Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA
Mención en Ingeniería del Software

Desarrollo de microservicios y consumo de recursos a través de una aplicación web para un sistema de gestión de alquileres de plazas de aparcamiento particulares por horas (Sparkea)

Alumno: Carlos Noé Muñoz Bastardo

Tutor: Yania Crespo González-Carvajal

Agradecimientos

A Yania Crespo por su dedicación y ayuda constante en la realización del proyecto, por ser una destacada profesional incansable y ser un ejemplo y un referente para muchos de nosotros. A Pablo, compañero de TFG además de amigo.

Resumen

Sparkea © es un sistema de gestión de alquileres por tiempo, que sirve como solución para plataformas que deseen integrar un modelo de negocio de pago por uso, en base a los servicios proporcionados.

Permite establecer una lógica de negocio por parte del consumidor cliente o gestor del servicio, y personalizar así múltiples aspectos, de manera que permite una gran flexibilidad a la hora de establecer los parámetros sobre el funcionamiento del negocio.

Aunque es posible utilizarlo para distintas instancias, adaptadas a distintas casuísticas y negocios, como pueden ser alquiler de bicicletas, vehículos, espacios de trabajo, etc... en el caso que nos ocupa se ha realizado una implementación para el caso concreto de alquileres de plazas de garaje en comunidades, el núcleo de la aplicación es el mismo con independencia del caso de uso.

Este proyecto ha servido también para el trabajo de fin de grado de mi compañero Pablo Verdejo, en el que ha desarrollado una aplicación cliente que conecta con las funcionalidades de Sparkea, En su caso se desarrolló una aplicación móvil híbrida, que permite ejecutarse en cualquier terminal móvil con independencia de su tecnología de sistema.

En el caso de Sparkea al margen de poderse utilizar con aplicaciones móviles, a través de interfases REST proporcionando al código cliente, se pueden desarrollar webApps que interconecten.

En este trabajo, se ha desarrollado como ejemplo, un cliente que se integra con una placa hardware para controlar y comandar sistemas, como la apertura y cierre de la puerta de un garaje, así como la identificación y el permiso de acceso de los usuarios.

En esta memoria se ha tratado de describir el Sistema en toda su dimensión y partes, desde el análisis, pasando por el diseño, el desarrollo y la integración realizada con el hardware utilizado, así como los servicios implementados y la conexión con la base de datos, y las parametrizaciones del sistema. El usuario gestor del sistema podrá disponer de un manual de usuario, al objeto de generar y parametrizar el sistema. Sparkea además de los múltiples desarrollos propios con los que cuenta, se apoya también en algunas librerías de mercado.

Abstract

Sparkea © is a time-based rental management system that acts as a solution for platforms wishing to integrate a pay-per-use business model grounded on the services provided.

It empowers the consumer or service manager to establish a business logic, thereby personalizing various facets, thus offering great flexibility when setting the parameters governing the business's operation.

While it can be utilized for various instances, tailored to different situations and businesses like bicycle rentals, vehicles, workspaces, etc... in our case, an implementation has been made specifically for the rental of garage spaces in communities; the application's core remains the same regardless of the use case.

This project has also served as the final degree work for my colleague, Pablo Verdejo, in which he developed a client application that interfaces with Sparkea's functionalities. In his scenario, a hybrid mobile application was developed, capable of running on any mobile terminal irrespective of its system technology.

In Sparkea's case, beyond its usability with mobile applications via REST interfaces provided to the client code, interconnecting webApps can be developed.

In this work, a client has been developed as an example that integrates with a hardware board to control and command systems, such as the opening and closing of a garage door, along with the identification and access permission of the users.

This report has attempted to describe the System in all its dimensions and components, from the analysis, through the design, the development, and the integration carried out with the used hardware, along with the implemented services and the connection with the database, and the system's parameterizations. The system manager will have a user manual to create and parameterize the system. Sparkea, in addition to its multiple in-house developments, also relies on several market libraries.

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

1.1. Contexto

Hoy en día uno de los modelos de negocio más destacados que es implantado en las grandes ciudades, es la renta de un servicio por horas. Se encuentran ejemplos desde el alquiler de casas con empresas muy conocidas, hasta el alquiler de patinetes, bicicletas etc. Aun así no todo el nicho de mercado está cubierto, y algunos sectores dentro del alquiler por horas faltan por explorar.

El objetivo de este proyecto ha sido dar la posibilidad a particulares o empresas de alquilar por cortos periodos de tiempo una propiedad que no use, o que no use durante determinadas horas, fortaleciendo la oferta prácticamente inexistente para este concepto, para una creciente demanda vista en el mercado.

El producto propuesto, se configura a través de instancias, y el cliente se encarga de implementar una instancia hacia Sparkea, el cual será el responsable de determinar el alcance final de la aplicación y como se decide tratar determinados aspectos que el servidor deja a libre elección de la instancia. Es decir, que el producto se limita de forma exclusiva a proporcionar la lógica y funcionalidad de rentas por horas, pero es el cliente (aplicación que instancia) el responsable interpretar dicha lógica.

El proyecto en si, se ha desarrollado en distintas tecnologías como Spring, Spring boot, Java, C, C++, SQL, Ionic, Angular, JS, TS y mas... por lo que ha supuesto en algunas tecnologías, posicionarse al principio de esa curva de aprendizaje lenta y costosa, pero conforme avanzaba el tiempo se ha hecho notable un dominio más significativo de la tecnología en cuestión.

Este proyecto, que es idea del estudiante, se ha desarrollado en el contexto del Trabajo de Fin de Grado de los estudios de Ingeniería Informática de la Escuela de Ingeniería Informática de Valladolid.

1.2. Motivación

Este proyecto destaca por su enfoque analítico, su habilidad para gestionar eficientemente los recursos y su integración con las últimas tecnologías.

1. **Problema o necesidad:** En el mercado actual, hay una creciente demanda y una oferta limitada en el sector de los alquileres de plazas de garaje.
2. **Beneficios:** Este proyecto no solo aborda este desequilibrio del mercado, sino que también ofrece la posibilidad de monetizar propiedades desocupadas, proporcionando un recursos económicos extra para los propietarios.
3. **Innovación:** Aunque el concepto no es nuevo, la innovación radica en la adaptabilidad del sistema, que se puede aplicar a diversos sectores, ampliando así su alcance y utilidad.
4. **Interés personal:** Este proyecto representa un reto personal que me propuse. Es un compromiso con la innovación y la resolución de problemas.
5. **Impacto a largo plazo:** A medida que este sistema se establezca y crezca, podríamos ver un aumento en el valor de las propiedades que adopten este modelo, de manera similar a cómo ha sucedido con las viviendas turísticas. Este efecto secundario podría ser de gran beneficio para los propietarios y contribuir al crecimiento económico en general.

Por lo tanto, estoy motivado para seguir adelante con este proyecto, que promete ser una solución eficaz para un problema existente, y que tiene el potencial de aportar beneficios económicos significativos para sus usuarios.

1.3. Objetivos

Se ha desarrollado el producto Sparkea con la finalidad de dar soporte a aplicaciones con interés en la integración de servicios de renta por horas entre varios usuarios.

No se trata de una versión en implantación en un mercado real, si no de un formato con una inclinación más educativa, por lo que algunas funcionalidades no están expuestas por la aplicación, aunque si disponibles. Por ejemplo, el pago con tarjeta real entre las diferentes compañías, o el traspaso de fondos a cuentas bancarias, se hacen a través de tarjetas de crédito que hemos denominado de pruebas. Si se desea pasar a una implementación real, se puede migrar hacia ella, pero debe ajustarse a más sistemas legales. Por ello, esta disponible para su uso un entorno de desarrollo, pero pasar a pro (producción) se deben hacer algunos ajustes legales, sobretodo a nivel de usuarios y pagos.

Los objetivos principales del proyecto Sparkea son los siguientes:

1. Desarrollar un producto para facilitar la implementación de servicios de renta por horas entre usuarios fundamentalmente particulares.

2. Creación de una solución adaptable que no solo esté limitado a la renta de plazas.
3. Proporcionar al menos un entorno de desarrollo y si aplica un entorno de producción.
4. Creación de lógica de usuarios y comunidades que se relacionan entre si a través de pagos.
5. Creación de una pasarela de pagos propia de Sparkea, aunque este apoyada en servicios externos.
6. Creación de un dashboard de usuarios que puedan ver los balances resumen de su actividad (pagos, reservas horas etc).
7. Creación de mapas y apoyos visuales para la mejor visibilidad y comprensión del sistema a nivel ya de usuario. (ver dónde están situados los puntos de reserva etc).
8. Creación de una solución escalable para poder añadir funcionalidades adicionales si la aplicación lo requiere.
9. Creación de un arranque endebido de la aplicación (plug and play).
10. Creación de un hardware que se encargue con interactuar con el servidor y se encargará de abrir y cerrar la propiedad.
11. Creación de una solución con un sistema de monetización a través de un SaaS.
12. Aplicación patrones de diseño y código limpio. De esta manera si las cosas se hacen siempre de la misma manera y siguiendo la misma estructura, es mucho más fácil modificar y añadir cambios.

1.4. Estructura de la memoria

Este documento se estructura de la siguiente forma:

Requisitos y planificación Se describe los requisitos del sistema a implementar, la metodología de trabajo seguida durante el desarrollo del proyecto y como se fundamenta, también se describen los riesgos identificados.

Análisis Se describe el análisis seguido en el proyecto, desde una idea temprana hasta una implementación real como veremos en diseño.

Tecnologías utilizadas Se describe las herramientas con las que se ha desarrollado el proyecto, así como la descripción de algunas consideradas como las importantes.

Arquitectura y Diseño Se describe el sistema de capas seguido en la aplicación, así como muchos de los diagramas presentes en el desarrollo del proyecto. Se insiste en la importancia de creación de un sistema aplicando los principios SOLID con una arquitectura limpia.

Implementación Se describe los paquetes troncales del servidor y la problema principal con cada uno de ellos. También se explica la metodología a seguir con cada uno de ellos.

Seguimiento del proyecto Se describe una ampliación de la metodología ágil nombrada durante el proyecto y cómo se ha seguido con precisión a lo largo del desarrollo del proyecto.

Conclusiones Se describe las conclusiones a las que llegamos una vez acabado el proyecto así como las líneas de trabajo futuras.

Anexo Resumen de enlaces Incluye enlaces de interés sobre el proyecto, como el repositorio de código.

Anexo Manuales: Incluye manuales de mantenimiento, de instalación, despliegue, y de uso.

Capítulo 2

Requisitos y Planificación

En este capítulo se abordará la especificación de requisitos del proyecto y la planificación seguida para abordar el cumplimiento de esos requisitos.

Como se menciona en este documento, se ha escogido una metodología ágil para el desarrollo del proyecto. Concretamente se intentará aproximarse lo máximo posible al marco de referencia de Scrum.

Los requisitos que se deberían abordar en este proyecto se mencionan a continuación:

- El arrendador podrá poner a disposición de los usuarios una propiedad para el uso de los arrendatarios con una compensación económica.
- El arrendador podrá consultar qué propiedades tiene.
- El arrendador podrá añadir, modificar y eliminar tanto propiedades como características de esas propiedades.
- El arrendador podrá solicitar retornar a su cuenta bancaria el dinero ganado por las propiedades alquiladas.
- El arrendador podrá usar dos modos de alquiler de la propiedad, bien que tenga dos estados; libre o ocupado, o bien una reserva por franjas horarias. Ambas opciones de reserva puede llevar consigo la posibilidad de una limitación horaria (en franja horaria) donde no será posible admitir a más clientes. Esta funcionalidad adicional se crea a partir de la necesidad que surge en un usuario de solo poner una plaza de aparcamiento a disposición de los arrendatarios solo en los momentos en el que el arrendatario no se encuentre usándola.
- Se podrá obtener las reservas de una determinada propiedad
- Se podrá crear booking a través de una propiedad
- Se podrá ver las propiedades activas por el usuario

- Se podrá actualizar una reserva.
- Se podrá obtener la reserva que tiene activa una propiedad.
- Se podrá ver si esta ocupada o no una propiedad en concreto.
- Se podrá obtener precio reserva hasta el momento actual o hasta la finalización de la misma.
- Se podrá obtener precio reserva hasta el momento actual o hasta la finalización de la misma a través de su id.
- Se podrá obtener todas las reservas de una determinada propiedad.
- Se podrá obtener el booking a través de su id.
- Se podrá eliminar booking a través de su id.
- Se proporcionará un método de apertura de la puerta con el nombre de logInIdBooking.
- Se proporcionará un método de cierre de la puerta con el nombre de logOutIdBooking.
- Se proporcionará un método de apertura de la puerta a través de RFID con el nombre de logInWithRFID.
- Se proporcionará un método de cierre de la puerta a través de RFID con el nombre de logOutWithRFID.
- Se podrá obtener todas las propiedades.
- Se podrá obtener una propiedad con sus reservas hasta la fecha asociadas a través de su id.
- Se podrá crear una propiedad.
- Se podrá actualizar una propiedad
- Se podrá ver las propiedades pertenecientes a un usuario.
- Se podrán ver propiedades pertenecientes a un usuario con sus booking asociadas.
- El arrendador podrá consultar el saldo que se ha generado por sus propiedades así como las retenciones impuestas por la plataforma.
- Se podrá eliminar una propiedad a través de su id
- Se podrá postear una foto o fotos de una propiedad con formato base 64.
- Se podrá obtener foto o fotos a una propiedad con formato que indica la ruta donde esta decodificada esa imagen.
- Se podrá obtener los ingresos de una determinada propiedad.
- Se podrá crear un cliente para el acceso a pagos de la plataforma implementada que permita la interacción económica con el arrendador.

- Se podrá crear un cuenta para la recepción de pagos de la plataforma implementada que permita la interacción económica con el arrendatario.
- Se podrá añadir una cuenta bancaria a un arrendador para recibir los pagos de los arrendatarios.
- Se podrá añadir una tarjeta de crédito a un arrendador para pagar a los arrendatarios.

Teniendo en cuenta la protección de datos, no todos los usuarios podrán obtener toda la información indicada por lo que tendrá que establecerse por cuestiones de privacidad y seguridad unas restricciones de acceso y una limitación en la publicación de información.

Abordando los requisitos mencionados para el funcionamiento mínimo de la aplicación, se menciona las historias de usuario, como agrupación de varios requisitos del sistema. A continuación se muestran los principales que se han desarrollado.

- Realizar una reserva
- Registrarse como arrendador.
- Registrarse como arrendatario.
- Crear una propiedad.
- Interacción económica entre pares arrendador y arrendatario
- Manejo de pagos entre pares
- Obtención información pagos. (costes pagos, facturas..., reclamaciones).

2.1. Marco de referencia Scrum

En esta sección se comenta qué es Scrum en términos generales y cómo se aplica en este proyecto para centrarnos más en su implementación.

Scrum es un marco de referencia para un modelo de proceso Ágil. Una de las razones por la que se escoge concretamente, es la capacidad de desarrollo en equipo y la duración del proyecto, dejando fuera otras metodologías de planificación de proyectos como métodos de cascada o orientación a prototipos mucho más pesados, donde es más difícil ir hacia atrás en la escala temporal y por tanto se hace mucho más robusto a cambios.

Dado que solo hay dos personas en este proyecto, quizás pierde más sentido la funcionalidad fuerte de trabajo y cooperación en equipo, pero sigue siendo lógica. Se debe aclarar que esto es un proyecto global en el que han trabajado dos personas de forma coordinada, pero cada una ha sido responsable de una parte, y cada una de estas partes se presenta separadamente como distintos trabajos de fin de grado.

En el marco de Scrum, existen varios roles esenciales, cada uno con responsabilidades únicas que son fundamentales para el buen funcionamiento del equipo y la consecución de los

objetivos del proyecto. La interacción eficaz y el cumplimiento de sus respectivas obligaciones son fundamentales para el éxito de la metodología Scrum.

Estos roles incluyen el Propietario del Producto (Product Owner), el Equipo de Desarrollo (Development Team) y el Scrum Master. El Product Owner es el reflejo del cliente y establece cuales son las prioridades para el desarrollo. El Equipo de Desarrollo se encarga de la realización de las tareas de desarrollo del producto, y finalmente, el Scrum Master actúa como un facilitador entre el Product Owner y el Development Team, ayudando a todos a entender y aplicar correctamente Scrum.

Estos tres roles distintos, cada uno con su propia contribución única, forman el equipo Scrum. Esta estructura permite un flujo de trabajo fluido y eficiente, fomenta la colaboración y mejora la capacidad del equipo para responder a los cambios. Además, contribuye al valor general del producto y al éxito de la organización.

El trabajo a realizar se divide en Épicas y las Épicas en historias, estas se agrupan en Sprints (líneas temporales), los cuales se van desarrollando y cerrando conforme se agoten las historias descritas. No todas las historias tienen el mismo peso y no todas las historias se logran abordar en un Sprint. La duración de cada Sprint queda determinada desde el inicio del proyecto y no se modifica a lo largo del mismo. Más adelante se determina la duración del sprint para este proyecto.

El objetivo es incrementar el valor de los entregables Sprint a Sprint a través de iteraciones hasta llegar a un producto final hacia el cliente.

A continuación de desarrolla de forma algo mas detallada los distintos roles que conforman la metodología de scrum:

- Product Owner: Encargado de gestionar el flujo de valor del producto a través del Product Backlog. Se focaliza en la parte de negocio y es el responsable del ROI del proyecto (entregar un valor superior al dinero invertido). Se encarga de hacer una inversión en desarrollo que tiene que producir valor. En otras palabras traduce las necesidades del cliente en conceptos entendibles a los desarrolladores, reflejándolo en un Product Backlog (un desglose del valor del producto en historias priorizados por valor de negocio). De esta manera el cliente ve lo que vale el producto en cada una de sus partes y el peso que puede llegar a tener. Por eso este desglose es tan importante, porque nos da una información más detallada de lo que valen las cosas. Necesito construir una mesa ¿Por qué vale cincuenta dólares? Se puede hacer un desglose de hacer las patas lo que cuesta cada pata montarla etc... También ayuda a un empleado a decir monta las patas.
- Scrum master: Encargado de guiar al equipo para asegurarse de que cumplen con su trabajo y la metodología de trabajo. Amortiza las dificultades que puedan surgir en el proyecto haciendo que lleguen menos abajo en la escala y trabaja con el Product Owner para maximizar su ROI. Esto último es importante en toda proceso, todos trabajan con todos, no hay un rol definido que tenga más peso que otro aunque parezca que no. Todos tienen el mismo peso y cooperan en la formación del entregable a través de ciclos de trabajo.

- Desarrolladores: Es un grupo de técnicos capaces de implementar la funcionalidad que les ha descrito anteriormente a través de historias. Se insiste que aunque sean distintos roles, no es que alguien sea más importante que alguien.

Toda funcionalidad a implementar estaría incompleta si no añadimos un seguimiento del trabajo realizado. En Scrum se hace a través de reuniones diarias, se denominan Daily sprint meeting donde se hace una realimentación diaria de como van las cosas y en qué puntos se está fallando para maximizar el trabajo realizado por todos los miembros del equipo. Se muestra una imagen del proceso general en dimensión tiempo de un desarrollo de un proyecto Scrum.

Figura 2.1: Metodología Scrum

Fuente: `https://nexoprofessional.com/metodologias-agiles-revolucion-forma-trabajar/`

2.1.1. Scrum en Sparkea

Como se ha mencionado anteriormente el marco de referencia que nos hemos basado dentro de nuestro proceso agile, es Scrum. Se ha intentado hacer una aproximación a este, aunque no todo se realiza de igual forma y se difiere en algunas cosas respecto de la orientación inicial.

Existe un capítulo al final de esta memoria llamado Seguimiento del proyecto que pretende explicar de forma clara cómo se ha llevado este proceso a través de la herramienta utilizada, que en este caso es gitlab boards. El seguimiento del proyecto se ha dividido en Sprints, cada uno de ellos de una duración aproximada de un par de semanas (los primeros son de 1 semana), con una duración de unas 30 horas de trabajo. Dicho trabajo se ha controlado a través de las llamadas issues, las cuales adquirían el peso de la tarea en horas y se posicionaban con estados (en el Sprint X, si está o no en progreso o el review o si todavía no está empezada...o quizás si aun no se ha posicionado en ese sprint). Cada issue pertenece a un proyecto en concreto y va enlazada a una Épica. Se explicarán las épicas como aquellas

historias de usuario grandes donde se abordan varias issues. Cada issue está asociada a una o varias personas del proyecto; por regla general suelen ser independientes.

Para formar un puente con lo mencionado anteriormente se podría hacer el símil de que una historia se aborda a través de la realización de varias issues generalmente.

