Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica

Universitat de Barcelona

Property Management System Web: análisis y ampliación de la implementación del PMS Web de Tesipro

---

Juan Luis Espinoza López

Director: Lluis Garrido

Realizado en: Departamento de Matemáticas y Informática UB

Resumen

Hoy en día, para cualquier clase de hotel u hostal, el poder contar con un sistema informático que les permita gestionar las diferentes tareas cotidianas, tanto las de recepción como las administrativas, es un tema de suma importancia. Por esa razón, el poder tener acceso a un PMS (Property Management System, en inglés) no debe ser un problema para esos alojamientos turísticos.

Con más de 350 clientes en España, entre cadenas hoteleras y hoteles independientes, Tesipro se sitúa a la cabeza de su sector ofreciendo la tecnología informática más avanzada al servicio del hotelero. La empresa reúne también clientes en el extranjero, en ciudades como Londres, Bruselas, Praga, Munich, Roma y Viena, dirigiendo parte de sus esfuerzos a aumentar y afianzar cada vez más su presencia a nivel internacional

Tesipro lleva a cabo un nuevo proyecto con el objetivo de obtener un nuevo PMS, un PMS más moderno, un PMS que se adapte a las nuevas tecnologías, usando nuevas herramientas y medios que nos proporciona la informática. Personalmente, he ampliado la API (BackEnd) ya desarrollada, usando Java y una seria de frameworks como Spring o Hibernate y parte de la interfaz gráfica usando HTML, AngularJS, Javascript, además de otras extensiones y librerías de este último (FrontEnd).

El objetivo del desarrollo del producto es proporcionar a los hoteles la posibilidad de alojar todo el software y la información online. Disponer de todos los datos en la nube permite un fácil acceso desde cualquier lugar, en cualquier momento y a través de cualquier plataforma o dispositivo. Aunque las pantallas de los smartphones no son lo ideal para gestionar un PMS, las apps para teléfonos y tablets están ya a la orden del día.

Abstract

Nowadays, for any kind of hotel or hostel, having a computer system that allows them to manage the different daily tasks, both reception and administrative, is an issue of extreme importance. For this reason, having access to a PMS (Property management system) should not be a problem for these tourist accommodations.

With more than 350 clients in Spain, between hotel chains and independent hotels, Tesipro is at the top of its sector offering the most advanced computer technology to the hotelier. The company also brings together customers abroad, in cities such as London, Brussels, Prague, Munich, Rome and Vienna, directing part of its efforts to increase and consolidate its presence at an international level.

Tesipro carries out a new project, aiming to get a new PMS, a more modern PMS, a PMS that adapts to new technologies, using new tools and means provided by computer science. Personally, I have developed part of the API (BackEnd) using Java and frameworks like Spring or Hibernate and part of the graphical interface using HTML, AngularJS, Javascript, in addition to other extensions and libraries of the latter (FrontEnd).

The goal of the product development is to provide hotels with the possibility to get the software and our information online. Having all of our data in the cloud allows easy access from anywhere, at any time and through any platform or device. Although smartphone screens are not ideal for managing a PMS, the apps for phones and tablets are present in our daily life.

Agradecimientos

En primer lugar, quiero agradecer a la empresa Tesipro Solutions, que me ha brindado la oportunidad no solo de tener mi primera experiencia profesional, sino también de permitirme hacer este proyecto final de carrera.

Además, estoy infinitamente agradecido no solo a mis compañeros de proyecto, de quienes he aprendido muchísimo, sino a otros compañeros en la empresa quienes me acogido como uno más en la empresa y han hecho sentirme muy a gusto durante la estadía.

En segundo lugar, agradecer a mi familia y amigos de siempre, todo el apoyo que también me han aportado, así como todas las experiencias vividas que hemos tenido juntos y que me han llevado a ser como soy y a poder estar donde estoy.

Quisiera también agradecer a mi tutor en este proyecto, el profesor Lluis Garrido, quien se ha mantenido en contacto conmigo en todo momento para tener un seguimiento del proyecto y me ha ayudado con recomendaciones respecto a la elaboración de la memoria.

Finalmente, quiero agradecer a todos los profesores que he tenido en esta facultad, aunque a algunos los haya conocido de manera fugaz, todos los conocimientos y valores que me han transmitido en mi paso por la universidad, ya que espero y estoy convencido de que los mismos me serán de gran ayuda en mi futuro profesional y personal.

Índice

1. Introducción

1.1. Que es un PMS?

Hoy en día, para cualquier clase de hotel, el poder contar con un sistema informático que les permita gestionar las diferentes tareas cotidianas, tanto las de recepción como las administrativas, es un tema de suma importancia. Por esa razón, el poder tener acceso a un PMS moderno no debería ser un problema para esos alojamientos turísticos.

Entonces, si nos enfocamos principalmente en cadenas hoteleras u hoteles independientes y pensamos en sus necesidades principales, con respecto a ese software de gestión hotelera, podríamos decir que él debería contar con 3 características principales:

1- Flexibilidad

Al hablar de flexibilidad, me refiero a la facilidad con la que podamos configurar y adaptar esa herramienta de gestión a las peculiaridades o características de cada hotel (tipologías de las habitaciones, planificación de la limpieza, personalización de las ofertas, facturación, etc.), sin perder fluidez a la hora de trabajar con él. El PMS tiene que poder integrar en él todas las reservas hechas a través del motor de reservas o del Channel Manager e introducirlas directamente en el planning (o calendario) de forma automática o por lo menos sencilla.

2- Accesibilidad

Accesible en cuanto al precio. Porque esa es una cualidad sumamente importante para todos aquellos pequeños hoteles u hostales a los que nos venimos refiriendo.

El PMS siempre es una inversión que debemos planificar a largo plazo (no se adquiere un PMS para usarlo tan solo unos meses, sino más bien algunos años). Por esa razón, si el mismo nos cuesta una fortuna, no podremos amortizar ese gasto rápidamente.

3- Simplicidad

Con simplicidad nos referimos a que no deberíamos tener la necesidad de realizar un master para poder comprender el uso de ésta herramienta. El PMS tiene que ser lo suficientemente intuitivo como para que el personal de nuestro alojamiento turístico pueda comprender sin grandes problemas su uso.

Es importante también que ese sistema de gestión hotelera sea rápido de configurar y modificar, además de compatible con cualquier ordenador.

Figura 1: vista de un PMS tradicional

1.2. Que es un PMS Web?

Existe una generación de sistemas de gestión hotelera que pueden aportarnos los tres parámetros de los que hablábamos, además de sumarles la disponibilidad de ese mismo PMS en todo momento y desde cualquier lugar.

La gran novedad en este campo es, sin duda, la posibilidad de alojar todo el software y nuestra información online, trabajando en modelo **SaaS** (**Software as a Service**). Disponer de todos nuestros datos así como el software en la nube permite un fácil acceso desde cualquier lugar, en cualquier momento y a través de cualquier plataforma o dispositivo (aunque las pantallas de los smartphones no son lo ideal para gestionar un PMS, las apps para teléfonos y tablets están ya a la orden del día).

Los PMS Web son plataformas desarrolladas con la última tecnología, customizables y multiplataforma. Se integran con la web, el motor de reservas y con algunos de los principales Channel Manager del sector. Permitirán optimizar la gestión del negocio, aumentar ventas y reducir costes operativos.

A continuación explicaré algunos tipos de tecnologías usadas por establecimientos turísticos u hoteleros para automatizar los procesos de reserva y de gestión.

Channel manager

El channel manager nos ayuda a distribuir nuestra disponibilidad y precio a todos los portales de venta desde un único sitio. Es una herramienta informática para la optimización de la distribución online hotelera, no han sido muy utilizados hasta ahora en hoteles de destinos vacacionales o de

tamaño reducido, pero a medida que la venta online ha aumentado y han aparecido en el mercado numerosas soluciones informáticas al alcance de todos, han pasado a tener un papel protagonista en hoteles de todo tipo.

Si disponemos de un channel manager, el PMS y el channel manager deben tener una perfecta integración, si bien es cierto que algunos PMS ya incluyen su propio channel manager. Hay soluciones para todos los gustos, pero conviene estudiar cada pequeño aspecto para acertar con la elección de un software PMS que sea adecuado para el hotel.

Figura 2: representación de un channel manager

Booking engine

El motor de reserva, más conocido como booking engine va integrado al PMS y a la WEB. Por qué es interesante disponer de un Booking Engine en un hotel? Su finalidad es permitir a los clientes reservar en el hotel a través de su canal directo. De esta forma, el hotel se asegura que el cliente ha contratado su estancia a través del propio hotel y esto, a su vez, permite un ahorro considerable de comisiones al alojamiento respecto a las reservas que se realizan a través de una OTA (portales, comparadores o agencias de viajes online en las que aparece un hotel).

1.3. Contexto y presentación del problema

Durante décadas, los hoteles han utilizado softwares para gestionar su trabajo y automatizar las tareas. Con el tiempo y la revolución tecnológica, los sistemas han evolucionado significativamente. Sin embargo, ha llegado el momento de ir más allá, mejorar aún más si cabe el PMS para que pueda apoyar a los modernos hoteleros en la solución de sus nuevos desafíos.

Atrás quedaron los días en que los sistemas de gestión hotelera se utilizan sólo para gestionar las reservas, inventario y check-in / out de los huéspedes. Con la nueva demanda proveniente de una nueva generación de viajeros y un ambiente de negocios completamente nuevo, los sistemas hoteleros necesitan un cambio de imagen completo: la gestión de canales OTA (Online Travel Agencies, en inglés), el motor de reservas web, el compromiso con los clientes y la lealtad.

Tener menos, o incluso mejor, tener un único software de gestión hotelera que ofrezca una serie de características y capacidades, es la mejor opción. Te hace la vida más fácil y tu trabajo más productivo al mismo tiempo.

La última década se ha visto el surgimiento de una nueva generación de PMS hotel con la introducción soluciones ágiles, flexibles y rentables basadas en la nube. Hoy en día, muchos hoteles están recurriendo cada vez más a nuevas tecnologías para impulsar la eficiencia operativa y comprometerse con los clientes**.** A continuación comentaré algunas de las mejoras que tendrá el PMS Web a diferencia de otros más antiguos.

- **Cloud**: la computación basada en la nube ya es la nueva norma para muchas industrias y negocios y se está volviendo más y más popular en todo el sector hotelero. La nube ofrece una plataforma para la innovación, así como el potencial para mejorar la eficiencia de los procesos de negocio, lo que puede convertirse en una fuente importante de ventaja competitiva.
  La mayor ventaja de la nube es el costo. Las instalaciones de PMS tradicionales a menudo pueden ser costosas y muchos están anticuados. Trabajar en la nube elimina la necesidad de hardware de servidor en el sitio y los gastos asociados; también es dinámico y escalable para el futuro. Además de estos ahorros, el PMS basado en la nube también ofrece ventajas inmediatas como el acceso remoto que permite el acceso en tiempo real, la gestión de la distribución, las tareas automatizadas, la gestión de propiedades múltiples, el análisis de datos en tiempo real, entre otras.

- **Movilidad**: la movilidad ha cambiado casi todo sobre la forma en la que se negocia. Hoy en día más y más propietarios de hoteles quieren realizar típicas operaciones de gestión de propiedades con dispositivos móviles. Una de las características de los sistemas antiguos y tradicionales es literalmente su incapacidad de moverse en tiempo real. Los sistemas antiguos por su propia naturaleza son sistemas fijos, lo que significa que los empleados están vinculados a un punto fijo. Sin embargo, con los nuevos PMS, la tecnología ha evolucionado hasta el punto que tiene la capacidad de entregar cualquier tipo de datos a cualquier tipo dispositivo en un momento dado. Aprovechando el modelo SaaS (software as a service), los sistemas de gestión de hotel que utilizan tablets y smartphones como dispositivo de hardware principal, hacen que los empleados estén capacitados para ir más allá y poder proporcionar una atención personalizada al cliente.

- **Compromiso con los huéspedes**: una característica muy atractiva de la tecnología en la nube es su capacidad para consolidar información pertinente de la estancia de huéspedes y datos de preferencias (de múltiples fuentes) en un solo lugar. Estos datos recopilados proporcionan una visión real de la estancia de los huéspedes y de los hábitos de gasto que permiten a los hoteleros promover ofertas en tiempo real y vender productos auxiliares, servicios y / o paquetes personalizados a los clientes adecuados en el momento adecuado del viaje. Otra ventaja de PMS de la nube sobre un hotel con PMS antiguo o tradicional es la capacidad de tener acceso a datos usando cualquier dispositivo en cualquier momento, lo que significa que el personal puede conectar con sus huéspedes a través de su método preferido de comunicación y / o moverse libremente por el hotel proporcionando servicios en tiempo real, mejorando la experiencia global y aumentando la satisfacción de los clientes.

- **Integración interminable**: el PMS tradicional limita la libertad de los hoteleros para trabajar con varios vendedores diferentes. A menudo, no se integran fácilmente con el software de diferentes proveedores, por lo que si se desea agregar nuevas características al sistema de

gestión que no son ofrecidos por el proveedor de software o bien se desea integrar sus diferentes sistemas de gestión juntos, se tienen dificultades para hacerlo. La realidad es que con el software de gestión hotelera tradicional, la adición y la eliminación de recursos es difícil y los hoteleros están atascados en un sistema hasta la fecha de caducidad de la licencia.
Una de los puntos fuertes de un sistema de gestión de hoteles basado en la nube es que puede conectarse fácilmente a otras aplicaciones de software, incluso si estas aplicaciones no comparten el mismo proveedor.

- **Diseño elegante**: el PMS web permite una interacción fácil e intuitiva con el usuario final. Diseñados con interfaces gráficas, ofrecen una fácil experiencia de usuario que permite a los empleados ser entrenados rápidamente. Incluso los miembros menos expertos del equipo de un hotel podrían acceder fácilmente a la información que necesiten siguiendo los pasos concretos. Con la evolución de las tecnologías de los móviles, los negocios ahora también necesitan ser móviles. Con el crecimiento de los PMS basados en la nube, los hoteles ahora pueden invertir en herramientas poderosas sin tener que soportar el enorme costo de la infraestructura; herramientas que permiten a los hoteleros reducir significativamente los costes, aumentar la eficiencia mediante la simplificación de los procesos y la automatización de las operaciones, mejorar la rentabilidad, interactuar con los clientes en tiempo real y proporcionar una experiencia personalizada, aumentando así la satisfacción del cliente.

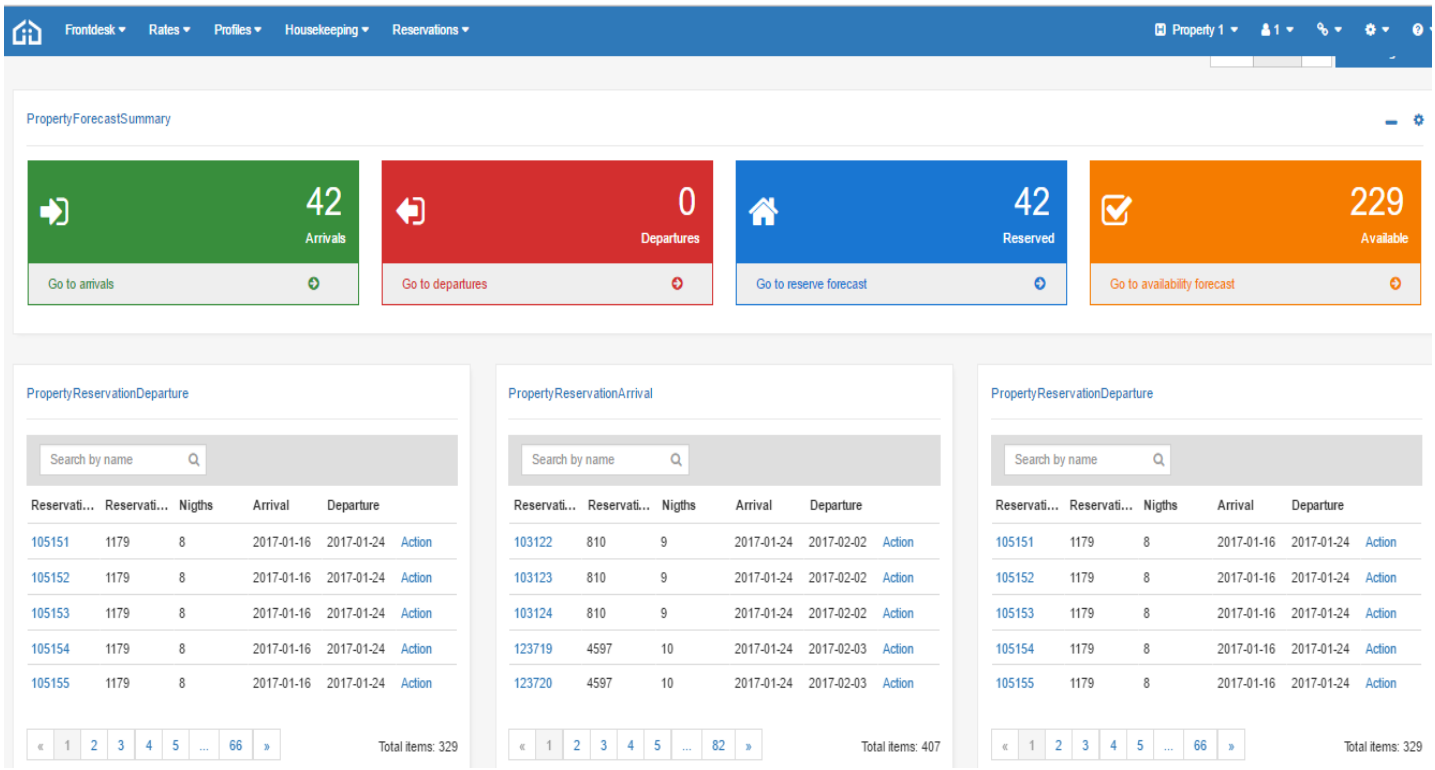


Figura 3: vista del PMS Web

1.4. Programador Frontend, Backend y Fullstack

Un desarrollador web no es una sola cosa, sino que abarca múltiples conjuntos de habilidades que se traducen en diferentes especialidades. Los tres términos más comunes que se utilizan para nombrar dichas especialidades de forma genérica son**: frontend, backend *y* fullstack**.

Frontend y backend son términos que se refieren a la separación de intereses entre una capa de presentación y una capa de acceso a datos, respectivamente.

En diseño de software, el frontend es la parte del software que interactúa con el o los usuarios y el backend es la parte que procesa la entrada desde el frontend. La idea general es que el frontend sea el responsable de recolectar los datos de entrada del usuario, que pueden ser de muchas y variadas formas, y los transforma ajustándolos a las especificaciones que demanda el backend para poder procesarlos, devolviendo generalmente una respuesta que el frontend recibe y expone al usuario de una forma entendible para este.

Programador frontend

Aquel programador que trabaja del lado cliente, en el navegador, en el lado de lo que se ve. Principalmente se ocupa de los componentes externos del sitio web o de la aplicación web.

En general se asocia a los desarrolladores frontend con los principios de diseño y de estructura de páginas. Sin embargo, un desarrollador web va más allá que un diseñador. Obviamente tiene que tener en cuenta la usabilidad y la legibilidad de la página o de la aplicación web, pero como buen programador es consciente de que su trabajo se ejecutará en el lado cliente, en la mayoría de los casos, en el navegador. Y la información no se almacena en el lado cliente.

Como consecuencia un desarrollador frontend deberían conocer lenguajes como:

- **HTML**: HyperText Markup Language, es el componente estructural clave de todas las webs de internet. Funciona a base de etiquetas para la estructuración y organización del contenido de la web. Sin él las páginas web no pueden existir.
- **CSS**: cascading style sheets, son hojas de estilos en cascada, encargada de dar formato al contenido, es decir, al HTML.
- **Javascript**: lenguaje de programación muy potente, orientado a objeto y desde hace muchos años usado para el desarrollo de aplicaciones web y hoy día y cara al futuro de los lenguajes que más se están extendiendo y evolucionando en el ámbito del desarrollo web con la cualidad de poder ser interpretado en equipo cliente y en cualquier navegador web e interactuar fácilmente con HTML y CSS entre otros.

Programador Backend

El programador backend trabaja del lado servidor, detrás del escenario, permitiendo con su trabajo que el usuario disfrute de su experiencia. Sin él, el desarrollo llevado a cabo por su anterior compañero no se sostendría.

Está enfocado a lenguajes de programación, orientado a funcionamientos, trabajar con datos internos, crear aplicaciones que controlen datos de la base de datos de la web, para poder ser

consultados por el usuario. Creación de funciones que realicen una acción o acciones que controlen el buen funcionamiento de las aplicaciones, crear estructuras de operaciones y cálculos para devolver unos resultados. Es una especialidad más orientada a una programación menos intuitiva y en ocasiones algo más compleja, ambas especialidades van a ir ligadas.

Para ser programador del lado servidor, son numerosos los lenguajes y frameworks entre los que elegir, todo dependerá de la empresa. A día de hoy, los más comunes son:

- **ASP.NET**: es la plataforma de desarrollo web de Microsoft. Muy utilizada en las empresas. Tiene las variantes Web Forms y MVC.
- **PHP**: por ejemplo, el famoso gestor de contenidos WordPress usa por detrás PHP. Laravel es uno de los frameworks usados con este lenguaje.
- **Ruby**: junto con su framework Ruby on rails.
- **Python**: fácil de aprender. Usado a menudo con Django como framework.
- **Node.js**: se está haciendo cada vez más popular debido a que usa el mismo lenguaje que en el lado cliente: JavaScript.

Programador Fullstack

Finalmente, el programador fullstack es un programador con un perfil técnico muy completo que conoce bien tanto lo referente a backend como lo referente a frontend, se maneja en sistemas y sabe entender. Es decir, un desarrollador fullstack se encargaría tanto de la parte de cliente, como la de servidor e incluso de la arquitectura de servidores y sistemas.

1.5. Objetivos

Los objetivos que tengo en este proyecto son básicamente el análisis y la documentación de lo que es un PMS, de todas las tecnologías usadas que se usan en este proyecto, así como las herramientas y el software que se va utilizar para llevar acabo el desarrollo de este producto.

Una vez documentado y analizado todo lo que se había hecho en el proyecto empecé a desarrollar los mantenimientos que se me pedían tanto en la parte del backend con la API como en la del frontend con la interfaz gráfica. Es cierto que en la empresa en la que he estado desde Julio del 2016 he hecho muchísimas cosas las cuales aquí no explicaré y me centraré en el desarrollo y la gestón de Allotments (cupos en español), Promotions (promociones en español) y Profiles (Perfiles en español).

1.6. Trabajo realizado

Antes que todo me gustaría recalcar que yo llegué a la empresa con el proyecto ya empezado y con muchas de las tecnologías de desarrollo ya elegidas. No tuve participación ninguna en la elección de las tecnologías que se usarían durante el proyecto, simplemente tuve que adaptarme al equipo, a la forma de trabajar y a las tecnologías o frameworks que ya se estaban utilizando.

Durante mi estadía en la empresa estuve desarrollando la aplicación tanto en la parte del backend como en la parte del frontend. En este aspecto, en la parte del backend le he dedicado la mayoría del tiempo a la ampliación de la API desarrollada con Spring Framework, usando como lenguaje de

programación Java, el cual he practicado a lo largo de la carrera de ingeniería informática y no me ha supuesto mucha dificultad la implementación. Otra parte no menos importante que traté en la parte de backend fueron las consultas a la base de datos, tanto las nativas como las consultas en JPQL (Java Persistence Query Language) la cual se podría considerar como una versión orientada a objetos de SQL.

En cuanto a la parte del frontend, me centré en la amplicación del desarrollo de algunas vistas de la aplicación, en donde se requerían frameworks desconocidos para mí como era NodeJS, del cual necesitaríamos el paquete NPM **(sección 3.3.5),** que nos permitirá a gestionar módulos Node. Además necesité el XAMPP **(sección 4.7)**, servidor el cual nos proporcionará un servidor web como es Apache. A partir de aquí todo el desarrollo del frontend se hace en JavaScript, HTML, CSS y AngularJS **(sección 3.2.1 – 3.2.4)**, además de una serie de librerías importadas. Angular JS ha sido un lenguaje que nunca antes había utilizado, por lo tanto al principio tuve que documentarme, mirar tutoriales, etc. para poder entender su funcionamiento.

Aparte de lenguajes de programación nuevos que he conocido y de muchos softwares de los cuales desconocía su función, este proyecto me ha ayudado a la hora de impregnarme de los conocimientos de mis compañeros de trabajo, de los cuales he aprendido mucho, no solo a nivel de conocimientos sino que me han ayudado a ver como es el mundo laboral, las responsabilidades que conlleva y las diversas metodologías que hoy en día son requeridas en cualquier empresa.

Antes de empezar a programar algunos mantenimientos en la API, compañeros del equipo de desarrollo, habían creado la base de datos, con sus respectivas tablas, triggers y procedimientos. Además se ha insertado datos falsos para una vez implementada la API, comprobar y testear que los métodos devolvían los JSON con la información correcta. A parte de la base de datos se crearon diagramas UML, los cuales me ayudaron a entender las relaciones de las tablas, en concreto se me proporcionaban diagramas de clases.

