Universidad de Valladolid

Escuela Técnica de Ingeniería Informática de Valladolid

Grado de Ingeniero Informático con mención en Ingeniería del Software

TRABAJO DE FIN DE GRADO

BUSY:
Una plataforma para la gestión de reservas por hora en PyMes

Autor: Juan Carlos González Cabrero
Tutora: Yania Crespo Carvajal

15 de julio de 2016

Resumen

El objetivo de este proyecto consiste en el desarrollo de una plataforma software enfocada a pequeñas y medianas empresas (PyMEs), para llevar a cabo la organización de servicios ofrecidos a los clientes.

buSy es una herramienta cómoda y sencilla para la gestión de reservas y control de horarios con un ámbito de negocio generalista, ya que su enfoque global permite la utilización de esta aplicación a una amplia gama de categorías de empresa.

Se trata de un proyecto web desarrollado en **Java** y **HTML5**, utilizando el framework **Spring** como base del proyecto.

Siguiendo la filosofía del desarrollo ágil, se han aplicado los principios TDD y BDD desde el principio del proyecto. Gracias a ello, la implementación ha sido guiada a través de las pruebas que definen el comportamiento de cada funcionalidad, lo que ha facilitado la determinación del ámbito o *scope* de cada historia de usuario del proyecto.

Se han aplicado además varios patrones de diseño sobre la arquitectura software, como son la **programación en capas** y el patrón **modelo-vista-controlador**.

«A la memoria de mi padre, un hombre fuerte y emprendedor, cuyos sueños se vieron truncados por una horrible enfermedad.»

Agradecimientos

A mi familia, por aguantarme en mis momentos de estrés, y porque sin ellos no habría podido llegar hasta aquí.

A mi tutora Yania, por su entusiasmo e implicación con los alumnos, siendo así una profesora ejemplar.

A mis amigos y compañeros de carrera, con los que he aprendido mucho, tanto como ingeniero informático como persona.

A mi novia Beatriz, por ayudarme y apoyarme en la realización de este proyecto.

Y por último, a todos aquellos que me han animado y han creído en mí en esta etapa final.

Gracias

Índice general

Índice de figuras

CAPÍTULO 1

Introducción

1.1. Objetivos del proyecto

El objetivo de este TFG es desarrollar una aplicación web destinada a un ámbito de PyMEs y, más concretamente, a centros de facturación por horas de servicio consumido.

Esta idea de proyecto, planteada como iniciativa propia, surge tras detectar una necesidad básica en el día a día del trabajo de múltiples sectores de negocios. Los objetivos que se han establecido para definir el alcance de este proyecto son los siguiente:

- Facilitar a los clientes reservar y organizar sus propios horarios.
- Facilitar a la empresa llevar una organización de su horario de servicios.
- Evitar los colapsos de horas y trabajadores en los períodos de trabajo.
- Mantener siempre informado al cliente de los horarios disponibles.
- Mantener almacenados los datos relativos a horarios y reservas, para evitar pérdidas de información.
- Prevenir malentendidos de organización entre empleado y cliente.

Debido a mi interés en aplicar todo lo aprendido y llevar a cabo una aplicación con grandes posibilidades de evolución y de implantación en un entorno laboral, decidí proponer ***buSy*** como mi proyecto final de grado, para llevar a cabo su desarrollo bajo la supervisión y el seguimiento de Yania.

La aplicación tendrá dos conjuntos de funcionalidades distintas ya que se podrá acceder a ella desde el punto de vista del cliente, que consume servicios de una o varias empresas, o desde el punto de vista del empresario o persona a cargo en una determinada empresa.

Desde el rol del encargado de la empresa la aplicación permitirá la gestión y organización de los servicios y horarios de la empresa, la posibilidad de facturación de dichos servicios, la organización de los datos tanto de clientes como de trabajadores de la empresa, y la visualización de un historial de servicios proporcionados para un posterior análisis que permita optimizar la gestión del tiempo y recursos de la empresa.

Desde el rol de cliente permitirá la búsqueda y comparación de las empresas que ofrezcan un determinado servicio deseado, contratar horas de servicio y la posibilidad de realizar los pagos de una forma cómoda y sencilla. Además, podrá mantener una organización horaria de una forma global de los horarios asociados a los diferentes servicios contratados.

1.2. Contexto de la aplicación

La idea de ***buSy*** surge tras la observación del proceso y la gestión de las reservas en varios negocios pequeños y medianos, como son una academia de enseñanza, una peluquería de barrio, o una pequeña autoescuela.

En el caso de la academia, las reservas se realizaban tanto por teléfono como en persona. Los registros de reservas se almacenaban en un simple cuadernillo en el que el responsable de la academia, quien también ejercía de profesor, anotaba con una letra diminuta y a modo de tablas la lista de personas apuntadas en cada hora y con cada profesor.

Dicha solución requería buscar en aquel cuadernillo el día concreto en el que se pedía una reserva, y una vez encontrado consultar el número de alumnos en cada hora para poder ofrecer las alternativas, además de determinar, dependiendo de la asignatura y el profesor(a) con el que se va a realizar dicha reserva, el número máximo de alumnos admitidos en la clase.

Otro de los problemas que advertí fue la necesaria intervención del responsable para poder realizar una reserva, ya qué al faltar dicha persona causaría posibles pérdidas de reservas, además del estrés al que se pueda ver sometida cuando aumentaba la acumulación de personas a la espera de reservar debido al tiempo que este proceso conllevaba. Este problema se incrementaba aún más cuando se solicitaban las reservas durante el período de una clase impartida por el responsable, perjudicando también a sus estudiantes y por ende al cliente final.

Al igual que en la academia, en la peluquería de barrio también se realizaban las reservas

tanto de forma presencial como de forma telefónica, al menos cuando el responsable de la peluquería o peluquero jefe escuchaba el tono del teléfono. Como es lógico, el entorno acústico dentro de una peluquería no siempre es adecuado para la comunicación, debido principalmente al ruido de los secadores de pelo. Esto complica el proceso de reserva de forma oral, haciendo que en ocasiones sucedan malentendidos en la hora de reserva.

En el caso particular que he observado, realizar con éxito una reserva no implicaba tener una hora fija a la que recibir el servicio deseado, puesto que una peluquería convencional no siempre precisa de la reserva de un cliente para poder atenderle. Esto implica la carga de nuevos servicios sin ningún tipo de control horario, lo que puede perjudicar a la satisfacción global de sus clientes debido al incumplimiento de tiempos. Mantener una organización de los servicios basada en el tipo de dicho servicio, su duración, y el horario común de reservas, podría mejorar el modelo de organización e incrementar la calidad del servicio ofrecido como consecuencia.

Por otro lado, el caso de la pequeña autoescuela suponía dos casos completamente diferentes.

En primer lugar, se tiene el servicio de clase teórica, que no necesitaba registro alguno, en el cuál aparecía un grave problema tanto cuando no asistía persona alguna como cuando había un exceso de clientes para la misma clase. Además, la comunicación de los horarios de clases, fijados dinámicamente, no se llevaba a cabo de una forma óptima, al tratarse de un escaso número de empleados, lo cuál resultaba en un inconveniente a la hora de mantener siempre informados a los clientes del servicio.

En segundo lugar, se brinda el servicio de clase práctica. La reserva de una sesión de este servicio sólo se podía ejecutar de forma presencial, es decir, concretando fecha y hora de la clase con el profesor de las prácticas, en este caso también el responsable de la autoescuela. Puesto que la reserva precisaba de la presencia de esta persona, se hacía imposible el proceso cuando no se encontraba disponible, por lo que la reserva no se llevaba a cabo en numerosas ocasiones.

Estos tres casos distintos sirven como base a una generalización de la aplicación para empresas pequeñas y medianas, que brinden sus servicios por franjas horarias con recursos limitados. Otros ejemplos podrían ser clínicas privadas, academias de música, talleres de reparación, negocios de limpieza, etc.

CAPÍTULO 2

Planificación y gestión del proyecto

2.1. PaaS

PaaS (Platform as a Service) es un tipo de servicio de computación en la nube, que lo que nos proporciona es un entorno completo en el que los desarrolladores pueden alojar, desplegar y gestionar sus aplicaciones y servicios, sin la complejidad de tener que configurar una infraestructura de servidores y de bases de datos, así como su posterior mantenimiento.

Esta es una gran alternativa a los tradicionales *hostings* de aplicaciones, aportando una mayor robustez al proporcionar una red de servidores, generalmente distribuida, para que la aplicación se mantenga en producción. Los proveedores de infraestructuras PaaS ponen a disposición del desarrollador diversas herramientas y módulos, pudiendo configurar el entorno según las necesidades del proyecto.

Gracias a las plataformas PaaS, la velocidad de la puesta a punto de una aplicación se incrementa notablemente, permitiendo que un producto de software pueda pasar de desarrollo a producción en apenas unos segundos.

Principales ventajas de usar un servicio PaaS:

- **FIABILIDAD:** las aplicaciones o servicios son alojados en particiones virtuales de una amplia red de servidores, en lugar de alojarse en un único servidor, como sucede con los *hostings* tradicionales. Cuando un servidor queda fuera de servicio, ésto no afecta a la disponibilidad del servicio o aplicación en producción, ya que sigue habiendo disponibilidad de recursos, lo que reduce considerablemente el riesgo.
- **ESCALABILIDAD Y FLEXIBILIDAD:** Los recursos pueden estar disponibles en

tiempo real y bajo demanda, es decir, sin limitación alguna de la capacidad de un servidor. Esto se logra gracias a poder disponer de particiones virtuales como recursos en lugar de máquinas físicas.

- **TARIFICACIÓN POR USO:** El modelo de tarifa de estos servicios no se basa en cuotas periódicas, sino en el pago por la utilización de los recursos contratados. Esto es así para evitar que el usuario de un servicio PaaS tenga que pagar por más recursos de los que necesita, y a su vez se optimice el rendimiento a través de la red de servidores evitando malgastar la capacidad de estos. Además se permitirá así soportar sin ningún problema los picos de demanda de nuestro servicio o aplicación en los que se supere el número habitual de peticiones.
- **BALANCEO DINÁMICO DE CARGA:** Los recursos contratados se pueden modificar dinámicamente, ajustándose a la demanda, para ofrecer en todo momento el mejor rendimiento sin desaprovechar recursos.

Desventaja a la hora de utilizar una *Plataforma como Servicio*:

- La capacidad de abstraerse de toda la complejidad que conlleva la configuración y gestión del entorno, tiene como consecuencia que el grado de configurabilidad depende del proveedor PaaS, al igual que el rango de herramientas que se pueden usar. Sin embargo muchos de los proveedores actuales disponen de un gran abanico de posibilidades en cuanto a herramientas, servicios y configuraciones.

Estos son algunos ejemplos de plataformas que ofrecen este tipo de servicios:

- **Heroku [34]:** Esta plataforma, desarrollada por la compañía *Salesforce*, destaca especialmente por su integración con una gran variedad de servicios por medio de *Add-ons*, lo que permite añadir multitud de funcionalidades extra a cualquier aplicación. Heroku dispone además de una serie de guías y gran documentación de referencia para los desarrolladores que ayudan a aprovechar todas las ventajas que nos ofrece su plataforma.

  La escalabilidad de las aplicaciones en Heroku se basan en unidades de cómputo denominadas *dynos*. Cuantos más dynos, más carga podrá soportar de media la aplicación alojada. [25]
- **Openshift [16]:** Openshift es una plataforma como servicio desarrollada por *Redhat*, que ofrece gran facilidad a la hora de empezar de cero un proyecto, ya que dispone de una gran variedad de arquetipos o aplicaciones base preconfiguradas, para un inicio rápido de tu aplicación, lo que permite así eliminar la complejidad de montar un entorno de desarrollo sobre el que poder utilizar ciertas tecnologías.

Una de sus desventajas es la limitación del plan de servicio gratuito. En dicho servicio, el servidor de despliegue que aloja nuestras aplicaciones puede quedar hibernado tras un período de 24 horas, obligando al desarrollador a realizar un reinicio manual de éste.

Ambas plataformas ofrecen una capa de servicio gratuita, y soportan multitud de tecnologías y lenguajes, entre ellos Java. Además, ofrecen integración con Git, permitiendo incluso automatizar el despliegue en una de las ramas para ejecutarlo con cada actualización o *commit* realizado.

Analizando ambas opciones, se ha decidido utilizar la plataforma de Heroku [34]. Sobre dicha plataforma se ha ido realizando un despliegue continuo, de forma que tras finalizar cada una de las historias de usuario se han ido realizando entregas continuas del producto.

Además gracias a la integración con Travis, se ha podido mantener un nivel de calidad del software al requerir que pase todas las pruebas satisfactoriamente de forma previa a cada despliegue.

2.2. Behaviour-Driven-Development

El Desarrollo guiado por comportamiento, conocido como BDD por sus siglas en inglés, es un proceso de desarrollo de software a partir de especificaciones de funcionalidades o requisitos escritas en lenguaje natural, que son conocidas como "Historias de usuario".

A pesar de ser lenguaje natural, es decir, sin tecnicismos y que puedan ser leídos y comprendidos por cualquier tipo de usuario, es recomendable que las historias de usuario sigan un estándar de escritura. En este caso se utilizará la sintaxis Gherkin, lo cuál permitirá llevar esas historias de usuario al código del proyecto, y a su vez poder conectarlas con una herramienta que automatizará los tests de integración, que se encargarán de validar el correcto comportamiento de las funcionalidades de la aplicación.

Es decir, las propias historias de usuario serán los tests de integración que debe pasar la aplicación para validar su comportamiento.

2.3. Test-Driven-Development

En el proceso de desarrollo de software, la fase de pruebas de la aplicación es vital para que el producto pueda ser desplegado y salir al mercado con la verificación de que

las funcionalidades y los requisitos están correctamente implementados. Esta fase además asegura un cierto nivel de calidad en el código, ya que se reducen riesgos de fallo y malas prácticas gracias a las pruebas y revisiones de código.

Sin embargo, el proceso de desarrollo no debe limitarse a empezar a escribir líneas de código y luego comprobar que nada falla. El desarrollo guiado por tests sugiere que sean éstos los que guíen el desarrollo del software.

Mediante esta técnica, el ciclo de vida de una funcionalidad o una historia de usuario comienza con la definición de los tests que van a probar que funciona. Aunque resulte en un principio contradictorio probar algo que no está aún implementado, gracias a ellos podemos guiar nuestro proceso de desarrollo hacia lo que queremos conseguir, sin desviarnos del objetivo de la historia de usuario y aplicando una modularización sobre la funcionalidad global. Al contrario que el ciclo de desarrollo original en el que veríamos los requisitos como algo global a implementar, con TDD lo podemos enfocar paso por paso, test por test, hasta llegar a nuestro objetivo final.

De esta forma además nos aseguramos de que toda funcionalidad nueva tendrá sus tests asignados, y reduciremos así la cantidad de código sin probar. También nos ayuda a evitar hacer más de lo necesario, siguiendo un principio acuñado por el ingeniero *Kelly Johnson*: **Keep it simple and straightforward**, mediante el que reduciremos la complejidad del código y eliminaremos malas prácticas, conocidas como *bad smells*. Además, teniendo en cuenta que en un proceso de ingeniería de software los tiempos y recursos son limitados, esto puede llegar a ser algo decisivo en el éxito del proyecto.

Esta filosofía no tiene sentido sin tests automatizados, ya que serán los que nos guíen durante el proceso de desarrollo, y a su vez definan los criterios de aceptación para cada historia de usuario.

2.4. Estimación basada en Sprints

En un proceso de desarrollo ágil, la estimación de recursos y de los plazos del proyecto se realizan de forma diferente a la habitual. Debido al modelo de desarrollo iterativo e incremental, no podemos realizar un cálculo detallado del coste de recursos que serán necesarios para la realización del proyecto en su totalidad. Siempre debemos tener en cuenta que dicho coste no sigue un patrón fijo, y la variabilidad de todos los factores implicados en el desarrollo del proyecto es impredecible.

Mediante este nuevo sistema de planificación y estimación, los requisitos del proyecto pasarán a ser historias de usuario, que serán pequeñas representaciones de partes del proyecto que aportarán un valor a este, utilizando un lenguaje cercano al usuario y sin detalles técnicos.

Estas historias de usuario serán repartidas en intervalos de tiempo, conocidos como **sprints**, lo cual nos permitirá mantener una visión del proyecto a corto plazo. Cada sprint tendrá una duración de entre dos y cuatro semanas, típicamente.

De esta manera estimaremos cada pequeño intervalo, al principio de cada sprint, asignando a cada historia de usuario un número determinado de **puntos-historia**, basado en el esfuerzo que será requerido para su realización. Dicha estimación es calculada por los miembros del equipo del proyecto en cuestión, ya que una de las características de los equipos ágiles es su capacidad de auto-organización.

Una vez estimados los puntos de cada historia de usuario, estas pueden ser introducidas en el sprint siguiente en función de la velocidad establecida para el proyecto. La velocidad consiste en una métrica calculada dinámicamente según el trabajo realizado en los sprints previos, que permite predecir la carga de trabajo en **puntos-historia** asignable al equipo de desarrollo.

El resto de historias no planificadas para el siguiente sprint, quedarán en una pila denominada **Sprint Backlog**, la cuál podrá ordenarse según las prioridades del proyecto para la definición del sprint posterior.

CAPÍTULO 3

Herramientas y tecnologías utilizadas

Las herramientas y tecnologías utilizadas, quedan englobadas en cinco categorías, conforme a la utilidad que han proporcionado para llevar a cabo este proyecto.

3.1. Gestión del proyecto

Este grupo de herrramientas engloba tanto la aplicación de gestión y seguimiento del proyecto, como servicios para el control del flujo de trabajo como son los sistemas de control de versiones, o plataformas de despliegue, las cuales permiten gestionar de una forma cómoda el paso a producción de una nueva versión del producto.

3.1.1. Pivotal Tracker

Pivotal Tracker [30] es una herramienta para la gestión de proyectos basados en iteraciones (sprints) en desarrollos ágiles. La elección de esta herramienta, además de por la recomendación de mi tutora, viene dada por ciertas ventajas que ofrece:

- Es gratuita para pequeños proyectos, con un máximo de 3 colaboradores.
- Es una herramienta SaaS, es decir, es un servicio de software ofrecido a través de la nube permitiendo así el trabajo con equipos descentralizados.
- Permite organizar las historias de usuario de forma apilada y mediante métricas como son la velocidad del sprint y los puntos/historia.

- Histórico de actividad, que nos permitiría realizar un análisis de trazabilidad de nuestro proyecto.

Las historias de usuario, además de estar apiladas en un orden, se dividen entre diferentes pilas según la planificación y su estado de desarrollo.

En primer lugar hay una pila denominada *Icebox*, en la que se almacenarán todas las nuevas historias de usuario, es decir, es la pila en la que añadiremos el resultado de una lluvia de ideas sobre los requisitos y funcionalidades que formarán parte de nuestra aplicación.

En segundo lugar, nos encontramos con el *Sprint Backlog*, en el cuál iremos colocando las historias generadas en la pila anterior. Si bien en el *Icebox* las historias de usuario no precisan de un orden determinado, en esta nueva pila sí que deberemos mantener un orden basado en prioridades, manteniendo en lo alto de la pila las historias de mayor prioridad para el proyecto.

A continuación se encuentra la pila de *Current*, en la que Pivotal Tracker irá añadiendo dinámicamente y en orden las historias de usuario al principio del sprint, según su estimación en puntos y la velocidad de sprint. También se permite la modificación tanto del orden como de las historias a introducir en el sprint.

Por último tenemos la pila *Done*, que constará de un histórico ordenado por sprints de todas las historias realizadas a lo largo del proyecto.

3.1.2. Git

Sistema de Control de Versiones diseñado por Linus Torvalds, el creador del sistema operativo Linux. De hecho, el desarrollo del núcleo de Linux utiliza el sistema **Git**.

Git [13] permite un flujo de trabajo distribuido, es decir, cada desarrollador puede tener tanto el rol de nodo de un repositorio, contribuyendo a éste, como servir de repositorios sobre el que puedan contribuir otros desarrolladores. Cada uno de los desarrolladores dispondrá del histórico completo de desarrollo del repositorio, pudiendo consultar establecer su flujo de ramas y contribuciones de manera sencilla.

Gracias a la gestión de ramas, Git permite un desarrollo no lineal, que facilita enormemente el trabajo en equipo y la rapidez del proyecto.

3.1.3. Github

Github [14] es una plataforma de desarrollo colaborativo, también denominado *forja*, para proyectos de código abierto que utilizan el sistema de control de versiones Git. Permite aprovechar toda la funcionalidad de Git a la vez que permite a los desarrolladores conectarse entre sí a modo de red social.

Además, Github ofrece una completa gestión de incidencias o *issues*, que fomenta el trabajo colaborativo y el control del desarrollo de los repositorios.

Para el proyecto aquí tratado, el repositorio utilizado para el desarrollo ha sido el siguiente:

https://github.com/malkomich/busy

3.1.4. Heroku

Plataforma como despliegue (*PaaS*), que permite el despliegue en la nube de aplicaciones software.

Su funcionamiento se basa en unas unidades de cómputo denominadas *dynos*, que básicamente son contenedores ligeros de Linux, aislados unos de otros, destinados a ejecutar comandos únicos.

Para este proyecto, la dirección de despliegue sobre Heroku [34] ha sido la siguiente:

https://busy-web.herokuapp.com

3.2. Diseño y desarrollo

En esta categoría encontramos las herramientas de desarrollador para el diseño de la aplicación, tanto de diagramas y esquemas necesarios como de la interfaz de usuario, y del proceso de desarrollo del código. En este último podemos encontrar entornos de desarrollo, denominados *IDEs*, tanto para el desarrollo de la aplicación como para la documentación del proyecto.

3.2.1. Draw.io

Herramienta para el diseño y creación de diagramas, que funciona también como servicio en la nube ó SaaS.

Esta herramienta permite además conectar con diferentes plataformas de almacenamiento en la nube, y mantener una sincronización dinámica de los diagramas creados, al igual que hace Google Drive.

Draw.io [24] permite crear varios tipos de diagramas, entre los cuales están:

- Diagramas de clases para el modelo de dominio
- Diagramas de casos de uso
- Diagramas de secuencia
- Diagramas Entidad-Relación
- Diagramas de colaboración o comunicación de componentes
- Diagramas de flujo

Además, nos da la facilidad de exportar nuestros diagramas creados a multitud de formatos como son PDF, formatos de imagen, o formato de lenguajes de marcas como HTML o XML.

El diseño del diagrama entidad-relación, que ha sido desarrollado mediante esta herramienta, se puede visualizar a través del siguiente link:

https://drive.google.com/file/d/0Bxc-ojQIGgAcRHBPUDBmcHF6SGs/view?usp=sharing

3.2.2. Balsamiq Mockups

Prototipado y creación de interfaces gráficas de usuario. Mediante una herramienta de prototipado o *mocking*, podemos llevar a cabo el diseño de la interfaz gráfica de nuestras aplicaciones. De esta manera podemos tener una visión aproximada del producto final que queremos lograr, lo cuál nos aportará una visión de objetivos y de posibles fallos de diseño antes de llevar a cabo el desarrollo.

Gracias a ello, podremos realizar cambios y guiar mejor el proyecto de una forma previa, además de mantener el foco hacia un objetivo de diseño fijado durante el proceso de desarrollo.

Balsamiq Mockups [2] nos facilita ese proceso de prototipado y agiliza la creación de bocetos. Su gran variedad de elementos a añadir, y su flexibilidad a la hora de personalizarlos y moldearlos al estilo de tu aplicación, hace que sea todo más sencillo y permite un diseño muy detallado de los mockups.

3.2.3. Eclipse

Eclipse [10] es un entorno de desarrollo integrado, desarrollado principalmente en Java y disponible para desarrollar en una gran multitud de lenguajes gracias a la amplia biblioteca de *plugins* disponibles.

Es multiplataforma ya que está disponible para sistemas operativos Windows, Mac OS X, Solaris, y para cualquier distribución de Linux. Su licencia es pública, puesto que es un proyecto desarrollado por la comunidad *open source* denominada **Eclipse Foundation**.

Esta herramienta ha sido utilizada para todo el desarrollo de la parte de servidor, ya que aporta multitud de tareas automáticas que facilitan la escritura de código Java, además de las herramientas que incorpora para la depuración de código y para monitorizar la ejecución de los tests automáticos de forma muy visual.

3.2.4. Atom.io

Atom [1] es un editor de texto de código abierto, cuyo proyecto fue lanzado por los desarrolladores de la plataforma GitHub. Es un editor multiplataforma y basado en módulos, es decir, su funcionalidad es incremental en base a los módulos o paquetes que se añadan a la instalación de serie. Dichos módulos también son de código libre y son creados por la comunidad de desarrolladores de GitHub.

Las funciones de Atom son similares a las de algunos IDE, pero presumiendo de una mayor rapidez de proceso y de una instalación más liviana.

Este editor ha sido utilizado para el desarrollo de la interfaz gráfica de la aplicación, debido principalmente a su coloreado de sintaxis y su mayor ligereza, cualidades que amenizan de forma natural la escritura del código.

3.2.5. ShareLaTeX

Editor online de documentos LaTeX.

Mediante ShareLatex [39] podemos construir documentos bien estructurados y de diseño altamente configurable, gracias a la construcción de macros y órdenes, la gestión de referencias y de bibliografía, sus tablas de contenidos, la modularidad que ofrece, y la calidad de impresión que logra, dando un aspecto de documento bien organizado y profesional.

Esta herramienta para documentos en formato LaTeX[27], tiene la ventaja de ser una aplicación SaaS, es decir, una aplicación 'Software como Servicio' que podemos encontrar online, y por tanto es multiplataforma.

3.3. Testing

A continuación se mencionarán herramientas y frameworks destinados al proceso de automatización de pruebas, lo que permite tener un control sobre el correcto comportamiento de la aplicación mediante la ejecución automática de un conjunto de tests.

3.3.1. JUnit

Framework para la realización de pruebas unitarias en Java.

JUnit [21] nos permite mantener controlado el buen funcionamiento de todas las partes de la aplicación de una forma aislada, lo que nos permite encontrar los focos de error de una manera efectiva a la vez que asegura la calidad del código.

3.3.2. Mockito

Framework Java que permite simular objetos y configurar su comportamiento y estado deseado para realizar pruebas de caja blanca en aislamiento.

Mediante Mockito [8] podemos aislar diversos componentes de nuestra aplicación para llevar a cabo la pruebas unitarias sin depender del funcionamiento de otras partes o componentes de dicha aplicación. Para verificar que el comportamiento de los objetos simulados ha sido el correcto, podremos especificar una serie de expectativas o resultados a obtener, o simplemente comprobar que no ha producido error alguno en el caso de prueba.

3.3.3. DBUnit

Herramienta que extiende JUnit para el testing de bases de datos. Gracias a ella podemos realizar test unitarios de persistencia, que manipulen los datos de la base de datos. Nos permite realizar estas pruebas de manera aislada ejecutando las operaciones contra la base de datos de forma transaccional.

Además, su API nos aporta una forma de precargar la base de datos, mediante archivos en formato XML denominados *datasets*. Dichos archivos se traducirán en sentencias de inserción, de forma abstracta al sistema de gestión de base de datos utilizado.

De la misma forma, con DBUnit [40] también podremos hacer un *rollback* de los datos almacenados, es decir, podremos restablecer los datos introducidos en nuestra base de datos como si no hubiera pasado nada, para mantener aislado cada caso de prueba y así evitar que influyan en los resultados de otras pruebas.

3.3.4. Selenium WebDriver

Herramienta de automatización de tests funcionales en aplicaciones web mediante un controlador del navegador.

Nos permite controlar un driver para simular acciones, eventos JavaScript y acceder a cualquier elemento del árbol *DOM* en el que se estructura cada página web, para poder comprobar el funcionamiento de la aplicación como si se estuviera probando en un entorno real.

Selenium [38] nos ofrece la posibilidad de emular además la mayoría de los navegadores más utilizados, como son Chrome, Firefox, IExplorer y Opera, gracias a su amplia variedad de drivers.

3.3.5. Cucumber

Herramienta de testing que interpreta historias de usuario en formato *Gherkin*, vinculándolas con los tests de aceptación del desarrollo basado en comportamiento (*BDD*).

Cucumber [23] nos permite ejecutar secuencialmente los casos de prueba que definen cada uno de los escenarios de una historia de usuario, permitiendo integrar en los tests desde la interacción con la interfaz hasta el correcto funcionamiento del *back-end* de la aplicación.

3.4. Back-End

En esta sección, se mostrarán las tecnologías utilizadas para llevar a cabo el desarrollo del lado de servidor de la aplicación.

Dichas tecnologías engloban, por un lado, los frameworks de aplicación y las herramientas de gestión del ciclo de vida y dependencias del proyecto. También se incluirán las tecnologías para la implementación de persistencia en la aplicación, como son los sistemas de gestión de base de datos, y las *APIs* que facilitan su conexión con los diferentes lenguajes de programación.

3.4.1. Maven

Herramienta de gestión de dependencias y de construcción de proyectos Java.