En la finalización de un Sprint o cada cierto tiempo que se requiriese iteración nos reuníamos mi compañero Pablo y yo para abordar los problemas que pudiesen surgir de ambas partes o necesidades de interconexión entre varios elementos que interactuasen con la aplicación. Las preguntas mas frecuentes que se formulaban en tales reuniones fueron algunas como, necesito esto, ¿Como me lo puedes suministrar? ¿Cuál es la comunicación y de qué forma debemos establecer ese dialogo entre ambas partes? La duración de esas reuniones no superaban la hora, pero dependían de la época y del día. A parte de las reuniones realizadas por Sprint entre los integrantes del equipo de desarrollo, cada martes con la responsable del proyecto Yania Crespo, abordamos los problemas finales que podíamos esperar y realizábamos esa review donde exponíamos todos a todos los avances generados en dicho sprint y aquellas cosas que se debía dejar para más adelante (otros Sprints). Este tipo de reuniones que se apoyan bajo una metodología ágil permiten la interacción de todos con todos y nadie por encima de nadie lo que ha potenciado la capacidad de trabajo en equipo, avanzar mas rápidos en los problemas expuestos y sentirse mucho más integrado en el proyecto.

A continuación se mostrará una lista con las épicas que se han escogido para este proyecto. También se mostrará una imagen total de todas las issues de todo el proyecto (el proyecto está formado por varios proyectos cada de ellos con sus issues). Se hace de esta manera para mostrar en una panorámica el seguimiento de todo el proyecto hasta ahora.

Figura 2.2: Lista de épicas

A continuación se muestra una vista panorámica de las issues pertenecientes a alguna épica asociada de las anteriores, donde ya están las issues asignadas a milestones diferentes (Sprints). Aunque se muestren dos milestones a la vez, no quiere decir que se hayan hecho en paralelo, los sprint se realizan en serie. La vista de este tablero muestra todos los Sprints realizados hasta ahora de ambos participantes del proyecto.

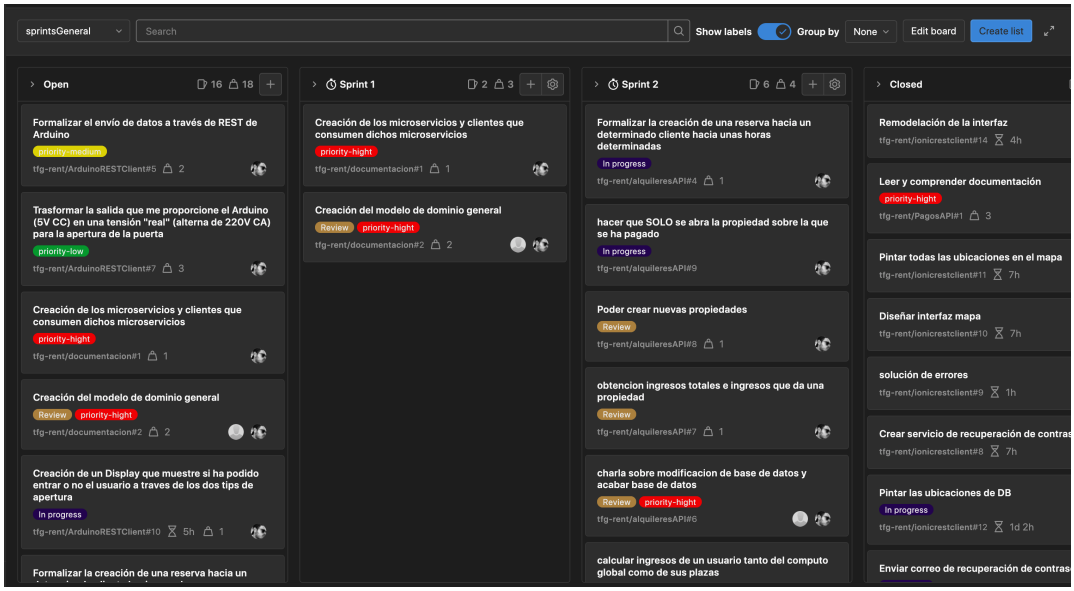


Figura 2.3: Vista de ejemplo de tableros de los primeros Sprints

2.1.2. Análisis de Riesgos

Se detallan una serie de riesgos que podrían afectar al desarrollo y mantenimiento del proyecto.

Se ha asignado una numeración que se aplicará para valorar tanto la probabilidad del riesgo como su impacto.

Valor	Probabilidades	Impacto del Riesgo
1	Muy Bajo	Muy leve
2	Bajo	Leve
3	Medio	Medio
4	Alto	Poco Critico
5	Muy Alto	Critico

Tabla 2.1: Niveles de riesgos y probabilidades

A continuación, se presentarán los distintos riesgos identificados para este proyecto dentro de un marco específico creado para tal propósito. Este marco incluye varios elementos clave: el riesgo en sí, la probabilidad de que ocurra, su posible impacto, un plan de prevención para mitigar su efecto y, finalmente, un plan de contingencia en caso de que el riesgo se materialice. Cada uno de estos elementos proporcionará una visión completa de los posibles desafíos que se podría enfrentar en el proyecto y cómo se planea abordarlos.

Este riesgo se ha materializado de forma habitual a lo largo del desarrollo del proyecto. Se trata del desconocimiento de una tecnología y su curva de aprendizaje, algo muy común en

nuestro sector y supone un riesgo fundamental en el desarrollo de un proyecto. Podría decirse que en toda gestión de proyectos de desarrollo de software se contempla esta posibilidad. Este riesgo está relacionado con el que se va a presentar a continuación que se refiere a la fuga de personal de la empresa.

tipo	Accion temporal
Riesgo	Mayor desconocimiento de las tecnologías
Probabilidad	3
Impacto	3
Plan de Prevención	Basar las estimaciones de tiempo en estimaciones de proyectos pasados con personas de conocimiento similar
Plan de Contingencia	Aumentar el número de horas dedicado al Sprint

Tabla 2.2: Riesgo Mayor desconocimiento de las tecnologías

El siguiente riesgo a comentar es la fuga de personal de la empresa. Si una persona tiene conocimiento y no es el equipo donde reside el conocimiento, si esa persona abandona la empresa, se va la información con ella. Por ello, se ha priorizado en generar buena documentación de los artefactos desarrollados y evitar que los conocimientos sean exclusivos de personas especificas, favoreciendo la construcción del equipo.

En los tipos de procesos no ágiles como la iteración suelen tender más hacia una mayor distanciación entre las responsabilidades de cada uno, en cambio con Scrum como se ha comentado anteriormente refuerza estos conceptos y equipara el poder de cada uno de los roles de cada uno en el proyecto fortaleciendo el equipo y derivando una funcionalidad en cada uno de ellos. De esta forma, la pérdida de alguno de los integrantes del grupo no hace una pérdida o fracaso total del proyecto, sino un ligero contratiempo. (Utilizar metodologías ágiles logra reducir el impacto del riesgo en la fuga de personal aunque no la elimina). Es conveniente decir que la fuga de personal no solo trae consigo el impacto directo sobre la fuga de conocimiento de la empresa, sino que también trae fenómenos evidentes como las indemnizaciones, pagos, etc que hacen que el impacto del riesgo suba aun algún punto más.

tipo	Accion temporal
Riesgo	Fuga de personal
Probabilidad	3
Impacto	3
Plan de Prevención	Fortalecer trabajo en equipo
Plan de Contingencia	Formar al equipo

Tabla 2.3: Riesgo Fuga de personal

Otro de los riesgos muy visibles y evidentes por otra parte que se han manifestado no solo a lo largo del proyecto, si no en todo proyecto (incluso fuera del desarrollo de software), es la estimación incorrecta de las tareas. Principalmente se suele tener una tendencia bajista a la hora de ofrecer una estimación de una tarea, por lo que tanto el product owner, scrum master y desarrolladores deben dejar aun más margen de error en el planteamiento del peso/-tiempo de los entregables hacia el producto. Se unen los riesgos comentados anteriormente que irían en cadena hacia este, como el desconocimiento de las tecnologías o que se marche

la persona que tenía el conocimiento o por ejemplo puede darse el caso de tener una escasa documentación.

tipo	Acción temporal
Riesgo	Estimación incorrecta horas tareas
Probabilidad	4
Impacto	2
Plan de Prevención	aumentar peso estimado
Plan de Contingencia	Ampliar o dejar para el siguiente Sprint

Tabla 2.4: Riesgo Fuga de personal

Otro riesgo principal a comentar son los errores de comunicación que existen entre las dos partes de la aplicación pudiéndose dar fallos críticos en el sistema. Este riesgo sobre todo se ha manifestado por el consumo de recursos a través de las interfaces proporcionadas por el gestor de Sparkea. La comunicación entre interfaces que ofrece Sparkea y clientes enganchados, debe ser muy precisa y no vale con pasarle parámetros como ejemplo en cualquier texto o formato. Por ello se ha decidido hacer una buena documentación sobre los servicios proporcionados, y cómo se han de usar ejemplificando todos los casos a través de una parecida batería de pruebas. Esto a parte de ayudar a cometer menos errores en la comunicación, favorece a disminuir el impacto del riesgo anterior, la fuga de personal que conllevaba en mayor o menor medida a la fuga de personal de la empresa.

tipo	Acción temporal
Riesgo	Comunicación incorrecta entre partes
Probabilidad	4
Impacto	2
Plan de Prevención	documentación
Plan de Contingencia	Reunión y aclaración

Tabla 2.5: Riesgo Comunicación incorrecta entre partes

Otro de los riesgos, menos probables pero no imposibles y que conllevarían un impacto alto, es la caída del servidor donde esta expuesto el servicio. Actualmente se ha decidido la contratación de un Cloud Computing, un SSAS donde a través de un pago mensual, se puede disfrutar de un pc en una localización determinada. Lo que da lugar a una derivación de responsabilidades, aunque la infraestructura la tengamos que montar nosotros. A parte de ello, tenemos copias de seguridad en distintos sitios y en entornos locales para enfrentarnos a caídas más fuertes.

Otro de los riesgos a comentar son los costes legales de implantación del servicio. Soportar un cliente sobre Sparkea que haga uso del servicio, incrementa los costes de permisos, licencias etc (aparte de la de Sparkea). En el escenario propuesto con el cliente móvil en el uso del sistema para aplicación de gestión de alquileres por horas, entra de lleno la solicitud al ayuntamiento o comunidad de una licencia que tendrá un determinado coste (quizás no nos dejen implantarlo por oposición de vecinos/comunidad), aunque lo más seguro es que se permita pagando unas tasas determinadas. Se considera un riesgo con una posibilidad pequeña, pero si ocurre, el impacto sería de lleno.

tipo	Acción temporal
Riesgo	Caída de servidor
Probabilidad	1
Impacto	4
Plan de Prevención	documentación
Plan de Contingencia	restauración de copias de seguridad y tener una integración automática con el sistema de versiones.

Tabla 2.6: Riesgo Caída de servidor

tipo	Acción temporal
Riesgo	Pago licencias
Probabilidad	1
Impacto	4
Plan de Prevención	reuniones
Plan de Contingencia	devolver el software.

Tabla 2.7: Riesgo Pago licencias

En definitiva como podemos observar hay diferentes tipos de grados de probabilidad e impacto del riesgo que debemos de convivir con ellos y reducirlos lo máximo posible, (y generalmente es añadiendo costes al entregable del producto). Es importante hacer un análisis de riesgos antes de comenzar un proyecto, ya que trata de una dimensión critica a la hora de estimar el peso de las tareas y del coste del proyecto. Mitigar un riesgo es muy difícil, pero convivir con él a través de una reducción de en la escala de probabilidad e impacto, es posible. A continuación se muestra una imagen que escala los distintos niveles de ambas dimensiones, y que en el caso de existir el riesgo debemos llevarlo hacia el punto de origen lo máximo posible.

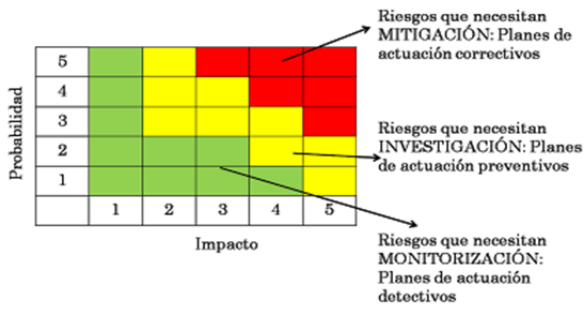


Figura 2.4: Metodología Scrum

Fuente: `https://www.calidadytecnologia.com/2014/02/en-mi-proyecto-solo-hemos-detectado-un.html`

La figura ilustra los niveles de probabilidad e impacto de los riesgos, además de las acciones apropiadas a tomar en cada zona de riesgo.

Si nos encontramos en una zona donde tanto la probabilidad como el impacto del riesgo son bajos, la estrategia apropiada es monitorizar la situación y asegurarnos de que no se intensifique. En caso de que nos situemos en un nivel medio en ambas dimensiones, es necesario llevar a cabo una investigación para comprender mejor el riesgo y considerar medidas preventivas. Sin embargo, si nos encontramos en una zona de alto riesgo, donde tanto la probabilidad como el impacto son altos, es esencial tomar medidas de mitigación para prevenir problemas críticos que podrían poner en peligro el éxito del proyecto.

Capítulo 3

Análisis

Una parte del tiempo que se dedica en el desarrollo del proyecto se hace hacia el campo del análisis. Como se está trabajando en un marco de metodología ágil, no se trata de hacer un modelo de análisis inicial e implementarlo, sino de plantear un análisis inicial e ir iterando y refinando ese razonamiento conforme avanza el proyecto. No obstante, es importante señalar que, aunque la iteración y refinamiento son fundamentales en el marco ágil, también es crucial mantener una visión clara y consistente del proyecto para garantizar que se alcancen los objetivos finales. Se ha creado un modelo de dominio inicial y posteriormente se ha ido refinando conforme se iban necesitando unas cosas u otras.
Con el fin de obtener una mejor encapsulación y derivar las responsabilidades correspondientes a cada entorno, se ha descompuesto el sistema inicial en una arquitectura similar a micro servicios de tal manera que puedan funcionar de forma autónoma y tengan la capacidad de ser más escalables. Todo ello conforma el core del producto. En la Figura 3.1 se muestra una aproximación de un modelo de dominio en una etapa muy inicial.

Según avanzan los sprint se va puliendo cada vez más el modelo de dominio y el cómo derivar y a quien las funcionalidades necesarias. Conceptualmente diríamos que existen varias funcionalidades con capacidad de ir separadas en paquetes diferentes; pagos, usuarios, accesos y reservas se encapsulan de forma independiente. Cubrir una funcionalidad en el sistema y hacerlo más escalable y atómico es uno de los objetivos a perseguir, sin embargo la funcionalidad que ofrecerían algunos de estos serían muy acotadas. Por ello debemos de fusionar algunas funcionalidades que bien conceptualmente están separadas, la lógica nos dicen que deben estar juntas.

Este dominio inicial es común a ambos trabajos de fin de grado realizados en un proyecto conjunto. Posteriormente la subdivisión hace que en cada uno de los TFGS se responsabilice de una parte y su funcionalidad asociada.

Por ello se decide que la descomposición inicial en paquetes debe tener la siguiente distribución:

- Pagos

- Reservas (donde también encontraremos la lógica de apertura; accesos).
- Usuarios (donde encontraremos la lógica de registro y acceso, sesiones etc).
- Chats (donde se aplicara la lógica de comunicación de los usuarios).
- Otros. (Se podrá añadir una funcionalidad adicional).

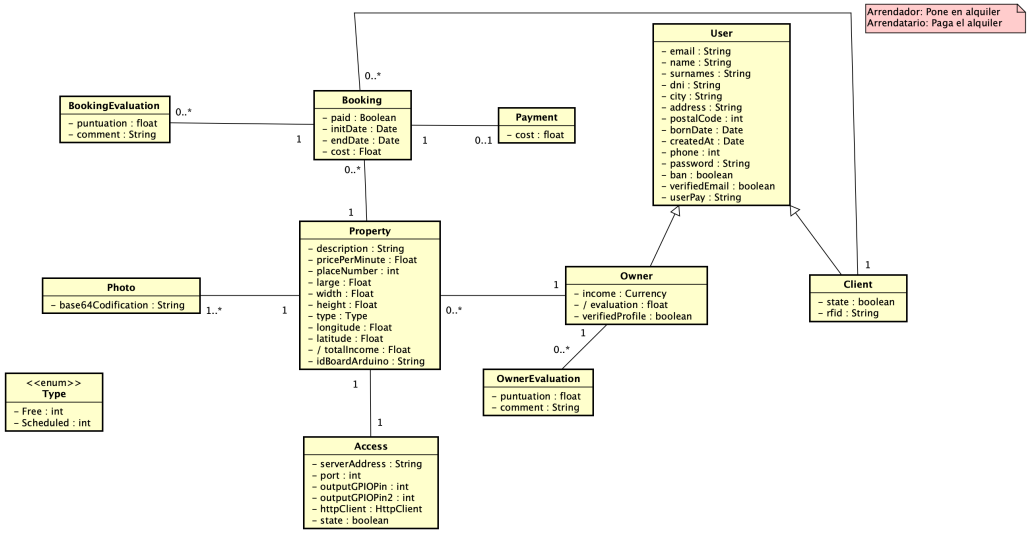


Figura 3.1: Modelo de dominio inicial

Dentro de cada paquete se seguirá la estructura de capas (arquitectura) comentado en la sección de Arquitectura y diseño, siguiendo en todo momento un orden lógico y un sistema de capas claro en toda la aplicación (Capa Presentación, Capa negocio, Capa Persistencia). En el conjunto del proyecto, se sigue la arquitectura de paquetes maven. A continuación se destallan algunos de los clientes externos que se enganchan y forman parte de Sparkea. Se presenta el ejemplo de ionic como la instancia específica consumidora.

- Arduino Client: Cliente que se encarga de de abrir una propiedad bajo unas determinadas condiciones impuestas
- Ionic: Se trata de una aplicación móvil desarrollado en web y ejecutado sobre un webview. Sería la instancia especifica a implementar en Sparkea. Se ha desarrollado sobre una capa de Angular para facilitar su desarrollo. Más adelante se integrará con Cordova o Capacitor para la ejecución de la app en android e IOS (o el SO que se quiera ejecutar).
- Angular web: Se tratará de una aplicación web angular para el desarrollo de una interfaz web donde muestre al usuario owner (arrendador) toda la posible información relevante sobre su cuenta. Informes de pagos, cuentas y más. Entre ellos, información sobre ingresos, posibles gráficos, clientes asociados. Este servicio no está implementado aún.

Capítulo 4

Tecnologías utilizadas

4.1. Tecnologías generales del proyecto

4.1.1. Componentes Sparkea

En este capítulo se presenta un resumen de las tecnologías utilizadas para la gestión, el desarrollo y la implementación del proyecto.

La aplicación cliente (instancia de Sparkea) está construida bajo el framework de Ionic, y esta, bajo Angular (y no angular JS). Se trata de un desarrollo híbrido basado en programación web y ejecutada en un entorno móvil. Esta tecnología da la posibilidad de desarrollar para dos entornos diferentes escribiéndolo una sola vez. Finalmente se ha ejecutado solo en el entorno de Android. Mi compañero Pablo Verdejo se ha encargado de esa parte consumidora.

Aunque existe un cliente como parte consumidora, no se trata del único. Otro cliente a encajar en el puzle, como parte del ecosistema de Sparkea, es el cliente de Arduino, el cual también consume datos del core de Sparkea Server y es el encargado de abrir y cerrar la puerta, cerradura cuando se den las circunstancias necesarias. En este caso, dicho cliente es un componente hardware, que a su vez hace el uso de antenas receptoras y más elementos. Se destaca la complejidad añadida de estos sistemas empotrados por la novedad del lenguaje y la forma de trabajar.

Para este cliente, se ha optado por utilizar un sistema ESP8266 debido a las características específicas del problema que se intenta resolver. Si el problema hubiera implicado la lectura de matrículas, por ejemplo, probablemente habríamos considerado utilizar una Raspberry Pi. La implementación de esta parte del proyecto se ha realizado en los lenguajes de programación C/C++.

4.1.2. Core Sparkea

En el core de Sparkea se ha desarrollado en su totalidad con Java 11+ ,y en vez de utilizar un tomcat como servidor de aplicaciones, utilizamos la solución Spring Boot, que tiene un tomcat embebido, el cual actúa como contenedor de Servlets donde se encuentra la aplicación. (La mayoría de las aplicaciones Spring Boot necesitan una configuración mínima de Spring, y facilita la creación de aplicaciones basadas en Spring independientes y de grado de producción que puede -Ejecutar dando a un botón-). Todos los servicios están disponibles en el puerto 808X.
La posibilidad de diferenciar servicios y exponerlos en diferentes puertos llevo a contemplar el uso de un apache por delante del tomcat, que serviría de frontend para el acceso a la aplicación en el caso de que tuviéramos que habilitar varios puertos en el consumo de datos. Se trataría de un proxy reverso al tomcat que redirigiría las peticiones. (Recordemos que un apache simple puede servir paginas estáticas y hacer mas funcionalidades, mientras que un apache tomcat sirve aplicaciones web dinámicas), aunque finalmente por simplicidad lo hemos montado sin ello. El uso de un servidor Apache frente a Tomcat habría proporcionado un cierto nivel de desacoplamiento y flexibilidad para el sistema. Además, el uso de un proxy inverso podría ofrecer ventajas adicionales, como proporcionar una capa adicional de seguridad, equilibrar la carga entre varios servidores back-end y ofrecer una mayor flexibilidad para el mantenimiento y la actualización del sistema sin interrumpir el servicio.

