Escola Universitària
Politècnica de Mataró

Centre adscrit a:

UNIVERSITAT POLITÈCNICA
DE CATALUNYA

Grado en Ingeniería Informática

SOFTWARE PARA LA GESTIÓN DE SERVICIOS DE INTERNET COMUNITARIO

Memoria

RAUL VILLAFRANCA ARCENEGUI

PONENT: EDUARD DE BRU DE SALA CASTELLS

TARDOR 2017

TecnoCampus
Mataró-Maresme

Dedicatoria

Dedico el proyecto a mis padres José Antonio y María José por haberme animado a cursar el Grado y por su paciencia durante estos años.

Agradecimientos

Me gustaría dar las gracias a Eduard de Bru de Sala por la tutorización del proyecto.

Por último agradecer a todos los docentes del Tecnocampus por sus enseñanzas durante todos estos años, ya que de manera directa o indirecta han hecho posible que este trabajo vea la luz.

Resumen

Este proyecto tiene como objetivo el desarrollo de una aplicación web para la gestión económica de un servicio de Internet Comunitario en una comunidad de propietarios. La aplicación permitirá la gestión de los usuarios del servicio, los costes del servicio y la emisión de recibos. La aplicación se desarrolla en lenguaje Java y haciendo uso de Spring Framework.

Resum

Aquest projecte té com a objectiu el desenvolupament d'una aplicació web per a la gestió econòmica d'un servei d'Internet Comunitari en una comunitat de propietaris. L'aplicació permetrà la gestió dels usuaris del servei, els costos del servei y l'emissió de rebuts. L'aplicació està desenvolupada en llenguatge Java fent ús de Spring Framework.

Abstract

The aim of this project is de development of a web application for the economic management of a Shared Internet service in a community of owners. The application will allow the management of the users, the management of the service cost and the emission of bills. The application is developed in Java language using Spring Framework.

Índice.

Índice de figuras.

Índice de tablas.

Glosario de términos.

API	Application Programming Interface
Bootstrap	Framework para el diseño de sitios y aplicaciones web
Cloud computing	Es un paradigma que permite ofrecer servicios de computación a través de Internet
Coc	Convention Over Configuration
CRUD	Create, read, update and delete
CSS	Cascading Style Sheets
DI	Dependency Injection
Framework	Marco de trabajo que consta de librerías y un flujo de trabajo pensado para facilitar el desarrollo de aplicaciones.
GIT	Sistema de control de versiones
HTML	HyperText Markup Language
IDE	Integrated Development Environment
IntelliJ IDEA	Entorno de desarrollo desarrollado por JetBrains
IoC	Inversion of Control
Java	Lenguaje de programación orientado a objetos
JDBC	Java Database Connectivity
JPA	Java Persistence Api
Maven	Herramienta de software para la gestión y construcción de proyectos Java. La asignación de dependencias se realiza mediante XML.

ORM	Object-Relational mapping
POJO	Plain Old Java Object
Spring Framework	Framework de código abierto para el desarrollo de aplicaciones en plataforma Java.
Thymeleaf	Librería Java que implementa un motor de plantillas HTML5/XML
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	eXtensible Markup Language

1. Objetivos.

1.1. Propósito.

Una comunidad de vecinos desea contratar a un operador de telecomunicaciones el servicio de acceso a internet para posteriormente distribuirlo entre todos los vecinos. El propósito es desarrollar un software que permita llevar la contabilidad y las necesidades de gestión pertinentes.

1.2. Finalidad.

La finalidad del proyecto es el estudio de la casuística de compartir internet en una comunidad de vecinos, y la valoración de las necesidades de gestión y contabilidad que se derivan de ello, así como el desarrollo de una aplicación web que solucione tales necesidades.

1.3. Objeto.

Al finalizar el proyecto se habrá desarrollado una aplicación web que permitirá la gestión de la contabilidad de un servicio de Internet comunitario en una comunidad de vecinos. A la aplicación se accederá mediante login y password, y la información así como las funcionalidades a las que cada usuario puede acceder estarán restringidas según su rol.

1.4. Abastecimiento.

El desarrollo de la aplicación se realiza en lenguaje Java y haciendo uso de Spring Framework, el cual es un Framework muy potente y que actualmente goza de popularidad. Para la interfaz de usuario se ha elegido que sea web, de forma que se pueda acceder a la aplicación desde diferentes dispositivos sin necesidad de ninguna instalación.

2. Estudio previo.

2.1. Uso de Internet en los hogares.

En los últimos años el número de personas que utilizan Internet está aumentando a una gran velocidad. Internet no se limita hoy en día a usos profesionales ni a la gente joven, sino que como podemos comprobar, su uso se está extendiendo a todas las edades y segmentos de la población.

Diferentes opciones de ocio online están apareciendo a una velocidad vertiginosa. Un ejemplo de esto son las redes sociales, las cuales están cambiando la forma en que nos relacionamos con otras personas, familiares o amigos. Otras opciones de ocio en expansión son la prensa digital, y el consumo de contenidos multimedia (youtube, spotify, etc.).

El uso de Internet no se limita al ocio, sino que por el contrario hay otros sectores que también están creciendo. Ejemplos de sectores en expansión incluyen el comercio electrónico y la realización de gestiones con administración en modo online. Podemos decir por tanto que actualmente el acceso a internet permite mejorar la calidad de vida de sus usuarios.

Según el instituto Nacional de Estadística [1], el 81.9% de los hogares del Estado Español disponen de conexión a Internet, y según se puede ver en el siguiente gráfico, la tendencia es que este porcentaje continúe aumentando hasta llegar casi al 100%.

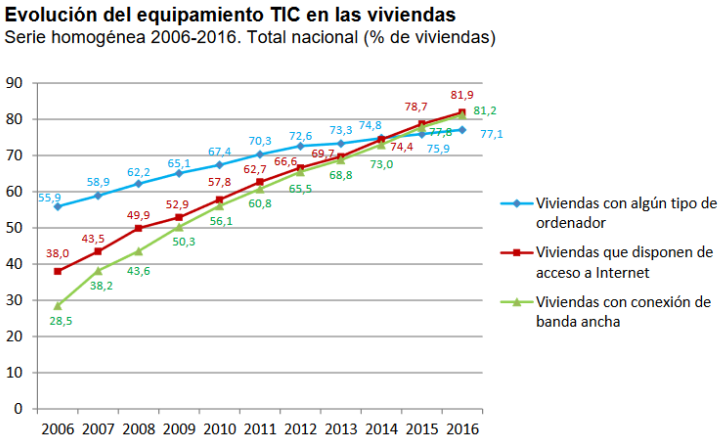

Fig. 1 Evolución del equipamiento TIC en las viviendas

2.2. Concepto de Internet Comunitario.

En los últimos tiempos, se puede observar una tendencia hacia la economía colaborativa y el ahorro. Actualmente por parte de las compañías distribuidoras de Internet se comercializan conexiones para el hogar con un gran ancho de banda, este ancho de banda es habitualmente mucho más amplio del que el usuario medio necesita para un uso cotidiano.

Esto abre la puerta a que comunidades de vecinos contraten una conexión a Internet y la repartan entre los vecinos. Existe un mercado para explotar en este sentido; y ya hay empresas instaladoras que lo están aprovechando, y ofrecen la infraestructura necesaria para compartir Internet de forma comunitaria.

2.3. Aspectos legales.

La legalidad sobre el hecho de compartir una conexión entre vecinos es un asunto que tiempo atrás estuvo envuelto de cierta polémica e indefinición. No obstante, en la actualidad la legislación deja bien claro que compartir una conexión de internet es legal.

Según la legislación vigente desde 2010 [2], una comunidad de propietarios puede ofrecer una conexión de Internet comunitario a los vecinos sin necesidad de inscribirse como operadora de servicios de telecomunicaciones en el registro de la CNMT siempre que se cumplan los siguientes requisitos:

- La comunidad no obtiene un beneficio y cobrará a los vecinos el coste del servicio. En los recibos mensuales de la comunidad ha de estar desglosada la cuota que corresponde al servicio de acceso a Internet.
- En los contratos ha de constar el operador con el que se subscribe el contrato y el cual presta el servicio.
- La red de Wifi (si existiera) no puede estar abierta al público en general.

2.4. Estudio de mercado.

2.4.1 Sondeo de mercado.

Para conocer que opinión tiene el gran público respecto a los servicios de internet comunitario y sus posibilidades de expansión he realizado dos encuestas. Una dirigida a administradores profesionales de fincas, y otra dirigida a residentes de comunidades de vecinos que no tienen contratado un administrador de fincas, y son ellos mismos los que gestionan la contabilidad.

Para la realización de la encuesta se ha usado la herramienta Google Forms, la cual permite enviar un link a los encuestados y que ellos puedan responderlo online de forma confidencial. En total se han obtenido 126 respuestas de particulares y 20 de administradores de fincas.

2.4.2 Encuesta realizada a particulares

1. ¿Actualmente disponen en su edificio de un servicio de Internet comunitario?

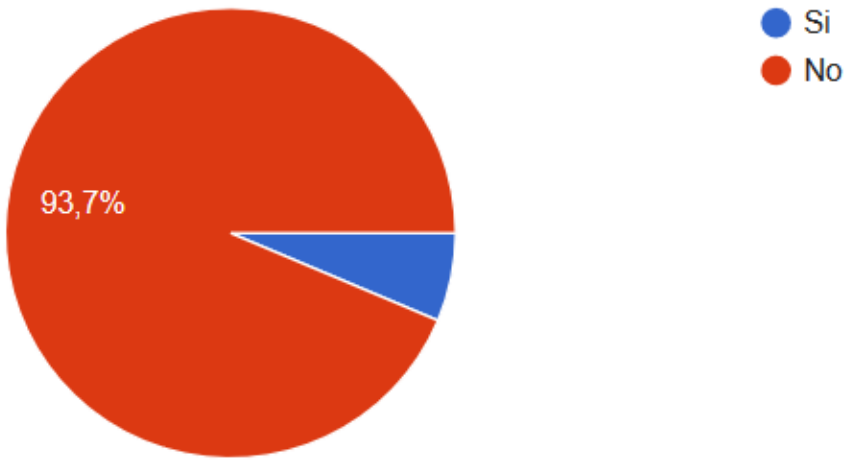

Fig. 2 Respuestas pregunta 1 encuesta a particulares

2. En caso de que no disponer de este servicio, ¿Se han planteado la opción de un servicio de Internet comunitario en el futuro?

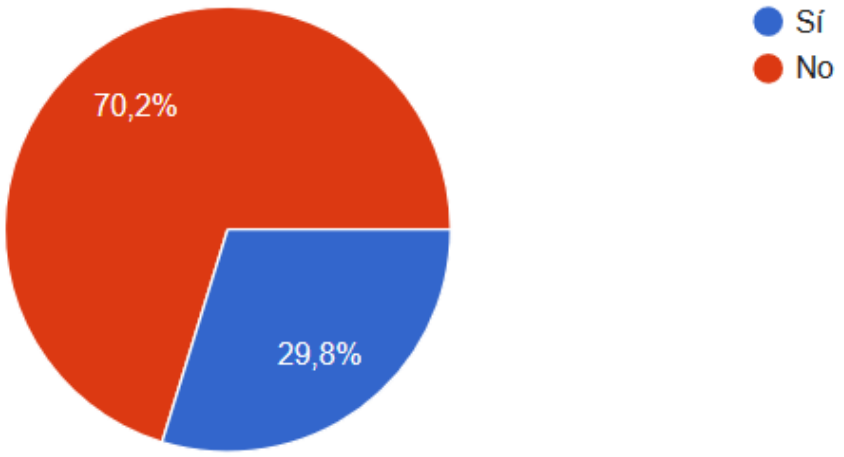

Fig. 3 Respuestas pregunta 2 encuesta a particulares

3. ¿Conoce alguna comunidad de vecinos de su entorno que disponga de Internet comunitario?

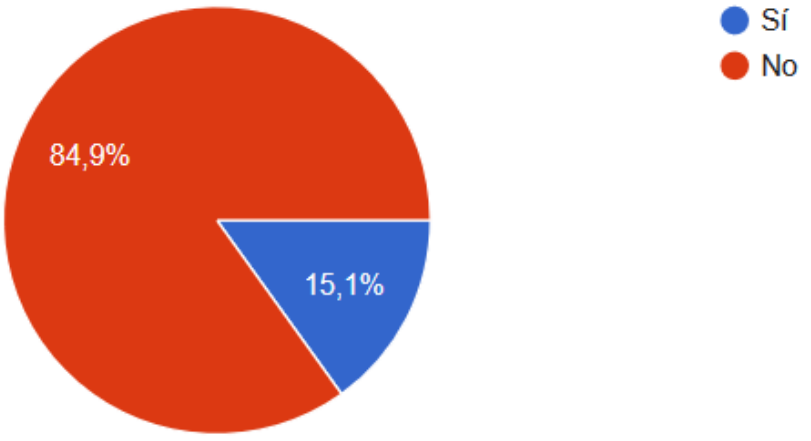

Fig. 4 Respuestas pregunta 3 encuesta a particulares

4. ¿Qué porcentaje de vecinos disponen actualmente de Internet en su hogar?

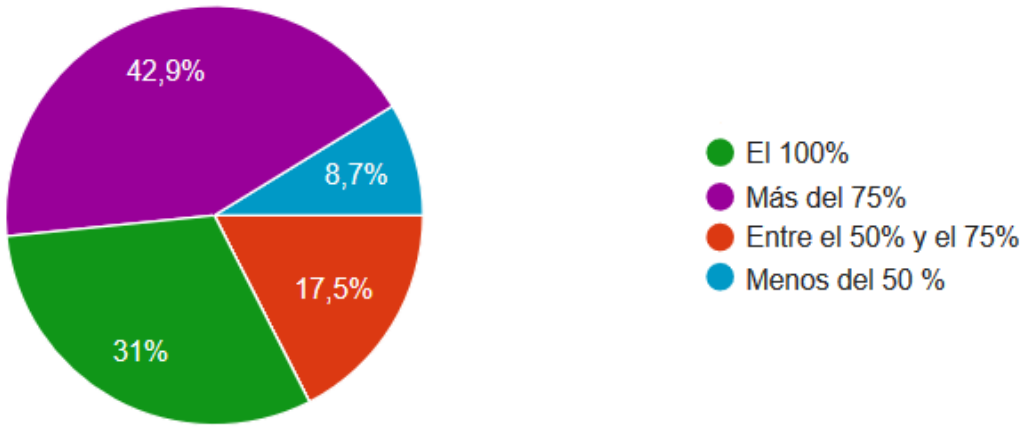

Fig. 5 Respuestas pregunta 4 encuesta a particulares

5. Valore de 1 a 5 el interés que los vecinos podrían tener en contratar un servicio de Internet comunitario.

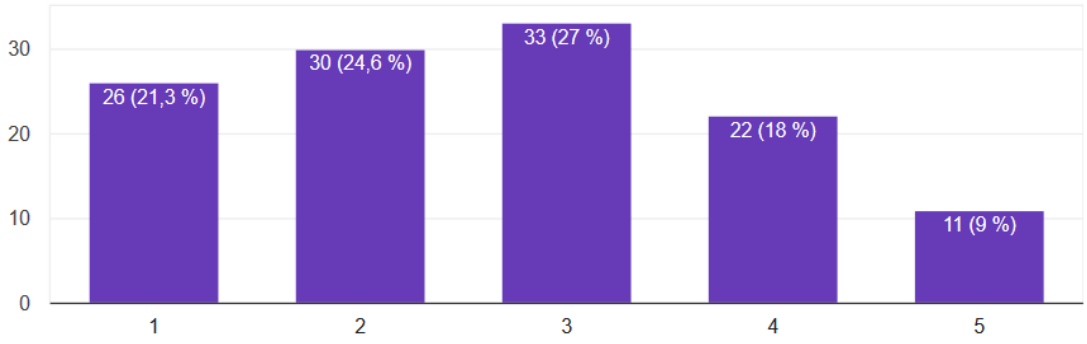

Fig. 6 Respuestas pregunta 5 encuesta a particulares

6. ¿Cómo valoraría disponer de un software que realizara el cálculo de los costes, las cuotas por vecino y el ahorro por vecino que supondría el usar un servicio de Internet Comunitario en su edificio?

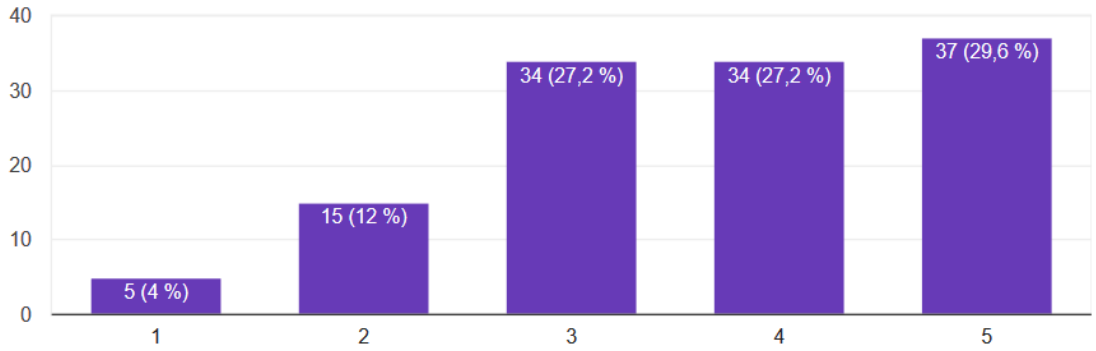

Fig. 7 Respuestas pregunta 6 encuesta a particulares

7. ¿Qué importancia tendría para usted la facilidad de uso a la hora de escoger un software de gestión de Internet comunitario?

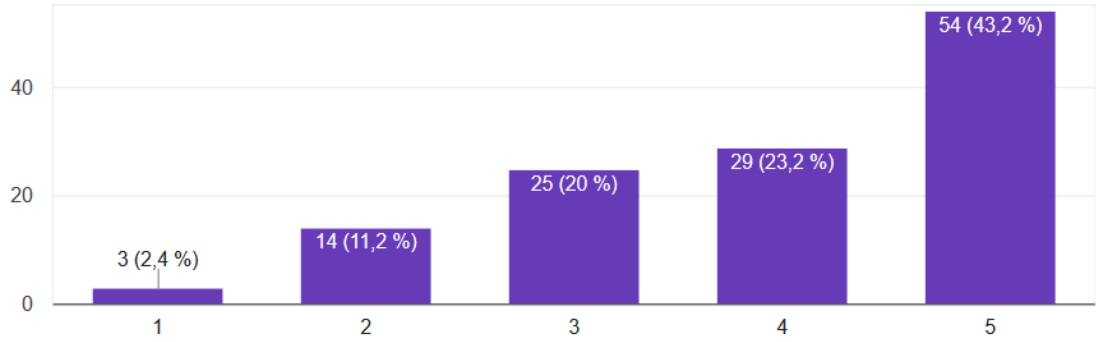

Fig. 8 Respuestas pregunta 7 encuesta a particulares

8. ¿Qué importancia tendría para usted el precio de un software de gestión a la hora de elegir uno?

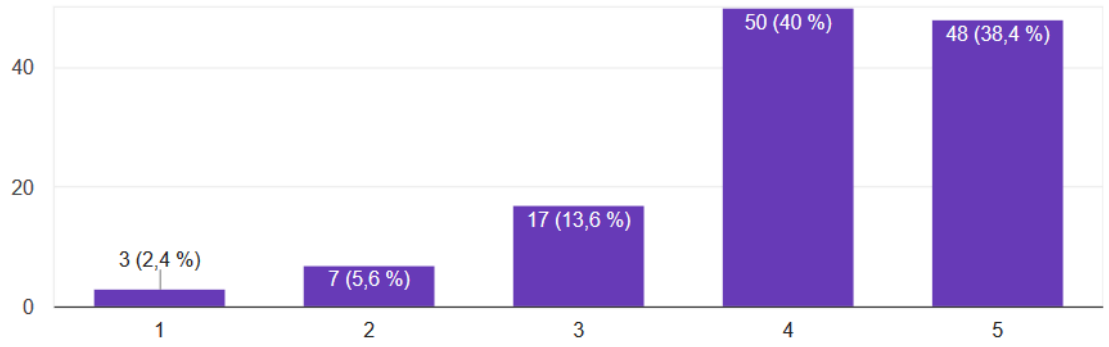

Fig. 9 Respuestas pregunta 8 encuesta a particulares

8. ¿A la hora de elegir un software de gestión de Internet comunitario que importancia daría usted al hecho de que fuera popular y lo usaran otras comunidades de vecinos?