Maven [9] utiliza archivos en formato **XML** denominado *Project Object Model (POM)* para definir las dependencias del proyecto, construir y configurar elementos o plugins utilizados, asignar tareas en las distintas fases de los ciclos de vida del proyecto, y personalizar multitud de propiedades del proyecto, incluso definiendo diferentes perfiles para una mayor precisión de configuración.

Esta herramienta, desarrollada por la fundación Apache, es similar a su antecesora, Apache Ant. Sin embargo, Maven proporciona una mayor sencillez de configuración de los ciclos de vida del proyecto, y además actúa como en framework que gestiona de manera automática todas las propiedades y dependencias del proyecto.

Maven nos proporciona gran parte del trabajo que realizaba Apache Ant, previamente configurado, facilitando y acelerando la configuración del proyecto, a cambio de estandarizar nuestro proyecto, ya que esta herramienta sigue el paradigma de *Convención sobre Configuración*.

Podemos ver el archivo **POM** de configuración del proyecto en la siguiente URL:

3.4.2. Spring

Framework de desarrollo de aplicaciones Java, con un servidor web embebido y contenedor de inversión de control que nos facilita la inyección de dependencias, además de multitud de módulos que nos proporcionan herramientas y una gran funcionalidad avanzada para el desarrollo de aplicaciones Java.

Spring [31] surgió para sintetizar las funcionalidades de Java EE, el API de Java Servlets y los Enterprise Beans, y crear un framework para ofrecer esa funcionalidad de una forma más sencilla y consistente para la construcción de aplicaciones en Java.

3.4.3. JDBC

Java DataBase Connector, es el API que nos permitirá conectar el sistema de base de datos con nuestra aplicación en lenguaje Java. Esta conexión ofrece la posibilidad de ejecutar sentencias contra la base de datos, en este caso sentencias relacionales SQL, a nivel de aplicación, de manera que podamos interactuar con los datos persistentes de nuestra aplicación.

Esta integración se lleva a cabo de forma independiente tanto al sistema operativo que aloja la aplicación, como al sistema gestor de la base de datos. Las operaciones realizadas contra la base de datos se ejecutan abstrayendo al usuario del proceso de serialización y deserialización de los bytes de datos, infiriendo además los formatos de codificación necesarios para mantener la consistencia de dichos datos.

La API de JDBC [26] nos proporciona un servicio llamado *DriverManager*, el cual se encarga de crear conexiones únicas con la base de datos. A partir de estas conexiones se crean una serie de sentencias en un formato entendible por el sistema de gestión de la base de datos. Una vez que dichas sentencias son ejecutadas, obtenemos un conjunto de resultados que pueden ser leídos de forma iterativa para llevar a cabo la deserialización de esos datos.

3.4.4. PostgreSQL

Sistema de gestión de base de datos relacional, de código abierto y desarrollado bajo una licencia libre por el grupo de desarrolladores *PostgreSQL Global Development Group*.

PostgreSQL [32] soporta transacciones **ACID**, cuyas siglas hacen referencia a cuatro características que son:

- Atomicidad
- Consistencia
- Aislamiento (*Isolation*)
- Durabilidad

Gracias a ello, este sistema de base de datos permite el acceso a los datos sin ningún tipo de bloqueo entre las operaciones y asegura una visión consistente de los datos sin riesgo de pérdida de información.

Además, ofrece funcionalidades como son la conexión mediante claves foráneas, el uso de disparadores de eventos para aportar una lógica interactiva entre nuestros datos, el uso de vistas para personalizar a fondo nuestras consultas, la herencia de tablas, el uso de transacciones distribuidas y una gran variedad de tipos de datos.

La búsqueda de un sistema gestor de base de datos se ha centrado en los modelos relacionales para poder aplicar los conceptos aprendidos durante mi proceso de aprendizaje a lo largo de los años en este grado.

La elección de **PostgreSQL** se debe principalmente al alto rendimiento y la gran escabilidad que aporta en comparación a otros sistemas. Además, su soporte de operaciones transaccionales hace que se imponga su uso sobre otras alternativas. En el caso de MySQL, también se dispone de un módulo que soporta transacciones, denominado *InnoDB*. Sin embargo, PostgreSQL presume de un mayor rendimiento y de más flexibilidad a la hora de realizar consultas complejas contra la base de datos.

3.5. Front-End

Las herramientas de la capa de presentación, o *Front-End* de la aplicación, engloban todo el software que contribuye a la mejora del diseño de la interfaz gráfica de la aplicación. Entre estas herramientas podemos encontrar frameworks de interfaces web, librerías JavaScript para el renderizado de elementos HTML, o motores de plantillas para entornos web.

3.5.1. Bootstrap CSS

Framework web, de código abierto y desarrollado por los creadores de *Twitter*, que está destinado al diseño del *front-end* de las aplicaciones. Nos proporciona una serie de plantillas y guías de estilo para moldear nuestra interfaz gráfica a partir de un diseño *responsive* y adaptado al usuario.

Bootstrap [28] ofrece hojas de estilo, ficheros JavaScript y una multitud de componentes HTML, que hacen que la interfaz gráfica de cualquier página o aplicación web sea totalmente configurable de una forma sencilla, abstrayendo al desarrollador de gran parte de toda la complejidad que supone el diseño web.

3.5.2. Thymeleaf

Thymeleaf es un motor de plantillas HTML, con una gran flexibilidad y una baja dependencia con la capa de controlador, lo que permite adaptar nuestra aplicación para funcionar tanto de forma dinámica como de forma estática.

Thymeleaf [41] tiene una potente integración con el framework Spring, y su flexibilidad permite construir plantillas altamente configurables, que pueden ser extendidas para ofrecer una gran modularidad en la capa de presentación de tu aplicación, funcionando casi a modo de framework de plantillas en formato HTML5.

Ofrece soporte para internacionalización de mensajes, con lo que podremos definir archivos de traducciones de mensajes en diferentes idiomas, los cuáles serán conectados a la aplicación por medio de la respectiva configuración en la lógica del framework Spring, y utilizar dichos mensajes posteriormente a partir de su código identificativo, siendo el motor *Thymeleaf* el encargado de interpretar dicho código y mostrar la traducción correcta dependiendo del lenguaje del navegador.

También añade una funcionalidad conocida como *form binding*, o vinculación de datos de formularios. Gracias a esto podremos crear vínculos de doble sentido sobre los elementos de entrada de un formulario, con lo que conectará los datos procedentes de una respuesta de tipo GET del servidor de manera correcta en el formulario, y enviará dichos datos con los correspondientes nombres de parámetros mediante las peticiones POST.

Incluye además gran variedad de operadores que aportan una lógica a nivel de presentación, y una alta configurabilidad de parámetros, permitiendo incluso definir parámetros personalizados dentro de cada elemento del modelo de objetos del documento, conocido como *DOM*.

3.5.3. JQuery

Una de las bibliotecas de JavaScript más utilizada por los desarrolladores. Nos ofrece la posibilidad de manipular cada elemento del árbol de objetos *DOM*, eventos sobre dichos elementos, y llamadas tanto síncronas como asíncronas que modifican la lógica de los elementos de la interfaz gráfica, de una forma muy simplificada respecto al modelo de interacción habitual de JavaScript.

Se trata de un único fichero JavaScript que aporta gran multitud de funciones para tratar con los elementos de la vista. Posee además una compatibilidad muy alta con todo tipo de navegadores, limitándose a escasas funciones o métodos cuya compatibilidad no es completa con todos, pero siempre con alternativas equivalentes o similares en funcionalidad.

El proyecto de **JQuery** [11] es de código abierto, y posee una excelente comunidad de soporte, lo que facilita la resolución de problemas, la optimización del código y la continua mejora de su funcionalidad. Gracias a dicha optimización ofrece también una gran rapidez y efectividad en sus operaciones.

Posee también una completa integración con AJAX, para poder implementar un comportamiento asíncrono de nuestra aplicación respecto a las conexiones contra el servidor. Por último, esta biblioteca presume de tener una altísima cantidad de plugins basados en ella, por lo que el crecimiento funcional de nuestra aplicación puede verse multiplicado gracias a ello.

CAPÍTULO 4

Arquitectura base de la aplicación

4.1. Visión general

En un vistazo rápido, la aplicación de ***buSy*** se divide en tres capas: la interfaz gráfica, la capa de servicios o de lógica de negocio, y la capa de persistencia.

Cada una de ellas se comunica con la siguiente a través de interfaces, en el sentido mostrado en la Figura 4.1, y todas ellas mantienen una relación unilateral con las clases del modelo de dominio.

Gracias a esta arquitectura, evitamos el acoplamiento entre partes de la aplicación destinadas a ámbitos de operación completamente distintos.

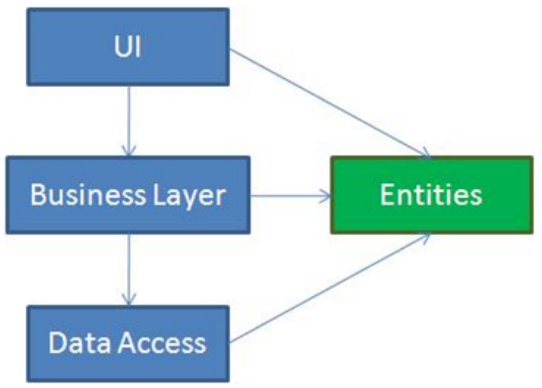


Figura 4.1: Arquitectura basada en 3 capas

Para la capa de interfaz gráfica se ha utilizado un patrón Modelo-Vista-Controlador, en el que mediante un controlador que recibe y envía peticiones HTTP se comunica con la parte cliente implementada a partir de "Java Server Pages", o simplemente JSP. Estos componentes de vista tienen además una lógica implementada mediante JavaScript para el renderizado de los elementos y para comunicarse con el controlador en el lado del servidor a través de peticiones asíncronas con AJAX[1].

El controlador será el encargado, además de gestionar las peticiones GET y POST entrantes y establecer un flujo de operación, de enviar los datos del modelo de la aplicación a la vista para que ésta pueda renderizar los datos correctamente.

Dicho controlador realizará llamadas a una interfaz de la lógica de negocio, para operar con los objetos del modelo de dominio abstrayéndose de la implementación de dicha lógica.

Las clases de la capa de negocio serán las que servirán a modo de intermediarias entre la capa de la vista y la capa de persistencia, tomando las respectivas decisiones para ejecutar correctamente las operaciones solicitadas por el controlador según la lógica de negocio deseada. Al tratarse de una aplicación en sus primeras fases de desarrollo, esta capa no constará de una extremada complejidad, sin embargo será la capa donde recae la responsabilidad de operar sobre las clases del dominio.

Los servicios, a su vez, se comunican con la capa de persistencia a través de interfaces que abstraen el manejo y persistencia de datos. La capa de persistencia ha sido implementada sobre una base de datos relacional con el DBMS[2] PostgreSQL, un sistema de licencia libre y código abierto. Para establecer una comunicación entre la aplicación y el sistema de base de datos, y poder persistir y recuperar objetos del dominio, se utiliza Spring JDBC, un modulo del framework Spring integrado con el API JDBC que gestiona las transacciones sobre cualquier sistema de base de datos relacional.

4.2. Spring Framework

Spring es un framework desarrollado por la compañía Pivotal Software, encargados de multitud de proyectos como por ejemplo la herramienta de seguimiento de este TFG, Pivotal Tracker.

La principal utilidad que proporciona el uso de Spring y el concepto a partir del cuál se crea dicho framework, es la **inversión de control**. Spring proporciona una infraestructura de soporte para el desarrollo de aplicaciones en Java, con multitud de funcionalidades y

[1]AJAX: Asynchronous JavaScript and XML
[2]DBMS: Database Management System (Sistema de gestión de base de datos)

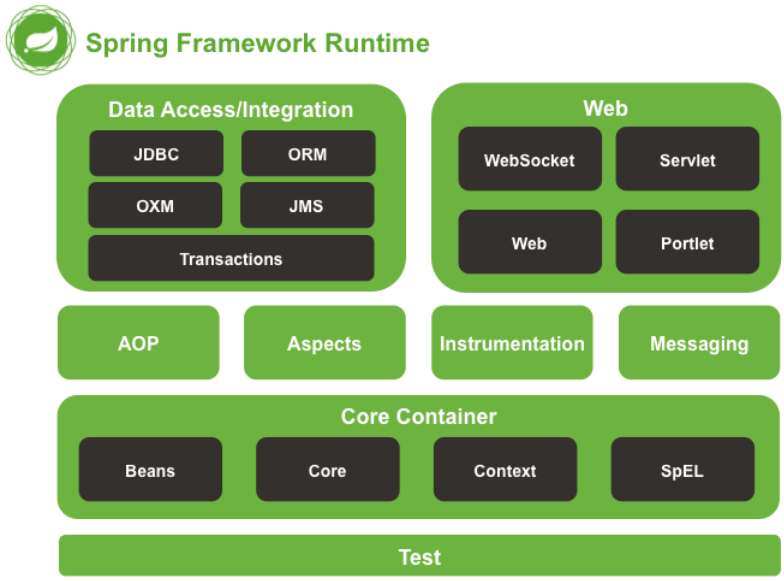


Figura 4.2: Arquitectura de módulos de Spring

mejoras que serán explicadas a continuación.

El núcleo del framework Spring se divide en varios módulos destinados a diferentes ámbitos dentro de una aplicación, como podemos observar en la Figura 4.2.

En primer lugar y como se ha comentado anteriormente, para aprovechar las funcionalidades del framework se utiliza la **inversión de control**, un patrón de diseño originalmente expuesto por Martin Fowler, quién lo definió como *Principio de Hollywood*, tras una anécdota en la que se le decía a un actor la frase *No nos llames, nosotros le llamaremos*. Y efectivamente eso es lo que hace la inversión de control, en vez de seguir un flujo de ejecución en la aplicación, es el framework el que invocará el código necesario como respuesta a ciertos eventos, y se encargará de la gestión de instancias en vez de ser la aplicación la que cree los nuevos objetos.

De la gestión de instancias realizada por el framework, se deriva otro concepto presente en Spring y no menos importante, denominado **inyección de dependencias**. Se trata de un patrón de diseño en el que básicamente una clase A inyecta objetos en otra clase B, en lugar de ser la clase B la encargada de crear dichos objetos. Spring posee un contenedor de instancias POJO, denominadas **Beans**, que inyecta en cada objeto las instancias necesarias según la configuración definida. Esta configuración se puede definir de dos formas diferentes: mediante archivos de configuración de contexto en XML, o mediante el uso de anotaciones

de las librerías del framework. En este caso se ha prescindido por completo de la utilización de ficheros XML en favor de las anotaciones, una funcionalidad de Java implementada desde la versión 1.5 del JDK.

Otra de las funcionalidades que aporta Spring, mencionada brevemente en el apartado anterior, es que facilita el acceso a datos y la gestión de transacciones. El framework se encarga de tomar y de soltar automáticamente los recursos para las operaciones con la base de datos, las cuales se se ejecutan como operaciones indivisibles y aisladas, y terminando siempre con un resultado de éxito o de error, evitando así dejar el sistema de base de datos en un estado inconsistente. Además proporciona multitud de formas de acceso y de operaciones contra la base de datos a partir de su abstracción Spring JDBC, en cuya librería también encontramos clases que facilitan el mapeo de los datos recibidos a instancias de nuestro modelo de dominio.

Respecto a la internacionalización de nuestra aplicación, Spring Framework nos permite tener una serie de archivos de configuración en los que almacenar todos los mensajes que queramos mostrar, en distintos idiomas. Se trata de una mejora del API JMS[3], y nos permite inyectar de una forma transparente el mensaje que queramos mostrar, siendo el framework el encargado de seleccionar dicho mensaje del archivo correspondiente según el idioma configurado en el cliente (en este caso, en el navegador web).

Spring posee una amplia guía de referencia, que podemos encontrar aquí [20].

4.3. Patrón MVC

Como se ha explicado al principio de este capítulo, la parte de de presentación de la aplicación está diseñada siguiendo el patrón Modelo-Vista-Controlador, cuya aplicación al proyecto en cuestión se muestra mediante la Figura 4.3. Según este patrón, la vista de la aplicación y la interacción del usuario con ella quedan aisladas del manejo de los datos del modelo y la gestión de eventos y peticiones contra la parte del servidor. Así, la parte de vista queda encargada únicamente de la presentación de esos datos, y de pedir nueva información. El controlador, por su parte, procesará esas peticiones y enviará los datos correspondientes de vuelta a la vista, que podrán ser encapsulados en objetos del modelo para permitir a la vista acceder a sus propiedades.

Para definir los controladores, se han definido ciertas clases con la anotación *@Controller* del framework Spring, para indicarle que esas clases van a ser encargadas del flujo de peticiones HTTP. A su vez, dentro de cada controlador definimos métodos que reciban esas peticiones, y para ello definimos dichos métodos con la anotación *@RequestMapping*,

[3]JMS: Java Message Service

indicando además la URI de la petición a la que debe atender el método, y opcionalmente el tipo de petición que debe atender, típicamente GET ó POST.

Una vez definidos los métodos que atiendan las peticiones del cliente, hay varias formas de intercambiar información entre el controlador y la vista.

A la hora de recibir datos, podemos recibirlos a partir de un formulario y representarlos en una instancia del modelo mediante la anotación *@ModelAttribute*, recibir parámetros incrustados en la URI por medio de *@RequestParam*, atributos volátiles almacenados en la sesión y utilizados en las peticiones de redirección entre controladores por medio de la clase *RedirectAttributes*, o incluso podemos recibir la petición como un objeto *WebRequest* para procesar los metadatos de dicha petición.

Si lo que queremos es almacenar los datos, Spring MVC nos proporciona una clase llamada *Model*, que básicamente se trata de una estructura Map en la que almacenamos todo tipo de objetos, típicamente serializables, con diferentes ámbitos de aplicación. Dicho ámbitos, que en la tecnología original JEE se limitan a ámbito de petición o *request scope* y ámbito de sesión o *session scope*, se dividen en 5 dentro del framework de Spring, ya que además de los dos mencionados añade *singleton* para instancias únicas y *prototype* para un número indefinido de instancias, los cuales son usados además para los componentes ó *beans* del **contenedor de beans** del framework; y por último disponemos de un quinto ámbito, *global session*, que engloba todos los datos a nivel de la aplicación, es decir, los datos compartidos por todos los usuarios.

4.4. Construcción del entorno de desarrollo

El primer paso ha sido la creación de un repositorio git, alojado en la plataforma online GitHub. También se ha utilizado Maven, una herramienta para la gestión de las dependencias y la configuración de las tareas de construcción y despliegue de proyectos Java, basada en Apache Ant y con formato XML.

Esta herramienta se encarga de gestionar las dependencias del proyecto, organizar las estructuras de directorios y simplificar las operaciones de compilación, empaquetado, ejecución de tests y despliegue entre otras mediante tareas predefinidas y configurables mediante la definición de plugins. Además, facilita la reutilización de configuraciones, permite definir propiedades incluso externamente al proyecto y permite establecer perfiles para mantener distintos procesos de construcción. Todo esto y mucho más desde un único fichero en el directorio raíz: **pom.xml**

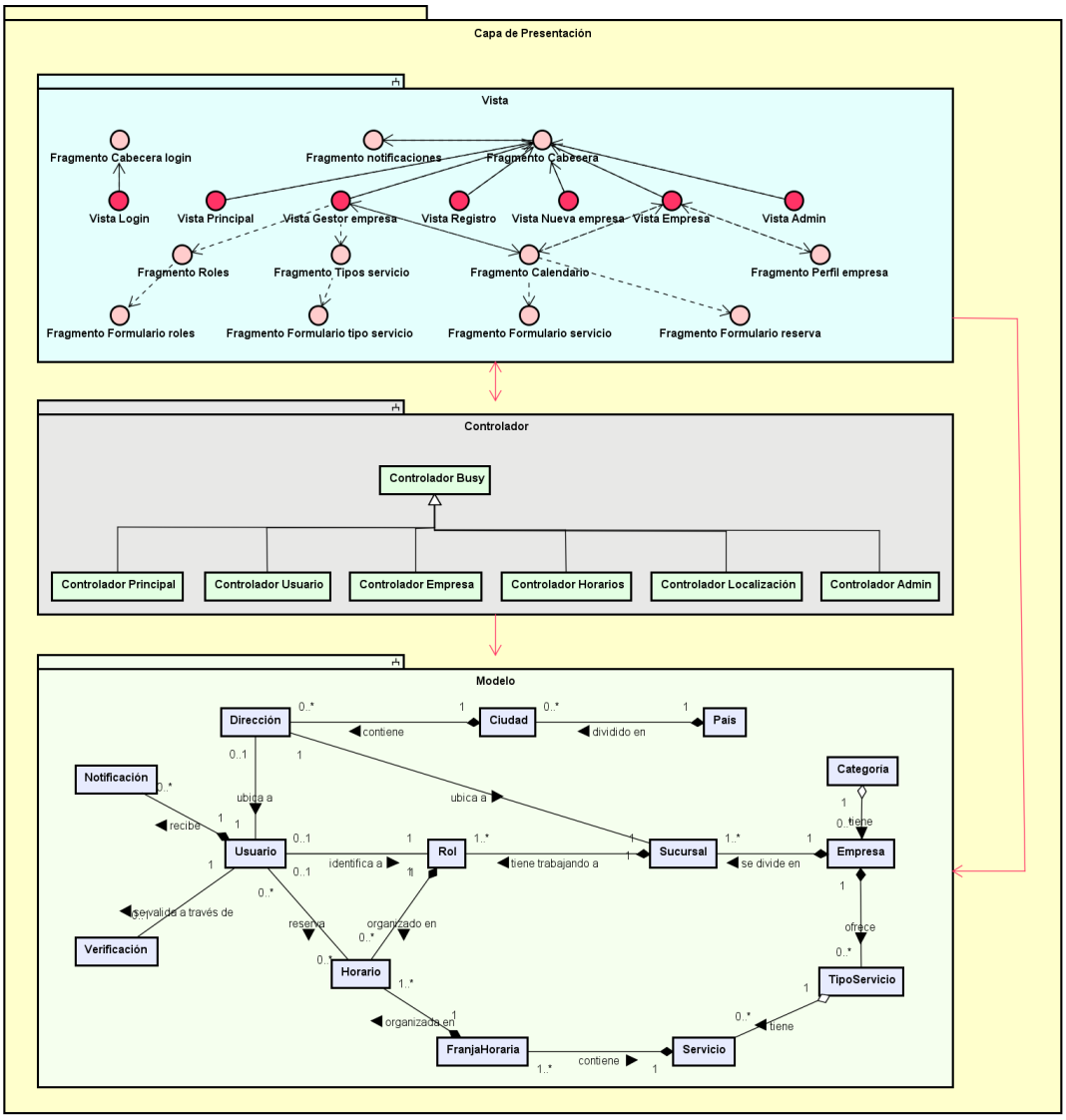
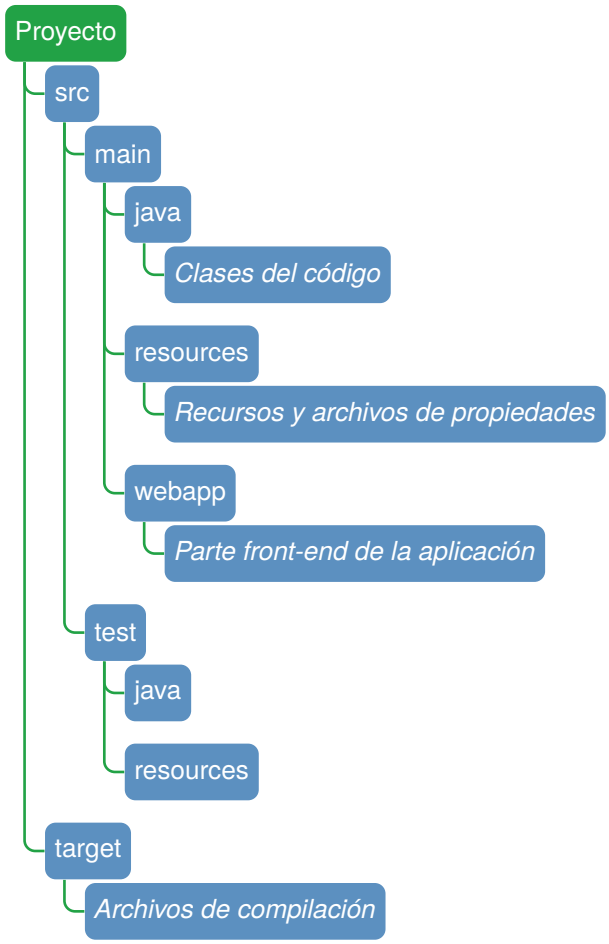


Figura 4.3: Diagrama Modelo-Vista-Controlador



Respecto al esqueleto de directorios se ha seguido el estándar de Maven, y se ha dividido la estructura de los ficheros fuente en cinco módulos diferenciados: un módulo para el código Java de la aplicación; otro para el código web de la vista basado en ficheros JSP; otro para los recursos de la aplicación, como son los archivos de configuración y de mensajes de Spring ó los recursos estáticos de la vista (CSS, JavaScript, Imágenes...); otro módulo para el código Java de los tests unitarios y de integración de la aplicación; y un último módulo destinado a los recursos para los test, como son los scripts y datasets de modificación de la base de datos ó las definiciones de User Stories en formato *Gherkin*.

Además, Maven lee toda la configuración de un archivo llamado *pom.xml* que debemos alojar en la carpeta raíz del proyecto. En este fichero debemos poner obligatoriamente unas características que son el identificador de la organización o grupo propietario del proyecto (groupId) y el identificador del proyecto (artifactId).

CAPÍTULO 5

Diseño de User Stories

En este capítulo se abordarán las historias de usuario que se han llevado a cabo en el desarrollo de este proyecto. A la hora de dividir dichas historias se han definido algunas como **Epics**, lo que indica que debido a su gran tamaño ha de descomponerse en varias historias de usuario más pequeñas para que puedan ser planificadas dentro de una iteración o **sprint**.

Esto nos ayudará a agrupar las historias de usuario, manteniendo una organización por funcionalidades o módulos, que paso a explicar a continuación.

5.1. Historias de usuario sobre *Gestión de cuentas*

En esta historia de usuario identificada como *Epic*, se han tratado las funcionalidades sobre cuentas de usuario, relacionadas con temas de autenticación y de gestión del perfil de usuario. Han sido detectadas cuatro historias de tamaño adecuado para ser estimadas dentro de un sprint. Dichas *User Stories* son:

- Identificación del usuario o administrador (Figura 5.1).
- Registro de usuario (Figura 5.3).
- Cierre de sesión del usuario (Figura 5.4).

Para la primera historia de usuario, descrita mediante la sintaxis del lenguaje *Gherkin* en la Figura 5.1, se ha desarrollado un primer diseño sobre los modelos de datos, lo cuáles han

Como usuario registrado o administrador
Para poder utilizar toda la funcionalidad de la aplicación
Quiero poder identificarme en el sistema con mis datos de cuenta

- **Antecedentes**: Empezar desde la página principal de ***Busy***
  Dado que estoy en la página principal de ***Busy***
  Cuando hago click en "*Sign in*"
- **Escenario**: Identificación de usuario con éxito
  Dado que estoy registrado
  Cuando introduzco mis datos en la pantalla de identificación:

Email	Contraseña
"user@domain.com"	"pass"

Entonces debería ver mi página principal
Y debería ver mis opciones de usuario
Pero no debería ver las opciones "*Sign in*" o "*Sign up*"

- **Escenario**: Identificación de usuario sin éxito
  Cuando introduzco datos inválidos en la pantalla de identificación:

Email	Contraseña
""	""
"wrong_user@dominio.com"	"pass"
"user@dominio.com"	"wrong_pass"

Entonces debería ver un mensaje de error en la misma pantalla

Figura 5.1: El usuario podrá identificarse en la aplicación

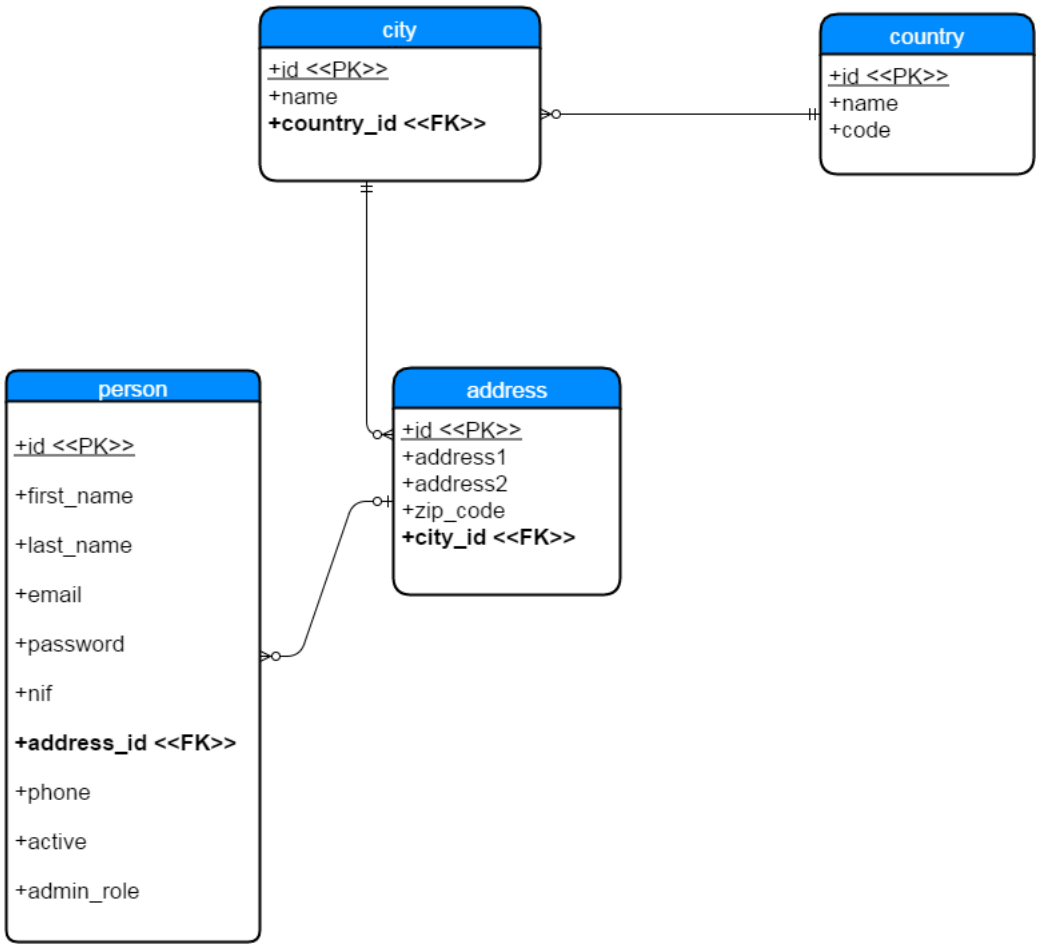


Figura 5.2: Estado del modelo entidad-relación para el login

sido solamente el modelo usuario (*User*) y los modelos relativos a la situación geográfica de dicho usuario, como son el modelo de dirección (Address), el de ciudad o región (*City*), y el de país (*Country*). Podemos ver la estructura del diagrama entidad-relación, diseñado para persistir dichos modelos de datos, en la Figura 5.2.