Aunque no se haya separado en puertos los tipos de servicios si se ha hecho a nivel de paquetes, dejando de una forma clara y aislada cada funcionalidad. Antiguamente (hace años) se agrupaban las clases entorno a capas y actualmente se suele hacer una agrupación entorno a funcionalidad, pero no es una cuestión de estar mal o bien si no una cuestión de diseño.

4.1.3. Entornos de desarrollo

Respecto a los entornos típicos de desarrollo, existe uno que es el de desarrollo, pero de forma sencilla nos podríamos montar un entorno de producción a nivel de database, a nivel de ficheros de configuración, a nivel de pagos etc.

4.1.4. Alojamiento

Tanto la aplicación general como la base de datos se encuentran expuestas en una máquina no custom, contratada con la empresa de digital-ocean [4] (más información en Sec. B.2) y a través de un SaaS, con un coste de unos diez euros mensuales, permite el acceso y consumo de los recursos proporcionados. La máquina se encuentra ubicada en Londres, lo que permite una latencia aceptable.
El coste de alquilar una database ya montada era superior a la de alquilar una máquina, por lo que se decidió esto último y se montó una database dentro, consiguiendo ahorrar más dinero. mientras se desarrolla este TFG, se mantiene el servicio levantado.

A continuación se detallan algunos detalles importantes del servidor, tales como la imagen que está montada, la región en la que está situada, y la ip publica a donde nos tendríamos que conectar.

image	region	ip
Ubuntu 20.04 (LTS) x64	Londres	142.93.35.77

4.1.5. Gestores de paquetes y dependencias

Como se ha mencionado anteriormente, se hace el uso de Spring boot [2]. Pero ¿Qué significa esto y por qué no se ha decidido la construcción de una aplicación Java EE pura?

El uso de Spring toma cada vez más fuerza para evitar la complejidad que se tenía a la hora de generar proyectos JavaEE, por ejemplo la formación de los EJBs resultaban mas engorrosos, y la forma de desplegar el proyecto era más - compleja - (generalmente implica montar el servidor de aplicaciones glassfish tomcat, etc, como mínimo, generar el war de la aplicación desplegarlo en el servidor de aplicaciones, etc). Por ello se llega a la idea de usar Spring boot - dar a un botón - y que funcione. En definitiva, no es más que subir un nivel en la escala de abstracción de un sistema complejo, lo que permite aumentar la funcionalidad (centrarse más en ella), y simplificar lo que esté a más bajo nivel.

En definitiva Spring boot es una herramienta que nos permite crear y desplegar un aplicación Spring de manera rápida evitando tener que realizar manualmente ciertas configuraciones. En muchas ocasiones se piensa que Spring boot y Spring framework es lo mismo, y no del todo. Realmente Spring Boot es una herramienta que nos permite crear un proyecto como con Spring Framework, pero Spring Boot elimina configuraciones algo complejas para desplegar el proyecto. Entre otras destacan las siguientes:

- Configuraciones automáticas,
- Ofrece un servidor (Apache Tomcat endebido) incorporado para burlar esa complejidad en el deployment de una aplicación Spring,
- Dependencias iniciales,
- Ciclo de vida de una aplicación Spring

Como se comentó anteriormente, respecto al sistema de versiones se usa git, y respecto del gestor de dependencias se usa maven [1] . Se podría haber hecho uso de Gradle, pero se decidió maven por la experiencia que ya se tenía con este gestor. En todo el proyecto se sigue las recomendaciones y sistemas de carpetas que dicta maven. La forma general del proyecto que se genera automáticamente al usar maven es:

- **src/main/java:** Este directorio es donde se almacenan todas las clases fuente de Java. Aquí es donde encontrará todos los archivos `.java` que componen la lógica del programa, incluyendo las clases, interfaces y enumeraciones.

- **src/main/resources:** Este es el lugar donde se almacenan los archivos de configuración, incluyendo los archivos específicos de Spring. Estos archivos de configuración pueden incluir archivos XML, propiedades y archivos de configuración YAML, entre otros. Estos archivos son esenciales para definir cómo funciona la aplicación y cómo se comporta en diferentes entornos.
- **src/test/java:** Este directorio es donde se sitúan las clases de prueba para las clases en `/main`. Estas pruebas son cruciales para verificar que el código funciona como se espera y ayuda a detectar y prevenir errores en la lógica del programa.
- **src/test/resources:** En esta carpeta se guardan los recursos que utilizan las pruebas. Esto puede incluir datos de prueba, archivos de configuración para las pruebas y otros recursos que sean necesarios para llevar a cabo las pruebas de las clases en `/main`.

Como id de desarrollo se ha usado IntelliJ, como gestor de base de datos DBeaber, como cliente de peticiones REST Postman, como editor arduino, arduinoIDE, como sistema de versiones git, como cliente de git kraken.

4.1.6. Stripe

Uno de los paquetes del core más importantes es la pasarela de pagos de Sparkea. Esta pasarela se apoya en la tecnología de Stripe, con el objetivo de garantizar la seguridad en el proceso de pago. [3]

Stripe es un servicio de procesamiento e intercambio de pagos en línea que permite intercambiar pagos entre usuarios mediante tarjetas de crédito y cuentas bancarias.(Actualmente se esta trabajando desde la compañía para el pago a través de criptodivisas).

Al utilizar Stripe, Sparkea se beneficia de un sistema de pagos seguro, confiable y escalable, que se ajusta a las necesidades de crecimiento del proyecto y cumple con los estándares de seguridad y de cumplimiento regulatorio.

Capítulo 5

Arquitectura y Diseño

5.1. Principios SOLID

Me gustaría presentar el -cómo- se ha trabajado, y no tanto el -qué- se ha trabajado a lo largo de esta sección. Ser programador, analista programador o desarrollador o ingeniero difiere en varios conceptos, aunque uno de ellos sin duda es la manera de hacer el software y llegar a conclusiones de cómo construir, es decir, derivar un modelo a través de un análisis.

La aplicación de los principios SOLID es un ejemplo de realizar buenas practicas en un desarrollo de software. A continuación se detallan las características de este principio que se han incluido en el proyecto:

- Principio de Responsabilidad Única

  "A class should have one, and only one, reason to change."

  Este principio se refuerza con la idea del TDD (desarrollo de test). Si se quiere comprobar la funcionalidad de un clase y la funcionalidad de esta está encapsulada en la propia clase, entonces se está aplicando de manera correcta el principio. Es decir, separar aquellas cosas que cambian por razones diferentes. Se considera que este es el principio más importante de SOLID.

- Principio de abierto-cerrado

  "Gather together the things that change for the same reasons. Separate those things that change for different reasons"

  Reúne las cosas que cambian por las mismas razones. Separa aquellas que cambian por razones diferentes.

- Principio de Sustitución de Liskov

  "Derived classes must be substitutable for their base classes."

  Viene con el uso de la herencia, el polimorfismo y ligadura dinámica, pilares de la OO muy conocidos ya por todos nosotros.

- Principio de Segregación de la Interfaz (ISP)

  “Make fine grained interfaces that are client specific.”

  implementa solo los métodos que vayas a necesitar en una interfaz, no sobrecargues. La idea de ser más generalista hace de un desarrollo más costoso donde el "por si se hace"queda muy lejos en la mayoría de los casos.

- Principio de Inversión de Dependencias

  “Depend on abstractions, not on concretions.”

  Se debe depender de abstracciones y no de clases concretas. Este principio puede dar mucho juego en cualquier entorno, donde el elemento juega como una caja negra con la funcionalidad que ofrece. Simplemente llama a ello, y nosotros te devolvemos el resultado. Este principio se aplica mucho a los frameworks donde se aplica el yo te llamo a ti. La llamada se hace a través de los metadatos de la clase puedo realizar desde allí la inyección.

5.2. Sistema de Capas en el servidor

Las APIs utilizan como gestor de paquetes maven, y por lo tanto siguen también una estructura de directorios maven. Dentro de esto, se ha decidido la realización de una arquitectura en 3 capas bien diferenciadas.

- La Capa 1 de negocio, se trata de la capa más externa de todas donde se encuentran los handlers (capturadores), es decir, los endpoints. Como se está usando Spring Boot se ha optado por el uso de Controladores MVC que ofrece Spring en lugar de los Servlets tradicionales, aunque en realidad se trate de lo mismo (Siendo un poco más purista se debe saber que realmente actúan en diferentes niveles dentro de una aplicación Java.).

  J2EE define el concepto de Servlets y cada servidor de aplicaciones Java, que nos permite desplegar aplicaciones web dinámicas, está diseñado para ejecutar Servlets (al igual que muchos otros artefactos que son familiares como EJB, JSP, JMS, JTA...). Por el contrario, un controlador de Spring MVC es una librería construida sobre un servlet (de J2EE) que proporciona mayor facilidad y funcionalidad a la hora de capturar peticiones externas que llegan a la API.

  Toda esta capa, facilita que las peticiones que llegan a través de REST (Representational State Transfer). REST es un estilo arquitectónico utilizado en el diseño en aplicaciones web. REST se basa en principios y restricciones que facilitan la creación de aplicaciones haciéndolas escalables, eficientes y fáciles de mantener. En resumen, REST es la manera de interactuar con la interfaz que se expone fuera hacia el cliente.

  Aunque se haya decidido la exposición de servicios a través de REST pero podríamos haberlo hecho a través de RMI (objetos remotos) o SOAP o similar. Las razones del uso de REST son muchas, pero entre ellas:

  1. Protocolo cliente-servidor

2. Stateless (ausencia de estado).
3. Cacheable (cacheado de recursos con implementación sistema e-tag) etc.
4. Interfaz uniforme (todo lo hacemos de la misma manera)
5. Sistema de capas

El modelo de acceso a las clases entre capas que se utiliza es muy estricto, como se ha descrito anteriormente, nos hemos de diferenciar de un programador base y aplicar labores de analisis y diseño correctas. Siendo puristas, y siguiendo un flujo lógico, los endpoints, contienen acceso a los DAOS, y los DAOS aprovechan los beans (elementos del dominio) para completar la funcionalidad. Así que esta última capa mas externa accede a la última mas profunda (o la anterior) con el objetivo de recuperar datos y esta ultima, se encarga de devolver los datos a través de la capa intermedia.

Respecto a la serialización que se hace de los datos que llegan al endpoint, no se hace una serialización customizada a través de un JsonAdapter, porque no se necesita, así que se ha dejado la implementación por defecto. Si se requiriese una serialización determinada se tendría que hacer una serialización customizada.

- La Capa 2, contiene los elementos del dominio, también llamados los beans. Son objetos planos (POJOS) encargados de modelar aquellas características que sean necesarias para la aplicación, y proporcionar la funcionalidad necesaria dentro del ámbito del propio objeto.
  Esto ultimo requiere una aclaración. Desde el punto de vista de Java 11 los objetos planos (y hasta Java 17) son beans, aunque a partir de esa frontera se separa ese concepto, con el de la clase con funcionalidad. El concepto de bean se asemeja a un Struct C o un simple Wrapped. Cuando hablamos del concepto de clase, es un POJO cobrando cierta funcionalidad, es decir rellena ese bean con funcionalidad. No se debe confundir ambos conceptos aunque estén agrupados como los mismos. Por regla general no se suele dar funcionalidad a estos objetos siguiendo esta estructura, y se suele derivar esa responsabilidad a los DAOS. Si necesitamos de un endpoint una información referente al modelo, solo usaremos como un contenedor a este objeto y se ejecutará alguna operación que se requiera en este contenedor. Los objetos del dominio nunca deben llamar a los DAOS, sino los endpoints a los DAOS y estos últimos utilizan los contenedores para almacenar y generar información.

- La Capa 3, DAOS, contiene el acceso a la persistencia. Usa los elementos del dominio para la construir información y computar datos para devolvérselos a la capa de negocio (1). Dentro de esta capa existen dos clases bien diferenciadas; la que accede a los datos (la que se relaciona directamente con la database) y se llama el daoXXXimpl, y la que aplica la lógica de construcción de esos datos, y se llama el daoXXX. De tal forma que el procesamiento recae en la reponsabilidad del daoXXX directamnte, descargando a las clases de la capa dos de hacerlo y potenciando el bean, y no el concepto de clase como tal.

  En esta capa también se incluye un soporte de JDBCConectionPool, construido desde cero (únicamente con fines académicos), que se encarga de controlar todas las conexión que entren desde los DAOS a la database.

No existe una arquitectura perfecta, se ha implementado la que se cree que es la mas purista. Es posible que algunos arquitectos propongan una separación en estas tres capas y otros en muchas más capas. Incluir más capas en una arquitectura aumenta el nivel de abstracción pero también el nivel de complejidad. Haciendo la separación actual de tres capas, se logra separar las tres barreras presentación, lógica y persistencia de manera adecuada.

A continuación se muestra una imagen que presenta el modelo arquitectónico de capas de Sparkea, y donde se ven las distintas capas comentadas de forma previa en la carpeta contenedora de java, siguiendo una arquitectura de carpetas maven. Como se observa, el paquete de login, no lleva la relación exacta de capas como sus hermanos, esto es debido a que se ha realizado un enfoque algo diferente, pero de igual concepto, así lo decidió mi compañero. Este último paquete está desarrollado en persistencia a través de Spring + JPA (Hibernate).

En la parte de más a la derecha se observa el paquete arduino, que se considera parte de la configuración del servidor.

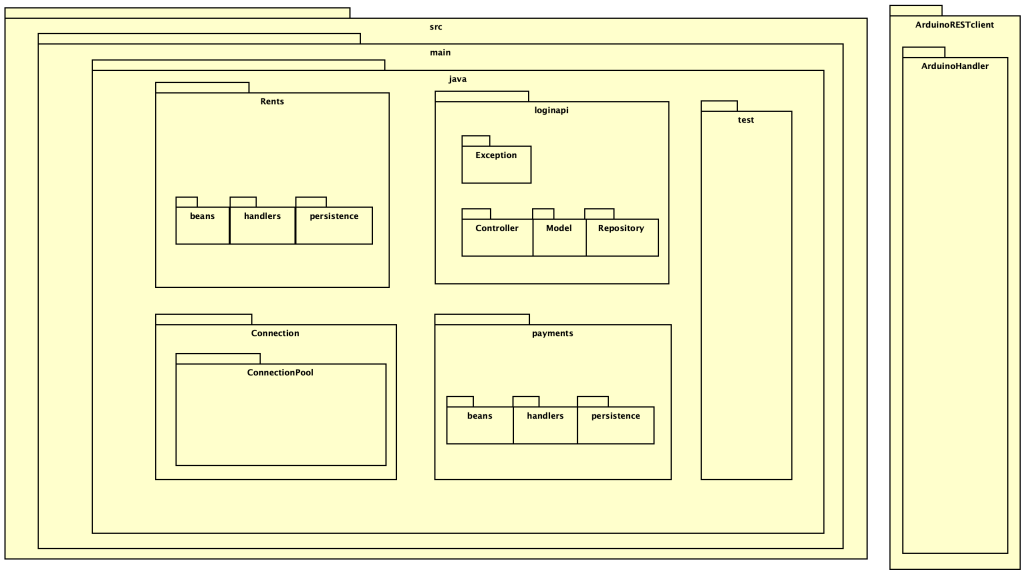


Figura 5.1: Modelo Arquitectura

5.3. Despliegue de la aplicación

Al estar mencionando arquitecturas, se comentará cómo se ha realizado el despliegue para esta aplicación. La idea de descomposición de dominio donde cada API funcionará de una manera aislada, distribuida y escalable, sirvió para aislar y derivar funcionalidades a las entidades correspondientes, reforzando aun más alguno de los principios SOLID los cuales están muy presentes en el desarrollo del proyecto. Realmente el despliegue de la aplicación se hace en un solo Tomcat, poniendo de ejemplo de instancia la de un cliente ionic desarrollado

por mi compañero y presentado en otro Trabajo de Fin de Grado como se ha mencionado anteriormente. En la siguiente imagen podemos observar como Sparkea que forma los tres paquetes -independientes- se despliega entera en un solo war. La instancia de mi compañero se hace ver mas a la derecha.

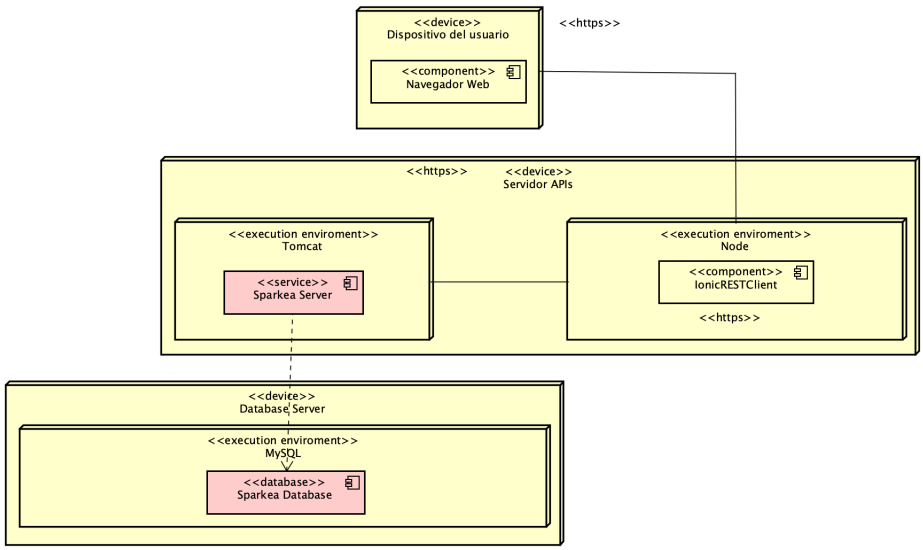


Figura 5.2: Diagrama de despliegue

Al igual que se ha comentado en el sistema de capas, no existe una arquitectura perfecta, pero sí que existen arquitecturas buenas y malas, por ello debemos de comprender bien los problemas y las funcionalidades a aplicar. Entender de manera correcta los patrones y buenas practicas de desarrollo, facilita notablemente la creación de un código y una arquitectura limpia.

5.4. Diagrama de clases y división en microdominios

A continuación se muestra una panorámica del diagrama de clases del servidor, se pinta en azul aquellas clases que actúan como handlers, es decir aquellas que reciben datos, los endpoints. Se pintan de rojo aquellas clases que tienen que ver con la capa de persistencia, es decir las encargadas de las operaciones con la database, y en verde, aquellas que tienen que ver con los beans (objetos del dominio). Aquellas clases con color marrón son las del cliente integrado Arduino.

Se ha de tener en cuenta que el diagrama esté dividido en 3 paquetes, y cada paquete posee sus tres capas, teniendo en cuenta que siempre se ha respetado todos los patrones de diseño.

Disgregar en colores diferentes clases con el fin de asociarlas a distintas capas, teniendo además una panorámica de la arquitectura del servidor, ayuda a apreciar separación de aspectos y responsabilidades así como la independencia de capas.

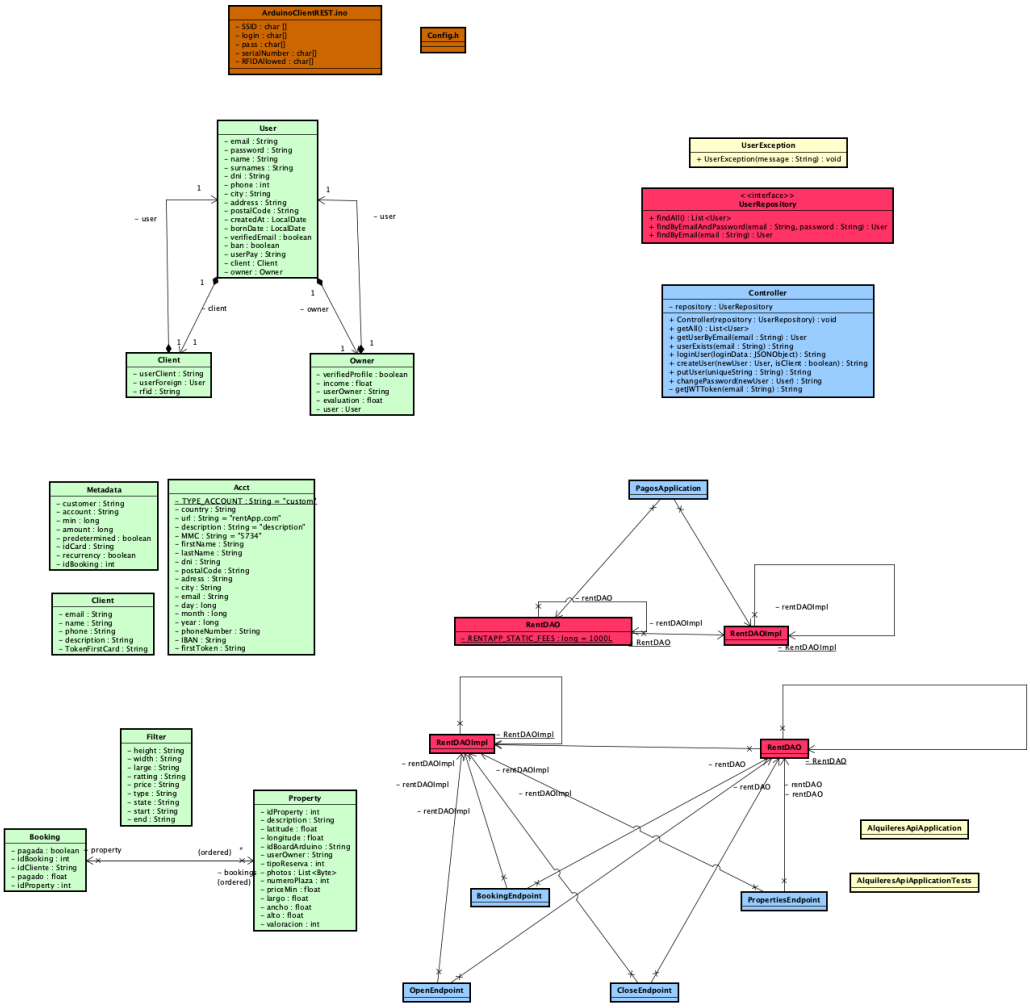


Figura 5.3: Panorámica estructura servidor

En el siguiente apartado se va a mostrar las diferentes paquetes en los que están las clases vistas en este apartado.