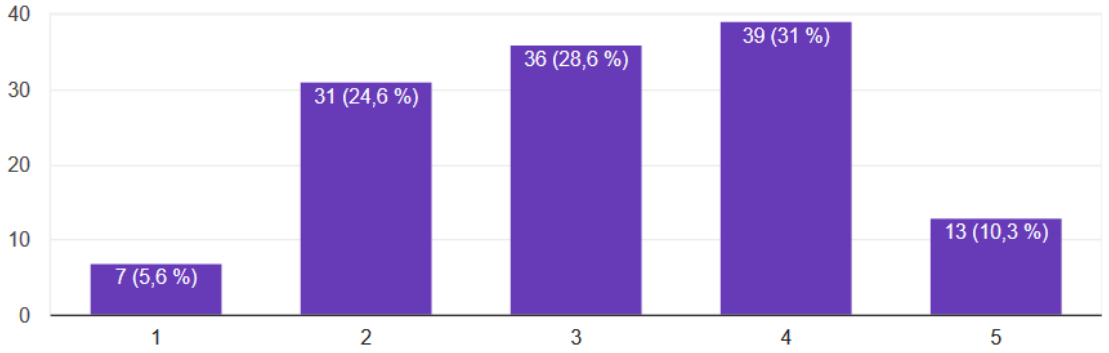

Fig. 10 Respuestas pregunta 9 encuesta a particulares

2.4.3 Encuesta realizada a administradores de fincas

1. ¿Actualmente ofrece algún servicio de Internet comunitario en alguna comunidad de vecinos?

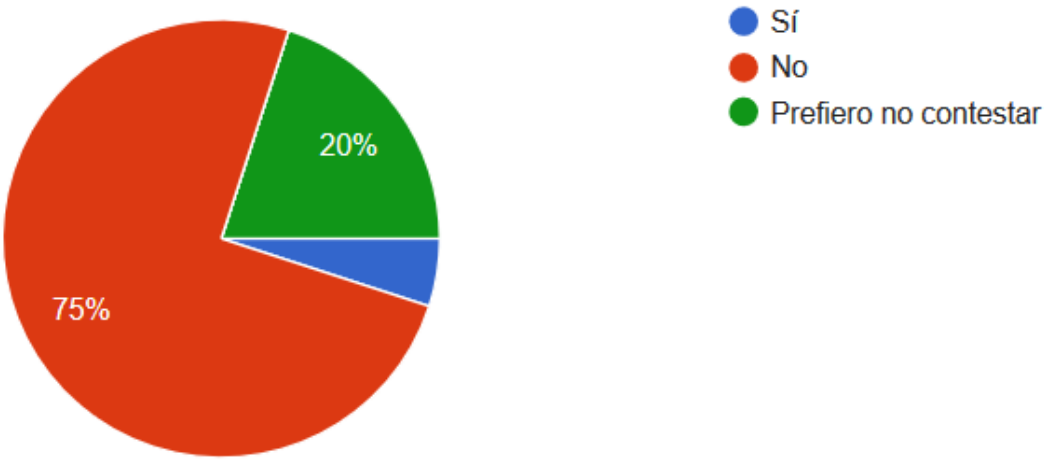

Fig. 11 Respuestas pregunta 1 encuesta a administradores de fincas

2. ¿Se ha planteado en alguna ocasión ofrecer un servicio de este tipo?

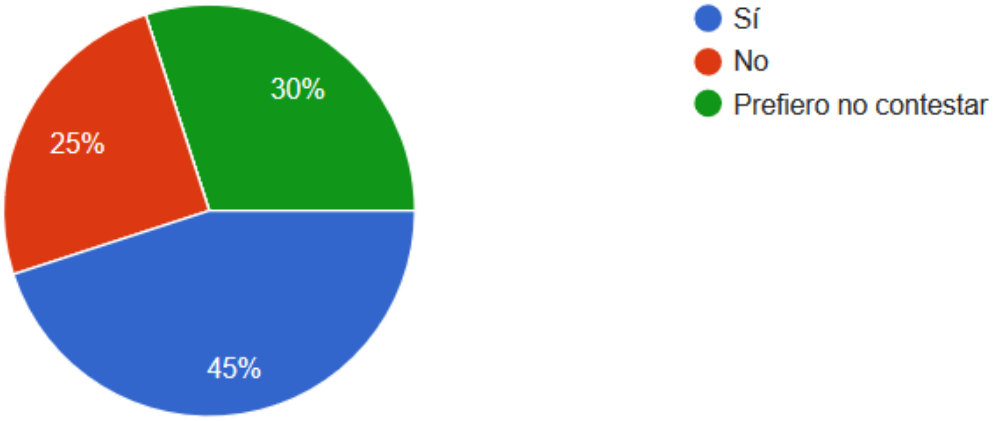

Fig. 12 Respuestas pregunta 2 encuesta a administradores de fincas

3. Valore de 1 a 5 el interés que cree que las comunidades pueden tener acerca de contratar un servicio de Internet comunitario.

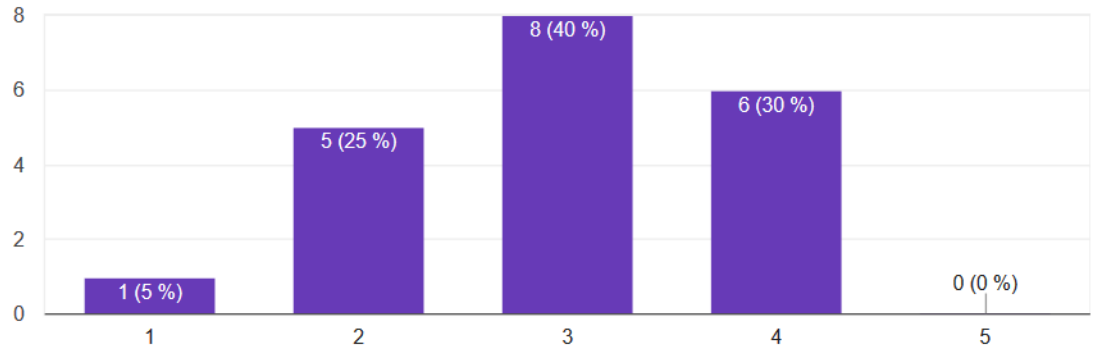

Fig. 13 Respuestas pregunta 3 encuesta a administradores de fincas

4. ¿Cómo valoraría disponer de un software que automáticamente realizara un cálculo de los costes, las cuotas por vecino y el ahorro por vecino que supondría utilizar un servicio de Internet comunitario?

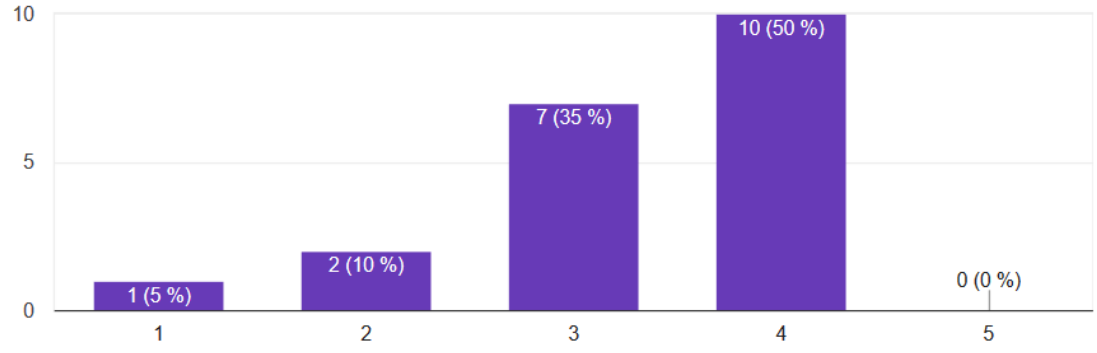

Fig. 14 Respuestas pregunta 4 encuesta a administradores de fincas

5. Valore el interés que puede tener en utilizar un módulo de gestión que se integre a su software de contabilidad y le permita gestionar las cuotas de Internet comunitario de forma automática.

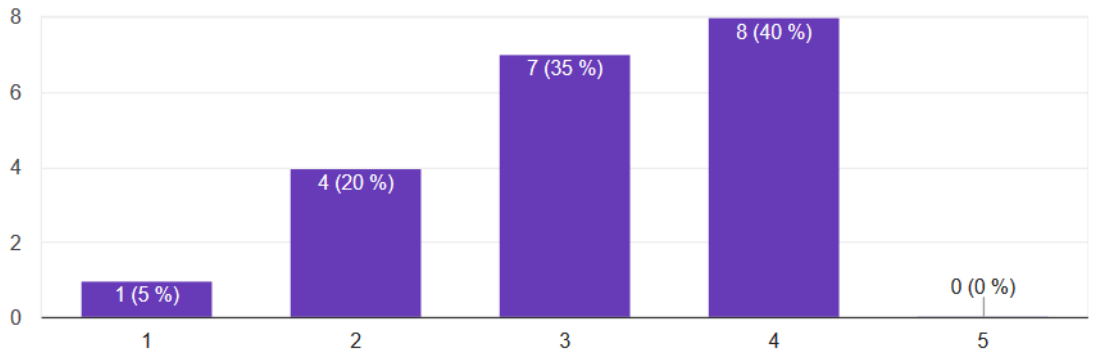

Fig. 15 Respuestas pregunta 5 encuesta a administradores de fincas

6. Valore si estaría dispuesto a adquirir un software de este tipo.

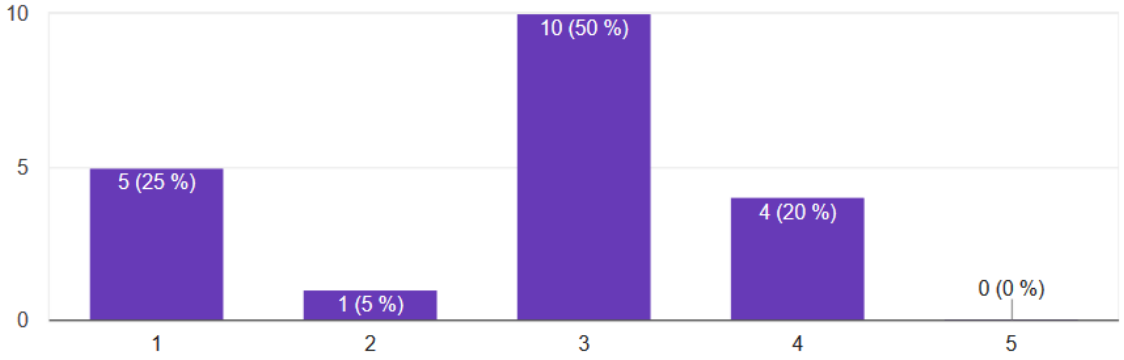

Fig. 16 Respuestas pregunta 6 encuesta a administradores de fincas

2.4.4 Valoración resultados

En ambos casos los resultados obtenidos han sido muy satisfactorios dado que indican que este tipo de servicios despiertan un interés en los usuarios, y mayoritariamente estarían dispuestos a plantearse la contratación de un servicio de internet comunitario. Además, al ser preguntados por la posibilidad de usar un software que les facilite la gestión de los costes y la gestión de los recibos del servicio comunitario también se observa que hay interés en este sentido.

2.5. Estudio de los diferentes software existentes actualmente.

Actualmente no hay ninguna aplicación que se comercialice específicamente para gestionar el cálculo de costes que supone la compartición de Internet entre vecinos.

Hay diferentes aplicaciones de software que están destinadas a la gestión de comunidades, u otras más informales para compartir gastos entre amigos o vecinos, y que podrían ser usadas para realizar un cálculo básico de costes. Pero en todo caso debería ser el propio usuario el que configurara todos los cálculos sin que la aplicación proporcione una ayuda concreta en este sentido.

Desde el punto de vista de amigos que quieren compartir gastos de una forma informal, encontramos diversas apps que están disponibles para usar como aplicación de escritorio o bien desde el teléfono móvil. Ejemplos de estas aplicaciones són Settle Up, Splitwise o Billpin, las cuales, aunque no permiten un cálculo complejo de gastos, tienen una buena usabilidad y permiten a vecinos o compañeros de piso compartir gastos, recibos, etc. Sin embargo, estas aplicaciones no son adecuadas más allá del uso informal, ya que carecen de las posibilidades de configuración y flexibilidad necesarias para realizar cálculos más complejos.

Por otro lado, desde un punto de vista de gestión de comunidades de vecinos encontramos que hay bastantes software diferentes que permiten llevar la contabilidad de comunidades de vecinos. Se trata de programas de tipo más técnico, que necesitan un mayor tiempo de adaptación y aprendizaje por parte del usuario.

Estos programas podemos encontrarlos en formato profesional para administradores de fincas, y de tipo más ‘amateur’ para comunidades que no disponen de administradores, y es el propio presidente de la comunidad el que se ocupa de las cuentas.

Dado que estos programas están diseñados de forma que sea posible realizar configuraciones para ejecutar cálculos complejos, estos programas podrían llegarse a utilizar para calcular los costes de compartición de una conexión a internet en una comunidad, pero sería una tarea tediosa y complicada, ya que estas aplicaciones no aportan ninguna ayuda en este sentido.

Se puede observar que prácticamente todas las compañías desarrolladoras de software de este tipo no se limitan a la clásica aplicación de escritorio sino que por el contrario, ofrecen versiones en la nube a las que poder acceder mediante ordenador, tablet o teléfono móvil.

A continuación enumeraré algunos de los software más populares e interesantes:

- **Gesfincas**: Se trata de un software para la administración de comunidades de propietarios que cuenta con más de 20 años de trayectoria, y uno de los más utilizados. Empezó en sus inicios como una aplicación de escritorio, pero actualmente también ofrece versión web. Usando los portales web de Gesfincas el administrador puede enviar información para ser consultada por los administrados, y también podrá recibir y tratar la información que estos depositen en dichos portales (cambios de domiciliación de recibos, notificaciones, etc.).

- **Fincasapro**: Se trata de un Software de gestión totalmente web que permite llevar la contabilidad de comunidades de propietarios; permite acceder a la aplicación desde cualquier lugar con conexión a Internet y un navegador, sin límite de usuarios ni puestos, y sin tener que instalar ningún programa en el ordenador, tablet o móvil. Además ofrece un servicio de Oficina Virtual, en la que poder mostrar toda la información deseada a los propietarios, los cuales pueden hacer la mayoría de gestiones, sin necesidad de llamar al administrador o acercarse por el despacho. Se puede contratar este software a partir de 5€ /mes.

- **Fincasoft**: Se trata de un programa online para la administración de comunidades de propietarios. Es un programa de uso sencillo pero que dispone de todas las funcionalidades necesarias para realizar la gestión de comunidades. Está orientado principalmente a comunidades que no disponen de administrador profesional y a administradores de fincas que se inician en la profesión. La licencia de software básica es gratuita y permite gestionar hasta 50 propiedades (pisos, garajes, etc).

- **NetFincas:** Sofware para la gestión de fincas que dispone de dos versiones: versión de escritorio y versión Cloud. Con más de 15000 usuarios en activo, permite gestionar todas las tareas que de gestión de comunidades de una manera fácil y cómoda. Además dispone de un módulo web y apps que permiten exportar los datos de las comunidades a Internet, con el fin de que los propietarios puedan conocer el estado de la comunidad desde cualquier dispositivo con conexión a Internet. La versión cloud puede contratarse a partir de 36 €/mes.

3 Spring Framework

3.1 Introducción

Spring es un Framework de código libre para aplicaciones basadas en Java, de forma que las aplicaciones desarrolladas con Spring se pueden ejecutar en cualquier sistema operativo con una máquina virtual de Java. Se trata de un Framework modular, flexible y poco intrusivo.

Spring Framework es una plataforma Java que proporciona una infraestructura de soporte para el desarrollo de aplicaciones Java. Spring se encarga de la infraestructura del proyecto de manera que el desarrollador puede centrarse en el desarrollo de la lógica de negocio de la aplicación.

Se trata de un Framework poco intrusivo y con una estructura modular, de manera que desarrollador puede usar únicamente los módulos necesarios para su aplicación sin necesidad de incluir el resto.

3.2 Resumen de características principales

Spring Framework se compone de diversos módulos, los cuales, cada uno ellos aportan una variedad de servicios que facilitan el desarrollo de aplicaciones. A continuación explicaré brevemente los módulos utilizados en este proyecto:

- **Core Container**: contiene los módulos spring-core y spring-beans, los cuales proporcionan las partes fundamentales del Framework, incluyendo IoC y la Inyección de Dependencias.
- **Convención sobre configuración (Coc):** mediante el uso de Sprin Boot se permite generar proyectos con facilidad, limitando las decisiones de configuración que el desarrollador debe tomar, pero sin que ello suponga una pérdida de libertad.
- **Spring MVC:** contiene la implementación de Spring model-view-controller (MVC) y REST Web Services para aplicaciones web. El Framework MVC de Spring proporciona una separación clara entre la parte del dominio y los formularios web, y se integra con todas las otras características de Spring Framework.

- **Acceso a datos**: permite trabajar con bases de datos relacionales usando Object-Relational mapping y JBDC. Ofrece integración para las más populares API de ORM como JPA y JDO.
- **Spring Security**: se centra en proporcionar autenticación y autorización a aplicaciones Java. Permite ser configurado para cumplir con los requisitos personalizados de cada aplicación.

En los siguientes apartados procederé a explicar con más detalle las principales características y módulos de Spring Framework.

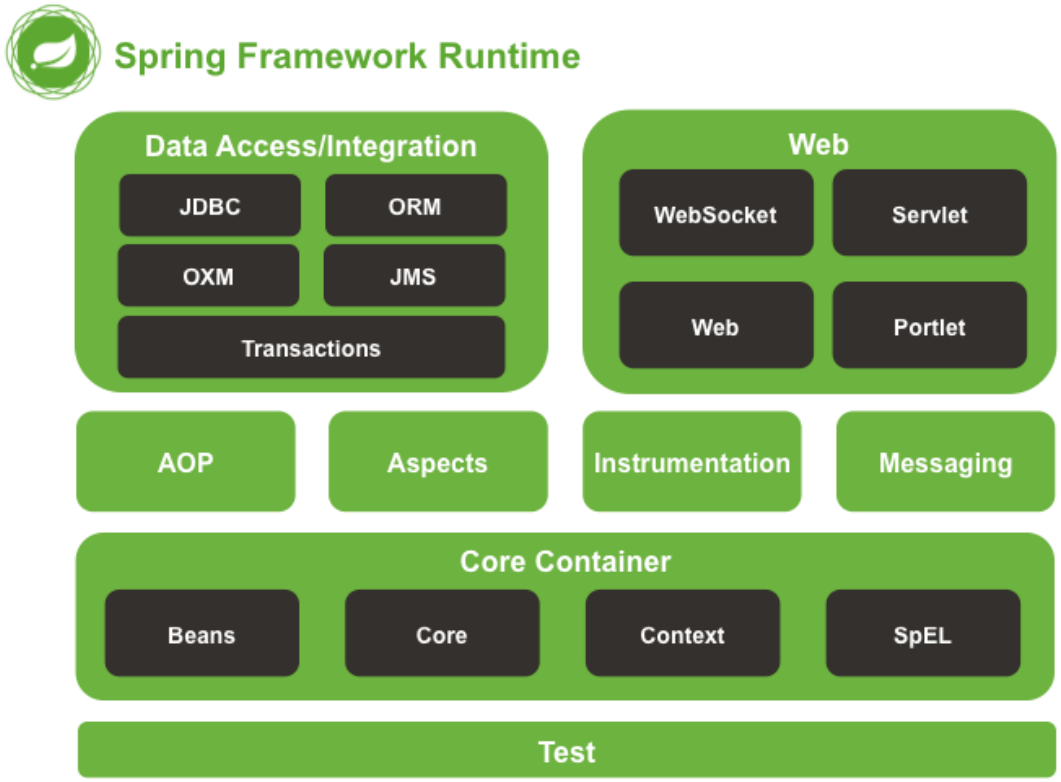

Fig. 17 Resumen de los principales módulos que constituyen Spring Framework

3.3 Spring Core Container

Spring Core Container constituye el núcleo de Sping Framework,y se compone entre otros, de los módulos Spring Core , Spring Beans y Spring Context.