Puesto que los datos de localización del usuario poseen un carácter opcional, no ha sido preciso operar con dichos datos. La identificación del usuario ha sido realizada mediante la implementación de una clase de validación, que comprueba si el usuario existe y, en caso de que exista, comprueba si la contraseña asociada a dicho usuario es correcta. Esta primera historia ha sido estimada en 3 *puntos-historia*.

En la siguiente historia de usuario, valorada en 3 *puntos-historia*, nos encontramos con la necesidad de permitir que un usuario almacene una cuenta que le identifique unívocamente.

Como usuario no registrado
Para poder hacer uso de las funcionalidades de ***Busy***
Quiero poder registrarme en el sistema con mis datos personales

- **Antecedentes**: Empezar desde la página principal de ***Busy***
  Dado que estoy en la página principal de ***Busy***
  Cuando hago click en *"Sign up"*
- **Escenario**: Registro de usuario como cliente con éxito
  Cuando escribo mis datos de usuario válidos:

Nombre	Apellidos	Email	País	Ciudad	Código Postal	Teléfono	Contraseña	Confirmación contraseña
"Nombre"	"Apellidos"	"user@domain.com"	"España"	"Valladolid"	"47007"	"654321987"	"pass"	"pass"

  Entonces debería recibir un email de validación
  Y debería ver un mensaje de éxito
  Cuando hago click en "Validación"
  Entonces debería ver un mensaje de confirmación
  Y debería ser redirigido automáticamente a la página de identificación

- **Escenario**: Registro de usuario como cliente sin éxito
  Cuando escribo datos de usuario inválidos:

Nombre	Apellidos	Email	País	Ciudad	Código Postal	Teléfono	Contraseña	Confirmación contraseña
""	""	""	""	""	""	""	""	""
""	"Apellidos"	"user@domain.com"	"España"	"Valladolid"	"47007"	"654321987"	"pass"	"pass"
"Nombre"	""	"user@domain.com"	"España"	"Valladolid"	"47007"	"654321987"	"pass"	"pass"
"Nombre"	"Apellidos"	""	"España"	"Valladolid"	"47007"	"654321987"	"pass"	"pass"
"Nombre"	"Apellidos"	"user@domain.com"	""	""	""	""	"pass"	"pass"
"Nombre"	"Apellidos"	"user@domain.com"	"España"	"Valladolid"	"47007"	"654321987"	""	"pass"
"Nombre"	"Apellidos"	"user@domain.com"	"España"	"Valladolid"	"47007"	"654321987"	"pass"	""
"Nombre"	"Apellidos"	"user@domain.com"	"España"	"Valladolid"	"47007"	"654321987"	"pass"	"wrong_pass"
"Nombre"	"Apellidos"	"wrong_email.com"	"España"	"Valladolid"	"47007"	"654321987"	"pass"	"pass"
"Nombre"	"Apellidos"	"user@domain.com"	"España"	"Valladolid"	"47007"	"5"	"pass"	"pass"
"Nombre"	"Apellidos"	"user@domain.com"	"España"	"Valladolid"	"123456789AB"	"654321987"	"pass"	"pass"

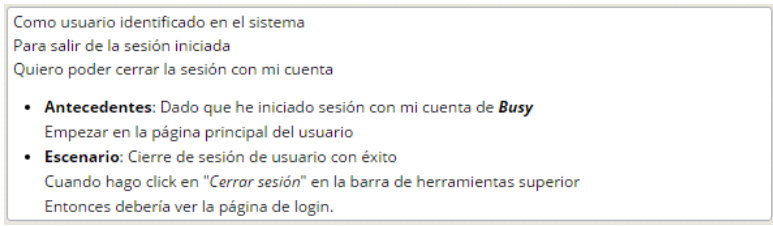
  Entonces debería ver un mensaje de error en la misma pantalla

Figura 5.3: El usuario podrá registrarse en la aplicación

Para ello el usuario podrá registrarse en el sistema completando un sencillo formulario a partir de sus datos personales. Esta funcionalidad, cuyo comportamiento se describe en la Figura 5.3, añade una capa de seguridad a la hora de registrarse.

Dicha capa de seguridad está basada en una tabla adicional de la base de datos, que almacenará una clave aleatoria que identificará unívocamente el registro de un usuario específico, y permitirá confirmar la cuenta a través de la dirección de correo para evitar la suplantación de identidad.

La tercera historia a la que se hace referencia, consiste básicamente en la finalización de sesión del usuario previamente identificado. Podemos ver por su descripción, en la Figura 5.4, que se trata de una historia de usuario sencilla y que por tanto es una de las que ha sido estimada en 1 *punto-historia*.


Como usuario identificado en el sistema
Para salir de la sesión iniciada
Quiero poder cerrar la sesión con mi cuenta

- **Antecedentes**: Dado que he iniciado sesión con mi cuenta de ***Busy***
  Empezar en la página principal del usuario
- **Escenario**: Cierre de sesión de usuario con éxito
  Cuando hago click en "*Cerrar sesión*" en la barra de herramientas superior
  Entonces debería ver la página de login.

Figura 5.4: El usuario podrá cerrar su sesión

5.2. Historias de usuario sobre *Gestión de empresa*

Respecto a la gestión de empresas, se han establecido cinco historias de usuario, que representan el conjunto de funcionalidades principales que permitirán, a las personas encargadas de una empresa, poder ofrecer los servicios de ésta a los clientes potenciales de la aplicación. Estas historias de usuario son:

- Registro de una nueva empresa (Figura 5.5).
- Ver estado del calendario de empresa (Figura 5.6).
- Gestionar tipos de servicio (Figura 5.7).
- Programación de nuevos servicios (Figura 5.8).
- Añadir trabajadores a la empresa (Figura 5.9).

La primera historia (Figura 5.5) se basa en la incorporación de una nueva empresa en el sistema. Esta funcionalidad es necesaria para que dicha empresa sea accesible, tanto para el gestor de la misma como para el usuario que va a buscar sus servicios. Su estimación ha sido de 2 *puntos-historia*.

La creación de una nueva empresa, inicializa su estado como bloqueado, evitando que los clientes puedan acceder a los servicios de la empresa hasta que ésta pase a estar activa y validada. Para dicho proceso de validación de empresas, se requerirá la intermediación de un usuario administrador de la aplicación, para aumentar la seguridad al disponer de una verificación manual, y evitar la suplantación de identidad o cualquier intento de engaño.

En segundo lugar tenemos una historia de usuario (Figura 5.6) relacionada con el tema de la calendarización de horarios, en la que al gestor de una sucursal de empresa se le debe permitir acceder a su calendario y comprobar los servicios ofrecidos por dicha sucursal, pudiendo visualizar a su vez el estado de disponibilidad de los servicios.

Como usuario registrado
Para poder hacer uso de las funcionalidades de una cuenta como empresa y ofrecer servicios
Quiero poder registrar en el sistema una empresa con sus datos fiscales

- **Antecedentes**: Empezar desde la página principal de mi cuenta de usuario de ***Busy***
  Dado que estoy registrado e identificado como usuario
  Cuando hago click en "Crear empresa"
- **Escenario**: Registro de nueva empresa con éxito
  Cuando escribo los datos de empresa válidos:

Nombre comercial	Nombre legal	Email	CIF	País	Ciudad	CP	Teléfono	Dirección	Categoría
"Boom"	"Boom S.A."	"jefe@boom.com"	"B12345678"	"España"	"Valladolid"	"47007"	666543210	"Calle Los Almendros, 2"	"Mobiliario"

  Entonces debería ver un mensaje informando de que su empresa está pendiente de aprobación
  Cuando se ha verificado manualmente por un administrador
  Entonces debería ver una notificación de confirmación
  Y debería ver en mi página principal una sección de empresa

- **Escenario**: Registro de nueva empresa con datos erróneos
  Cuando escribo datos de empresa inválidos:

Nombre comercial	Nombre legal	Email	CIF	País	Ciudad	CP	Teléfono	Dirección	Categoría
""	""	""	""	""	""	""	""	""	""
"Boom"	""	"jefe@boom.com"	"B12345678"	"España"	"Valladolid"	"47007"	"666543210"	"Calle Los Almendros, 2"	"Mobiliario"
"Boom"	"Boom S.A."	""	"B12345678"	"España"	"Valladolid"	"47007"	"666543210"	"Calle Los Almendros, 2"	"Mobiliario"
"Boom"	"Boom S.A."	"jefe@boom.com"	""	"España"	"Valladolid"	"47007"	"654321987"	"Calle Los Almendros, 2"	"Mobiliario"
"Boom"	"Boom S.A."	"jefe@boom.com"	"B12345678"	""	""	"47007"	"666543210"	"Calle Los Almendros, 2"	"Mobiliario"
"Boom"	"Boom S.A."	"wrong_email.com"	"B12345678"	"España"	"Valladolid"	"47007"	"666543210"	"Calle Los Almendros, 2"	"Mobiliario"
"Boom"	"Boom S.A."	"jefe@boom.com"	"123456789A"	"España"	"Valladolid"	"47007"	"666543210"	"Calle Los Almendros, 2"	"Mobiliario"
"Boom"	"Boom S.A."	"jefe@boom.com"	"B12345678"	"España"	"Valladolid"	"47007"	"5"	"Calle Los Almendros, 2"	"Mobiliario"

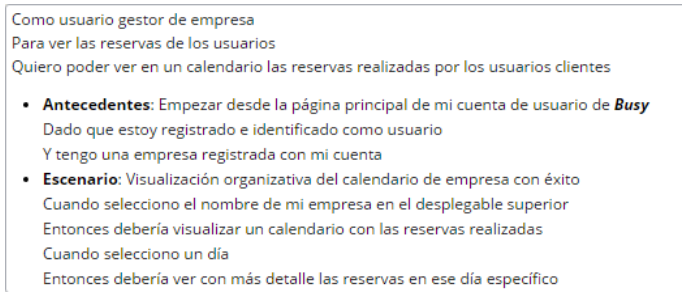
  Entonces debería ver un mensaje de error en la misma pantalla

- **Escenario**: Registro de nueva empresa invalidada por un administrador
  Cuando escribo datos de empresa correctamente de una empresa no existente:

Nombre comercial	Nombre legal	Email	CIF	País	Ciudad	CP	Teléfono	Dirección	Categoría
"X"	"X"	"user@gmail.com"	"B00000000"	"España"	"Valladolid"	"47007"	"00000000"	"Calle Los Almendros, 2"	"Mobiliario"

  Entonces debería ver un mensaje informando de que su empresa está pendiente de aprobación
  Cuando se ha rechazado manualmente por un administrador
  Entonces debería recibir una notificación en mi cuenta de ***Busy***
  Y NO debería ver en mi página principal la sección de empresa

Figura 5.5: El usuario podrá registrar una nueva empresa


Como usuario gestor de empresa
Para ver las reservas de los usuarios
Quiero poder ver en un calendario las reservas realizadas por los usuarios clientes

- **Antecedentes**: Empezar desde la página principal de mi cuenta de usuario de ***Busy***
  Dado que estoy registrado e identificado como usuario
  Y tengo una empresa registrada con mi cuenta
- **Escenario**: Visualización organizativa del calendario de empresa con éxito
  Cuando selecciono el nombre de mi empresa en el desplegable superior
  Entonces debería visualizar un calendario con las reservas realizadas
  Cuando selecciono un día
  Entonces debería ver con más detalle las reservas en ese día específico

Figura 5.6: El usuario podrá ver el calendario de su empresa

Cada servicio visualizado en el calendario corresponde a una franja horaria en la que se ha programado dicho servicio, disponiendo cada uno de un número variable de trabajadores asignados, conocidos en la aplicación como *roles*. La estimación ha sido de 3 *puntos-historia*.

La siguiente historia (Figura 5.7) añade una funcionalidad que permitirá al gestor de la empresa, tanto añadir nuevos tipos de servicio que ofrecerá dicha empresa a los usuarios, como modificar los que se encuentren ya creados. Dichos tipos de servicio se mostrarán tanto al gestor como al usuario en un listado, para que puedan interactuar fácilmente con ellos sin perder de vista el calendario de reservas. Esta historia ha sido valorada en 3 *puntos-historia*.

La creación de nuevos servicios en el horario de la empresa, es con diferencia la historia de usuario que ha entrañado más problemas en el proceso de desarrollo. Como vemos en la Figura 5.8, se han establecido 4 escenarios posibles. Sin embargo, la implementación se ha realizado de forma que ofrezca la mayor flexibilidad posible a la hora de añadir nuevos servicios.

En primer lugar nos permitirá crear tantos servicios como queramos dentro del horario de cada día del calendario, pudiendo añadir multiples franjas horarias del mismo servicio de forma dinámica.

Además, dentro de cada servicio se establecerá un tipo de repetición, pudiendo definir un servicio tanto diario como semanal sin necesidad de añadir información extra. A la hora de elegir trabajadores que vayan a realizar el servicio, se pueden seleccionar todos los que se deseen, asignándoles automáticamente a cada una de las franjas horarias definidas.

Esta historia, estimada bajo una valoración de 3 *puntos-historia*, ha supuesto un gran desvío entre la estimación y el tiempo final de desarrollo, debido principalmente a una amplia actualización en el modelo entidad-relación.

Por último en lo relativo a la gestión de la empresa, se ha llevado a cabo en el último sprint una historia de usuario, valorada en 2 *puntos-historia*, cuyo objetivo reside en permitir

Como usuario gestor de una empresa
Para poder ofrecer distintos servicios a los usuarios
Quiero poder añadir, modificar y eliminar tipos de servicio

- **Antecedentes**: Empezar desde la página principal de mi cuenta de usuario de ***Busy***
  Dado que tengo una empresa registrada
  Y que estoy registrado e identificado como usuario
- **Escenario**: Adición de un nuevo tipo de servicio
  Cuando selecciono el nombre de mi empresa en el desplegable superior
  Entonces debería ver vacía la lista de servicios
  Cuando selecciono añadir
  Entonces debería ver una ventana en la que introducir los datos:

"Nombre"	"Descripción"	"Número de reservas por trabajador"	"Duración de servicio"
"Tipo 1"	"Un tipo de servicio"	"2"	"60"

Cuando selecciono Aceptar
Entonces debería ver 'Tipo 1' en la lista de servicios con los datos introducidos

- **Escenario**: Modificación de un tipo de servicio
  Cuando añado un nuevo tipo de servicio 'Tipo 1'
  Y selecciono el nombre de mi empresa en el desplegable superior
  Entonces debería ver 'Tipo 1' en la lista de servicios
  Cuando selecciono modificar
  Entonces debería ver una ventana en la que introducir los datos:

"Nombre"	"Descripción"	"Número de reservas por trabajador"	"Duración de servicio"
"Tipo 1"	"Un tipo de servicio"	"2"	"60"

Cuando selecciono Aceptar
Entonces debería ver 'Tipo 1' en la lista de servicios con los datos actualizados

- **Escenario**: Eliminación de un tipo de servicio con éxito
  Cuando añado un nuevo tipo de servicio 'Tipo 1'
  Y selecciono el nombre de mi empresa en el desplegable superior
  Entonces debería ver 'Tipo 1' en la lista de servicios
  Cuando selecciono borrar
  Entonces no debería seguir viendo 'Tipo 1' en la lista de servicios
- **Escenario**: Eliminación de un tipo de servicio fallida
  Cuando añado un nuevo tipo de servicio 'Tipo 1', y al menos un servicio con al menos una reserva realizada en él
  Y selecciono el nombre de mi empresa en el desplegable superior
  Entonces debería ver 'Tipo 1' en la lista de servicios
  Cuando selecciono borrar
  Entonces debería ver un mensaje de error
  Y debería seguir viendo 'Tipo 1' en la lista de servicios

Figura 5.7: El usuario podrá gestionar los tipos de servicio ofrecidos por su empresa

Como usuario gestor de una empresa
Para establecer el horario de reserva de mi empresa
Quiero poder añadir servicios al horario de trabajo de mi empresa

- **Antecedentes**: Empezar desde la página principal de mi cuenta de usuario de ***Busy***
  Dado que tengo una empresa registrada
  Y que estoy registrado e identificado como usuario
  Y he seleccionado el nombre de mi empresa en el desplegable superior
- **Escenario**: Añadir un servicio sin tipos de servicio creados
  Cuando hago doble clic en un día del calendario ("15")
  Entonces debería ver un mensaje para que cree al menos un tipo de servicio
- **Escenario**: Añadir un servicio con datos inválidos
  Cuando añado al menos un tipo de servicio
  Y añado los roles/trabajadores que quiero tener disponibles para los servicios
  Y hago doble clic en un día del calendario ("15")
  Entonces debería ver un diálogo en el que introducir datos
  Cuando introduzco los siguientes datos:

Hora Inicio	Tipo de Servicio	Trabajadores
""	"Tipo 1"	"Juan"
"9:00"	""	"Juan"
"9:00"	"Tipo 1"	""

  Y hago clic en el botón 'Guardar'
  Entonces debería ver un mensaje de error

- **Escenario**: Añadir un servicio sin repetición con éxito
  Cuando añado al menos un tipo de servicio
  Y añado los roles/trabajadores que quiero tener disponibles para los servicios
  Y hago doble clic en un día del calendario ("15")
  Entonces debería ver un diálogo en el que introducir datos
  Cuando introduzco los siguientes datos:

Hora Inicio	Tipo de Servicio	Trabajadores
"9:00"	"Tipo 1"	"Juan"
"23:30"	"Tipo 1"	"Juan", "Carlos"

  Y hago clic en el botón 'Guardar'
  Entonces debería ver el servicio o servicios creados correctamente en el día seleccionado

- **Escenario**: Añadir un servicio con repetición con éxito
  Cuando añado al menos un tipo de servicio
  Y añado los roles/trabajadores que quiero tener disponibles para los servicios
  Y hago doble clic en un día del calendario ("15")
  Entonces debería ver un diálogo en el que introducir datos
  Cuando introduzco los siguientes datos:

Hora Inicio	Tipo de Servicio	Trabajadores	Tipo repetición
"9:00"	"Tipo 1"	"Juan"	"Diaria"
"9:00"	"Tipo 1"	"Juan"	"Semanal"
"9:00"	"Tipo 1"	"Juan", "Carlos"	"Diaria"
"9:00"	"Tipo 1"	"Juan", "Carlos"	"Semanal"

  Y hago clic en el botón 'Guardar'
  Entonces debería ver el servicio o servicios creados correctamente en el día seleccionado
  Y debería ver el servicio o servicios repetidos según el tipo de repetición elegido

Figura 5.8: Un usuario podrá crear servicios en el horario de su empresa

Como usuario gestor de una empresa
Para poder llevar el control de la carga de trabajo de los empleados
Quiero poder añadir la ficha de un trabajador de la empresa

- **Antecedentes**: Empezar desde la página de gestor de la empresa
  Dado que tengo una empresa registrada
  Y que estoy registrado e identificado como usuario
- **Escenario**: Inclusión de un nuevo trabajador con éxito
  Cuando selecciono el nombre de mi empresa en el desplegable superior
  Y despliego el menú de trabajadores
  Entonces debería ver la lista de trabajadores sólo con mi nombre
  Cuando selecciono "Añadir trabajador"
  Entonces debería mostrarse un diálogo en el que introducir los datos del trabajador
  Cuando meto sus datos correctamente:

Nombre	Apellidos	Email	Teléfono
“Pepito”	"González"	“pepito@domain.x”	"666789012"

  Y hago clic en 'Guardar'
  Entonces debería ver un mensaje de confirmación
  Y debería ver automáticamente su nombre en la lista de empleados de la empresa

- **Escenario**: Registro de usuario como empresa sin éxito
  Cuando selecciono el nombre de mi empresa en el desplegable superior
  Y despliego el menú de trabajadores
  Entonces debería ver la lista de trabajadores sólo con mi nombre
  Cuando selecciono "Añadir trabajador"
  Entonces debería mostrarse un diálogo en el que introducir los datos del trabajador
  Cuando escribo datos inválidos:

Nombre	Apellidos	Email	Teléfono
“”	"González"	“pepito@boom.com”	"666789012"
“Pepito”	""	“pepito@domain.x”	"666789012"
“Pepito”	"González"	“”	"666789012"
“Pepito”	"González"	“pepito@boom.com”	""
“Pepito”	"González"	“wrong_email.com”	"666789012"
“Pepito”	"González"	“pepito@boom.com”	"5"

  Y hago clic en 'Guardar'
  Entonces debería ver un mensaje de error en la misma pantalla

Figura 5.9: El usuario podrá registrar nuevos trabajadores en su empresa

la inserción de nuevos trabajadores a la empresa. Como vemos en la Figura 5.9, sólo se ha considerado la opción de nuevo usuario, al contrario que en el caso de los tipos de servicio en los que se implementaron además las funcionalidades de modificación y borrado de tipos.

Sin embargo, en esta historia se añade un nuevo comportamiento, basado en la actualización asíncrona del formulario. Esto consiste en que al rellenar el campo de correo electrónico, se comprueba de forma dinámica contra el servidor si existe o no algún usuario vinculado ya a dicho correo eléctronico. En caso afirmativo, los campos del formulario se rellenar automáticamente, y se inhabilitaran los demás campos para evitar que el gestor de la empresa pueda modificar la ficha del usuario existente. En caso negativo, el gestor podrá continuar añadiendo los datos para crear un nuevo usuario con la información básica que introduzca.

5.3. Historias de usuario relacionadas con *Reserva de servicios*

La siguiente historia *epic*, engloba las historias de usuario destinadas al proceso de solicitud y reserva de un servicio por parte de un usuario cliente.

Esta es la sección de historias de usuario que aportará un mayor valor a la aplicación. Si bien todas las anteriores son necesarias para el buen funcionamiento de ***buSy***, las que se comentarán a continuación serán las que aporten esa interactividad entre los usuarios cliente y los gestores de empresa. Gracias a ello, la aplicación facilitará los trámites para el proceso de reserva de una gran variedad de servicios, debido al enfoque generalista que se ha adoptado para este proyecto, y por el cuál se reducen considerablemente las limitaciones en cuanto al dominio de usuarios finales.

- Búsqueda de empresas por nombre (Figura 5.10).

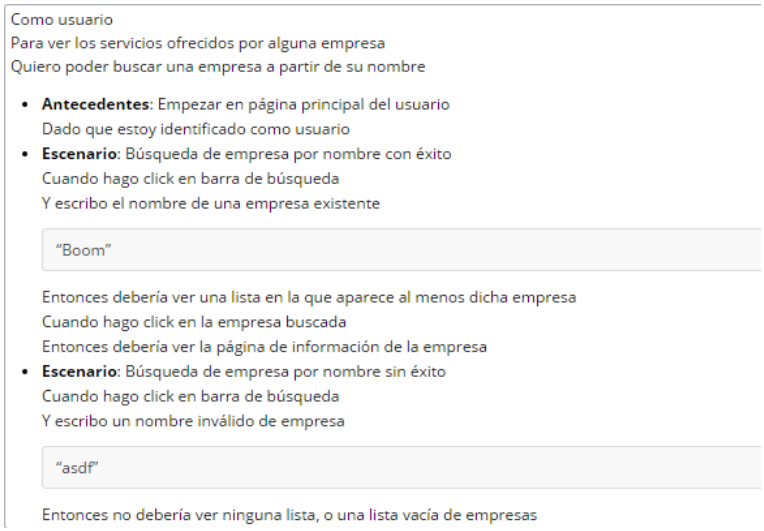
- Reserva de servicios en una empresa (Figura 5.11).

Para poder realizar una reserva, lo primero que se debe permitir hacer al usuario será buscar la empresa con la que quiere llevar a cabo la reserva de uno de sus servicios. Para ello, esta historia de usuario (Figura 5.10) se basa en la posibilidad de buscar una empresa a través de su nombre comercial o su razón social. Así, el usuario podrá acceder fácilmente a la página de la empresa solicitada.

La búsqueda se basa en un campo de auto-completado, en el que se filtrará una lista con el nombre de las empresas cuyo nombre contenga la secuencia de letras que va siendo introducida. Una vez encontrado el nombre de la empresa deseada, el usuario podrá acceder a la página de dicha empresa mediante un simple clic en el nombre. la estimación ha sido de 2 *puntos-historia*.

A la hora de reservar servicios, podemos ver en la Figura 5.11 la siguiente historia de usuario, cuyos escenario se componen de una serie de pasos sencillos y metódicos, simplificando así el proceso de reserva.

Una vez que el usuario cliente se encuentra en la página de la empresa, podrá acceder al calendario de ésta a través de la sección de reservas, y seleccionando la franja horaria que más le convenga podrá reservar un servicio con el trabajador que elija en un diálogo posterior. Esta historia de usuario ha sido valorada también en 2 *puntos-historia*.


Como usuario
Para ver los servicios ofrecidos por alguna empresa
Quiero poder buscar una empresa a partir de su nombre

- **Antecedentes**: Empezar en página principal del usuario
  Dado que estoy identificado como usuario
- **Escenario**: Búsqueda de empresa por nombre con éxito
  Cuando hago click en barra de búsqueda
  Y escribo el nombre de una empresa existente

  "Boom"

  Entonces debería ver una lista en la que aparece al menos dicha empresa
  Cuando hago click en la empresa buscada
  Entonces debería ver la página de información de la empresa
- **Escenario**: Búsqueda de empresa por nombre sin éxito
  Cuando hago click en barra de búsqueda
  Y escribo un nombre inválido de empresa

  "asdf"

  Entonces no debería ver ninguna lista, o una lista vacía de empresas

Figura 5.10: El usuario podrá buscar empresas por nombre

5.4. Historias de usuario relacionadas con *Administración*

La siguiente sección engloba las historias relacionadas con el usuario administrador, quien debe tener un control total sobre la aplicación, y por tanto debe poder realizar cualquier acción que bloquee al usuario, o que asegure el buen funcionamiento tanto de la aplicación como del proceso de negocio entre empresa y cliente.

- Verificación de empresas (Figura 5.12).
- Bloqueo de usuarios.

Una de las principales tareas del administrador consistirá en comprobar la validez de las nuevas empresas creadas, por tanto gracias a la implementación de esta primera historia de usuario (Figura 5.12) sobre administración, las empresas podrán ser marcadas como activas o inactivas, evitando posibles suplantaciones de identidad fiscal o cualquier otro tipo de incidencia con la empresa que se pretende registrar. La estimación ha sido de tan sólo 1 *punto-historia*.

La última historia de usuario se trata del bloqueo de usuarios por parte del administrador. Con el transcurso del tiempo de uso de la aplicación pueden surgir problemas con alguno de los usuarios registrados, ya sea por suplantación de datos como por falta repetida de asistencia a las reservas realizadas.