Normalmente, para cada entidad se crea su respectivo CRUD (Create. Read, Update, Delete, en inglés) básico, la lista de ese objeto, el objeto, modificar el objeto, crear el objeto y finalmente eliminarlo. Dependiendo de las funcionalidades que se requiera o de las relaciones con otras entidades, la API se tendrá que ampliar.

Una vez se implementaba el CRUD para las entidades y se testeaba la API mirando que el resultado fuera el correcto, es decir, se introducían datos falsos en la base de datos y comprobábamos que la API funcionaba correctamente. Por ejemplo, si se creaba el CRUD de una habitación, se tenía que testear que se devolvía, la lista de habitaciones, que devolvía información de una sola habitación pasándole la id de esta como parámetro, que se creaba una habitación nueva, que se podía editar la habitación, que se podía eliminar la habitación entre otros requerimientos.

Para finalizar, cuando se testeaba que la API funcionaba correctamente, se procedía a elaborar la interfaz gráfica en la parte del frontend. Se tuvo que instalar todo el software necesario para ello y por supuesto aprender las nuevas tecnologías que se usaban en este proyecto para llevar a cabo el desarrollo de la interfaz gráfica.

1.7. Estructura de la memoria

A continuación, pasaremos a detallar las secciones de las que consta la memoria, cada una con un breve resumen para que el lector pueda conocer de un vistazo de qué trata cada una:

Análisis de la aplicación

1. **Introducción**: se presenta el concepto y define lo que es un PMS y se compara con un PMS Web. Además de una presentación genérica de la aplicación, los objetivos y el trabajo realizado.
2. **Arquitectura de la aplicación**: se explica la estructura principal de la aplicación de forma concisa.
3. **Tecnologías utilizadas:** consta de un pequeño resumen de cada tecnología que se ha utilizado en el desarrollo de la aplicación, así como el motivo por el que se ha hecho uso de cada una de ellas.
4. **Herramientas de desarrollo:** consta de un pequeño resumen de cada herramienta que se ha utilizado en el desarrollo de la aplicación.
5. **Metodologías de trabajo:** consta de un pequeño resumen de cada metodología que se ha utilizado en el desarrollo de la aplicación, así como el motivo por el que se ha hecho uso de cada una de ellas.

Ampliación de la aplicación

6. **Desarrollo del backend:** consta de la explicación del desarrollo de la API, tanto la arquitectura de clases como las anotaciones que se utilizan en las clases.
7. **Desarrollo del frontend:** consta de la explicación del desarrollo de las diferentes vistas de la aplicación, así como la estructura de las carpetas, la arquitectura y el análisis del modelo.

2. Arquitectura de la aplicación

La arquitectura que se ha implementado en el desarrollo del PMS Web es la siguiente:

Figura 4: esquema de la arquitectura de la aplicación

- **Navegador web de usuario**: obtiene los datos del servidor y en él muestra la vista, que se genera modificando el modelo web, que consta a su vez de los ficheros HTML5 y CSS3, con los

controladores, que son ficheros JavaScript que obtienen del servicio web REST los datos a representar en dicho modelo y los imprimen o envía.

- **Servicio web**: servicio web con arquitectura REST, desarrollado enteramente mediante el uso del framework Java Spring Framework. Su función principal es coordinar la autenticación de usuarios y la realización de operaciones básicas CRUD (Create, Read, Update and Delete, en inglés) de usuarios, locales y eventos. Cabe destacar, que el formato de ficheros de intercambio entre el controlador web JavaScript y el servicio web Spring por el que el equipo de desarrollo optó fue JSON.
- **Base de datos**: La base de datos es el lugar donde almacenaremos todos los datos relacionados con información de usuarios, locales y eventos. Está diseñada siguiendo un modelo entidad-relación y usando el gestor de Bases de Datos SQL server.

A continuación explicaré con detalles las tecnologías que se han usado para llevar a cabo la implementación del PMS web.

3. Tecnologías utilizadas

Normalmente cuando nosotros trabajamos en cualquier plataforma solemos utilizar algún tipo de framework (como algunos de los que hemos mencionado anteriormente). Estos frameworks no son ni más ni menos que un esquema (un esqueleto, un patrón) para el desarrollo y/o la implementación de una aplicación y así facilitarnos trabajo cotidiano. Utilizamos los frameworks para crear un conjunto de objetos que nuestra aplicación necesita.

3.1. Tecnologías utilizadas en backend

3.1.1. SQL Server

SQL es un sistema de gestión de bases de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones; usado por muchos sitios web grandes y populares, como Wikipedia, Google (no para búsquedas), Facebook, Twitter, Flickr, y YouTube.

Utilización en el proyecto

Se ha optado por SQL como opción de almacenamiento de todos los datos de la aplicación, esto significa que utilizaremos dicho gestor con el fin de que guarde toda la información relacionada con usuarios, locales y eventos.

SQL permite combinar de forma eficiente diferentes tablas para extraer información relacionada, además permite gestionar los datos junto con las relaciones existentes entre ellos.

3.1.2. Servidor de aplicaciones

Se denomina servidor de aplicaciones a un servidor en una red de computadores que ejecuta ciertas aplicaciones. Un servidor de aplicaciones generalmente gestiona la mayor parte (o la totalidad) de las funciones de lógica de negocio y de acceso a los datos de la aplicación. Los principales beneficios de la aplicación de la tecnología de servidores de aplicación son la centralización y la disminución

de la complejidad en el desarrollo de aplicaciones, es decir proporciona servicios de aplicación a las computadoras clientes.

Utilización en el proyecto

En el desarrollo del PMS web se utilizará Tomcat, un servidor web con soporte de servlets y JSPs (JavaServer Pages). El motor de servlets de Tomcat a menudo se presenta en combinación con el servidor web Apache.

3.1.3. Conexión a la Base de datos (pool de conexiones)

Un pool de conexiones es un conjunto limitado de conexiones a una base de datos, que es manejado por un servidor de aplicaciones de forma tal, que dichas conexiones pueden ser reutilizadas por los diferentes usuarios. Este pool es administrado por un servidor de aplicaciones que va asignando las conexiones a medida que los clientes van solicitando consultas o actualizaciones de datos.

El pool de conexiones consiste en tener un conjunto (pool) de conexiones ya conectadas a la base de datos que puedan ser reutilizadas entre distintas peticiones.

Utilización en el proyecto

Cuando nuestra aplicación trabaja con bases de datos, lo que se realiza básicamente es obtener una conexión, realizar consultas u operaciones SQL y por último cerrar la conexión. El problema está en que las conexiones a bases de datos son limitadas en número y además suponen un coste de procesador abrirlas y cerrarlas.

El pool de conexiones se configurará en el archivo **application.properties** ubicado en la carpeta resources del proyecto ya que cada servidor web puede usar su propia implementación del pool de conexiones y se configura de forma específica.

```
# DATABASE
spring.datasource.url=jdbc:sqlserver://1                .\SQL2014;databaseName=PMS_V02
spring.datasource.username=
spring.datasource.password=
spring.datasource.driverClassName=com.microsoft.sqlserver.jdbc.SQLServerDriver
spring.datasource.maxActive=200
```

Figura 5: configuración del pool de conexiones con la base de datos.

3.1.4. Spring

Spring es un framework que se utiliza para el desarrollo de aplicaciones y a su vez es un contenedor de inversión de control, es de código abierto y utiliza Java como lenguaje de programación.

En la mayoría de las ocasiones para desarrollar la aplicación que necesitamos no nos es suficiente con usar un único framework sino que necesitamos utilizar varios frameworks. Cada uno de los cuales generará su propio conjunto de objetos.

Spring ayuda a solventar este problema ya que cambia las responsabilidades y en vez que el propio desarrollador sea el encargado de generar los objetos de cada uno de los frameworks, es Spring basándose en ficheros xml o anotaciones, el encargado de construir todos los objetos que la aplicación va a utilizar.

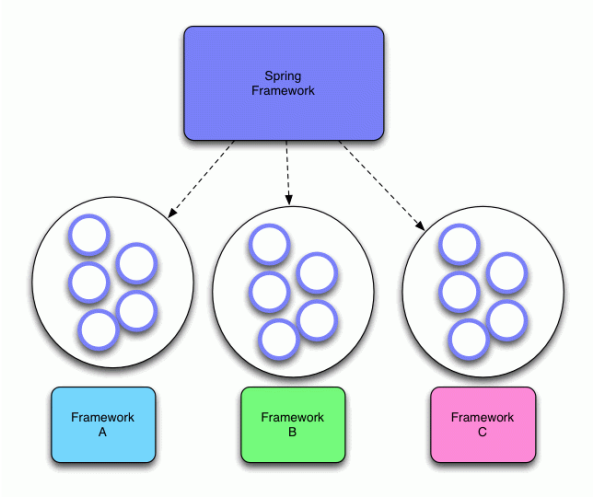


Figura 6: tratamiento de frameworks por Spring

Utilización en el proyecto

Se ha utilizado Spring como framework de construcción del servicio web REST, ya que aporta una gran cantidad de herramientas que elimina ciertas complejidades asociadas con el desarrollo de servicios web.

A grandes rasgos, se declara en un XML los componentes de la aplicación y sus dependencias. Spring lee este XML, crea los componentes y sus relaciones entre ellos. Las últimas versiones de Spring, ya permiten anotaciones, y se puede anotar una propiedad en una clase mediante **@Autowired** para que Spring busque la clase correspondiente, la instancie y la inyecte, ahorrándonos bastante código XML.

```
<!-- Documentation -->
<dependency>
    <groupId>org.jsondoc</groupId>
    <artifactId>spring-boot-starter-jsondoc</artifactId>
    <version>1.2.10</version>
</dependency>
<dependency>
    <groupId>org.jsondoc</groupId>
    <artifactId>jsondoc-ui-webjar</artifactId>
    <version>1.2.10</version>
</dependency>
```

Figura 7: declaración de dependencias en el archivo POM.xml

3.1.5. Maven

Maven proporciona a los desarrolladores un framework completo de ciclo de vida de la compilación. Facilita la vida del desarrollador al crear informes, verificaciones, crear y probar configuraciones de automatización.

Maven utiliza un fichero denominado Project Object Model (POM) para describir el proyecto software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado.

Utilización en el proyecto

Se ha utilizado Maven para producir los ficheros JAR del servicio web. Se ha optado por esta herramienta para la construcción del PMS web porque al definir en el fichero POM.xml las dependencias de librerías que necesitamos, procede a su descarga y compilación automáticamente al generar el JAR.

Además el proyecto JAR se habrá que pasar a WAR para poder desplegarlo en un servidor de aplicaciones. Se tendrá que cambiar el archivo pom.xml y se modificará la clase PmsApplication.

3.1.6. Hibernate – ORM

El mapeo objeto-relacional (en inglés**, Object-Relational mapping**, ORM) es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y la utilización de una base de datos relacional como motor de persistencia. En la práctica esto crea una base de datos orientada a objetos virtual, sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo).

Muchos paquetes han sido desarrollados para reducir el tedioso proceso de desarrollo de sistemas de mapeo relacional de objetos proveyendo bibliotecas de clases que son capaces de realizar mapeos automáticamente **(Hibernate/ORM)**. Dada una lista de tablas en la base de datos, y objetos en el programa, ellos pueden automáticamente mapear solicitudes de un sentido a otro.

Utilización en el proyecto

Se utiliza Hibernate, que es una herramienta de mapeo objeto-relacional (ORM) para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) o anotaciones en los beans de las entidades que permiten establecer estas relaciones.

Se sitúa entre la capa de acceso a datos de aplicaciones Java y la base de datos relacional. La aplicación Java utiliza las API de Hibernate para cargar, almacenar, consultar, etc. Sus datos de dominio.

3.2. Tecnologías utilizadas en frontend

3.2.1. HTML5

HTML, sigla en inglés de **HyperText Markup Language** (lenguaje de marcas de hipertexto), hace referencia al lenguaje de marcado para la elaboración de páginas web. Es un estándar que sirve de referencia del software que conecta con la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página web, como texto, imágenes, videos, juegos, entre otros

HTML5 (HyperText Markup Language, versión 5) es la quinta revisión del lenguaje básico de la World Wide Web, HTML. HTML5 especifica dos variantes de sintaxis para HTML: una «clásica», HTML (text/html), conocida como HTML5, y una variante XHTML conocida como sintaxis XHTML5 que se sirve de una sintaxis XML (application/xhtml+xml).

Utilización en el proyecto

Se ha utilizado HTML5 para desarrollar el modelo web, en conjunto con CSS3 y otras tecnologías sobre el que se representan todos los datos de usuario y que sirve como base para que el usuario tenga una interfaz gráfica para poder interactuar con el servicio web REST, en lugar de tener que hacerlo por otros medios.

3.2.2. CSS 3

Junto con HTML y JavaScript, CSS es una tecnología usada por muchos sitios web para crear páginas visualmente atractivas, interfaces de usuario para aplicaciones web, y GUIs para muchas aplicaciones móviles.

La idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación.

CSS3 es la nueva versión de CSS, retrocompatible con este último, por lo que pueden utilizarse conjuntamente. La principal diferencia es la separación en módulos respecto al CSS original.

Utilización en el proyecto

Se ha utilizado CSS3 para desarrollar el aspecto gráfico del modelo sobre el que se representan todos los datos de usuario en conjunto con HTML5 y que sirve como base para que el usuario tenga una interfaz gráfica para poder interactuar con el servicio web REST, en lugar de tener que hacerlo por otros medios.

3.2.3. Javascript

JavaScript (abreviado comúnmente JS) es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Se utiliza principalmente en su forma del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas aunque también existe una implementación de JavaScript en el lado del servidor. Su uso en aplicaciones

Utilización en el proyecto

Tradicionalmente se venía utilizando en páginas web HTML para realizar operaciones y únicamente en el marco de la aplicación cliente, sin acceso a funciones del servidor. Actualmente es ampliamente utilizado para enviar y recibir información del servidor junto con ayuda de otras tecnologías como AJAX o AngularJS, en nuestro caso se ha utilizado AngularJS. JavaScript se interpreta en el agente de usuario al mismo tiempo que las sentencias van descargándose junto con el código HTML.

3.2.4. Angular

AngularJS es un framework de aplicaciones web front-end de código abierto, principalmente mantenido por Google y por una comunidad de individuos y corporaciones para abordar muchos de los desafíos encontrados en el desarrollo de aplicaciones de una sola página.

Este framework adapta y amplía el HTML tradicional para servir mejor contenido dinámico a través de un data binding bidireccional que permite la sincronización automática de modelos y vistas. Como resultado, AngularJS pone menos énfasis en la manipulación del Document Object Model (DOM) y mejora la testeabilidad y el rendimiento.

Con el uso de la inyección de dependencias, Angular lleva servicios tradicionales del lado del servidor, tales como controladores dependientes de la vista, a las aplicaciones web del lado del cliente. En consecuencia, gran parte de la carga en el backend se reduce, lo que conlleva a aplicaciones web mucho más ligeras.

3.3. Otras tecnologías de desarrollo

3.3.1. Intercambio de datos – JSON

JSON (en inglés, *JavaScript Object Notation*) es un tipo de formato para la representación de datos siguiendo una sintaxis específica, cuya función fundamental consiste en que diferentes aplicaciones puedan intercambiar información independientemente de la plataforma o lenguaje en el que han sido desarrolladas.

Utilización en el proyecto

Tal y como puede observarse en la **Figura 4**, el equipo de desarrollo optó por JSON como estándar de transmisión de datos en cliente y servicio web. Antes de mí llegada a la empresa ya se trabajaba con JSON y esto es debido a tres factores importantes, que presento a continuación:

- Su análisis sintáctico (parsing, en inglés) y su generación son relativamente sencillas en JavaScript, lenguaje de programación en el lado del cliente que nosotros utilizamos.
- Si bien es cierto que la arquitectura de servicio web, que es REST, soporta también otros lenguajes de representación de datos como XML, Spring Framework en su última versión utiliza JSON como formato estándar de intercambio de datos en desarrollo de servicios web REST.

3.3.2. API Rest

El servicio web un servicio ofrecido por un dispositivo electrónico a otro dispositivo electrónico, se comunican entre sí a través de la **World Wide Web (WWW)**. En un servicio web, la tecnología Web como HTTP (protocolo de comunicación que permite las transferencias de información en la World Wide Web), diseñado originalmente para la comunicación de persona a máquina, se utiliza para la comunicación de máquina a máquina, más concretamente, para la transferencia de lectura mecánica formatos de archivo como XML y JSON.

En la práctica, el servicio web normalmente proporciona una interfaz basada en Web orientado a objetos a un servidor de base de datos, utilizado por ejemplo por otro servidor Web, o mediante una aplicación móvil, que proporciona una interfaz de usuario para el usuario final.

Podríamos considerar REST como un framework para construir aplicaciones web respetando HTTP. Por lo tanto REST es el tipo de arquitectura más natural y estándar para crear APIs para servicios orientados a Internet.

3.3.2.1. Características de REST

Existen tres niveles de calidad a la hora de aplicar REST en el desarrollo de una aplicación web y más concretamente una API:

- **Uso correcto de URIs**
- **Uso correcto de HTTP.**
- **Implementar Hypermedia.**

Además de estas tres reglas, debemos tener un **protocolo cliente/servidor sin estado** donde cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. Sin embargo, en la práctica, muchas aplicaciones basadas en HTTP utilizan cookies y otros mecanismos para mantener el estado de la sesión.

Uso correcto de URIs

Cuando desarrollamos una web o una aplicación web, las URLs nos permiten acceder a cada uno de las páginas, secciones o documentos del sitio web.

Cada página, información en una sección, archivo, cuando hablamos de REST, los nombramos como recursos. El recurso por lo tanto es la información a la que queremos acceder o que queremos modificar o borrar, independientemente de su formato.

Las **URL (Uniform Resource Locator)** son un tipo de **URI (Uniform Resource Identifier),** que además de permitir identificar de forma única el recurso, nos permite localizarlo para poder acceder a él o compartir su ubicación.

Existen varias reglas básicas para ponerle nombre a la URI de un recurso:

- **Los nombres de URI no deben implicar una acción, por lo tanto debe evitarse usar verbos en ellos.**
- **Deben ser únicas, no debemos tener más de una URI para identificar un mismo recurso.**

La URI **/hotel/2/editar** sería incorrecta ya que tenemos el verbo editar en la misma.

Para el recurso hotel con el identificador 2, la siguiente URI sería la correcta, independientemente de que vayamos a editarla, borrarla, consultarla o leer sólo uno de sus conceptos: **/hotel/2**

- **Deben ser independiente de formato.**

  Por ejemplo, la URI **/hotel/2.pdf** no sería una URI correcta, ya que estamos indicando la extensión pdf en la misma.

  Para el recurso factura con el identificador 2, la siguiente URI sería la correcta, independientemente de que vayamos a consultarla en formato pdf, txt, xml o json: **/hotel/2**

  Deben mantener una jerarquía lógica.

  Por ejemplo, la URI **/habitación/234/hotel/2** no sería una URI correcta, ya que no sigue una jerarquía lógica.

  Para el recurso hotel con el identificador 2 de la habitación 3, la siguiente URI sería la correcta: **/hotel/2/habitación/234**

- **Los filtrados de información de un recurso no se hacen en la URI.**

  Para filtrar, ordenar, paginar o buscar información en un recurso, debemos hacer una consulta sobre la URI, utilizando parámetros HTTP en lugar de incluirlos en la misma.

  Por ejemplo, la URI **/reservas/orden/desc/fecha-desde/2007/pagina/2** sería incorrecta ya que el recurso de listado de reservas sería el mismo pero utilizaríamos una URI distinta para filtrarlo, ordenarlo o paginarlo.

  La URI correcta en este caso sería:

  /reservas?fecha-desde=2007&orden=DESC&pagina=2

Uso correcto de HTTP

Conocer bien HTTP no es opcional para un desarrollador web al que le importe su trabajo. Para desarrollar APIs REST los aspectos claves que hay que dominar y tener claros son:

- **Métodos HTTP**

Como hemos visto en el anterior nivel, a la hora de crear URIs no debemos poner verbos que impliquen acción, aunque queramos manipular el recurso.

Para manipular los recursos, HTTP nos dota de los siguientes métodos con los cuales debemos operar:

- **GET**: para consultar y leer recursos
- **POST**: para crear recursos
- **PUT**: para editar recursos
- **DELETE**: para eliminar recursos.
- **PATCH**: para editar partes concretas de un recurso.

Por ejemplo para un recurso de hoteles.

GET /hotel: nos permite acceder al listado de hoteles.

POST /hotel: nos permite crear un hotel nuevo

GET /hotel/4: nos permite acceder al detalle de una factura

PUT /hotel/4: nos permite editar el hotel, sustituyendo la totalidad de la información anterior por la nueva.

DELETE /hotel/4: nos permite eliminar el hotel.

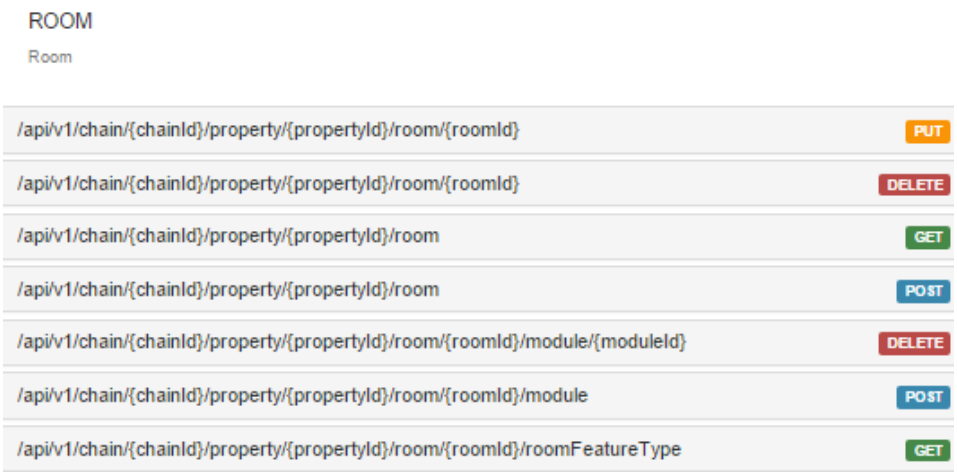


Figura 8: método creados para un room en la API

- **Códigos de estado**

Uno de los errores más frecuentes a la hora de construir una API suele ser el reinventar la rueda creando nuestras propias herramientas en lugar de utilizar las que ya han sido creadas, pensadas y testadas. La rueda más reinventada en el desarrollo de APIs son los códigos de error y códigos de estado.

Cuando realizamos una operación, es vitar saber si dicha operación se ha realizado con éxito o en caso contrario, por qué ha fallado.

- **Aceptación de tipos de contenido**

HTTP nos permite especificar en qué formato queremos recibir el recurso, pudiendo indicar varios en orden de preferencia, para ello utilizamos el header **Accept**.

Nuestra API devolverá el recurso en el primer formato disponible y, de no poder mostrar el recurso en ninguno de los formatos indicados por el cliente mediante el header Accept, devolverá el código de estado **HTTP 406**.

En la respuesta, se devolverá el header **Content-Type**, para que el cliente sepa qué formato se devuelve.

Figura 9: formato de una respuesta

Implementar Hypermedia

El concepto de Hypermedia y la finalidad que busca describir es bastante sencillo: **conectar mediante vínculos las aplicaciones clientes con las APIs**, permitiendo a dichos clientes despreocuparse por conocer de antemano del cómo acceder a los recursos.

Con Hypermedia básicamente añadimos información extra al recurso sobre su conexión a otros recursos relacionados con él.

Sin embargo, necesitamos que el cliente que accede a nuestra API entienda que esa información no es propia del recurso, sino que es información añadida que puede utilizar para enlazar el hotel con la habitación por ejemplo.

Para ello conseguir esto, debemos utilizar las cabeceras **Accept** y **Content-Type**, para que tanto el cliente como la API, sepan que están hablando **hypermedia**.

El servicio web por lo tanto, como implementa hypermedia, le devuelve la información de recurso y la información de hypermedia que puede utilizar el cliente.

Hypermedia es útil por ejemplo para que el cliente no tenga que conocer las URLs de los recursos, evitando tener que hacer mantenimientos en cada uno de los mismos si en un futuro dichas URLs cambian (cosa que no debería pasar).

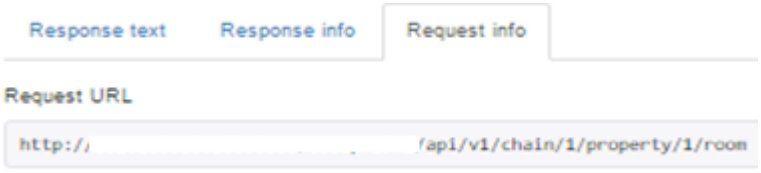


Figura 10: url de petición que muestra la respuesta

3.3.2.2. Ventajas que ofrece REST para el desarrollo

1. **Separación entre el cliente y el servidor**

El protocolo REST separa totalmente la interfaz de usuario del servidor y el almacenamiento de datos. Eso tiene algunas ventajas cuando se hacen desarrollos. Por ejemplo, mejora la portabilidad de la interfaz a otro tipo de plataformas, aumenta la escalabilidad de los proyectos y permite que los distintos componentes de los desarrollos se puedan evolucionar de forma independiente.