Los módulos Spring Core y Spring Beans proporcionan las funcionalidades fundamentales del Framework, incluyendo la Inversión de Control (IoC) y la Inyección de Dependencias.

3.3.1 Beans

Son Objetos Java simples gestionados por el IoC de Spring, de manera que es el propio contenedor de IoC el que gestiona su ciclo de vida. Spring está diseñado para utilizar como beans a POJOs (Plain Old Java Objects), esto permite simplificar el código y reducir el acoplamiento entre el propio código y las diferentes librerías.

Los beans se pueden configurar mediante XML o bien mediante anotaciones, las cuales se indican en el propio código mediante el símbolo @.

3.3.2 Core Container

El Core Container es uno de los puntos centrales de Spring, se ocupa de crear los objetos, de conectarlos entre sí, de configurarlos, y además controla los ciclos de vida de cada objeto mediante el patrón de Inyección de Dependencias (DI).

El contenedor de Spring se puede personalizar mediante un archivo de configuración XML o mediante una configuración basada en anotaciones. La configuración mediante anotaciones se introdujo en la versión 2.5 de Spring.

En el contenedor de Spring se suelen crear y almacenar los objetos de servicios, y objetos que nos permitan conectarnos a otras partes del sistema como BBDDs, sistemas de colas de mensajes, etc.

3.3.3 Inversión de control

La Inversión de Control, también conocido como Inyección de Dependencias, es un principio de programación en el que el flujo de ejecución de un programa se invierte respecto a los métodos de programación tradicionales, ya que se otorga a un agente externo (en Spring este agente se denomina IoC Container) cierto control del flujo de la aplicación, el cual se encarga de realizar las conexiones necesarias entre las clases dependientes.

En el patrón de Inyección de Dependencias (DI, del inglés Dependency Injection) los componentes declaran sus dependencias, pero no se ocupan de crearlas ni conseguirlas. En el caso de Spring Framework, de la obtención e inyección de las dependencias se ocupa el Spring Container.

Las ventajas del uso de la Inyección de Dependencias es conseguir un código más desacoplado, lo cual facilita la realización de testeos, y mejora la mantenibilidad del código.

3.4 Spring Boot

3.4.1 Introducción

Spring Boot es un proyecto de Spring que busca facilitar la creación de proyectos Spring Framework facilitando la configuración y eliminando la necesidad de archivos de configuración XML.

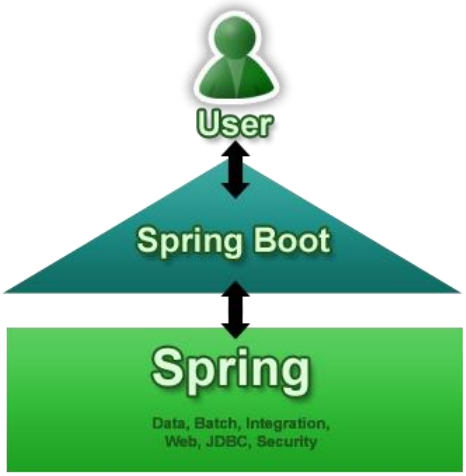

Fig. 18 Situación de Spring Boot dentro del ecosistema de Spring [3]

Un inconveniente de Spring Framework es el tiempo que se pierde realizando configuraciones de los distintos módulos de Spring. Se trata de configuraciones que a menudo son repetitivas, y pueden resultar especialmente complicadas de realizar para los programadores con poca experiencia en Spring. Spring Boot soluciona este inconveniente mediante la introducción del paradigma Convención sobre Configuración o CoC.

El paradigma de programación CoC consiste en minimizar el número de decisiones que han de tomar los desarrolladores, de manera que ahorra tiempo y permite a los desarrolladores dedicarse a la lógica de negocio de las aplicaciones.

Con esto se consigue evitar una repetición de tareas básicas a la hora de realizar las configuraciones más comunes. No obstante Spring Boot no hace perder flexibilidad, ya que en caso de necesitar una configuración específica, se podrán añadir las configuraciones necesarias sin ningún tipo de limitación.

3.4.2 Ventajas

Spring Boot permite crear aplicaciones independientes basadas en Spring, de manera rápida, y listas para ejecutarse en entornos de producción. Incorpora librerías propias y de terceros que permiten empezar a trabajar con un mínimo esfuerzo. La mayor parte de aplicaciones de Spring Boot necesitan muy poca configuración.

Funciones que aporta Spring Boot:

- Permite crear aplicaciones Spring independientes.
- Incorpora Tomcat incrustado: al realizar un deploy del proyecto se arranca un servidor Tomcat automáticamente. Esto nos evita la necesidad de utilizar un servidor Tomcat externo y nos facilita el despliegue de aplicaciones en servidores Cloud .
- Proporciona un archivo POM por defecto para facilitar la configuración de Maven.
- Siempre que es posible configura Spring automáticamente.
- No genera absolutamente ningún tipo de código extra.
- No requiere ningún tipo de configuración mediante archivos XML.

```xml
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.6.RELEASE</version>
</parent>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
</dependencies>
```

Fig. 19 Ejemplo de fichero POM creado por Spring Boot

```java
package hello;

import org.springframework.boot.*;
import org.springframework.boot.autoconfigure.*;
import org.springframework.stereotype.*;
import org.springframework.web.bind.annotation.*;

@Controller
@EnableAutoConfiguration
public class SampleController {

    @RequestMapping("/")
    @ResponseBody
    String home() {
        return "Hello World!";
    }

    public static void main(String[] args) throws Exception {
        SpringApplication.run(SampleController.class, args);
    }
}
```

Fig. 20 Hello World codificado usando Spring Framework

3.5 Inyección de dependencias en Spring

Habitualmente una aplicación no consiste de un único objeto (o bean si hablamos de Spring). Incluso las aplicaciones más simples tienen objetos que interaccionan entre sí para presentar lo que el usuario percibe como una aplicación coherente. En este apartado se explica cómo a partir de unas beans que en principio están definidas sin estar relacionadas las unas con las otras, podemos crear una aplicación donde los objetos colaboran entre sí.

La inyección de dependencias trata de un proceso por el cual los objetos definen sus dependencias (es decir los objetos con los que trabaja) solo mediante argumentos en un constructor, argumentos en un patrón factoría, o propiedades que se añaden a una instancia de un objeto después de ser construido por un patrón factoría. Posteriormente el contenedor inyecta las dependencias cuando crea el bean.

Este proceso es fundamentalmente el inverso, de ahí el nombre Inversión de Control (IoC), del propio bean controlando la instanciación o localización de sus dependencias mediante la construcción directa de clases.

3.5.1 Ventajas de la inyección de dependencias

Usando el principio de Inyección de dependencias se consigue un código más limpio, y un menor acoplamiento ya que los objetos están provistos de sus dependencias. Además las clases son más fáciles de testear, especialmente cuando las dependencias están en interfaces o clases abstractas, lo cual permite realizar implementaciones tipo mock para ser usadas en los test.

3.6 Formas de inyección de dependencias

En este apartado se detalla cómo se realiza la inyección de dependencias, y las diferentes formas de inyectar dependencias. Primero se muestra un código en el que no se usa inyección de dependencias, posteriormente se muestra como será ese mismo código aplicando la inyección de dependencias.

3.6.1 Código sin inyección de dependencias

A continuación se muestra un ejemplo de código sin inyección de dependencias:

```
pubic class NotUsingDependencyInjectionExample {
    private ServiceA serviceA;
    private ServiceB serviceB;

    public NotUsingDependencyInjectionExample(){
        serviceA = new ServiceA();
        serviceB = new ServiceB();
    }

}
```

Fig. 21 Ejemplo de código sin inyección de dependencias

En este ejemplo tenemos una clase llamada NotUsingDependencyInjectionExample en la cual la lógica de negocio se ha movido a dos servicios: Uno llamado ServiceA y otro ServiceB.

De esta forma, cada vez que se crea un objeto NotUsingDependencyInjectionExample, en el controlador de esta misma clase, se necesitan instancias de ServiceA y ServiceB, y estas instancias se obtienen mediante la realización de un new para cada uno de los dos servicios dentro del controlador.

Esto produce un acoplamiento entre el controlador y los dos servicios, es decir, no podríamos crear una instancia de NotUsingDependencyInjectionExample sin disponer de una instancia de ServiceA y ServiceB.

Este acoplamiento dificulta la realización de pruebas y test, ya que en caso de desconocer el contenido de los servicios (por ejemplo si se conectan a una base de datos, y su contenido es cambiante), esto dificulta la realización de pruebas unitarias sobre el controlador de la clase NotUsingDependencyInjectionExample.

3.6.2 Código usando inyección de dependencias

Según lo indicado anteriormente, la inyección de dependencias es un patrón en el que los objetos dependientes se inyectan en la clase. Spring framework se ocupa de realizar esta inyección, pero el desarrollador es quien ha de codificar las clases para que Spring pueda realizar la inyección.

Básicamente hay 2 tipos de inyección: una basada en constructor, y otra basada en setters. También hay una tercer tipo llamado field injection, pero su uso se desaconseja por estar considerado una mala práctica.

3.6.2.1 Inyección de dependencias mediante constructor

El equipo de Spring recomienda como norma general el uso de inyección de dependencias basada en constructor, tal y como se refleja en las documentaciones de Spring a partir de la versión 4.x. [4]

La inyección de dependencias basada en constructor permite asegurar que al instanciar un objeto, sus dependencias se inyectarán y por tanto no serán null; se garantiza además que las instancias en el momento de ser instanciadas estarán en un estado totalmente inicializado y listo para ser usado, cosa que por el contrario en inyecciones de dependencias basadas en setters no se garantiza.

Otra consecuencia del uso de constructor para inyectar dependencias es que se impide la dependencia circular entre dos objetos (en la inyección mediante setter sí es posible la dependencia circular). De hecho esto es positivo ya que las dependencias circulares se deben evitar y generalmente son muestra de un mal diseño. De esta manera se evita el uso de una práctica de este tipo.

```
public class ConstructorDependencyInjectionExample {

    private ServiceA serviceA;
    private ServiceB serviceB;

    @Autowired
    public   ConstructorDependecyInjectionExample(ServiceA serviceA,
    ServiceB serviceB) {
            this.serviceA = serviceA
            this.ServiceB = ServiceB;
    }
}
```

Fig. 22 Inyección de dependencias mediante constructor

3.6.2.2 Inyección de dependencias mediante en setters

Si bien la recomendación que venía reflejada en la documentación de Spring en las versiones 3.x alentaba el uso de la Inyección basada en setters [5], en actuales versiones de Spring se recomienda evitar su uso, en la medida que sea posible, para la inyección en dependencias obligatorias (mandatory).

Se recomienda reservar la inyección de dependencias mediante setter para la inyección de dependencias opcionales. Es decir, en casos en los que la clase es capaz de funcionar aun cuando estas dependencias opcionales no le han sido inyectadas.

```
public class SetterBasedDependencyInjectionExample {

    private ServiceA serviceA;
    private ServiceB serviceB;

    @Autowired
    public void  SetServiceA(ServiceA serviceA) {
        this.serviceA = serviceA;

    }

    @Autowired
     public void  SetServiceB(ServiceB serviceB) {
        this.ServiceB = ServiceB;
    }
}
```

Fig. 23 Inyección de dependencias mediante setter

3.6.2.3 Inyección de dependencias mediante inyección de campo

Se basa en inyectar las dependencias directamente en los atributos de una clase, sin hacer uso de setters ni de constructor, en inglés se denomina Field Injection. Si bien a simple vista puede parecer muy tentador por la simplicidad del código, esta forma de inyección de dependencias está totalmente desaconsejada, y se considera una mala práctica [6].

```
public class FieldInjectionExample {

    @Autowired
    private Service serviceA;

    @Autowired
    private ServiceB serviceB;

}
```

Fig. 24 Inyección de dependencias mediante inyección de campo

Uno de los problemas del DI mediante inyección de campo es que es muy fácil agregar nuevas dependencias. Demasiado fácil, no hay ningún problema en añadir, diez o incluso más dependencias. Cuando se utilizan constructores para DI, en ocasiones, el número de parámetros del constructor se vuelve demasiado alto, esto permite detectar que el diseño del constructor o de la clase son mejorables. Tener demasiadas dependencias generalmente significa que la clase tiene demasiadas responsabilidades. Esto puede provocar una violación de los principios de diseño de responsabilidad única y de la separación de intereses.

Otro inconveniente de DI mediante inyección de campo es que se omite el modo por el cual se inyectan las dependencias. Es decir, cuando se usa la inyección de dependencias, significa que la clase ya no es responsable de administrar sus propias dependencias. La responsabilidad de obtener las dependencias corresponde al propio Spring DI container. Cuando una clase ya no es responsable de obtener sus dependencias, debe comunicarlo claramente usando métodos o constructores. De esta manera queda claro las dependencias que la clase requiere y también si la dependencia es opcional (setters) o obligatoria (constructor).

Por otro lado, indicar que la DI mediante inyección de campo produce un acoplamiento entre las dependencias, y el DI container, contraviniendo así, una de las ideas principales de la Inyección de dependencias, la cual indica que la clase administrada no debe tener ninguna dependencia del contenedor DI utilizado.

En otras palabras, la clase administrada debe ser sólo un POJO simple, que puede ser instanciado de forma independiente, siempre que se le pasen todas las dependencias requeridas. De esta manera se puede instanciar en un test unitario sin iniciar el contenedor DI y probarlo por separado. Cosa que no se puede hacer con DI mediante inyección de campo, ya que queda acoplada con el DI container, y no se puede testear fuera de él.

Por último indicar que con este método de inyección de dependencias, no se indica una forma directa de instanciar una clase con sus requeridas dependencias, esto implica que si se llama al constructor por defecto cuando algunas de las dependencias necesarias no está correctamente instanciada, se producirá un error de NullPointerException.

3.6.2.4 Uso de interfaces en la inyección de dependencias

El uso de interfaces a la hora de realizar la inyección de dependencias es muy recomendable, ya que proporciona una gran flexibilidad y facilita las tareas de testing.

Se trata de definir una instancia de tipo interfaz como dependencia que se va a inyectar en una clase, así se consigue que cualquier objeto que implemente la interfaz se pueda inyectar en esa clase.

A continuación se muestra un ejemplo en donde el objeto que se va a inyectar es de la clase ServiceExample. En la figura se puede ver la creación de la interfaz.

```
public interface ServiceExample {
    Object methodExample();
}
```

Fig. 25 Interfaz ServiceExample

Se crea una clase ServiceExampleImpl la cual implementa la interfaz.

```
@Service
public class ServiceExampleImpl implements ServiceExample {
    @Override
    public Object methodExample(){
        return new Object();
    }
}
```

Fig. 26 Implementación de la Interfaz ServiceExample

Por último inyectamos la dependencia usando el constructor de la clase, nótese que se realiza la inyección de una instancia de tipo interfaz, pero en tiempo de ejecución se instanciará el objeto de la clase que la implementa.

```
@Controller
public class ControllerExample {

    private ServiceDependency serviceDependency;

    @Autowired
    public ControllerExample (ServiceExample serviceExample) {
        this.serviceExample = serviceExample;

    }

    public Object SomeMethod (){
        return this.serviceExample.methodExample();
    }
}
```

Fig. 27 Se inyecta el servicio en el controlador

3.6.2.5 Uso de dependencias en este proyecto

Siguiendo las buenas prácticas recomendadas por el equipo de desarrolladores de Spring, en este proyecto se ha usado la inyección de dependencias mediante constructor para realizar la inyección de los Servicios. Además la inyección de dependencias de los servicios se realiza usando una interfaz, lo que permite mayor flexibilidad.

A continuación se muestran dos fragmentos de códigos: por un lado tenemos la interfaz ProviderService en la que se declaran 2 métodos, y por otro la clase ProviderServiceImp que implementa la interfaz ProviderService.

```
public interface ProviderService {

    Iterable<Provider> listAllProvider();

    Provider save(Provider provider);
```

Fig. 28 Fragmento de código de la interfaz ProviderService

```
@Service
public class ProviderServiceImpl implements ProviderService {
    private ProviderRepository providerRepository;

    @Autowired
    public ProviderServiceImpl(ProviderRepository providerRepository) { this.providerRepository = providerRepository; }

    @Override
    public Iterable<Provider> listAllProvider() { return providerRepository.findAll(); }

    @Override
    public Provider save(Provider Provider) { return providerRepository.save(Provider); }
```

Fig. 29 Fragmento de código donde se implementa la clase ProviderService

Por último tenemos la inyección de una instancia tipo interfaz ProviderService en el controlador ProviderController. Nótese que en tiempo de ejecución se instanciará un objeto de la clase que implementa la interfaz, esto es: un objeto de la clase ProviderServiceImp.

```
@Controller
@EnableGlobalMethodSecurity(securedEnabled = true)
public class ProviderController {
    private ProviderService providerService;
    private Logger log = Logger.getLogger(ProviderController.class);

    @Autowired
    public ProviderController(ProviderService ProviderService) {
        this.providerService = ProviderService;
    }
```

Fig. 30 Inyección de dependencia mediante constructor de ProviderService

3.7 Configuración de dependencias mediante anotaciones

En este apartado se trata el uso de la configuración de las dependencias; hay dos maneras de realizar la configuración de dependencias: mediante un fichero XML o mediante anotaciones. Dado que la configuración XML está cada vez más en desuso, nos centraremos en la configuración mediante anotaciones.

La configuración mediante anotaciones es una manera muy útil e interesante para configurar aplicaciones Spring, ya que no únicamente se facilita el proceso de configuración, sino que además la configuración resultante es clara y fácil de interpretar.

3.7.1 Anotación de Spring beans

Los beans son el núcleo de nuestra aplicación, son los componentes que se cargarán en el contenedor de Spring (aplicationContext), y será el propio Spring el que se ocupe de su carga, unión (wire) y administración.

Los principales componentes de Spring son los 4 siguientes:

- Component
- Service
- Repository
- Controller

A continuación se procede a explicar las características de cada uno de ellos:

1. @Component

Esta anotación es un indicador genérico para Spring beans, e indica que el objeto sobre el cual se anota es un componente gestionado por Spring. Es la anotación más genérica de un componente Spring.

No es una anotación muy usada habitualmente, ya que en vez de ello suele usarse alguna de sus especializaciones: @Repository, @Service y @Controller son especializaciones de @Component para casos concretos (persistencia, servicios y presentación).

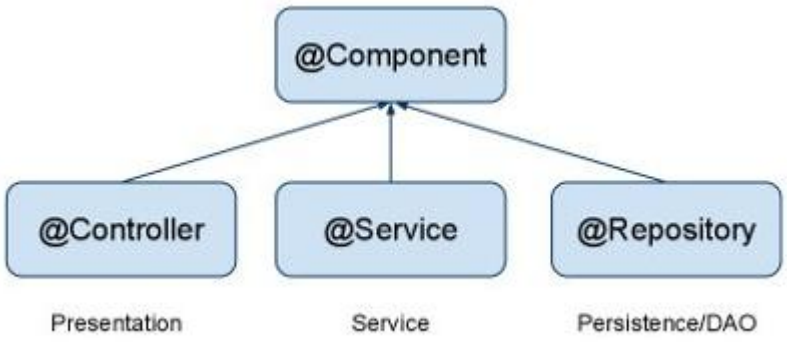

Fig. 31 Anotacion @Component en realacion a @Controller, @Service y @Repository

2.@Repository

Se trata de una especialización de @Component para la indicación de Objetos de Acceso a Datos (DAO), en otras palabras, para indicar que clases se utilizan para persistencia.

3. @Service

Es otra especialización de @Component que se usa para informar a Spring de como cargar las clases de Servicio. Según las recomendaciones de Spring la lógica de negocio de las aplicaciones se debe implementar en las clases de Servicio.

4. @Controller

Se usa para indicar que las clases con la anotación @Controller son usadas como Controlador. Esta anotación es muy utilizada en aplicaciones que usan Spring MVC.