Como usuario cliente
Para reservar hora en alguna empresa
Quiero poder reservar una hora de servicio en una empresa determinada

- **Antecedentes**: Empezar en página principal del usuario
  Dado que estoy identificado como usuario
  Y estoy en la página de información de una empresa
- **Escenario**: Reserva de una hora con éxito
  Cuando hago click en "Reservas"
  Entonces debería ver una lista con las sucursales de la empresa
  Cuando selecciono una sucursal
  Entonces debería mostrarse un calendario con las horas de servicio libres y ocupadas
  Cuando selecciono una hora libre en el calendario
  Entonces debería ver una pantalla que muestre los trabajadores disponibles
  Cuando selecciono un trabajador
  Y selecciono 'Reservar'
  Entonces debería recibir una notificación de la reserva
  Y no debería ver la hora seleccionada como hora disponible
- **Escenario**: Reserva de una hora ya reservada
  Cuando hago click en "Reservas"
  Entonces debería ver una lista con las sucursales de la empresa
  Cuando selecciono una sucursal
  Entonces debería mostrarse un calendario con las horas de servicio libres y ocupadas
  Cuando selecciono una hora libre en el calendario
  Entonces debería ver una pantalla que muestre los trabajadores disponibles
  Cuando la misma hora está siendo reservada por otra persona
  Y selecciono un trabajador
  Y selecciono 'Reservar'
  Entonces debería ver un mensaje de error
- **Escenario**: Reserva de una hora cuando la empresa está inactiva
  Cuando hago click en "Reservas"
  Debería mostrarse un mensaje indicando que la empresa no se encuentra activa

Figura 5.11: El usuario cliente podrá reservar una hora en una empresa determinada

Como administrador
Para verificar la identidad de una empresa
Quiero poder confirmar la identidad de una empresa

- **Antecedentes**: Empezar en la página principal de administrador de ***Busy***
  Dado que estoy identificado como administrador
- **Escenario**: Confirmación de empresa con éxito
  Cuando voy a "Validar nuevas empresas"
  Y selecciono la empresa pendiente de confirmación de nombre 'Boom S.A.'
  Entonces debería ver la empresa como bloqueada o no activa
  Cuando hago click en "Confirmar empresa"
  Entonces el responsable de la empresa debería ver una notificación de confirmación de la operación
  Y se mandará un email al responsable de dicha empresa
  Cuando vuelvo a hacer click en "Validar nuevas empresas"
  La empresa 'Boom S.A.' aparece como activa

Figura 5.12: El administrador podrá verificar empresas

Como administrador
Para evitar el uso inadecuado de la aplicación
Quiero prohibir el uso de la aplicación a un usuario

- **Antecedentes**: Empezar en la página principal de administrador de ***Busy***
  Dado que estoy identificado como administrador
- **Escenario**: Bloquear un usuario con éxito
  Cuando selecciono la sección de usuarios
  Entonces debería mostrarse una lista detallada con todos los usuarios del sistema
  Y debería ver al usuario "Usuario1" como activo
  Cuando hago click en Bloquear en la fila del usuario "Usuario1"
  Entonces debería ver un mensaje de confirmación del bloqueo
  Y debería cambiar el estado del usuario a bloqueado

Figura 5.13: El administrador podrá bloquear usuarios

En estos casos se podrá bloquear a dichos usuarios problemáticos, para mantener un control sobre el buen funcionamiento ***buSy***. En la Figura 5.13 vemos la descripción de esta historia de usuario, que está basada en un sencillo escenario, por lo que su estimación en *puntos-historia* es de 1.

CAPÍTULO 6

Diseño de la Interfaz gráfica

En este capítulo de analizará la capa de vista de la aplicación, a través de los prototipos de diseño realizados para guiar el desarrollo de la interfaz gráfica.

6.1. Interfaz de la *Gestión de cuentas*

Para el conjunto de historias de usuario englobadas en la gestión de cuentas, las páginas a desarrollar han sido tanto las páginas de *login* o identificación y de registro del usuario, como las páginas de perfil personal y jurídico de usuario y de empresa respectivamente.

Para la realización de bocetos se ha empleado la herramienta Balsamiq Mockups, comentada en el capítulo 3. Esto ha permitido una visión del resultado final de la parte gráfica de la aplicación previa a su desarrollo.

En la Figura 6.1 podemos ver la interfaz creada para la identificación de un usuario registrado.

Es un diseño minimalista y sencillo para evitar distraer al usuario de la función principal de esta página, que no es más que la de entrar al sistema o acceder al registro de un nuevo usuario en caso de no tener cuenta alguna.

La Figura 6.2 pertenece a la historia relativa al registro de un nuevo usuario. La vista de esta funcionalidad se basa en un formulario para introducir los datos del usuario, el cuál incluye un evento que actualiza un icono de obligatoriedad en los campos que se vayan dejando vacíos dinámicamente.

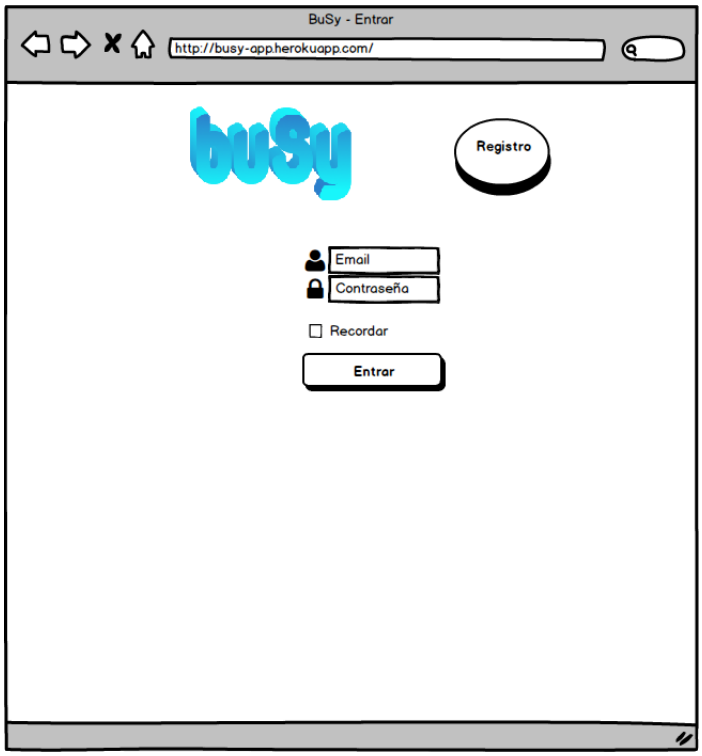


Figura 6.1: Prototipo de la vista de identificación

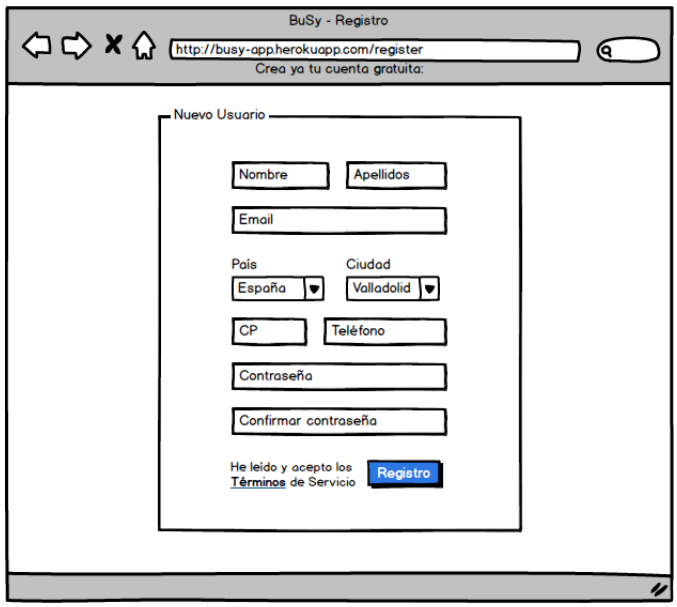


Figura 6.2: Prototipo de la vista de registro

Además cuenta con dos selectores dinámicos, uno para elegir el país y otro para elegir la ciudad. Los valores para el primero son cargados previamente, sin embargo los del segundo selector se cargan dinámicamente cuando el usuario selecciona un determinado país.

La siguiente pantalla que veremos será la pantalla principal (Figura 6.3), que contará en primer lugar con una barra de navegación superior. Esta barra constará de dos partes diferenciadas; la primera de ellas situada a la izquierda, la cuál incluirá tanto el icono identificativo de la plataforma ***buSy***, como la barra de búsqueda de empresas con auto-completado. En la segunda parte de la barra de navegación, situada a la derecha de la pantalla, podemos ver tanto el menú de acciones del usuario como el contador desplegable de notificaciones.

6.2. Interfaz de la *Gestión de empresa*

Para el registro de nuevas empresas, se ha implementado un botón que será accesible por cualquier usuario, siempre y cuando no haya registrado ya alguna empresa.

Esto es así para evitar la creación indiscriminada de empresas ya que, debido al enfoque del proyecto hacia la pequeña y mediana empresa, se ha establecido que un usuario típicamente no tendrá más de una empresa a su cargo a la vez.

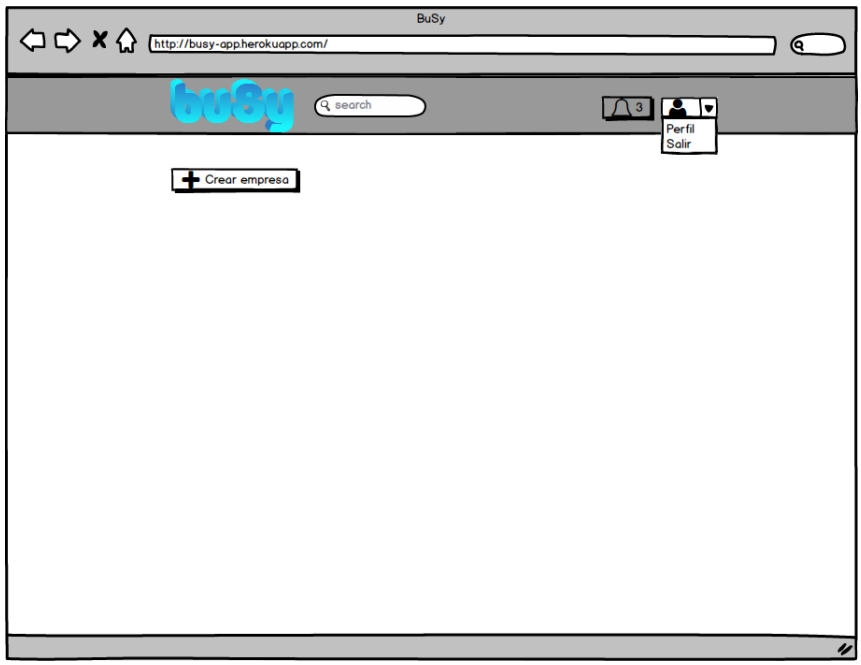


Figura 6.3: Prototipo de la vista principal

Una vez dentro del formulario, el cuál podemos visualizar en la Figura 6.4, se podrán rellenar todos los datos de la empresa, indicándose dinámicamente los campos obligatorios que se van dejando vacíos, con un reborde sobre el campo de color rojizo, y un símbolo de un asterisco del mismo color.

Una vez tengamos una empresa creada, podemos acceder a la sección de gestor de empresa, en la que visualizaremos un calendario de servicios similar al mostrado en la Figura 6.5. En ella, además del calendario se nos mostrarán dos menús desplegables con la lista de empleados de la empresa y tipos de servicio, respectivamente.

Desde el menú

6.3. Interfaz para la zona de *Reserva de servicios*

Cada empresa registrada en el sistema dispondrá de una vista propia, accesible desde cualquier usuario mediante el campo de búsqueda de empresas, siempre y cuando se encuentre en estado activo y no esté bloqueada por un administrador.

La página de cada empresa consta de varias secciones. La primera de ellas, la cual

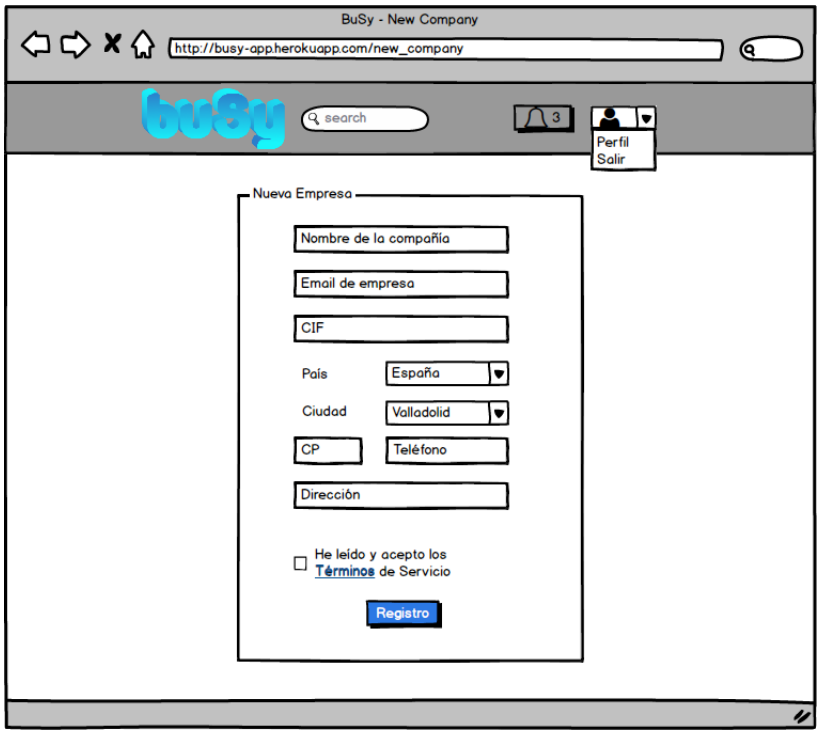


Figura 6.4: Prototipo de la vista de nueva empresa

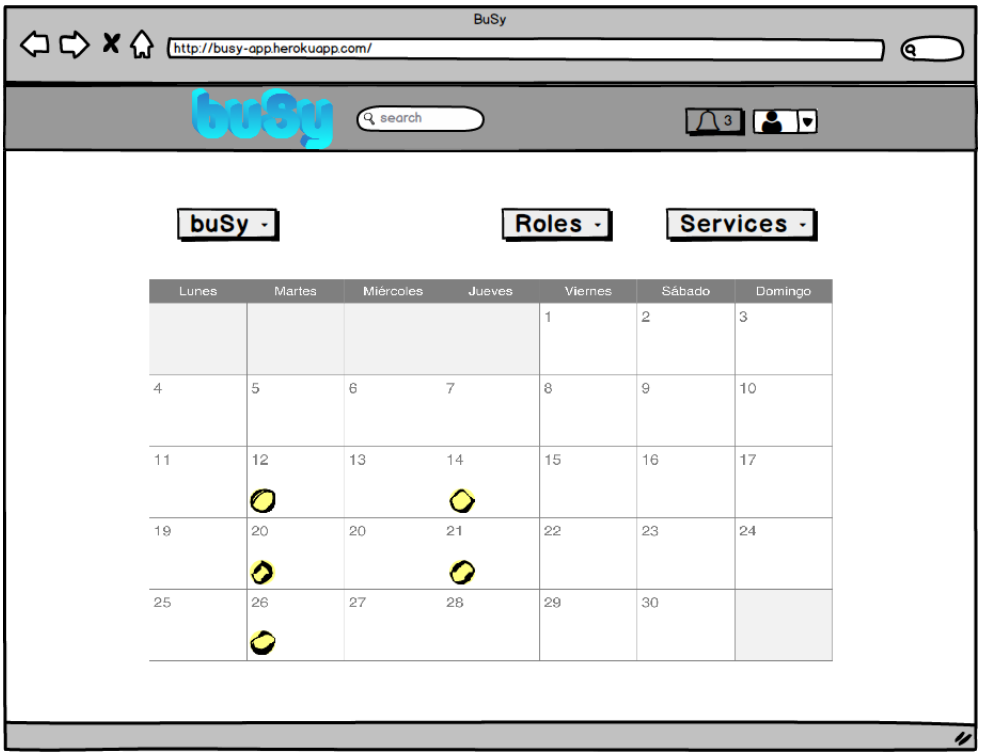


Figura 6.5: Prototipo de la vista de gestión de empresa

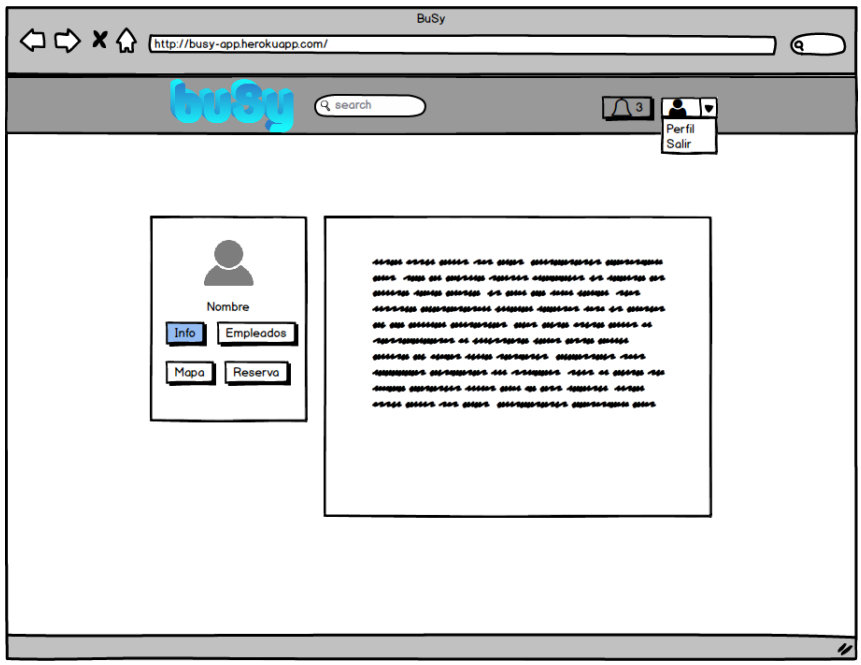


Figura 6.6: Prototipo de la vista de información de la empresa

podremos observar en la Figura 6.6, recopila toda la información que se dispone de dicha empresa. Como vemos en la imagen, mediante el menú podremos cambiar rápidamente entre las distintas secciones dentro de la misma empresa.

La sección de reservas de una empresa nos mostrará en primer lugar una lista de oficinas o sucursales de dicha empresa, como podemos observar en la Figura 6.7. Dicha vista contendrá los respectivos datos específicos de cada oficina, como por ejemplo la situación geográfica para su localización.

Una vez que el usuario haya seleccionado una de las oficinas, se le mostrará un calendario de servicios, en el cuál se mostrarán tanto las franjas horarias disponibles como las no disponibles, diferenciadas por un icono de distinto color, cuyo significado vendrá explicado en unas etiquetas a modo de leyenda. El prototipado de la vista del calendario de reservas lo podemos observar en la Figura 6.8.

La sección de trabajadores de una empresa, cuyo boceto se puede ver en la Figura 6.9, constará de una lista de todos los trabajadores registrados en cada una de las sucursales de la empresa, la cuál será seleccionada previamente por el usuario en una vista similar a la comentada para la vista anterior.

Por último, la página de empresa contendrá también una sección destinada a la ubicación

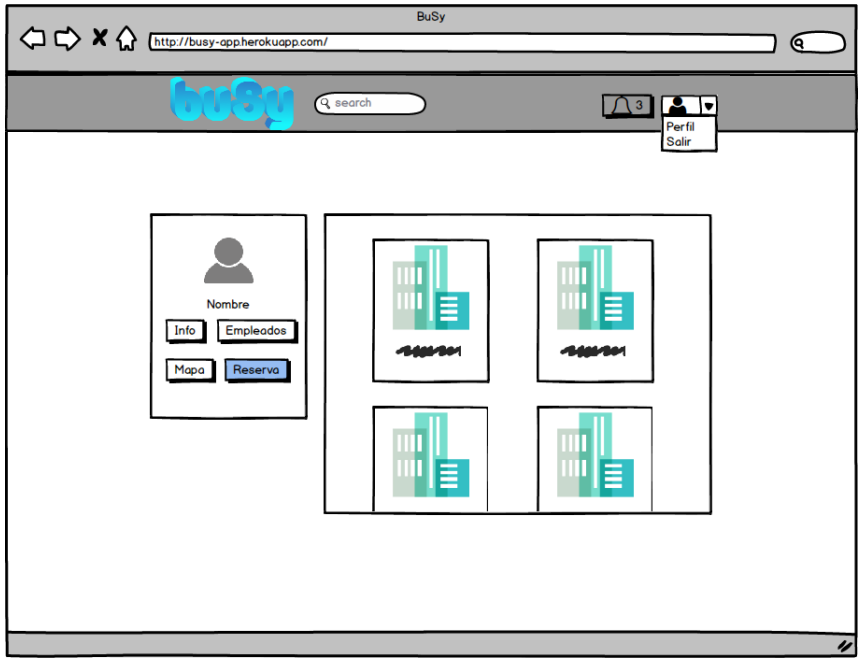


Figura 6.7: Prototipo de la vista de sucursales de la empresa

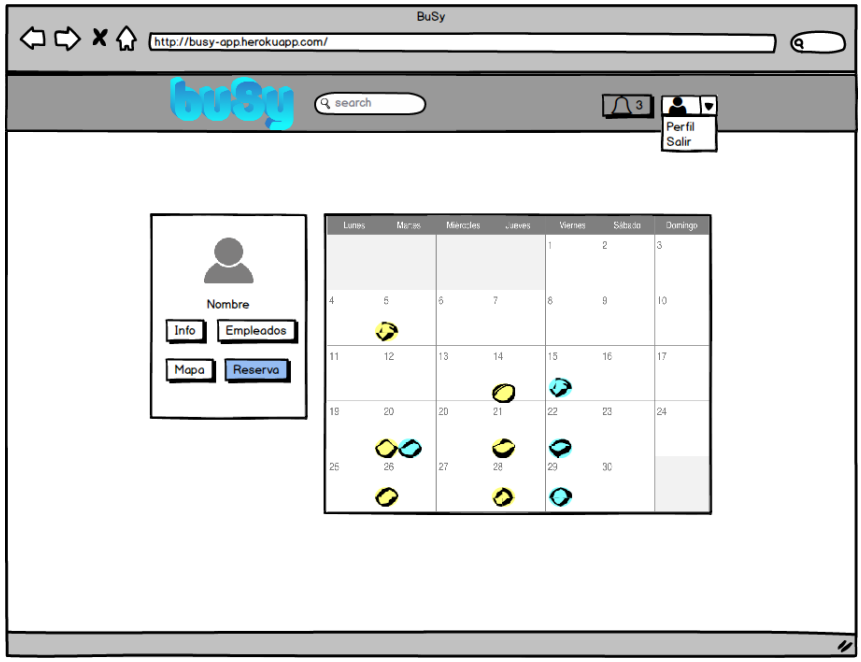


Figura 6.8: Prototipo de la sección de calendario de la empresa

Figura 6.9: Prototipo de la sección de trabajadores de la empresa

geográfica de sus oficinas dispuesta sobre un mapa. Esto permitirá que los usuarios puedan identificar la sucursal más cercana, de una forma dinámica y sencilla.

6.4. Interfaz para la zona de *Administración*

La página de administración, accesible únicamente por usuarios con privilegio de administrador, constará de una serie de bloques para cada una de las secciones.

En primer lugar, en la Figura 6.11 se muestra el bloque de administración de empresas, con su respectiva sección ya desplegada. El bloque superior mostrará el número total de empresas actualmente registradas en la aplicación.

Una vez se accede a la sección de empresas, podemos ver que aparecen todas las empresas listadas. Cada una de ellas dispondrá de los datos administrativos, además de un interruptor de estado. Dicho interruptor permitirá al usuario administrador activar una nueva empresa, bloquear ciertas empresas por uso indebido de la aplicación, o reactivarlas tras un bloqueo manual.

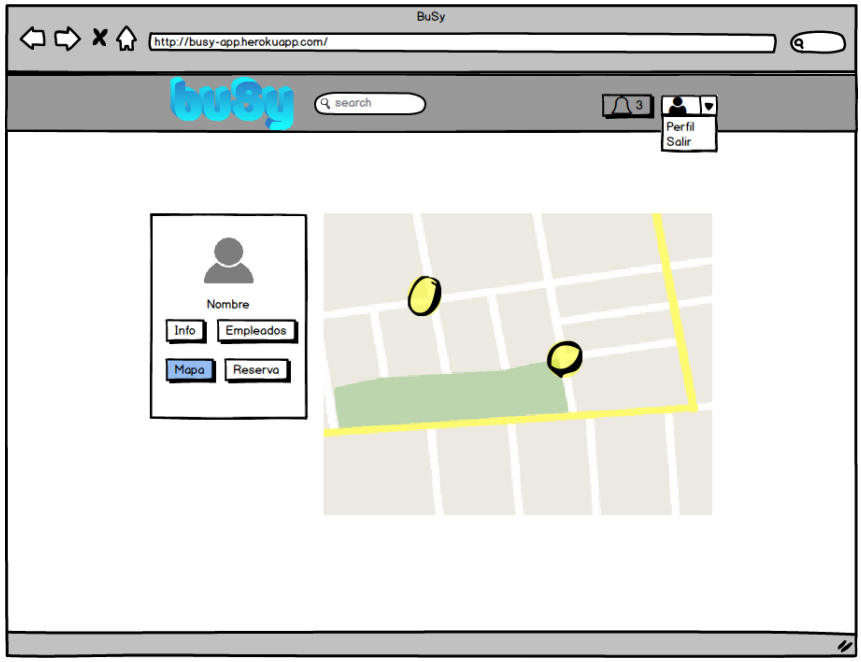


Figura 6.10: Prototipo de la sección del mapa de la empresa

Figura 6.11: Prototipo de la página de administración

CAPÍTULO 7

Diseño Final

En este proyecto se ha aplicado el patrón de desarrollo por capas, por lo que a continuación comentaré los diagramas resultantes de cada una de ellas, lo cuál nos proporcionará una visión detallada de cada módulo de la aplicación. Seguidamente se mostrará una visión más global del sistema a través de un diagrama de la arquitectura del proyecto, y tres diagramas de secuencia para ilustrar la comunicación a través de las distintas capas.

7.1. Capa de presentación

Como vimos en la Figura 4.3 del capítulo 4, la capa de presentación se ha organizado siguiendo el patrón **Modelo-Vista-Controlador**, garantizando la mantenibilidad del código y la separación de responsabilidades.

Cada uno de estos componentes presenta un modelo diferente de organización, como veremos a continuación.

7.1.1. Modelo

A lo largo de todas las historias de usuario desarrolladas, se ha ido completando el modelo de dominio que podemos observar en la Figura 7.1.

En este diagrama se muestra el modelo conceptual de todas las clases del dominio, que soportarán toda la información necesaria dentro de la aplicación para poder implementar la lógica de negocio deseada. Como se puede observar, las entidades no presentan ninguna

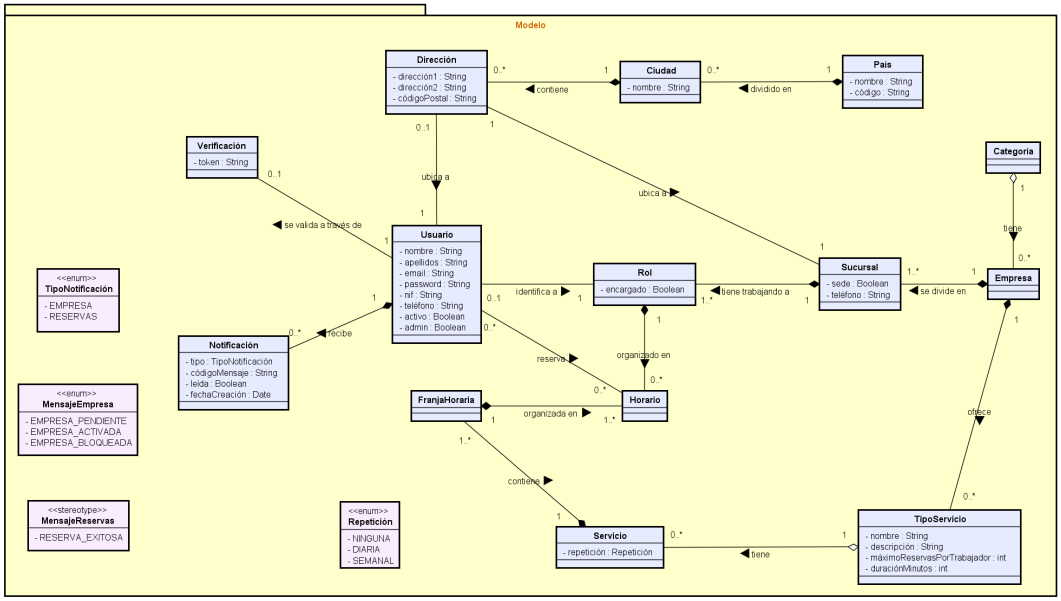


Figura 7.1: Modelo de dominio

operación. Esto es así debido a la separación de la capa del modelo, delegando todas las operaciones sobre cada entidad en los controladores correspondientes. Estos a su vez delegan las operaciones de actualización y recuperación de los datos del dominio, las cuales pasan a ser responsabilidad de una capa intermedia entre la capa de persistencia y la de presentación. Esta capa se denomina capa de servicio, y se encargará de conectar cada operación con la clase de implementación de la persistencia correspondiente al modelo sobre el que se opera.