2. **Visibilidad, fiabilidad y escalabilidad**

La separación entre cliente y servidor tiene una ventaja evidente y es que cualquier equipo de desarrollo puede escalar el producto sin excesivos problemas. Se puede migrar a otros servidores o realizar todo tipo de cambios en la base de datos, siempre y cuando los datos de cada una de las peticiones se envíen de forma correcta.

3. **La API REST siempre es independiente del tipo de plataformas o lenguajes**

La API REST siempre se adapta al tipo de sintaxis o plataformas con las que se estén trabajando, lo que ofrece una gran libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo. Lo único que es indispensable es que las respuestas a las peticiones se hagan siempre en el lenguaje de intercambio de información usado, normalmente XML o JSON.

En el proyecto se generará la documentación de la API de forma automática utilizando la librería **JSONDoc**. Escribir una buena API a menudo no es suficiente para los clientes que necesitarán usarlo. Los usuarios de API necesitan una forma más legible para entender el significado de una API y cómo usarla.

```
# DOCUMENTATION
jsondoc.version=0.136.2-DEV
jsondoc.basePath=http://                    /tesiproPMS
jsondoc.packages[0]=com.tesipro.pms.api.controller
jsondoc.packages[1]=com.tesipro.pms.core.persistence.domain
jsondoc.packages[2]=com.tesipro.pms.api.model
```

Figura 11: configuración del JSONDoc en el archivo application.properties

3.3.3. Git

Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente.

Cada vez que se confirma un cambio, o guardas el estado de tu proyecto en Git, él básicamente hace una foto del aspecto de todos tus archivos en ese momento, y guarda una referencia a esa instantánea. Para ser eficiente, si los archivos no se han modificado, Git no almacena el archivo de nuevo, sólo un enlace al archivo anterior idéntico que ya tiene almacenado.

Git tiene tres estados principales en los que se pueden encontrar tus archivos: **confirmado** (committed), **modificado** (modified) y **preparado** (staged).

- Confirmado significa que los datos están almacenados de manera segura en tu base de datos local.
- Modificado significa que se ha modificado el archivo pero todavía no lo has confirmado a tu base de datos.
- Preparado significa que se ha marcado un archivo modificado en su versión actual para que vaya en tu próxima confirmación.

Esto nos lleva a las tres secciones principales de un proyecto de Git: el **directorio de Git** (Git directory), el **directorio de trabajo** (working directory), y el **área de preparación** (staging area).

El **directorio de Git** es donde Git almacena los metadatos y la base de datos de objetos para el proyecto. Es la parte más importante de Git y es lo que se copia cuando se clona un repositorio desde otro ordenador.

El **directorio de trabajo** es una copia de una versión del proyecto. Estos archivos se sacan de la base de datos comprimida en el directorio de Git y se colocan en disco para que se pueda usar o modificar.

El **área de preparación** es un sencillo archivo, generalmente contenido en tu directorio de Git, que almacena información acerca de lo que va a ir en la próxima confirmación.

El flujo de trabajo básico en Git es algo así:

1. Se modifica una serie de archivos en el directorio de trabajo.
2. Se prepara los archivos, añadiéndolos al área de preparación.
3. Se confirma los cambios, lo que toma los archivos tal y como están en el área de preparación y almacena esas instantáneas de manera permanente en el directorio de Git.

3.3.4. Bower

Los sitios web están hechos de muchas cosas: frameworks, bibliotecas, recursos y utilidades. Bower administra todas estas cosas para nosotros.

Utilización en el proyecto

Bower puede administrar componentes que contienen HTML, CSS, JavaScript, fuentes o incluso archivos de imagen. No concatena ni limita el código ni hace nada más, simplemente instala las versiones correctas de los paquetes que necesitas y sus dependencias.

Bower trabaja recolectando e instalando paquetes de todo, cuidando de cazar, encontrar, descargar y guardar las cosas que buscas. Bower proporciona ganchos para facilitar el uso de paquetes en nuestras herramientas y flujos de trabajo.

3.3.5. Node.js

Node es un intérprete Javascript del lado del servidor que cambia la noción de cómo debería trabajar un servidor. Su meta es permitir a un programador construir aplicaciones altamente escalables y

escribir código que maneje decenas de miles de conexiones simultáneas en una sólo una máquina física.

Utilización en el proyecto

Se añadirá el módulo Bower que nos permitirá realizar una serie de operaciones con objetos y utilizará npm (Node Package Manager), el gestor de paquetes de nodos. En este proyecto se utilizará la herramienta NPM para gestionar los paquetes Bower instalados.

4. Herramientas de desarrollo

Las herramientas por sí solas no resuelven problemas que no sabemos hacer, también es cierto que si se conoce la tarea a realizar y qué herramienta debemos utilizar, la cantidad de tiempo dedicada a dicha tarea es menor, aumentando nuestra productividad, efectividad y reduciendo el tiempo de desarrollo. Por todo esto, pienso que el hecho de escoger unas buenas herramientas de desarrollo software es fundamental para llevar a cabo un proyecto de este calibre.

Una vez explicadas las tecnologías más importantes utilizadas a lo largo del desarrollo, procederé a exponer las herramientas que he utilizado para llevar a cabo el desarrollo del PMS web.

Introduciré las herramientas de desarrollo que utilicé para desarrollar tanto la parte de backend como la de frontend ya sea para programar como para la manipulación de la base de datos o la visualización de diagramas de clases.

4.1. Intellij IDEA

Es un ambiente de desarrollo integrado (IDE) para el desarrollo de programas informáticos. Entre unos de los lenguajes que soporta esta Java o SQL, que serán los lenguajes que utilicé para programar en la parte de backend y Javascript, HTML o CSS que fueron los que utilicé en la parte del frontend.

4.2. Visual Paradigm for UML

En proyectos Java, donde debemos aplicar intensivamente los conceptos avanzados de orientación a objeto, es vital realizar un buen diseño del sistema. Para realizar un buen diseño es necesario abstraerse de los problemas particulares y tratar de modelar comportamientos. Para poder realizar esta abstracción podemos ayudarnos con diagramas UML.

Visual paradigm es una herramienta UML (Unified Modified Language en inglés) que nos servirá para el modelado de la aplicación.

4.3. Microsoft SQL Server Management Studio

SQL Server Management Studio (SSMS) es un entorno integrado para la gestión de cualquier infraestructura SQL, desde SQL Server a SQL Database. SSMS proporciona herramientas para

configurar, supervisar y administrar instancias de SQL desde dondequiera que lo despliegue. SSMS proporciona herramientas para implementar, supervisar y actualizar los componentes de nivel de datos, como bases de datos y almacenes de datos utilizados por las aplicaciones, y crear consultas (queries) y secuencias de comandos (scripts).

4.4. Source Tree

SourceTree es quizás uno de los mejores clientes GUI para manejar repositorios git y mercurial que existen en la actualidad.

Entre otras muchas cosas Source Tree nos permite:

- Crear y clonar repositorios de cualquier sitio, tanto Git como Mercurial. Además de integrarse perfectamente con Bitbucket o Github.
- Commit, push, pull y merge de nuestros archivos
- Detectar y resolver conflictos
- Consultar el historial de cambios de nuestros repositorios.

4.5. JIRA

JIRA Software es una aplicación de gestión de proyectos que soporta cualquier metodología ágil, ya sea scrum, kanban, etc. Puede planificar, controlar y administrar todos sus proyectos de desarrollo de software ágil desde una sola herramienta.

En nuestro caso, como metodología agile hemos seguido scrum, una metodología ágile donde los productos se construyen en una serie de iteraciones de longitud fija. Hay cuatro pilares que aportan estructura a este framework: planificación de sprint, stand ups (también llamados scrums diarios), sprints y retrospectivas. Fuera de la caja, JIRA Software viene con un conjunto completo de herramientas ágiles que ayudan a nuestro equipo scrum a realizar estos eventos con facilidad.

4.6. Bitbucket

Bitbucket es un servicio de alojamiento basado en web, para los proyectos que utilizan el sistema de control de revisiones Mercurial y Git. Bitbucket ofrece planes comerciales y gratuitos. Se ofrece cuentas gratuitas con un número ilimitado de repositorios privados (que puede tener hasta cinco usuarios en el caso de cuentas gratuitas) desde septiembre de 2010. Está escrito en Python usando Django como framework.

4.7. XAMPP

XAMPP es un paquete de instalación independiente de plataforma, software libre, que consiste principalmente en el sistema de gestión de bases de datos MySQL, el servidor web Apache y los intérpretes para lenguajes de script: PHP y Perl.

En nuestro caso solo necesitaremos el servidor Apache. En la parte del frontend se necesitará instalar este servidor de páginas web. Una vez descargado el XAMPP, habrá que configurarlo acorde a nuestras necesidades.

5. Metodologías de trabajo

Durante el desarrollo se han establecido ciertas metodologías de trabajo ya sea para aumentar la productividad o para que el trabajo se haga de forma organizada y lo que es más importante que el grupo de desarrolladores este acoplado y puedan trabajar de manera eficiente. A continuación explicaré algunas de las metodologías que he utilizado al largo del desarrollo de la aplicación.

5.1. Integración continúa

La integración continua es una práctica de desarrollo software donde los miembros del equipo integran su trabajo frecuentemente (como mínimo una vez al día, aunque normalmente se realizan múltiples integraciones diarias).

Cada integración se verifica compilando el código fuente y obteniendo un ejecutable (a esto se le llama **build**, y debe hacerse de forma automatizada). Además también se pasan las pruebas y métricas de calidad para detectar los errores tan pronto como sea posible.

Es muy recomendable hacer builds periódicamente y comprobar que funcionen correctamente, para conseguir un producto final más fiable, con menos fallos en producción (**develop**). La integración continua a menudo reduce la fragmentación de los esfuerzos de los desarrolladores por falta de comunicación sobre lo que puede ser reutilizado o compartido**.**

Utilización en el proyecto

Se utiliza la integración continua porque mejora la calidad del software ya sea el proceso, el producto y el equipo.

5.2. Agile-Scrum

El desarrollo de software ágil se refiere a un grupo de metodologías de desarrollo de software basadas en el desarrollo iterativo, donde los requisitos y las soluciones evolucionan a través de la colaboración entre equipos multifuncionales auto-organizados.

Scrum es un subconjunto de Agile. Es un framework de proceso ligero para el desarrollo ágil, y el más ampliamente utilizado.

Scrum se utiliza con mayor frecuencia para administrar software complejo y desarrollo de productos, utilizando prácticas repetitivas e incrementales. Scrum aumenta significativamente la productividad y reduce el tiempo a los beneficios en relación con los clásicos “cascadas

Un proceso ágil Scrum beneficia a la organización ayudándola a:

- Aumentar la calidad de los entregables
- Lidiar con los cambios (y esperar los cambios)
- Proporcionar mejores estimaciones mientras gasta menos tiempo creándolas
- Tener más control sobre el cronograma del proyecto

Utilización en el proyecto

Se usa esta metodología puesto que ayuda a tener un seguimiento de las tareas que tiene cada desarrollador y controlar posibles bugs que tenga la aplicación entre otras funciones.

5.3. Gitflow

GitFlow es un modelo de ramificación para Git, creado por Vincent Driessen. Ha atraído mucha atención porque está muy bien adaptado a la colaboración y la ampliación del equipo de desarrollo**.**

Cada vez que queramos hacer algo en el código, tendremos que crear la rama que corresponda, trabajar en el código, incorporar el código donde corresponda y cerrar la rama.

A lo largo de nuestra jornada de trabajo necesitaremos ejecutar varias veces al día los comandos git, merge, push y pull así como hacer checkouts de diferentes ramas, borrarlas, etc. Git-flow son un conjunto de extensiones que nos ahorran bastante trabajo a la hora de ejecutar todos estos comandos, simplificando la gestión de las ramas de nuestro repositorio.

Utilización en el proyecto

Se utiliza git-flow para ver el flujo de trabajo de cada desarrollador de manera limpia y clara en la historia del repositorio, cuando se inició y cuando se finalizó el desarrollo de cada una de las features branches.

6. Desarrollo del BackEnd

Una vez explicadas las tecnologías usadas, así como los herramientas de desarrollo que utilizamos a lo largo del proyecto, además de una breve explicación de estos, me gustaría dejar claro cómo estaba montado el proyecto cuando me incorporé al grupo de desarrollares de la empresa. Sin embargo quisiera recalcar que yo no participé en el montaje de la estructura de este proyecto, puesto que cuando yo me incorporé, llevaban ya meses desarrollando la aplicación. A continuación explicaré en la **sección 6.1** la arquitectura de las clases, es decir, que es lo que se hace en las carpetas que uso (core y API), las anotaciones proporcionadas por los frameworks, etc. Y partir de la **sección 6.2** empiezo a detallar los mantenimientos que he ampliado en la aplicación.

6.1. Arquitectura de clases

Clasificación de las clases según su uso:

Carpeta Configuration

- Clase de configuración de Spring, Hibernate y Security

Carpeta Core

- **Entity**
  - Se definen las diferentes clases correspondientes a las tablas de la base de datos para poder trabajar con ellas, utilizamos las anotaciones de JPA2.
- **Repository**
  - Se definen las queries para acceder a los datos. Todas las consultas a la base de datos estarán en esta parte.

- **Service**
  - Se definen las reglas de negocio o procesos, es decir, se ejecutan los procesos utilizando las queries del repository.

Carpeta API

- **Controller**
  - Se definen las diferentes rutas de acceso a los datos basados en la definición API RESTFUL, utiliza los servicios para acceder a los datos.
- **Interceptor**
  - Clases para interceptar las peticiones a la API.

Como dije anteriormente se utilizan diversas tecnologías entre ellas Hibernate (framework que facilita el mapeo entre atributos de una base de datos relacional tradicional y el modelo de objetos de una aplicación).

A medida que se van creando las tablas en la base de datos, se podía empezar a crear las entidades en el proyecto Java.

Principalmente, hice uso de dos carpetas:

- **API**: se encuentra el controlador, que accederá a los datos mediante la capa de servicio.
- **Core**: esta la capa de servicio, el dominio y los repositorios donde se harán las queries.

En este proyecto no se sigue un solo patrón de diseño, sino varios. Se utiliza el patrón **diseño guiado por el dominio** (DDD, domain-driven design en inglés) un enfoque para el desarrollo de software con necesidades complejas mediante una profunda conexión entre la implementación y los conceptos del modelo y núcleo del negocio. Además se utiliza el **patrón fachada** (design pattern facade, en inglés), un patrón que se utiliza a menudo cuando un sistema es complejo o difícil de entender porque el sistema tiene un gran número de clases interdependientes.

6.1.1. Carpeta Core

En esta carpeta están situadas las carpetas de dominio, servicio y repositorio.

- **Dominio**

Una vez tenemos la base de datos, en el dominio se creara la entidad correspondiente a aquella tabla de la base de datos, mediante Hibernate ya no tendremos tablas sino objetos. En el dominio, se crean las entidades con sus atributos, sus respectivos getters, setters y relaciones con otras entidades. Se utilizaran anotaciones de JPA2 como **@Entity** para crear una entidad a partir de un POJO (Plain Old Java Object), es decir, una clase simple independiente de cualquier framework en especial.

Una entidad es un objeto de dominio de persistencia. Normalmente, una entidad representa una tabla en una base de datos relacional y cada instancia de entidad corresponde a una fila de esa tabla.

También hay otras notaciones para relacionas entidades entre ellas, por ejemplo, **@ManyToOne** en la que muchas instancias de una entidad puede pertenecer a otra, en el caso del PMS web, la entidad muchas instancias de Chain Channel pertenecen a una Chain (cadena de hotel).

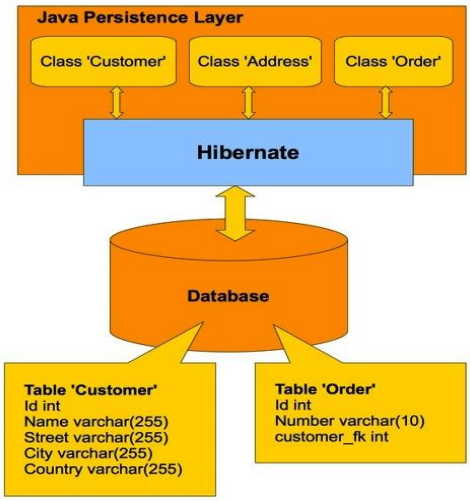


Figura 12: esquema de mapeo de la base de datos con Hibernate

- **Servicio**

Una vez tenemos, las entidades, podemos empezar a implementar los métodos que necesitemos en la carpeta controlador. A medida que necesitamos métodos como listar o guardar, declararemos estos métodos en la carpeta servicio y desde esta se llamará al repositorio donde tendremos todas las queries.

En la carpeta servicio, tendremos que para cada mantenimiento que se requiera tendremos una interface y para cada interface tendremos otra clase implementado a esta. En la interface tendremos todos los métodos que serán llamados desde el controlador. Básicamente, habrá métodos para contar elementos, listar elementos, guardar, editar y eliminar, es decir los que se utilizan para implementar el CRUD del mantenimiento correspondiente. Una vez tenemos los métodos en la interface, se implementarán estos métodos llamando a la capa repositorio.

La clase que implementa los métodos de la interface utiliza la anotación de Spring **@Service**, puesto que es una clase servicio y donde se encontrará toda la lógica de negocios. Otra anotación Spring que se usa es **@Transactional**. Se tiene en cuenta dos aspectos:

- El contexto de persistencia
- La transacción de base de datos

La anotación transaccional define el ámbito de una única transacción de base de datos. La transacción de base de datos ocurre dentro del ámbito de un contexto de persistencia.

El contexto de persistencia está en el JPA EntityManager, implementado internamente usando una Hibernate (cuando se utiliza Hibernate como proveedor de persistencia).

El contexto de persistencia es sólo un objeto sincronizador que rastrea el estado de un conjunto limitado de objetos Java y se asegura de que los cambios en esos objetos se persisten de nuevo en la base de datos.

Además, desde la clase donde se implementan los métodos de la clase interface, se llamará a la capa repositorio. Se utilizará otra anotación Spring **@Autowired**, esta anotación le dice a Spring dónde debe ocurrir una inyección. En este caso se inyectará una instancia del repositorio de la entidad con la que estemos trabajando.

- **Repositorio**

En la carpeta repositorio, al igual que en la carpeta servicio, tendremos que para cada mantenimiento que se requiera tendremos una interface y para cada interface tendremos otra clase implementado a esta. En la interface tendremos todos los métodos que serán llamados desde la capa servicio.

En la clase que implementará todos los métodos de la interface tendremos todas las queries que necesitaremos para acceder a los datos de la base de datos.

Se utilizará la anotación **@Repository**, para determinar que la clase es un repositorio, definida por el Domain-Driver design como un mecanismo para encapsular el almacenamiento, la recuperación y el comportamiento de búsqueda que emula una colección de objetos. Básicamente es una anotación que marca la clase específica como un objeto de acceso a datos.

Además se utiliza la anotación **@PersistenceContext** para tener un **EntityManager**. Las entidades son gestionadas por el gestor de entidades, que está representado por las instancias **javax.persistence.EntityManager**. Cada instancia de EntityManager está asociada con un contexto de persistencia: un conjunto de instancias de entidades administradas que existen en un almacén de datos determinado. Un contexto de persistencia define el ámbito bajo el cual las instancias de entidades particulares se crean, persisten y se eliminan. La interfaz EntityManager define los métodos que se utilizan para interactuar con el contexto de persistencia.

Una entidad puede ser creada, editada o eliminada vía un objeto EntityManager. El EntityManager en sí mismo es creado por el contenedor utilizando la información en persistence.xml, por lo que

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence" version="2.0">
    <persistence-unit name="PMS_V02"/>
    <persistence-unit name="PMS_V01"/>
</persistence>
```

para usarlo en tiempo de ejecución, simplemente necesitamos solicitar que se inyecte en uno de nuestros componentes. Lo hacemos a través de **@PersistenceContext**.

Figura 13: configuración del archivo persistence.xml para la persistencia de datos con la BBDD

Otro factor importante a tener en cuenta en el desarrollo del backend es sin duda el papel de la JPA (Java Persistance API).

La Java Persistence API proporciona a los desarrolladores de Java un recurso de mapeo objeto/relacional para administrar datos relacionales en aplicaciones Java. La persistencia de Java consta de cuatro áreas:

- La API de persistencia de Java
- El lenguaje de consulta
- La API de criterios de persistencia de Java
- Metadatos de mapeo objeto/relacional

La API Java Persistence nos proporciona método para consultar entidades. El lenguaje de consulta Java Persistence (JPQL) es un lenguaje simple similar al SQL utilizado para consultar entidades y sus relaciones.

```
public int countByChainIdAllProperties(int chainId, String name) {
    String stringQuery = "SELECT count(distinct crt.id) " +
            "FROM ChainRateType crt " +
            "WHERE crt.chain.id = :chainId ";

    // Optional parameters
    if(Util.isNotNullAndNotEmpty(name))                    stringQuery += " AND crt.name LIKE :name ";

    Query query = entityManager.createQuery(stringQuery, Long.class)
            .setParameter("chainId",chainId);

    // Optional parameters
    if(Util.isNotNullAndNotEmpty(name))                    query.setParameter("name",name + "%");