3.7.2 Otras anotaciones

@RequestMapping

Se trata de una de las anotaciones importantes en el desarrollo de aplicaciones web en Spring. Se usa para definir las peticiones URLs que invocan a métodos del controlador. Adicionalmente se puede especificar el método de petición HTTP que se espera (GET, POST). Esta anotación se puede definir tanto para la clase en sí como para métodos concretos.

@PathVariable

Se trata de una anotación que sirve para indicar a Spring que un parámetro de un método está enlazado a un patrón determinado en el path del URI. Se usa en métodos de controlador que estén anotados con @RequestMapping.

3.6.3 Conexión de dependencias

@Autowired

En versiones previas de Spring, las dependencias debían ser especificadas en archivos XML, actualmente las dependencias pueden ser automáticamente detectadas y unidas (wired) por Spring usando la anotación @Autowired. Esto elimina la necesidad de realizar configuraciones XML.

@SpringBootApplication

Esta anotación es núcleo de las aplicaciones que usan Spring Boot. La clase anotada con @SpringBootAplication será la clase principal de la aplicación, en otras palabras ‘el main’ de la aplicación. La anotación @SpringBootAplication sustituye el uso de las anotaciones @Configuration, @ComponentScan y @EnableAutoConfiguration simplificando así el código.

4 Entorno de desarrollo

4.1 Plataforma de desarrollo

4.1.1 IntelliJ IDEA

El IDE que he utilizado para el desarrollo de la aplicación es IntelliJ IDEA, el cual está desarrollado por la empresa JetBrains. El motivo de elegir este IDE es su buena integración con Spring Framework y las facilidades que aporta a la hora del desarrollo. A continuación explico algunas de sus características más interesantes.

Inspección exhaustiva del código: IntelliJ IDEA analiza el código del proyecto, buscando conexiones entre símbolos entre todos los archivos. Usando esta información proporciona ayuda de codificación en profundidad y navegación rápida.

Función autocompletar inteligente: al presionar Ctrl + Shift + Space se obtiene una lista de los símbolos más relevantes en el contexto concreto. Estas y otras acciones de autocompletar están continuamente aprendiendo de las preferencias del usuario, de esta forma en la lista de sugerencias se muestran primero los objetos, métodos de las clases y paquetes más comúnmente usados por el usuario.

Fig. 32 Función autocompletar en IntelliJ

Función autocompletar en clases estáticas: Permite usar métodos estáticos o constantes de manera sencilla. Aun no teniendo las clases importadas, IntelliJ ofrece símbolos pertenecientes a esas clases en la función autocompletar. Además IntelliJ se ocupa de añadir automáticamente todos los imports necesarios.

Fig. 33 Ejemplo autocompletar clases estáticas IntelliJ

Inspección de flujo del código: al ofrecer opciones de autocompletar, IntelliJ IDEA analiza el flujo del código para investigar las características que cada símbolo tendrá en tiempo de ejecución. De esa manera es capaz de afinar aún más las sugerencias e incluso, añadir automáticamente castings en caso de ser necesarios.

Fig. 34 Función autocompletar sugiriendo el uso de de casting

Refractorización avanzada: dado que IntelliJ inspecciona todo el código y conoce la función de cada objeto y clase utilizado, al hacer una refactorización de una clase, método u objeto, utiliza esta información para aplicar la refactorización de manera correcta en todo el código.

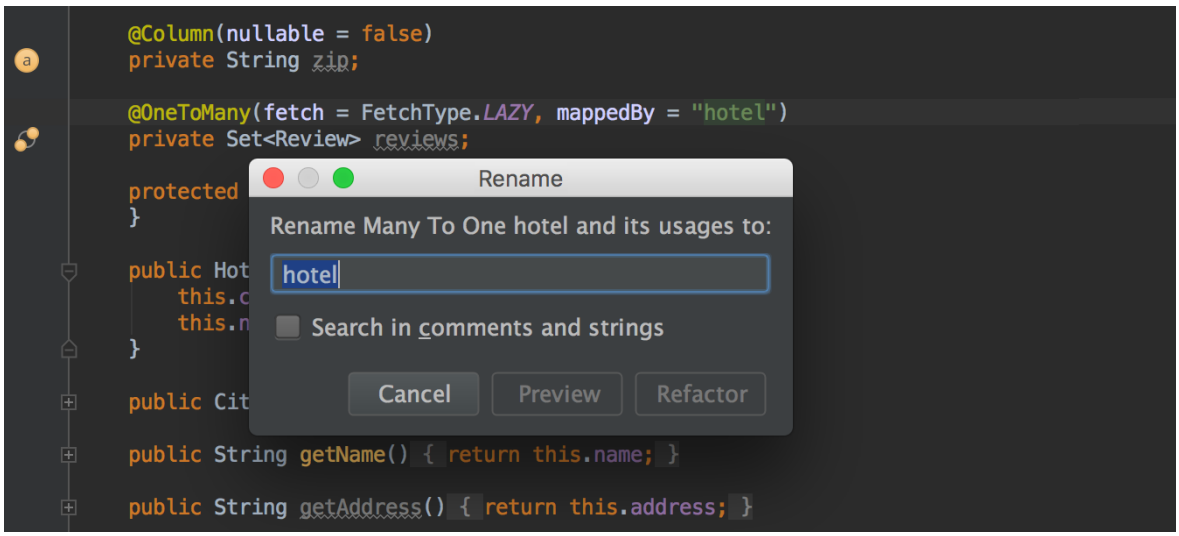

Fig. 35 Refactorizando con IntelliJ

Detección automática de errores: si en cualquier momento IntelliJ detecta que está cometiendo un error te avisa de ello y te ofrece las mejores alternativas para solucionar el error.

Interfaz de usuario basada en el editor: la mayor parte de su tiempo el editor (y el código) es lo único visible en su pantalla, y no es necesario salir del editor para realizar tareas no relacionadas con el editor.

Depurador avanzado: Cuando se usa el depurador de código IntelliJ nos indica el valor de las variables en cada línea y nos muestra el valor de las variables que se modifican sin necesidad de realizar ninguna configuración adicional.

Integración con Control de Versiones: proporciona integración con los principales sistemas de control de versiones: GIT, Mercurial, SVN.

Herramientas de Bases de datos: herramientas para trabajar con MySQL, Oracle, PostgreSQL, Microsoft SQL Server.

Control de dependencias: IntelliJ soporta Maven y Gradle.

Integración con Spring Framework: ofrece una buena inspección de código y asistencia a la codificación usando Spring Framework.

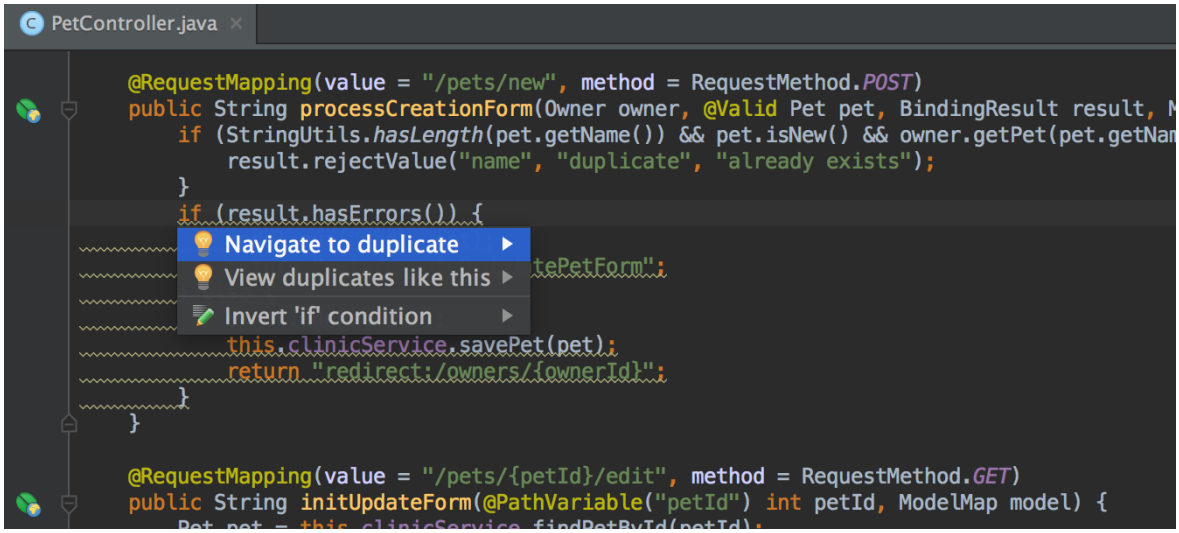

Fig. 36 Navegación por el código de un proyecto Spring

4.1.2 Java 1.8

En el desarrollo de este proyecto utilizo la versión 1.8 de Java, que es la versión más reciente de JDK disponible actualmente. Algunas de las mejoras que aporta esta versión frente a JDK 1.7 son la incorporación de expresiones Lambda y una seguridad mejorada.

4.1.3 Mamp

Dado que para la persistencia de la aplicación hago uso de una base de datos MySQL, se hace necesario el uso de un servidor de MySQL. Me he decantado por el uso de Mamp porque es un servidor ligero y sencillo de configurar.

Se trata de una aplicación desarrollada por la compañía Appsolute, y además del servidor MySQL ofrece un servidor Apache Tomcat.

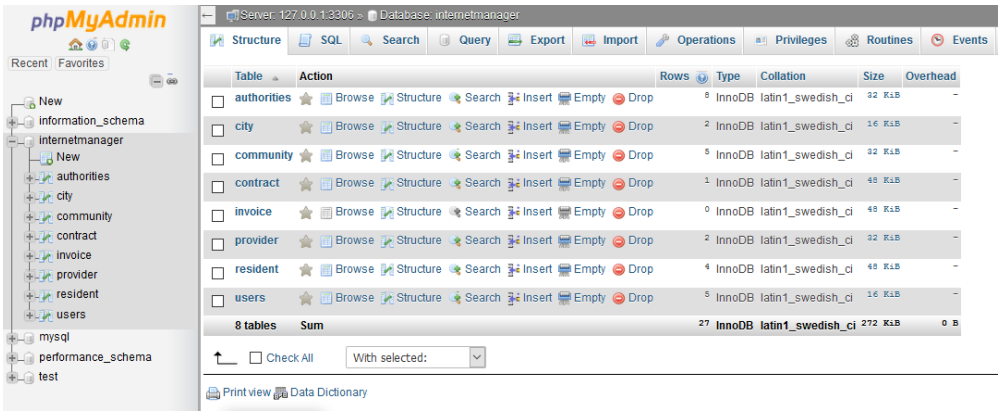

Fig. 37 Imagen del menú phpMyAdmin de Mamp

4.2 Tecnologías usadas en el desarrollo

4.2.1 Spring Boot

Spring Boot es un proyecto de Spring que nos ayuda a iniciar proyectos de Spring de una manera ágil y evitando, en gran medida, la necesidad de configuración. Esto permite mejorar en gran medida la experiencia del desarrollador, ya que puede dedicar su tiempo al desarrollo de la aplicación.

Las características más interesantes de Spring son las siguientes:

Convención sobre configuración: en lugar de escribir manualmente la configuración necesaria y validar si es correcta, Spring Boot nos aporta las configuraciones necesarias para diferentes escenarios. Así se evita al desarrollador de esta tarea repetitiva.

Autoconfiguración: Spring Boot realiza la autoconfiguración del proyecto siempre que sea posible.

Administración de dependencias: No es necesario indicar la versión de las dependencias. Spring Boot se ocupa de administrar las versiones de las dependencias, añadiendo las dependencias estables más recientes en cada caso.

Servidor incrustado: soporta un servidor Tomcat incrustado. Con ello se permite la ejecución del proyecto sin necesidad de desplegar archivos .war.

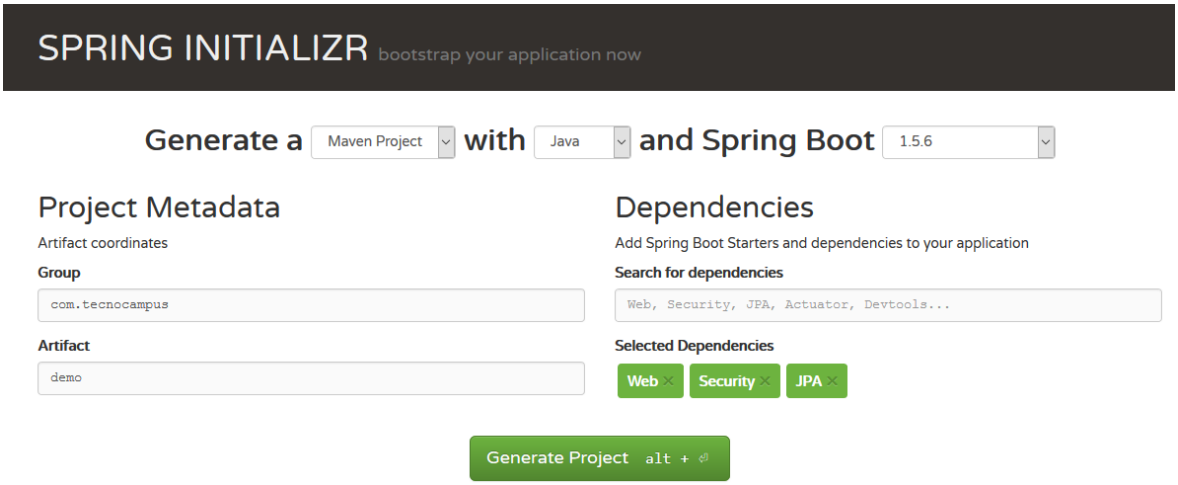

Fig. 38 Herramienta Spring Initializr para iniciar proyectos Spring Boot

4.2.1.1 Buenas prácticas en Spring Boot

Siguiendo la documentación de Spring Boot voy a explicar ciertas prácticas que el equipo de Spring considera 'Best Practices'[7]:

4.2.1.2 Estructura del código:

Si bien Spring Boot no requiere una estructura determinada las siguientes prácticas son recomendables [4]:

- **Las clases han de estar dentro de un package**. Cuando no se especifica un package para una clase, se considera que está en el paquete por defecto ('default

package'), y esto puede dar problemas en aplicaciones que usan las anotaciones @SpringBootApplication, @ComponentScan o @EntityScan

- **Localización del main:** Se recomienda que el main esté en la raíz del proyecto por encima de las demás clases. En esa clase se usará la anotación @SpringBootAplication.

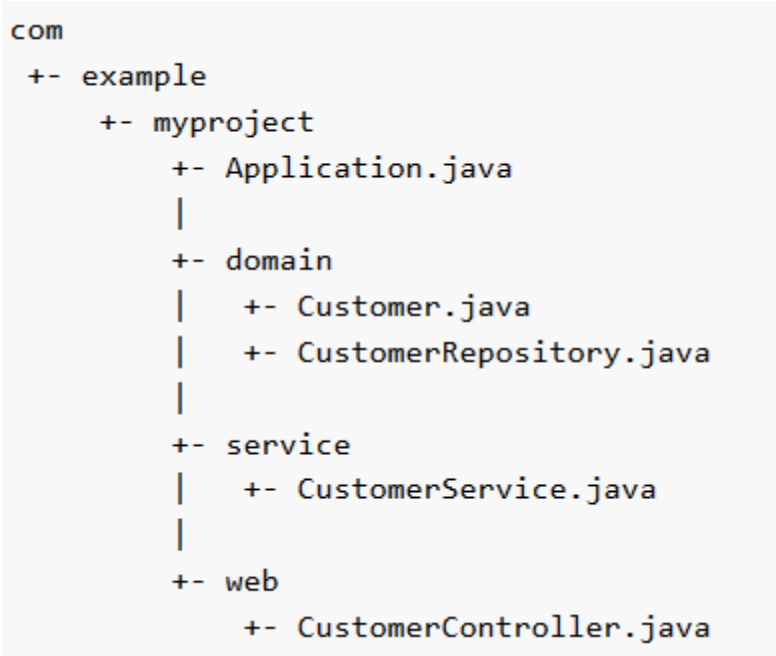

Fig. 39 Estructura típica de un proyecto Spring Boot

4.2.1.3 Configuración

- El equipo de Spring Boot recomienda el uso de la configuración basada en anotaciones Java en detrimento de las configuraciones mediante archivos XML.
- Se recomienda que la clase principal se defina como una clase de configuración principal (@Configuration).
- Habitualmente la clase que se define como main es una buena candidata para ser la clase de configuración principal.

4.2.1.4 Inyección de dependencias

Aunque cualquier técnica estándar de inyección de dependencias se considera correcta, el equipo de Spring recomienda usar @ComponentScan para encontrar las beans, en combinación con una inyección de dependencias mediante constructor, anotando además al constructor con @Autowired.

En los proyectos que siguen la estructura recomendada anteriormente, se puede añadir la notación @ComponentScan sin argurmentos. Y todos los componentes de la aplicación (@Component, @Service, @Repository, @Controller, etc.) se registrarán automáticamente como Spring Beans.

4.2.1.5 @SpringBootApplication

Si se siguen las buenas prácticas mencionadas anteriormente, es muy probable que la clase principal de cada proyecto esté anotada con las anotaciones @Configuration, @EnableConfiguration y @ComponentScan.

Teniendo esto en cuenta, Spring Boot ha creado la anotación @SpringBootApplication la cual es equivalente al uso de las anotaciones @Configuration, @EnableConfiguration y @ComponentScan con sus atributos por defecto.

4.2.3 Maven

Maven es una herramienta de software para la gestión y desarrollo de proyectos Java actualmente desarrollada por Apache Software Foundation y distribuida bajo licencia open source apache.

Maven permite facilitar la gestión de las dependencias de los proyectos, ya que al definirlas en el archivo POM el propio núcleo de Maven es capaz de descargarlas de un repositorio público de Maven que provee acceso a gran cantidad de proyectos Open Source.

Cada proyecto Maven cuenta con un archivo XML llamado pom.xml (Project Object Model) en el cual se describe el proyecto, sus dependencias con otros módulos y componentes externos.

Un detalle muy interesante en la gestión de dependencias de Maven es que también resuelve las dependencias transitivas. Es decir, si a un proyecto le añadimos una dependencia A, que a su vez contiene una dependencia a un proyecto B y éste otra dependencia a un proyecto C, Maven automáticamente descargará las dependencias A, B y C.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>

	<groupId>cat.tecnocampus</groupId>
	<artifactId>internet-manager</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<packaging>jar</packaging>

	<name>internet-manager</name>
	<description>Demo project for Spring Boot</description>

	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>1.5.2.RELEASE</version>
		<relativePath/> <!-- lookup parent from repository -->
	</parent>

	<properties>
		<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
		<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
		<java.version>1.8</java.version>
	</properties>

	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-data-jpa</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-thymeleaf</artifactId>
		</dependency>
```

Fig. 40 Fichero POM del proyecto Internet-Manager

4.2.4 MySQL

MySQL es un sistema de gestión de bases de datos relacional multihilo y multiusuario distribuida por Oracle. Está considerado como el gestor de bases de datos de código abierto más popular del mercado.

Por su velocidad, fiabilidad y facilidad de uso se trata de una base de datos muy utilizada en proyectos web, siendo la base de datos elegida por ejemplo por facebook, Twiter y youtube.

4.2.5 HTML, CSS, Bootstrap, jQuery

Para el diseño de la interfaz web, dado que el coste en tiempo de realizar un diseño HTML y CSS desde cero es muy alto, he optado por trabajar sobre un template predefinido. Hay gran cantidad de empresas y diseñadores que ofrecen templates tanto en versiones open source como versiones personalizadas de pago.

He optado por la utilización del template Admin LTE disponible en la página https://adminlte.io, se trata de un template open source distribuido bajo licencia MIT. Este template hace uso de las librerías Bootstrap3 y jQuery1.11+, y ofrece variedad de componentes y estilos que se pueden utilizar para la interfaz del proyecto.

4.2.6 Thymeleaf

Se trata de una librería de Java que actúa como un motor generador de plantillas web. Thymeleaf es capaz de aplicar un conjunto de transformaciones a las plantillas para mostrar datos y textos producidos por las aplicaciones.

Está especialmente diseñado para funcionar con aplicaciones web diseñadas con XHTML y HML5, pero también puede procesar cualquier archivo XML tanto en web como en aplicaciones independientes.

Thymeleaf dispone de un módulo que se integra perfectamente con Spring MVC y permite sustituir totalmente el uso de JSP (JavaServer Pages). Con Thymeleaf la creación de plantillas dinámicas es más sencilla ya que solo es necesario etiquetar el código de los archivos HTML donde irán los datos dinámicos.

A continuación voy a poner un ejemplo de código de mi proyecto generado por Thymeleaf y su comparación con la plantilla HTML para que se vean las diferencias.

En la plantilla residents.html de mi proyecto, se define un bucle con Thymeleaf mediante la etiqueta **th:each**. Una vez procesado por Thymeleaf se creará una fila por cada residente que se envíe desde el controlador.