7.1.2. Vista

La organización de la capa de las vistas, se ha llevado a cabo aprovechando la modularidad que ofrece *Spring Framework* y el motor de plantillas *Thymeleaf*. Gracias a ello, se ha divido la capa en plantillas completas HTML por un lado, y en fragmentos inyectables en dichas plantillas.

En la Figura 7.2 podemos observar la dependencia de cada una de las vistas, con los respectivos fragmentos inyectados en ellas. Esta modularización nos permite dos cosas muy importantes para el desarrollo de la aplicación:

En primer lugar, aporta un gran valor de reutilización, con lo que evitamos a su vez la duplicidad de código.

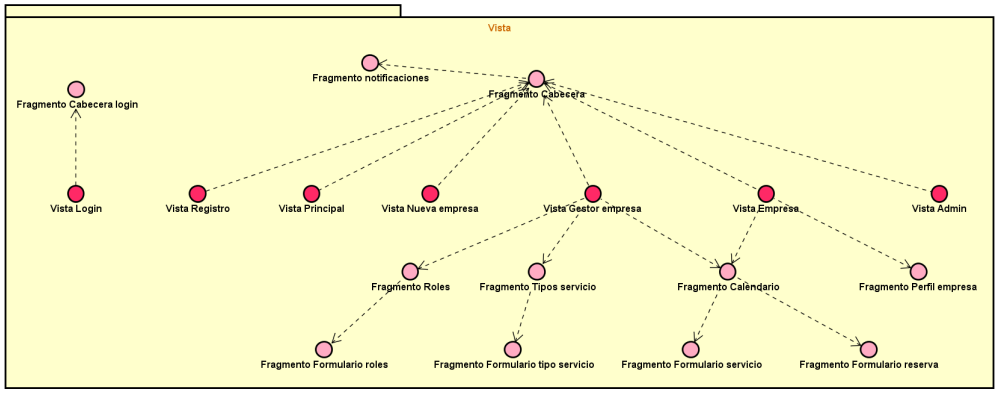


Figura 7.2: Modelo de vistas

En segundo lugar, gracias a la separación en fragmentos, *Spring Framwork* nos permite renderizar dicho fragmento en concreto, permitiendo así ejecutar peticiones AJAX al controlador, de forma que actualice o inyecte una parte de la vista sin alterar el resto de componentes de la misma.

7.1.3. Controlador

Debido al enfoque del proyecto hacia una aplicación web, sabemos que en la capa de los controladores reside gran parte de la funcionalidad. Esta capa será la encargado de conectar las peticiones de parte del cliente con las operaciones de la aplicación del lado del servidor. Para poder ver en profundidad la organización de dicha capa, se muestra en la Figura 7.3 un diagrama con los distintos tipos de controladores implementados y las operaciones que llevarán a cabo.

Como se puede observar, se ha intentado encapsular las operaciones en diferentes controladores según la parte de la funcionalidad a la que están destinadas, siguiendo así el **Principio de responsabilidad única**, definido dentro de los patrones **SOLID**

7.2. Capa de lógica de negocio

La política de organización de paquetes del código de la aplicación ha estado guiada por funcionalidades, enfocándose de esta forma hacia una modularización por componentes.

Gracias a ello se han podido definir con claridad las clases de implementación de la lógica de negocio, separando correctamente cada ámbito de responsabilidad. En la Figura

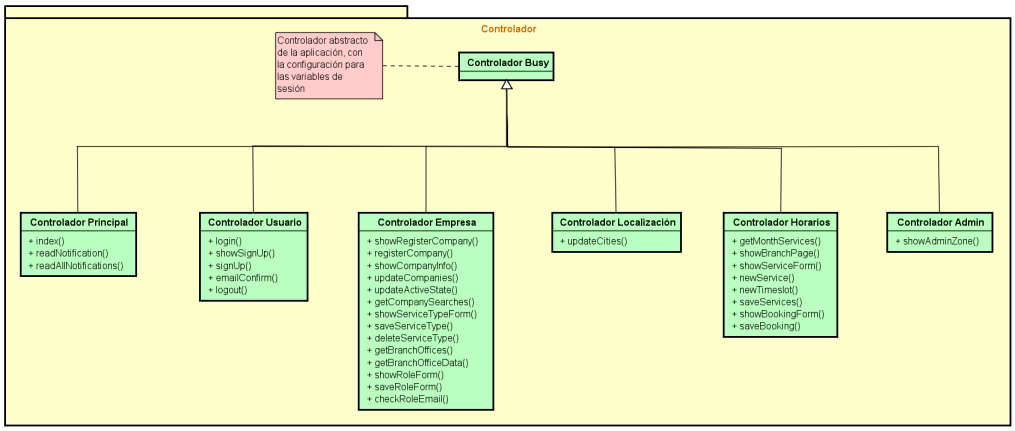


Figura 7.3: Modelo de controladores

7.4 podemos ver una visión detallada de cada uno de los servicios y de los métodos públicos que ponen a disposición de la capa de presentación mediante las interfaces de desarrollo correspondientes.

Como vemos en el diagrama, se dispone de seis módulos principales dentro de la aplicación.

En primer lugar tenemos el módulo de usuario, el cuál abarca tanto las operaciones sobre el usuario como la gestión de verificaciones de nuevos usuarios. A continuación vemos el módulo de notificaciones, el cuál incluye un subpaquete para la gestión de mensajes definidos a partir de tipos enumerados. El módulo de empresa comprende además las operaciones sobre categorías, de las que la empresa depende directamente, y las relativas a sucursales de empresa. Otro de los módulos está dedicado a la gestión de los roles, ya que va a suponer un punto de comunicación entre las sucursales de empresa y los servicios programados. El módulo de servicios, ligeramente mayor al resto, se compone tanto de los servicios y su calendarización en franjas horarias como de las operaciones relativas a la gestión de reservas y planificación de los trabajadores dentro de cada servicio. Por último se dispone de un módulo para la localización, en el que encontramos la implementación de operaciones sobre países, ciudades y direcciones postales.

7.3. Capa de persistencia de datos

Dentro de la capa de persistencia, se dispone de una serie de clases organizadas en los módulos anteriormente comentados. Dentro de cada módulo se define una clase de implementación por cada entidad que precise de persistencia contra la base de datos.

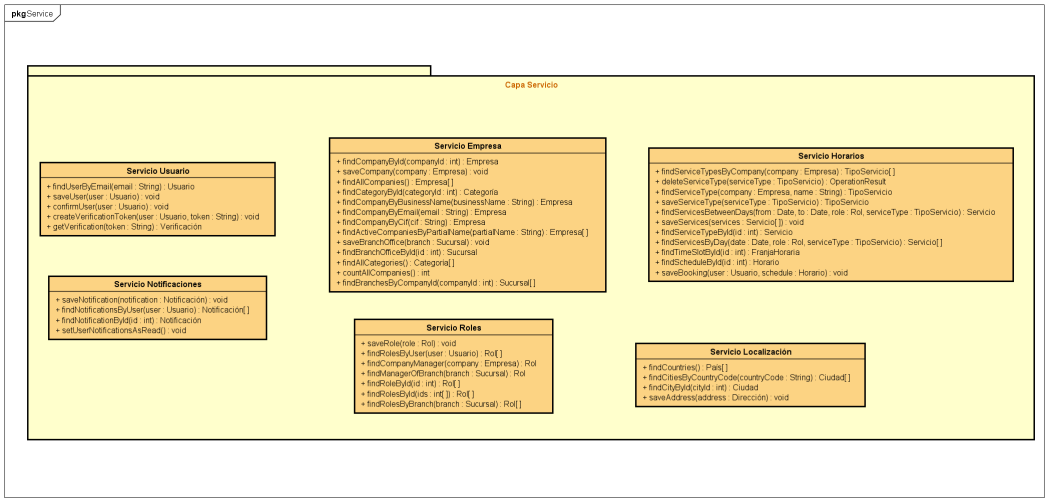


Figura 7.4: Diagrama de la capa de lógica de negocio

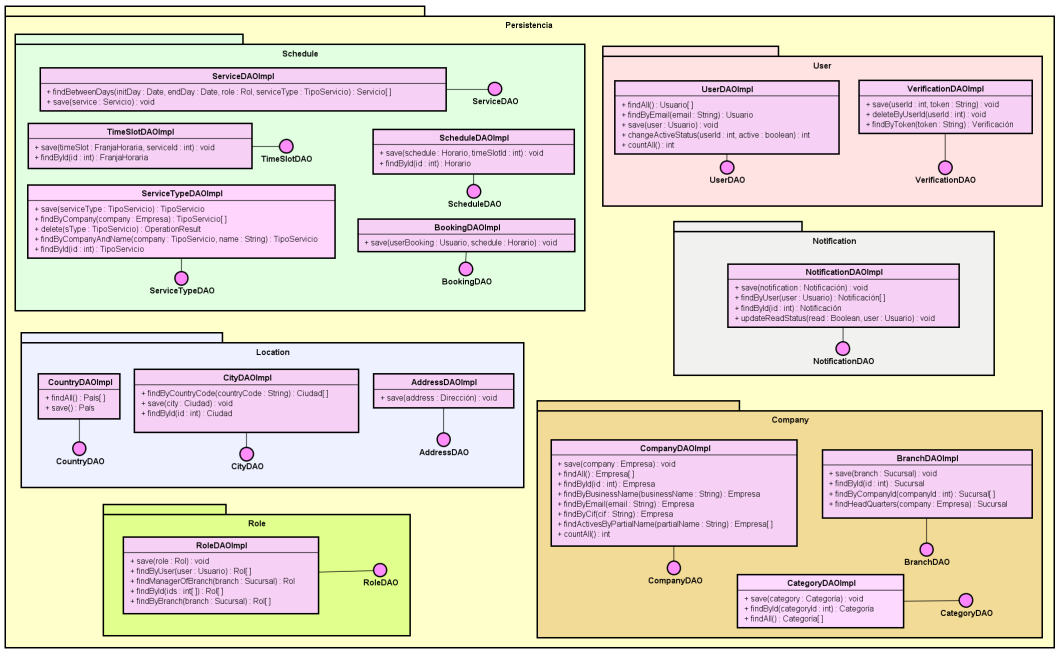


Figura 7.5: Diagrama de la capa de persistencia

La Figura 7.5 nos muestra la organización detallada de las clases de persistencia. En el diagrama se definen además cada una de las interfaces a través de las que se conecta con la capa de lógica de negocio, poniendo a su disposición la interacción con el sistema de base de datos de forma abstracta a su implementación.

7.4. Visión global

Para mostrar la arquitectura global de la aplicación, se presenta el diagrama de la Figura 7.6, en la que se muestra la comunicación por capas entre los distintos módulos.

Gracias a **Spring Framework**, los componentes se inyectan automáticamente mediante la anotación *@Autowired* permitiendo así, además de mantener una única instancia por cada uno en la aplicación, reducir el acoplamiento entre componentes y facilitar la implementación de un código mantenible y escalable.

7.5. Modelos dinámicos

A continuación se muestran los modelos de secuencia de operaciones para tres de las historias de usuario llevadas a cabo en este proyecto.

7.5.1. Identificación de usuario

En primer lugar, se muestra en la Figura 7.7 la historia de usuario sobre identificación de usuarios, en la que podemos ver un flujo básico en el que el usuario introduce sus datos de acceso y la aplicación los verifica a través de la comunicación entre capas.

El caso mostrado en este diagrama se trata del escenario de éxito, en el que la petición llega a la capa de persistencia ejecutando una operación de lectura. De esta forma los datos de la cuenta de usuario serán recuperados y enviados de vuelta a la capa de presentación, donde el controlador procederá a redirigir al usuario a la vista correspondiente según el tipo de usuario.

En el siguiente diagrama (Figura 7.8) se muestra el proceso de búsqueda de una empresa en el sistema. Como vemos, el ciclo de ejecución precisa de dos interacciones diferentes por parte del usuario.

Por un lado, el usuario deberá introducir una cadena de texto en el campo de búsqueda,

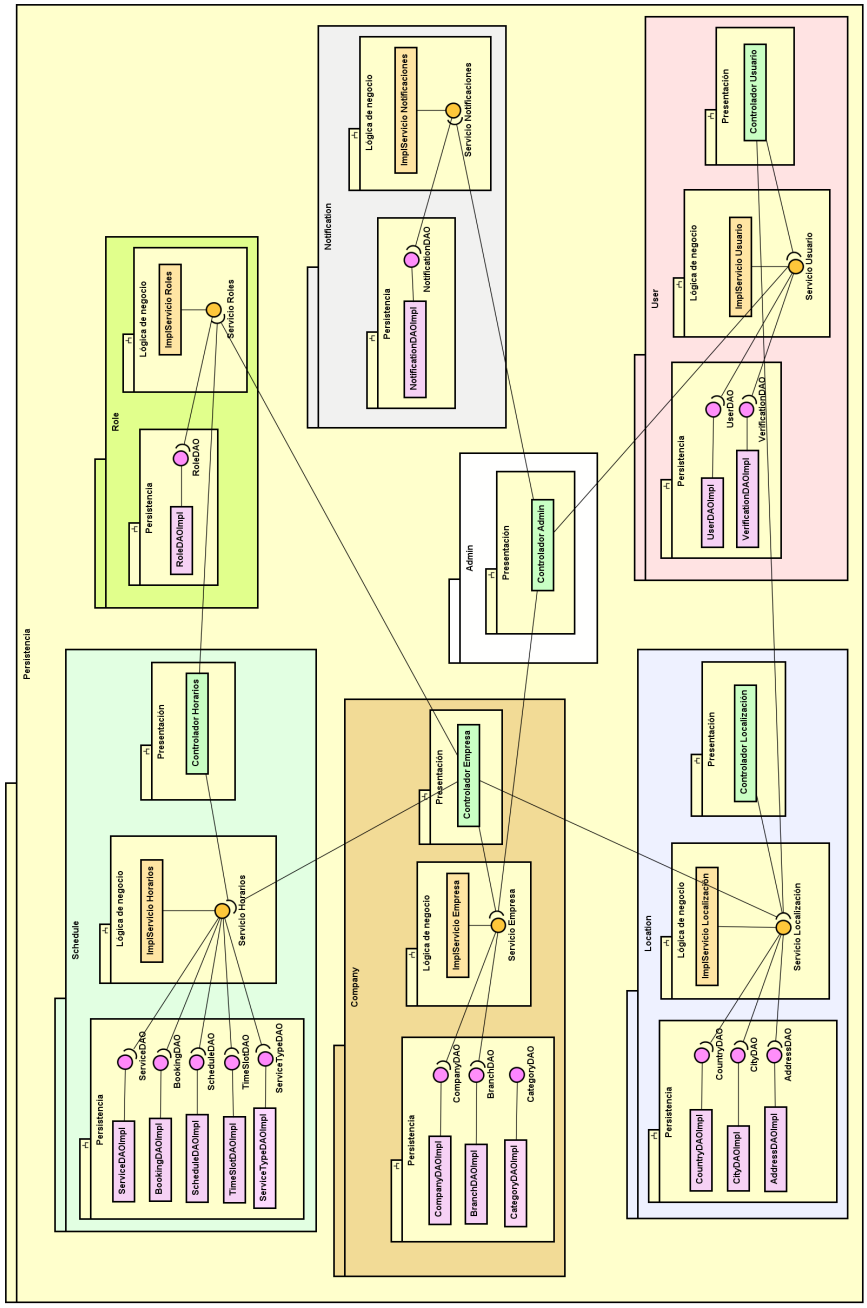


Figura 7.6: Diagrama de la arquitectura en capas de la aplicación

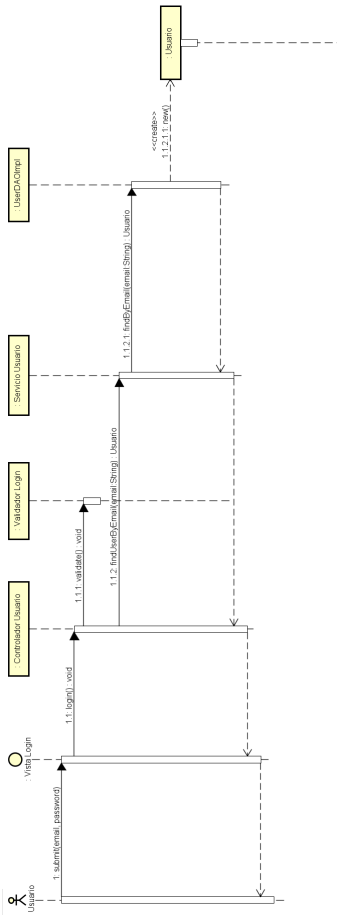


Figura 7.7: Diagrama de secuencia de identificación de usuario

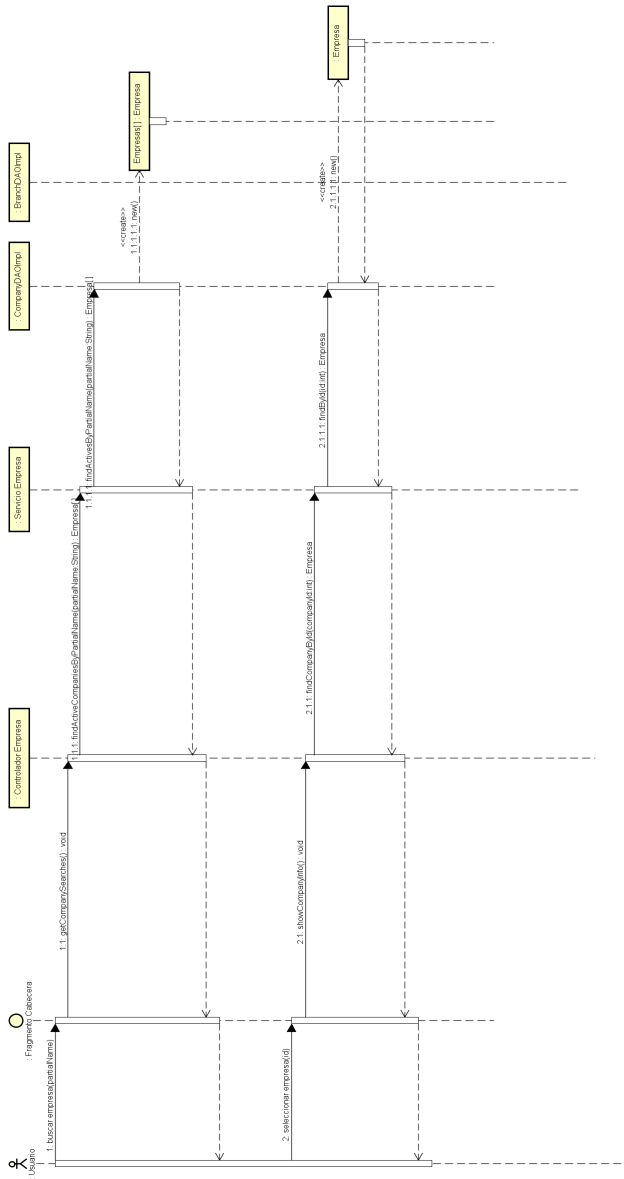


Figura 7.8: Diagrama de secuencia de búsqueda de empresas

enviando así de forma dinámica operaciones **AJAX** hacia el servidor para obtener la lista de empresas que coincidan con el patrón de texto.

Po otro lado, una vez que el usuario visualice en la lista la empresa deseada, deberá seleccionar el elemento de la lista para mandar una petición al controlador que devuelva la ficha de esa empresa.

Por último tenemos un diagrama de secuencia, más complejo que los anteriores, en el que se aborda la reserva de servicios por parte del usuario.

Como vemos en la Figura 7.9, esta historia de usuario se divide en varios pasos.

El primero de ellos consiste en acceder a la zona de reservas, a través de una menú lateral con las distintas secciones dentro de la vista de empresa. Una vez seleccionada la sección de reservas, se mostrará al usuario una lista con las distintas oficinas de la empresa.

A continuación, el usuario debe seleccionar la oficina que desee, para obtener una vista del calendario de servicios en el que visualizar la disponibilidad horaria de cada uno de ellos.

El siguiente paso será seleccionar uno de los servicios que se muestren como disponibles, teniendo para ello un indicador en la parte superior que explica el estado de los servicios según el color con el que se muestran. En este punto, los servicios marcados como no disponibles no serán accesibles para el usuario, por lo que evita cualquier tipo de malentendido a la hora de gestionar una nueva reserva.

Una vez elegido un servicio, el último paso será la selección del trabajador con el que realizar la reserva, puesto que cada servicio podrá ser realizado por un número indeterminado de empleados.

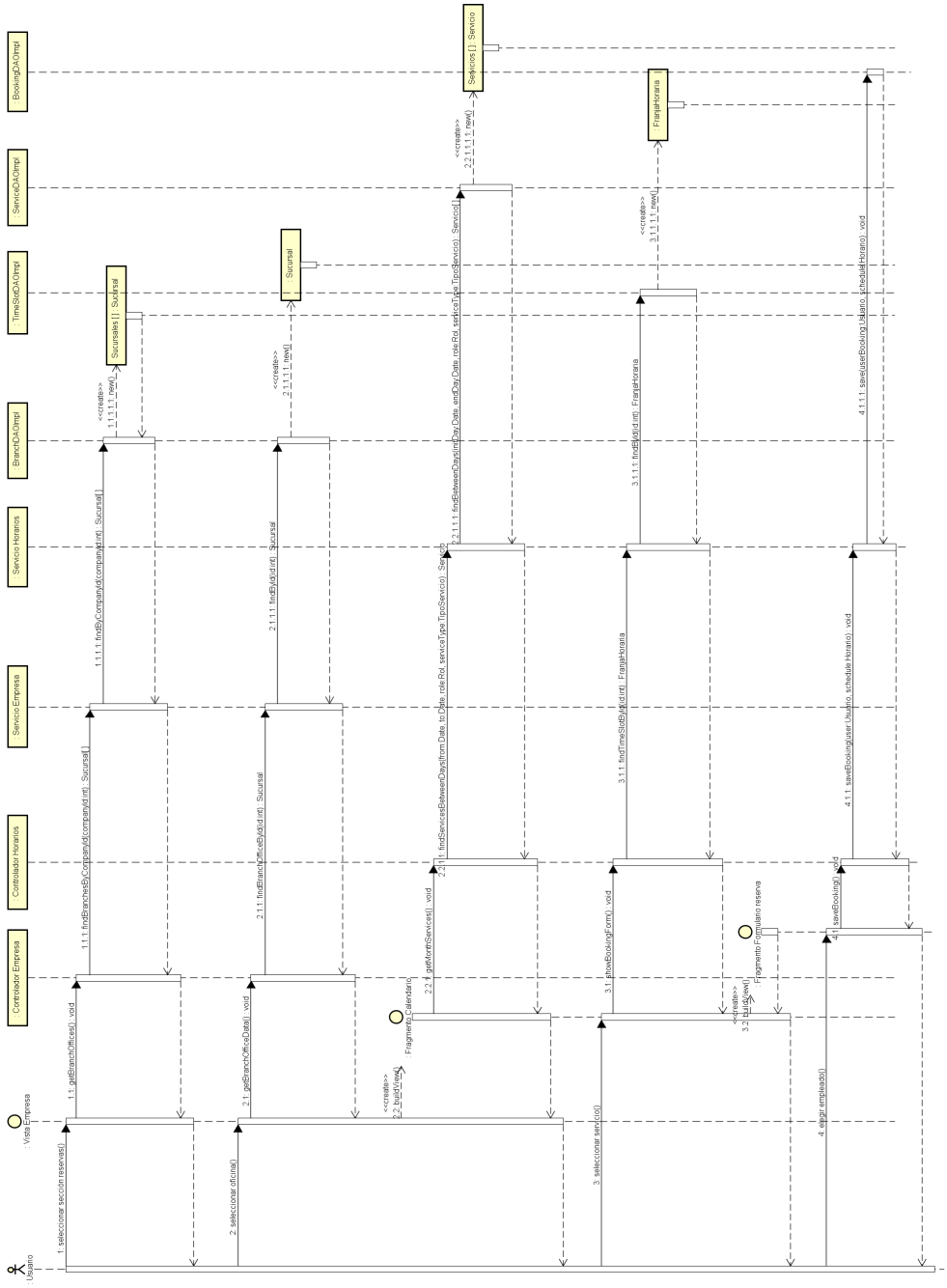


Figura 7.9: Diagrama de secuencia de reserva de servicios

CAPÍTULO 8

Arquitectura de los datos

8.1. Restricciones de la base de datos

Los tamaños y tipos de datos de algunos de los campos de la base de datos se han establecido conforme al volumen 2 del **Catálogo de Estándares de Datos del Gobierno** [17].

8.1.1. Tabla person

Los campos *first_name* y *last_name* no permitirán valores nulos, y tendrán un tamaño máximo de 35 caracteres alfanuméricos.

Los campos *email* y *password* tampoco permitirán valores nulos, pero tendrán un tamaño máximo mayor, de 50 caracteres alfanuméricos. El campo *email* además se define como UNIQUE, lo que evita que haya duplicidad de un mismo valor en registros diferentes.

El campo *nif* será opcional y admitirá un máximo de 9 caracteres alfanuméricos. Se define como UNIQUE para evitar que existan diferentes usuarios con un mismo nif.

El campo *phone* será de tipo alfanumérico con un tamaño máximo de 12 caracteres, ya que así se facilita la internacionalización permitiendo insertar números de teléfono que comiencen por 0, utilizados por ejemplo en el Reino Unido. Este campo admitirá también valor NULL.

El campo *active* será de tipo booleano y será obligatorio, definiendo el valor TRUE como valor por defecto. Este campo permitirá al administrador definir bloqueos de un usuario por

diversas razones.

El campo *admin_role* también será de tipo booleano y obligatorio, pero en este caso su valor por defecto será FALSE, ya que ningún usuario registrado a través de la aplicación debe poder crearse una cuenta con permisos administrativos.

El campo *address_id* será de tipo entero de 4 bytes, y se define como clave foránea de la tabla **address**. Por tanto requiere que, en caso de no tener valor NULL, exista un registro en la tabla **address** con el mismo id que este valor.

8.1.2. Tabla address

Los campos *address1* y *address2* serán de tipo alfanuméricos (varchar) con un número máximo de caracteres de 35. Ambos campos son opcionales, por lo que aceptarán valores NULL.

El campo *zip_code* también será alfanumérico y de carácter opcional, pero de tamaño máximo de 10 caracteres.

El campo *city_id* será de tipo entero de 4 bytes, y se define como clave foránea de la tabla **city**. Este campo es obligatorio, por lo tanto debe existir un registro en la tabla **city** con el mismo id para que el registro de la tabla **address** sea válido.

8.1.3. Tabla city

El campo *name* no permitirá valores NULL, y constará de un máximo de 35 caracteres alfanuméricos. Además, este campo tendrá la propiedad *UNIQUE*, que garantizará que no haya nombres de ciudad duplicados en la base de datos.

El campo *country_id* será de tipo entero de 4 bytes, y se define como clave foránea de la tabla **country**. Este campo es obligatorio, por lo tanto debe existir un registro en la tabla **country** con el mismo id para que el registro de la tabla **city** sea válido.

8.1.4. Tabla country

El campo *name* no permitirá valores NULL, y constará de un máximo de 35 caracteres alfanuméricos. Además, este campo tendrá la propiedad *UNIQUE*, que garantizará que no haya nombres de países duplicados en la base de datos.

El campo *code* constará de un máximo de 2 caracteres alfanuméricos, ya que indicará el código representativo del país según el estándar ISO 3166-1 [18]. Será tanto obligatorio como único este campo, ya que un código no debe representar más de un país.

8.1.5. Tabla verification

El campo *person_id* será de tipo entero de 4 bytes, y se define como clave foránea de la tabla **person**. Este campo es obligatorio, por lo tanto debe existir un registro en la tabla **person** con el mismo id para que el registro de la tabla **verification** sea válido.

El campo *confirmation_key* será un campo de texto, sin limitación de caracteres. El campo será obligatorio y único, ya que debe representar unívocamente al usuario de forma aislada a la tabla **person**, es decir, sin dependencia de los campos de la tabla **person**. Esto permite un nivel mayor de seguridad a la hora de identificar un usuario para su verificación.

8.1.6. Tabla category

El campo *name* será un valor alfanumérico de hasta 30 caracteres, que indicará el nombre de la categoría al que pueda pertenecer una empresa. Este campo se define como UNIQUE para evitar ambigüedades en la categoría de empresa.

8.1.7. Tabla company

Los campos *trade_name* y *business_name* serán caracteres alfanuméricos que definen diferentes nombres de la empresa. El primero, *trade_name*, será el nombre comercial o alias de la empresa, obligatorio y con un máximo de 30 caracteres. El segundo, *business_name*, será el nombre registrado o nombre legal de la empresa, y será obligatorio con hasta 50 caracteres ya que típicamente puede contener una cadena más larga, y además tendrá clave única para evitar la duplicidad.

El campo *email* no permitirá valores nulos, y tendrá un tamaño máximo de 50 caracteres alfanuméricos. Se define además como UNIQUE, lo que evita que haya duplicidad de un mismo valor en registros diferentes.