5.5. Rents

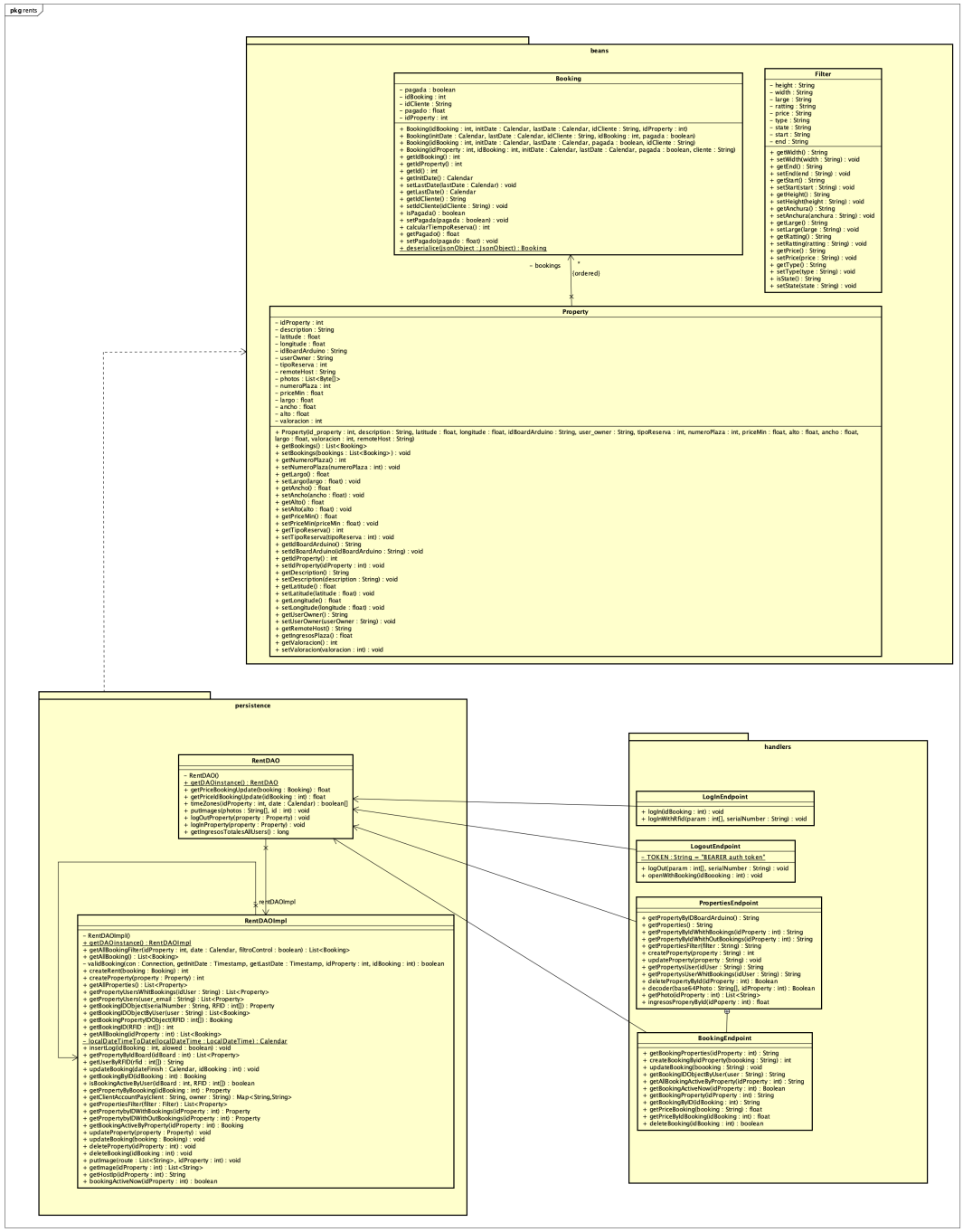


Figura 5.4: Panorámica estructura del paquete Rents

5.6. Payments

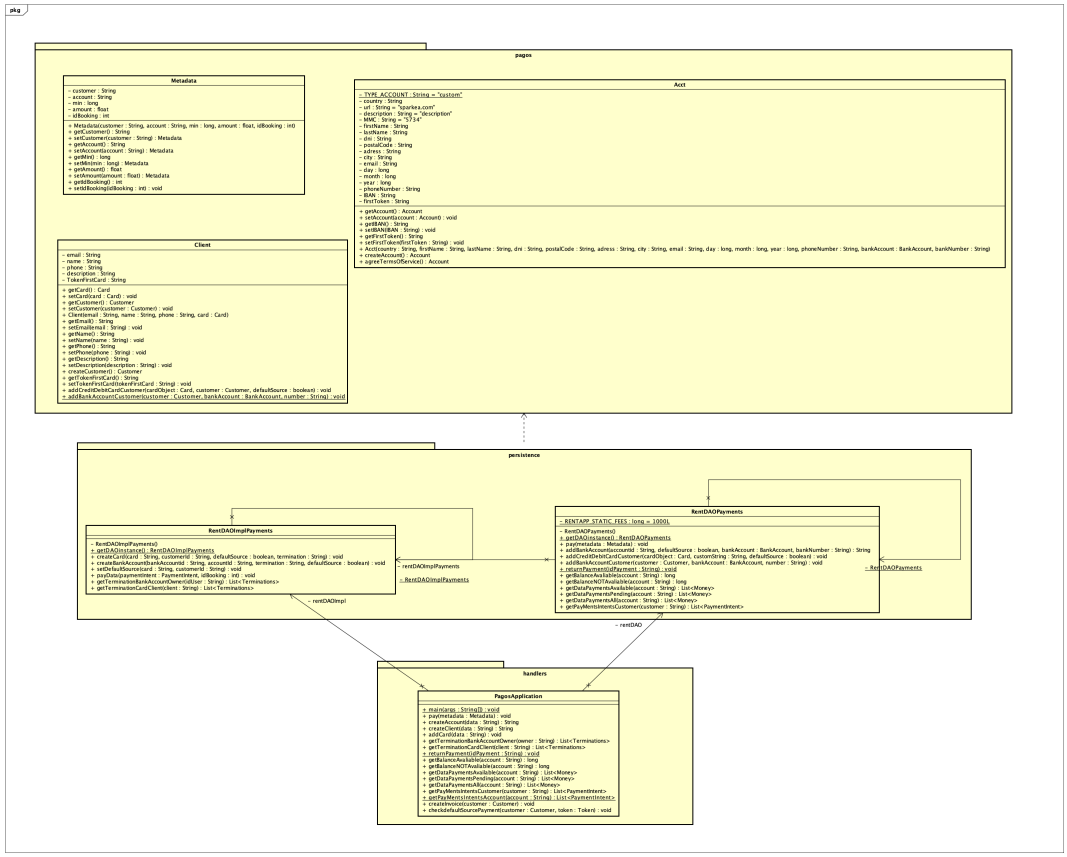


Figura 5.5: Panorámica estructura del paquete Payments

5.7. Login API

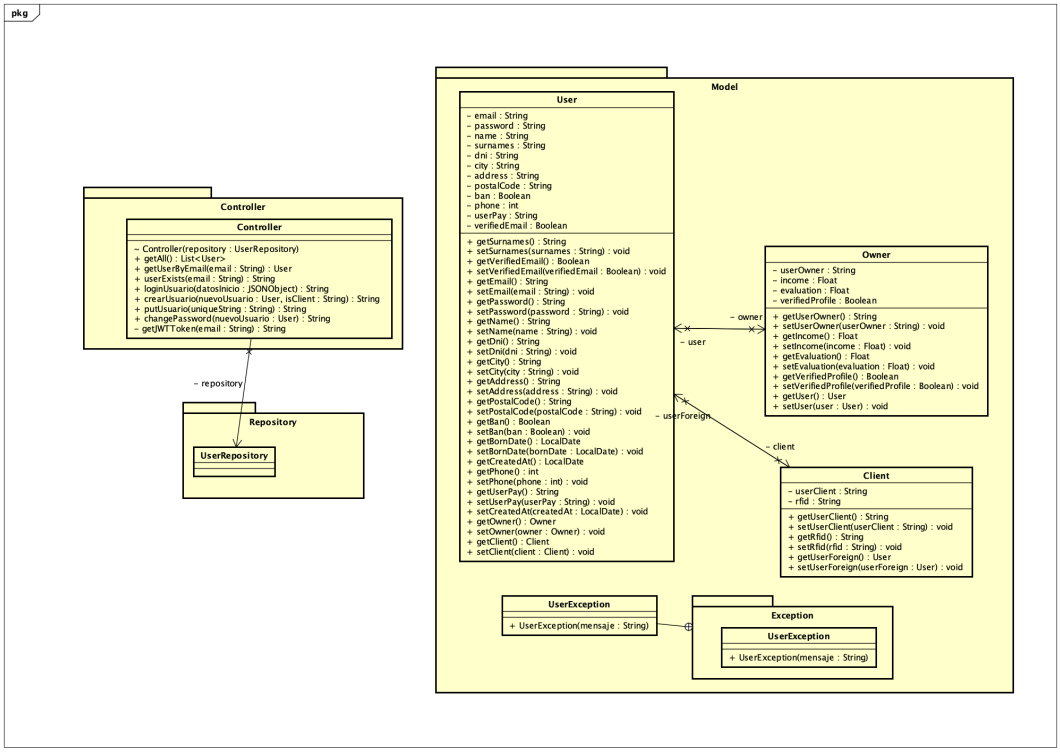


Figura 5.6: Panorámica estructura del paquete Login API

Capítulo 6

Implementación y pruebas

6.1. ArduinoRESTClient

En este Capitulo se abordará las diferentes partes del proyecto de arduino. Se tiene en cuenta que nunca antes se había trabajo con Arduino (o similares ESP8266) y se empieza con la curva de aprendizaje desde el principio con todo lo que conlleva.

Arduino trabaja con dos funciones bien diferenciadas:

- setup() Se ejecuta cuando se proporciona corriente a la máquina
- loop() Se ejecuta continuamente una vez se haya ejecutado el setup()

La apertura de la puerta se hará mediante dos procesos bien diferenciados.

1. Apertura a través de radiofrecuencia RFID.
2. Apertura a distancia, es decir, desde la propia aplicación.

Poner en marcha ambos procesos de apertura en una misma placa, (y por lo tanto en un mismo bucle de ejecución), es una tarea quizás algo compleja, y hay que ser cuidadoso de como hacerlo.

1. Apertura a través de radiofrecuencia RFID. Para la apertura a través de RFID, se cuenta con una placa de radiofrecuencia y varias tarjetas identificadoras. Se trata de un proceso común en aperturas a lugares restringidos hacia determinadas personas.

   Funcionamiento:

   La placa RFID envía una señal con tensión 5V (13.5 ma) hacia una tarjeta, esta alimenta el circuito que tiene en su interior (una antena y una pequeña memoria) y con

la energía suministrada por el aparato, retorna una onda de voltaje modificado que contiene la información de dicha memoria (o lo que especifiquemos que se lea). En este caso se precisa de la lectura del identificador de la tarjeta con la finalidad de identificar de forma univoca a los usuarios de la aplicación. La conexión de la placa RFID se hace a través de una interfaz SPI típica con arquitectura de ME (Maestro-Esclavo). Esta parte se presentó especialmente difícil por la peculiaridad de la placa (Arduino uno wifi rev 2) que se utilizó, la cual la interfaz SPI no se encuentra en los pines 11,12,13 como en las demás placas de Arduino (además de funcionar como salidas digitales). La interfaz SPI en este dispositivo se encuentra en la IPCS, estos pines se encuentran en la parte baja de la placa. Se ha documentado este problema que mucha gente desconocía, debido a que la mayoría de las personas tienden a reutilizar placas más antiguas donde la interfaz sigue en el mismo sitio. Se muestra que contiene esta interfaz SPI (permite la Interconexión con la placa base).

Figura 6.1: SPI interface

Esta parte fue preguntada (y resuelta por mí mismo) en foros.

`https://stackoverflow.com/questions/71265948/communication-failure-rfid-reader-and-arduino-uno-wifi-rev-2`

La interfaz SPI (Serial Peripheral Interface) establece una ruta de comunicación multicanal con la placa, lo que significa que no se limita a leer y escribir en el mismo bus. Este enfoque de comunicación bidireccional es importante para las operaciones de lectura y escritura concurrentes. Vamos a desglosar los componentes clave de esta interfaz para entender mejor su funcionamiento y estructura.

- **MISO (Master In Slave Out):** Este componente es la ruta de datos que permite la transferencia de información desde el esclavo al maestro. Funciona como el canal de comunicación de entrada para el dispositivo maestro y el canal de salida para el esclavo.
- **MOSI (Master Out Slave In):** Por otro lado, el MOSI funciona de la forma contraria a la del MISO. Esta línea de comunicación permite la transferencia de datos desde el maestro hacia el esclavo. Es, esencialmente, la vía de salida del dispositivo maestro y la vía de entrada del dispositivo esclavo.
- **SCK (Serial Clock):** Esta es la señal del reloj del bus, que regula la velocidad a la que se transmiten los bits de datos entre el maestro y el esclavo. Es responsable de la sincronización de la comunicación entre los dispositivos.

- **Reset:** Finalmente, tenemos el botón de Reset, que permite reiniciar la comunicación o restablecer el estado del sistema a su estado inicial. Es una herramienta vital para manejar errores y garantizar la robustez del sistema.

Por lo tanto, cada uno de estos elementos juega un papel crucial en la gestión de la comunicación entre los dispositivos maestro y esclavo en la interfaz SPI.

Los elementos nombrados anteriormente de la interfaz SPI deberán ir obligatoriamente en el IPCS si la placa de Arduino ha cambiado las interfaces SPI.

Si se proporcionase otra placa, las conexiones 11,12,13 donde se encuentran los pines del SPI, se migraría las conexiones del IPCS a los pines 11,12,13. Así es como quedaría la imagen si utilizamos una placa sin modificar la SPI:

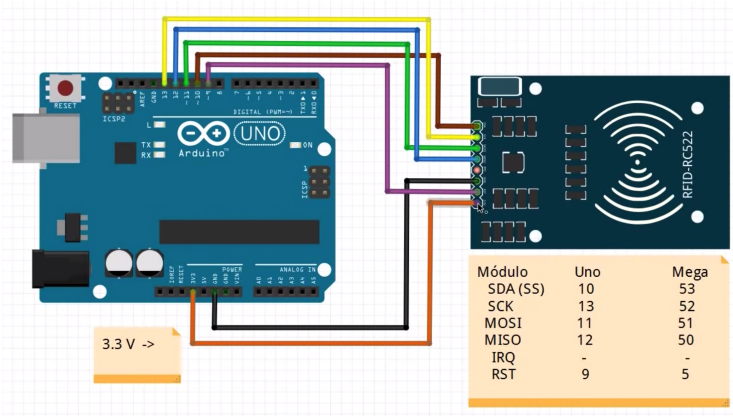


Figura 6.2: Arduino conexión SPI no version 3

2. Apertura a distancia, desde la propia aplicación.

   Cuando un usuario haya reservado una propiedad e intente acceder a ella, se pondrá en alto una salida digital (5V). A través de una interfaz REST implementada en Arduino a través de C, estará a la escucha de nuevas peticiones (a la vez que escuchando si hay o no una tarjeta nueva que comprobar), y cuando haya una nueva envía una petición al endpoint correspondiente. Este tipo de acceso consta de la misma lógica de apertura que la de RFID, trasforma los 5V de salida del Arduino en los 220V de la corriente alterna con el objetivo de mover motores y electroimanes. Se amplificará esta a través de relés y contactos adecuados.

6.1.1. Pantalla digital LCD

Se ha incluido una pantalla LCD capaz de realimentar el estado que nos proporcione el servidor de Sparkea. La pantalla LCD tiene una distribución de 16 x 2 (16 filas y 2 columnas) y cuenta con 16 pines distribuidos de la siguiente manera:

1	GND (Tierra)
2	5 Voltios
3	Control de contraste pantalla
4	RS – Selector entre comandos y datos
5	RW – Escritura y lectura de comandos y datos
6	Sincronización de lectura de datos
7-14	Pines de datos de 8-bit
15	Alimentación luz de fondo (5V)
16	GND (Tierra) luz de fondo (0V)

Tabla 6.1: Pantalla digital LCD

Para regular el contraste de la pantalla se ha utilizado un potenciómetro de 10K. Para regular el brillo se ha usado una resistencia de 220.

6.1.2. Bibliotecas y entornos

Se ha trabajado con dos tipos de placas; la oficial de Arduino (Arduino uno wifi rev 2) y la placa de desarrollo WiFi ESP8266 WEMOS D1 (no original). La primera la encontramos en torno a 49€ y la segunda en torno a 5€. Se destaca la rapidez con la que se configuró y posteriormente se arrancó la primera placa reconociéndola automáticamente respecto a la lentitud que presento la puesta en marcha de la segunda placa. Para la placa WEMOS, requiere de unas instalaciones adicionales como la placa.

Referencia de montaje 1

Referencia de montaje 2

La placa WEMOS D1 se trata de una placa de desarrollo WIFI basada en ESP8266. El funcionamiento es similar al de NODEMCU, excepto que el hardware está construido de forma similar a Arduino UNO. La placa D1 se puede configurar para trabajar en el entorno Arduino usando BOARDS MANAGER.

Para que se reconozca el board manger la ESP8266, se debe descargar:

`http://arduino.esp8266.com/stable/package_esp8266com_index.json`

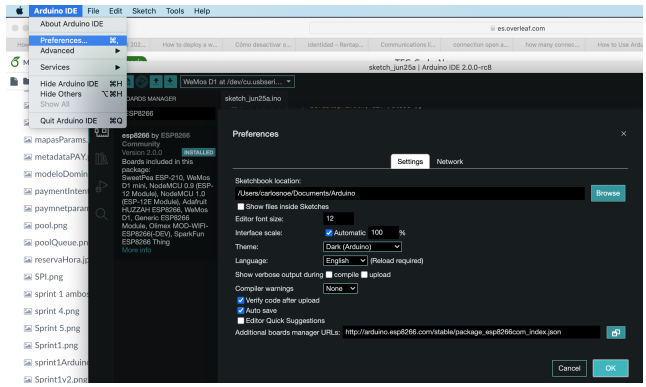


Figura 6.3: Librería Arduino

Una vez descargada, se busca el tablero (conjunto de ellos) con el que se vaya a trabajar:

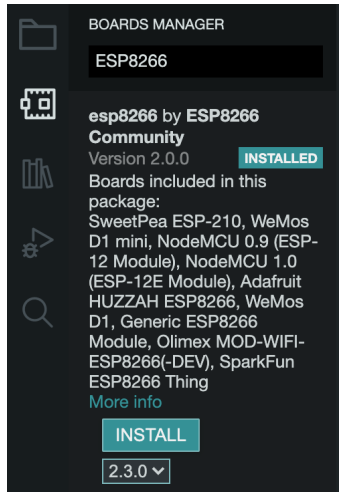


Figura 6.4: ESP8266

No todo es exactamente igual, por ejemplo los pines de conexión con los RFID no son los mismos y se deben de cambiar de un modelo a otro. Por lo general se tienen que hacer pocos cambios entre ambos tableros.

```
#define RST PIN          9
#define SS PIN           10

MFRC522 mfrc522(SS PIN, RST PIN);  // Create MFRC522 instance
```

6.2. Acceso y reserva

Se consta de dos placas hardware ESP8266 como las que se han comentado anteriormente, se usan para la apertura de puertas, identificación de los usuarios, realización de transacciones de datos de entrada-salida, así como los de la lectura de la tarjeta. Ambas cuentan con módulos wifi capaces de comunicarse con el servidor para determinar la posible apertura a un determinado usuario según los diferentes factores que se considere. Se dispone de un paquete en el servidor encargado de realizar tanto las reservas, como de proporcionar el acceso a las propiedades (si aplica) que se vayan dando a los usuarios de la aplicación. Se utiliza la lógica de acceso siguiente:

- Si se encuentra en un sistema de reserva sin hora, comprobará que la plaza no esté en estado ocupado.
- Si se encuentra en un sistema de reserva con hora, comprobará que la plaza no esté en estado ocupado durante la región de tiempo que quiera reservarla.