```
<tr th:each="resident : ${residents}">
  <td th:text="${resident.id}"><a href="/resident/${resident.id}">Id</a></td>
  <td th:text="${resident.nif}">NIF</td>
  <td th:text="${resident.name}">Resident Name</td>
  <td th:text="${resident.surname}">Resident Surname</td>
  <td th:text="${resident.secondSurname}">Resident Second Surname</td>
  <td th:text="${resident.floor}">Floor</td>
  <td th:text="${resident.door}">Door</td>
  <td th:text="${resident.stairsRoute}">Stairs Route</td>
  <td th:text="${resident.phone}">Phone</td>
  <td th:text="${resident.email}">email</td>
  <td th:if="${resident.active} == true"><span class="badge bg-green" th:text="${resident.active}">Active</span></td>
  <td th:if="${resident.active} == false"><span class="badge bg-red" th:text="${resident.active}"> Not Active</span></td>
```

Fig. 41 Bucle definido con Thymeleaf

Una vez Thymeleaf procesa el código, se crea una fila con sus correspondientes columnas con los datos de cada residente. El fichero que genera y el cual se envía al navegador es HTML5 estándar sin ningún tipo de anotación extra:

```
<tr>
  <td>1</td>
  <td>77777777F</td>
  <td>Jose</td>
  <td>Palomo</td>
  <td>Lopez</td>
  <td>8</td>
  <td>8</td>
  <td>B</td>
  <td>934445525</td>
  <td>president2@gmail.com</td>
  <td><span class="badge bg-green">true</span></td>


  <td><a href="/resident/1">View</a></td>
  <td><a href="/resident/edit/1">Edit</a></td>
</tr>