El campo *cif* será opcional y admitirá un máximo de 9 caracteres alfanuméricos. Se define como UNIQUE para evitar que existan diferentes empresas con un mismo cif.

El campo *active* será de tipo booleano y será obligatorio, definiendo el valor FALSE como valor por defecto ya que se requerirá una previa verificación manual de la empresa.

Este campo además permitirá al administrador definir bloqueos de una empresa por diversas razones.

El campo *create_date* será de tipo timestamp y obligatorio, facilitando el ordenamiento de empresas por su fecha de creación. Este campo tendrá como valor por defecto su momento exacto de creación.

El campo *category_id* será un entero que actuará como clave foránea de la tabla **category**, y permitirá incluir a las empresas dentro de una categoría específica. Su carácter será opcional.

Se definirá una clave única compuesta por los campos *trade_name* y *category_id*, para evitar la sobrecarga de nombres iguales al menos en las mismas categorías.

8.1.8. Tabla branch

El campo *company_id* será la clave foránea que referencia la empresa (tabla **company**) a la que pertenece la sucursal, de carácter obligatorio y tipo entero.

El campo *address_id* será de tipo entero de 4 bytes, y se define como clave foránea de la tabla **address**. Por tanto requiere que exista un registro en la tabla **address** con el mismo id que este valor, puesto que se define como campo obligatorio para facilitar la posterior búsqueda de empresa por ubicación.

El campo *main* será un campo de tipo booleano obligatorio que definirá si la sucursal es la sede de la empresa o no. Su valor por defecto será FALSE.

El campo *phone* será de tipo alfanumérico con un tamaño máximo de 12 caracteres, ya que así se facilita la internacionalización permitiendo insertar números de teléfono que comiencen por 0, utilizados por ejemplo en el Reino Unido. Este campo admitirá también valor NULL.

Se definirán dos claves únicas compuestas: la primera compuesta por *company_id* y *address_id* para evitar la duplicidad, y la segunda compuesta por *company_id* y el campo *main*, lo cuál permitirá definir una única sede principal por cada empresa.

8.1.9. Tabla role

Los campos *person_id* y *branch_id* serán enteros de 4 bytes que actuarán como claves foráneas de la tabla **person** y **branch**, respectivamente. El primero, *person_id*, será opcional ya que permitirá asociar el rol de empleado a una cuenta ya existente. El segundo, *branch_id*,

será obligatorio y por tanto no aceptará valores nulos, asegurando la relación del rol de trabajador con una sucursal específica.

El campo *is_manager* será de tipo booleano y obligatorio, teniendo el valor FALSE por defecto para evitar la asignación errónea de nuevos encargados de empresa.

Esta tabla además tendrá una clave única compuesta por *person_id* y *branch_id*, evitando la duplicidad de un trabajador en una misma empresa, cuando el rol tenga asignado el campo *person_id*.

8.1.10. Tabla notification

El campo *person_id* será un entero de 4 bytes que referencie a un registro de la tabla **person**. Será de carácter obligatorio.

El campo *notif_type* será de tipo alfanumérico y contendrá un máximo de 20 caracteres, además de no aceptar valores NULL. Este campo contendrá un código que identifique el tipo de notificación, definiendo así una lista de notificaciones fijas en la aplicación.

El campo *message* será también obligatorio, pero en este caso de tipo texto, por lo que no habrá limitación de caracteres. Este campo almacenará también un código, en este caso del mensaje concreto dentro de cada tipo de notificación. De esta manera se facilita la internacionalización de los mensajes mostrados.

El campo *read* permitirá controlar si la notificación ha sido leída o no. Su tipo será booleano y su valor por defecto será FALSE, no admitiendo valores NULL en ningún caso.

El campo *create_date* será de tipo timestamp y obligatorio, facilitando el ordenamiento de notificaciones por su fecha de creación. Este campo tendrá como valor por defecto su momento exacto de creación.

8.1.11. Tabla service_type

El campo *name* será de tipo alfanumérico, con un máximo de 20 caracteres y que no aceptará valores NULL.

El campo *description* será opcional, y de tipo texto, por lo que no habrá limitación de caracteres.

El campo *bookings_per_role* será de tipo entero y de carácter obligatorio. Este campo indicará el número máximo de reservas que se pueden hacer por trabajador para un

determinado servicio.

El campo *duration* será de tipo entero, ya que se trata de un dato destinado a almacenar el número de minutos de la duración de cada servicio. Será de carácter obligatorio, tomando por defecto el valor de 60, que son los minutos que componen el período de 1 hora.

El campo *company_id* será un número entero obligatorio que referencie a la tabla **company**.

Los campos *name* y *company_id* compondrán una clave única, evitando la duplicidad de varios tipos de servicio en la misma empresa.

8.1.12. Tabla service

El campo *service_type_id* será un número entero obligatorio que referencie a la tabla **service_type**.

El campo *repetition_type* será un entero que indique el tipo de repetición del servicio, siendo 0 el valor para los servicios no repetidos, y por tanto el valor por defecto. Este campo será obligatorio. Se ha deinifido el valor 1 para las repeticiones de servicio diarias, y el valor 2 para repeticiones por semana. De esta manera quedan identificados como servicios repetidos, lo que evita la innecesaria duplicación de datos para establecer dichas repeticiones.

8.1.13. Tabla time_slot

El campo *start_time* será de tipo timestamp, y servirá para indicar la hora de inicio de cada intervalo de servicio. Será de carácter obligatorio.

El campo *service_id* será una clave foránea de tipo entero, de carácter obligatorio, y que referencie a un elemento de la tabla **service**.

8.1.14. Tabla schedule

El campo *time_slot_id* será una clave foránea de tipo entero, de carácter obligatorio, y que referencie a un elemento de la tabla **time_slot**.

El campo *role_id* también será una clave foránea de tipo entero y obligatoria, que en este caso referencie a la tabla **role**.

8.1.15. Tabla booking

El campo *user_id* será un número entero obligatorio que referencie a la tabla **user**.

El campo *schedule_id* será un número entero obligatorio que referencie a la tabla **schedule**.

Los campos *user_id* y *schedule_id* compondrán la clave primaria de la tabla, ya que identificarán entre ambos unívocamente una reserva realizada.

8.2. Modelo entidad-relación

El modelo entidad-relación final, resultante del diseñado realizado para la estructura de la base de datos, se puede visualizar en la Figura 8.1.

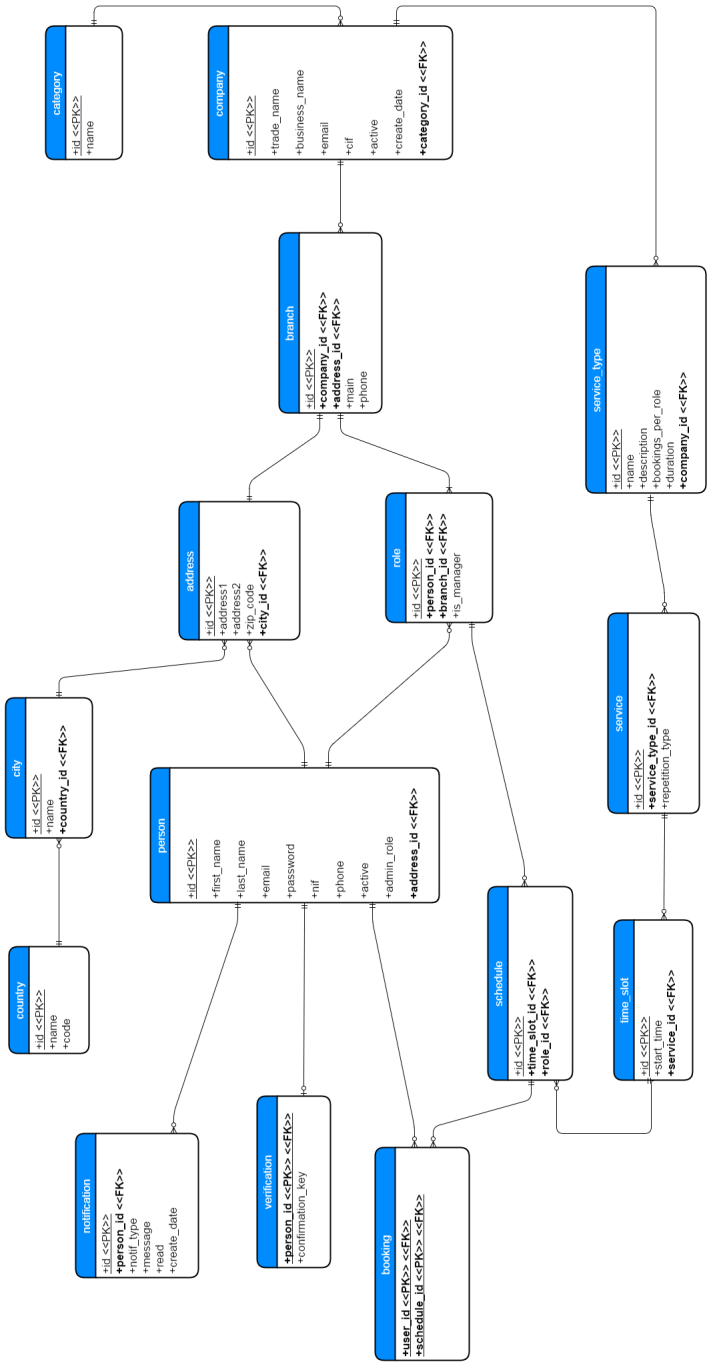


Figura 8.1: Modelo entidad-relación final

CAPÍTULO 9

Pruebas de calidad de software

El proceso de automatización de pruebas realizado para el desarrollo de este proyecto se divide en dos grupos significativos.

En primer lugar tenemos el conjunto de pruebas unitarias sobre la capa de persistencia de la aplicación. Gracias a estas pruebas podemos garantizar una correcta comunicación entre el software implementado y el sistema gestor de la base de datos.

En segundo lugar, aparecen las pruebas de integración. Estas han sido guiadas por la definición de cada una de las historias de usuario, cuyos escenarios se traducen en pasos a seguir en cada ejecución de las pruebas. El nombre de *pruebas de integración* se debe a que su misión es garantizar que todos los elementos que componen una parte del software funcionen correctamente entre sí, es decir, se integren entre ellos para llevar a cabo la funcionalidad requerida. Gracias a las herramientas utilizadas para estas pruebas, hemos podido simular cada una de las interacciones con la aplicación a nivel de interfaz gráfica, para comprobar que funcionen correctamente las operaciones a nivel de servidor o *back-end*.

9.1. Pruebas unitarias

Las pruebas unitarias permiten la validación del correcto funcionamiento de cada módulo del código de forma aislada. En este proyecto, se han limitado este tipo de pruebas a las clases que implementan la capa de persistencia.

La idea reside en especificar varios casos de prueba por cada método, de forma que garanticemos que cumplen con el comportamiento deseado independientemente del resto de métodos. Podemos encontrar todas las pruebas unitarias realizadas en este proyecto en el

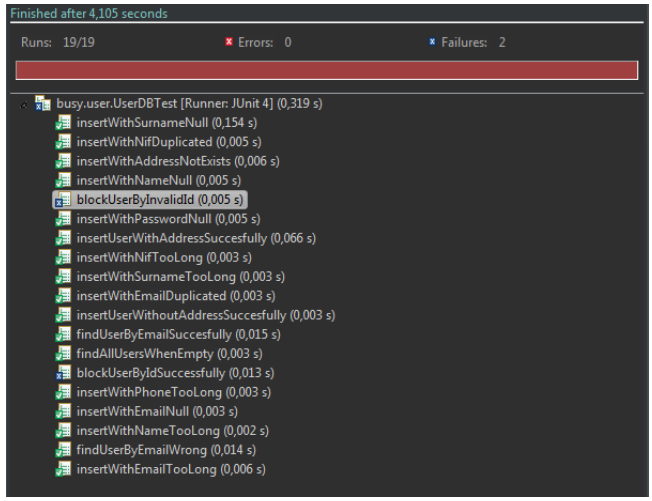


Figura 9.1: Ejecución fallida de *UserDBTest.java*

contenido del código, a través de la ruta: *'busy / src / test / java / busy'*.

Mediante las pruebas sobre consultas de lectura de datos, se han intentado cubrir dos tipos de situaciones.

En primer lugar, la búsqueda a partir de datos nulos o inválidos, es decir, datos que no se encuentran en la base de datos. Para este tipos de casos se comprueba que las operaciones devuelvan un resultado vacío, y sin ningún tipo de error.

En segundo lugar, la búsqueda a partir de datos correctos. En este caso debemos comprobar que el resultado que devuelve la operación no es vacío, y se ajusta tanto a la cantidad de objetos deseados como a la información de estos.

Por otro lado tenemos las pruebas sobre consultas de modificación de los datos, en la que se ha hecho un estudio más intensivo para cada una de las entidades del dominio.

La definición de este tipos de pruebas se ha basado en garantizar que cumplan las restricciones definidas en la base de datos a la hora de crear cada una de las tablas. Entre estas restricciones se encuentran la obligatoriedad de los campos, por la que no se aceptan valores nulos, las referencias correctas de cada una de las claves foráneas, las limitaciones de tamaño de los datos, y las claves únicas de cada tabla.

A continuación, comentaré una de las ejecuciones de pruebas unitarias realizadas en la aplicación, en este caso para la historia de usuario de bloqueo de usuarios.

Lo primero que hacemos a la hora de empezar a desarrollar la historia de usuario para

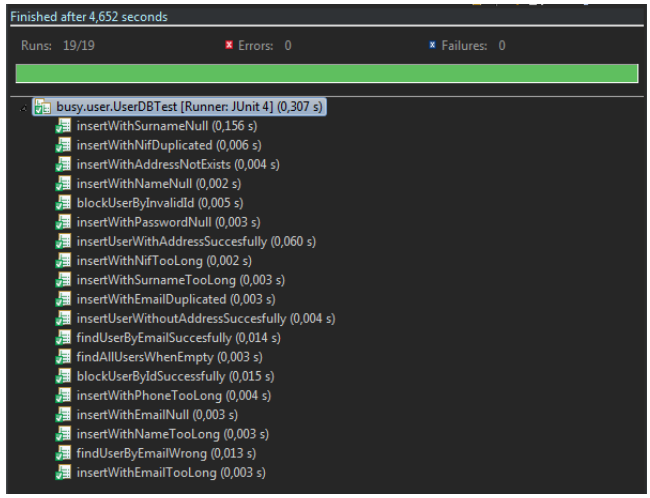


Figura 9.2: Ejecución exitosa de *UserDBTest.java*

bloquear usuarios, es escribir un test. Esto nos fuerza a pensar acerca de cómo se desea que el código se use, se comporte, e interaccione con sus colaboradores, si existiese.

Tras una primera definición de las nuevas pruebas a implementar sobre la entidad del usuario, se muestra en la Figura 9.1 una resultado fallido sobre la ejecución de la suite de pruebas relacionadas con las operaciones del usuario. En este caso se han añadido dos pruebas: la primera, denominada *blockUserByInvalidId()*, comprueba que al intentar actualizar el estado de un usuario que no existe no actualiza ningún dato; la segunda, denominada *blockUserByIdSuccessfully()*, comprueba que al ejecutar la actualización sobre un usuario existente su estado cambia a bloqueado.

Puesto que la funcionalidad no está aún implementada, el motivo del fallo de las pruebas es que lel método llamado no hace nada, por lo que cualquier validación nos devolverá un fallo en este caso. El siguiente paso, según el ciclo de **desarrollo guiado pos tests** [3], consiste en desarrollar el código necesario para que las pruebas pasen en verde, es decir, que la validación definida en ellas se cumpla con éxito.

Una vez implementada la funcionalidad necesaria, como podemos ver en la Figura 9.2, la ejecución de las pruebas pasa en verde, lo que indica que el comportamiento deseado se cumple y las condiciones se validan de forma correcta.

El último proceso en este ciclo TDD se trata de la refactorización del código implementado, lo que nos permite mejorar la estructura del código para eliminar redundancia y *bad smells*.

```
Feature: An admin will be able to block useraccounts
    In order to avoid the misuse of the application
    As a registered admin
    I want to deny the use of the application to a user

    Background:
        Given I am logged as an admin
        And I am on the main admin page

    Scenario: Block a user successfully
        When I click on users section
        Then I should see a detailed list of all the users in the system
        And I should see the user "Usuario1" as active
        When I click on Block in the row of user "Usuario1"
        Then I should see a confirmation message
        And I should see the user "Usuario1" as blocked
```

Figura 9.3: Historia de usuario de bloquear usuarios. Formato Gherkin

9.2. Pruebas de integración

Por otro lado tenemos las pruebas de integración, encargadas de validar que *todas las piezas del puzzle encajen entre sí.*

Ya vimos en el capítulo 5 (Figura 5.13) la descripción de esta historia de usuario de ejemplo. A la hora de definir las pruebas de integración se ha utilizado dicha descripción para, una vez pasada a formato **Gherkin**, utilizarla como prueba de integración para esta funcionalidad.

El formato **Gherkin** [4] se trata de un lenguaje entendible por cualquier usuario sin conocimientos técnicos, con el que se tratan de documentar historias de usuario explicando el comportamiento deseado del software sin entrar en detalles técnicos sobre la implementación de dicho comportamiento.

Podemos ver en la Figura 9.3 la historia una vez escrita en este formato. Para poder ejecutarla como test de integración, se ha utilizado la herramienta **Cucumber** que conecta el fichero de definición de la historia de usuario con el código Java que contiene cada uno de los métodos que implementan los pasos de cada escenario.

La ejecución de este tipo de pruebas difiere de la que hemos visto para las pruebas unitarias, en las que cada test se ejecuta en un orden aleatorio y de forma independiente al resto. En este caso las pruebas seguirán un orden estricto de ejecución. Además, como observamos en la Figura 9.4, en el momento en que uno de los pasos de la prueba resulte en cualquier tipo de error o fallo de validación se detendrá el proceso de ejecución del escenario en curso.

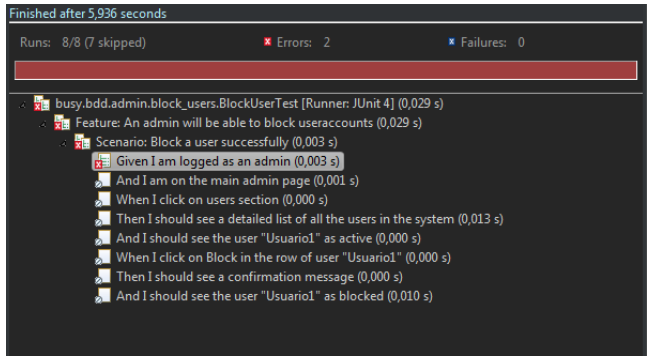


Figura 9.4: Ejecución fallida del test de integración *BlockUserTest.java*

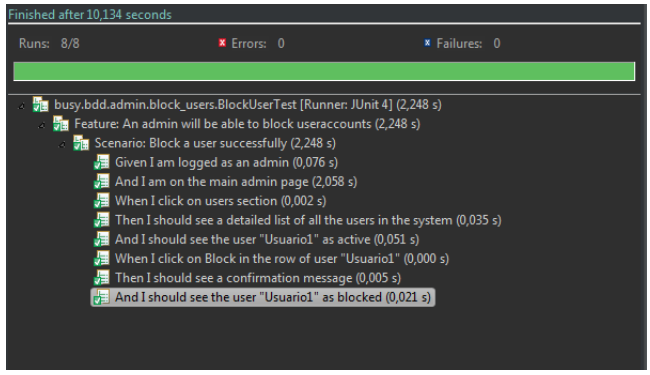


Figura 9.5: Ejecución exitosa del test de integración *BlockUserTest.java*

Esto es debido a que cada paso de un escenario es dependiente del anterior, siendo en su conjunto una sucesión de operaciones hasta llegar al resultado final. Sin embargo, la detención de uno de los escenarios no impide que se ejecuten las pruebas sobre cada uno de los escenarios restantes.

Una vez alcanzados los criterios de aceptación de la historia de usuario, ejecutamos de nuevo las pruebas de integración para comprobar, como vemos en la Figura 9.5, que la validación de comportamiento es correcta.

CAPÍTULO 10

Seguimiento

El origen del proyecto aquí tratado se sitúa en Noviembre del 2015, mes en el que data el primer sprint del desarrollo. Sin embargo, el seguimiento y desarrollo planificado para el proyecto comienza a finales del mes de Enero de 2016. El proceso continuado del desarrollo se puede observar, gracias a una de las gráficas facilitadas por la plataforma *GitHub*, en la Figura 10.1.

En la figura podemos ver la carga de trabajo a lo largo del tiempo sobre el software de la aplicación, es decir, el seguimiento de todos los cambios sobre el código implementado para el desarrollo de este proyecto. Como se observa en la gráfica, el nivel máximo de aportaciones es alcanzado en la semana entre el 26 de Junio y el 2 de Julio. Debemos tener en cuenta para este análisis que la plataforma *GitHub* se basa en el formato de calendario norteamericano, siendo la semana de domingo a sábado.

10.1. Avance del proyecto

Para llevar a cabo la realización de este proyecto, me han ido surgiendo una serie de cuestiones, a las que me he tenido que enfrentar. Para poder resolver dichas cuestiones ha sido preciso plantearme una serie de objetivos a resolver, como son los que expondré a continuación:

- **Hito 1:** Gestión de usuarios
- **Hito 2:** Gestión de empresas
- **Hito 3:** Gestión de reservas

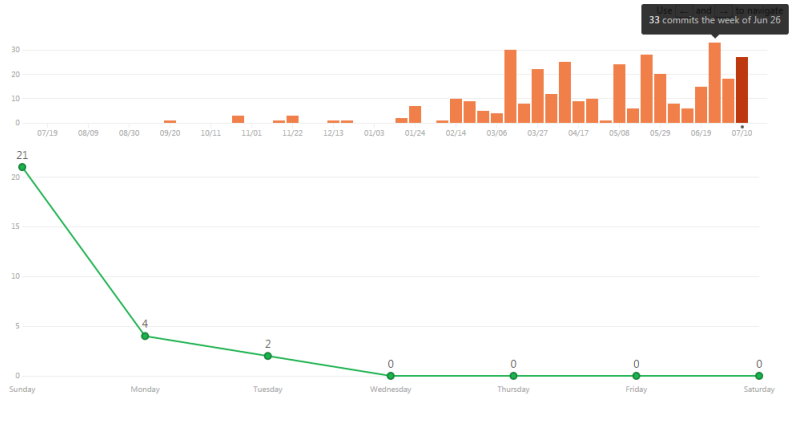


Figura 10.1: Gráfica de evolución de commits

10.1.1. Hito 1: ¿Cómo gestiono una cuenta de usuario?

A través de esta pregunta, surge el primero de los hitos que se pretenden abarcar en este proyecto: la gestión de usuarios.

En primer lugar, me he planteado la necesidad de que un usuario precise de un método de identificación, que permita la interacción con la aplicación bajo una identidad determinada. De esta forma cada operación, desde la creación de nuevas empresas hasta la reserva de servicios, se podría relacionar con el usuario que las realiza.

Además, debido a la necesidad de una zona de administración, la identificación de usuario permitiría mostrar una vista diferente a cada usuario dependiendo de su nivel de privilegios.

En primer lugar, he empezado diseñando los bocetos o *mockups* de la vista de *login* de usuario, el logo de proyecto, y a adquirir conocimientos sobre las herramientas y tecnologías a utilizar.

Uno de los mayores problemas a los que me enfrento en este punto se trata de como establecer el proceso de automatización de pruebas. Para ello he decidido utilizar la herramienta Cucumber, mediante la cual conecto los escenarios de mis historias de usuario escritas en Pivotal Tracker, con el código Java que genero para cada paso de los escenarios.

Dicho código lo conecto con un driver que controle la aplicación web como si de un usuario real se tratase, es decir, interaccionando con cada uno de los elementos web. Para realizar esa conexión dispongo del framework Selenium, y de la librería FluentLenium para

simplificar tareas complejas de automatización de pruebas.

El driver a utilizar para controlar los elementos del DOM será una implementación personalizada de HtmlUnitDriver, a la que he habilitado la ejecución de código JavaScript.

Una vez que todo estaba listo para empezar, comencé a realizar un primer diseño del diagrama entidad relación, para disponer de una base de datos básica para soportar la persistencia de cuentas de usuario.

A partir de los modelos creados desarrollé un primer prototipo en el que la única operación posible fuera la posibilidad de identificarse como usuario en el sistema, entrando a una página vacía en funcionalidad. Esta historia de usuario, junto a las necesarias para comenzar de forma satisfactoria el proyecto aquí reflejado, ocuparon el primer sprint del proceso. Entre estas historias se encuentran los conocidos como *Spikes*, una categoría de historias de usuario aplicable a la búsqueda de información y aprendizaje teórico-práctico de nuevas tecnologías a emplear en el desarrollo de las sucesivas historias de usuario.

En un principio se estableció una velocidad por sprint de 5 puntos historia. Sin embargo, a causa de la baja compatibilidad con el estudio y desarrollo de las prácticas de las asignaturas del primer cuatrimestre del curso en proceso, se interrumpió durante una temporada el desarrollo del proyecto, volviendo al transcurso de su desarrollo hacia finales de Enero.

Como podemos observar en la Figura 10.2, dos de las historias de usuario comenzadas en el primer sprint se finalizaron al comienzo del sprint posterior. Estas dos historias tratan el registro de usuario, y la automatización de los scripts sobre la base de datos en los tests de integración, respectivamente.

Otra de las historias que se incluyen en este sprint es el cierre de sesión de usuarios.

Para el siguiente sprint llevado a cabo, cuyo informe de actividad se puede ver resumido en la Figura 10.3, se incluyen tanto la historia de registro de empresa como la resolución de un error.

Respecto al error, se trata de solventar un problema de seguridad en el proceso de verificación por correo electrónico de una cuenta nueva de usuario.

En el caso del registro de empresa, realicé un primer diseño considerando dos tipos de cuenta de usuario aislados entre sí: la cuenta de cliente receptor de servicios, y la de empresa que ofrece dichos servicios. Sin embargo, durante un primer estudio de esta funcionalidad, es cuando comienzo a pensar en un cambio del diseño y a plantearme preguntas que dieron lugar al inicio del siguiente hito.

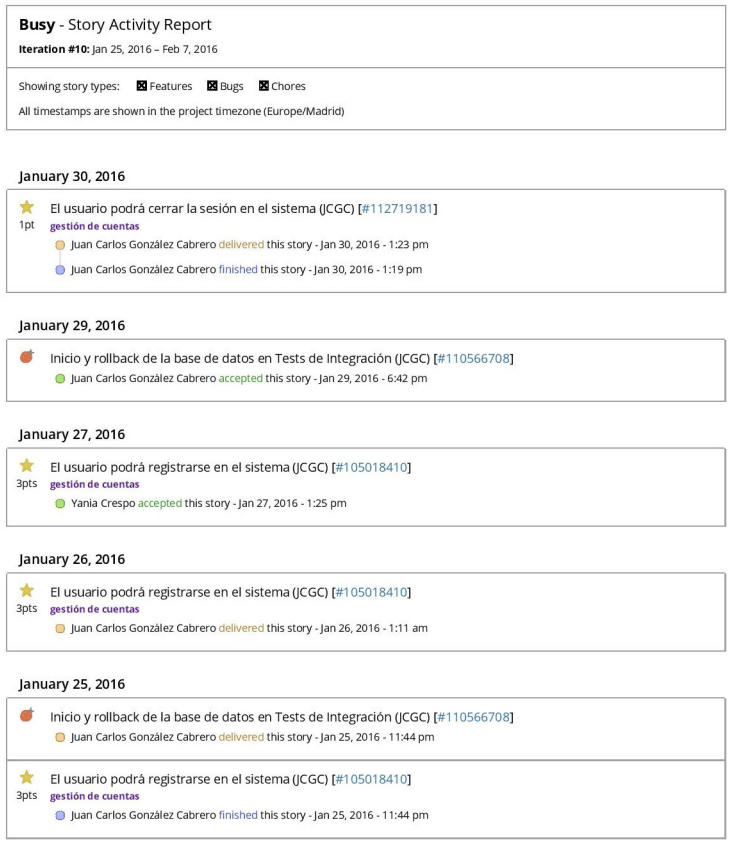


Figura 10.2: Análisis Sprint 2

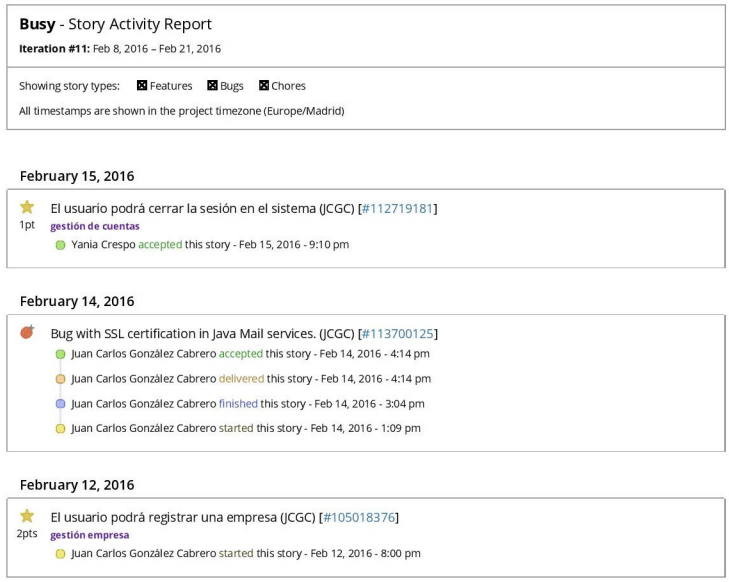


Figura 10.3: Análisis Sprint 3

10.1.2. Hito 2: ¿Cuál es el papel de la empresa?

A la hora de pensar en la empresa como un perfil diferente al usuario habitual, me planteé está cuestión: ¿cuál es el papel de la empresa en mi proyecto? ¿cómo debo enfocar la interacción de un encargado o gestor de empresa con la aplicación?