    int chainRateTypeCount = ((Long) query.getSingleResult()).intValue();
    return chainRateTypeCount;
}
```

Figura 14: consulta a base de datos hecha en JPQL

El **EntityManager.createQuery** se utiliza para consultar el almacén de datos mediante consultas de JPQL. El método **createQuery** se utiliza para crear consultas dinámicas, que son consultas definidas directamente dentro de la lógica empresarial de una aplicación:

6.1.2. Carpeta API

En esta carpeta se encuentran los controladores, interceptores y otros subdirectorios los cuales yo no participe en su implementación. Principalmente, desarrollé los algunos controladores en donde se definen las diferentes rutas de acceso a los datos basados en la definición API RESTFUL y se utiliza los servicios para acceder a los datos.

Tesipro está pensado para trabajar tanto en hoteles independientes como en cadenas hoteleras. Tanto unos como otros tienen sus peculiaridades así que el equipo diseñó una BBDD pensando en el máximo número de casuísticas posibles. En términos generales, una cadena hotelera está compuesta por hoteles, y su configuración la realiza básicamente el usuario. Sin embargo, los datos de las cadenas son un conjunto de información introducida por el usuario junto a información heredada por el sistema.

En el controlador usaremos librerías de JSON para obtener anotaciones como **@ApiAuthBasic** para especificar que se requiere acceso a la autenticación básica. También debe especificar roles que pueden acceder al método y una lista de usuarios de prueba **@ApiAuthBasicUser.**

Además se usa la anotación **@Api**. Esta anotación debe utilizarse en la clase del controlador en un patrón MVC y describe en pocas palabras lo que hace el API.

Se utiliza también la anotación **@RestController**. Todos los componentes de Spring MVC deben utilizar la anotación **@Controller** común para marcarlo como servlet del controlador. Cuando se implementa servicios web RESTful, la respuesta siempre se envía con el cuerpo de respuesta. Para hacer esto simple, Spring ha proporcionado una versión especializada del controlador. Es un caso de uso muy común que los controladores implementen una API REST, por lo tanto, solo sirven contenido JSON, XML o Custom MediaType. Para mayor comodidad, en lugar de anotar todos sus métodos **@RequestMapping** con **@ResponseBody**, puede anotar su clase Controller con **@RestController**. Entonces **@RequestMapping** se utiliza para definir urls de peticiones web entrantes para los niveles de clases y métodos. Con la anotación **@RequestMapping**, sólo podemos definir URI genéricas. Para las asignaciones dinámicas de URI en caso de pasar variables junto con el URI, se utiliza **@PathVariable**. Proporciona un estilo consistente entre los entornos Servlet y Portlet, con la semántica adaptándose al entorno concreto.

En cada una de las clases controladores se accederán mediante el servicio del mantenimiento en el que estemos trabajando. Normalmente, se implementa el CRUD básico que consta de:

- Un método GET para obtener una lista.
- Un método GET por ID para obtener información de un elemento de esa lista con la información que se requiera.
- Un método POST para crear un elemento, por ejemplo, crear una habitación o una oferta.
- Un método PUT para editar información de un elemento de la lista.
- Un método DELETE para eliminar algún elemento de esa lista.

Como he mencionado antes hay mantenimientos que dependiendo de la casuística requieren métodos adicionales debido a sus relaciones con otras entidades y los requerimientos del product owner.

6.2. Gestión de Allotments (cupos en español)

La distribución de habitaciones a través de los diferentes canales de venta es una decisión crítica para cualquier responsable de ventas. Así pues, el objetivo de este desarrollo es la de controlar el número de habitaciones que se quiere vender en cada uno de los canales. Para ello será necesario

gestionar mediante calendarios diarios el número máximo de habitaciones a la venta, a que agencias están asignadas y en que IDS (Internet Distribuition System) se permite la venta.

Antes de comenzar la programación en la parte del backend, mis compañeros se encargaron de hacer el diagrama de clases de Allotment para ver las relaciones que tiene esta entidad, así como la base de datos con datos falsos para poder testear una vez acabado el CRUD.

Se creará la entidad allotment con sus respectivos atributos, seguidamente se implementarán las capas de servicio y repositorio, en esta última se harán las consultas en JPQL a la base de datos.

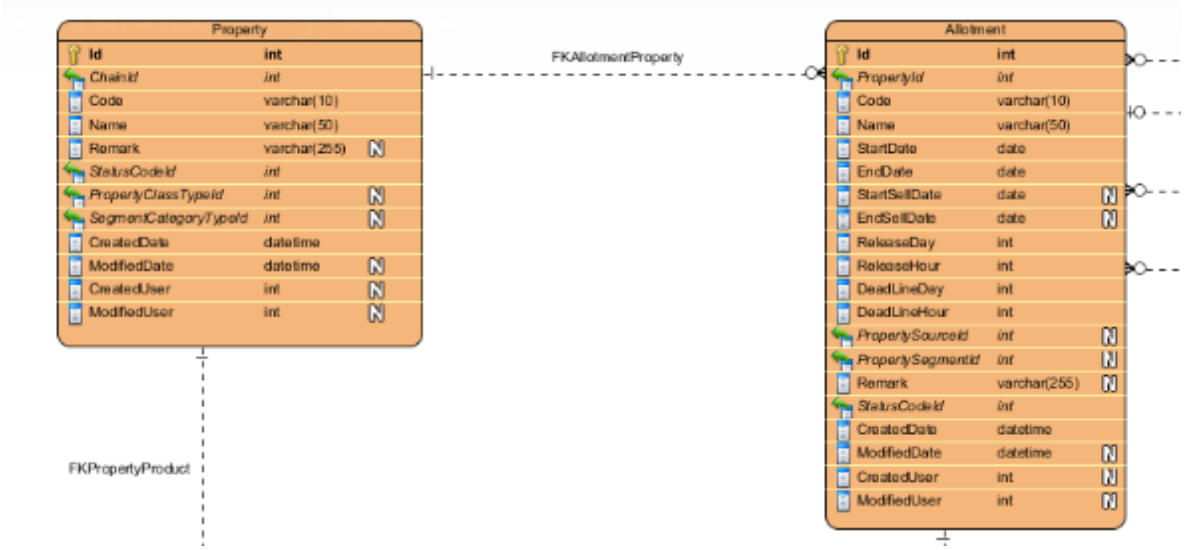


Figura 15: diagrama de clases y relación entre Allotment y Property

Como podemos ver en el diagrama de clases, Allotment es una entidad de property (hotel), en la que podemos ver sus atributos y las relaciones de estas. Este diagrama nos será muy útil a la hora de crear la entidad en el dominio.

Una vez tenemos el dominio, se creará la capa de servicio que a su vez llamará a la de repositorio donde estarán todas las consultas a la BBDD. Una vez tenemos el repositorio creado con las respectivas consultas, desde el controlador se accederán a esos datos mediante la capa de servicio

Se sigue un orden en el cual, se crean siempre las listas, luego el GET por ID, crear, editar y eliminar.

En el caso de allotment, tendremos dos métodos GET, uno para ver la lista de allotments y otro para obtener un allotment especificado mediante la id que se le pasa por parámetro. Allotment tendrá además un método POST para crear un cupo, un método PUT para editar este cupo y otro método DELETE para eliminarlo, a estos dos últimos métodos se les pasará la ID del cupo por parámetro.

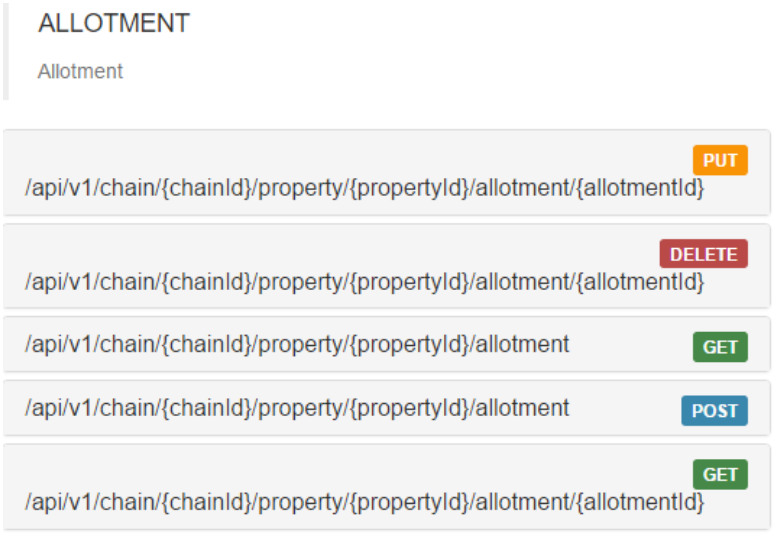


Figura 16: métodos de Allotment en la API

6.2.1. Allotment list

Antes de empezar con el método para listar los allotments se definió la ruta común para esta sección, esto se hizo mediante el **@RequestMapping.**

Se creó el método list que es de tipo Page, donde Page es una clase implementada por mis compañeros en la cual tiene un **offset**, un **limit**, un **count** y una **lista.** Esto es necesario para que el usuario tenga parámetros opcionales y pueda decidir cosas como la medida de su lista, el número de elementos que quiere que esta lista tenga, etc.

Para acceder a la información, el usuario tendrá que entrar obligatoriamente la ID de la cadena y la ID de la propiedad (**Path parameters**). Además tendrá otros parámetros opcionales en los que el usuario podrá obtener la información a su gusto, por ejemplo, una ordenación de la lista, el límite de elementos, un filtro por nombre o código de estado en el caso de allotment, es decir, una lista personalizada dependiendo de los parámetros que le entre el usuario (**Query parameters**).

Por defecto, el offset será 0 y el limit será 10, es decir queremos que la lista nos muestre todos los allotments desde el primero hasta un maximó de 10 allotments. El count nos dirá el número de allotments que tiene esa propiedad.

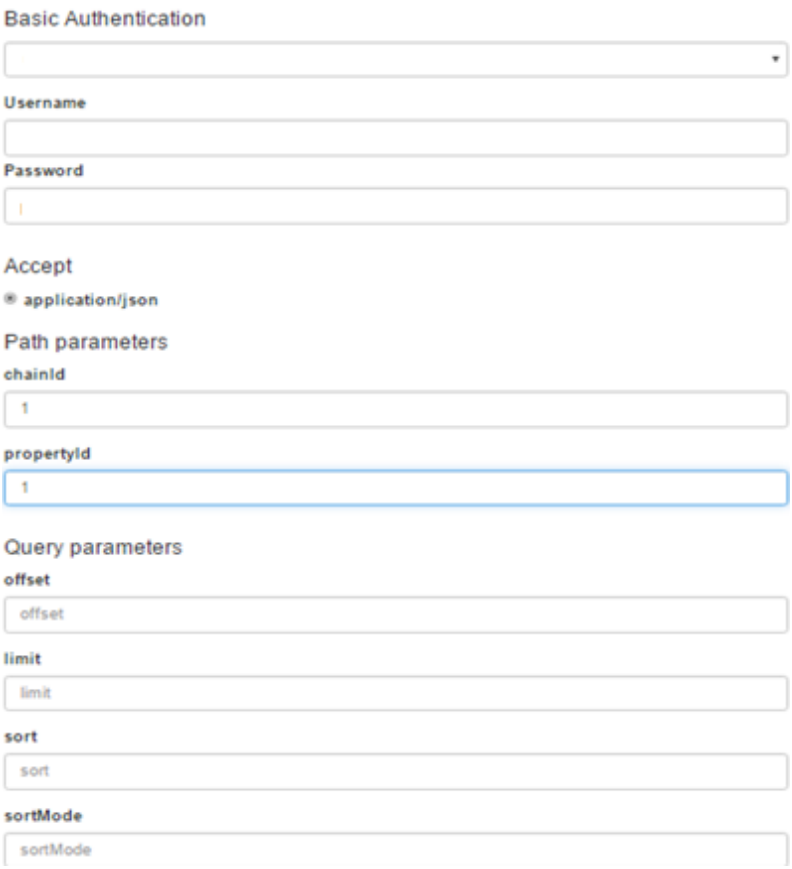


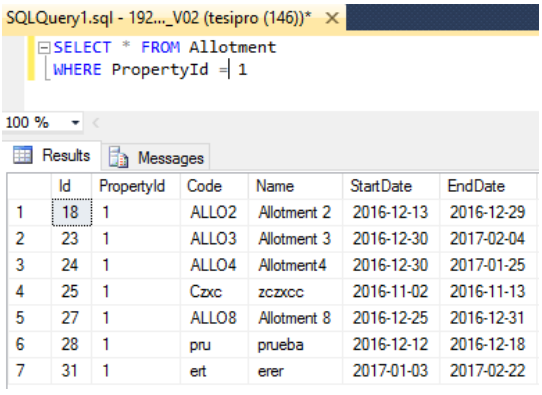
Figura 17: parámetros de consulta

En el método de listar allotments y por lo general en todos los mantenimientos se llamarán a los métodos de los servicios, en este caso al servicio de allotment. Métodos como **countByAllProperties** y **findByAllProperties** que a su vez desde el servicio se llamarán al repositorio donde obtendremos el resultado mediante las queries que se hagan. Un ejemplo del resultado de la lista de alloments entrando la ID de la cadena igual a 1 y la ID de la propiedad a 1 también.