6.2.1. Tipos de reserva

Se considera que cada usuario que tenga una propiedad concreta (una plaza con un número determinado), pueda poner su propiedad en alquiler escogiendo uno de los dos métodos de alquiler:

- Reserva sin hora. La propiedad adquiere dos estados (Disponible y Ocupada), pudiendo alquilar solo si la propiedad aparece disponible.
- Reserva con hora. La propiedad adquiere un conjunto de regiones horarias, solo se podrá alquilar si se reserva una determinada región horaria no ocupada (no reservada/no alquilada).

Figura 6.5: Reserva Hora

Ambos métodos están disponibles para poder aplicarse si un arrendatario decide poner su propiedad en alquiler. Dependiendo del uso y necesidad de esa propiedad le convendrá usar una determinada opción u otra. Se es consciente de que ambas propuestas pueden traen consigo problemas logísticos, los cuales se han ido refinando tras iterar sobre ellos en los determinados Sprint.

6.2.2. Problemas

En el caso de que la instancia implementada por arriba, sea un parking:

Uno de los problemas que nos podemos enfrentar con esta primera opción es, que si no se realiza una reserva (aunque sea en el momento de abrir), ¿En que plaza se sabe que se está posicionando? En resumen, se necesitará saber con anterioridad, para principalmente saber a qué persona pagar el arrendamiento de esa estancia durante un periodo de tiempo. Por ello no podemos dejar que se entre sin haber elegido una plaza en concreto y haciendo una "reserva" de la misma. Por lo que en el sistema de reserva sin hora, para que un usuario pueda abrir un parking privado, deberá primero (antes de entrar) decir en que plaza se va a posicionar y empezar a cobrarle directamente esa propiedad concreta, y materializar el cobro cuando pase por la casilla de salida.

Otro problema que podemos encontrarnos en la reserva con hora es más que evidente. Pongamos que un usuario A reserva de 10:00-12:00, y un usuario B de 12:05-13:00. Si el usuario A abandonase su plaza fuera del periodo establecido habría colisiones con el usuario B.

Para abordar este segundo problema, más difícil que el anterior, debemos imponer periodos de tiempo de multa entre ambos usuarios, los cuales se calculan de forma proporcional al tiempo de espera, y penalizar al usuario de una determinada manera que no pueda volver a usar dicho sistema durante un tiempo o hasta que vuelva a ganar puntos por buen comportamiento.

El común de los problemas de ambos métodos es: ¿Qué pasa si el usuario se posiciona en otra plaza que no sea la que ha reservado? Dicho problema se disminuye cuando al usuario según entra ya está pagando por una plaza en concreto, siendo así, ¿para que se iba a poner en otra que no fuese la suya? En muchos menos casos sucedería. La solución definitiva pasaría por la implantación de barreras físicas a la hora de estacionar, siendo desbloqueadas de forma electrónica o dejando una llave en una caja en el propio estacionamiento, la cual se abriría a través del móvil o del sistema RFID.

Como se puede apreciar, ninguno de los métodos mostrados cubre en su totalidad toda la casuística de malas prácticas que pueda hacer un determinado usuario. Pero aborda de forma general los diferentes usos que el usuario arrendatario pueda hacer sobre la plaza.

6.2.3. Sistemas en comunidades

En esta sección se abordará la posibilidad de poder poner nuestra plaza de garaje que no usemos en alquiler. Para ello se deberá disponer de un equipo de control de acceso hardware arduino que lo proporcionará Sparkea. Se hablará de costes muy reducidos.

Si el sistema se encuentra en un garaje comunitario (como será en la gran mayoría de casos), como era de esperar, no hará falta que cada vecino haga la compra de una placa, sino que para todos ellos pertenecientes a la misma comunidad servirá la misma placa, logrando aun más reducir el coste de la propia placa. Por ello, si un usuario ha reservado en alguna

de las diferentes plazas dentro de un garaje comunitario, en un determinado horario, pueda obtener acceso.

6.3. PagosAPI

En esta sección se abordará la implementación que se ha establecido en la pasarela de pagos. Se ha decidido no implementar un sistema de pagos desde cero, y montar la pasarela de pagos sobre el servicio deStripe, que nos ofrece funcionalidades de pagos ya descritas por ellos que facilitan en definitiva a modelar nuestro sistema.

Como se describe en el seguimiento del proyecto, Sprint 4, la curva de aprendizaje para empezar a utilizar esta API ha sido mas pronunciada al no haber hecho nunca uso de ella.

La idea principal se sustenta en crear dos tipos diferentes de cuentas (Este tipo de cuentas se crea automáticamente cuando el usuario se registra en nuestra aplicación):

- Cuentas para usuarios que son clientes. Se trata de cuentas que llevan asociada una tarjeta de crédito/débito para pagar según finalice el servicio proporcionado. No es imprescindible establecer la tarjeta según se realice el registro, pero sí cuando se vaya a efectuar un pago. Es importante destacar que el cliente puede guardar sus tarjetas si lo decide (una o varias) para sus posteriores usos sin tener que introducir de nuevo el número. Además, PagosAPI cuanta con un servicio que abstrae de todas las funcionalidades que pueda solicitar un cliente.
- Cuentas para usuarios que son propietarios. Este tipo de cuentas gestionan unos pagos y fondos asociados.

6.3.1. Encriptación de datos en PagosAPI

La lógica de encriptación que hay bajo nuestra pasarela de pagos, está basada en un par de claves, privada y publica, y el proceso de tokenización a través de las cuales conseguimos realizar las operaciones de forma autorizada. Con la clave pública se autoriza de que eres "tu.[el] que va a realizar una petición al servidor pero no puedes realizar esa operación si no posees la clave privada. Todo este concepto se aproxima al concepto de autenticación.

El elemento crucial es que todas las clases (wrapped) que contienen la información sensible, se crean y se identifican mediante una clave única, la cual solo puede ser desencriptada utilizando la clave privada que se ha mencionado de forma previa. Este principio se aplica de manera integral en la API de Stripe.

Para entender mejor el apartado de encriptación y recuperación de datos se proporciona el siguiente ejemplo: Tenemos modelada una clase (Wrapped) que representa los intentos de pagos de un usuario concreto. Cada vez que se genera una instancia de esta, consecuencia de la realización de un intento de pago, se genera un HASH asociativo de dicho objeto. De esta manera se logrará identificar de forma univoca ese intento de pago y además se podrá recuperar del servidor suyo a través de ese identificador y la clave privada.

En este ejemplo la clave publica se asocia hacia `pi-3Kj4q2KHxUyIHSm01A00C57U` y sería imposible la recuperación del mismo sin estar en posesión de la clave privada.

El uso de HASH para comprobar la identificación e integración de datos es muy común en muchos tipos de aplicaciones y no existe dos identificadores iguales para dos tipos de contenidos distintos.

Lo que acabamos de mencionar es el denominado proceso de tokenización, y Stripe hace uso de él para asegurar los datos sensibles del titular de la tarjeta. Durante este proceso, los detalles de la tarjeta son reemplazados por un token único generado aleatoriamente, como los identificadores que se han mostrado anteriormente. Este token no tiene valor si se ve comprometido y puede ser utilizado para procesar transacciones sin exponer los detalles reales de la tarjeta.

```
public static void returnPayment(String idPayment) throws StripeException {
    PaymentIntent pi = PaymentIntent.retrieve("pi_3Kj4q2KHxUyIHSm01A00C57U");
    if(!pi.getStatus().equals("succeeded")){
        pi.cancel();
    }
}
```

Figura 6.6: Payment Intent

Lo que se esta haciendo en ese método es cancelar el pago, `pi-3Kj4q2KHxUyIHSm01A00C57U` si este no ha sido éxito.

El modelo de encriptación y recuperación se aplica de la misma manera a los demás objetos del dominio encapsuladores de información como es el ejemplo de Customer, Account, Card, Balance etc.

La forma de encapsular un objeto de esta manera a través de un identificador único, oculta más lógica aparte de la ya mencionada. Por ejemplo, en objetos como Cards o Bank Accounts (o cualquier bean (wrapped) que almacene información), el HASH en sí solo se pasa una vez cuando creamos el id de todo el objeto, y una vez pasado nunca se volverá a reflejar. Se podrá recuperar el resto de valores pero nunca el número.

Por ello, como clientes de dicho servicio Stripe, es nuestra responsabilidad es que dichos identificadores se almacenen de forma persistente en nuestro sistema. A través de ellos y la clave privada, se puede efectuar una operación (aunque no se retorne ya nunca el número de tarjeta o cuenta bancaria etc).

Para entender mejor la lógica de encriptación de datos se proporciona el siguiente ejemplo: Se crea una Bank account, se le pasan todos los parámetros, incluido el número de cuenta, pero cuando se nos retorna ese objeto a través de su id, no aparece dicho número, y aun así podemos realizar transacciones a través de ese identificador y la clave privada que se nos ha proporcionado a través de Stripe.

```
Map<String, Object> bankAccount5 = new HashMap<>();

bankAccount5.put("country", bankAccount.getCountry()); //"ES"
bankAccount5.put("currency", bankAccount.getCurrency()); //"eur"
bankAccount5.put(
        "account_holder_name",
        bankAccount.getAccountHolderName()
);
bankAccount5.put(
        "account_holder_type",
        "individual"
);
bankAccount5.put("routing_number", bankAccount.getRoutingNumber()); //BSCHESMM
bankAccount5.put("account_number", IBAN);
Map<String, Object> params8 = new HashMap<>();
params8.put("bank_account", bankAccount5);

Token token = Token.create(params8);

/**
 * AÑADIMOS LA CUENTA DEL BANCO
 *
 */
Map<String, Object> params6 = new HashMap<>();
params6.put(
        "external_account", token.getId()

);
BankAccount bankAccountStripe =
        (BankAccount) account
                .getExternalAccounts()
                .create(params6);
```

Figura 6.7: BankAcount create

```
BankAccount bankAccount =
  (BankAccount) customer.getSources().retrieve(
    "ba_1KpWRREr5jgnxncSgFTf2FAB"
  );
```

Figura 6.8: haciendo un retrieve con el identificador de ese objeto

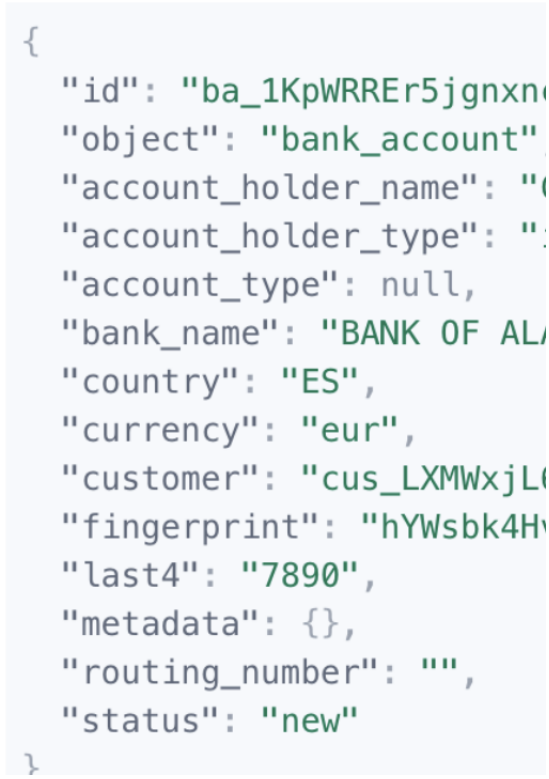


Figura 6.9: el objeto ya deserializado

El número de rooting pertenece a el Banco Santander y la cuenta bancaria realmente es una cuenta bancaria de prueba.

Como se comenta más adelante en el seguimiento de uno de los Sprints, todas estas funcionalidades añadidas a nuestra API usando la pasarela de pagos de Stripe, se han encapsulado de forma concreta para la correcta integración que se haga desde la API de usuarios, ocultando y absorbiendo toda la complejidad que pueda mostrar Stripe en ese aspecto. Por ello se usan Wrappers propios. Dichos Wrappers son clases envolventes que encapsulan diversas funcionalidades y hacen una integración mucho más compacta. Un ejemplo se encuentra en la clase Client o Account o Payments, etc.

6.3.2. Persistencia en Pagos API

Uno de los puntos a tener en cuenta de la API, es qué vamos a tener almacenado en nuestra base de datos y qué es directamente recuperable a través de Stripe. Tras comprobaciones, se modela una estructura en la base de datos y se decide tanto qué guardar y cómo guardarlo. El aspecto de la seguridad sobretodo en esta API es fundamental.

Se guardará información del identificador del objeto, a través del cual, como se ha expli-

cado anteriormente, se podrá recuperar toda la información junto con su private key.

Identificadores de objetos a guardar:

- Arrendadores que reciben pagos a través de la API de Stripe. Estas son las cuentas que recibirán los pagos a través de la API de Stripe. Se almacena un identificador de cadena único para cada propietario de la cuenta.
- Arrendatarios que realizarán pagos a través de la API de Stripe. Dado que no almacenamos la información de las tarjetas de crédito asociadas por razones de seguridad, guardaremos en su lugar la información relacionada con el identificador de la tarjeta, lo que nos permitirá procesar transacciones sin necesidad de almacenar los detalles mas sensibles de la tarjeta.
- Tarjetas de crédito: Almacenamos identificadores de las tarjetas de crédito para facilitar las transacciones. Las tarjetas de crédito son para los arrendatarios.
- Cuentas Bancarias: Similar a las tarjetas de crédito, también almacenamos identificadores de las cuentas bancarias para facilitar las transacciones. Las cuentas bancarias son para los arrendadores.

Capítulo 7

Seguimiento del proyecto

Para la realización del proyecto, se ha decidido usar un desarrollo Agile, en contraposición con otras metodologías de desarrollo cada vez menos habituales y más pesadas.

Dentro del desarrollo Agile, se ha escogido SRCUM como marco de referencia, y gitlab boards como herramienta para gestionarlo. Existen muchas herramientas para el desarrollo Agile, como gitKrakenBoards (cliente de git GitKraken con funcionalidad adaptada), o Jira ... todos ellos ofrecen el manejo de issues, aunque el enfoque que se proporciona es ligeramente diferente. No se trata de realizar un recorrido mostrando cómo se utiliza esta herramienta, si no de mostrar el resultado obtenido con ella, poniendo en evidencia el potencial que ofrece GitLab no solo como control de versiones hacia un desarrollo, si no también hacia un seguimiento en un desarrollo Agile (incluidos todos los marcos de referencia que nos queramos posicionar dentro de Agile).

La arquitectura general de la aplicación se ha englobado entorno a un **Grupo** GitLab formado por los integrantes Carlos Noé (yo), Pablo Verdejo, y la responsable Yania Crespo. Dicho grupo se ha dividido en varios proyectos. A continuación se muestra de manera gráfica como quedaría este planteamiento:

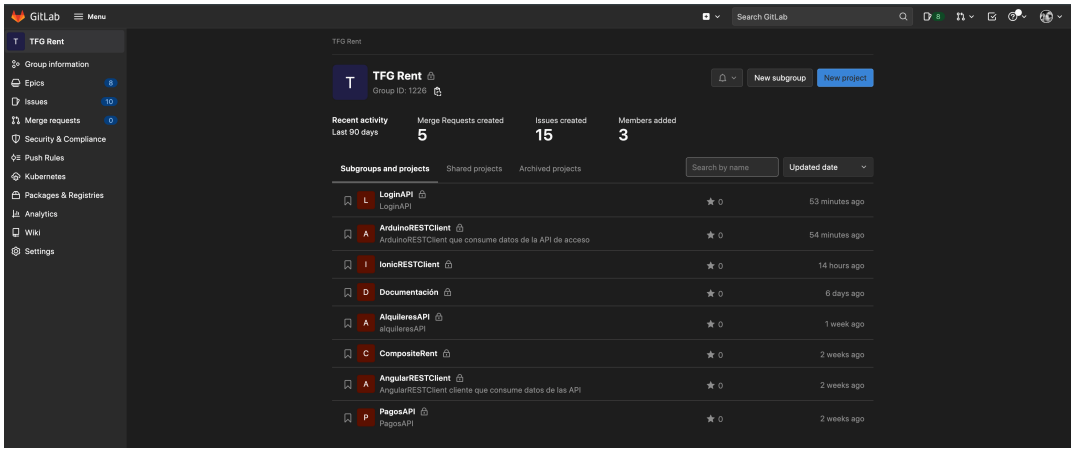


Figura 7.1: Arquitectura

Cuando estamos en un nivel de grupo, estamos posicionándonos a un nivel de épicas y de issues de todo el proyecto. En el caso de las épicas: solo son aplicables a nivel de grupo, es aquella funcionalidad que se soluciona en varias historias de usuario, que en nuestro caso hablaremos del termino de issues refiriéndonos a lo mismo. Y en el caso de las issues (las generales) ofrecen una "amplia" lista de todas las issues que contengan todos los proyectos en su conjunto).

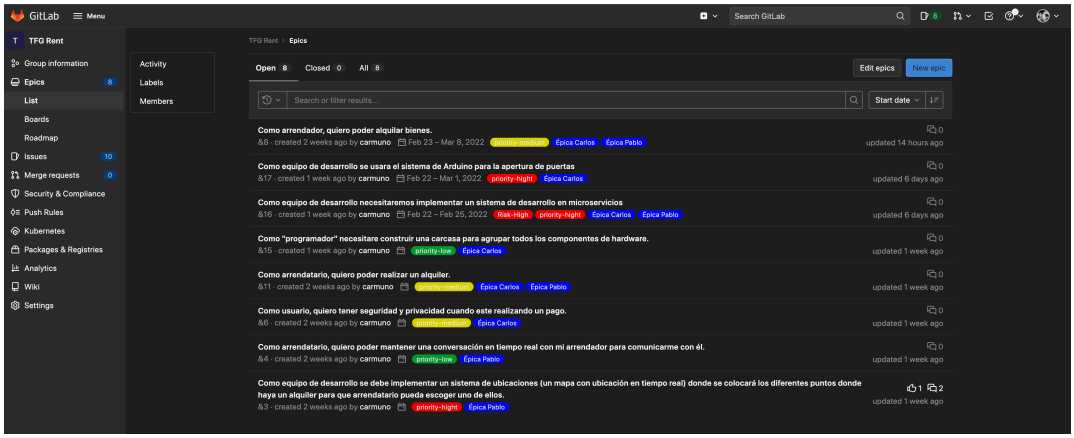


Figura 7.2: Lista de épicas

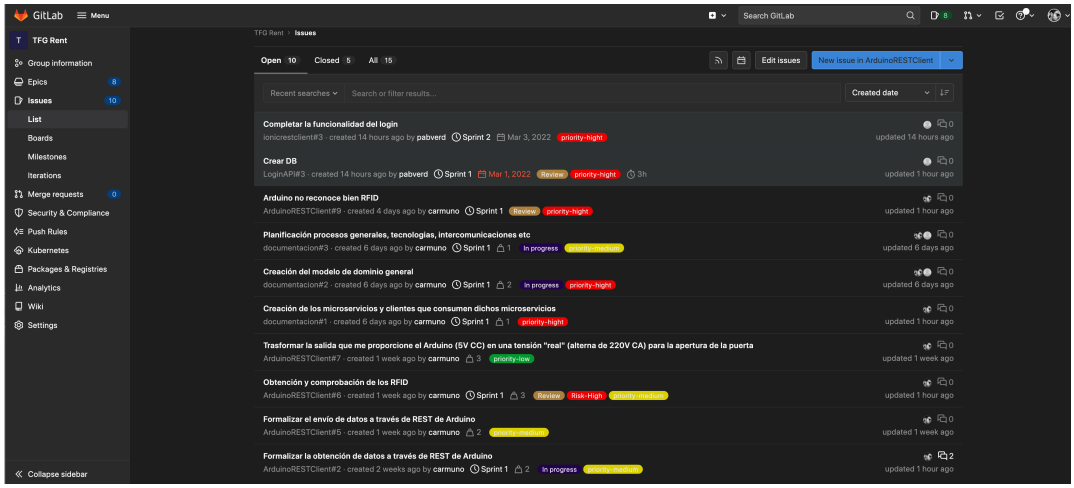


Figura 7.3: Lista de issues general

En cambio a nivel de proyecto (librería) es el propio proyecto, el encargado de manejar sus propias issues (issues particulares del propio proyecto) siendo inconscientes de las issues de sus proyectos hermanos. Ambos compañeros tenemos asociados varios de estos repositorios, cada uno el suyo. Más adelante se proporcionará información especifica de los tableros con los que se gestionan estas issues.

Teniendo clara la distribución del proyecto, se proporcionará una serie de tableros, tanto para épicas, issues generales, como issues propias de cada proyecto diseñados para comprender mejor el funcionamiento y desarrollo del proyecto. A continuación se detalla uno de los tableros que se han proporcionado para el manejo de las épicas conforme a la prioridad que tengan dichas épicas.

