligence* podría no requerir de ningún conocimiento de programación, la realidad es que en Paynopain se han esforzado por crear un proyecto en el que se aprende tanto la profesión de *software developer* tan demandada en la actualidad como la de un analista de datos junior. Si bien es cierto que el proyecto se centro quizás demasiado en la implementación *software* y no tanto en la inteligencia de negocios, agradezco que haya sido así pues he podido ampliar mi

experiencia y descubrir realmente en que dirección encaminar mi carrera profesional.

Personalmente, me hubiera gustado haber podido trabajar más en la parte de *backend* pero he disfrutado haber hecho mucho *frontend* puesto que durante el grado no hay tantas oportunidades para trabajar en esto en una manera más profesional.

A partir de ahora quisiera seguir enfocándome en el desarrollo software, y progresar tranquilamente en el campo de la información pues esto último suele concernir a puestos más elevados en una empresa. Me gustaría también enfocarme en dispositivos móviles en el futuro, ya que las plataformas web ya las he probado.

Ámbito formativo

La estancia ha sido perfecta para aprender lenguajes como TypeScript y reforzar los conocimientos en tecnologías web. Como he mencionado anteriormente, este proyecto es muy completo pues he podido aprender muchísimas tecnologías que me servirán de ayuda en el futuro, pues son más universales que las centradas en los sistemas de información. Mientras que otros alumnos que hayan realizado algo parecido a este proyecto solo podían usar herramientas específicas como Power Bi de Microsoft para la creación de dashboards, yo he tenido la oportunidad de crear un sistema entero desde cero. Aunque me hubiera gustado haber podido trabajar algo más en la predicción de datos, el proyecto ha sido muy completo desde el punto de vista formativo.

Ámbito profesional

En cuanto a lo profesional, la experiencia ha sido muy gratificante pues he aprendido a trabajar con un equipo competitivo y además muy amigable. Los trabajadores de Paynopain han trabajado duro enseñándome el funcionamiento de un proyecto profesional y de su implementación. Por su puesto las opiniones del equipo durante las reuniones semanales fueron de gran ayuda pues avance con firmeza a los puntos a los que quería llegar. Dado que el proyecto de prácticas estaba incluido en el propio sistema de la empresa, pude sentir como era parte del equipo en todas sus formas, inclusive cometiendo errores y trabajando bajo cierta presión para solucionarlos. Si bien me hubiera gustado una organización algo mejor, pues había ocasiones en las que no podía avanzar a un ritmo más acelerado pues dependía en gran medida de las correcciones por parte de mi supervisor a través de GitLab para poder *mergear* mis implementaciones y dado que es un equipo pequeño habían retrasos en las correcciones por la carga de trabajo. Pese a todo, siento que en Paynopain hicieron todo lo posible por enseñarme la profesión y como trabajar en un equipo con una comunicación constante y con un ámbito laboral saludable.

Bibliografía

[1] SoftwarePara.net. ¿Qué es Business Intelligence? Ventajas y Objetivos. https://softwarepara.net/definicion-business-intelligence/. [Consulta: 29 de Abril de 2022].

[2] Alberto Dominguez. ¿qué es scrum y por qué tu equipo debería evaluarlo? https://www.albertodominguez.co/que-es-scrum/. [Consulta: 20 de Mayo de 2022].

[3] Marta Estudillo. ¿Qué es Know Your Customer (KYC) y qué implica? https://blog.signaturit.com/es/que-es-know-your-customer-kyc-sector-financiero#:~:text=Conclusi%C3%B3n-,%C2%BFQu%C3%A9%20significa%20KYC%3F,en%20cumplimiento%20de%20exigencias%20legales. [Consulta: 24 de Mayo de 2022].

[4] Helvacı, Mustafa y Yemen, Sercan. Fuse Theme. https://fusetheme.com/. [Consulta: 24 de Mayo de 2022].

[5] Dhaduk, Hiren. Angular vs React 2022 : Which JS Framework your Project Requires? https://www.simform.com/blog/angular-vs-react/#:~:text=Angular%20is%20a%20Javascript%20framework,app%20with%20frequently%20variable%20data. [Consulta: 26 de Mayo de 2022].

[6] Microsoft. TypeScript. https://www.typescriptlang.org/es/. [Consulta: 4 de Junio de 2022].

[7] Google Inc. , Angular. Angular Routing. https://angular.io/guide/routing-overview. [Consulta: 20 de Mayo de 2022].

[8] Combe, Olivier. ngx-translate. http://www.ngx-translate.com/. [Consulta: 4 de Junio de 2022].