```
{
  "count": 7,
  "offset": 0,
  "limit": 10,
  "list": [
    {
      "id": 18,
      "code": "ALLO2",
      "name": "Allotment 2",
      "startDate": "2016-12-13",
      "endDate": "2016-12-29",
      "startSellDate": "2016-12-01",
      "endSellDate": "2016-12-01",
      "releaseDay": 2,
      "releaseHour": 2,
      "deadLineDay": 2,
      "deadLineHour": 2,
      "createdDate": "2016-11-30T12:20:11.200Z",
      "modifiedDate": "2016-12-19T15:33:34.107Z",
      "modifiedUser": 1,
      "property": null,
      "statusCode": {
        "id": 4,
        "code": "4",
        "name": "Deflagged",
        "colorCode": "#795548",
        "createdDate": "2016-03-22T11:51:36.083Z"
      },
      "propertySource": null,
      "propertySegment": null,
      "allotmentProfileSet": null,
      "allotmentProfileExcludedSet": null,
      "allotmentChainProfileGroupSet": null,
      "reservationRoomStayDailySet": null
    },
    {
      "id": 23,
      "code": "ALLO3",
      "name": "Allotment 3",
      "startDate": "2016-12-30",
      "endDate": "2017-02-04",
      "startSellDate": "2016-11-30",
      "endSellDate": "2016-11-30",
```


Figura 18: respuesta al solicitar lista de allotments

Vemos que para la cadena 1 y la propiedad 1 de esta cadena tenemos 7 allotments. Cada allotments tendrá la información “fake” que nos llegará de la base de datos (en ese momento, después con el método POST o PUT podremos añadir o editar la lista directamente desde la API).



	Id	PropertyId	Code	Name	StartDate	EndDate
1	18	1	ALLO2	Allotment 2	2016-12-13	2016-12-29
2	23	1	ALLO3	Allotment 3	2016-12-30	2017-02-04
3	24	1	ALLO4	Allotment4	2016-12-30	2017-01-25
4	25	1	Czxc	zczxcc	2016-11-02	2016-11-13
5	27	1	ALLO8	Allotment 8	2016-12-25	2016-12-31
6	28	1	pru	prueba	2016-12-12	2016-12-18
7	31	1	ert	erer	2017-01-03	2017-02-22

Figura 19: consulta en la base de datos

6.2.2. Get Allotment By ID

En este método GET de tipo Allotment, tendremos que añadir la ID del allotment (**/allotmentId**) a la ruta común que configuramos al principio. Además tendremos que añadir la ID del allotment a la queremos acceder a los parámetros obligatorios, a parte de la ID de la cadena y la ID de la propiedad.

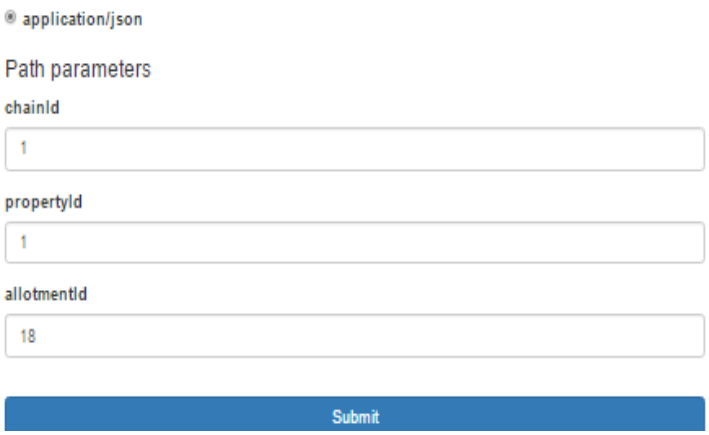


Figura 20: parámetros de petición

Figura 21: respuesta de la petición

Vemos la información del allotment con ID 18 que nos devuelve la API. Debido a la extensa información del JSON no he podido encontrar una imagen acorde a la información entera del allotment, pero sin duda el JSON generado es bastante grande. Esto es debido a que allotment está relacionado con otras tablas de las cuales queremos esa información, además de los atributos propios que tiene, es decir, un allotment está relacionado con un source, un segment, perfiles de usuarios, etc. Para acceder a esa información, en la query que se hace en el repositorio, utilizamos la cláusula **JOIN FETCH** para traer aquellos objetos relacionados con allotment reduciendo el número de consultas que se hacen contra la base de datos.

6.2.3. Save Allotment

En el método POST, para crear un allotment, tendremos que indiciarle los parámetros obligatorios como el chainId o el propertyId en donde queremos crear dicho allotment.

Tendremos que crear el requestBody donde mediante un formato JSON indicaremos el código, el nombre del allotment, la fecha de inicio y la fecha final como mínimo ya que son campos obligatorios para poder crear un allotment.

Estos campos obligatorios me los indica la base de datos diciéndome que aquellos que sean not null, se deberán especificar en el body object cuando estemos creando el allotment. Cuando se cree el allotment, automáticamente se generará la ID para ese allotment que se ha creado.

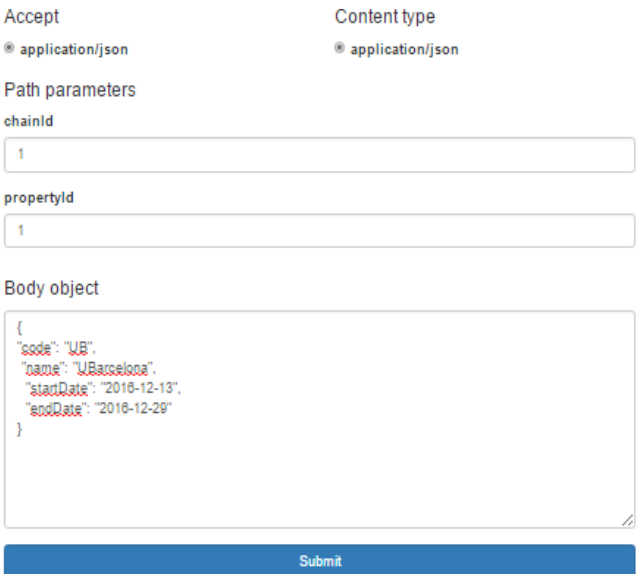


Figura 22: creación de Allotment en la Api

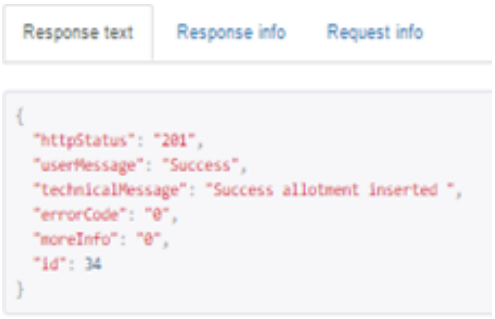


Figura 23: respuesta a la creación de Allotment

Como podemos ver en las imágenes, para la cadena con id 1 y la propiedad 1, se ha creado un allotment, introduciéndole en un formato JSON, el código, el nombre, la fecha de inicio y la fecha

final. Además vemos que mediante la clase ResponApiMessage, la Api nos devuelve un mensaje donde se especifica la id del nuevo allotment creado, además de un mensaje de éxito en la creación.

Para la creación de un nuevo allotment, así como cualquiera otra entidad, se utilizará el método persist que nos proporciona EntityManager y así podremos guardar cada instancia que se cree de esa entidad en la base de datos.

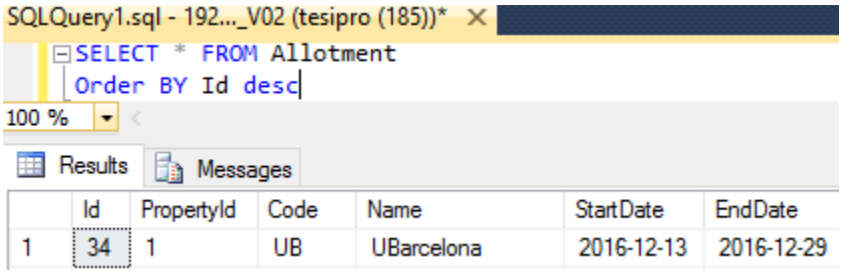


Figura 24: consulta en la base de datos con Micosoft SQL Server

6.2.4. Update Allotment

En este método se podrá editar un objeto de tipo allotment. Para ello se tendrá que indicar a que cadena, a que propiedad de la cadena pertenece y la ID del allotment que se quiere editar. Es decir, se tendrá que introducir estos tres parámetros obligatoriamente para poder editar el allotment.

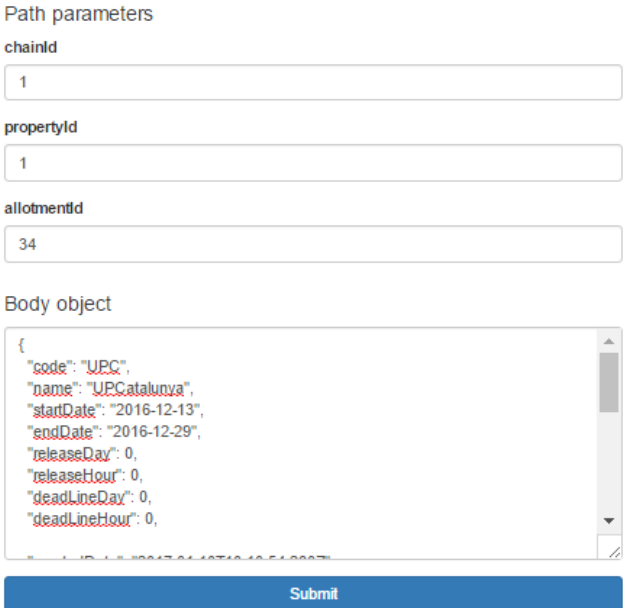


Figura 25: edición del allotment con id 34

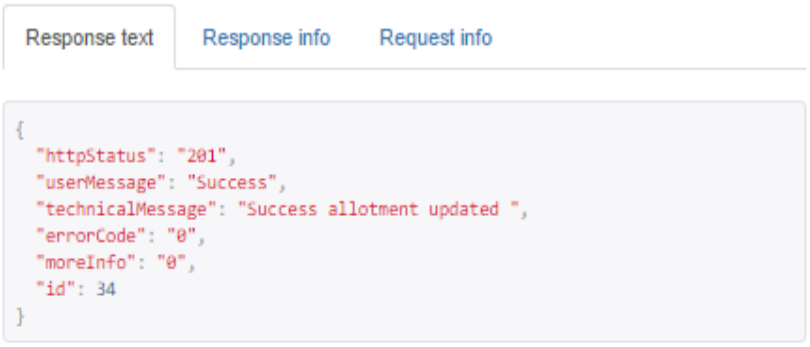


Figura 26: respuesta a la edición del allotment

Al igual que al crear un allotment, para editar tenemos que introducir en el body object los atributos del allotment que deseamos editar en formato JSON. En la imagen podemos ver que hemos editado el allotment con ID 34 que hemos creado anteriormente y hemos cambiado el nombre y el código. Para poder editar tanto el allotment como cualquier otra entidad usamos el método merge del EntityManager de la JPA, que nos permitirá crear una copia de los datos de la entidad pasada por parámetro a la instancia que está dentro del contexto de persistencia y poder manejarla.

6.2.5. Delete Allotment

Finalmente, para poder eliminar un allotment, como en el update, tendremos que introducir unos parámetros obligatorios como son la id de la cadena, la ID de la propiedad y la ID del allotment a eliminar. En este caso no se nos pedirá ningún Body Object ni otro dato más, solo los tres parámetros comentados anteriormente. Al igual que en el método Update o el GetById, la URL se extenderá con la ID del allotment a eliminar.

Como vemos en las imágenes siguientes, el allotment con la ID 34, se he eliminado exitosamente como nos dice el mensaje del ResponApiMessage.

Para poder eliminar un alloment y cualquier otra entidad, se utiliza el método remove del EntityManager de la JPA, para eliminar la entidad existente en la base de datos.

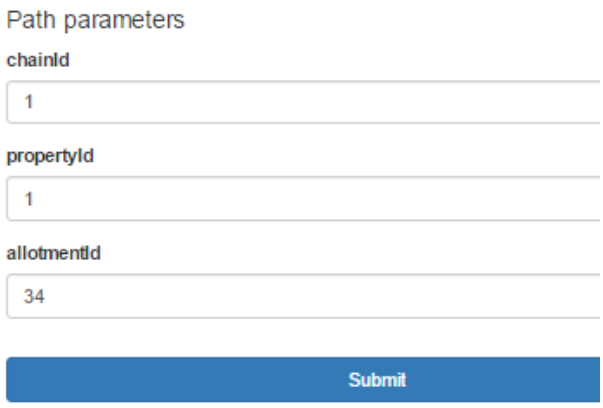


Figura 27: parámetros de petición para eliminar un allotment

Figura 28: respuesta al allotment eliminado con éxito

6.3. Gestión de AllotmentProduct

Una vez tenemos creada la entidad Allotment, se tiene que profundizar y relacionar los productos que tiene un cupo. Para esto mis compañeros me proporcionaron el diagrama de clases con las relaciones de esta nueva tabla y los atributos que tendrá.

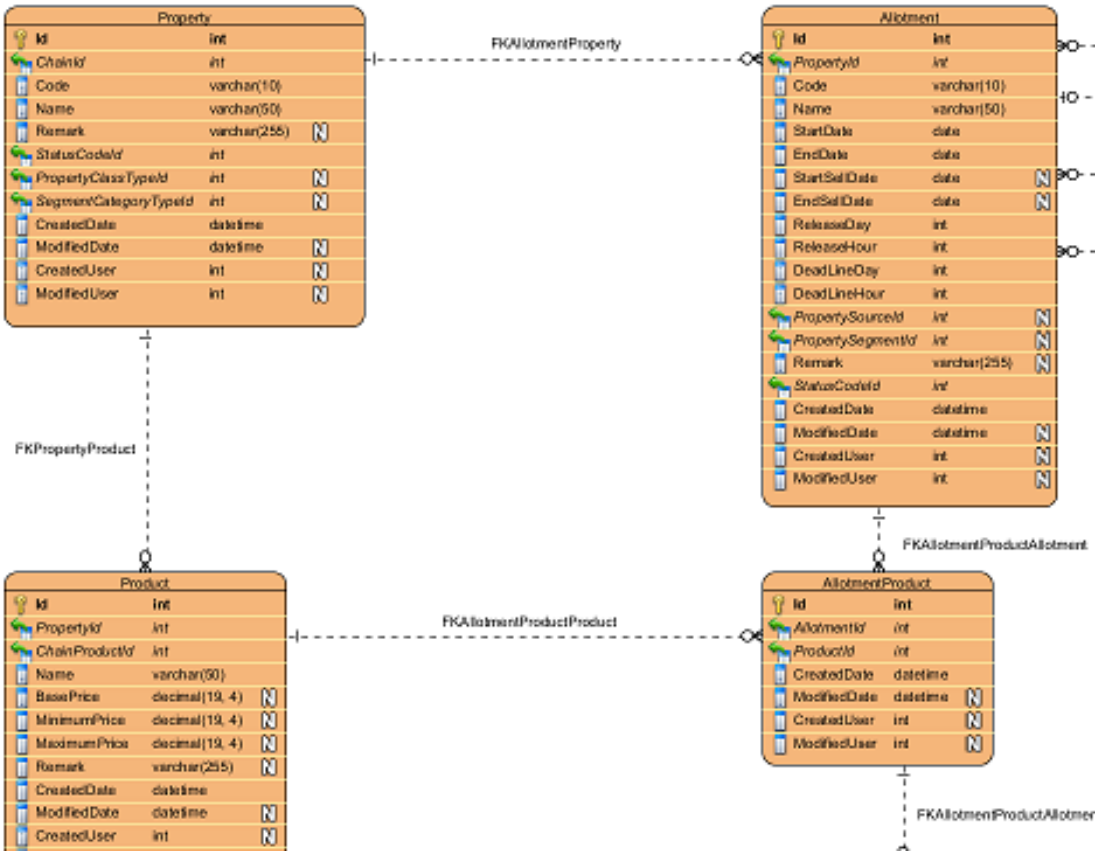


Figura 29: diagrama de clases de AllotmentProduct

Mediante el diagrama de clases podemos ver que un allotment tiene muchos allotmentProducts, cada allotmentProduct tiene su ID y dos claves foráneas como son la ID del producto y la ID del cupo.

Una vez entendido el comportamiento del AllotmentProduct y con datos en la BBDD, se empezó el desarrollo de la API para esta sección. Se creó la entidad de la tabla, además de los atributos de esta juntos con sus respectivos getters y setters y las anotaciones de Spring e Hibernate para la correcta inyección de dependencias y el buen funcionamiento del programa.

En el allotmentProduct tendremos los siguientes métodos:

- Método GET para listar los productos que tiene un allotment.
- Método GET para obtener un allotmentProduct pasando la ID por parámetro.
- Método POST para añadir un producto a un allotment.
- Método PUT para editar un allotmentProduct que se haya añadido previamente.
- Método DELETE para eliminar un allotmentProduct pasándole la ID por parámetro.

Se crearon más métodos pero para llevarse a cabo se necesitó las creación de nuevas tablas y la creación de la entidad de estas. Más adelante, una vez tenga explicado las nuevas entidad que se crean, se explicarán los método que se añadieron al allotmentProduct.

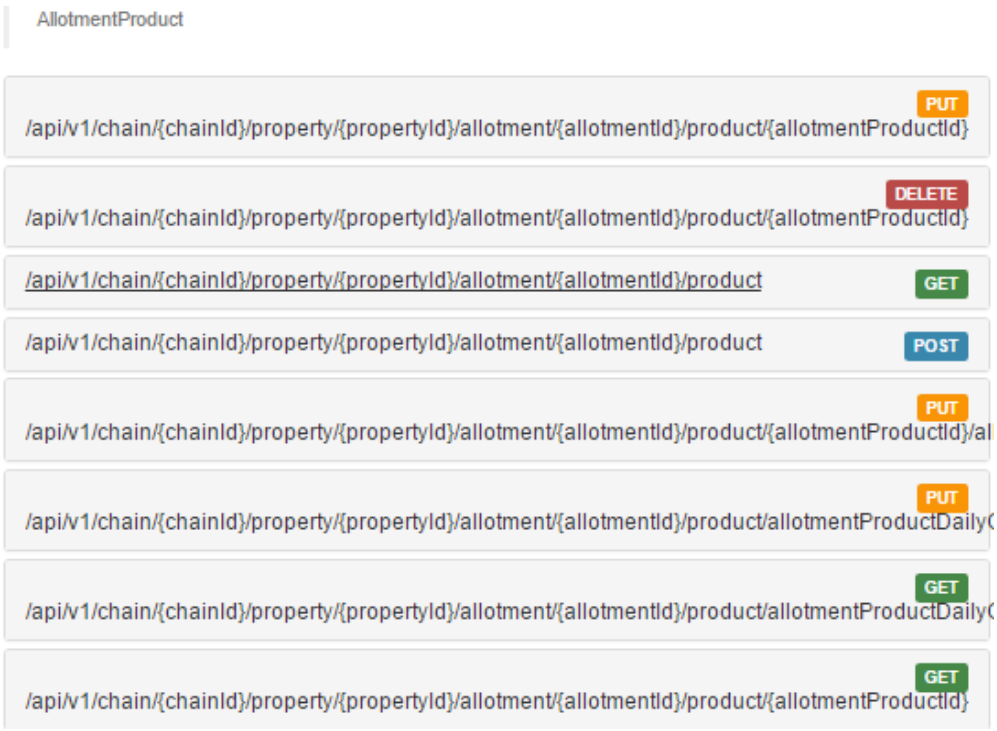


Figura 30: métodos de AllotmentProduct en la API

Como podemos ver en la imagen, hay métodos los cuales no he nombrado antes ya que se necesita de otras tablas para poder explicarlo. A continuación explicaré el CRUD básico del **AllotmentProduct** y después introduciré las nuevas entidades que se necesitan para poder explicar los métodos que faltan. A partir de aquí evitaré repetir explicaciones como los parámetros que se usan en la API o las anotaciones de JSON o Spring que se usan en las clases e iré directo a la explicación de los métodos los cuales son casi siempre bastante parecidos.

6.3.1. Get AllotmentProduct list

En este método GET se mostrará la lista de los productos que tiene un allotment. Se entrará como parámetros obligatorios la ID de la cadena, la ID de la propiedad y la ID del allotment.

Como en allotment, en el controlador se llamará el servicio de AllotmentProduct del cual obtendremos los métodos countByAllProperties y findByAllProperties para contar los productos y listarlos.

6.3.2. Get AllotmentProduct by ID

En este método se obtendrá la información específica de un AllotmentProduct. En este caso se necesitará además de los parámetros anteriores, la ID del AllotmentProduct al cual se quiere acceder. Para esto utilizaré una de las ID que me devolvía la lista de AllotmentProducts y vemos lo que devuelve información solo del allotmentProductId que se le pasa por parámetro.

6.3.3. Save AllotmentProduct

Con el método save se podrá añadir un producto al allotment que el usuario quiera. Tendremos que pasarle por parámetro la ID del allotment, además de la ID de la cadena y de la propiedad.

Como en todos los métodos de creación se tendrá que crear el Body Object, introduciendo los parámetros requeridos para poder llevar a cabo la creación de una instancia de la entidad.

6.3.4. Update AllotmentProduct

También se podrá editar un allotmentProduct pasándole la ID y un BodyObject en formato JSON que contendrá las ediciones de ese allotmentProduct en concreto. Para esto como he dicho anteriormente se utilizará el método merge que nos proporciona el EntityManager de la JPA.

6.3.5. Delete AllotmentProduct

Para llevar a cabo la eliminación de un allotmentProduct se pasará por parámetros las IDs de la cadena, la propiedad, el allotment y el allotmentProduct. De esta manera, con el remove que nos proporciona el EntityManager de la JPA, se eliminará dicho allotmentProduct de la base de datos.

6.4. Gestión de AllotmentProductDailyQuantity

Como dije antes, allotmentProduct está relacionada con otra tabla, AllotmentProductDailyQuantity. Esta tabla tiene otros atributos como son quantity para ver la cantidad de productos que tiene un allotment o closed para ver el estado de ese producto, entre otros. En el diagrama de clases siguiente podemos ver estos atributos y las relaciones con la tabla AllotmentProduct y Product.

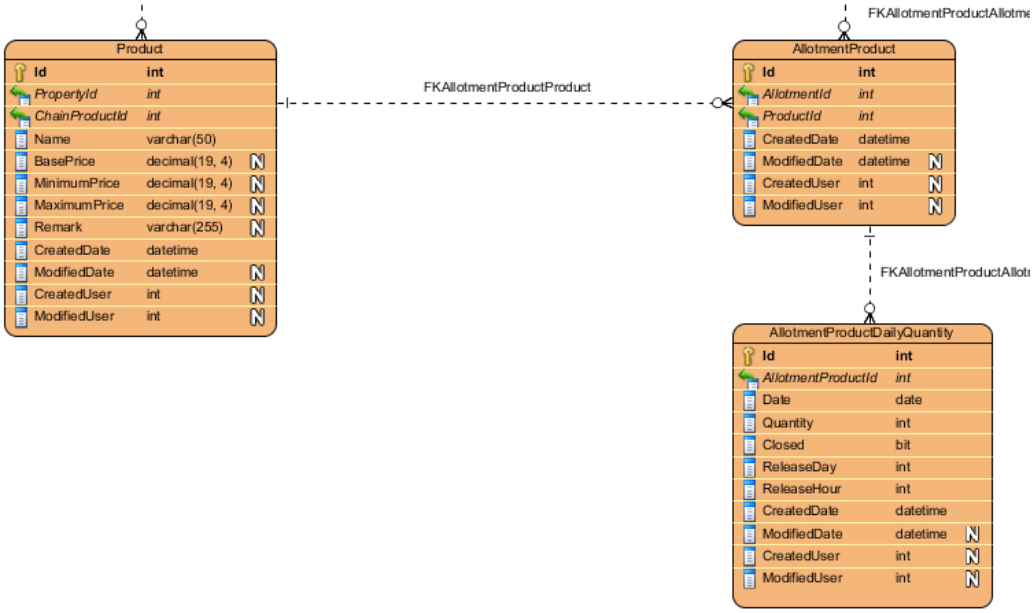


Figura 31: diagrama de clases de AllotmentProductDailyQuantity

Podemos ver mediante el diagrama de clases la relación entre AllotmentProduct y AllotmentProductDailyQuantity.

Se creará la entidad de AllotmentProductDailyQuantity con sus respectivos gettes y setters y las respectivas anotaciones para relacionarla con AllotmentProduct.

Una vez creada la entidad, se crearon otros tres métodos en AllotmentProduct:

- Método GET para obtener la lista de los productos de un allotment con toda la información de los atributos del AllotmentProductDailyQuantity como quantity, closed, releaseHour o releaseDay.
- Método PUT para editar atributos de AllotmentProductDailyQuantity de una lista de productos en un periodo determinado.
- Método PUT para editar un AllotmentProductDailyQuantity de un AllotmentProduct.

6.4.1. AllotmentProductDailyQuantity list

En este método GET obtendremos la lista de allotmentProducts, pasándole parámetros como la ID de la cadena, la ID de la propiedad y la ID del allotment.

Para un allotmentId obtendremos una lista de productos y para cada producto tendremos una lista de allotmentProducts, donde nos devolverá una ID para cada uno de estos y además una lista de allotmentProductDailyQuantity donde se especificará la ID de cada uno de estos, la fecha y el valor de los atributos que tiene este objeto.

Además en la API podremos especificar el intervalo de fechas en la que queremos que se nos muestre el allotmentProductDailyQuantity, mediante dos parámetros opcionales como son el offset y el limit.

6.4.2. Update AllotmentProductDailyQuantity

Para poder editar un allotmentProduct, tenemos que indicar dónde se va de editar, otra vez tendremos que entrar las ids correspondientes, cadena, propiedad, allotment y allotmentProduct.

Una vez accedemos al allotmenProduct correspondiente, tenemos que escribir en el body object la estructura del allotmentProduct que queremos modificar.

Se creerá una estructura en formato JSON, con la lista de allotmentProduct a modificar, especificando la id de cada uno de ellos y seteando a los atributos el valor que el usuario quiera.

6.4.3. Update intervalo de un AllotmentProductDailyQuantity

Este método PUT, también modifica un allotment, pero está pensado de tal forma que se puedan hacer cambios a gran escala, introduciendo en el BodyObject una lista de productos, una lista de intervalos de fechas y los atributos seteado son valores que el usuario quiera.

En el frontend, se hará un wizard donde llamará a este método de tal forma que nos abrirá un formulario y nos permitirá introducir estos campos de una manera rápida y clara, y así hacer cambios a gran escala para x productos de un allotment.

6.5. Gestion de Promotions (promociones en español)