<tr>
  <td>2</td>
  <td>89447994M</td>
  <td>Antonio</td>
  <td>Padilla</td>
  <td>Garcia</td>
  <td>7</td>
  <td>7</td>
  <td>A</td>
  <td>934423525</td>
  <td>president1@gmail.com</td>
  <td><span class="badge bg-green">true</span></td>
```

Fig. 42 Código HTML de un bucle generado por Thymeleaf

.

4.3 Despliegue de la aplicación

En este capítulo explico como realizar la ejecución de la aplicación en un entorno local y como desplegarla en un servidor Cloud.

4.3.1 Despliegue local

Para el despliegue en local de nuestra aplicación se necesitan las siguientes características de software:

- IntelliJ IDEA
- JDK 1.8
- Un servidor local MySQL

El primer paso que tenemos que realizar para desplegar la aplicación es configurar el entorno de desarrollo. En IntelliJ se configura de la siguiente forma:

- Se agrega una nueva configuración de Run/Debug:

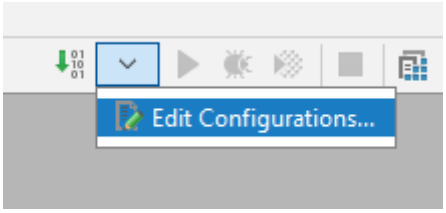

Fig. 43 Menú de nueva configuración para el despliegue

- Se añade una nueva configuración de Application, y se despliega un menú donde hay que seleccionar la clase principal y el path al jre local:

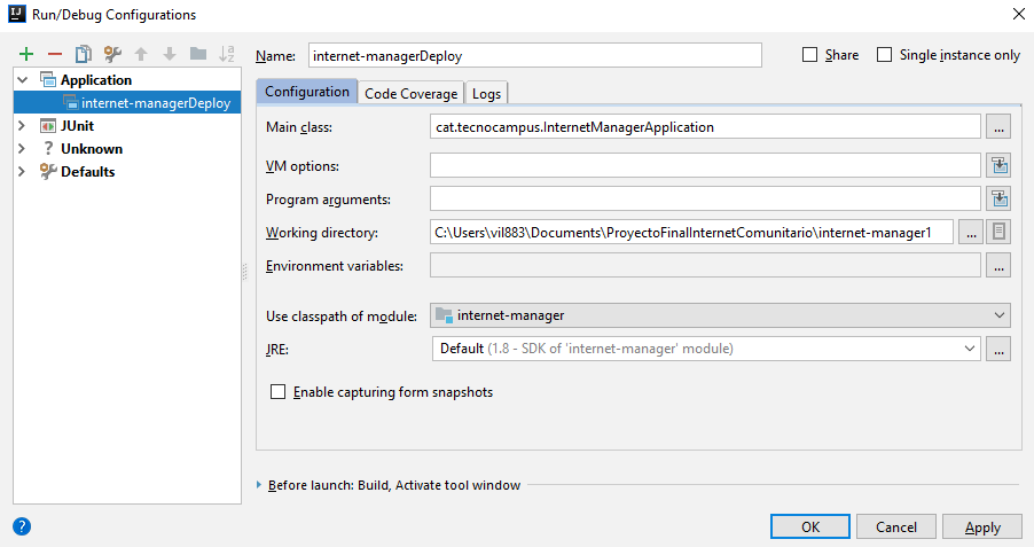

Fig. 44 Configuración para desplegar la aplicación

Dado que en el proyecto se usa MySQL para persistencia, es necesario tener un servidor MySQL ejecutándose en local para que al ejecutar la aplicación funcione correctamente. Cualquier servidor es válido, si bien en mi caso he utilizado Mamp por su sencillez de configuración.

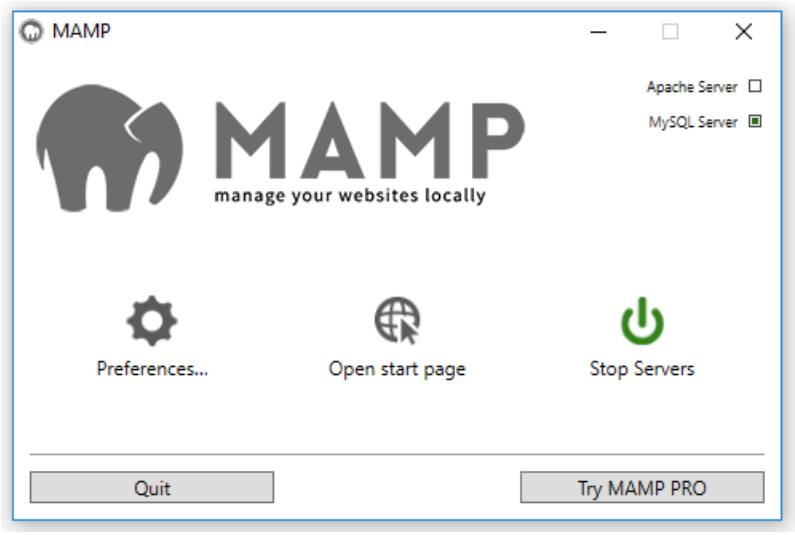

Fig. 45 Aplicación Mamp con un servidor MySQL desplegado en local

Una vez hayamos desplegado la aplicación en local, se podrá acceder a la interfaz de la aplicación mediante el navegador, como se aprecia en la siguiente imagen.

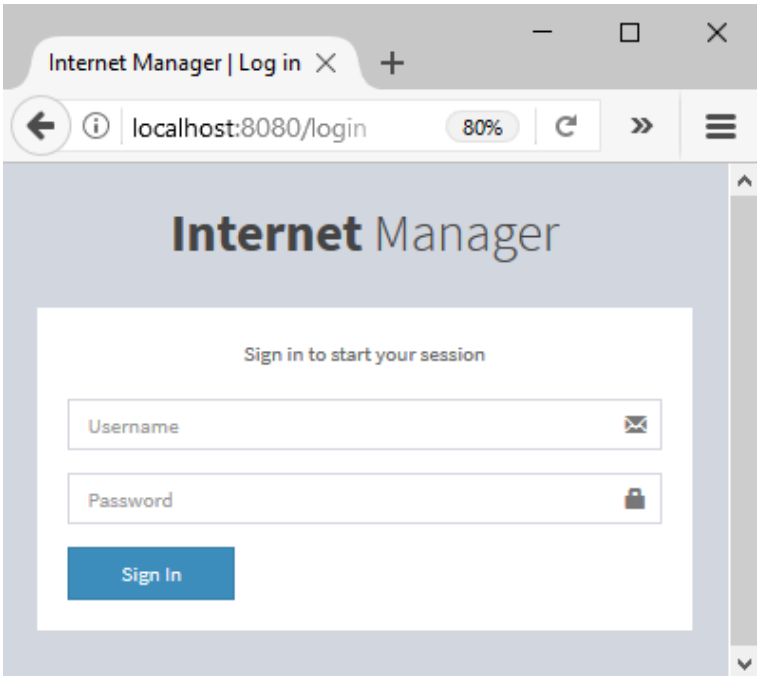

Fig. 46 Imagen del login de la aplicación ejecutándose en local

4.3.2 Despliegue de en servidor Cloud (Heroku)

Heroku es una empresa subsidiaria de Salesforce, que ofrece servicios Cloud Paas (platform-as-a-service). Heroku ofrece las herramientas necesarias para facilitar el despliegue de aplicaciones en la nube.

Una característica muy interesante de Heroku es que si lo deseamos, se puede subir la aplicación desde una rama de un repositorio Git y permitir deploys automáticos. Es dedir que cada vez que hagamos un push de la rama seleccionada en Git automáticamente se desplegará la aplicación con el código actualizado en Heroku.

El despliegue de aplicaciones en Heroku es gratuito si se opta por un servicio básico. Si se opta por un servicio avanzado se ofrecen diferentes planes de precios según los servicios que se necesiten. Entre las limitaciones del servicio gratuito está que las aplicaciones desplegadas en Heroku se pondrán en modo sleep tras 30 minutos de inactividad.

El proceso para desplegar una aplicación en Heroku es muy sencillo e intuitivo, a continuación lo detallo:

- Para dar de alta una aplicación en Heroku tenemos que registrarnos en la página www.heroku.com.
- Una vez registrados con nuestro usuario seleccionamos la opción Create New App y se nos despliega un menú en el que debemos indicar el nombre de la aplicación y la región, y presionar Create App:

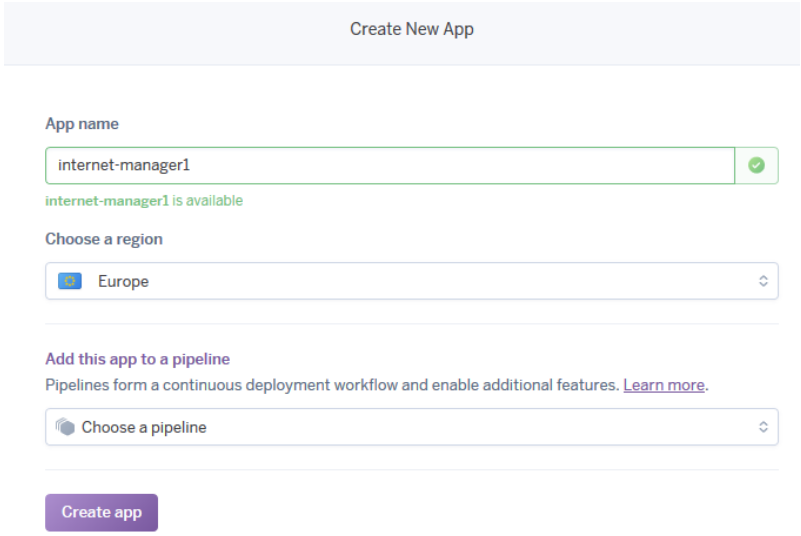

Fig. 47 Menú Create New App

- El siguiente paso es seleccionar el método para subir el código de la aplicación. En mi caso tengo el código en un repositorio de GitHub así que elijo la opción subir desde GitHub, y selecciono el nombre del repositorio donde tengo la aplicación.

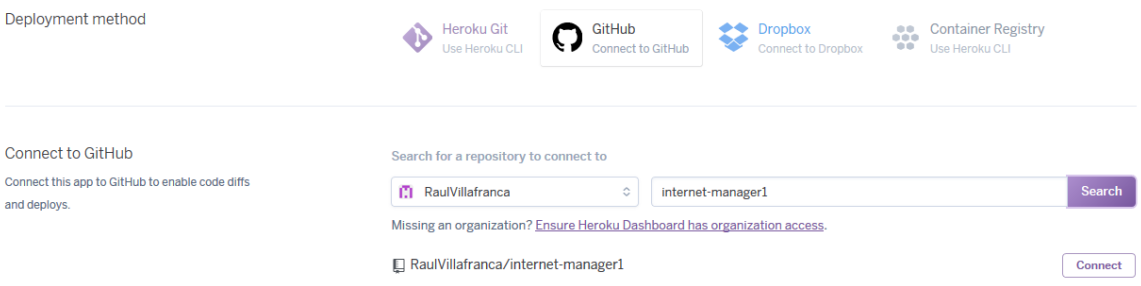

Fig. 48 Subir la aplicación desde repositorio GitHub

- En el siguiente paso se selecciona la rama desde la cual subir el proyecto y seleccionar deploy manual o deploys automáticos y se desplegará la aplicación.

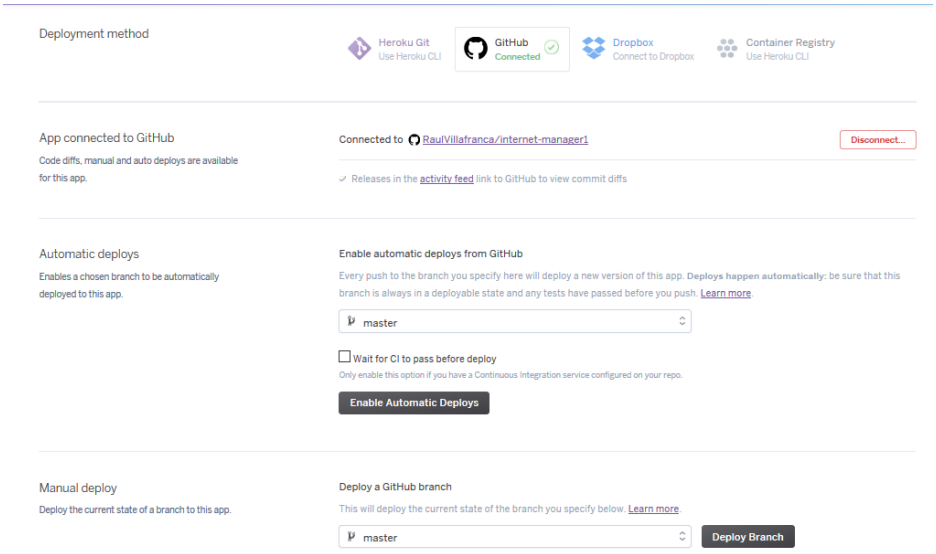

Fig. 49 Selección de rama del proyecto a desplegar y despliegue del proyecto

- Dado que este proyecto necesita de un servidor MySQL, activaremos un add-on que Heroku ofrece, y que proporciona un servidor MySQL a la aplicación.

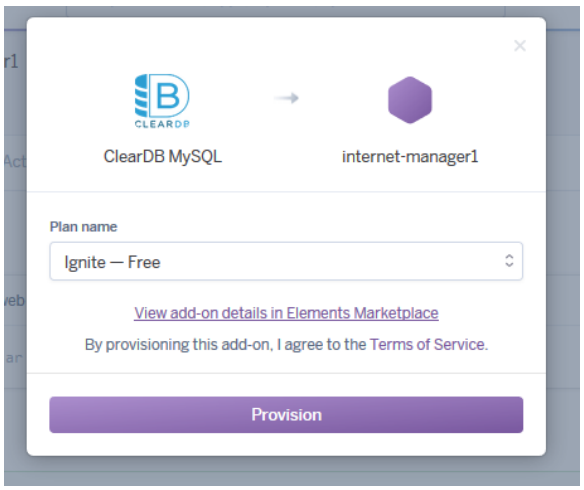

Fig. 50 Add-on servidor MySQL de Heroku

- Dado que tenemos la aplicación desplegada en la nube, podemos acceder a ella desde la URL que nos proporciona Heroku:

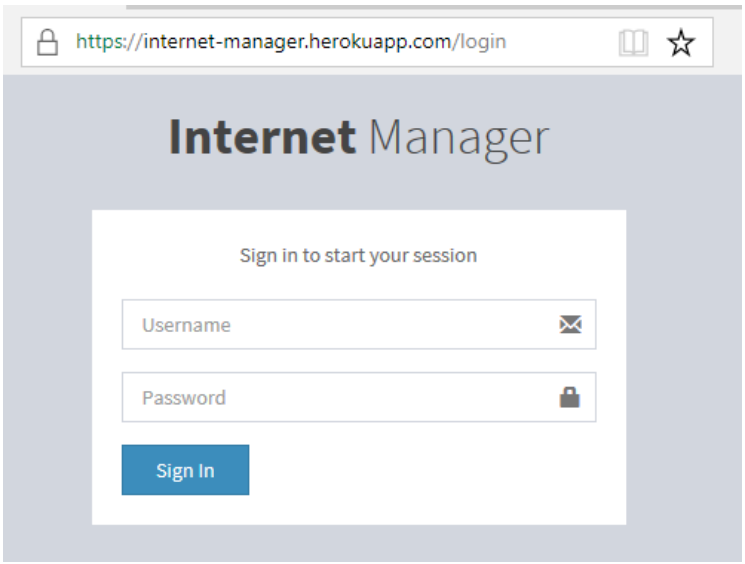

Fig. 51 Acceso a la aplicación desplegada en la nube

5 Análisis de requisitos de la aplicación

5.1 Especificación

Se va a desarrollar una aplicación web para la gestión y administración de los gastos de servicios de internet comunitario en comunidades de vecinos. La aplicación permitirá la administración de los residentes, las comunidades, los contratos con proveedores así como la emisión de facturas. Se establecerán 3 tipos de usuarios: administrador, presidente de comunidad y residente con roles diferenciados.

El desarrollo del prototipo de la aplicación se realizará en inglés, dado que se ha considerado que esto facilitaría su futura traducción a distintos idiomas.

5.1.1. Usuarios

A continuación se describen las características de cada tipología de usuario:

Administrador: este usuario es el administrador total de la aplicación. Únicamente se usará para la administración de la aplicación. Este usuario lo usará el administrador de fincas (si lo hubiere).

Presidente: este usuario es el presidente de la comunidad de copropietarios, o en su defecto la persona responsable del Servicio de Internet Comunitario en la comunidad de copropietarios. A su vez también es un usuario de Internet Comunitario. Puede haber más de un presidente por comunidad.

Residente: Por cada vivienda dada de alta en el servicio de Internet Comunitario hay una persona responsable de la vivienda. A esta persona se le otorga un usuario de la aplicación de tipo residente. Es decir, por cada vivienda dada de alta en el servicio hay un único usuario residente.

Dado que la aplicación es únicamente para la gestión del servicio de Internet Comunitario de comunidades de vecinos, la consigna de uso de la aplicación es dar de alta únicamente a los residentes que son usuarios del servicio de Internet Comunitario. Así si en una comunidad de propietarios hay algún vecino que no participa en este servicio, no se le daría de alta en la aplicación.

Por otro lado, es habitual que en las comunidades de copropietarios el presidente de la comunidad sea rotativo entre los vecinos de la misma. Por este motivo, la transición entre residente y presidente ha de ser sencilla.

5.1.2. Roles de usuario

A continuación se explican brevemente los roles de los usuarios dentro de la aplicación:

Administrador: puede ver y gestionar todas las comunidades de copropietarios, así como los residentes de las mismas, contratos y recibos

Presidente: puede ver y gestionar la comunidad de copropietarios de la que es presidente, así como los residentes de la misma, sus recibos y los contratos con proveedores.

Residente: puede visualizar sus facturas y realizar su pago. Adicionalmente puede visualizar algunos datos básicos correspondientes al Servicio de Internet comunitario pero no realizar ninguna modificación sobre ellos.

5.1.3. Cálculo importe de Internet comunitario por vecino

Los contratos con los proveedores de servicios de Internet los realiza la comunidad, y el coste mensual de dichos contratos se reparten de forma igualitaria entre número de viviendas dadas de alta en el servicio de Internet comunitario.

Es decir, no se tienen en cuenta los metros de la vivienda (coeficientes de propiedad), ni el uso que cada vivienda hace de la conexión. El coste por vivienda dada de alta en el servicio se calcula así:

Recibo mensual por vivienda = (recibo proveedor de Internet / nº de usuarios de Internet Comunitario).

5.2. Requerimientos

5.2.1 Login

Los usuarios deben logarse para acceder a la aplicación mediante usuario y password. Cada usuario tiene asignado un rol determinado y solo puede acceder a las funciones que tenga

disponible. Cualquier intento de acceso a cualquier página de la aplicación sin estar logado remitirá a esta pantalla. Los usuarios también pueden realizar el logout de la aplicación.

5.2.2 Gestión de comunidades

Se permite dar de alta y modificar las comunidades de copropietarios en la aplicación según el rol del usuario. El usuario administrador es quien se ocupa de dar de alta las comunidades y también puede modificarlas. Adicionalmente, el presidente de la comunidad también puede realizar modificaciones en la comunidad en la cual es presidente.

Al acceder a visualizar una comunidad se ha de observar un resumen del estado de la misma que incluirá: los datos de la comunidad, el número de usuarios del servicio de Internet comunitario y los contratos con proveedores.

5.2.3 Gestión de residentes

Se permite dar de alta, modificar y visualizar los usuarios de los servicios de Internet Comunitario (residentes y presidentes). Las altas y modificaciones de dichos usuarios la pueden realizar los administradores y los presidentes de cada comunidad de copropietarios (los presidentes solo pueden gestionar a los residentes de su comunidad).

5.2.4 Gestión de proveedores

Se ha de implementar la gestión de proveedores de servicios de Internet, permitiendo su alta, baja y modificación. El alta, baja y modificación la realizan los presidentes de cada comunidad o el administrador.

5.2.5 Gestión de contratos con proveedores

Un punto importante de la aplicación es la gestión de contratos con proveedores, permitiendo el alta, baja y modificación por parte del administrador o los presidentes de cada comunidad. Datos claves de los contratos son la fecha de alta y el importe mensual a pagar a la compañía proveedora.

5.2.6 Gestión recibos

Se trata de la parte clave de la aplicación, se implementará la emisión de recibos de Internet Comunitario a los usuarios del servicio. Esta tarea de emisión de recibos la podrán realizar administradores y presidentes de comunidad.

El importe de los recibos que el proveedor de servicios de Internet carga a cada comunidad, se dividirá entre el número de viviendas usuarias del servicio de Internet comunitario y se emitirá un recibo por cada vivienda. Esta tarea de cálculo y emisión de recibos se implementará para que sea sencilla de realizar por parte de los usuarios de la aplicación.

Los residentes de la comunidad podrán visualizar los recibos y proceder a su pago. Los presidentes podrán visualizar todos los recibos pagados y pendientes de pago de su comunidad, mientras que el administrador podrá visualizar los recibos de todas las comunidades.

5.3 Requisitos no funcionales:

RNF1: Concurrencia

La aplicación podrá ser usada al mismo tiempo por diversos usuarios sin que esto provoque ningún conflicto en la aplicación.

RNF2: Tiempo de respuesta

La aplicación responderá con suficiente velocidad para permitir una interacción correcta con el usuario. Por este motivo se evitará el envío de información innecesaria entre el navegador y el servidor, optimizando de este modo la velocidad y el tiempo de respuesta de la aplicación.

RNF3 Compatibilidad

Se garantizará que la aplicación funciona correctamente con los navegadores más usados actualmente: Internet Explorer, Firefox, Chrome, Safari, etc.

RNF4 Fiabilidad i feedback

Para que el usuario pueda obtener una sensación de fiabilidad al usar la aplicación, siempre que sea posible, la aplicación retornará feedback. Especialmente a la hora de introducir los datos, estos se comprobaran y se avisará al usuario de posibles errores.

RNF5: Simplicidad de uso

La interfaz gráfica deberá ser sencilla e intuitiva. La medida de la fuente será la adecuada para poder ser leída con facilidad.

RNF6 Estilo

La interfaz gráfica mantendrá un mismo estilo en toda la aplicación. Se evitarán colores demasiado vivos o contrastes desagradables.

5.4 Diagramas de casos de uso

5.4.1 Diagrama de casos uso de gestión de comunidades

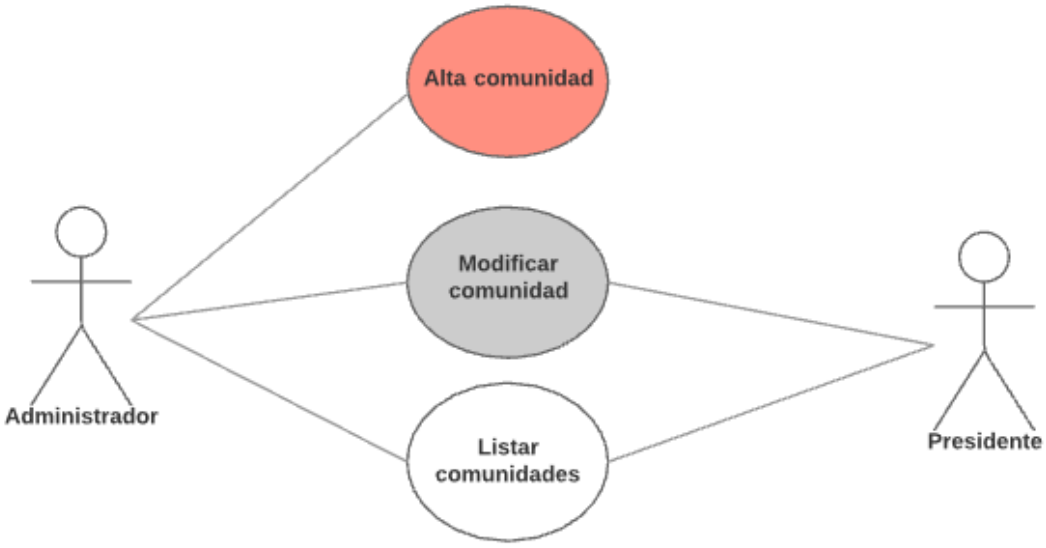

Fig. 52 Diagrama casos de uso de gestión de comunidades

5.4.2 Diagrama de casos de uso de gestión de residentes

Fig. 53 Diagrama de casos de uso de gestión de residentes

5.4.3 Diagrama de casos de uso de gestión de contratos

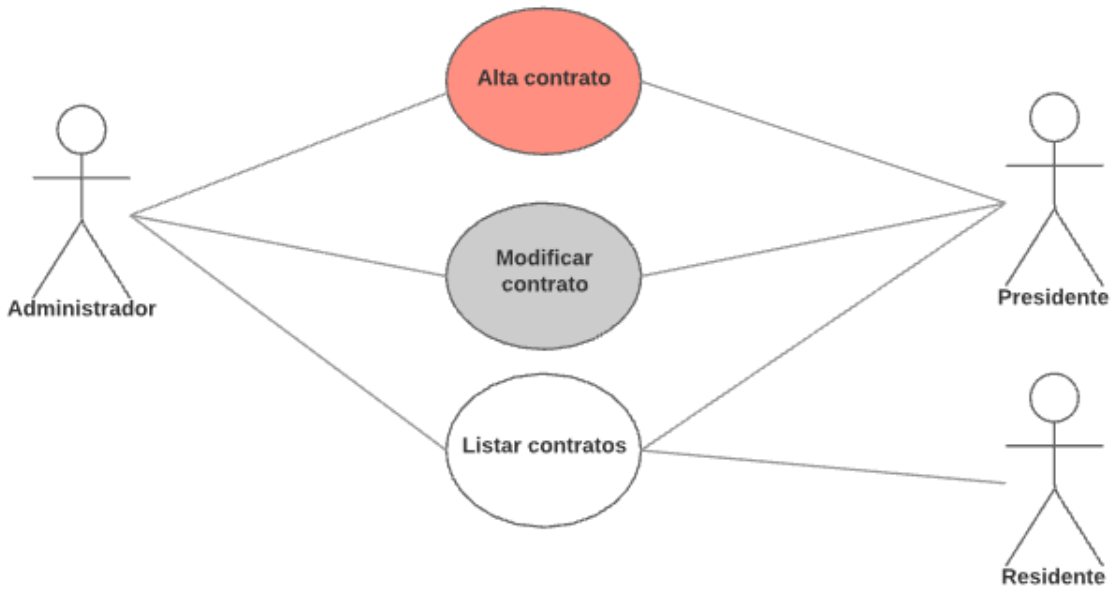

Fig. 54 Diagrama de casos de uso de gestión de contratos

5.4.4 Diagrama de casos de uso de gestión de recibos

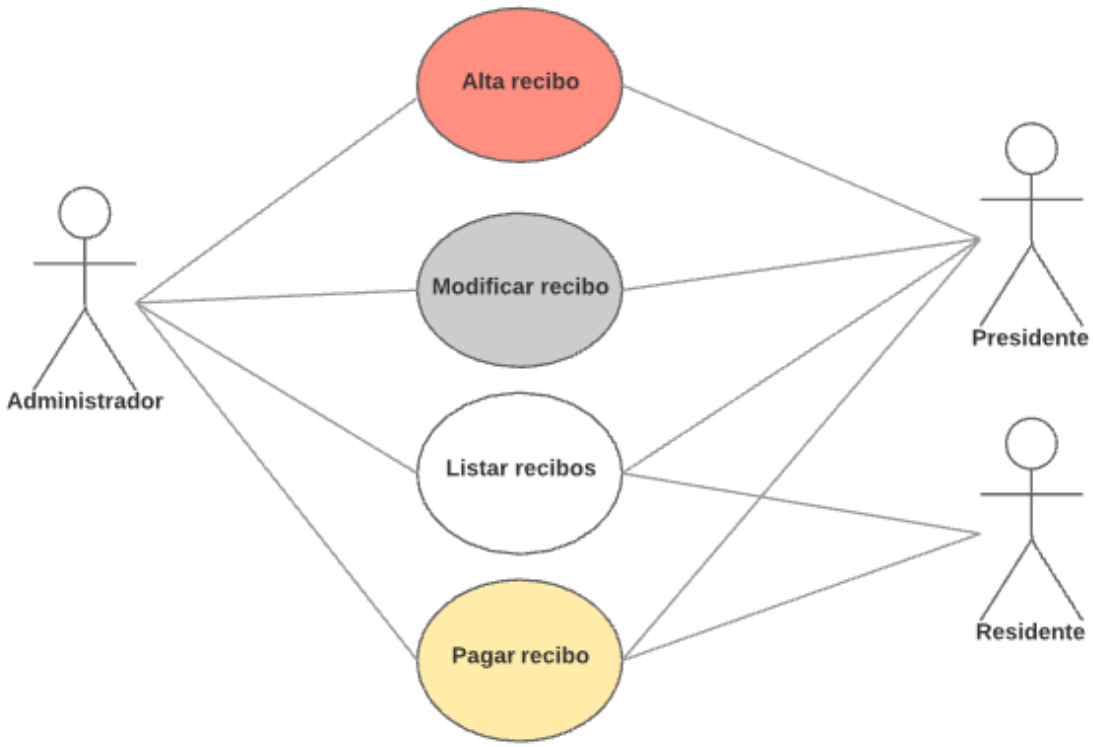

Fig. 55 Diagrama de casos de uso de gestión de recibos

5.5 Casos de uso

5.5.1 Inicio de sesión

Descripción: Es el control de acceso a nuestra aplicación. El usuario se identifica mediante login y password para acceder a nuestra aplicación.

Actores: Residente, Presidente y Administrador.

Precondición: El usuario se encuentra dado de alta en el sistema y tiene asignado al menos un rol.

Postcondición: El usuario estará logado en la aplicación y podrá interactuar con ella según su rol.

Flujo principal:

1. El usuario inicia el caso de uso.
2. El sistema solicita que se identifique mediante login y password.
3. El usuario introduce login y password.
4. Si el login es correcto el sistema muestra según el rol del usuario, las diferentes opciones que puede realizar.
5. El sistema finaliza el caso de uso.

Flujo alternativo:

El usuario en cualquier momento puede solicitar al sistema abortar el caso de uso.

3.1 Si el login i/o password no son correctos, el sistema volverá al paso 2 donde se solicitará repetir el proceso.

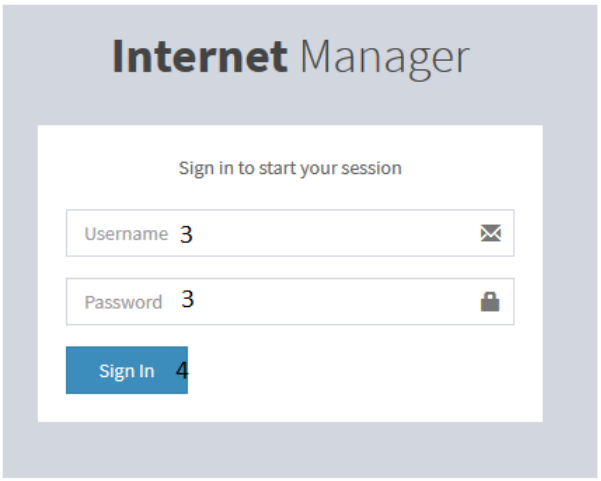

Fig. 56 Pantalla de login con los correspondientes números del flujo principal anotados

5.5.2 Logout

Descripción: el usuario cierra su sesión realizando un logout.

Actor: Residente, Presidente o Administrador

Precondición: el usuario está logado en el sistema.

Postcondición: La sesión del usuario está cerrada.

Flujo principal:

1. El usuario inicia el caso de uso.
2. El sistema le informa que su sesión ha sido cerrada.

Flujo alternativo:

2.1. Si la sesión no se ha podido cerrar, el sistema retorna al paso 1 del flujo principal.

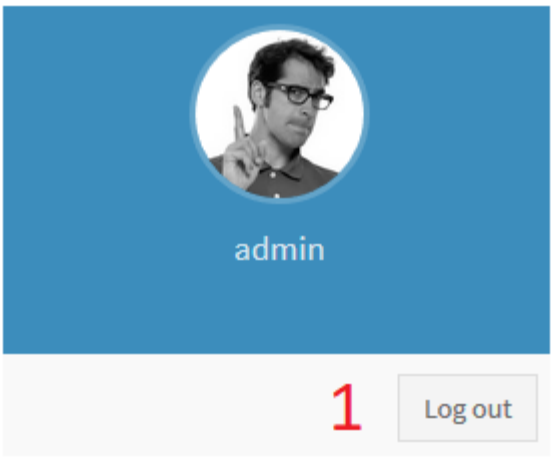

Fig. 57 Inicio de caso del caso de uso Logout

5.5.3 Listar comunidades

Descripción: El usuario visualiza una lista con todas las comunidades que tiene permiso visualizar según su rol.

Actores: Presidente y Administrador.

Precondición: El usuario se encuentra dado de alta en el sistema.

Postcondición: Ninguna.

Flujo principal:

1. El usuario inicia el caso de uso.
2. El sistema muestra una lista con las comunidades. De cada comunidad muestra: Id, NIF, nombre de la comunidad, localidad, código postal y dirección.
3. El sistema finaliza el caso de uso.

Flujo alternativo:

Si no es posible acceder a la persistencia, la aplicación informa de ello y finaliza el caso de uso.

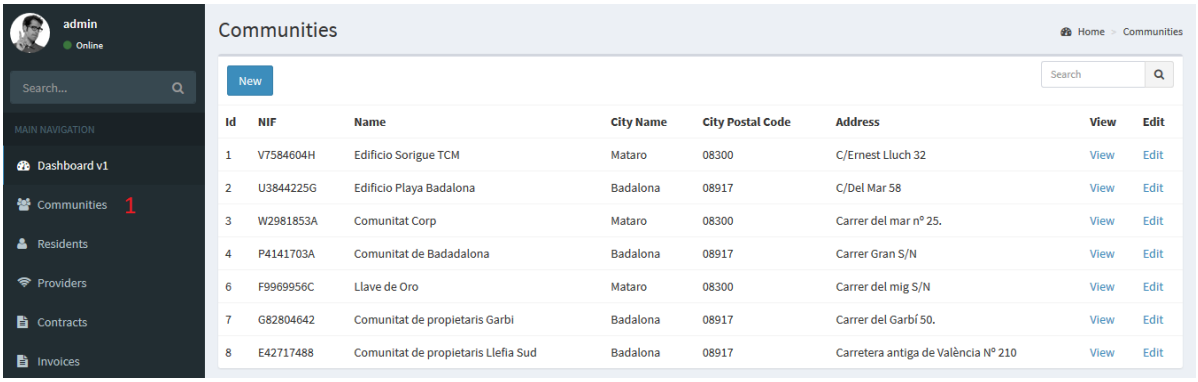

Fig. 58 Flujo principal listar comunidades

5.5.4 Alta Comunidad

Descripción: El usuario realiza una alta de comunidad en el sistema.

Actores: Administrador.

Precondición: El usuario se encuentra dado de alta en el sistema y tiene asignado al menos un rol.

Postcondición: La nueva comunidad está dada de alta.