En un primer momento, enfoqué la empresa como un tipo de usuario diferente, para de esta forma aislar en cierta manera las cuentas de usuario para clientes por un lado, y las cuentas para gestores o encargados de empresa por otro.

Sin embargo, me di cuenta de que el usuario que posee una empresa no sólo vive para dicha empresa, sino que probablemente le interese utilizar la aplicación como un cliente más que busca servicios de otras empresas para reservar, al igual que la utiliza para ofrecer los servicios de su propia empresa.

Otra de las historias de usuario implementada, tratada como una simple tarea dentro del registro de nuevas empresas, ha sido la implementación del sistema de notificaciones de la aplicación. Estas notificaciones serán mostradas a los usuarios según realicen ciertas acciones que impliquen cambios relevantes en la aplicación.

Gracias al nuevo enfoque del sistema a implementar, surgen nuevas historias de usuario, las cuales podemos observar en el informe de la Figura 10.4.

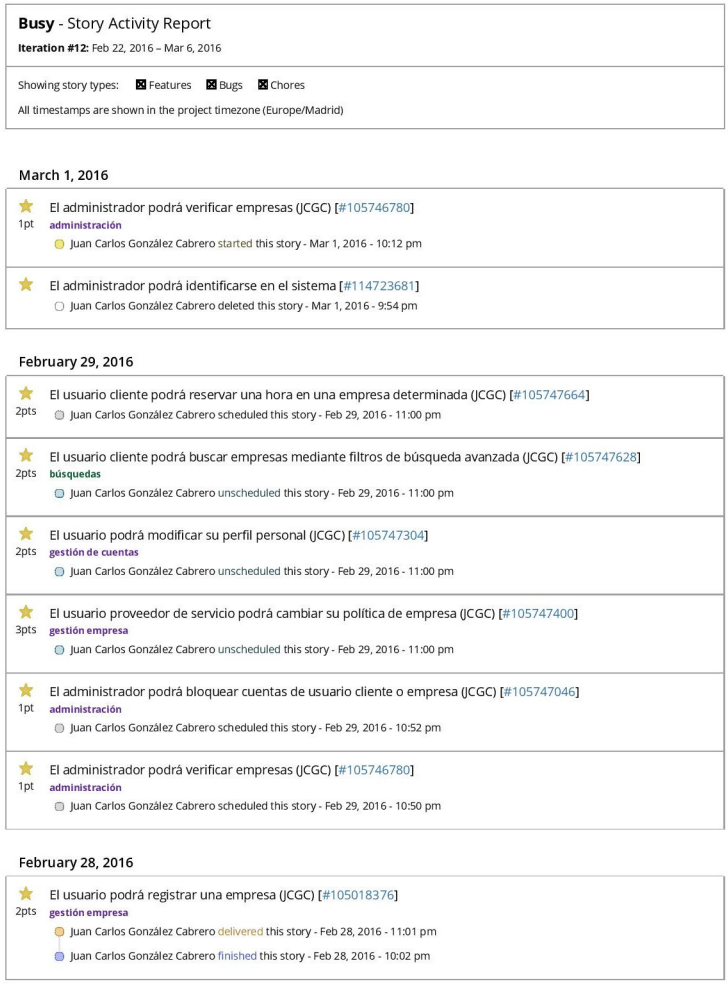


Figura 10.4: Análisis Sprint 4

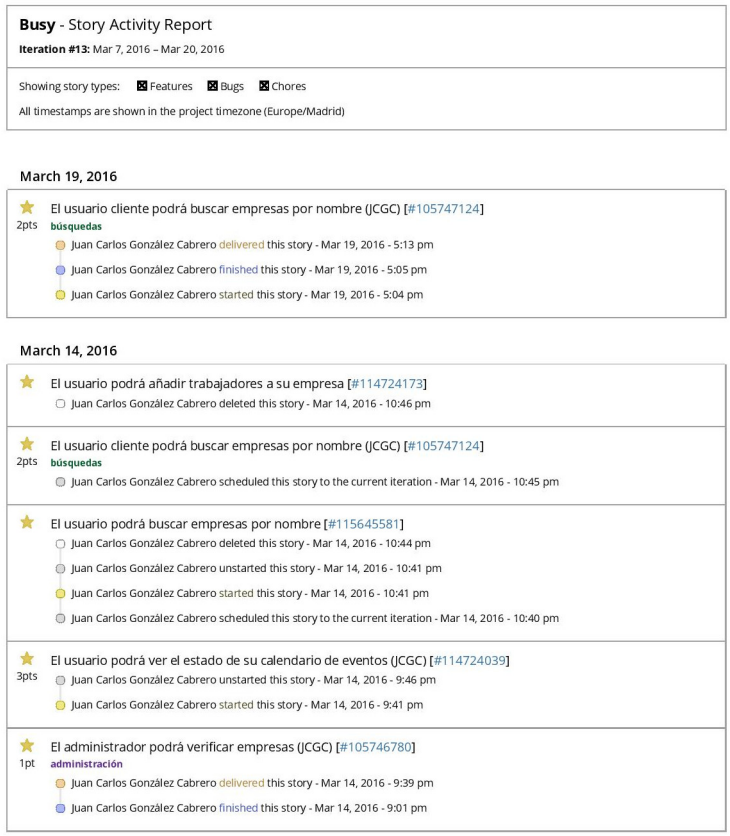


Figura 10.5: Análisis Sprint 5

Una de las historias resultantes, que trata sobre la verificación de empresas a través de la interacción de un usuario administrador, acabó adquiriendo una alta prioridad en medio del sprint en curso. Esto fue debido al bloqueo que suponía la falta de esta funcionalidad, tras la finalización del registro de empresa.

En el sprint siguiente, como vemos en la Figura 10.5, se concluye el desarrollo de la verificación de empresas.

Tras la finalización de esta historia de usuario, se llevó a cabo la búsqueda de empresas creadas en el sistema. Dicha historia alcanzó los criterios de aceptación dentro del período del sprint, por lo que también se dio por finalizada.

La visualización del calendario de eventos de una empresa, por parte del gestor o encargado de dicha empresa, quedó considerado como una funcionalidad relacionada más directamente con la organización de servicios, por lo cuál entraría dentro del siguiente hito.

Figura 10.6: Análisis Sprint 6

Para la planificación del sexto sprint, se prioriza una historia de usuario en la que se pretendía resolver un problema de codificación de caracteres, y de internacionalización de las notificaciones implementadas con el registro de empresas.

Este sprint (Figura 10.6) además ha marcado el inicio de un nuevo hito, en el que se comienza a enfocar el desarrollo hacia la gestión de los servicios de empresa.

10.1.3. Hito 3: ¿Cómo organizar los servicios?

A partir del hito anterior, me planteé como llevar a cabo la interacción de la empresa con el usuario cliente, para facilitar ese proceso de reserva en el que se centra el valor de negocio del proyecto ***buSy***.

Lo primero a implementar para ello fue la visualización de todos los servicios generados, por parte del gerente de empresa. Esta funcionalidad, comenzada en el sprint 6, dio lugar a que me planteara las siguientes historias de de usuario.

En primer lugar pensé en que debía llevar a cabo el desarrollo de la creación de nuevos servicios, para que la anterior historia sobre visualización de servicios cobrara sentido. Sin embargo, para permitir la creación de servicios, se precisa que existan tipos de servicio. Es

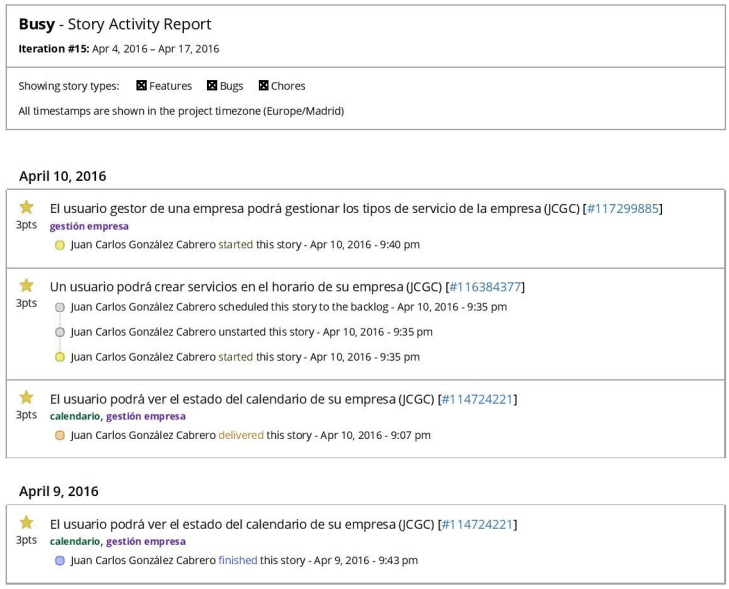


Figura 10.7: Análisis Sprint 7

por ello que en el siguiente sprint, cuyo informe queda reflejado en la Figura 10.7, comienzo el desarrollo de una nueva historia de usuario: la gestión de tipos de servicio de una empresa.

Durante el siguiente sprint (Figura 10.8), además de dar por finalizada la historia anteriormente comentada sobre los tipos de servicio, he aprovechado para priorizar la implementación de un sistema de integración continua de la aplicación. Para ello he utilizado la herramienta Travis CI, que comprueba de forma automática que todo siga funcionando con cada cambio realizado en el proyecto, he incluso permite el despliegue directo de la aplicación una vez pasada toda la batería de pruebas con éxito.

Además de ello, doy por finalizado y comprobado el problema de codificación e internacionalización, tras verificar el comportamiento una vez desplegado en la plataforma Heroku.

Durante el mes de mayo, debido a tareas ajenas al proyecto, y al avance en la realización de la parte teórica del mismo, quedó interrumpido uno de los sprints del proyecto. Respecto al desarrollo, vemos en la Figura 10.9 que durante el sprint 9 se realizó un importante cambio en el modelo de datos de la aplicación.

Debido al profundo estudio sobre el diseño del sistema de creación de servicios, llegué a la conclusión de la falta de viabilidad a la hora de implementarlo mediante el modelo de datos diseñado hasta el momento. Por tanto comencé a modificar el diseño, el cuál podemos

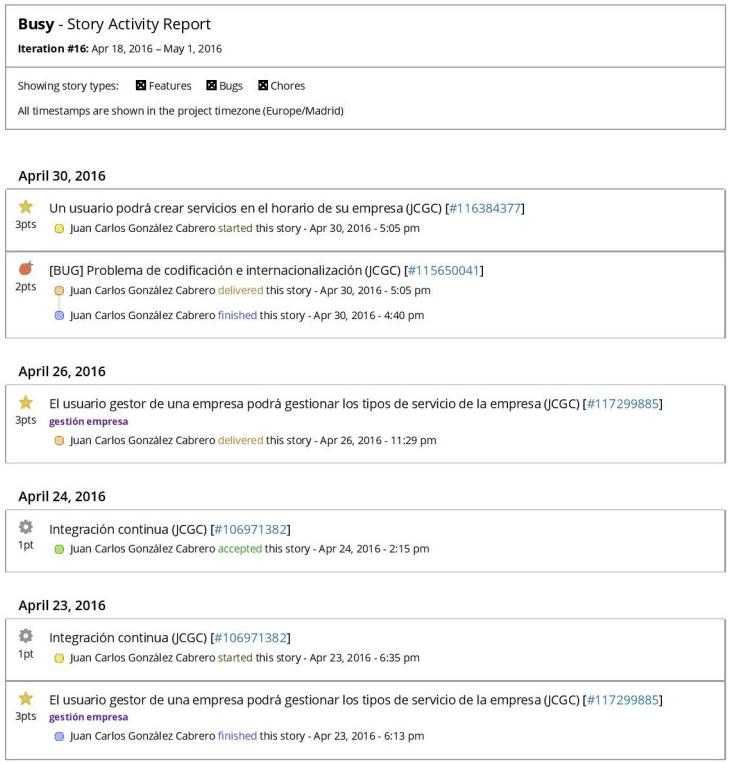


Figura 10.8: Análisis Sprint 8

Figura 10.9: Análisis Sprint 9

ver en la Figura 10.10, hasta llegar a convertirlo en el modelo actual, comentado en el capítulo 8.

Pese a la suspensión del último sprint de Mayo, se llevó a cabo un largo proceso de desarrollo sobre la creación de servicios, cuya historia quedo completada en el siguiente sprint. En dicho sprint, cuyo resumen de actividad visualizamos en la Figura 10.11, comencé la historia de usuario para permitir la reserva de servicios a los usuarios cliente. Además, se incorporó el uso de un motor de plantillas para facilitar la implementación de la capa de presentación y aplicar un patrón de reutilización de dichas plantillas, gracias a la modularidad ofrecida por la herramienta Thymeleaf.

En el sprint undécimo, surgen algunas ideas de mejora y problemas de usabilidad en las historias de usuario implementadas, lo que implica un rechazo de dos de las historias de usuario por parte del *Product Owner*, en este caso la tutora encargada de la supervisión del proyecto.

Podemos observar en la Figura 10.12 el resumen de actividad para la implementación de las ideas o sugerencias de dichas historias de usuario.

10.2. Integración continua

Para esta aplicación se ha implementado un flujo de integración continua mediante la herramienta **Travis CI**.

El flujo de integración continua nos permite pasar nuestra aplicación por un servidor de automatización de pruebas previamente al despliegue, para así evitar pasar a producción una versión del software que esté fallando, y garantizarnos que se mantiene la calidad de código que buscamos mediante los tests automatizados.

Para el proceso de entrega continua llevada a cabo en este proyecto, mediante el cuál se entrega una nueva versión tras cada sprint de desarrollo, se ha utilizado la integración continua como fase previa al despliegue. Es por ello que podemos considerarlo como un proceso subsumido en la entrega continua, que nos garantiza la calidad de software en cada despliegue a producción.

10.3. Problemas de la aplicación

Uno de los problemas ha sido el manejo de la base de datos en PostgreSQL a través de la consola de comandos.

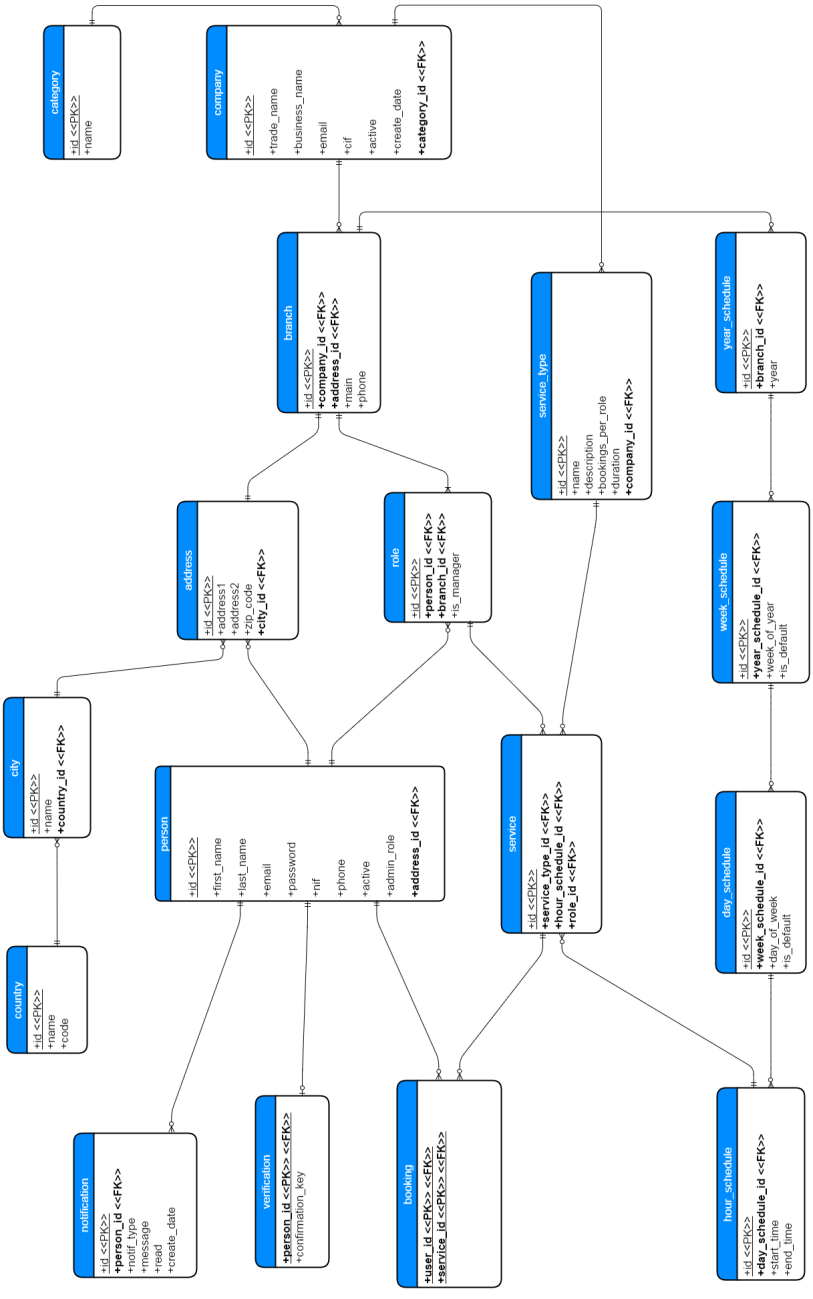


Figura 10.10: Modelo entidad-relación previo a la actualización

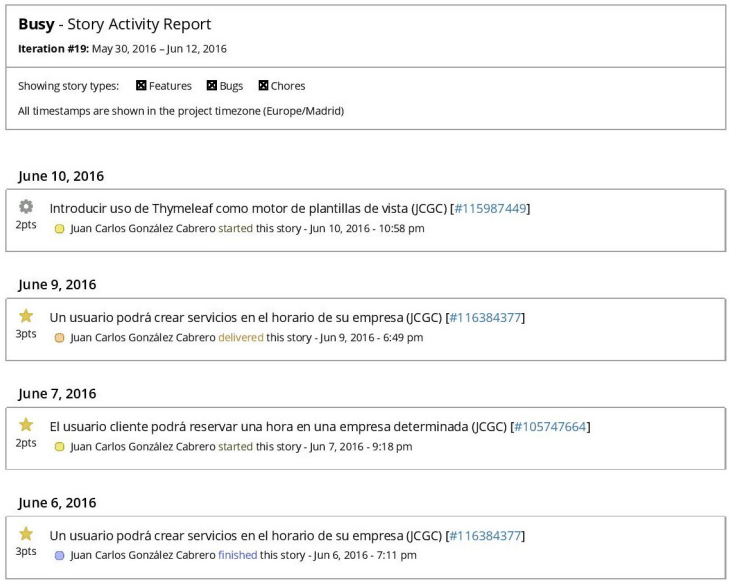


Figura 10.11: Análisis Sprint 10

El primer inconveniente fue la imposibilidad de replicación de la base de datos local sobre la plataforma remota de SaaS en Heroku, lo cuál dió lugar a una conversación durante varios días con el servicio de soporte de Heroku, los cuáles realizaron algunas actualizaciones en su herramienta de integración con la interfaz de línea de comandos, *Heroku Toolbet*.

Otro de los problemas trabajando con PostgreSQL fue la incorrecta codificación de los datos al ejecutar sentencias de persistencia contra la base de datos a través de línea de comandos. La base de datos de este DBMS tiene dos sistemas de codificación, uno para la codificación interna del servidor de base de datos, y otro para el cliente. Ambos pueden estar configurados con diferentes formatos de codificado, los cuáles serán identificados internamente por las propiedades *SERVER_ENCODING* y *CLIENT_ENCODING*, respectivamente.

La configuración por defecto de PostgreSQL es tener el formato del servidor como UTF-8, mientras que el formato del cliente es WIN1252. Esto último es un problema ya que mientras mediante el formato UTF-8 se pueden representar cualquier carácter Unicode, el cuál representa un estándar de codificación y permite disponer de un lenguaje universal de carácteres, el formato WIN1252, también conocido como Windows-1252 ó CP-1252, se limita a la codificación de carácteres del lenguaje latino solamente, con lo cuál reduce considerablemente las posibilidades de internacionalización de los datos. Además de ello, el hecho de utilizar diferentes formatos implica distintos mecanismos de codificación y decodificación, lo cuál podría llevar a inconsistencias en la persistencia de datos.

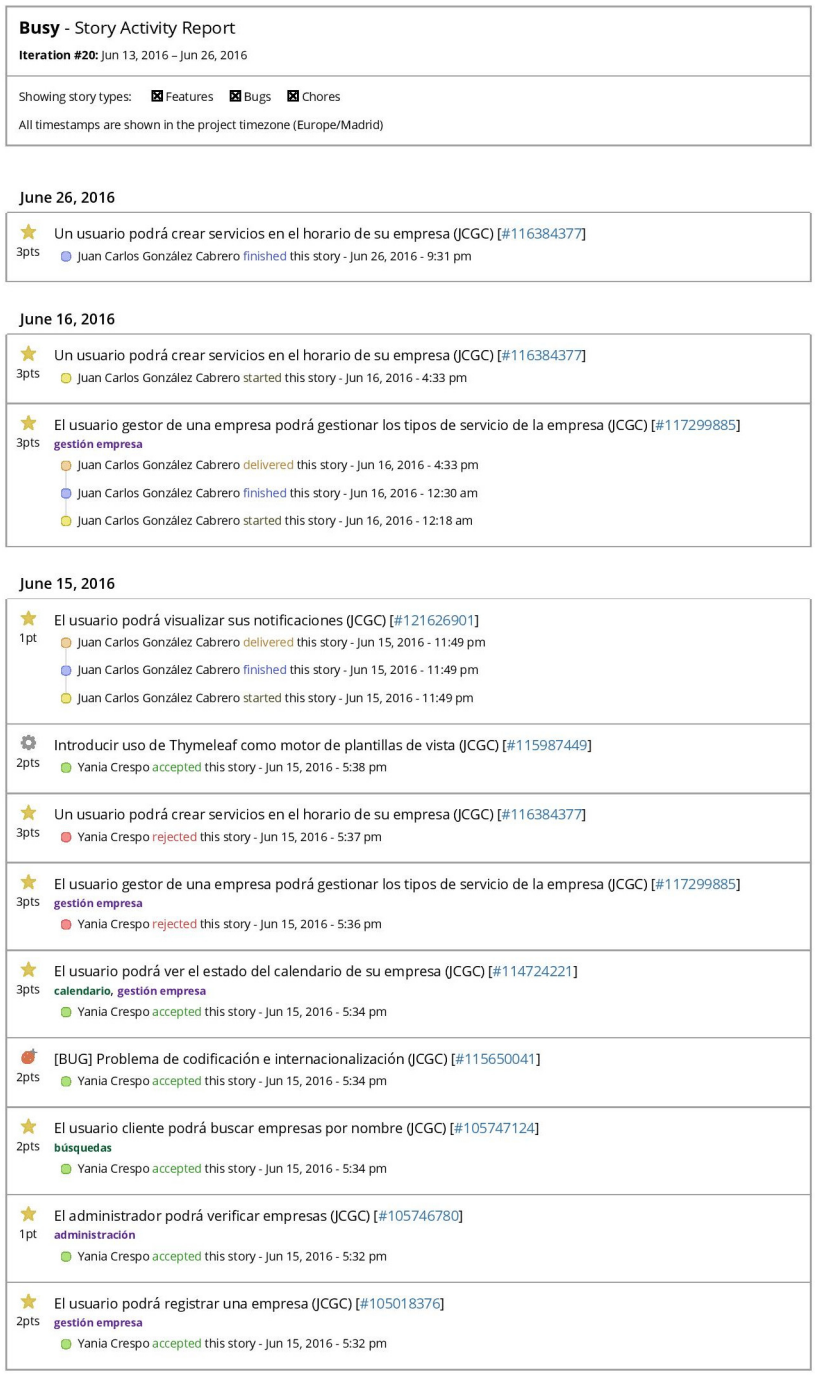


Figura 10.12: Análisis Sprint 11

Spring JDBC configura automáticamente el sistema de codificación de la parte de cliente a la hora de acceder a la base de datos, siendo UTF-8 el formato por defecto, pero configurable mediante el archivo de propiedades de Spring. Para el acceso a través de línea de comandos, se ha precisado añadir una sentencia que modifique el formato de codificación de cliente, previo a la inserción de datos, para mantener correctamente el formato UTF-8 de los datos persistidos. Dicha sentencia es la siguiente:

```
1    SET CLIENT_ENCODING TO 'UTF-8';
```

Otro de los inconvenientes surgidos respecto a la base de datos, fue la ejecución de scripts SQL en la automatización de los tests de integración. Mediante el framework DBUnit se nos permite definir scripts para que se ejecuten previamente en los tests, sin embargo la anotación que implementa esta funcionalidad está diseñada únicamente para tests unitarios. La solución ha sido implementar un algoritmo básico de lectura de ficheros en la clase de la cual heredan las implementaciones de los pasos de cada test, además de inyectar la clase plantilla de Spring para las operaciones con JDBC.

CAPÍTULO 11

Conclusiones

11.1. From Waterfall to Agile

Una de las cosas que he aprendido en la realización de este proyecto, ha sido que el enfoque de las metodologías ágiles ayuda a mejorar incrementalmente, a crecer profesionalmente y que tus proyectos crezcan contigo.

Un ciclo de vida ágil propone casi todo lo contrario a un ciclo de vida en cascada. En primer lugar, un desarrollo en cascada requiere una capacidad de planificación que podríamos catalogar como adivinatoria, puesto que aboga por el diseño previo y la gestión de recursos del proyecto en su totalidad.

Este razonamiento implica aumentar de una forma considerable los riesgos a la hora de embarcarse en un nuevo proyecto:

En primer lugar, hay que tener en cuenta los riesgos de estimación de recursos ya que, según la encuesta realizada por la compañía *McKinsey* [5], un 66 por ciento de los proyectos se realizan por encima del presupuesto estimado, y un 33 por ciento no llegan a tiempo, superando así la estimación temporal planificada.

En segundo lugar, aparecen los riesgos técnicos o de calidad de producto, puesto que al planificar la fase de pruebas como fase final del proyecto, no se puede tener la seguridad de que no vaya a fallar todo al final. Además, particularmente en proyectos software, es vital realizar una serie de tests para comprobar que no se está rompiendo nada con la implementación de nuevas funcionalidades.

En último lugar, se presentan los riesgos sobre requisitos del producto. Para explicar este

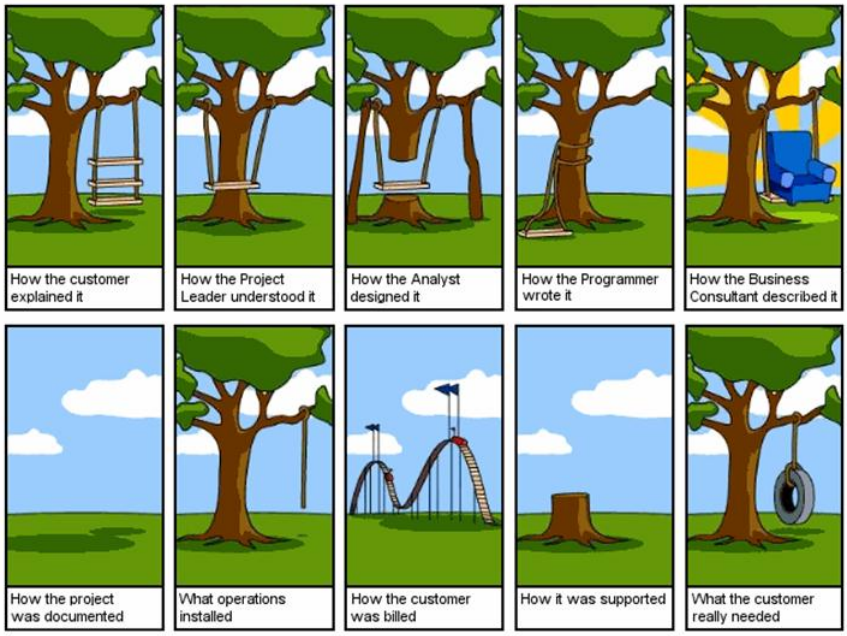


Figura 11.1: Desarrollo de un proyecto en cascada [37]

tipo de riesgo, podemos observar en la Figura 11.1 una famosa ilustración sobre el proceso de un proyecto en cascada. Esta imagen nos explica como van cambiando los requisitos en el transcurso de un proyecto, debido a la diferente visión o interpretación de cada uno de los miembros de un proyecto, e incluso del propio cliente. Además de ello, es necesario tener en cuenta que la visión del cliente puede cambiar en cualquier momento, y es preciso que conozca el progreso del proyecto para moldear esos requisitos cambiantes que van a definir el objetivo de nuestro proyecto final.