La venta de productos es esencial para cualquier hotelero. Las campañas que estos realizan en los diferentes canales les pueden generar la necesidad de crear nuevos códigos que hagan sus ventas más atractivas que en otros canales de venta. En este contexto es necesario realizar un desarrollo que permita a los equipos de venta a generar nuevos códigos promocionales que permitan desde descuentos en una reserva hasta descuentos sobre productos añadidos a una reserva.

El desarrollo de la parte de promotions es a grandes rasgos muy similar a la de allotments, la cual he explicado anteriormente con detalles, así que en este apartado seré un poco más breve para evitar ser repetitivo.

En la parte de la API, igual que en allotments se crea la entidad en el dominio con sus respectivos atributos y especificando las relaciones que tenga promotion con otras entidades como pueden ser los productos o las propiedades de una cadena hotelera.

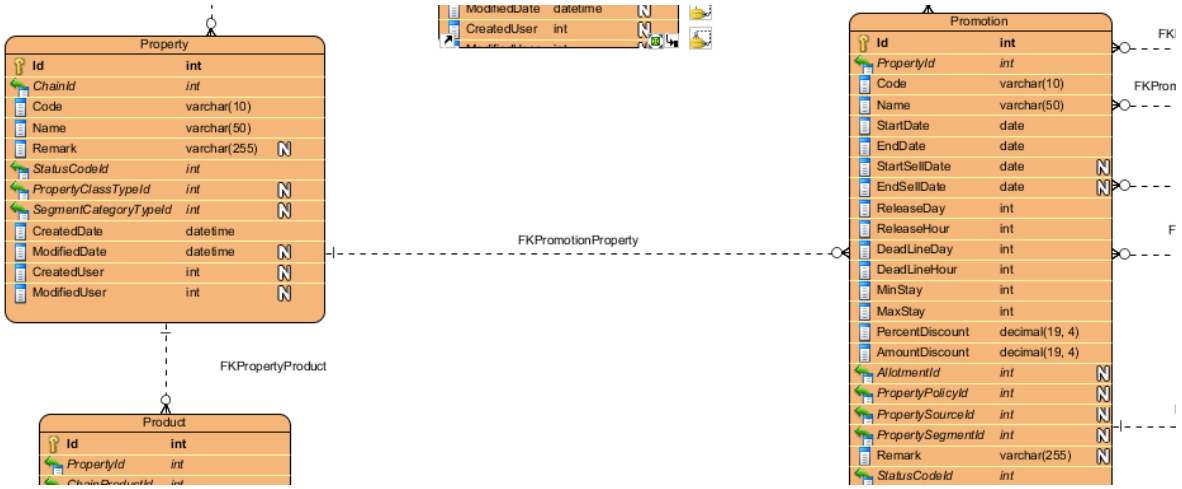


Figura 32: diagrama de clases de Promotion

Una vez creado la entidad, se crea el CRUD básico de una promotion. Eso se hará como siempre en el controlador. Si bien allotment y promotion tiene casi los mismos atributos y relaciones parecidas, una promotion está relacionada con allotment ya que un allotment puede tener muchas promotions.

Además más adelante veremos que entraran en juego los products occupancies, una parte muy importante de la gestión de promotions y que allotment no tendrá. Por lo tanto, veremos que promotion abarca mucho más y está relacionada con más tablas que allotment, siendo el desarrollo de la API más trabajado y complejo.

A continuación podemos ver los métodos de promotion de los que disponemos en la API.

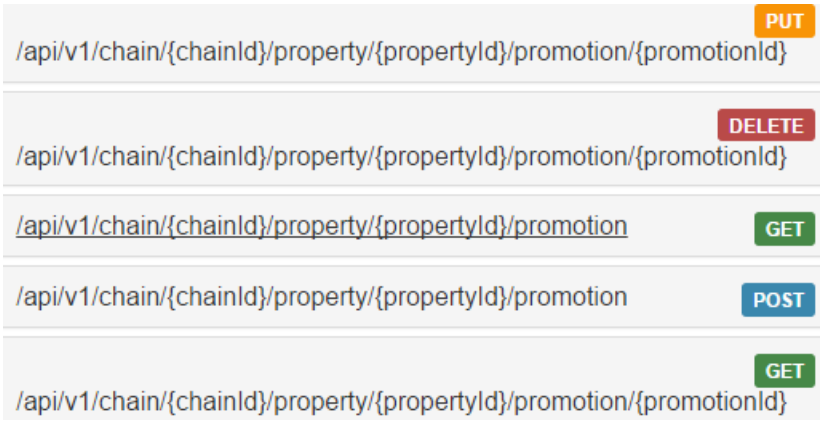


Figura 33: métodos de Promotion en la API

6.5.1. Get Promotion list

Como en allotment, la API nos devolverá una lista de promotions de la propiedad del hotel que le indiquemos en los parámetros que le estamos pasando. Cada promotion de la lista tendrá su ID y los atributos con sus respectivos valores, es decir, aquellos que se insertaron en la base de datos.

6.5.2. Get Promotion by ID

Para cada promotion de la lista devuelta en el método de antes, conseguiremos la ID de la promotion a la cual queremos acceder y obtener los datos de ese promotion detalladamente. Se tendrá que introducir esa ID junto con la de propiedad y cadena del hotel para poder obtener la información.

6.5.3. Create Promotion

En el método save del controlador se creerá una instancia de un nuevo promotion. Se tendrá que especificar la cadena y la propiedad de la cadena donde se quiere crear, además de un Body object con el objeto promotion en formato JSON donde se escribirá aquellos datos que son obligatorios como pueden ser el código, el nombre, fecha de inicio de la promoción, fecha final de la promoción, el percent discount y el amount discount en este caso.

6.5.4. Update Promotion

Se podrá editar la promotion mediante el método update. Para ellos, se tendrá que pasar por parámetro la ID de la promotion a editar y en el Body Object el objeto promotion a editar con los nuevos valores seteados por el usuario.

6.5.5. Delete Promotion

Si se desea eliminar una promotion, se tendrá que indicar la ID de la promotion a eliminar indicando la cadena y la propiedad a la que pertenece dicha promotion. Si la ID es la correcta se enviará un mensaje de éxito indicando la eliminación de la promotion.

6.6. Gestión de PromotionProductOccupancy

Como he dicho anteriormente, a diferencia de allotment, promotion está relacionada con PromotionProductOccupnacy y esta con ProductOccupancy. De esta manera se quiere acceder las ocupaciones que tiene un producto.

Como podemos ver en el diagrama de clases PromotionProductOccupancy estará relacionada tanto con Promotion como con ProductOccupancy.

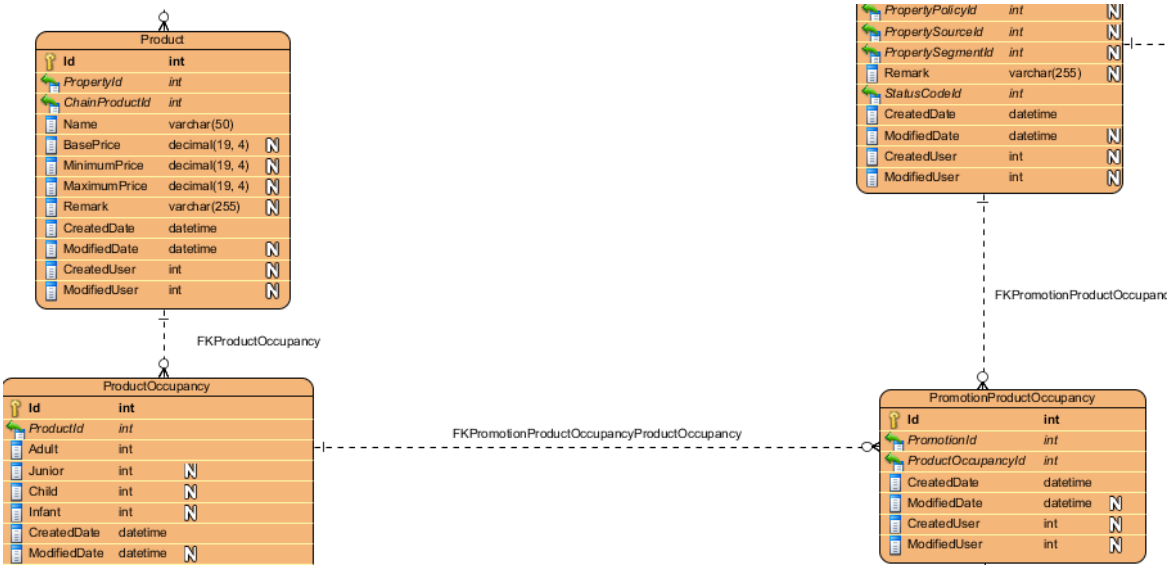


Figura 34: diagrama de clases de PromotionProductOccupancy

A partir de aquí podemos generar los métodos de PromotionProductOccupancy en la API en el controlador.

6.6.1. PromotionProductOccupany list

Se creará un método para obtener la lista de promotionOccupancyList. Como en los mantenimientos anteriores tendremos que pasar la ID de la cadena, la propiedad y además en este caso la ID de la promotion de la cual queremos ver los productos que tiene con las respectivas ocupaciones de cada uno de los productos de la lista.

6.6.2. PromotionProductOccupancyAvailability list

Para llevar a cabo este método tendremos que crear previamente la entidad PromotionProductOccupancyAvailability en el dominio y relacionarlo con PromotionProductOccupancy ya que como veremos en el diagrama de clases, un promotionProductOccupancy tiene una lista de PromotionProductOccupancyAvailability. Entonces para una promotion podremos ver la lista de productos con sus ocupaciones y la disponibilidad de cada uno de estos productos.

6.6.3. Create a PromotionProductOccupancy

Con este método se podrá añadir un productOccupancy a la promotion. Entrándole la ID de la promotion donde se quiere añadir el productOccupancy y creando el body object con la información necesaria.

6.6.4. Create a ProductOccupancy

En este método se podrá añadir listas de productos, junto con sus ocupaciones a una promotion. Esta hecho de tal forma para que en el frontend se permita añadir muchos productos con ocupaciones de una manera rápida y clara.

6.6.5. Update PromotionProductOccupancyAvailability

Como en allotment, se podrá editar la disponibilidad, en este caso de un promotionProductOccupancy. Se necesitará la ID de este promotionProductOccupancy para así editar sus atributos. En el body object se creará la lista con la disponibilidad con sus respectivos atributos editados a gusto del usuario, para cada uno de los días que tenga el promotionProductOccupancy.

6.6.6. Update PromotionProductOccupancyAvailability interval

Para poder editar diferentes promotionProductOccupancy de una manera rápida, se creará este método, el cual nos permitirá crear un BodyObject con listas de los products, listas de occupancies y lista de intervalos. De esta forma se podrá editar los productos que queramos en intervalos de tiempo que el usuario desee.

6.6.7. Delete PromotionProductOccupancy

Finalmente, se podrá eliminar la promotionProductOccupancy que se desee. Solo se necesitará la ID de esta para pasarla por parámetro, además de la propiedad y la cadena, y poder así eliminarla con éxito.

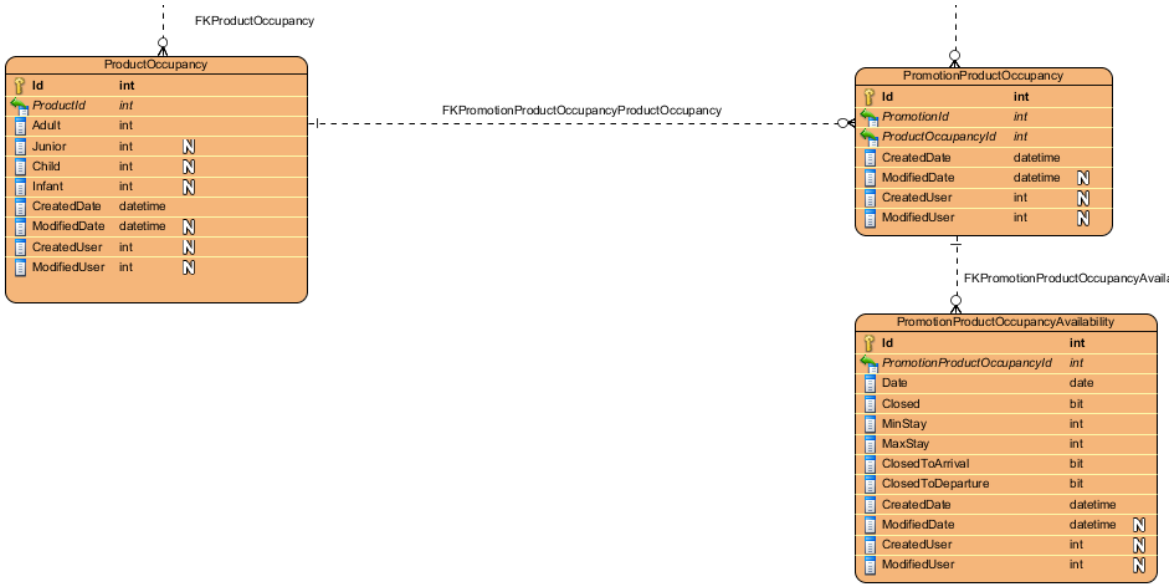


Figura 35: diagrama de clases de PromotionProductOccupancyAvailability

6.7. Gestión de Profiles (perfiles)

Los establecimientos hoteleros necesitan administrar dos grupos de perfiles; particulares o huéspedes, y agencias o compañías. Para ello es necesaria la implementación de mantenimientos para gestionar estas entidades. El objetivo de este desarrollo es permitir al usuario final identificar cada uno de ellos fácilmente, para poder realizar minería de datos en su explotación.

Como en los otros mantenimientos (allotment, promotion) usaré los diagramas de clase para ver las relaciones y los atributos que tiene profiles y poder así empezar la implementación de los métodos que se necesitarán en la API.

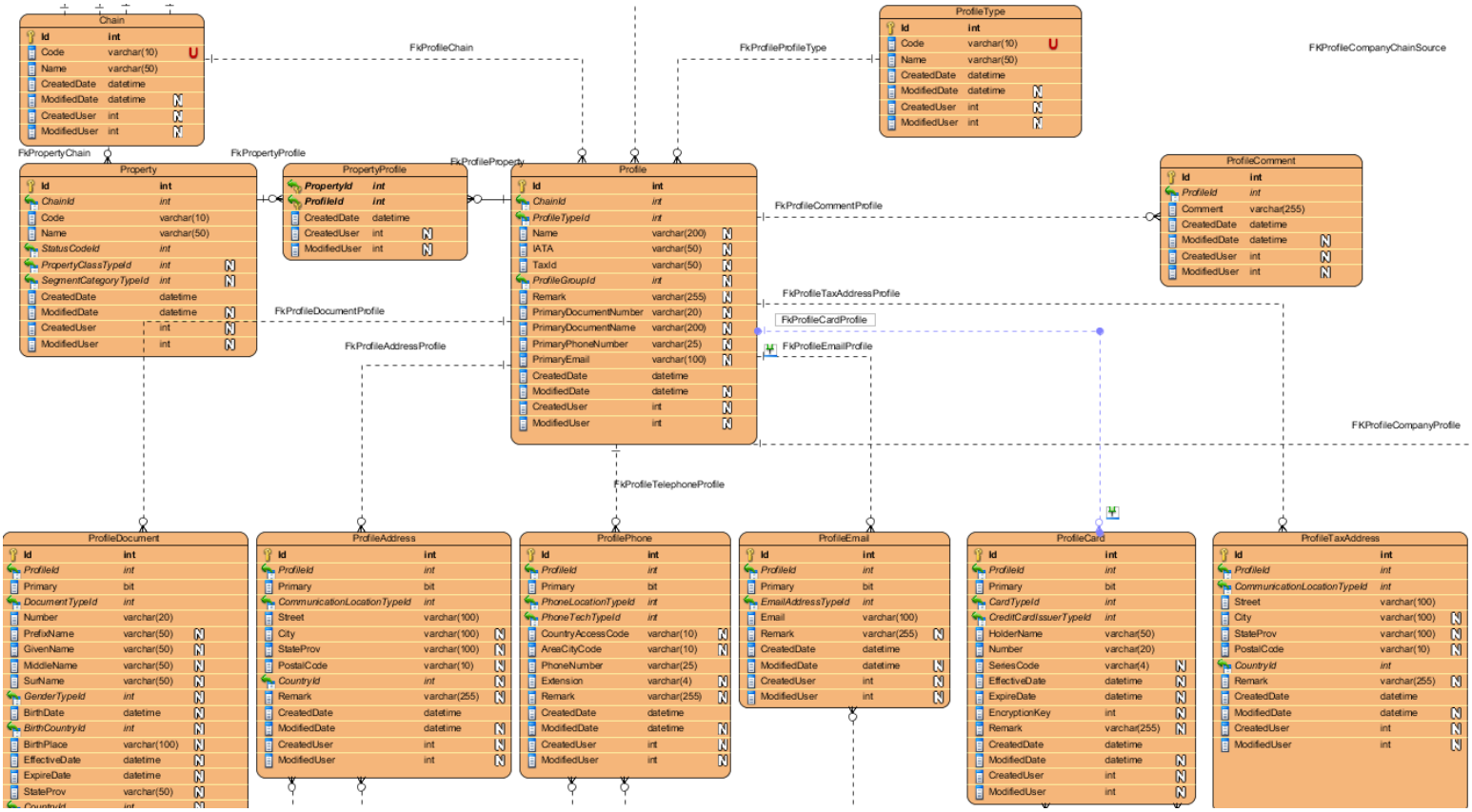


Figura 36: diagrama de clases de profiles

Podemos ver en el diagrama de clases que Profile está relacionado con muchas otras tablas, puesto es que los perfiles son una parte muy importante de la aplicación.

Debido a la cantidad de profiles que tiene una cadena de hoteles, se implementarán diversos filtros para poder facilitar la búsqueda al usuario. Dependiendo del tipo de perfil, se tendrá unos filtros u otros ya que perfiles particulares no tendrán exactamente los mismos atributos que una compañía o agencia.

Se creará la entidad de Profile en el dominio. Además se tendrán que crear otras muchas entidades como pueden ser ProfileAdress o ProfilePhone ya estas entidades irán relacionadas con Profile. Una vez tenemos todos las entidades creadas podremos empezar a hacer el CRUD básico en el controlador para poder obtener lista de profles, crear, editar y eliminar dichos profiles.

6.7.1. Get Profile list

Se obtendrá la lista de profiles de la cadena de hoteles cuya ID le pasemos por parámetro. Como dije anes debido a la numerosa cantidad de profiles que tiene una cadena hotelera habrá filtros para diferenciar entre tipos de profiles, (profileTypeId), grupos de perfiles (profileGroupId), email,

documento de identidad (documentNumber) en el caso de perfiles particulares o iata y taxId en el caso de un perfil de compañía o agencia, además de los filtros por defecto como el offset, limit, sort, o name.

6.7.2. Get Profile by ID

Se accederá a la información de un perfil, se deberá entrar por parámetro la ID del profile. Obtendremos información completa y detallada del perfil, por medio de las queries en el repositorio ya que con el JOIN FETCH podremos traer toda la información de las entidades relacionadas con Profile.

6.7.3. Create Profile

Se creará un profile en la cadena hotelera especificada mediante el body object creado por el usuario. El usuario deberá crear un JSON una estructura con los datos que debe tener un profile como pueden ser el tipo de profile (particular, compañía o agencia). Si fuera un profile particular se necesitarían datos como documento de identidad, nombre, fecha de nacimiento, etc. Sin embargo si fuera una compañía o agencia se necesitaría la iata o taxId

6.7.4. Update Profile

Como en los demás mantenimiento se podrá editar un profile, pasándole al método update la ID de la cadena donde se encuentra el profile a editar y la ID de dicho profile. Además en el body object se cambiará todos los valores del profile que el usuario haya editado.

6.7.5. Delete Profile

Finalmente para eliminar un profile, al igual que en los otros mantenimientos se necesitará la ID del profile a eliminar. Si el profile se ha eliminado con éxito se enviará un mensaje al usuario. Para asegurarnos que la operación se ha realizado con éxito, podríamos usar uno de los métodos GET y ver que no existe el profile o sino mediante consultas a la base de datos.

7. Desarrollo del Frontend

A medida que se iba implementado y ampliando la API comprobando que todos los método funcionaban correctamente, se podía pasar al frontend y empezar la interfaz gráfica. Vuelvo a repetir que cuando yo llegue a la empresa la aplicación ya estaba en desarrollo y las tecnologías o herramientas de desarrollo ya habían sido elegidas y configuradas. Sin embargo tuve que adaptarme y aprender dichas tecnologías o herramientas, de las cuales desconocía muchas.

7.1. Arquitectura de clases

Una vez tenía la API creada y testeada para cierta entidad, me disponía a desarrollar en frontend su interfaz de usuario, siguiendo la arquitectura limpia.

A continuación explicaré la arquitectura de las clases que se utilizan en el frontend y lo que cada una de estas contiene.

- **assets :** almacena los recursos gráficos, estilos y librerías
    - **images**
        - Almacena Imágenes generales de la aplicación
    - **styles**
        - Archivos CSS de la aplicación
    - **vendor**
        - Librerías de terceros vía Bower

- **src**: código fuente de la aplicación
    - **components :** Componentes reutilizables de la aplicación
        - ***/component***
            - **component.js**
            - **controller.js**
            - **view.html**
            - **configFactory.js**
                - Configuraciones del componente dependiendo de donde para que se use
    - **configs:** configuraciones varias de AngularJS, principalmente de módulos
    - **modals:** modales de la aplicación
        - **modals**
            - Modales personalizados
        - **asides**
            - Modales laterales
        - **forms**
            - Modales normales (formularios)
    - **sections**: vistas de los módulos de la aplicación
        - ***/section***
            - **controller.js**
            - **view.html**
    - **services:** servicios para organizar y compartir código en la app
    - **translations**: ficheros de traducciones
    - **app.js**: definición de la aplicación AngularJS
- **tests**
- .**bowerr:** Configuración para Bower
- **.editorconfig**: definición de reglas de edición de archivos
- **.gitignore**: reglas de exclusión de directorios y archivos para VCS.
- **htaccess**: reglas de acceso para Apache Web Server
- **bower.json**: configuración de dependencias Bower
- **index.html**: archivo de acceso a la aplicación
- **favicon.ico**: icono de la aplicación
- **README.md**: información general sobre el proyecto
- **robots.txt**: reglas de indexado para buscadores
- **karma.conf.js**: configuración de tests unitarios
- **jshintrc:** configuración del jshint

7.2. Análisis del modelo utilizado

Ser capaz de categorizar un marco y ponerlo en uno de los patrones MV* (MVC, MVVM, MVP, entre otros) tiene algunas ventajas.

Puede ayudar a los desarrolladores a estar más cómodos con sus apis facilitando la creación de un modelo mental que represente la aplicación que se está construyendo con el framework. También puede ayudar a establecer la terminología utilizada por los desarrolladores

Angular nos da mucha flexibilidad para separar bien la lógica de la presentación de la lógica de negocio y el estado de la presentación.

MVC y sus derivados (MVP, PM, MVVM) son todos buenos dentro de un solo agente, pero una arquitectura de servidor-cliente es para todos los propósitos un sistema de dos agentes, y las personas están a menudo tan obsesionadas con estos patrones que se olvidan de que el problema es mucho más complejo. Al tratar de adherirse a estos principios terminan con una arquitectura defectuosa.

Para entender un poco más la arquitectura que se sigue en este proyecto, explicaré el flujo de datos y las diferentes capas en las que trabajo.

7.2.1. Views

En el contexto de angular, la vista es el DOM (Doman Object Model, en inglés). Presenta la variable scope (solo lectura) y se llama al controlador para acceder a funciones o los modelos.

7.2.2. Controller

El controller tiene como características principales:

- Vincular la vista con el modelo colocando datos en el Scope.
- Responder a las acciones del usuario
- Trata la lógica de la presentación
- Es muy recomendable evitar la lógica de negocio
- Se puede comunicar con otros controladores invocando métodos o usando $emit, $broadcast, $on para manejar los eventos que suceden en nuestra aplicación

7.2.3. Model

A menos que esté haciendo una aplicación web offline, o una aplicación que sea terriblemente simple (pocas entidades), es probable que el modelo de cliente sea:

- **Parcial**: o bien no tiene todas las entidades, o no tiene todos los datos.
- **Stale**: si el sistema tiene más de un usuario, en cualquier momento no se puede estar seguro de que el modelo que el cliente tiene es el mismo que el que tiene el servidor.

El modelo es la única cosa que persiste. Siempre que hablamos de modelos, éstos deben persistir en algún momento.

Como consecuencia, los dos puntos anteriores deben servir como una advertencia, es decir, el modelo que el cliente posee sólo puede implicar una lógica de negocios parcial, en su mayoría simple.

Tal vez uno de los conceptos más importantes sobre los modelos de negocio es que se puede subdividirlos en dos tipos:

- **Lógica de dominio**: la lógica que es independiente de la aplicación. Por ejemplo, dar un modelo con propiedades code y name, un getter como getName () puede considerarse independiente de la aplicación.
- **Lógica de aplicación**: que es específico de la aplicación. Por ejemplo, las comprobaciones de errores y la manipulación.

La lógica de la aplicación (no la lógica del dominio) debe tener la responsabilidad de facilitar la comunicación con el servidor y la mayoría de la interacción del usuario, mientras que la lógica del dominio es en gran parte de pequeña escala y específica de la entidad.

Todos los modelos MVC y sus derivados (MVP, MVVM, PM) utilizan 3 capas: Modelo, Vista y Controlador.

Pero hay dos cuestiones fundamentales con esto cuando se trata de clientes:

- El modelo es parcial, rancio y no persiste.
- No hay lugar para poner la lógica de la aplicación.

Una alternativa a esta estrategia es la estrategia de cuatro capas:

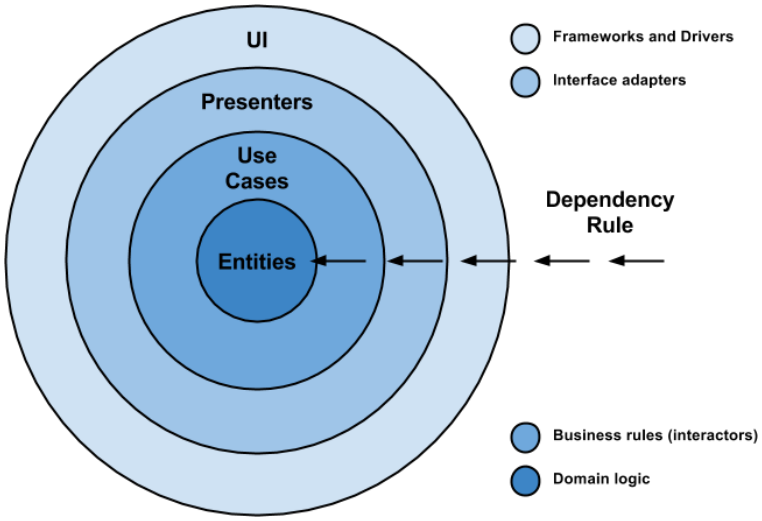


Figura 37: diagrama de la arquitectura utilizada

Esta arquitectura tiene como objetivo la separación de preocupaciones dividiendo el software en capas. Tiene al menos una capa para las reglas de negocio y otra para las interfaces.

El problema real aquí es la capa de reglas empresariales de aplicación (casos de uso), que a menudo va mal en los clientes.

Esta capa es realizada por los interactores que es más o menos lo que Martin Fowler llama una “operation script service layer”.

Considere la siguiente aplicación web:

- La aplicación muestra una lista paginada de usuarios.
- El usuario hace clic en "Agregar usuario".
- Se abre un modelo con un formulario para rellenar los detalles del usuario.
- El usuario rellena el formulario y pulsa enviar.

Algunas cosas deben suceder ahora:

- El formulario debe ser validado por el cliente.
- Se enviará una solicitud al servidor.
- Se debe manejar un error, si lo hay.
- La lista de usuarios puede o no (debido a la paginación) actualizarse.

Pero, como se trata todo esto?

Figura 38: diagrama en el que se añade la capa de lógica de aplicación

Una posible solución sería añadir una capa de lógica de aplicación entre el controlador y $resource, a la que se le llama “interactor”.

- Es un servicio, en el caso de los usuarios, puede denominarse UserInteractor.
- Proporciona métodos correspondientes a casos de uso, encapsulando la lógica de la aplicación.
- Controla las peticiones hechas al servidor. En lugar de un controlador que llame a $resource, esta capa garantiza que las solicitudes hechas al servidor devuelvan datos sobre qué lógica de dominio puede actuar.
- Decora la estructura de datos devuelta con el prototipo de lógica de dominio.

7.3. Gestión de Allotment en el frontend

A continuación explicaré como se llevó a cabo el desarrollo de la interfaz gráfica de allotment. Se creará una carpeta dentro de **sections** ubicada en el directorio src. Se le llamará propertyAllotment y contendrá dos archivos: un controller.js y una view.html, además de otra carpeta llamada details que contendrá otros dos archivos llamados igual que los anteriores.

Para llevar a cabo el desarrollo de la interfaz gráfica primeo habrá que seguir unos pasos que nos permitirá ser más ordenados y evitar errores que nos puedan llevar tiempo identificar y resolver.

Para empezar se define el lugar donde queremos que se sitúe la section allotment. Esto se hará en la carpeta **navigation** ubicada en el directorio de **components**. Se definirá el título de la section y el path.

El proyecto tendrá además una carpeta **breadcrumbs** ubicada también en el directorio de components, donde se definirán unas rutas que nos permitirá navegar hacia atrás o adelante en el entorno de allotment.

Además se tendrá que configurar la ruta de allotment en el archivo ngRoute.js ubicado en la carpeta config. Para ello necesitaremos el módulo **ngRoute** para gestionar las rutas internas de la aplicación. El papel de la inyección de dependencias juega un papel muy importante en todo este desarrollo ya que tendremos que añadir **ngRoute** como dependencia en el archivo app.js. Este módulo nos proporcionara el **$routeProvider** que nos permitirá configurar las diversas rutas de la aplicación.

La vista de Allotment tendrá una grid principal, donde podremos ver la lisa de allotments para una property y una cadena. Cada elemento de la grid tendrá su detalle donde accederemos clicando sobre el código del allotment. Una vez dentro del detalle, tendremos dos pestañas:

- **Details**: donde tendremos toda la información del allotment la cual se podrá editar el allotment o eliminar el allotment.
- **Availability**: la cual nos mostrará una grid con los productos de un allotment, además de poder editar la disponibilidad, añadir o quitar productos del allotment.

Para la **grid** principal se usa el módulo **UI-Grid** que nos proporciona AngularJs para mostrar los datos de allotment además de poder filtrar, ordenar o agrupar los datos.

7.3.1. Vista principal de Allotment

En la vista principal tendremos la tabla con la lista de allotments en la grid, el breadcrumb, el botón para crear un allotment, un filtro por nombre y un botón para configurar la información que se muestra en la grid.

7.3.1.1. Get Allotment list

En la vista se pasará el nombre del tipo de grid, en este se llamará "**propertyAllotments**", desde donde mediante la etiqueta de la component grid se accederá a la configuración de la grid hecha en la carpeta grid ubicada en el directorio de components. Desde el archivo configFactory.js se definirán las columnas de la grid, las respectivas traducciones, en el caso de que la lista de allotment este vacía se dará feeddback y se configurará el filtro por nombre.

Desde el controlador de la componente grid, se harán llamadas al servicio ApiService, donde dependiendo de la id de la grid, se hará una llamada u otra. En este caso la id es la que tenemos en la view del controlador “propertyAllotments”, y en el ApiService se hará la llamada de un get para obtener la lista de allotments especificada por la id.

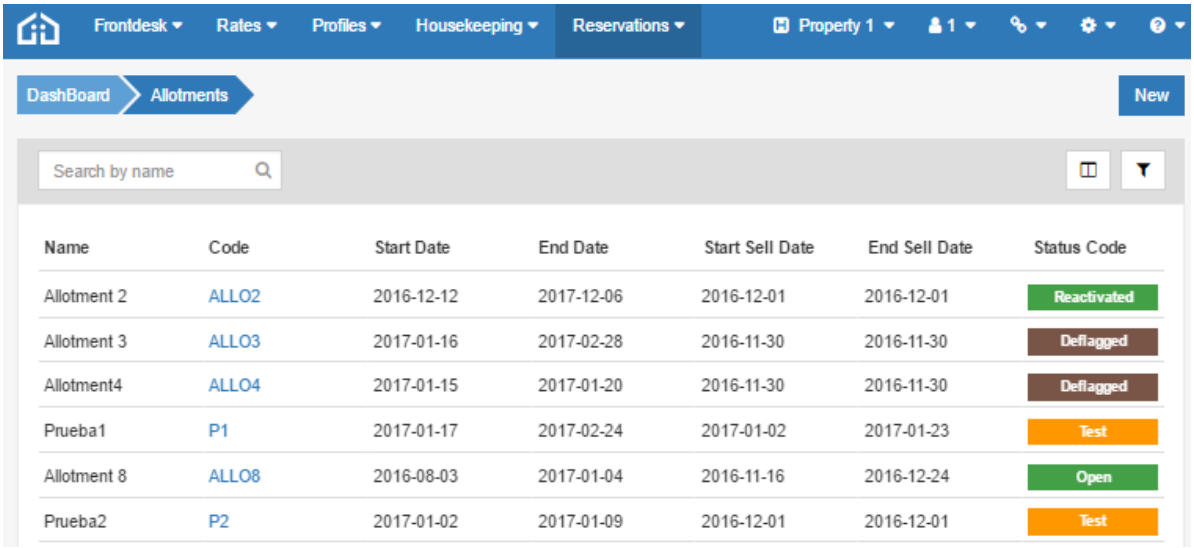


Figura 39: vista principal con la lista de allotments

7.3.1.2. Create Allotment

Desde la vista también se hará que el botón llame al modal de creación del allotment mediante la función definida en el controlador de allotment.

En el controlador de allotment se inyectara la dependencia de Modal, que será un servicio que nos permitirá hacer enlace entre el controlador y los modales de creación, en este caso será el modal de creación del allotment. Desde el servicio modal se definirá la ruta del directorio de la vista donde está el modal de creación de allotment, el nombre del controlador del modal de creación y un alias que se usará en la vista.

En el controlador del modal de creación de un allotment se gestionará la llamada a API para crear un allotment, mediante la creación de un modelo y los campos que se añadirán al formulario de creación.

Una vez creada la vista principal del allotment, se pasará a gestionar el detalle del allotment. Desde el configFactory de la componente grid, en la columna code, se permitirá el enlace desde el code del allotment en la grid al detalle de esta.

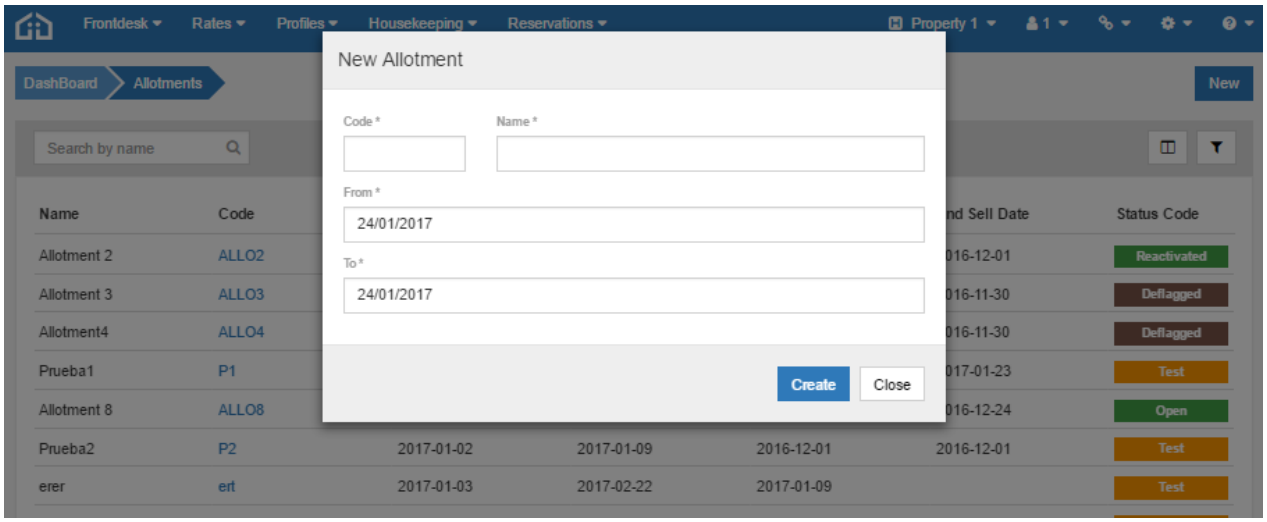


Figura 40: modal de creación de un allotment

7.3.2. Vista del detalle del Allotment

En la carpeta details de la section de propertyAllotments tendremos el controlador de la vista y su respectiva vista.

En el controlador, se crearán todos los métodos que permitirán al usuario interactuar con un allotment en concreto.

El detalle consta de dos pestañas: detail y availability. En la pestaña de detail, se nos mostrará toda la información del allotment, se permitirá editar y eliminar el allotment y podremos ver el breadcrumb para poder volver atrás. Se tendrá un header con la información básica del allotment la cual no será modificable, se modificará una vez editemos los inputs que están debajo de este header.

Tendremos dos botones en el footer tal que al editar alguno de los campos del detalle el botón save estará habilitado y nos permitirá guardar estos cambios. Si queremos cancelar los cambios podremos clicar al label cancel para cancelar los cambios y hacer que el botón save se deshabilite. También tendremos información del día y la hora en la que se modificó o se creó el allotment.

Se utilizará el módulo **angular-formly,** que tiene como finalidad trasladar mucho del código html a javascript, permitiendo la creación y personalización de templates, el uso de validaciones a través de expresiones o incluso ser extendidas mediante la creación de funciones personalizadas, además de una serie de atributos sumamente interesantes para la creación de formularios. Se utilizará un template que añade clases específicas para **BootStrap**. Se permitirá la creación de template customizados en el archivo formly.js mediante la inyección del servicio **formlyConfig**.

7.3.2.1. Get allotment by ID

Para obtener la información del allotment, se hará una llamada desde la función activate del controlador al servicio **ApiService**. Desde este servicio se harán las llamadas a la API. En este caso

mediante el servicio **$routeParams** inyectado en el controlador del detalle de Allotment, se obtendrá y se pasará la id de la property y la id del allotment de la actual ruta, que serán los parámetros para mandar a la petición.

Se necesitará instalar el módulo **ngResource** para el acceso a datos de la API REST ya que desde el servicio **ApiService**, devolveremos el servicio **$resource** proporcionado por **ngResource**, en función de la llamada que se haga, en este caso queremos obtener la información de un allotment concreto.

Para obtener la información deseada de la API se recurrirá a otro servicio, en este caso el **ApiRoute** donde se configurarán las rutas para acceder a la API.

La respuesta que se reciba de la llamada a la API, puede tener éxito o no. Para eso se tendrán dos funciones para tratar las respuestas de las peticiones que se hagan.

En la view.html mediante la directiva **formly-form** podemos tratar el formulario a ser representado, accediendo a él mediante el controlador, además de configurar los campos del formulario que se crean en el controlador.

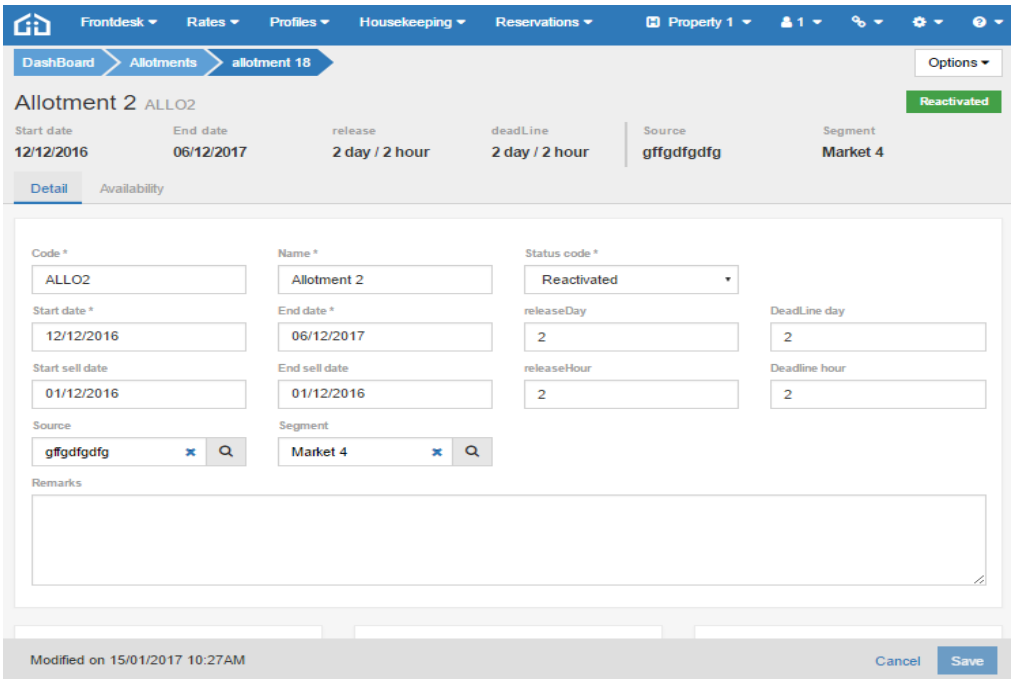


Figura 41: vista del detalle de un allotment

7.3.2.2. Update Allotment

En la pestaña details, se podrá editar la información del allotment. Para llevar a cabo esto, desde la función **submitGeneralForm**, se tendrá que hacer una llamada a API, en este caso será la llamada update, donde se pasará la id del allotment como parámetro de la petición y el **generalFormModel** creado con los cambios hechos por el usuario.

Para poder guardar los cambios, tendremos un botón save que estará deshabilitado si la función **generalFormModified()** devuelve true, y lo hará si el **generalFormModel** y el **generaFormModelOriginal** son iguales. Para esto utilizaremos **el angular.equals()** que compara dos objetos, en este caso los modelos.

7.3.2.3. Delete Allotment

Para eliminar un allotment se podrá hacer desde la pestaña details o la pestaña availability, se creará un método **deleteAllotment** () en el controlador que será llamado desde el botón delete de la vista. Este método llamará al servicio modal, donde se creará el método **openDeletionConfirm(action, entity)** y mediante el servicio **$uibModal** se creará un modal donde se obtendrá el path del template y el controlador, donde se gestionará la eliminación del allotment, es decir aparecerá un modal con un mensaje pidiendo confirmación para eliminar el allotment. Después se hará una llamada a API del método delete donde se pasará la id del allotment como parámetro de la petición y seguidamente se gestionará la respuesta de la petición.

En la pestaña availability tendremos el header con la información más importante del allotment, la grid creada en la view.html, los botones en el footer que nos permitirá guardar o cancelar los cambios de la grid y otros botones que serán editar allotmentProduct, añadir allotmentProduct y eliminar un allotmentProduct.

La grid constará de unas filas que son los allotmentProducts y las columnas que corresponden a los días. También se utilizará el componente dateSelect para navegar en el calendario. Se mostrará otra columna donde para cada allotmetnProduct tendremos los atributos como quantity, closed, releaseDay o releaseHour, además se mostrará los atributos que el usuario quiera y poniendo colores a las columnas del día actual, días pasados y fines de semana.

7.3.2.4. Update AllotmentAvailability desde la grid

Para editar la grid de allotmentAvailability hay dos maneras. Una manera es haciendo una llamada a API en la función **submitAllotmentProductsAvailabilityGridModel().** Se llama al método **setAllotmentProductDailyQuantity** pasándole la id del allotment y la id del allotmentProduct como parámetros de la petición.