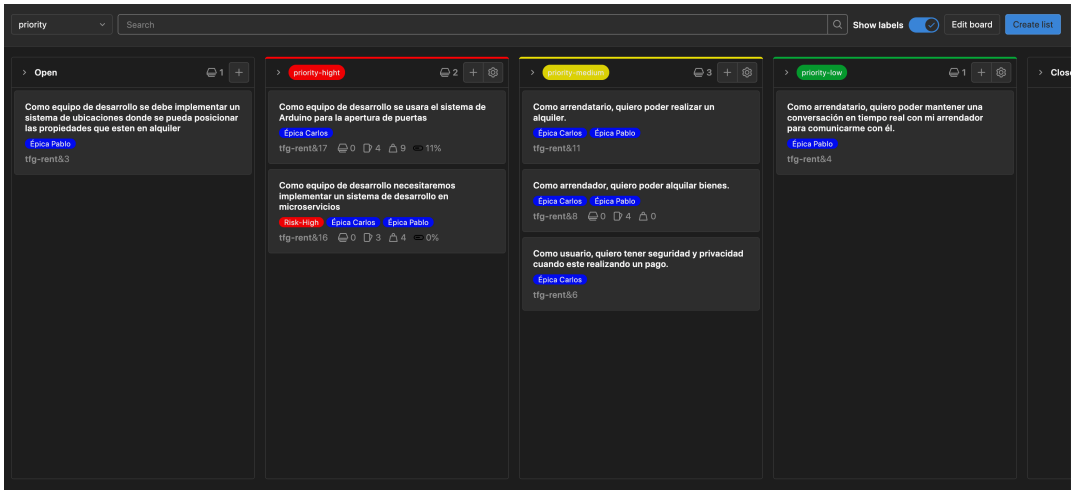


Figura 7.4: Prioridades de épicas

En el tablero de las issues generales (aquellas que englobaban las issues de todo el proyecto) se definirán los llamados milestones, (Sprints) donde se irán colocando las issues asociadas a cada milestone (Sprint). Es cierto que de esta manera ambos compañeros compartiríamos el mismo Sprint, pero con ayuda del desarrollo de tableros se consigue diferenciar sin problema ambas partes.

Para los Sprints, el equipo de desarrollo contamos con diferentes labels que facilitan el seguimiento del mismo. Es el ejemplo de "in Progress" refiriéndose a que ya esta dentro del Sprint X y además esta en proceso de desarrollo y "review" estado al que se ponen aquellas tareas que finalizan pero aún no han sido supervisadas por la responsable Yania Crespo.
Existen otras etiquetas destinadas a la mejor comprensión del estado de una issue y no esta determinada a estar dentro de un Sprint X, como puede ser el ejemplo de "High risk" que nos indican el alto grado de peligro que nos podemos enfrentar al realizar dicha tarea o "priority" para indicarnos si debemos enfocarnos antes o después en esa tarea. Estas ultimas etiquetas no tienen porque ir asociadas a una issue, si no que se pueden asociar a una épica por ejemplo.

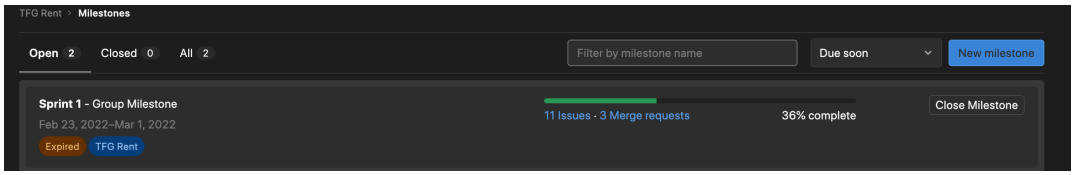


Figura 7.5: Sprint 1

A continuación se hará énfasis en el desarrollo de los Sprints X.

7.0.1. Sprint 1 (23/02/2022 - 01/03/2022)

Como se trata del primer Sprint se hará un resumen mas elaborado y con mayor profundidad , en los siguientes Sprints se abordará el problema directo y haciendo énfasis en lo mas importante. Como se ha mencionado anteriormente, se va a proporcionar varios tableros intentando cada uno de ellos proporcionar una vista diferente hacia un mismo problema.

El comienzo de la primera etapa del proyecto se aborda en el Sprint número 1, y para indicar las issues que se abordarán en esa etapa, se deberán mover a la columna del milestone Sprint 1. Comenzaremos con las issues en visión general de grupo. (todas las issues de todos los proyectos). A continuación se mostrará de en forma de lista el conjunto de issues que hay en el proyecto general (grupo), y solo las que se indican mediante un icono de reloj como Sprint 1 están (al menos en "open") en el Sprint 1. En cada issue nos encontramos toda la información asociada a ella, como es el ejemplo de la épica asociada, el proyecto al que esta asociado, el milestone que esta asociado, la situación que tiene asignada dentro de ese Sprint (con etiquetas, por ejemplo, esta) "In progress", y demás etiquetas como la prioridad que requieren independientes del milestone.

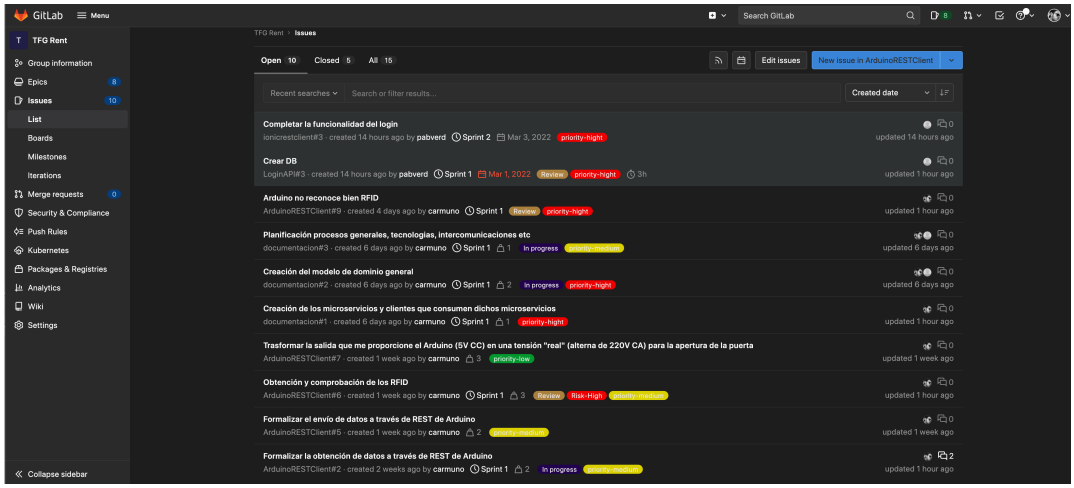


Figura 7.6: Lista de issues general

A continuación se muestra uno de los tableros generales de todos los integrantes para poder indicar de todas las issues, la situación en la que se encuentran. Por ejemplo, si movemos a la columna Sprint 1, cualquier issue de cualquier proyecto, automáticamente comenzará esa issue el Sprint 1. Dicho cambio se mostrará de forma individual el la issue que sea el proyecto.

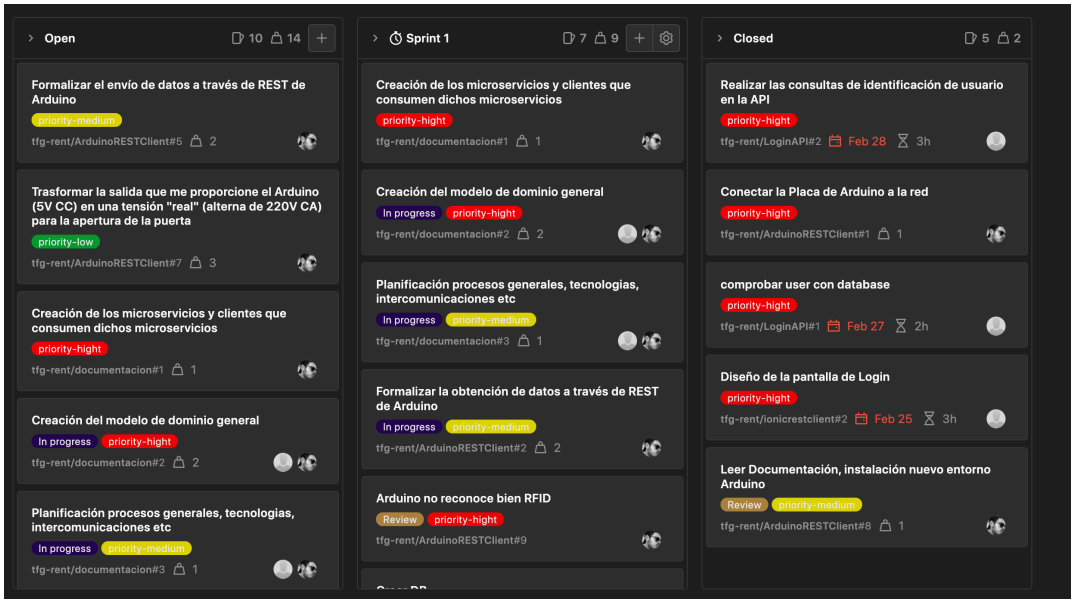


Figura 7.7: Sprint 1 general

Si generamos un tablero para la vista de como esta el Sprint 1 en todos los usuarios del grupo, en estos momentos del proyecto, es decir, mostrar las issues que hayan entrado en la columna de Sprint 1, mostraremos por todos los estados en los que estén pasando dichas historias.

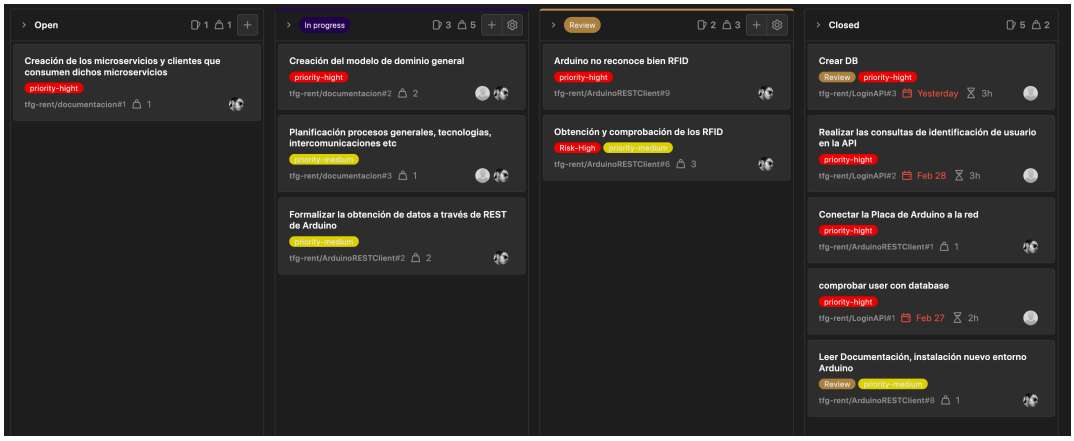


Figura 7.8: Sprint 1 general ambos miembros del grupo

El tablero de Team Programmer muestra el reparto de tareas entre los diferente proyectos dependiendo de la persona que este asignada:

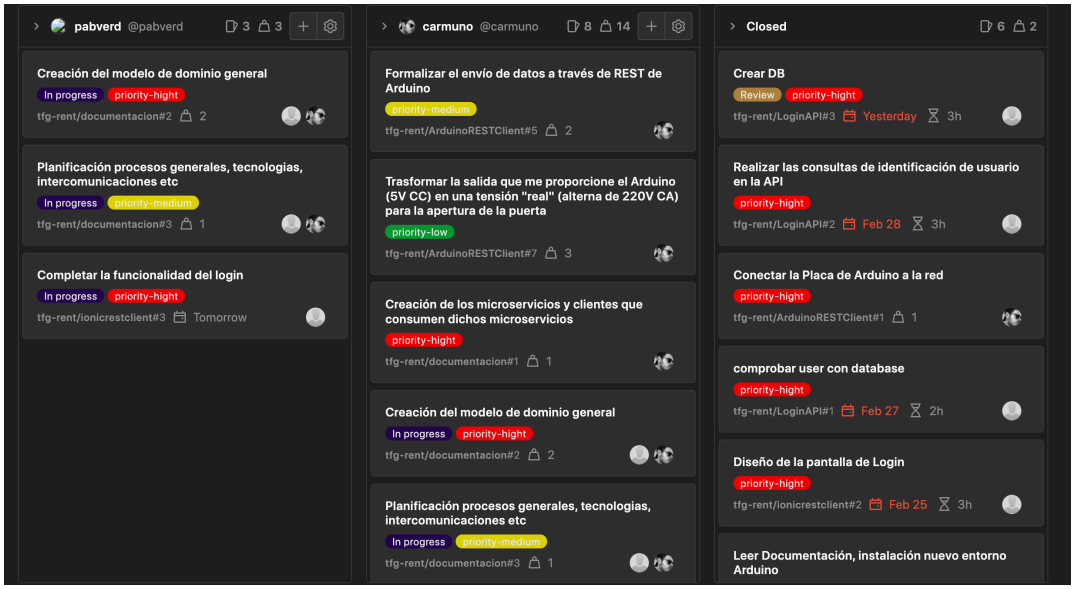


Figura 7.9: Team programmer

Otra vista interesante de proporcionar el de la vista del Sprint 1 a través del participante

Carlos Noé y realizando un filtrado por Sprint 1:

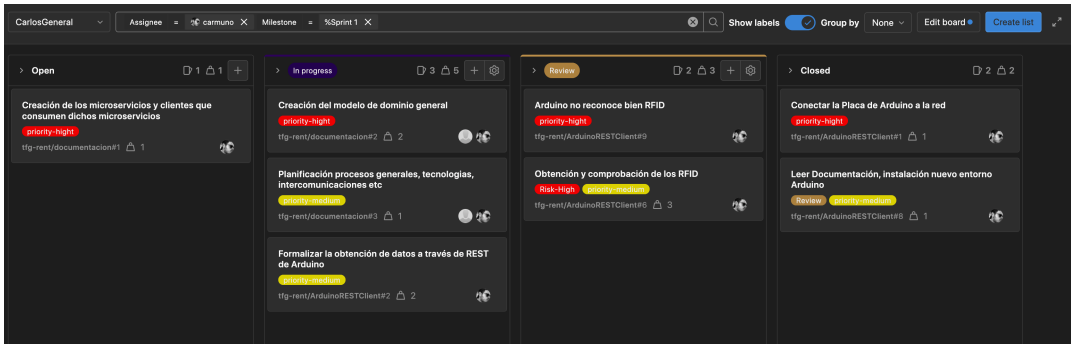


Figura 7.10: Tablero filtrado por Carlos Noé y Sprint 1

Se han proporcionado muchas más vistas que se pueden consultar directamente desde el repositorio que se ha creado para este proyecto, incluyendo todas las combinaciones que se nos puedan ocurrir. Cuando manejamos este tipo de herramientas nos damos cuenta del potencial que tiene y lo que puede aportar a nuestro proyecto.

En este Sprint se ha abordado sobre todo issues de ArduinoRESTclient:

Figura 7.11: Tablero que representa la progresión de las issues, cuales han entrado ya en el Sprint 1 y cuales no

Lista de algunas de las issues preparadas para este cliente. Aquellas que tienen un reloj y la anotación de Sprint 1, están ya dentro del Sprint 1:

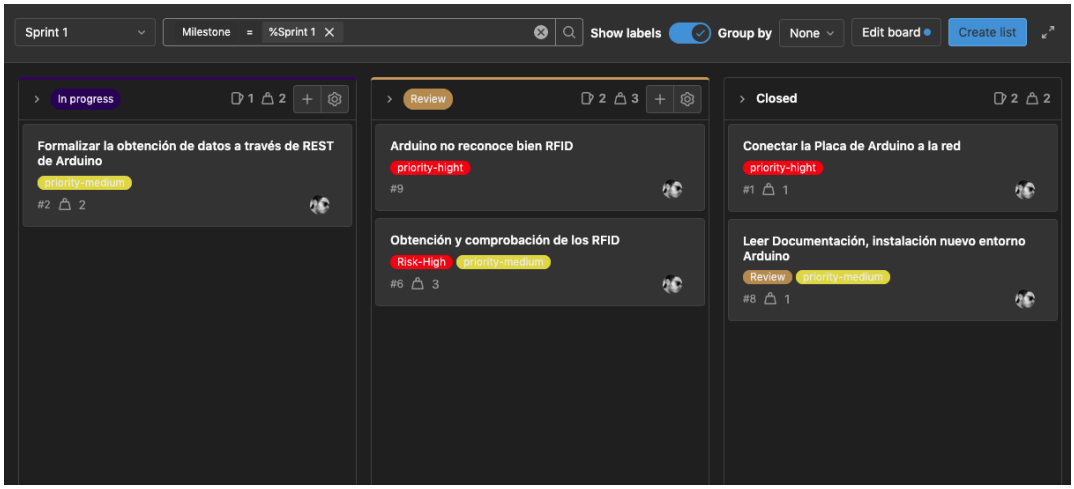


Figura 7.12: Sprint 1 Arduino

Las issues abordadas de arduinoRESTclient tenían especial complejidad por ser programadas en un lenguaje de programación diferente (C) al que siempre usamos en resto del backend (Java). Una parte del tiempo dedicado a este Sprint se dedica a la formación en este ámbito.

Uno de los problemas más serios abordados en el Sprint 1 con arduinoRESTclient es la imposibilidad de leer los identificadores RFID con la placa arduino wifi rev 2. Tras muchas horas de investigación se llega a la conclusión de que el sistema de comunicación de la placa base con la antena se realiza a través de la interfaz SPI, la cual está basada en la conocida metodología de maestro-esclavo. Es decir la comunicación de ida y de vuelta no se hace a través de un solo bus de datos. Tras tiempo de investigación se resuelve el enigma. La interfaz SPI no se encuentra en los pines habituales de la salidas digitales, si no conjuntamente en la interfaz ICSP. Se ha explicado anteriormente de forma detallada 1 como se ha resuelto el problema por si quisiera implementarse tanto en una placa normal (sin la interfaz SPI cambiada), como en una con una interfaz SPI cambiada. Una vez que se logra obtener los RFID id, se truncan dichos números y se pasan a formato numérico y el servidor los procesa.

Otro de los problemas, es la incorporación de la red a la placa wifi que integra. Se puede obtener acceso a través de varias librerías, pero la única que ha mostrado respuesta ha sido la implementada actualmente, la librería de <ESP8266>. Se ha comprobado el alcance y la banda de esta señal, siendo solamente posible su conexión a través de bandas de 2.4 GH y pudiendo estar a una distancia amplia del dispositivo que emite la señal.

7.0.2. Sprint 2 (02/03/2022 - 15/03/2022)

Figura 7.13: Sprint 2

La mayor parte del tiempo dedicado a este Sprint 2 ha sido para dar posibilidad de funcionar ambos procesos de la placa de Arduino juntos (lectura de RFID y apertura vía REST). había un problema que si no recibía respuesta de la solicitud HTTP, se quedaba en estado de bloqueo y no podía leer mas tarjetas. se soluciona poniendo condiciones.

7.0.3. Sprint 3 (16/03/2022 - 28/03/2022)

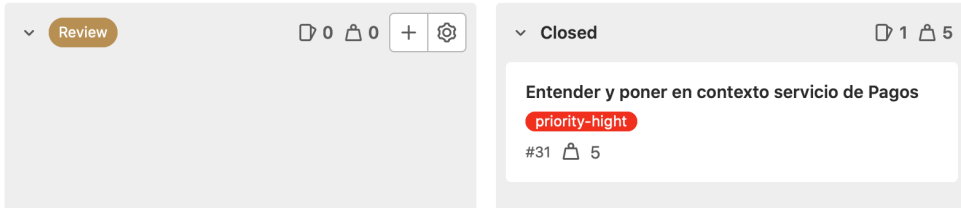


Figura 7.14: Sprint 3

En este Sprint 3 se aborda de forma generalizada el servicio de pagos y su integración con los demás paquetes que forman el sistema. Es imposible pensar la realización de una pasarela de pagos desde cero tanto por exceso de trabajo como por poder derivar en problemas de seguridad e integridad en la aplicación.

Se decide utilizar una capa de una API externa llamada Stripe 4.1.6. La mayor parte del tiempo se ha dedicado a entender la documentación de la misma. Es importante destacar el aprendizaje que se adquiere al ver las posibles diferentes formas en las que se puede modelar un concepto y como se utiliza a través de la API que se esta utilizando por debajo, por ello a parte del objetivo principal de crear una pasarela de pagos, se consigue la mejor comprensión de diferentes modelos hacia un mismo problema. Principalmente Stripe al ser una API, su principal modelo de comunicación y procesamiento es a través de datos JSON, como se ha mencionado previamente, que gracias a su librería, librerías externas y modelado propio se logra una perfecta interacción con nuestra aplicación proporcionando una solución acorde con nuestro problema.

```
Map<String, Object> card = new HashMap<>();
card.put("number", "4242424242424242");
card.put("exp_month", 10);
card.put("exp_year", 2023);
card.put("cvc", "409");

Map<String, Object> tokenee = new HashMap<>();
tokenee.put("card", card);

Token token = Token.create(tokenee);
```

Figura 7.15: Mapas Params

Una de las curiosidades más importantes es que Stripe trabaja en dos modos, modo test y el live mode, y cada modo cuenta con un par de claves únicas, una publica y una privada, siendo un total 4. Toda esta semana hasta que me familiarizado con la API subyacente, se ha utilizado principalmente el modo testing por si algo no fuese como es debido.