Otra de las desventajas de una metodología en cascada frente a la metodología ágil, estriba en la falta de visión a lo largo del proyecto. Es complicado saber en que punto se encuentra cada fase, y hasta dónde debe llegar. De esta forma, en requisitos sencillos podemos atascarnos llevando a cabo una tarea más compleja de la necesaria, o el caso contrario, implementar una versión sencilla de algo importante en nuestro proceso de desarrollo. En cambio en la metodologías ágiles definimos, de una forma más o menos clara y precisa, cuando parar y hasta dónde debe llegar el alcance de la funcionalidad. Para ello se consensúa entre el equipo una serie de criterios de aceptación, en las reuniones de planificación de cada sprint, justo al principio de dichos sprints.

Otro problema derivado de la planificación en cascada, consiste en la reducción de la calidad del producto software. Debido a que, como se comentó anteriormente, la fase de pruebas del proyecto se encuentra al final del proceso, esto puede llegar a ser motivo de

recortar, o incluso eliminar, las tareas de validación previas a la puesta en producción. De esta manera una mala planificación de recursos temporales puede desencadenar en una crítica disminución de la calidad.

Por último, una de las mayores aportaciones de las metodologías ágiles al proceso de desarrollo de un proyecto, se trata de la rápida y sencilla adaptación al cambio. Gracias a las reuniones típicas de las metodologías ágiles, podemos iterar e incluir mejoras o eliminar fallos que se han sucedido en el último sprint. Por ejemplo, una reunión de demostración, o simplemente *demo*, puede desencadenar en un cambio propuesto por el *Product Owner*, que será el encargado de la comunicación con el cliente, o la representación del mismo. En las reuniones de retrospectiva del sprint, o *retros*, se puede aportar ideas para mejorar tanto el trabajo en equipo como la relación entre sus miembros. En una reunión de refinamiento del sprint, o *grooming*, se puede decidir que historias de usuario prioriza en la pila de historias pendientes, o *Backlog*. En una charla diaria de información de la actividad de cada uno, o *daily*, se puede decidir cambiar de tareas, o proponer una ayuda a un compañero para salir de algún tipo de bloqueo. En el caso de las charlas de preparación del siguiente sprint, o *sprint plannings*, el equipo puede plantearse añadir una nueva historia de usuario, cambiar el flujo de tareas, o incluso adaptar historias de usuario ya fijadas a los requisitos del *Product Owner*.

Podemos observar, gracias a un blog sobre metodologías de gestión de proyectos citado en la bibliografía [29], algunas de las ventajas y desventajas de cada una de las metodologías en desarrollo de software.

11.2. Problemas de planificación y cosas a mejorar

El mayor problema que me encontrado a la hora de planificar el proyecto conforme a la metodología ágil, ha resultado ser la división correcta de las funcionalidades en historias de usuario. Poco a poco he ido aprendiendo a definir correctamente las historias, con un pequeño alcance y unos criterios fijos de aceptación. Gracias a haber podido realizar las prácticas de empresa concurrentemente en un proyecto basado en metodologías ágiles, he ido aprendiendo y aplicando conceptos para mejorar mi capacidad de planificación.

Una de las cosas a mejorar sobre este tema, la cuál me ha sido aconsejada además por mi tutora de proyecto, es una mejor identificación de las historias de usuario que engloban funcionalidades genéricas del sistema, y por tanto pueden y deben estar comprendidas por varias historias de usuario de menor tamaño. Estas historias de usuario son conocidas como *epics*.

Otro de los problemas con los que me he enfrentado ha sido la estimación en puntos-

historia. El hecho de no haber trabajado hasta este año en ningún proyecto de desarrollo ágil, ha supuesto una imprecisión en muchas de las historias de usuario estimadas a lo largo de este proyecto.

Una de las posibles soluciones a este problema, sería enfocar las tareas que *rompen* esa estimación, haciendo que la historia de usuario precise de más recursos. Una vez enfocadas dichas tareas, se debería tomar la decisión de crear una o varias historias de usuario adicionales que engloben cada tarea, siempre y cuando se sigan cumpliendo los criterios de aceptación solicitados para cada historia.

11.3. Líneas Futuras

Debido al tiempo limitado para los proyectos de fin de carrera, enfocados hacia la realización del "Mínimo Producto Viable", queda pendiente la realización de múltiples historias de usuario del *Backlog* del proyecto. Como vemos en la Figura 11.2, la herramienta Pivotal Tracker posee una pila de historias de usuario identificada como *Icebox*, la cual comprenderá todas las historias de usuario del *Backlog* pero que no pasarán al sprint actual, es decir quedan congeladas mientras se encuentren en dicha pila.

En primer lugar, el desarrollo de operaciones de administrador, el cuál debe ser capaz de realizar al menos cualquier acción que pueda hacer un usuario sin los privilegios de administración.

Otra de las funcionalidades a implementar será la gestión de horarios de los trabajadores. De esta forma se podrán identificar las vacaciones de cada trabajador, los festivos de la empresa, y las franjas horarias de apertura de negocio, para así permitir el uso de la aplicación como una completa herramienta de gestión de horarios, además de para la gestión de reservas.

La posibilidad de editar la información tanto de una cuenta de usuario como de una empresa registrada, es también algo básico que aportará una mayor libertad de configuración al usuario.

Otra de las mejoras pensadas para la aplicación será la aplicación de filtros, tanto para las búsquedas de empresas por parte de un usuario convencional, como para la búsqueda de cualquier elemento editable o visible en la zona de administración.

Una de las ideas más ambiciosas para este proyecto, se trata de la monetización de los servicios contratados por los clientes, mediante la implementación de un sistema de pago que dará la posibilidad al usuario de pagar desde la aplicación. Este sistema dará lugar a la necesidad de una capa de seguridad, tanto para el proceso de cobro como en el tratamiento de morosidad, lo que podrá dar lugar a bloqueos y restricciones de reservas.

Figura 11.2: Icebox de Pivotal Tracker

Para la adaptación de la aplicación a todos los públicos, y para facilitar su accesibilidad, se implementará una versión del proyecto como aplicación móvil, siguiendo la misma línea de diseño que la actual versión como aplicación web.

Por último, y debido a la visión que aportará el proyecto ***buSy*** a algunas empresas como plataforma para buscar el servicio más barato, o que mejor se ajuste a las necesidades del cliente en cuanto a horario o localización, se buscará una solución para mejorar la imagen de cada empresa. Dicha solución será ofrecer a los usuarios la posibilidad de personalizar las páginas de sus empresas, añadiendo secciones como blogs de noticias, ofertas o incluso fotos sobre los servicios. Por medio de esta funcionalidad, se tratará de adaptar la aplicación hacia un enfoque de red social.

Bibliografía

[1] Atom. Atom. Accessed: 2016-07-14. 2016. URL: https://atom.io.

[2] LLC Balsamiq Studios. Balsamiq. Rapid, effective and fun wireframing software. Accessed: 2016-07-14. 2016. URL: https://balsamiq.com.

[3] Kent Beck. Test driven development by example. Addison-Wesley, 2002.

[4] behat. writing features - gherkin language. Accessed: 2016-07-14. 2016. URL: http://docs.behat.org/en/v2.5/guides/1.gherkin.html.

[5] M. Bloch, S. Blumberg y J. Laartz. Delivering large-scale IT projects on time. Accessed: 2016-07-06. McKinsey & Company. 2012. URL: http://www.mckinsey.com/business-functions/business-technology/our-insights/delivering-large-scale-it-projects-on-time-on-budget-and-on-value.

[6] Iskren Ivov Chernev. Moment JS library. Accessed: 2016-07-09. 2013. URL: http://momentjs.com/docs/.

[7] Mike Cohn. User Stories Applied. Addison-Wesley, 2004.

[8] Szczepan Faber y col. Mockito. Accessed: 2016-02-16. 2016. URL: http://mockito.org.

[9] The Apache Software Foundation. Maven - Welcome to Apache Maven. Accessed: 2016-07-13. 2016. URL: https://maven.apache.org.

[10] The Eclipse Foundation. Eclipse. Accessed: 2016-07-14. 2016. URL: https://eclipse.org/home/index.php.

[11] The jQuery Foundation. jQuery. Accessed: 2016-07-14. 2016. URL: https://jquery.com.

[12] Armando Fox y David Patterson. Engineering Long-Lasting Software. Strawberry Canyon LLC, 2012.

[13] Git. Git. Accessed: 2016-07-13. 2016. URL: https://git-scm.com.

[14] Inc. GitHub. GitHub. Accessed: 2016-07-14. 2016. URL: https://github.com.

[15] The PostgreSQL Global Development Group. PostgreSQL 9.3.10 Documentation. Accessed: 2016-04-12. Mar. de 2016. URL: http://www.postgresql.org/docs/9.3/interactive/index.html.

[16] Red Hat. Openshift: PaaS by Red Hat. Accessed: 2016-04-12. 2016. URL: https://www.openshift.com.

[17] e-Government Interoperability Framework. Government Data Standards Catalogue. Accessed: 2016-07-14. 2015. URL: http://xml.coverpages.org/govtalkCat2.pdf.

[18] ISO. Country Code in ISO 3166. Accessed: 2016-07-14. 1974. URL: https://www.iso.org/obp/ui/.

[19] jdewit. Timepicker Component for Twitter Bootstrap. Accessed: 2016-07-09. 2012. URL: https://github.com/jdewit/bootstrap-timepicker.

[20] Rod Johnson y col. Spring Framework Reference Documentation. Accessed: 2016-04-12. Pivotal Software, Inc. 2014-2015. URL: http://docs.spring.io/spring/docs/current/spring-framework-reference/html.

[21] JUnit. JUnit. Accessed: 2016-07-10. 2016. URL: http://junit.org/junit4/.

[22] Tomas Kirda. jQuery Autocomplete. Accessed: 2016-07-09. Devbridge Group. 2013. URL: https://github.com/devbridge/jQuery-Autocomplete.

[23] Cucumber Ltd. Cucumber. Accessed: 2016-07-13. 2016. URL: https://cucumber.io.

[24] JGraph Ltd. Flowchart Maker & Online Diagram Software. Accessed: 2016-07-14. 2016. URL: https://www.draw.io.

[25] Neil Middleton y Richard Schneeman. Heroku: Up and Running. O'Reilly Media, Inc., 2013. ISBN: 9781449341381.

[26] Oracle. Java JDBC API. Accessed: 2016-05-13. 2016. URL: https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/.

[27] Henry Oswald y col. ShareLatex Documentation. Accessed: 2016-07-09. ShareLatex. 2016. URL: https://www.sharelatex.com/learn.

[28] Mark Otto y Jacob Thornton. Bootstrap. Accessed: 2016-07-14. 2016. URL: http://getbootstrap.com.

[29] Lana Pavkovic. Top 4 Project Management Methodologies And When To Use Them. Accessed: 2016-06-11. 2016. URL: https://yanado.com/blog/top-4-project-management-methodologies-and-when-to-use-them/.

[30] Inc. Pivotal Software. Pivotal Tracker. Accessed: 2016-07-14. 2016. URL: https://www.pivotaltracker.com.

[31] Inc. Pivotal Software. Spring Framework. Accessed: 2016-07-13. 2016. URL: https://projects.spring.io/spring-framework/.

[32] PosgreSQL-es y Rafael Martinez. PostgreSQL. Accessed: 2016-06-20. 2016. URL: http://www.postgresql.org.es.

[33] Sergey Romanov. Bootstrap Calendar. Accessed: 2016-07-09. 2013. URL: https://github.com/Serhioromano/bootstrap-calendar.

[34] Salesforce. Cloud Application Platform - Heroku. Accessed: 2016-04-12. Mar. de 2016. URL: https://www.heroku.com.

[35] Salesforce. Heroku Toolbet. Accessed: 2016-06-29. 2016. URL: http://devcenter.heroku.com/articles/getting-started-with-java#set-up.

[36] Salesforce. Java on Heroku. Accessed: 2016-07-12. 2016. URL: https://devcenter.heroku.com/categories/java.

[37] Salesforce. Waterfall vs Agile Salesforce implementation. Accessed: 2016-06-11. 2013. URL: https://lasalesforce.wordpress.com/2013/09/07/waterfall-vs-agile-salesforce-implementation/.

[38] SeleniumHQ. Selenium WebDriver. Accessed: 2016-07-13. 2016. URL: http://www.seleniumhq.org/projects/webdriver/.

[39] ShareLaTeX. ShareLaTeX. Accessed: 2016-07-14. 2016. URL: https://www.sharelatex.com.

[40] DbUnit Developer Team. DbUnit. Accessed: 2016-04-12. Mar. de 2016. URL: http://dbunit.sourceforge.net.

[41] The Thymeleaf Team. Thymeleaf. Accessed: 2016-07-14. 2016. URL: http://www.thymeleaf.org.

[42] TutorialsPoint. Java Design Patterns. TutorialsPoint, 2015.

[43] Wenzhixin. Multiple Select library. Accessed: 2016-07-09. 2013. URL: http://wenzhixin.net.cn/p/multiple-select/docs/.

APÉNDICE A

Manual de despliegue

Para poder desplegar la aplicación de ***buSy***, necesitaremos una plataforma que soporte un servidor web embebido de Tomcat 7 o superior, que pueda lanzar la ejecución del paquete .jar con la aplicación.

Como plataforma de despliegue PaaS (Platform as a Service) utilizaré Heroku, la cuál ofrece una capa de servicio gratuita y soporta el lenguaje Java. Heroku dispone de un entorno de administración a través de línea de comandos basado en Ruby on Rails.

Lo primero que necesitaremos será tener instalado Ruby On Rails en nuestro sistema.

A continuación debemos instalar la herramienta Heroku Toolbet, que nos permitirá poder configurar nuestro entorno, la cuál se encuentra disponible en la sección *Set Up* [35] de la guía de Heroku.

Una vez que tengamos todo instalado, y con las rutas de los binarios ejecutables correctamente añadidas al PATH, abriremos un entorno de línea de comandos, para poder entrar con nuestras credenciales de Heroku. Al identificarse en la plataforma veremos algo similar a la Figura A.1.

A continuación, para crear la aplicación en nuestro entorno de Heroku, desde la carpeta

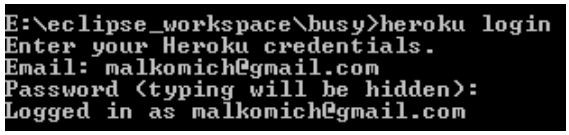


Figura A.1: Login Heroku

raíz de mi proyecto, ejecutamos la orden:

```
heroku create busy-app
```

Al ejecutar dicha orden, lo que hacemos será asociar al repositorio local del proyecto un nuevo repositorio remoto donde se alojará la aplicación como *"busy-app"*. Por tanto ahora tendremos el proyecto alojado tanto en **GitHub** como en **Heroku**.

Para desplegar la aplicación, simplemente sincronizamos la rama master de nuestro repositorio local de GIT, contra el repositorio remoto alojado en Heroku:

```
git push heroku master
```

Una vez actualizado el código del proyecto en el repositorio de producción, Heroku se encargará automáticamente de compilar y desplegar la aplicación si todo va bien y no se encuentran fallos de aplicación.

Para visualizar el resultado podemos dirigirnos a la URL del proyecto, que en el caso de mi proceso de producción será busy-web.herokuapp.com. En caso de no conocer la URL de despliegue, podemos acceder a la aplicación simplemente lanzando el siguiente comando de ejecución del proyecto:

```
heroku open
```

Una vez hecho esto, podremos visualizar correctamente la aplicación en nuestro navegador predeterminado. Para conocer el flujo de operación a seguir dentro de la aplicación, y a modo de tutorial para un primer contacto, el siguiente apéndice nos mostrará una pequeña guía de pasos a seguir por los usuarios de ***buSy***.

APÉNDICE B

Manual de usuario

Una vez estemos en la página de la aplicación, se nos mostrará una pantalla de identificación de usuarios, tal y como podemos ver en la Figura B.1

Para acceder por primera vez, lo primero que tendremos que hacer es crear una cuenta de usuario, en caso de no disponer de ninguna. Mediante el botón de **Registrarse**, el cual le podemos encontrar en la parte superior derecha de la vista de identificación de usuario, podremos acceder al formulario de registro de nuevos usuarios.

En dicho formulario, cuyo aspecto es el que observamos en la Figura B.2, debemos introducir al menos todos los datos obligatorios. Para saber cuáles son obligatorios y cuáles opcionales, se actualizará la vista de cada campo una vez se pase por ellos sin introducir dato alguno. En caso de ser un campo obligatorio, se añadirá un color rojizo al borde, y un asterisco como símbolo de error. Además, al intentar guardar se indicará de forma visual los

Figura B.1: Página de identificación de usuario

Figura B.2: Página de registro de nuevo usuario

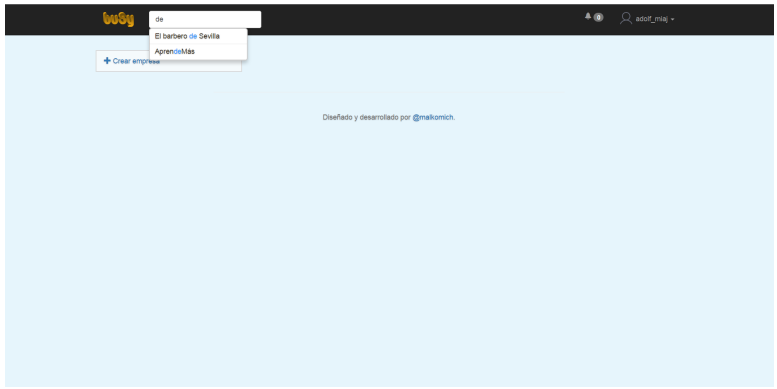


Figura B.3: Página principal de usuario

fallos de cada campo erróneo, con un mensaje explicativo.

En todo momento tendremos un botón para volver a la vista de identificación de usuario, por medio de un enlace inferior al bloque del formulario de registro.

Una vez creado el nuevo usuario, saldrá un mensaje que nos indicará lo que debemos hacer. En este caso, debemos activar la cuenta recién creada por medio de un enlace enviado a la dirección de correo eléctronico facilitada en el proceso de registro.

Una vez activada, podremos acceder desde la ventana de identificación, escribiendo la dirección de correo electrónico como usuario, y la contraseña que hayamos elegido.

Cuando hayamos entrado a nuestra zona de usuario, podremos ver una página con varias zonas de interacción (Figura B.3). En primer lugar tenemos una barra de navegación superior, desde la que podemos buscar el nombre de una de las empresas registradas en el sistema,

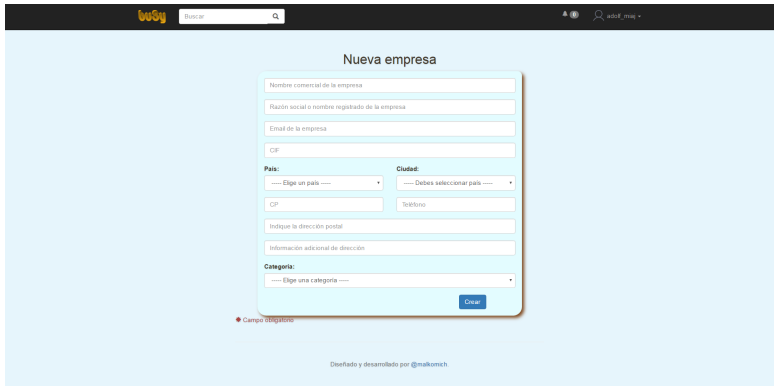


Figura B.4: Página de creación de nueva empresa

y así acceder a su página de información y reservas. Por otro lado, tendremos un icono de notificaciones desplegable, que mostrará la visualización de todas las notificaciones que tengamos actualmente. Inmediatamente a la derecha de dicho icono, podemos desplegar un menú de acciones de usuario, en el que disponemos de una opción para cerrar la sesión de usuario y volver a la vista de identificación de usuario. Por último, tendremos un botón de creación de una nueva empresa.

Dicho botón sólo se encontrará disponible mientras no se haya creado una empresa con el usuario actual. Una vez que presionamos el botón, nos muestra un nuevo formulario como el que vemos en la Figura B.4, en este caso para indicar la información de la nueva empresa que queremos registrar en el sistema.

Su comportamiento de validación es similar al que encontramos en el formulario de registro de usuario.

Una vez hayamos registrado una empresa, podemos acceder al calendario de eventos. Para ello desplegaremos el botón de menú que encontramos donde antes teníamos el botón para crear una empresa. Dentro de este pequeño menú desplegable dispondremos de dos opciones: usuario o empresa. Dependiendo de la opción elegida se cargará una vista u otra.

Seleccionando el nombre de la empresa, accedemos a la página de gestor de empresa, como vemos en la Figura B.5. En esta vista visualizamos un calendario en el que se mostrarán todos los servicios que se hayan creado. Para poder crear nuestro primer servicio debemos disponer de al menos un tipo de servicio. Para ello desplegamos el menú de tipos de servicio que encontramos a la derecha. Tendremos la opción de crear un tipo de servicio nuevo, o de modificar o eliminar alguno de los ya existentes.

Una vez tengamos tipos de servicio creados, podemos acceder al formulario de creación de nuevos servicios, haciendo un simple clic en el número del día que deseemos, o doble

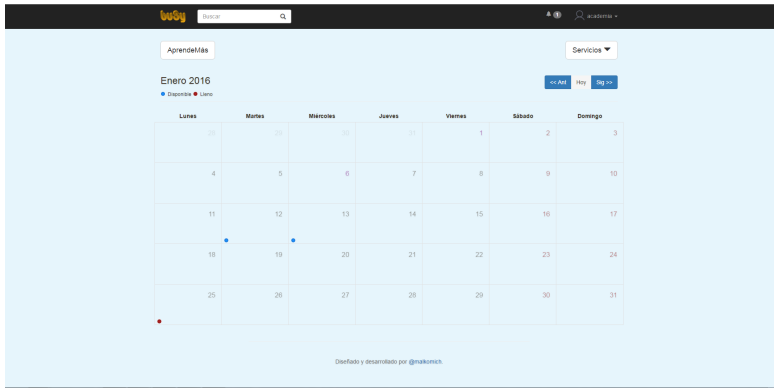


Figura B.5: Página principal de gestor de empresa

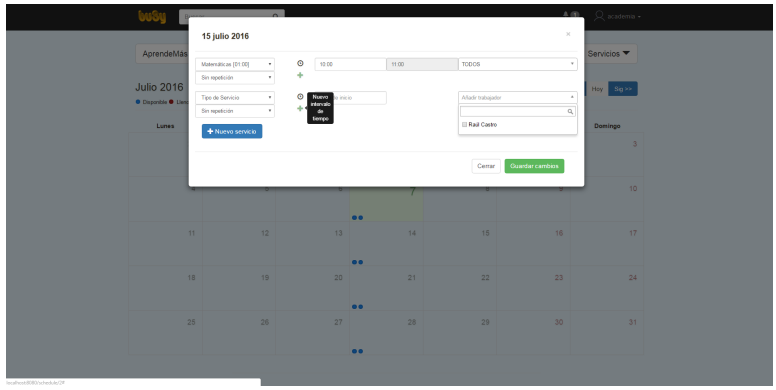


Figura B.6: Creación de servicios

clic en la celda que contiene dicho día.

Podremos ver como se despliega un formulario en una ventana flotante, similar a la que podemos observar en la Figura B.6. Debemos seleccionar siempre el tipo de servicio que queramos y la hora de inicio, ya que la hora final se completará automáticamente en función del tipo de servicio elegido. También debemos seleccionar al menos un trabajador que vaya a realizar ese servicio, y si queremos que se repita diaria o semanalmente. Dentro de cada servicio podemos añadir franjas horarias, las cuales también se repetirán de forma conjunta si así se selecciona en el formulario.

Dentro de la vista de calendario, podremos navegar entre todos los meses que queramos libremente, disponiendo siempre de un botón para regresar rápidamente al mes actual.

Desde el rol de un usuario que busca algún servicio que poder reserva, lo primero que tendremos que hacer es buscar la empresa cuyos servicios queremos contratar. Para

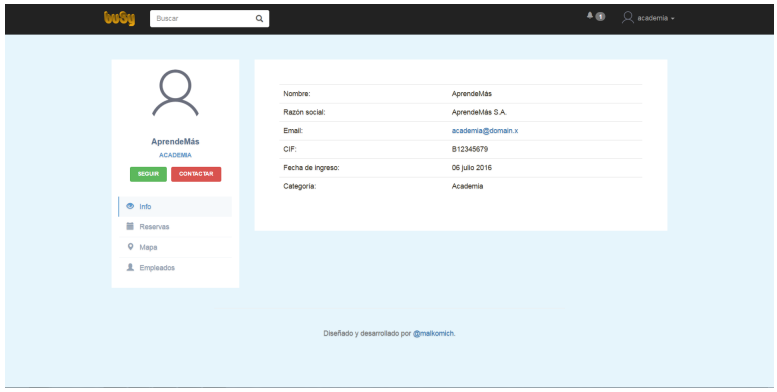


Figura B.7: Página de empresa

ello escribiremos parte del nombre de la empresa en la barra de búsqueda del menú de navegación superior.

Una vez encontremos la empresa deseada entre la lista de sugerencias dispuesta por el servicio de auto-completado de búsqueda, seleccionaremos su nombre para poder acceder a la página de dicha empresa.

Lo primero que encontraremos será una ventana con información detallada de la empresa, como vemos en la Figura B.7. Dentro de esta página podremos cambiar de sección en el menú lateral. Para llevar a cabo una reserva de servicio, tendremos que acceder a la sección de reservas. Dentro de la sección, se nos presentará una lista de oficinas que posee la empresa. En cada bloque mostrará una imagen, que será un icono por defecto en caso de no tener una imagen de la oficina, y su información de contacto y localización.

Al seleccionar la oficina en la que queramos reservar un servicio, se nos mostrará un calendario, similar al de la página del gestor de empresa. En dicho calendario podremos visualizar todos los servicios que han sido creados para esa oficina. Para llevar a cabo de uno de los servicios, el icono debe mostrar el color que representa la disponibilidad del servicio. Dicho color vendrá mostrado en las etiquetas a modo de leyenda situadas sobre el bloque del calendario.

Al seleccionar alguno de los servicios disponibles, se nos mostrará una sencilla selección del trabajador preferido para realizar la reserva. Como se ha comentado anteriormente, sólo serán accesibles los servicios disponibles, y además sólo aparecerán los trabajadores asignados y disponibles para ese servicio. Sin embargo, cada reserva representa una transacción, y por tanto será siempre validada previamente.

Por último, queda comentar cómo utilizar la sección de administración. Para acceder a esta sección debemos introducir nuestros credenciales de usuario con privilegios adminis-

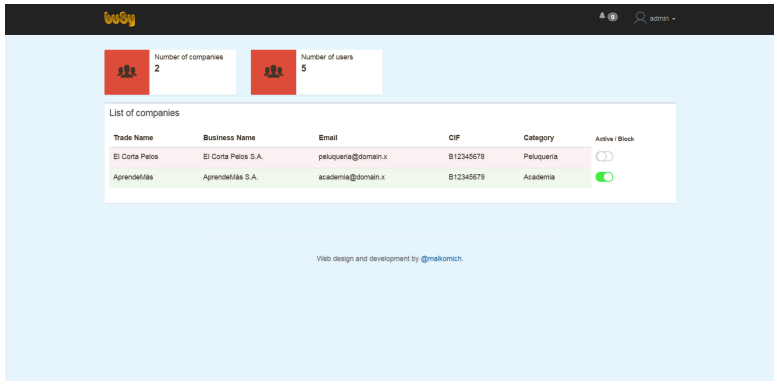


Figura B.8: Página de administración

trativos en la ventana de identificación. Una vez dentro, veremos una página similar a la mostrada en la Figura B.8.

Podemos ver en la sección superior un bloque en el que nos mostrará el número de empresas actualmente registradas en el sistema. Si seleccionamos dicho bloque, se nos mostrará una lista con todas las empresas, con un ligero fondo que nos indicará el estado de activación de cada empresa. Verde indicará que la empresa se encuentra activada, mientras que rojo nos indicará que se encuentra en un bloqueo o que no ha sido aún verificada. Si queremos cambiar el estado de alguna de las empresas, simplemente con un click sobre el interruptor situada a la derecha en cada fila realizaremos dicha opción.

APÉNDICE C

Contenido del CD

Se detalla a continuación el contenido del disco que acompaña este documento:

- **Código fuente** Carpeta que contiene todo el código fuente de la aplicación desarrollada en este proyecto
- **Memoria** Carpeta que contiene la presente memoria en formato PDF