De esta manera si un usuario edita los valores de la grid y se quieren persistir estos cambios se tendrá que guardar con el método save que estará habilitado si los modelos **allotmentAvailabilityGridModel** y **allotmentAvailabilityGridModelOriginal** son diferentes.

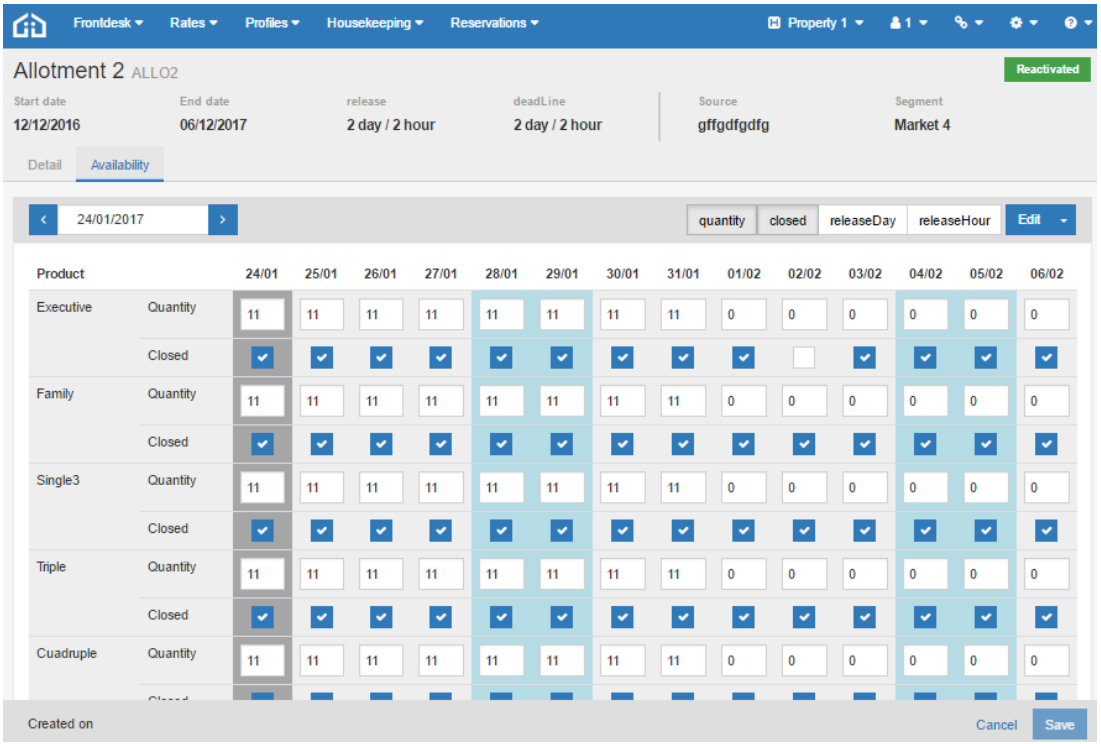


Figura 42: vista del pestaña availability de un allotment

7.3.2.5. Update AllotmentAvailability desde un modal

Se podrá editar la grid del allotmentAvailability a partir del botón editar en la cual se creará un modal mediante diferentes servicios. En esta ventana se podrá indicar el allotmentProduct a añadir, el intervalo de tiempo, modificar los valores de los atributos que desee el usuario. Además se podrán añadir diferentes allotmenProducts y diferentes intervalos de tiempo.

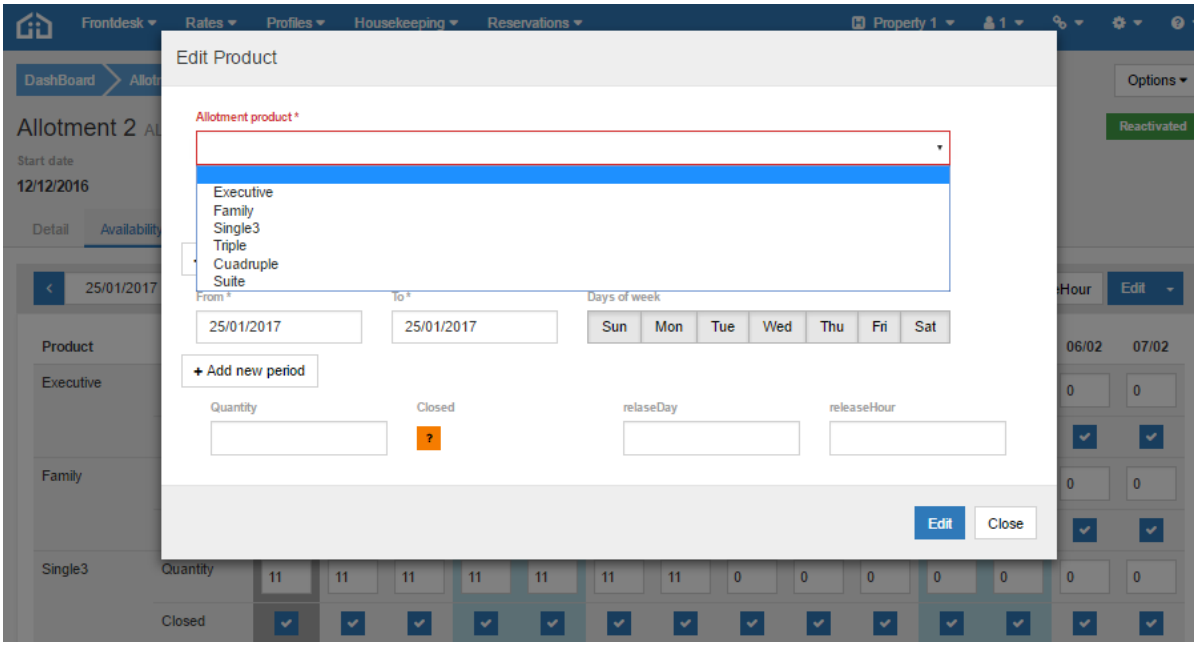


Figura 43: modal para editar la availability grid los allotmentProducts

Todo esta gestión, es decir, la creación de los campos, la creación del modelo, las llamadas a la API como **setAllotmentProductDailyQuantityInterval**() se lleva a cabo desde el la carpeta **forms** ubicada en el directorio **modals**, más concretamente en el controlador de **PropertyAllotmentProductUpdate**.

7.3.2.6. Añadir AllotmentProduct

También se podrá añadir un allotmentProduct al allotment. Mediante el botón add, se añadirá un allotmentProduct a partir del modal de creación. Se creará la función **openAddProductToPropertyAllotment()** en el controlador y mediante el servicio modal se enlazara con el controlador y el template que llevará la gestión de la adición del allotmentProduct. Es decir en este nuevo controlador se harán las diferentes llamadas a API, se creará el model y los campos necesarios para añadir el producto que el usuario desee.

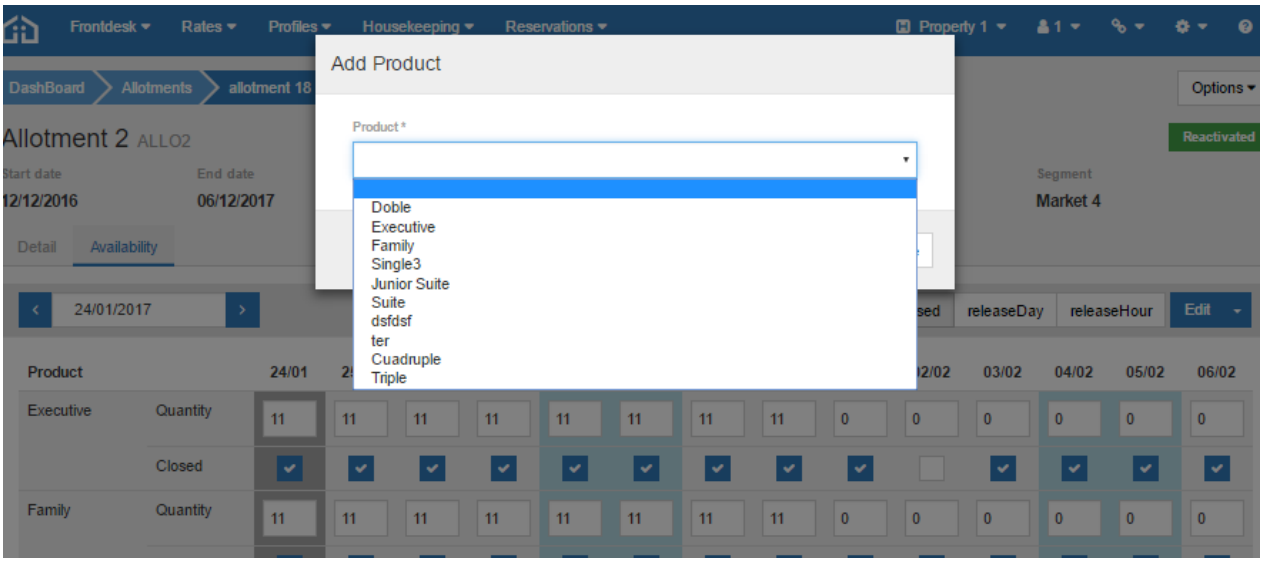


Figura 44: modal para añadir un allotmentProduct a la grid

7.4. Gestión de Promotions en el frontend

Puesto que el mantenimiento de promotions es muy similar al de allotments, seré muy breve en la explicación de esta gestión ya que no varía mucho excepto por la introducción de nuevos atributos y las **ocupaciones** que tienen los productos.

La configuración del **breadcumb** y **navigation** será la misma cambiando de nombre. Así mismo, la configuración de las rutas en el **ngRoute** será parecido solo cambiando las nomenclaturas de allotments por promotions.

7.4.1. Vista principal de Promotion

La grid principal de promotions se configurará igual que se hizo en allotments, pero esta vez promotions tendrá más atributos como **percent o amount discount**, para poner en las columnas.

La grid también tendrá filtros por nombre y se podrán hacer las mismas operativas como ordenar que se hacían en allotment.

A la hora de crear una promotion, se utilizaran diferentes servicios para que mediante un modal de creación aparezca la ventana con el formulario parecido al de allotment pero introduciendo dos campos obligatorios como el percent discount y el amount discount.

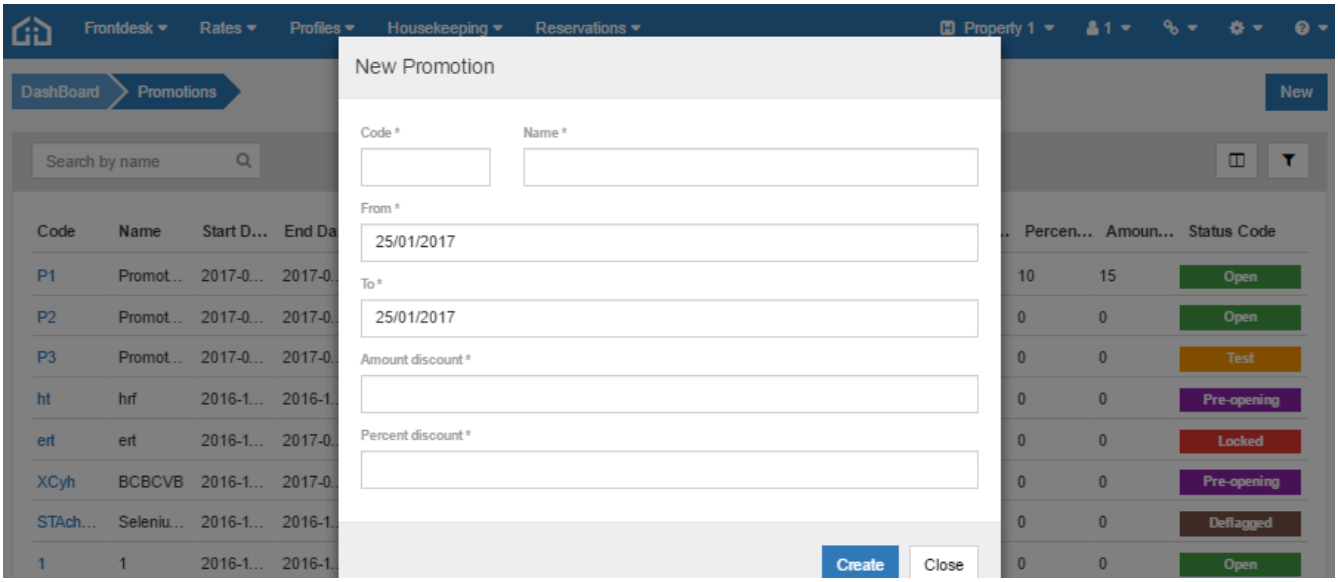


Figura 45: modal de creación de una promotion

7.4.2. Vista del detalle de Promotion

En el detalle de una promotion, la pestaña de details y la pestaña de availability serán muy parecidas. Se podrá eliminar una promotion desde cualquiera de las dos pestañas. Y también se podrá editar y guardar un promotion igual que se hacía en allotment.

En la pestaña de availabilty está la gran diferencia ya que entran en juego las ocupaciones de los productos. La grid se crea en el html al igual que en allotment pero con la diferencia que un promotionProduct tiene ocupaciones.

De esta manera para cada promotionProduct se tendrá unas ocupaciones. Al modificar una ocupación se modificará el parent del promotionProduct y si se modifica el parent, se emitirá el cambio hacia los hijos, cambiándose el valor en todas sus ocupaciones.

Como en allotment, habrá un componente **dateSelect** para navegar entre fechas, además de poder seleccionar los atributos que se quieran mostrar en la grid. Además, habrá diferentes colores en las columnas según el día actual, los días pasados y los fines de semana, en el caso que sean días pasados la grid estará deshabilitada y no se podrá editar la grid.

Al igual que en allotment se podrá editar las grid de dos formas. De una manera interactuando con la grid y cambiando los valores que se deseen y luego guardando estos cambios. También se podrá editar la grid de manera específica mediante un botón edit vinculado a una función que mediante

diferentes servicios como el modal, nos permitirá abrir una ventana de edición en donde se podrá especificar mediante selectores el producto y la ocupación o todas las ocupaciones a editar. Además, se podrán especificar los intervalos de tiempo, los días de la semana a modificar, modificar los atributos de promotion, y nos permitirá añadir más de un producto y más de un intervalo de tiempo.

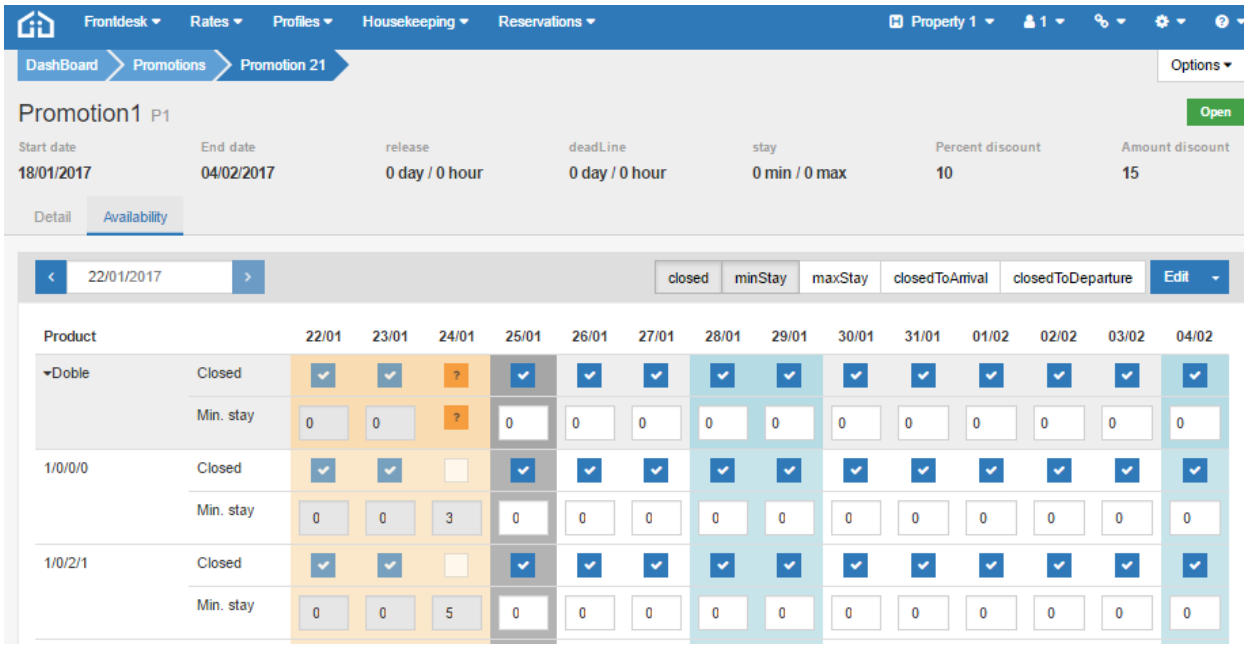


Figura 46: vista de la pestaña de availability de una promotion

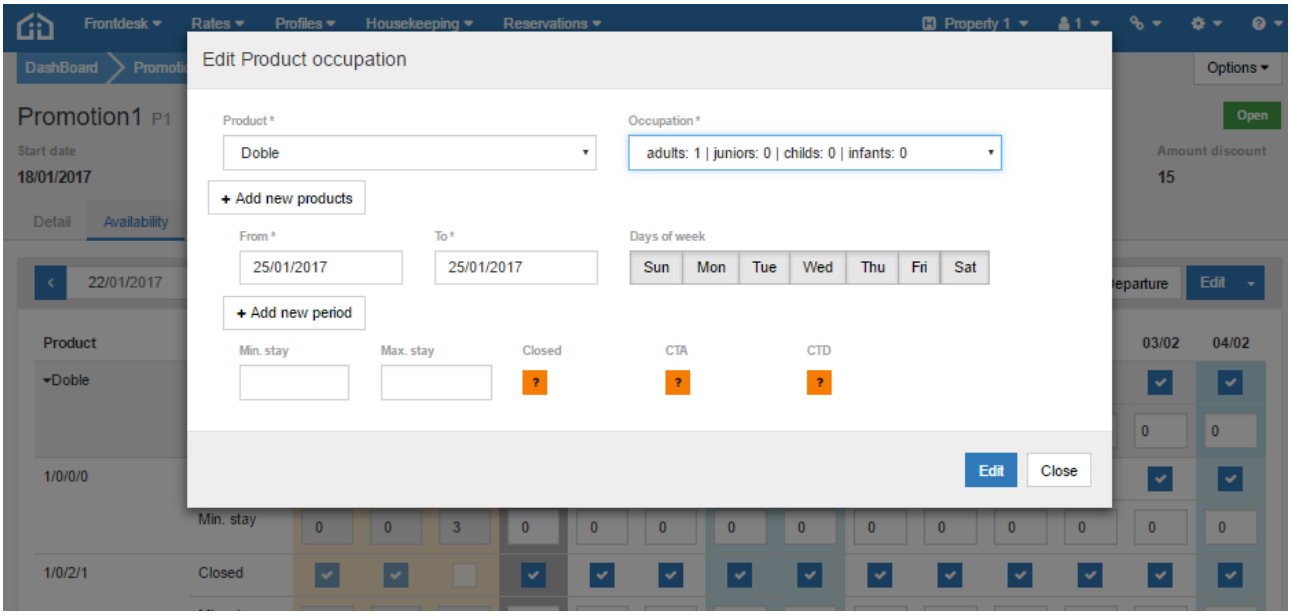


Figura 47: modal para editar la grid de availability en promotion

Para añadir una promotionProduct, se hará mediante el botón de add que llamará al modal respectivo y abrirá una ventana para especificar el o los productos con las ocupaciones que se quieran añadir mediante los selectores correspondientes.

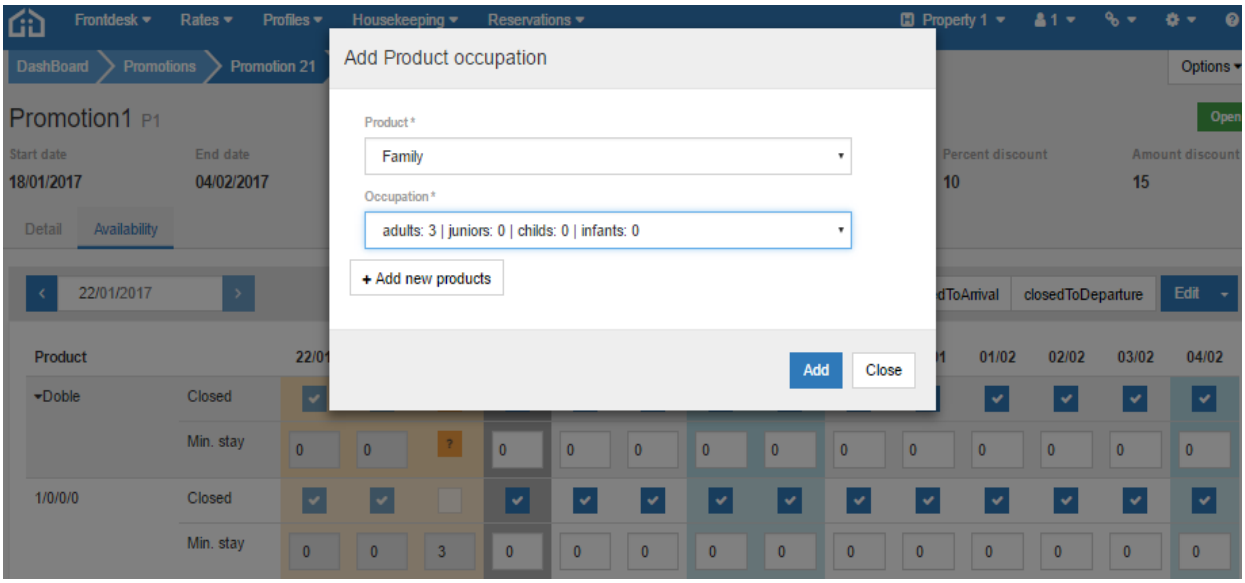


Figura 48: modal para añadir un productOccupation a la grid

Finalmente para eliminar, se clicará el botón de remove del grupo de botones y permitirá eliminar ocupaciones que el usuario desee para un producto especificado mediante un selector.

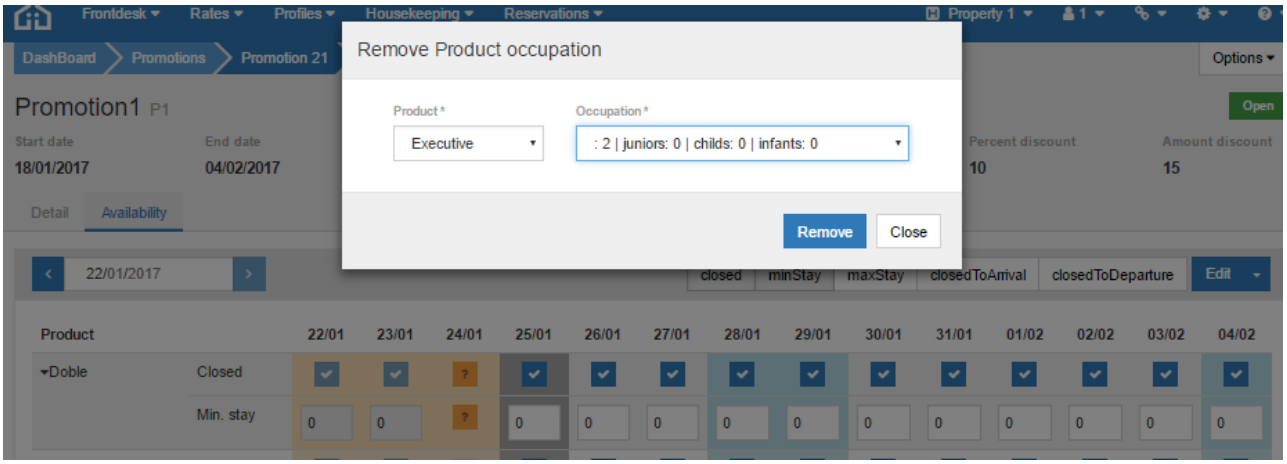


Figura 49: modal para eliminar una productOccupation

7.5. Gestión de Profiles en el frontend

La sección de Profile consta de dos tipos profiles: customers y companies. En cuanto a la vista de ambas serán muy parecidas ya que no dejan de ser profiles pero tendrán diferente información ya que companies posee atributos que customers no y viceversa.

Se configurarán las componentes como el navigation, el breadcrumb y rutas de ambas secciones. La configuración será muy similar y solo habrá que cambiar la nomenclatura de customers por companies y estarán agrupadas como profiles en el header de la pantalla principal del aplicación.

7.5.1. Vista principal de Profiles

Tanto customers como companies tendrán una grid principal. Para cada una de ellas, tendrán columnas diferentes de otras dependiendo de los atributos que estas tengan. Por ejemplo companies tendrá una columna con iata pero customers tendrá name.

También cada una de ellas poseerá filtros no solo por nombre sino por otros parámetros como el documentNumber o el name en el caso de customers y iata o profileGroups en el caso de companies.

En la vista principal podremos ver además del breadcrumb para ir hacia atrás o adelante. Además se podrán crear customers o companies mediante el botón new que utilizara servicios como el modal, el cual nos abrirá una ventana con un formulario el cual será diferente dependiendo si se pretende crear un customer o una Company. En el caso de companies, al a hora de añadir se distinguen dos casos: si es company o travel agent puesto que travel agent tiene un atributo iata que es requerido a la hora de crear y una company no tiene ese atributo.

Para acceder al detalle de una customer o una company, se referenciara el name en las columnas tanto de customer como company en el configFactory de la componente grid.

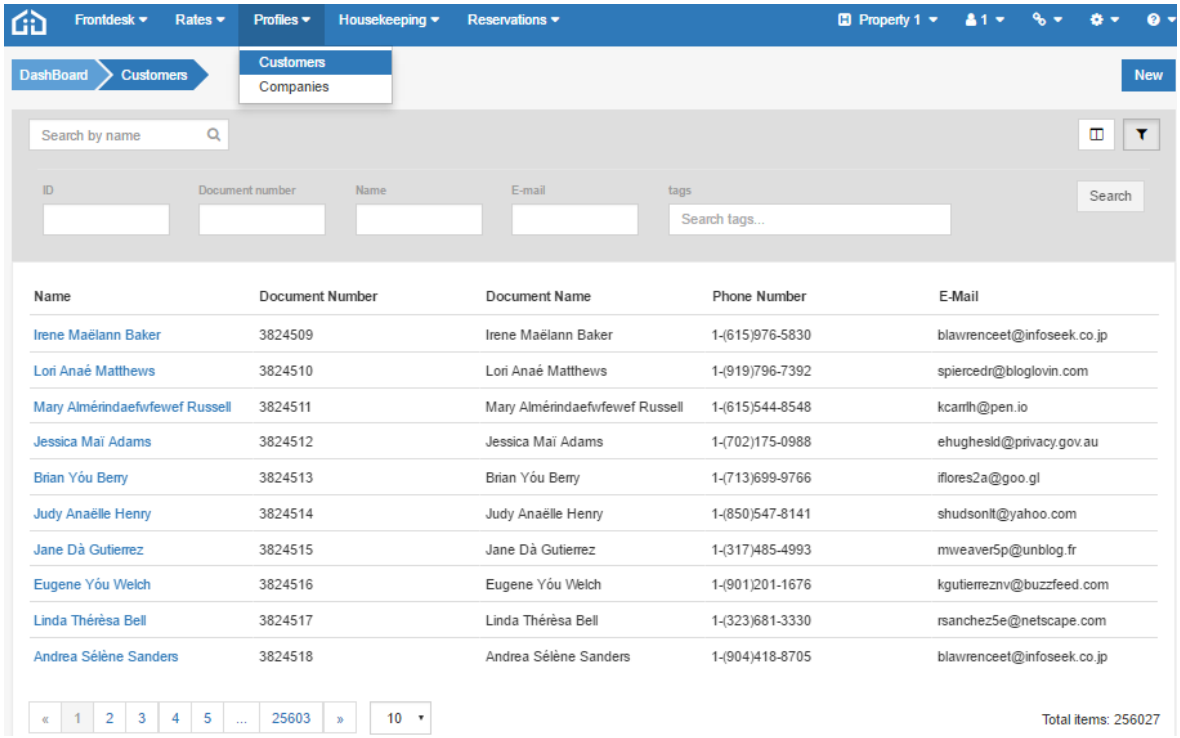


Figura 50: vista principa de la lista de customers

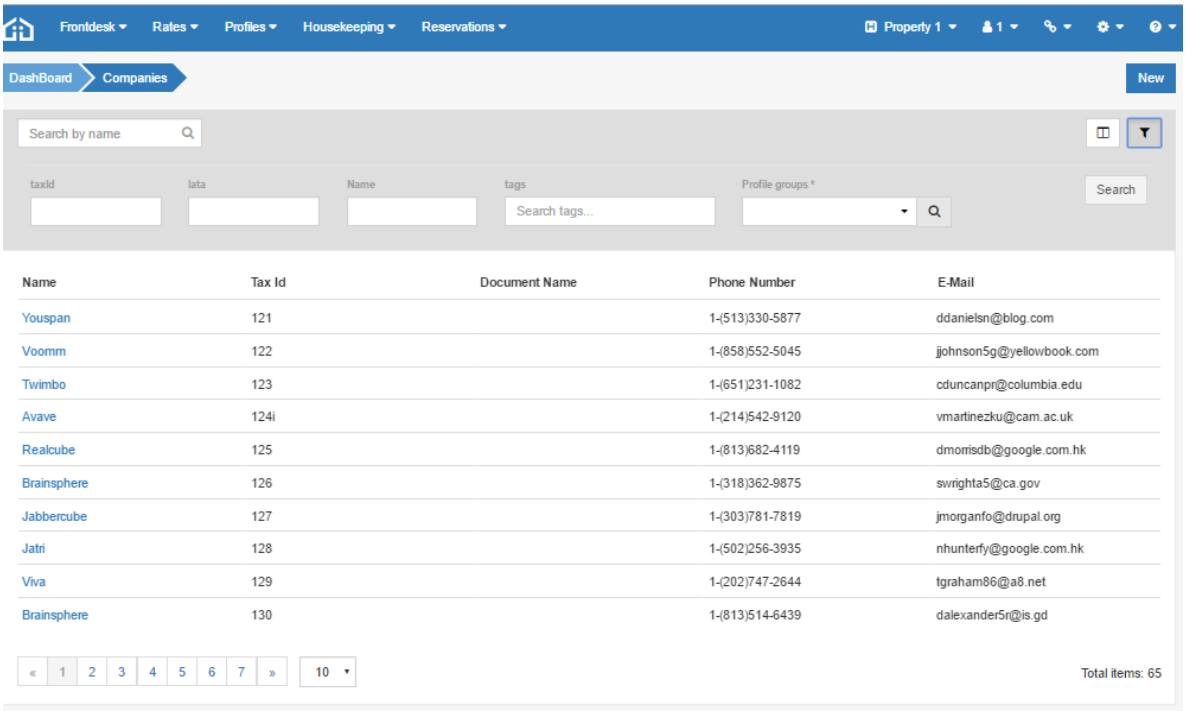


Figura 51: vista principal de la lista de companies

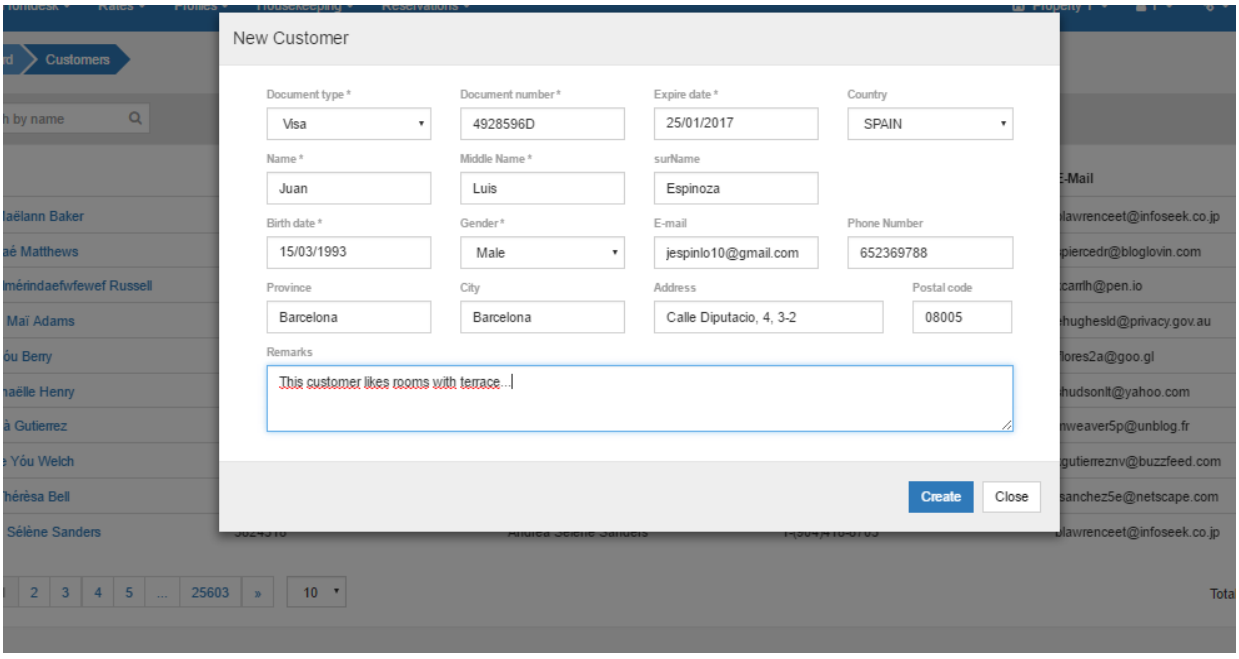


Figura 52: modal de creación de un customer

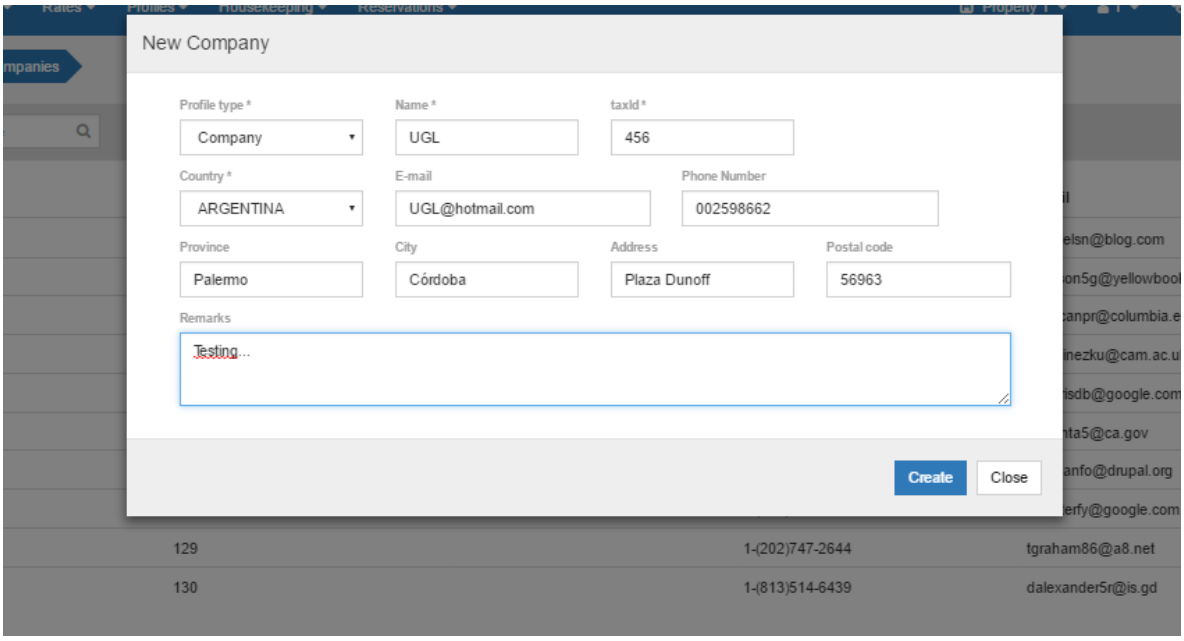


Figura 53: modal de creación de una Company

7.5.2. Vista del detalle de Profiles

La vista del detalle tanto de customers como de companies será muy parcidad y solo cambiarán los atributos que muestra un profile en concreto.

En el detalle solo hay una pestaña que es details, en esta pestaña habrá un formulario separado por secciones mostrando la información del profile seleccionado. En el header del detalle habrá la información más relevante de cada profile y no será modificable exceptuando los tags.

Para obtener dicha información, se tendrá que hacer llamadas API pasando la id del profile como parámetro de petición. Además como en todos los otros detalles estos campos, se podrán editar y guardar. Mediante la comparación de los modelos creados en los controladores de customers o companies se podrá ver si dichos modelos se han modificado o no. De esta manera el botón save podrá estar habilitado o no, en función de si ha habido cambios en el modelo.

Por lo tanto, también se podrá editar un profile, mediante llamadas a la API, en este caso el update, se enviará la id del profile y el modelo con los cambios hechos por el usuario.

Finalmente se podrá eliminar un profile, esto se hará igual que en allotment, mediante el botón de delete que está en el grupo de botones en la parte superior derecha del header. Este botón delete llamará a funciones en el controlador de customers o companies y mediante servicios se enviarán llamadas API enviando la id de profile como parámetro de petición para poder eliminar el profile en concreto.

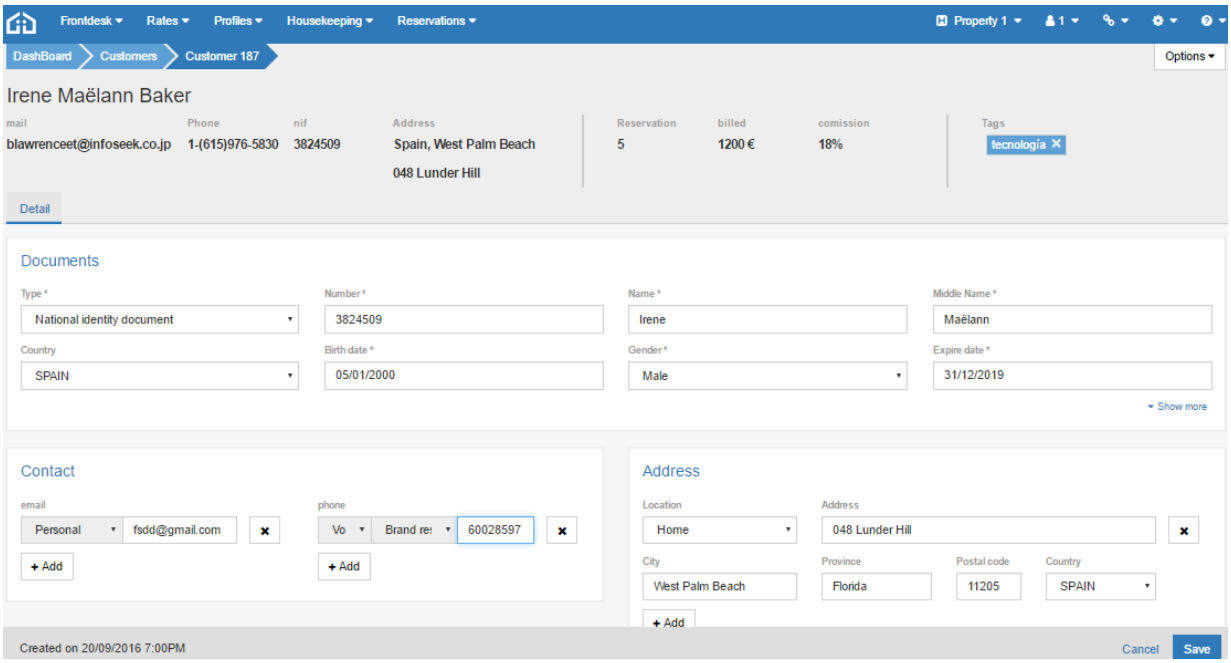


Figura 54: vista del detalle de customer

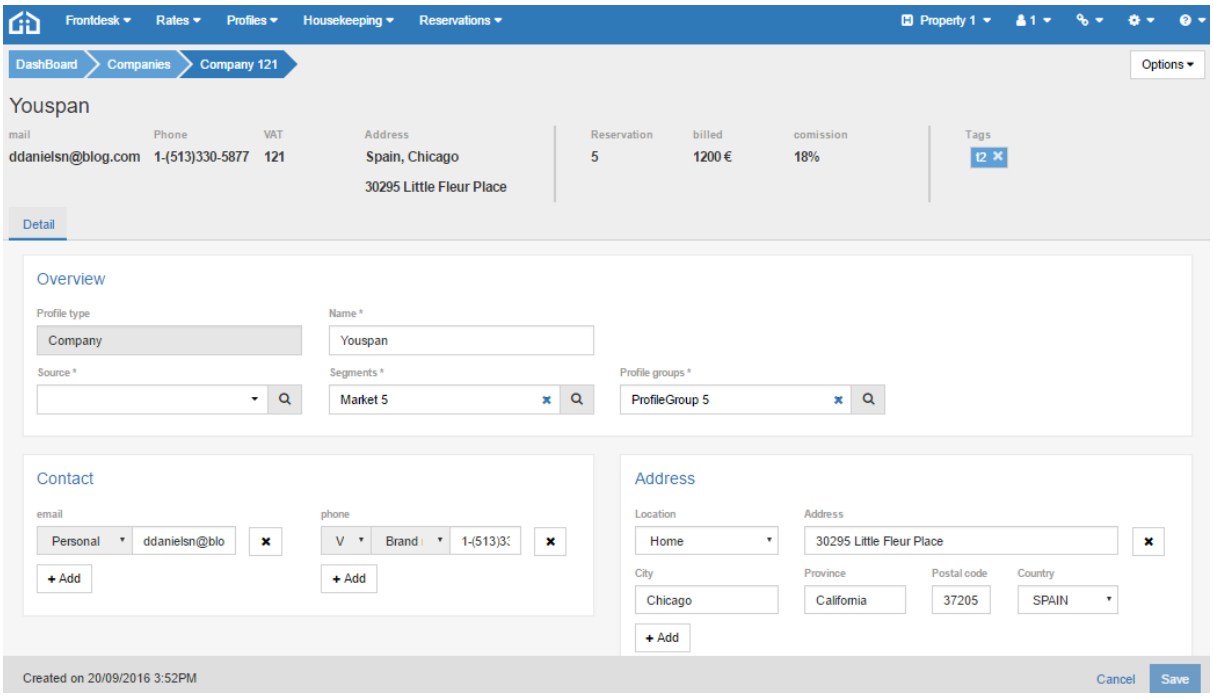


Figura 55: vista del detalle de company

8. Conclusión

Una vez dada mi estadía acabada en la empresa, paso a detallar las conclusiones que puedo observar y valorar ya sea positiva o negativamente.

He participado en el desarrollo de un producto que utiliza un gran número de tecnologías y herramientas modernas. De esta manera, al trabajar con dichas herramientas, he ganado conocimientos de los cuales estoy seguro me ayudarán en el futuro cuando salga al mundo laboral y se requiera el conocimiento con los que he trabajado todos estos meses.

Si bien es cierto que conocía algunos de los lenguajes de programación ya que los aprendí durante la carrera, hay otros con los que no había trabajado nunca, teniendo que aprender a adaptarme a lo que la empresa requería. Además, he aprendido a trabajar en equipo, aprendiendo nuevas metodologías de trabajo y sobretodo aprendiendo de mis propios compañeros.

Con respecto al PMS web, es un producto que aún está en fase de desarrollo pero que va avanzando a buen ritmo y promete tener muy buena acogida en el mundo hotelero. La utilidad y su sencillez de uso, así como de diseño, la hacen ideal para el usuario y para gestionar una gran cantidad de información.

Finalmente, destacar que todo lo que saco de este proyecto es positivo y enriquecedor ya que a mi entender he mejorado los conocimientos que he obtenido durante los años que he estado en la carrera de ingeniería informática.

Bibliografía

La bibliografía y referencias que he consultado para la realización de este trabajo son exclusivamente recursos electrónicos, es decir, la gran mayoría se encuentran en Internet y están en continua actualización. Tenemos dos motivos para escogerlos en lugar del formato tradicional en papel: el primero es que tanto Spring como Angular JS disponen de una magnífica documentación en línea y no sólo eso, sino que también ofrecen tutoriales paso a paso y en otro formato que ellos denominan guías, con lo que no es necesario acceder al formato papel para obtener información. El segundo motivo es que ya sea Spring, Angular JS o BootStrap se encuentran en evolución constante por la gran aceptación que está teniendo entre la comunidad de desarrolladores en general, por lo que cualquier libro que lleve publicado más de seis meses es posible que haga uso de librerías o funciones que se encuentren a día de hoy caducadas (*deprecated*, en inglés), bien porque las hayan sustituido por una versión nueva y mejor o bien porque se hayan dejado de utilizar por cuestiones de seguridad. La mayoría de consultas a webs y tutoriales las hice en el verano-otoño del 2016, ya que el análisis requería muchos conocimientos, una vez adquirido los conocimientos, empecé a programar y durante el desarrollo hice consultas puntuales. Las referencias web que hemos consultado para la elaboración son las siguientes:

En relación a Spring Framework y módulos relacionados:

- https://spring.io/docs
- https://spring.io/guides
- http://www.mkyong.com/tutorials/
- http://www.baeldung.com
- http://www.techferry.com/articles/spring-annotations.html

En relación a Hibernate – ORM:

- http://hibernate.org/orm/
- https://www.tutorialspoint.com/hibernate/orm_overview.htm
- http://www.techferry.com/articles/hibernate-jpa-annotations.html

En relación a los conocimientos generales del Property Management System:

- http://tesipro.com
- https://en.wikipedia.org/wiki/Property_management_system

En relación a JSON:

- http://jsondoc.org/

En relación a HTML5:

- http://www.w3schools.com/html/html5_intro.asp

En relación a CSS3:

- http://www.w3schools.com/css/css3_intro.asp

En relación a JavaScript y librerías relacionadas:

- http://www.w3schools.com/js/

En relación a bootStrap y las librerías usadas:

- http://getbootstrap.com/

En relación a AngularJs y los módulos usados:

- https://docs.angularjs.org
- https://github.com/formly-js/angular-formly-templates-bootstrap
- http://docs.angular-formly.com/

En relación a la API Rest:

- http://www.restapitutorial.com/
- https://ayudawp.com/wordpress-rest-api-que-es-como-funciona/

En relación a la arquitectura del modelo en el frontend:

- https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html

En relación a las metodologías de desarrollo:

- https://www.softeng.es/es-es/empresa/metodologias-de-trabajo/metodologia-scrum.html
- https://www.scrumalliance.org/why-scrum
- https://git-scm.com/book/es/v1/Empezando-Fundamentos-de-Git