7.0.4. Sprint 4 (29/03/2022 - 12/04/2022)

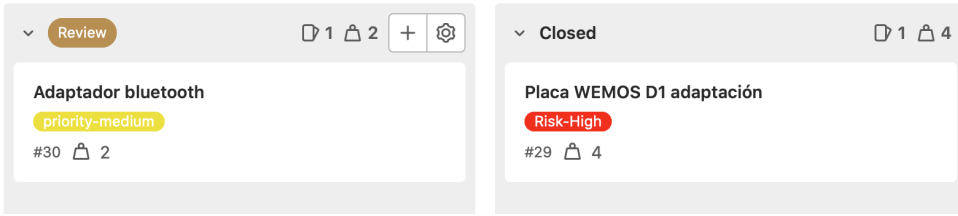


Figura 7.16: Sprint 4

En este Sprint se ha dedicado el tiempo a varias tareas, entre ellas la realización de la propia documentación de la memoria, descargar y también adaptar el software a las nuevas placas de arduino (modelos WeMos d1) (mucho más baratos y los que se implementarían de cara a un desarrollo real), acelerar las consultas y optimizar la REST API de alquileres por problemas de latencia. En definitiva se han abordado issues pertenecientes a varias librerias diferentes. Como propuesta de mejora se ha intentado establecer una comunicación con la placa ESP8266 para que según se instala poder proporcionar las claves wifi sin necesidad de que vaya una persona a su instalación y cambie las contraseñas en el propio código.

7.0.5. Sprint 5 (20/04/2022 - 03/05/2022)

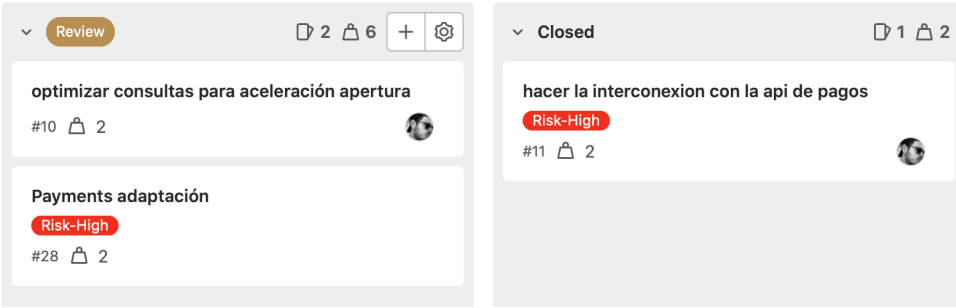


Figura 7.17: Sprint 5

Este Sprint se ha dedicado principalmente a la organización del código abordado en el Sprint 4 (pasarela de pagos) así como la la redacción de la parte de la memoria que engloba dicha API. Se ha ajustado, modelado y clasificado las diferentes funcionalidades presentes en la API en torno a sus respectivas clases. De esta forma se adapta y se crea esa interconexión entre las necesidades nuestras y los servicios proporcionados con la API de Stripe. Actualmente aquella persona que consuma recursos de la API de pagos no se debe preocupar de absorber la complejidad (si es que la hubiera) de Stripe, si no que viene la funcionalidad ya adaptada para el uso que nosotros queremos.

Un ejemplo de ello es la creación de un usuario. Cuando creamos un usuario, para que pueda realizar pagos a través de la pasarela de pagos, no es necesario involucrarse en pasar parámetros desconocidos o preocuparse en el cómo pasárselo. A través de la adaptación propuesta lo podemos hacer más fácilmente. Otro ejemplo (de muchos) es la creación de una cuenta Account, es decir, la creación de una cuenta para propietarios. Simplemente a través de la creación de un sencillo objeto que hace de WRAPPED hacia los demás servicios que hemos creado se encarga de crear la cuenta.

```
private static final String TYPE_ACCOUNT = "custom";
private String country;
private String url;
private String description;
private String MMC;
private String firstName;
private String lastName;
private String dni;
private String postalCode;
private String adress;
private String city;
private String email;
private long day;
private long month;
private long year;
private String number;
private Account account;
private BankAccount bankAccount;
private transient String bankNumber;
```

```
public Account createAccount() throws StripeException {

    AccountCreateParams params =
            AccountCreateParams
                    .builder()
                    .setCountry(country) //"ES"
                    .setType(AccountCreateParams.Type.CUSTOM)
                    .setBusinessType(AccountCreateParams.BusinessType.INDIVIDUAL) //si queren
                    .setBusinessProfile(AccountCreateParams.BusinessProfile.builder().setUrl(
                            .setProductDescription(description)
                            .setMcc(MMC).build()) //"5734"
                    .setIndividual(AccountCreateParams.Individual.builder()
                            .setFirstName(firstName)
                            .setLastName(lastName)
                            .setIdNumber(dni)
                            .setAddress(
```

Figura 7.18: AccountParameters y createAccount

Es interesante comentar la novedad de la palabra reservada transient delante del tipo de

dato que lleva el número de cuenta bancaria. Significa que cuando serialicemos o exportemos ese objeto, ese atributo no será visible. Si lo pensamos bien, solo necesitamos ese atributo para pasárselo a Stripe la primera vez que se crea esa cuenta bancaria, posteriormente la plataforma nos devuelve su objeto a través de un id, por ejemplo "ba-KpWRREr5jgnxncSgFTf2FAB", el cual no lleva el número de cuenta, pero si se le pasamos a Stripe lo reconoce para poder llegar a hacer trasferencias a dicha cuenta. Todo esto es por medidas de seguridad. En las tarjetas de crédito pasa exactamente lo mismo. Se proporciona un ejemplo de una cuenta bancaria ligada a un cliente:

Figura 7.19: bank acccount y bank object

Si nos fijamos correctamente, en ningún lado sale el número de cuenta en nuestro objeto de Bank Account, como se ha comentado antes, por razones de seguridad. Aun así, con ese identificador nosotros podemos guardarlo en la base de datos, y poder realizar operaciones sin problema a través de Stripe.

Se aceleran además las consultas de apertura y cierre de puerta, proceso muy costoso de muchas pruebas pero muy necesario.

7.0.6. Sprint 6 (04/05/2022 - 17/05/2022)

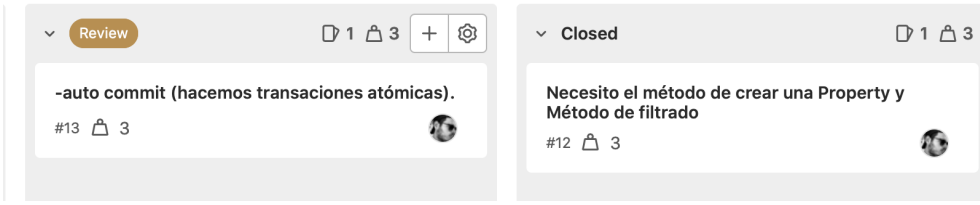


Figura 7.20: Sprint 6

Uno de los problemas asociado a este Sprint es la decisión de usar o no un autocommit. El uso de autocommit es muy cómodo pero muy peligroso. Deriva la problemática de no

poder ejecutar dos consultas juntas sin asegurarse de que otro usuario se haya colado en otra conexión a hacer un execute() de su consulta entre medias de las otras dos. Si dicha consulta no tiene nada que ver quizás no tengamos problemas, pero si la consulta trata de insertar una reserva, e inmediatamente después recuperar la id con la que se ha insertado, si el otro usuario crea una reserva entre medias de esas dos tendríamos una inconsistencia. Finalmente se opta por desactivar el autocommit en aquellas consultas que requieran ejecución atómica.

7.0.7. Sprint 7 (18/05/2022 - 31/05/2022)

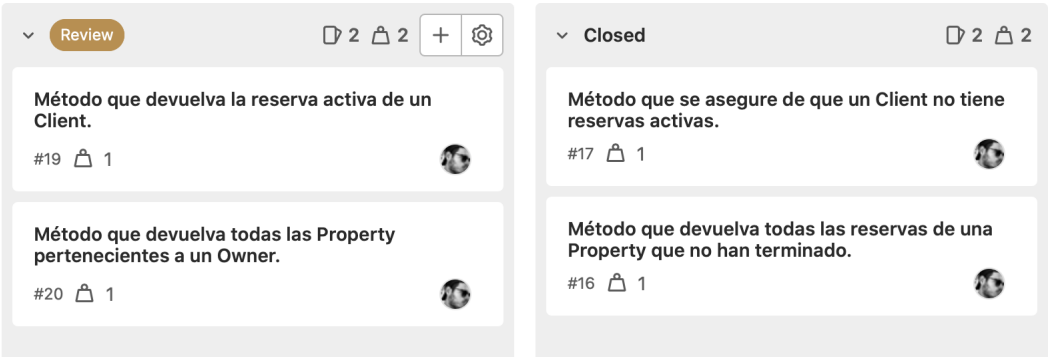


Figura 7.21: Sprint 7

Este Sprint se ha dedicado ha realizar las tareas que se muestran en la foto, un par de ellas al finalizar el Sprint han quedado en estado review, ya que falta probarlas aunque ya estén hechas. La mayoría de ellas han sido solicitudes de mi compañero Pablo el cual las necesitaba para la instancia especifica que estaba construyendo.

Se ha encontrado con que algunas de las propuestas de servicio que se han realizado ya se habían realizado con anterioridad con otro fin y podrían ser reutilizables para la complejidad que se planteaba actualmente. Esto hace que el tiempo de dedicación sea menor en alguna de las tareas, incluso se ha logrado el no tener que poner una historia por valer otra funcionalidad ya implementada.

Si se tiene que destacar alguna tarea que ha costado más que el resto, sin duda la de recoger las reservas activas de un usuario, ya que se tiene que contar con que hay dos tipos de reservas dependiendo de la propiedad, por lo que se hace algo más complejo las consultas a la base de datos.

7.0.8. Sprint 8 (01/06/2022 - 14/06/2022)

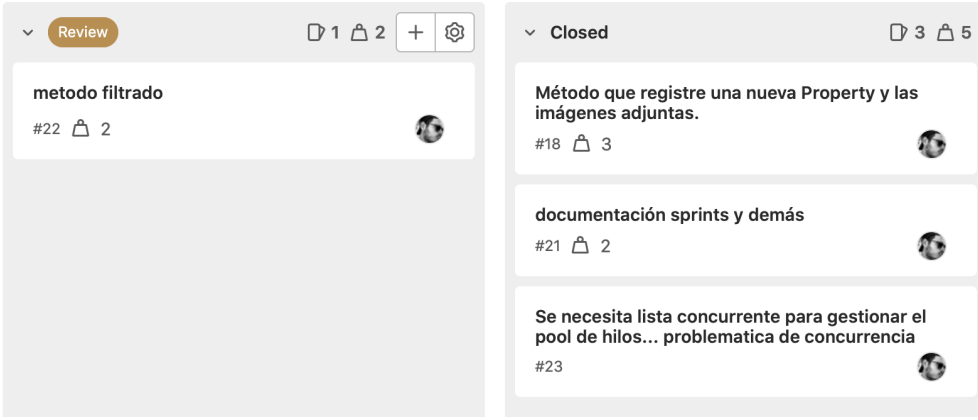


Figura 7.22: Sprint 8

Este Sprint se ha dedicado a acabar cosas del Sprint anterior por lo que ha llevado más horas, y aparte abordar las issues de este Sprint. Lo que más ha costado sin duda es el método de filtrado, y el más laborioso el generar y almacenar las imágenes asociadas a las propiedades. El método de filtrado ha llevado más tiempo por la principal característica que se incluyó donde (varios tipos de reserva) y que haya muchas condiciones y alguna se pueda pisar con otra.

El almacenamiento de imágenes se hace a través de una decodificación de la base 64 que se pasa del cliente y se trasforma en una foto que se guarda en un directorio especifico del servidor. De esta forma la instancia cuando pide la foto, se le proporciona la ruta correspondiente a la imagen pedida.

7.0.9. Sprint 9 (15/06/2022 - 28/06/2022)

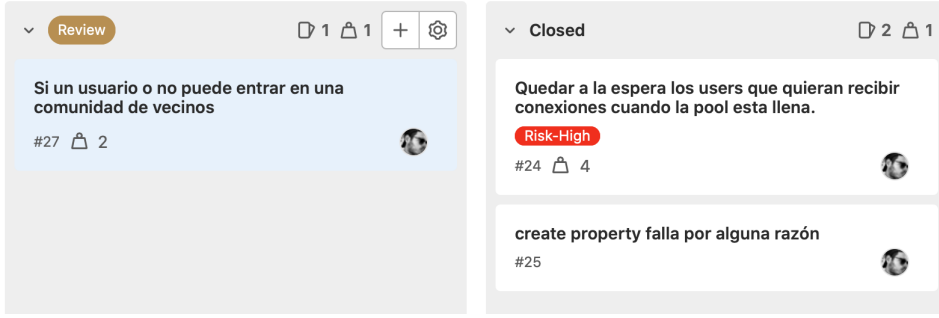


Figura 7.23: Sprint 9

Este sprint se ha dedicado principalmente a tratar el acceso a la base de datos desde el servidor. Se hace un wrapped genérico para tratar las conexiones hacia la base de datos.

```
/**
 * Wrapper de conexiones hacia el datasource.
 * Carlos Noé Muñoz (cnoemunoz@gmail.com)
 */
public class BasicConnectionPool {
    private static HikariConfig config = new HikariConfig();
    private static HikariDataSource ds;

    static {
        Properties props = null;
        try {
            props = loadProperties();
        } catch (IOException e) {
            System.out.println("Error al cargar las propiedades de la base de datos:
        }
        if (props != null) {
            config.setJdbcUrl(props.getProperty("jdbcUrl"));
            config.setUsername(props.getProperty("username"));
            config.setPassword(props.getProperty("password"));
            config.setMaximumPoolSize(
            Integer.parseInt(props.getProperty("maximumPoolSize")));
        }
        ds = new HikariDataSource(config);
    }

    private BasicConnectionPool() {}

    public static Connection getConnection() throws SQLException {
        return ds.getConnection();
    }

    private static Properties loadProperties() throws IOException {
        Properties props = new Properties();
        try (InputStream input = BasicConnectionPool.class.getClassLoader()
        .getResourceAsStream("db.properties")) {
            if (input == null) {
                throw new IOException("No se pudo encontrar"
                + "el archivo db.properties");
            }
            props.load(input);
        }
        return props;
    }
}
```

Capítulo 8

Conclusiones

En esta sección se hará una comparación entre la situación previa al realizar el proyecto y la situación posterior a terminar el TFG, Como si fuesen dos Snapshots donde veamos las mejoras ocurridas a lo largo del tiempo.

- Tras haber abordado diferentes problemas que se presentaban en el camino, y teniendo que abordar una solución para ellos, se ha hace notable la mejora en la capacidad de análisis, siendo una de las aptitudes fundamentes de todo ingeniero de desarrollo.
- Gracias al apoyo en diferentes tecnologías y métodos de trabajo se ha logrado una mejora distribución y planificación del trabajo.
- Descubrimiento de nuevas tecnologías y aplicación de nuevos métodos de trabajo que solo se habían planteado en escenarios teóricos. Puede ser el ejemplo del desarrollo con sistemas ágiles, o la aplicación de principios de diseño.
- Se fortalece la capacidad de trabajo en equipo, siendo muy necesaria en entornos actuales y los puestos que se esperan. (El uso de metodologías ágiles para el desarrollo del proyecto ha potenciado enormemente esta característica).
- Aumento de la panorámica de tecnologías utilizadas, dando mas herramientas para resolver los mismos o mas problemas (Cuando la única herramienta que tienes es un martillo, todo problema comienza a parecerse a un clavo).
- Mejora de la parte ya no solo de la programación, si no de otros entornos ligados al desarrollo pero que no implicar programar, es el ejemplo de la fase de despliegue o fases iniciales de recogida en análisis.
- Realizar labores de aprendizaje que tengan que ver con nuevas tecnologías no es una perdida de tiempo para el proyecto. Es cierto que hace que el desarrollo del proyecto sea mas lento al inicio pero la curva de aprendizaje hace que tengas mas herramientas de trabajo en un futuro.

A continuación se muestran los requisitos y el cumplimiento de cada uno de ellos al final del proyecto:

- El arrendador podrá consultar el saldo que se ha generado por sus propiedades así como las retenciones impuestas por la plataforma. (hecho)
- El arrendador podrá poner a disposición de los usuarios una propiedad para el uso de los arrendatarios con una compensación económica. (hecho)
- El arrendador podrá consultar qué propiedades tiene y qué ingresos ha obtenido por cada una de ellas así como las restricciones impuestas por la plataforma. (hecho)
- El arrendador podrá añadir, modificar y eliminar tanto propiedades como características de esas propiedades. (hecho)
- El arrendador podrá solicitar retornar a su cuenta bancaria el dinero ganado por las propiedades alquiladas. (hecho)
- El arrendador podrá usar dos modos de alquiler de la propiedad, bien que tenga dos estados; libre o ocupado, o bien una reserva por franjas horarias. Ambas opciones de reserva puede llevar consigo la posibilidad de una limitación horaria (en franja horaria) donde no será posible admitir a más clientes. Esta funcionalidad adicional se crea a partir de la necesidad que surge en un usuario de solo poner una plaza de aparcamiento a disposición de los arrendatarios solo en los momentos en el que el arrendatario no se encuentre usándola. (hecho)
- Se podrá obtener las reservas de una determinada propiedad (hecho)
- Se podrá crear booking a través de una propiedad (hecho)
- Se podrá ver las propiedades activas por el usuario
- Se podrá actualizar una reserva. (hecho)
- Se podrá obtener la reserva que tiene activa una propiedad. (hecho)
- Se podrá ver si esta ocupada o no una propiedad en concreto. (hecho)
- Se podrá obtener precio reserva hasta el momento actual o hasta la finalización de la misma. (hecho)
- Se podrá obtener precio reserva hasta el momento actual o hasta la finalización de la misma a través de su id. (hecho)
- Se podrá obtener todas las reservas de una determinada propiedad. (hecho)
- Se podrá obtener el booking a través de su id. (hecho)
- Se podrá eliminar booking a través de su id. (hecho)
- Se proporcionará un método de apertura de la puerta con el nombre de logInIdBooking. (hecho)

- Se proporcionará un método de cierre de la puerta con el nombre de logOutIdBooking. (hecho)
- Se proporcionará un método de apertura de la puerta a través de RFID con el nombre de logInWithRFID. (hecho)
- Se proporcionará un método de cierre de la puerta a través de RFID con el nombre de logOutWithRFID. (hecho)
- Se podrá obtener todas las propiedades. (hecho)
- Se podrá obtener una propiedad con sus reservas hasta la fecha asociadas a través de su id. (hecho)
- Se podrá crear una propiedad. (hecho)
- Se podrá actualizar una propiedad (hecho)
- Se podrá ver las propiedades pertenecientes a un usuario. (hecho)
- Se podrán ver propiedades pertenecientes a un usuario con sus booking asociadas. (hecho)
- Se podrá eliminar una propiedad a través de su id
- Se podrá postear una foto o fotos de una propiedad con formato base 64. (hecho)
- Se podrá obtener foto o fotos a una propiedad con formato que indica la ruta donde esta decodificada esa imagen. (hecho)
- Se podrá obtener los ingresos de una determinada propiedad. (hecho)
- Se podrá crear un cliente para el acceso a pagos de la plataforma implementada que permita la interacción económica con el arrendador. (hecho)
- Se podrá crear un cuenta para la recepción de pagos de la plataforma implementada que permita la interacción económica con el arrendatario. (hecho)
- Se podrá añadir una cuenta bancaria a un arrendador para recibir los pagos de los arrendatarios. (hecho)
- Se podrá añadir una tarjeta de crédito a un arrendador para pagar a los arrendatarios. (cumplido)

Finalmente concluimos la realización de todos los requisitos iniciales que se propusieron a la formación del sistema.

8.1. Líneas de trabajo futuras

Se ha realizado un buen trabajo para el desarrollo del proyecto, pero aun así no ha acabado aquí la funcionalidad que pueda proporcionar Sparkea. Sparkea se presenta como un suministrador de soluciones orientado a instancias que desean implementar un alquiler por horas de sus servicios. Pero no todo problema es abordable de forma sencilla y no todos los problemas o necesidades salen a la luz el día de hoy. El tiempo es una buena dimensión para darnos cuenta de cuáles son las tendencias que marcarán el camino de todos estos proyectos. Hoy en día existen infinidad de soluciones software que prometen solventar problemas actuales, pero realmente si no existe una integración constante en el producto adaptándose a los nuevos problemas, pronto acabará desactualizandose.

Se propone una lista de cosas que mejorar o implementar en un futuro para potenciar la funcionalidad de Sparkea.