[9] Vázquez González, Marcos. ¿Qué es Spring Boot? https://blog.codmind.com/que-es-spring-boot/. [Consulta: 4 de Junio de 2022].

[10] Cea, Osman. Inyección de Componentes y Directivas en Angular. https://medium.com/angular-chile/inyecci%C3%B3n-de-componentes-y-directivas-en-angular-6ae75f64be66. [Consulta: 20 de Mayo de 2022].

[11] IONOS, Digital Guide. Patrón singleton: una clase propia. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/patron-singleton/. [Consulta: 4 de Junio de 2022].

[12] Rodriguez, Eduardo. MVVM – Qué es y como funciona. https://inmediatum.com/blog/ingenieria/mvvm-que-es-y-como-funciona/#:~:text=MVVM%2C%20por%20sus%20siglas%20en,parte%20visual%20sea%20totalmente%20independiente. [Consulta: 4 de Junio de 2022].

Anexo A

Demostración del sistema

En este anexo se muestran algunas capturas de pantalla del sistema creado y un enlace a un vídeo del sistema en funcionamiento.

Enlace vídeo demostración

`https://drive.google.com/file/d/1TA4ZT94V_fruHtFU4YoCoftkV2dKu1Ru/view?usp=sharing`

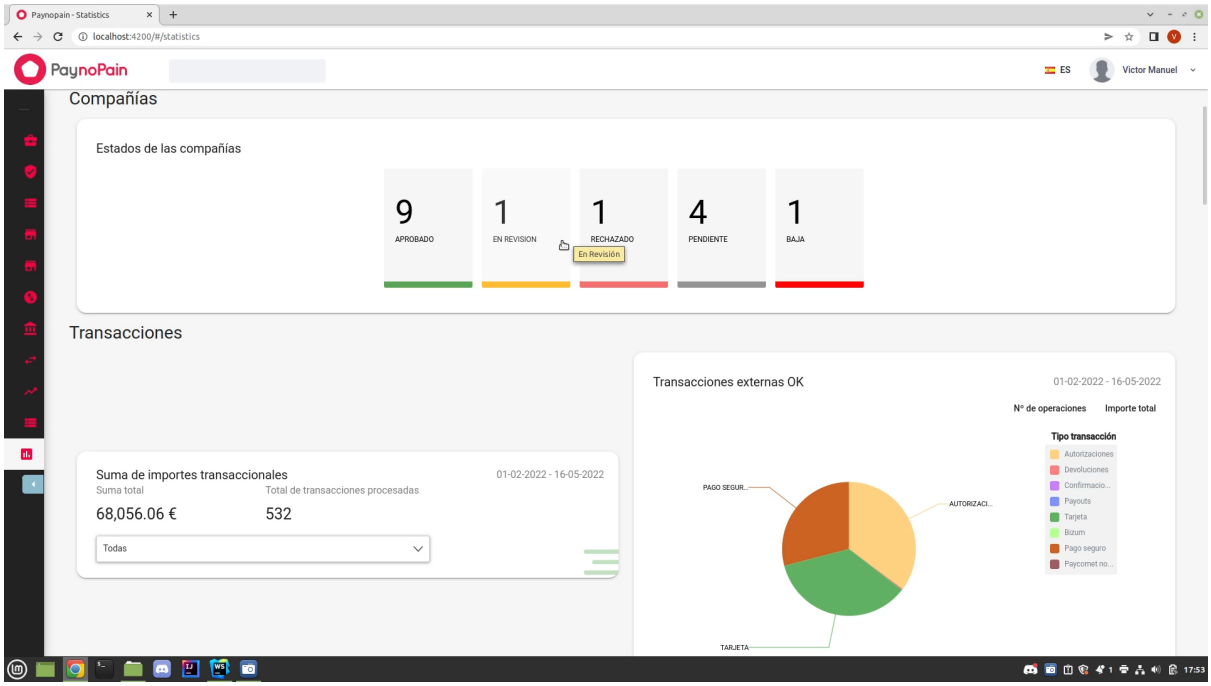

Figura A.1: Captura de pantalla nº 1 del sistema

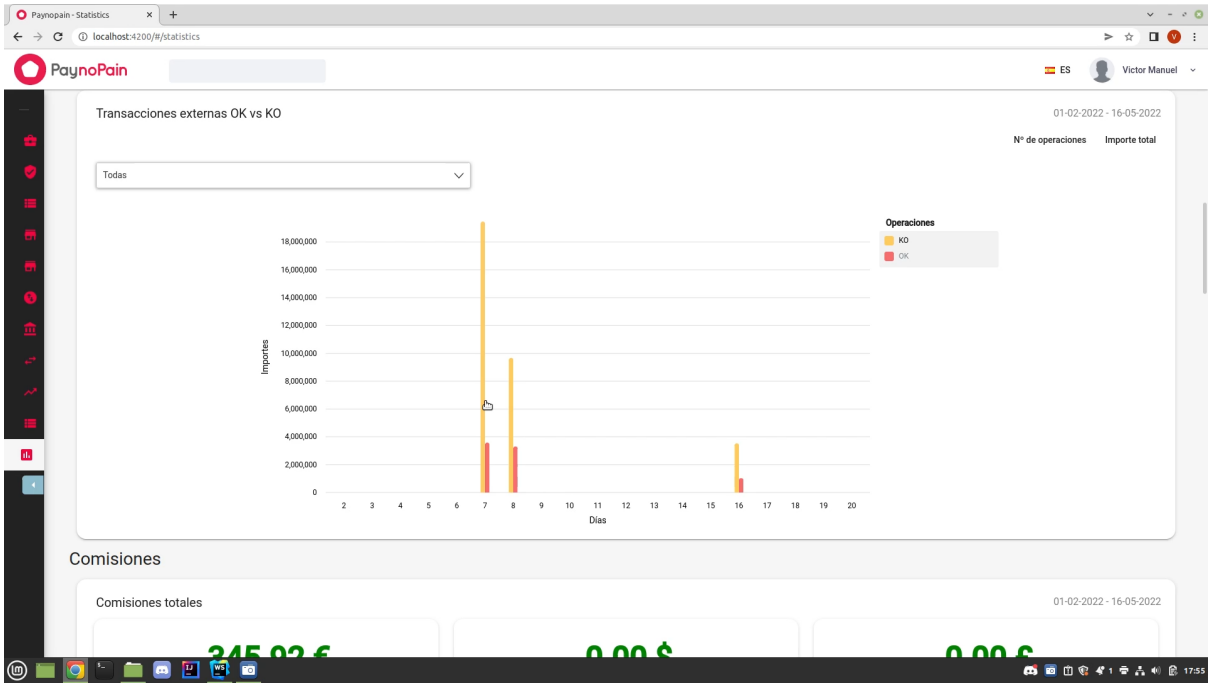

Figura A.2: Captura de pantalla nº 2 del sistema