Flujo principal:

1. El usuario inicia el caso de uso.
2. El sistema le solicita los datos de la comunidad (NIF, nombre de la comunidad, localidad y dirección).
3. El usuario introduce los datos.
4. El sistema valida los datos y los hace persistentes.
5. El sistema muestra la nueva comunidad creada.
6. El sistema finaliza el caso de uso.

Flujo alternativo:

El usuario en cualquier momento puede solicitar al sistema abortar el caso de uso.

4.1. En caso de haber dejado algún campo en blanco el sistema informa de ello y retorna al punto 2 del flujo principal.

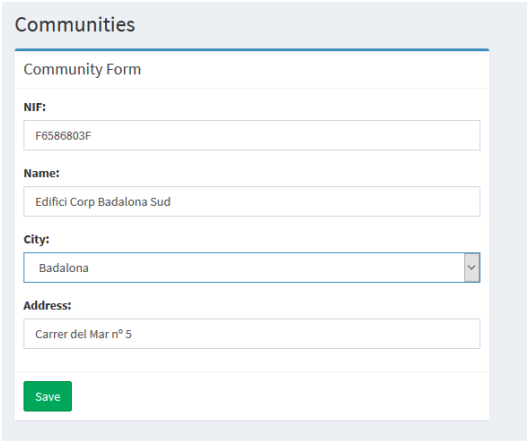

Fig. 59 Flujo principal alta comunidad

5.5.5 Modificación de una comunidad

Descripción: El usuario realiza la modificación de una comunidad.

Actores: Presidente y Administrador.

Precondición: El usuario se encuentra dado de alta en el sistema. El usuario ha de estar visualizando la comunidad que desea modificar.

Postcondición: La comunidad está modificada.

Flujo principal:

1. El usuario inicia el caso de uso.
2. El sistema le muestra los datos de la comunidad (NIF, nombre de la comunidad, localidad i dirección).
3. El usuario modifica datos deseados.
4. El sistema valida los datos y los hace persistentes.
5. El sistema muestra la comunidad modificada.
6. El sistema finaliza el caso de uso.

Flujo alternativo:

El usuario en cualquier momento puede solicitar al sistema abortar el caso de uso.

4.1. En caso de haber dejado algún campo en blanco el sistema informa de ello y retorna al punto 2 del flujo principal.

5.5.6 Listar residentes

Descripción: El usuario visualiza una lista con todos los residentes de su comunidad.

Actores: Residente, Presidente y Administrador.

Precondición: El usuario se encuentra dado de alta en el sistema.

Postcondición: Ninguna.

Flujo principal:

1. El usuario inicia el caso de uso.
2. El sistema muestra una lista con los residentes. De cada residente muestra: Id, NIF, nombre, apellidos, piso, puerta, escalera, teléfono, email y si está activo.
3. El sistema finaliza el caso de uso.

Flujo alternativo:

Si no es posible acceder a la persistencia la aplicación informa de ello y finaliza el caso de uso.

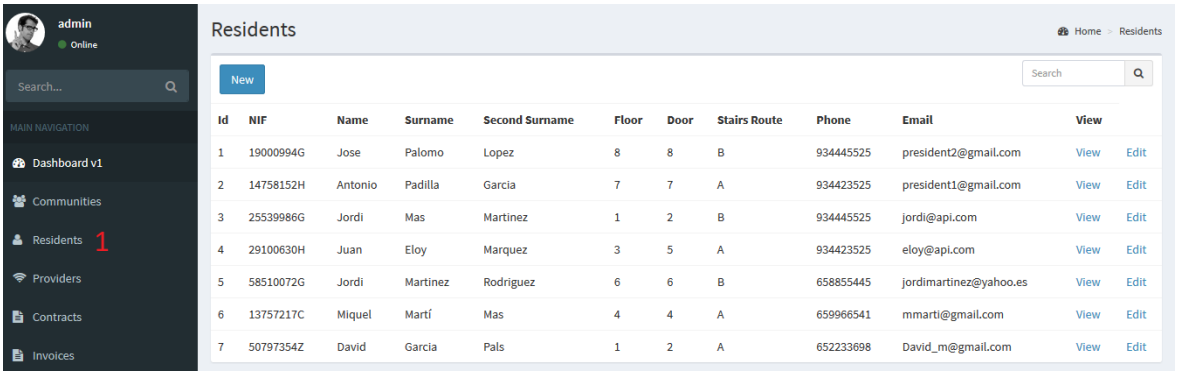

Fig. 60 Flujo principal de listar residentes

5.5.7 Alta residente

Descripción: El usuario realiza el alta de un residente en la comunidad.

Actores: Presidente y Administrador.

Precondición: El usuario se encuentra dado de alta en el sistema. El usuario solo puede dar de alta a residentes de las comunidades de las cuales es administrador o presidente.

Postcondición: el nuevo residente está introducido en el sistema.

Flujo principal:

1. El usuario inicia el caso de uso.
2. El sistema le solicita los datos del residente (NIF, nombre, primer apellido, segundo apellido, piso, puerta, escalera, teléfono, email, comunidad, password, y si es o no presidente).
3. El usuario introduce los datos.
4. El sistema valida los datos y los hace persistentes.
5. El sistema muestra el nuevo residente creado.
6. El sistema finaliza el caso de uso.

Flujo alternativo:

El usuario en cualquier momento puede solicitar al sistema abortar el caso de uso.

4.1. Si alguno de los siguientes campos no está cumplimentado: Nif, nombre, primer apellido, nº de puerta, teléfono, email o password, el sistema informa de ello y retorna al punto 2 del flujo principal.

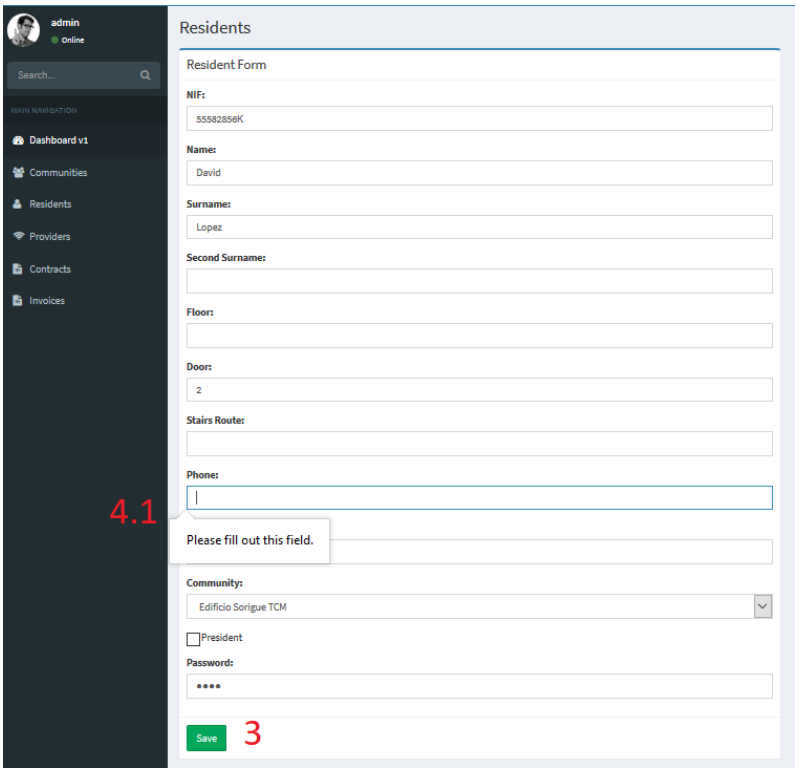

Fig. 61 Muestra el punto 4.1 del flujo alternativo del caso se uso alta residente

5.5.8 Modificación de residente

Descripción: El usuario realiza la modificación de un residente en la comunidad.

Actores: Presidente y Administrador.

Precondición: El usuario se encuentra dado de alta en el sistema. El usuario solo puede modificar las comunidades de las cuales es administrador o presidente.

Postcondición: el residente estará modificado.

Flujo principal:

1. El usuario inicia el caso de uso.
2. El sistema le muestra los datos del residente (NIF, nombre, primer apellido, segundo apellido, piso, puerta, escalera, teléfono, email, comunidad, password, si es o no presidente, y si está o no está activo).
3. El usuario modifica los datos.
4. El sistema valida los datos y los hace persistentes.
5. El sistema muestra el nuevo residente creado.
6. El sistema finaliza el caso de uso.

Flujo alternativo:

El usuario en cualquier momento puede solicitar al sistema abortar el caso de uso.

4.1. Si alguno de los siguientes campos obligatorios no está cumplimentado: NIF, nombre, primer apellido, nº de puerta, teléfono, email o password, el sistema informa de ello y retorna al punto 2 del flujo principal.

5.5.9. Listar proveedores de Internet

Descripción: El usuario visualiza una lista con todos los proveedores de Internet que tiene permiso visualizar según su rol.

Actores: Presidente y Administrador.

Precondición: El usuario se encuentra dado de alta en el sistema.

Postcondición: Ninguna.

Flujo principal:

1. El usuario inicia el caso de uso.
2. El sistema muestra una lista con los proveedores de Internet, de cada proveedor muestra: Id, NIF, nombre y tipo de proveedor.
3. El sistema finaliza el caso de uso.

Flujo alternativo:

Si no es posible acceder a la persistencia la aplicación informa de ello y finaliza el caso de uso.

5.5.10 Alta proveedor de Internet

Descripción: El usuario realiza el alta de un proveedor de Internet.

Actores: Presidente y Administrador.

Precondición: El usuario se encuentra dado de alta en el sistema.

Postcondición: el nuevo proveedor está introducido en el sistema.

Flujo principal:

1. El usuario inicia el caso de uso.
2. El sistema le solicita los datos del proveedor (NIF, nombre y tipo de proveedor).
3. El usuario introduce los datos.
4. El sistema valida los datos y los hace persistentes.
5. El sistema muestra el nuevo proveedor creado.
6. El sistema finaliza el caso de uso.

Flujo alternativo:

El usuario en cualquier momento puede solicitar al sistema abortar el caso de uso.
4.1. Si alguno de los campos solicitados no está cumplimentado se informa de ello al usuario y retorna al paso 2 del flujo principal.

5.5.11 Modificar proveedor de servicios de Internet

Descripción: El usuario modifica un proveedor de servicios de Internet.

Actores: Presidente y Administrador.

Precondición: El usuario se encuentra dado de alta en el sistema.

Postcondición: Los datos del proveedor están modificados.

Flujo principal:

1. El usuario inicia el caso de uso.
2. El sistema le muestra los datos actuales del proveedor (NIF, nombre y tipo de proveedor).
3. El usuario modifica los datos deseados.
4. El sistema valida los datos y los hace persistentes.
5. El sistema muestra el nuevo usuario creado.
6. El sistema finaliza el caso de uso.

Flujo alternativo:

El usuario en cualquier momento puede solicitar al sistema abortar el caso de uso.

4.1. Si alguno de los campos no está cumplimentado se informa de ello al usuario y retorna al paso 2 del flujo principal.

5.5.12 Listar contratos de Internet

Descripción: El usuario visualiza una lista con todos los contratos de Internet que tiene permiso visualizar según su rol.

Actores: Usuario, Presidente y Administrador.

Precondición: El usuario se encuentra dado de alta en el sistema.

Postcondición: Ninguna.

Flujo principal:

1. El usuario inicia el caso de uso.
2. El sistema muestra una lista con los contratos de Internet.
3. El sistema finaliza el caso de uso.

Flujo alternativo:

Si no es posible acceder a la persistencia la aplicación informa de ello y finaliza el caso de uso.

5.5.13 Alta contrato con proveedor de Internet.

Descripción: El usuario realiza el alta de un contrato con un proveedor de servicios de Internet para dar el servicio de Internet a una comunidad.

Actores: Presidente y Administrador.

Precondición: El usuario se encuentra dado de alta en el sistema. El usuario ha de ser administrador o presidente de la comunidad que va a realizar el contrato

Postcondición: El nuevo contrato está introducido en el sistema.

Flujo principal:

1. El usuario inicia el caso de uso.
2. El sistema le solicita los datos del contrato: nombre, fecha de inicio, precio mensual, comunidad de propietarios, proveedor y activo/inactivo.
3. El usuario introduce los datos solicitados.
4. El sistema calcula el precio mensual a pagar por el residente y lo muestra al usuario
5. El usuario guarda el contrato.
6. El sistema valida los datos y los hace persistentes.
7. El sistema muestra el nuevo contrato creado.
8. El sistema finaliza el caso de uso.

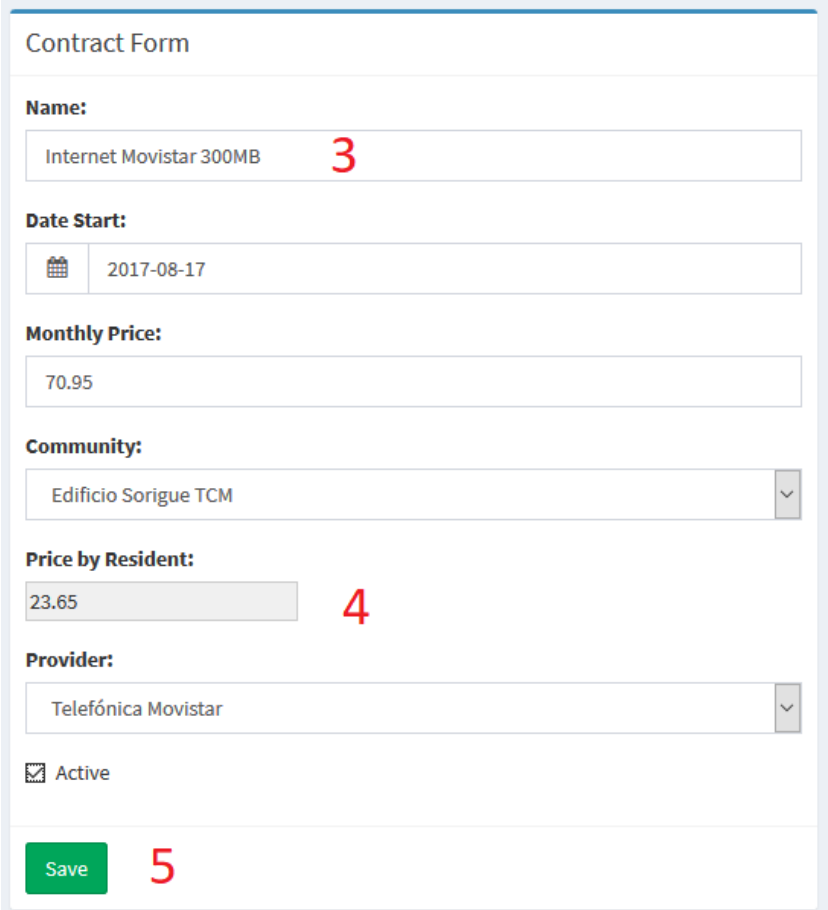

Fig. 62 Flujo principal alta contrato

Flujo alternativo:

El usuario en cualquier momento puede solicitar al sistema abortar el caso de uso.

6.1. Si alguno de los campos no está cumplimentado el sistema informa al usuario y retorna al punto 2 del flujo principal.

5.5.14 Modificación contrato con proveedor de servicios de Internet.

Descripción: El usuario realiza el alta de un contrato con un proveedor de servicios de Internet para dar el servicio de Internet a una comunidad.

Actores: Presidente y Administrador.

Precondición: El usuario se encuentra dado de alta en el sistema. El usuario ha de ser administrador o presidente de la comunidad que tiene el contrato.

Postcondición: Las modificaciones del contrato se encuentran introducidas en el sistema.

Flujo principal:

1. El usuario inicia el caso de uso.
2. El sistema le muestra los datos del contrato: nombre, fecha de inicio, precio mensual, comunidad de propietarios, proveedor y activo/inactivo.
3. El usuario modifica los datos deseados.
4. El sistema calcula el precio mensual a pagar por el residente y lo muestra al usuario
5. El usuario guarda el contrato.
6. El sistema valida los datos y los hace persistentes.
7. El sistema muestra el contrato modificado.
8. El sistema finaliza el caso de uso.

Flujo alternativo:

El usuario en cualquier momento puede solicitar al sistema abortar el caso de uso.
6.1. Si alguno de los campos no está cumplimentado se informa de ello al usuario y retorna al paso 2 del flujo principal.

5.5.15 Listar recibos

Descripción: El usuario visualiza una lista de los recibos que puede visualizar según su rol.

Un residente únicamente puede visualizar sus recibos, un presidente puede visualizar los recibos de su comunidad y un administrador puede visualizar los recibos de todas las comunidades.

Actores: Residente, Presidente y Administrador.

Precondición: El usuario se encuentra dado de alta en el sistema.

Postcondición: Ninguna.

Flujo principal:

1. El usuario inicia el caso de uso.
2. El sistema muestra una lista con las facturas, de cada recibo muestra Id, fecha de emisión, nombre del contrato, nombre del residente, total y pagado/pendiente de pago.
3. El sistema finaliza el caso de uso.

Flujo alternativo:

Si no es posible acceder a la persistencia la aplicación informa de de ello y finaliza el caso de uso.

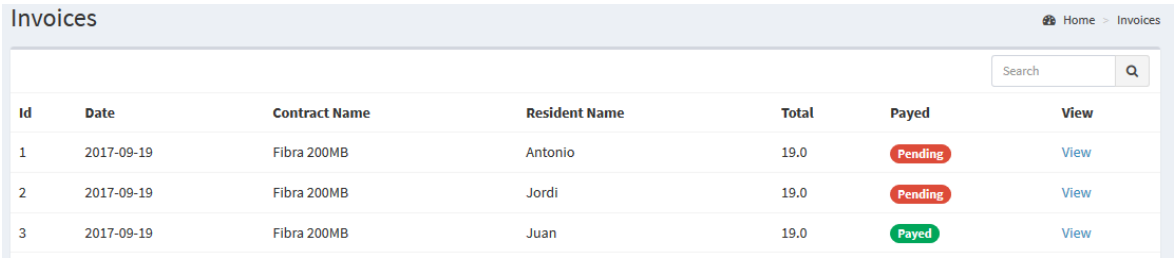

Fig. 63 Caso de uso listar recibos

5.5.16 Emisión de recibos.

Descripción: El usuario realiza la emisión de los recibos correspondientes a un contrato de Internet Comunitario. El sistema crea un recibo para cada usuario de la comunidad activo. Únicamente puede emitirse un recibo cada mes.

Actores: Presidente y Administrador.

Precondición: El usuario se encuentra dado de alta en el sistema. El usuario ha de ser administrador o presidente de la comunidad de la cual se van a emitir recibos.

Postcondición: Habrá un recibo emitido para cada residente de la comunidad que está activo.

Flujo principal:

1. El usuario inicia el caso de uso.
2. El sistema crea un recibo para cada residente de la comunidad que está activo.
3. El sistema informa que los recibos han sido creados.
4. El sistema finaliza el caso de uso.

Flujo alternativo:

El usuario en cualquier momento puede solicitar al sistema abortar el caso de uso.

3.1 Si los recibos de este mes ya han sido creados anteriormente, o por otro motivo no pueden crearse los recibos, el sistema informa al usuario y finaliza el caso de uso.

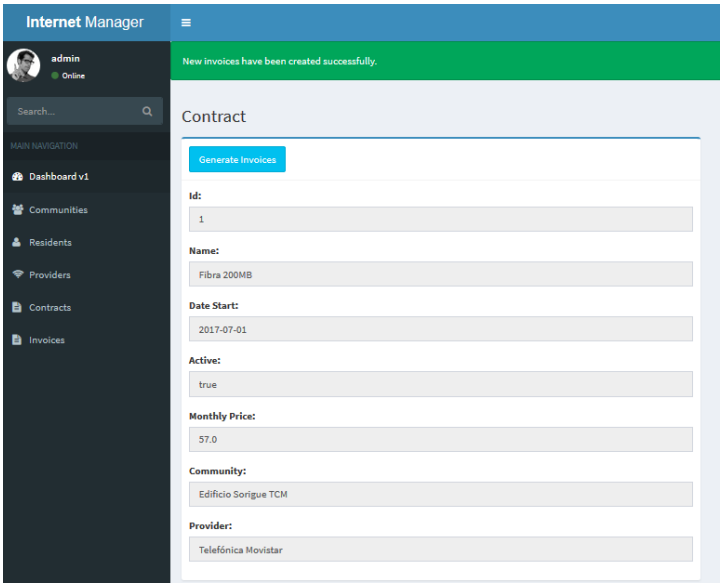

Fig. 64 Punto 3 del flujo principal del caso de uso emisión de recibos

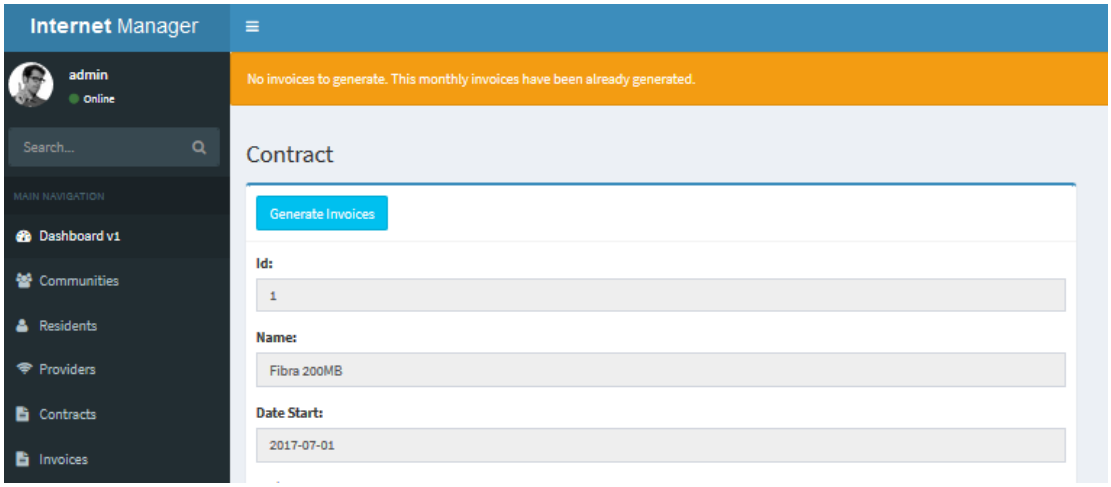

Fig. 65 Punto 3.1 del flujo alternativo del caso de uso emisión de recibos

5.5.17 Pago de recibos.

Descripción: El usuario realiza el pago de un recibo. Los residentes pueden realizar el pago de sus recibos, los presidentes pueden realizar el pago de recibos correspondientes a residentes de su comunidad, y el administrador puede realizar el pago de cualquier recibo. El pago de recibos se realiza mediante tarjeta bancaria.

Actores: Residente, Presidente y Administrador.

Precondición: El usuario se encuentra dado de alta en el sistema. El recibo ha de estar pendiente de pago.

Postcondición: El recibo constará registrado como pagado.

Flujo principal:

1. El usuario inicia el caso de uso.
2. El sistema le pide los datos de la tarjeta bancaria.
3. El usuario introduce los datos de la tarjeta bancaria.
4. El sistema comprueba los datos.
5. El sistema informa al usuario que el pago ha sido realizado correctamente y el recibo pagado.
6. El sistema finaliza el caso de uso.

Flujo alternativo:

El usuario en cualquier momento puede solicitar al sistema abortar el caso de uso.

4.1 Si los datos de la tarjeta no son correctos el sistema informa del hecho y retorna al paso 2 del flujo principal.

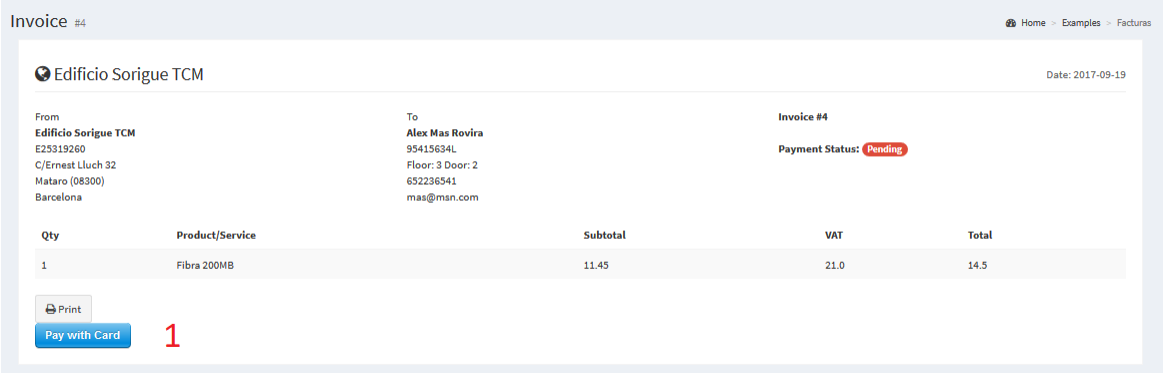

Fig. 66 Se muestra el paso 1 del caso de uso pago de recibo

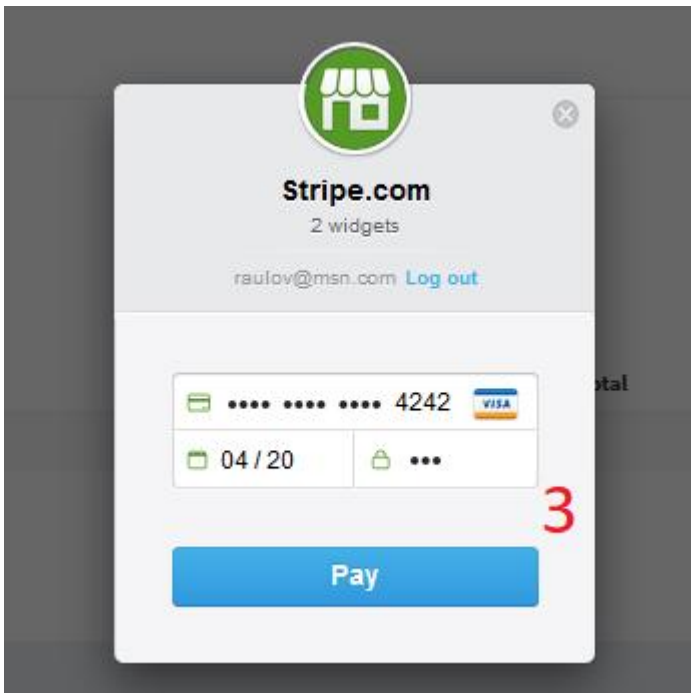

Fig. 67 Se muestra el paso 3 del caso de uso pago de recibo

5.6 Diagrama de clases

A continuación se muestra el diagrama de clases del proyecto.

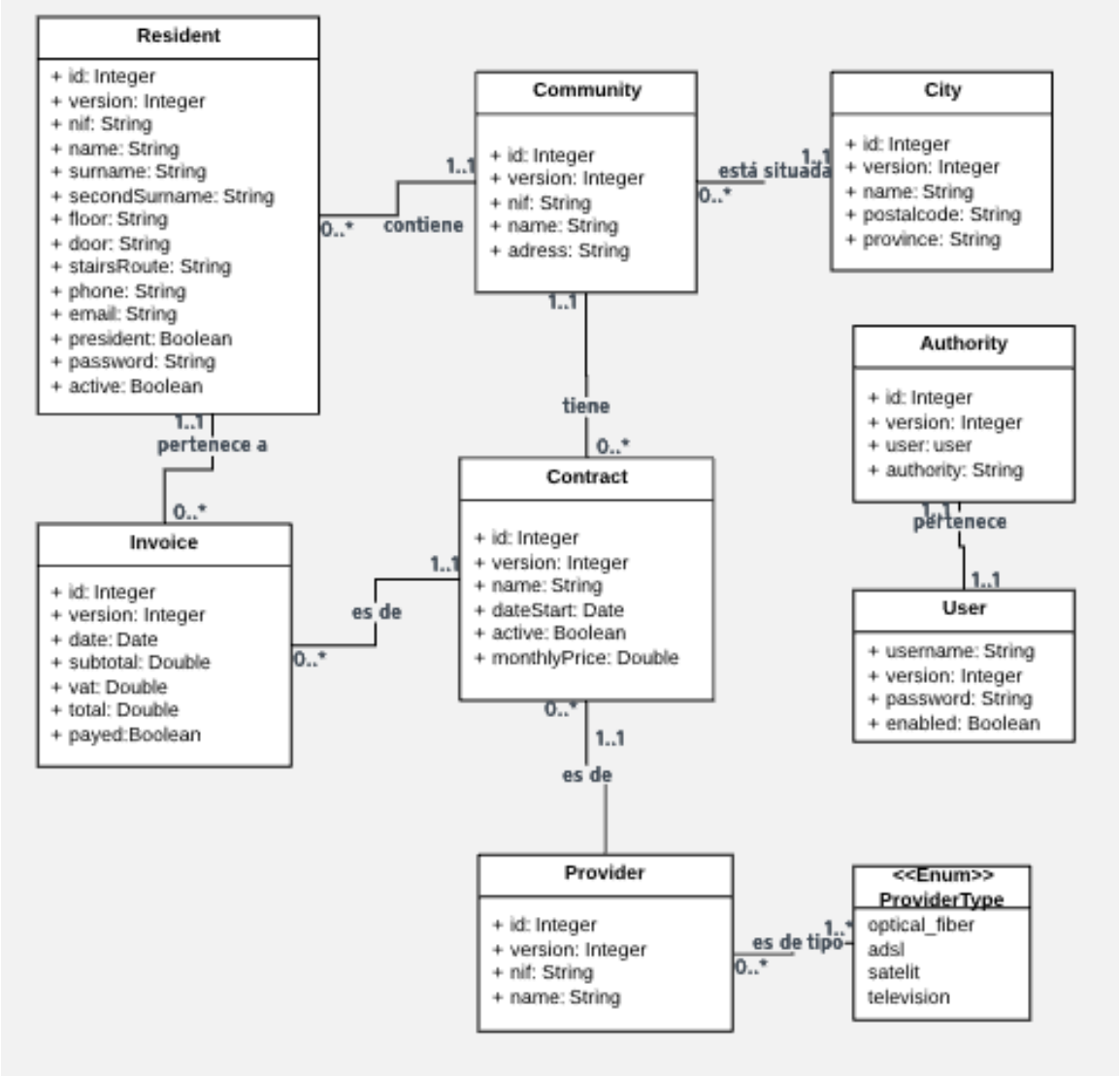

Fig. 68 Diagrama de clases del proyecto

6. Estructura de la aplicación

En este capítulo voy a explicar la función de los distintos archivos del proyecto, y también como está estructurado el código.

En la siguiente imagen se pueden observar las carpetas principales del proyecto: las carpetas main y resources. La carpeta main es la carpeta principal de proyecto, y contiene los ficheros Java. La carpeta resources contiene los recursos estáticos y templates del proyecto.

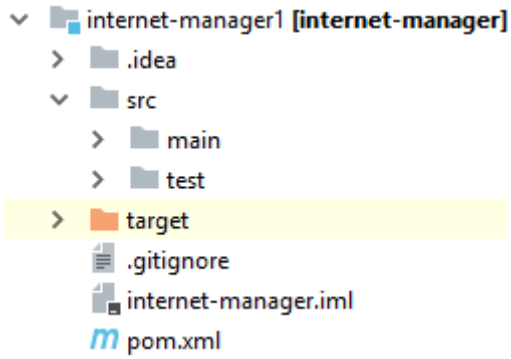

Fig. 69 Estructura de del proyecto

En la raíz del proyecto puede observarse el fichero pom.xml. Dado que este proyecto se ha realizado con Spring y usando Maven, las dependencias del proyecto (librerías que se necesitan) se asignan en fichero pom.

6.1 Configuración archivo pom y dependencias usadas

A continuación procedo a explicar la configuración del archivo pom y las dependencias usadas en este proyecto:

Dado que en este proyecto se ha usado Spring Boot, se especifica como parent la versión de Springboot que se va a usar. En este caso, se puede observar que la versión de Spring Boot con la que se lanzará el proyecto es la 1.5.2 RELEASE.

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.2.RELEASE</version>
    <relativePath/>
</parent>
```