- Aumento de la capacidad de los servicios de pagos entre clientes. Actualmente se cuenta con servicio de pagos de cuenta bancaria entre clientes con tarjeta de crédito, pero se pueden dar algunos casos mas, como es el ejemplo del pago a través de criptomonedas.
- Replanteamiento de alguna estructura como puede ser el paquete de Login que accede mediante JPA, mientras que en todos los demás paquetes se hace un acceso JDBC. Se puede mejorar para intentar hacer accesos unificados.
- Mejora de la latencia de los servicios. Puede ser el caso de la apertura de la puerta o de la obtención de servicios a través de una instancia.
  (Actualmente no hay una excesiva latencia, teniendo en cuenta que el servicio de database se encuentra en UK).
- Aumento de la capacidad de usuarios concurrentes. Aunque se ha preparado el servidor para ello, no se ha sometido a unas pruebas con con fuerza contra el servidor. En los casos de test ha funcionado, pero no quiere decir que con un escenario de muchos mas usuarios se siga funcionando igual. Todo depende en la maquina y las condiciones de la misma donde se instalen los servicios, red y otros factores determinantes.

Espero que este trabajo pueda servir para un uso real aplicado al mercado, y se pueda así continuar y perfeccionar su desarrollo y la incorporación de nuevas funcionalidades, que sirvan de aplicación a nuevos casos de uso que fueran saliendo en el mercado.

Bibliografía

[1] *Maven*, https://maven.apache.org

[2] *Spring*, https://spring.io

[3] *Stripe*, https://stripe.com/en-gb-es

[4] *DigitalOcean*, https://www.digitalocean.com

[5] *Issue Board*, https://docs.gitlab.com/ee/user/project/issue_board.html

[6] *GitKraken*, https://www.gitkraken.com

[7] *Atlassian*, https://www.atlassian.com/software/jira

[8] *Principios Solid*, https://profile.es/blog/principios-solid-desarrollo-software-calidad/

Apéndice A

Resumen de enlaces

Los enlaces de interés en este Trabajo Fin de Grado son:

A.1. Enlaces

- Repositorio del código: `https://gitlab.inf.uva.es/tfg-rent/alquileresAPI`.
- URL de la database empleada.

Listing A.1: mysqldump

```
#!/bin/bash
mysqldump --user=cnoe --password=cnoe --all-databases
> /backups/backup_22_11_22.sql
```

- URL postman librería `https://api.postman.com/collections/17511452-0fe16869-4f27-4ec6-9504-c06eefa90896?access_key=PMAT-01GWFC4WKNPB6F0EQ4X454ZGT8`

Apéndice B

Manuales

B.1. Manual de despliegue e instalación

El arranque de una aplicación con Spring Boot puede realizarse de una forma sencilla. Esto se puede hacer mediante el uso de Maven, el gestor de paquetes y dependencias que usamos a través del siguiente comando:

```
mvn spring-boot:run
```

Este comando arranca la aplicación utilizando las configuraciones predeterminadas y la pone a disposición en el puerto predeterminado.

Además, Spring Boot ofrece soporte para la depuración de la aplicación. Para iniciar la aplicación en modo de depuración, se debe especificar una serie de argumentos al arrancar la JVM adicionales. A continuación se muestra cómo se puede hacer esto:

```
mvn spring-boot:run -Dspring-boot.run.jvmArguments="-Xdebug \
-Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=5005"
```

El comando para arrancar en modo debug se compone de las siguientes partes:

- `-Xdebug`: Este parámetro habilita el modo de depuración de la Máquina Virtual de Java (JVM).
- `-Xrunjdwp`: Este parámetro permite configurar las opciones de depuración. Consta de varios subparámetros:
    - `transport=dt_socket`: Establece el método de comunicación para la depuración. En este caso, se utiliza un socket de datagrama.

- `server=y`: Indica que esta JVM actuará como servidor de depuración.
- `suspend=y`: Hace que la JVM se detenga al inicio y espere una conexión del depurador antes de comenzar a ejecutar el programa.
- `address=5005`: Establece el puerto donde el depurador puede conectarse.

De esta manera, se puede arrancar y depurar una aplicación Spring Boot.

Para la configuración de la database se ha de modificar el fichero "db.properties" localizado en los resources de la aplicación y especificar los parámetros de conexión.

```
jdbcUrl=jdbc:mysql://142.93.35.77:3306
username=cnoe
password=....
maximumPoolSize=30
```

y como se ha mencionado anteriormente, la configuración del servicio de login, se hace a través de frameworks ORM, y debemos modificar el fichero aplication.properties:

```
security.basic.enable: false
server.port=8084
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
spring.datasource.url=jdbc:mysql://cnoe@142.93.35.77:3306/
    Users?createDatabaseIfNotExist=true
spring.datasource.username=cnoe
spring.datasource.password=...
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
server.error.include-message = always
```

Para el correcto funcionamiento de la API de pagos se ha de modificar el fichero stripe.properties e indicar la key asociada a nuestro servicio de Stripe:

```
key=sk_test_51KhkNBEr5jgnxncSsNZPE2dRKli4KJggoEjEH8ncsoeBb4ZTIpm
    meO5rNlWdmLcCFP1thtaQfZ4jFvGHR1anRPXp00B83Qln5H
```

Dicho identificador se consigue a través del registro un puesta en marcha del servicio de stripe en la plataforma.

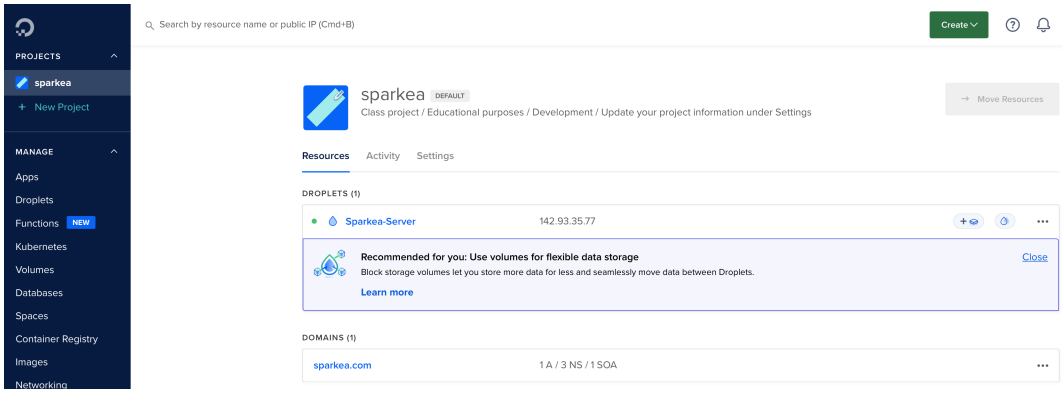
Para poner en marcha la database, se han de ejecutar los tres parches existentes en el directorio "patch".

Archivo	Descripción
Users.sql	Este archivo define la tabla de usuarios. Aquí se define toda la persistencia que pertenece al paquete de Login.
Access.sql	Este archivo define la tabla de acceso. Define toda la persistencia que pertenece al paquete de ac.
Payments.sql	Este archivo contiene la estructura de la tabla de pagos. Se definen las columnas para la información del pago como ID del pago, ID del usuario, cantidad, fecha, etc.

Tabla B.1: Parches SQL para la configuración de la base de datos

B.2. Manual de mantenimiento

Como la aplicación se encuentra almacenada en un SaaS, mensualmente se paga una cuota de mantenimiento de la aplicación, si se quiere mantener en estado activos los servicios se debe seguir pagando esa cuota o migrar el servicio a una opción gratuita. A continuación se muestra el dashboard de host donde está alojado Sparkea (hasta el momento), junto con su IP.

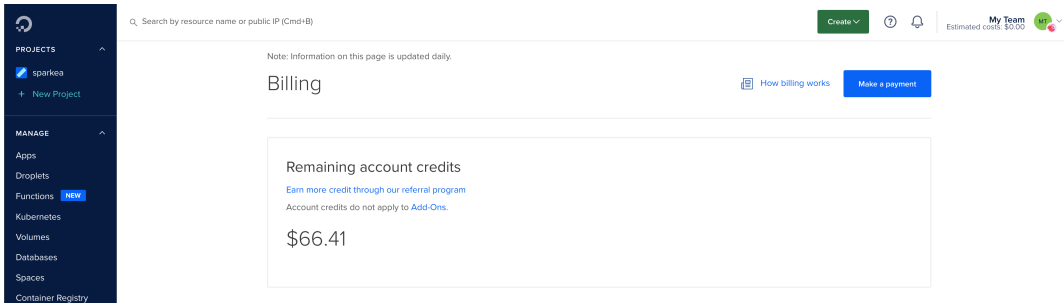


A continuación se muestra los costes que han llevado hasta ahora para mantener el servidor activo. Se encuentra también allí alojada la database con la que hemos cooperado ambos compañeros a la vez. Si se hubiese tenido una database en local, se habrían producido muchos incidentes a la hora de sincronización, derivando en una mayor perdida de tiempo.

El coste que se muestra, 66 dolares es lo correspondiente a droplet mantenido durante 5 meses, es decir unos 13,2 dolares mensuales de coste de mantenimiento.

image	region	ip
Ubuntu 20.04 (LTS) x64	Londres	142.93.35.77

RAM	Almacenamiento	tamaño
1GB	SSD	25GB



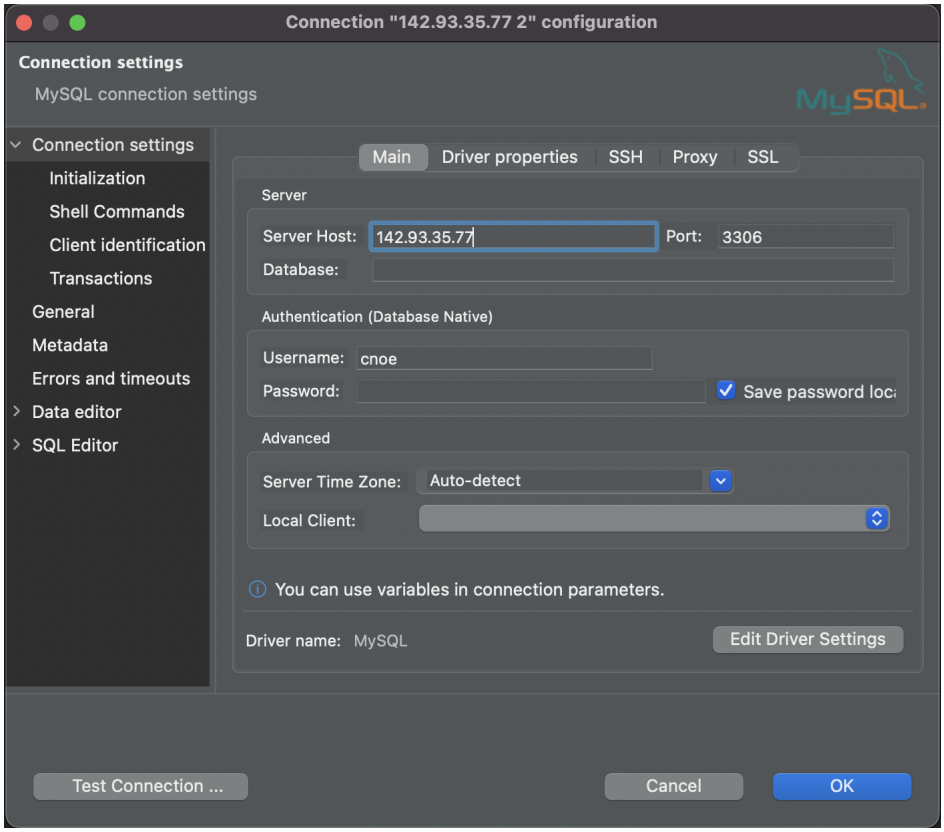
Para mayor seguridad se cuenta con una copia de seguridad de todas las databases interconectadas que conforman la aplicación. Se ha realizado con el comando, mysqldump que esta incluido desde la instalación de mysql. El comando completo para realizar la copia de seguridad de todas las databases es:

Listing B.1: mysqldump

```
#!/bin/bash
mysqldump --user=cnoe --password=... --all-databases
> /backups/backup_22_11_22.sql
```

Como se refleja en el comando anterior, en el servidor ubicado en la dirección IP 142.93.35.77, en la raíz del sistema, encontraremos la carpeta 'backups'. Dentro de ella, se almacenan las copias correspondientes a cada fecha en que se hayan realizado.

En caso de que necesitemos acceder directamente a la base de datos sin ingresar al servidor global, se recomienda el uso de DBeaver o algún cliente similar de base de datos. Durante la realización de este trabajo, personalmente he utilizado DBeaver como cliente. A continuación, procederé a explicar cómo configurarlo adecuadamente.



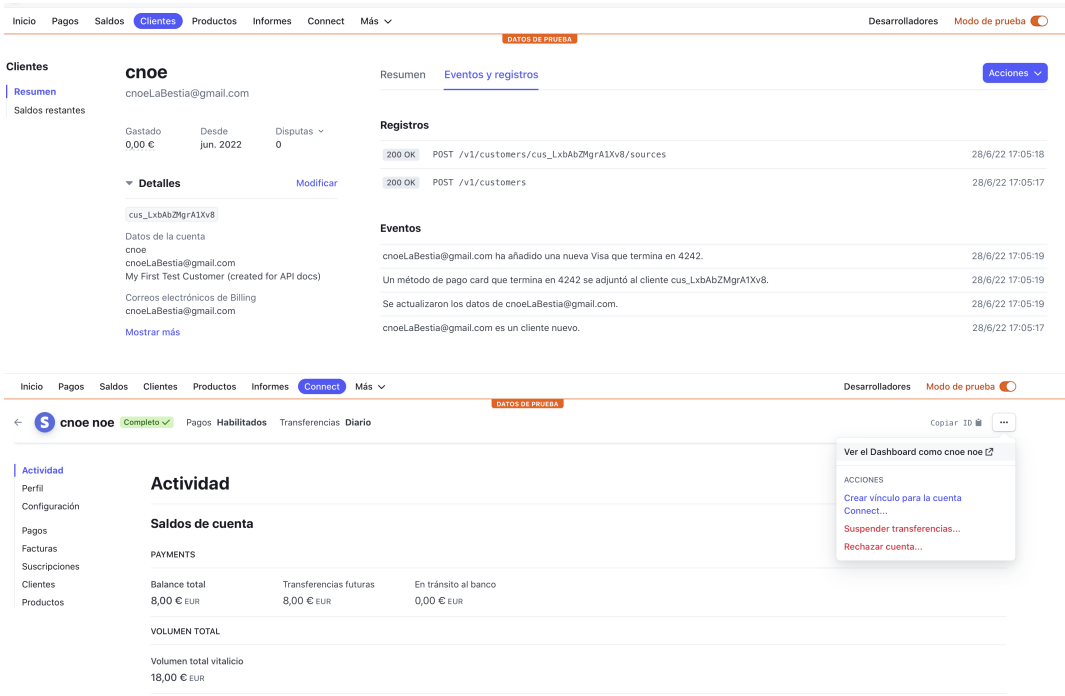
Existen otros servicios que operan en paralelo con la API. A diferencia de un modelo de pago recurrente, estos servicios generan costes únicamente cuando se ejecuta una operación a través de la API, lo cual es característico de algunos sistemas de cloud computing.

Una plataforma que se emplea en nuestro sistema es Stripe. Como se ha mencionado a lo largo del documento, esta plataforma actúa como pasarela de pagos en nuestra aplicación, integrándose en la infraestructura de Sparkea.

Cada vez que se procesa un pago a través de Stripe, la plataforma recibe una comisión correspondiente al tres por ciento del total, además de un coste fijo por cada operación realizada.

Nombre	Tipo Servicio	login
Stripe	Connect	correo

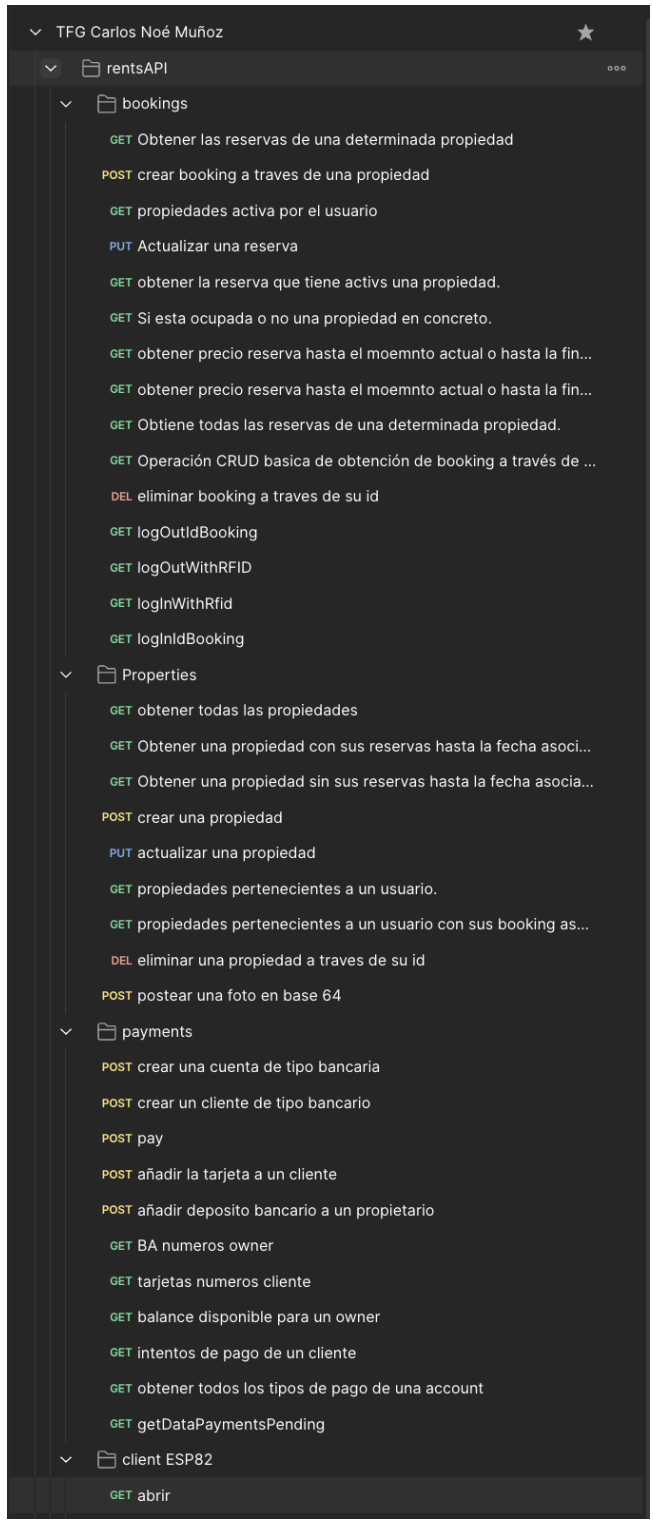
De momento está restringido el caso en el cada usuario pueda tener una visión de su dashboard, mostrando sus pagos, cobros etc, aunque se puede implementar. Se muestra pantallazos.



B.3. Manual del programador

En este proyecto se han desarrollado fundamentalmente servicios y no hay una interfaz de cara al usuario, por lo que se brinda un manual del programador con la documentación de los servicios desarrollados. Se proporciona el enlace a una librería en postman (cliente REST) que da la oportunidad de interactuar con el servidor con todo tipo de ejemplos de los servicios proporcionados. (El enlace se ha de abrir en la aplicación).
`https://api.postman.com/collections/17511452-0fe16869-4f27-4ec6-9504-c06eefa90896?access_key=PMAT-01GWFC4WKNPB6F0EQ4X454ZGT8`

y a continuación una imagen del conjunto de la librería creada.


TFG Carlos Noé Muñoz
rentsAPI
bookings
GET Obtener las reservas de una determinada propiedad
POST crear booking a traves de una propiedad
GET propiedades activa por el usuario
PUT Actualizar una reserva
GET obtener la reserva que tiene activs una propiedad.
GET Si esta ocupada o no una propiedad en concreto.
GET obtener precio reserva hasta el moemnto actual o hasta la fin...
GET obtener precio reserva hasta el moemnto actual o hasta la fin...
GET Obtiene todas las reservas de una determinada propiedad.
GET Operación CRUD basica de obtención de booking a través de ...
DEL eliminar booking a traves de su id
GET logOutIdBooking
GET logOutWithRFID
GET logInWithRfid
GET logInIdBooking
Properties
GET obtener todas las propiedades
GET Obtener una propiedad con sus reservas hasta la fecha asoci...
GET Obtener una propiedad sin sus reservas hasta la fecha asocia...
POST crear una propiedad
PUT actualizar una propiedad
GET propiedades pertenecientes a un usuario.
GET propiedades pertenecientes a un usuario con sus booking as...
DEL eliminar una propiedad a traves de su id
POST postear una foto en base 64
payments
POST crear una cuenta de tipo bancaria
POST crear un cliente de tipo bancario
POST pay
POST añadir la tarjeta a un cliente
POST añadir deposito bancario a un propietario
GET BA numeros owner
GET tarjetas numeros cliente
GET balance disponible para un owner
GET intentos de pago de un cliente
GET obtener todos los tipos de pago de una account
GET getDataPaymentsPending
client ESP82
GET abrir