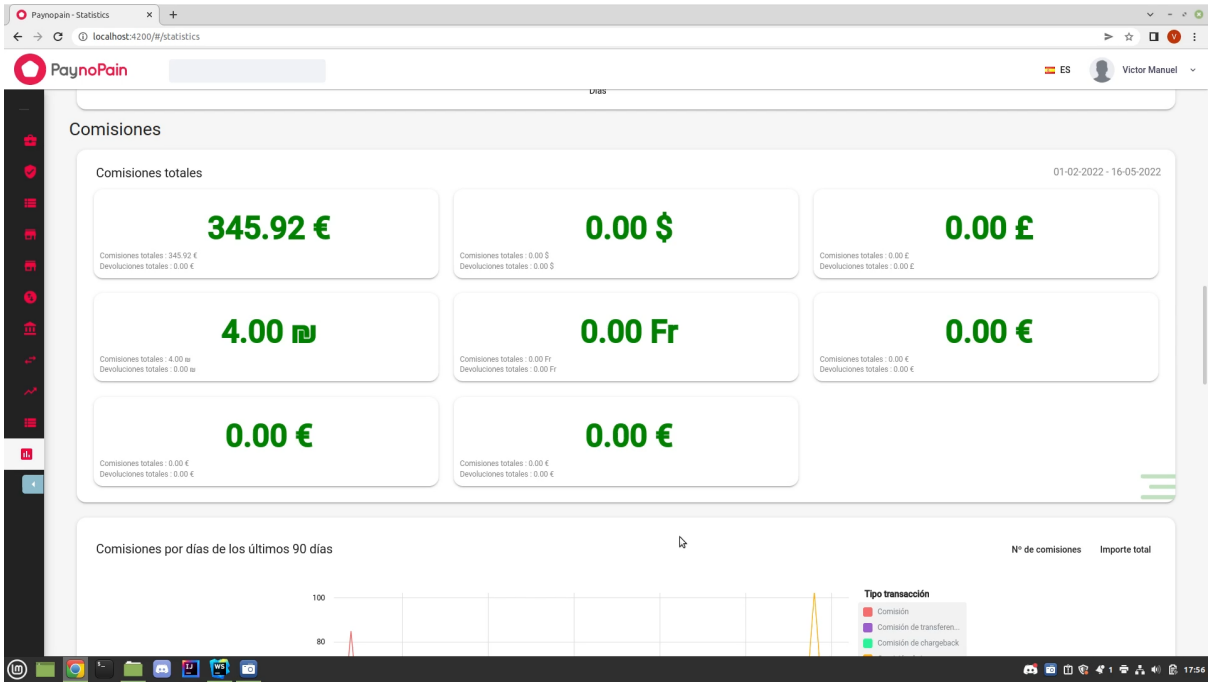

Figura A.3: Captura de pantalla nº 3 del sistema

Figura A.4: Captura de pantalla nº 4 del sistema

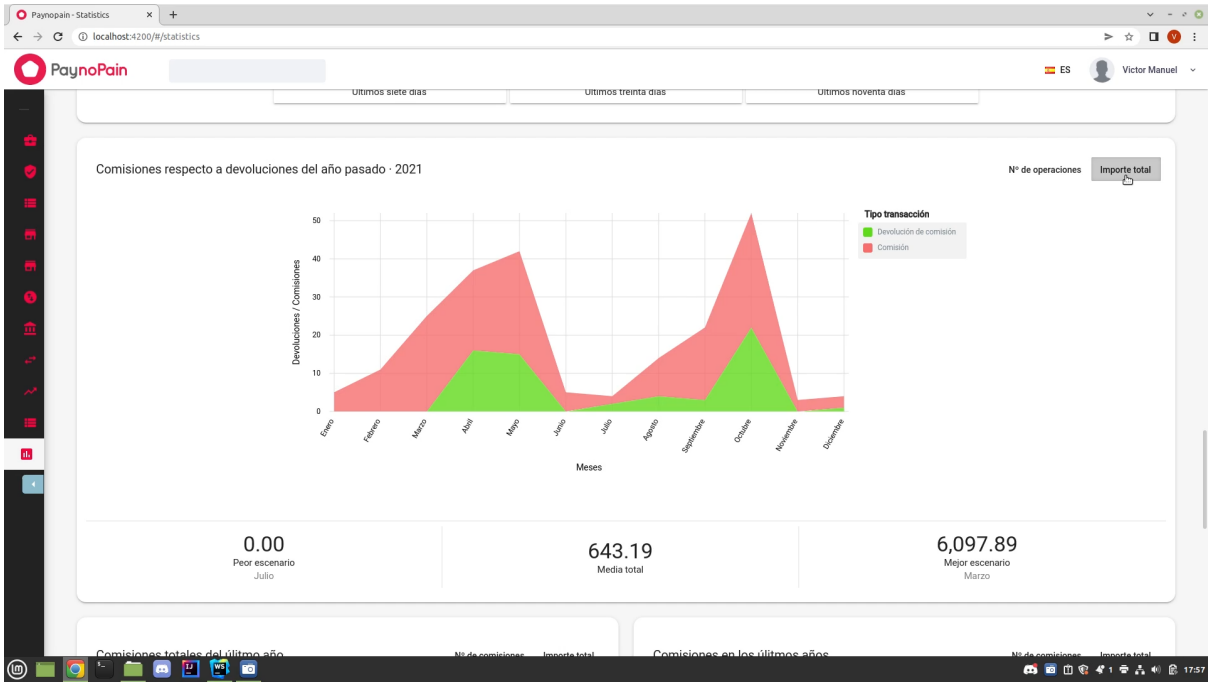

Figura A.5: Captura de pantalla nº 5 del sistema

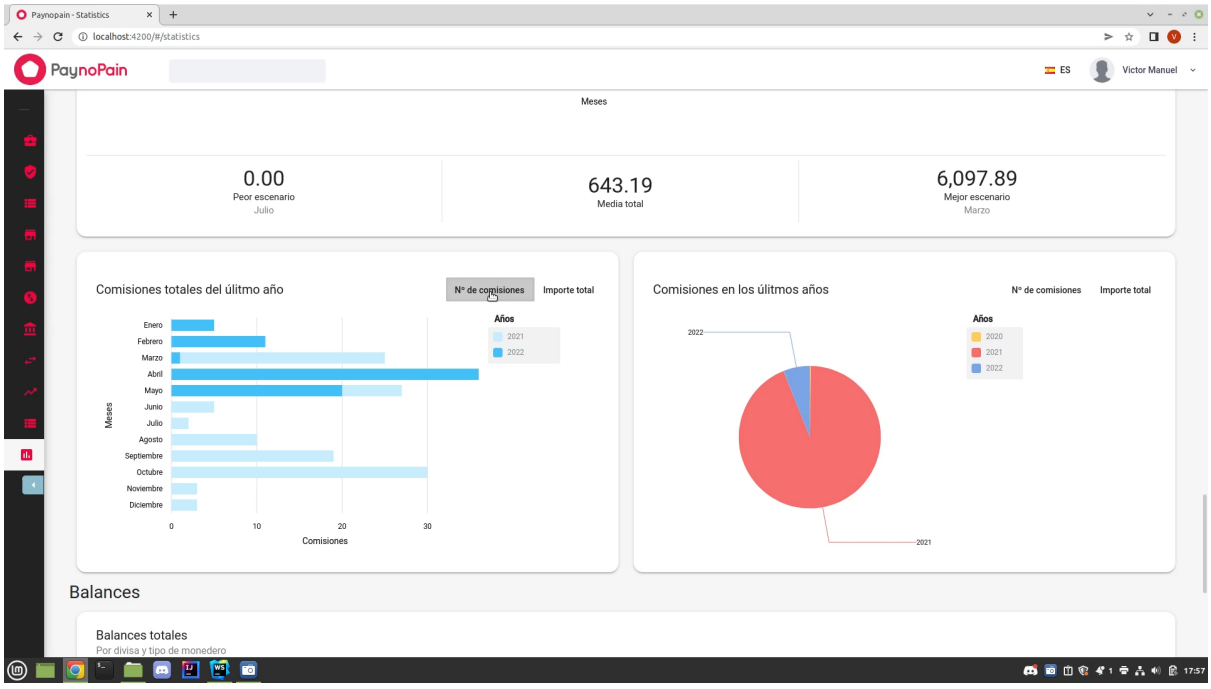

Figura A.6: Captura de pantalla nº 6 del sistema

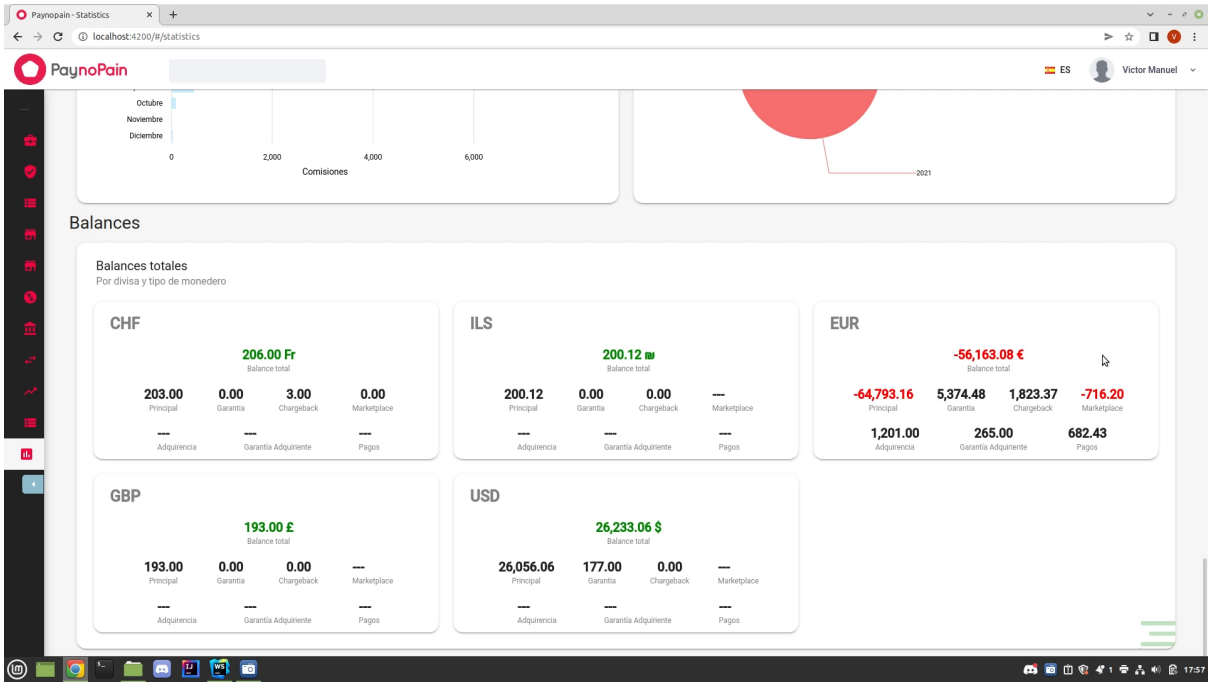

Figura A.7: Captura de pantalla nº 7 del sistema