Fig. 70 Dependencia parent del fichero POM

En el archivo POM también se ha de especificar la versión de Java que se va a usar en el proyecto, en este caso es la 1.8:

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    <java.version>1.8</java.version>
</properties>
```

Fig. 71 Especificación de la versión Java usada en el proyecto

Por último se han de especifican las dependencias usadas en el proyecto las cuales son las siguientes:

- Starter web: dependencia para el desarrollo de aplicaciones web usando Spring MVC, por defecto esta dependencia añade un Tomcat incrustado en el proyecto.
- Starter data JPA: dependencia para la para la persistencia usando JPA.
- Starter Thymeleaf: para la capa de presentación y la interacción con templates.
- Starter Security: implementación de seguridad de Spring.
- Conector MySQL: para conectar el proyecto con un servidor MySQL

Adicionalmente se han añadido 3 dependencias para trabajar con templates y formularios: bootstrap, jquery y una extensión de Thymeleaf Security.

6.2 Estructura del código

A continuación procedo a explicar la arquitectura del proyecto, que como se ha indicado anteriormente se divide 2 carpetas principales: java y resources.

Dentro de la carpeta resources encontramos los siguientes elementos:

- **Templates**: se trata de ficheros HTML, algunos de ellos etiquetados con Thymeleaf y que generarán las vistas de la aplicación
- **Static**: se trata de ficheros estáticos principalmente CSS y Javascript.
- **Application.properties:** es el fichero de propiedades del proyecto, y en él se indican los entornos de la aplicación.

Las clases Java del proyecto se encuentran ordenadas en paquetes según su función, a continuación el tipo de clases que contiene cada paquete:

- **Controllers**: son las clases que reciben las peticiones web de los clientes de la aplicación, se comunican con los servicios y retornan una respuesta.
- **Services**: contiene los servicios de la aplicación. En los servicios se encuentra la lógica de negocio de la aplicación.
- **Domain**: contiene las clases del dominio.
- **Repositories**: contiene los repositorios para la persistencia.
- **Security**: contiene la configuración de seguridad de la aplicación.
- **Exception**: contiene clases para tratar excepciones en la aplicación.

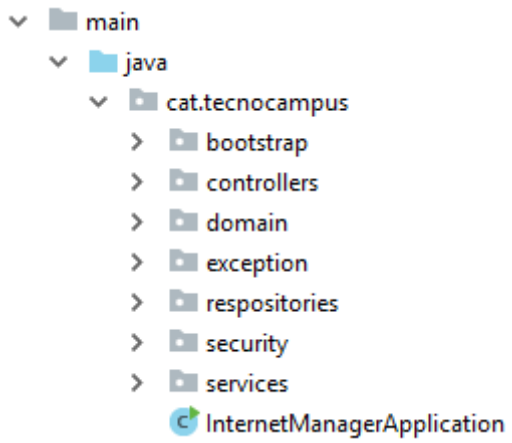

Fig. 72 Estructura de clases del proyecto

7. Estudio económico

7.1 Coste de desarrollo de software

En este apartado se muestran los gastos relacionados con el desarrollo de este proyecto, se tienen en cuenta las siguientes cuestiones:

- Costes de desarrollo y programación
- Costes administrativos (redacción de la memoria)
- Gastos de material y software
- Costes de amortización del material empleado

7.1.1 Coste de material necesario para el desarrollo de la aplicación

Tabla 1 Costes materiales

Descripción	Cantidad	Precio Unitario (€)	Total (€)
Equipo Hardware			
PC Portátil	1	800	800
Licencias de Software			
Licencia Office 2016 Hogar y Empresa	1	279	279
IntelliJ IDEA	1	0	0
Mamp	1	0	0
Total coste material			**1079**

7.1.2 Costes de recursos humanos

Tabla 2 Costes recursos humanos

Concepto	Horas	Precio/hora (€)	Total (€)
Aprendizaje Spring Framework (programador)	150	40	6000
Análisis y diseño (analista)	160	50	8000
Desarrollo de la aplicación (programador)	600	40	24000
Redacción memoria (administrativo)	100	30	3000
Coste total recursos humanos			**35000**

El tiempo invertido en el aprendizaje del framework y sus diferentes módulos no se ha tenido en cuenta al calcular los costes de desarrollo del proyecto, dado que se ha considerado que en caso de contratar a un programador para el desarrollo de la aplicación, sería requisito el indispensable conocer estas herramientas y tener experiencia en el desarrollo de aplicaciones web.

7.1.3 Amortización de equipos y software

Tabla 3 Amortización

Equipos y software	Horas de uso	Precio/hora (€)	Total (€)
PC Portátil	860	0.5	430
Licencia Office 2016 Hogar y Empresa	279	1	279
IntelliJ IDEA	1	0	0
Mamp	1	0	0
Total amortizaciones			**709**

7.1.4 Coste del desarrollo de la aplicación

Se ha considerado que los costes indirectos del proyecto por el gasto en conexión a Internet, uso de servidores cloud, luz, y utilización de un local representan el 10% del coste del proyecto.

Tabla 4 Coste del desarrollo de la aplicación

Concepto	Coste (€)
Costes de material	1079
Costes recursos humanos	35000
Costes de amortización	709
Subtotal	36788
Costes indirectos (+10%)	3678.8
Coste total	**40466.8**

7.2 Precio de venta del software

7.2.1 Cálculo del coste por unidad

La comercialización de este software haría por modalidad de alquiler anual de licencia. Se considerará el precio de licencia por comunidad. Esto es, si un cliente desea administrar diversas comunidades, se facturaría el precio de una licencia por cada una de ellas.

A continuación se va a realizar el estudio económico para la comercialización de 600 licencias, es decir 600 comunidades administradas con este software.

- Para la administración y mantenimiento de la aplicación se considera necesario un analista programador a tiempo completo.
- No hay costes de fabricación, dado que se plantea subir la aplicación a un servidor cloud para que los usuarios accedan de forma online.

Tabla 5 Coste mantenimiento anual por licencia

Equipos y software	**Coste Anual (€)**
Salario bruto analista programador	36000
Costes gestoría	3600
Costes servidor Cloud	4200
Local y gastos	9000
Coste mantenimiento anual por licencia	**88 €**

7.2.1 Cálculo del precio de venta

Dado que el coste de mantenimiento anual por licencia es de 88€ voy a añadir el coste de desarrollo de la aplicación, teniendo en cuenta que se desea recuperar la inversión realizada en un año, y añadiré un margen comercial del 25%

Tabla 6 Precio de venta al público

Concepto	**Coste (€)**
Coste soporte para 600 licencias	52800
Costes del desarrollo de desarrollar el software	40466.80
Coste total	93266.80
Coste por licencia	155.45
Margen comercial por licencia 25%	38.86
Precio de venta licencia anual	**194.31**

Finalmente se obtiene un precio de alquiler de licencia anual de 194.31€ por cada comunidad, que divido entre 12 meses nos da un coste mensual de 16,2€. Se trata de un coste bastante competitivo teniendo en cuenta que en este precio se incluye el soporte.

Se ha de tener en cuenta la posible necesidad de adecuar el software a las necesidades de cada cliente, lo cual haría necesario estudiar cada caso y presupuestar en caso necesario el coste de las adaptaciones.

8. Conclusión.

Después de estudiar Spring, Spring boot, así como sus principales módulos, puedo destacar que la experiencia que he tenido al utilizar este framework en el desarrollo del proyecto ha sido muy positiva. Se trata de un framework muy potente e interesante, y personalmente considero que por su facilidad de uso y la flexibilidad que permite, es un framework a tener muy en cuenta si se desea desarrollar una aplicación web.

Para finalizar considero que el resultado del proyecto es adecuado ya que se ha conseguido el desarrollo una aplicación con una estructura sólida con la que poder seguir trabajando, mejorando y añadiendo nuevas funcionalidades.

La aplicación es totalmente funcional y está lista para ser desplegada en un servidor Cloud, permitiendo de esta forma su uso desde cualquier dispositivo sin necesidad de ninguna instalación, siendo el único requisito el disponer de un navegador y conexión a internet.

Dada la rápida evolución e implantación de los dispositivos móviles y pantallas táctiles, se ha desarrollado una interfaz web para ser accesible desde estos dispositivos, aumentando así las posibilidades de uso de la aplicación.

Para seguir mejorando la aplicación se deberían implementar las siguientes mejoras:

- Implementación multiidioma de la aplicación: realizar su traducción a diversos idiomas y permitir que el usuario seleccione el idioma deseado.
- Implementación del pago de recibos mediante cuenta bancaria (domiciliación de recibos).
- Envío de correos electrónicos a los usuarios que no tienen domiciliado el pago de recibos, informando que se les ha emitido un recibo y han de proceder a su pago.
- Implementar el cambio y recuperación de contraseñas en caso de que un usuario olvide su password de acceso a la aplicación.
- Permitir que un mismo residente esté dado de alta en dos comunidades a la vez con la misma dirección de correo electrónico.

9. Referencias.

[1] http://www.ine.es/prensa/np991.pdf Nota de prensa del Instituto Nacional de Estadística sobre Equipamiento y uso de las Tecnologías de Información y Comunicación en los hogares. 3 Octubre 2016.

[2] https://www.cnmc.es Página de la CNMC 'Comisión Nacional de los Mercados y la Competencia'. Se trata del organismo que regula el sector de las comunicaciones.

https://blog.cnmc.es/2010/09/14/vecinos-que-comparten-wifi/ Entrada al blog de la CNMC donde específicamente se trata el tema de compartir Internet en una comunidad de propietarios. 14 Septiembre 2010.

https://blog.cnmc.es/wp-content/uploads/2010/09/RO_2009_630.pdf Documento donde el Consejo de la Comisión del Mercado de las Telecomunicaciones se pronuncia a favor de la legalidad de Internet Comunitario en una comunidad de propietarios. La consulta fue realizada a petición de una comunidad de propietarios de Sant Cugat. 7 Septiembre 2010.

[3] https://docs.spring.io/spring/docs/current/spring-framework-reference/html/overview.html,Spring Framework reference Documentation 4.3.11.RELEASE. Copyright © 2004-2016.

[4] http://docs.spring.io/spring/docs/4.2.x/spring-framework-reference/html/beans.html#beans-constructor-injection, Spring Framework reference Documentation 4.2.9.RELEASE. Copyright © 2004-2016.

[5] https://docs.spring.io/autorepo/docs/spring/3.2.x/spring-framework-reference/html/beans.html Spring Framework reference Documentation 3.2.18.RELEASE. Copyright © 2004-2016.

[6] https://spring.io/blog/2015/11/29/how-not-to-hate-spring-in-2016 P. Webb. How not to hate Spring in 2016. November 29th 2016

http://olivergierke.de/2013/11/why-field-injection-is-evil/ O. Vierke. Why Spring Injection is Evil. November 22th 2013.

[7] https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#_working_with_spring_boot Spring Boot reference guide Documentation 1.5.7.RELEASE. Copyright © 20012-2017.