Facultad de Ciencias

Desarrollo de aplicación de control horario
(Development of time control application)

Trabajo de Fin de Grado
para acceder al título de

GRADUADO EN INGENIERÍA INFORMÁTICA

Autor: Pablo Pereda Barberarena

Director: Enrique Vallejo Gutiérrez

Noviembre – 2021

A mi familia, mis amigos y mi novia, gracias por el apoyo que siempre me han dado, por estar ahí siempre que lo he necesitado y por animarme a seguir adelante.

A mis compañeros de trabajo que siempre me han ayudado, me han tratado increíblemente bien desde el principio y que me han enseñado, ayudado y confiado en mi para realizar proyectos como el de este TFG.

Por último, me gustaría dar las gracias a todos mis compañeros y profesores de la Universidad de Cantabria por todos estos años de formación que acaban con este TFG. En especial a mi tutor Enrique Vallejo Gutiérrez, por acompañarme y guiarme en este camino, gracias por implicarte tanto y por la rapidez con la que me has atendido siempre que lo he necesitado.

Pablo Pereda Barberarena

Tabla de contenido

Resumen 5
Abstract 6
1. Introducción 7
1.1 Motivación y objetivos 7
1.2 Alcance 7
1.3 Metodología y organización del proyecto 7
1.4 Estructura del documento 8
2. Tecnologías y herramientas utilizadas 10
2.1 Java 10
2.2 Spring 10
2.3 Angular 10
2.4 Jasmine 11
2.5 JQuery 11
2.6 HTML 11
2.7 Oracle SQL Developer 11
2.8 Maven 12
2.9 SonarQube 12
2.10 Jenkins 12
2.11 Git 13
2.12 MyBatis 13
2.13 JBoss 13
2.14 Documentación i18n 13
3. Especificación de requisitos 14
3.1 Captura requisitos 14
3.2 Explicación de la aplicación 14
3.3 Requisitos funcionales 17
3.4 Requisitos no funcionales 17
4. Diseño 19
4.1 Diseño arquitectónico 19
4.2 Diseño detallado 20
5. Implementación 23
5.1 Implementación requisitos 23
5.2 Conexión con la base de datos 23
5.3 Utilización de la base de datos 25
5.4 Servicios de la aplicación 26

5.5 Controladores de la aplicación 26
5.6 Clases adicionales 28
6. Pruebas y despliegue 29
6.1 Pruebas unitarias 29
6.2 Pruebas de Integración 29
6.3 Pruebas de aceptación 30
6.4 Pruebas de seguridad 30
6.5 Despliegue 31
7. Tareas realizadas 33
7.1 Desarrollo de las pantallas 33
7.2 Desarrollo de consultas de base de datos 34
7.3 Creación de validaciones 34
7.4 Realización de pruebas unitarias y de integración 35
7.5 Traducciones mediante documentación i18n 35
7.6 Corrección de vulnerabilidades 35
8. Conclusión 36
Bibliografía 37

Índice de Figuras

Figura 1: Diagrama de Despliegue 20
Figura 2: Diagrama Relacional Base de Datos 21
Figura 3: Diagrama de Flujo 22
Figura 4: Archivo Configuración Servidor "StandAlone.xml" 23
Figura 5: Archivo Configuración "JBoss-Web.xml" 24
Figura 6: Archivo Configuración "Web.xml" 24
Figura 7: “Mapper.xml” 25
Figura 8: Condición “Mapper.xml” 26
Figura 9: Ejemplo Clase *Service* 26
Figura 10: Creación Clase Controlador 27
Figura 11: Método Interno Controlador 27
Figura 12: Ejemplo Llamada Controlador 27

Índice de Tablas

Tabla 1: Requisitos Funcionales 17
Tabla 2: Requisitos No Funcionales 18

Resumen

A lo largo de este TFG se describe el desarrollo y funcionamiento de una aplicación centralizada creada para una empresa internacional propia del ámbito de las energías renovables. La empresa que desarrolla la aplicación usará una arquitectura en tres capas mediante Angular, Java y Oracle junto a MyBatis y una metodología ágil mediante *Agile.*

Esta aplicación ha sido desarrollada con el objetivo de controlar la asistencia de personal, tanto trabajadores como administradores, a parques de energías renovables a nivel global y de sus actividades en ellos.

La aplicación consta de las siguientes funcionalidades:

- Gestión de la presencia de personal en las plantas de trabajo con las opciones de registrar y finalizar dicha presencia.

- Gestión de las tareas del personal con las opciones para crear y finalizar trabajos.

- Un sistema de *login* para los usuarios que utilicen la aplicación.

- Opciones de control de idioma de la aplicación.

Esta aplicación actualmente está en producción, permitiendo así la gestión y control de todos los accesos y trabajos de los asistentes a estos parques renovables de una manera más sencilla.

Palabras clave: Gestión de Personal, Java, Desarrollo Web, Angular.

Abstract

This TFG describes the development and operation of a centralized application created for an international company in the field of renewable energy. The company that develops the application will use a three-tier architecture using Angular, Java and Oracle together with MyBatis and an agile methodology using Agile.

This application has been developed with the objective of controlling the attendance of personnel, both workers and administrators, to renewable energy parks at a global level and their activities in them.

The application consists of the following functionalities:

- Management of the presence of personnel in the work plants with the options of registering and terminating such presence.

- Management of personnel tasks with the options to create and terminate jobs.

- A login system for users using the application.

- Language control options for the application.

This application is currently in production, allowing the management and control of all the accesses and jobs of the attendants to these renewable energy parks in a simpler way.

Keywords: Personnel Management, Java, Web Development, Angular.

1. Introducción

El proyecto de este TFG ha sido desarrollado en una empresa consultora multinacional que ofrece soluciones de negocio, estrategia, desarrollo y mantenimiento de aplicaciones tecnológicas y outsourcing. Específicamente, se ha desarrollado en el área de energías renovables.

A lo largo de este apartado se explica detalladamente tanto el objetivo del proyecto como el alcance y metodologías utilizados para su desarrollo.

1.1 Motivación y objetivos

Esta aplicación fue encargada por una empresa internacional dedicada al ámbito de las energías renovables. La necesidad de este cliente era la de gestionar de manera eficiente y sencilla los controles de los accesos y trabajos que tiene que realizar cada trabajador en sus parques.

Antes del desarrollo de esta aplicación no existía ningún mecanismo para controlar los accesos o las tareas realizadas en los parques.
Con esta aplicación se les permite controlar esto de una manera muy rápida y sencilla, ayudando así tanto a las personas que acceden al parque como a las que gestionan los accesos.

1.2 Alcance

El área afectada por esta aplicación es el de las energías renovables.

La aplicación será accesible para todos los trabajadores de los parques de esta empresa. Estos parques no solo están ubicados en España, sino que se pueden encontrar en distintos lugares del mundo como, por ejemplo, Europa, Oceanía, Norteamérica o Sudamérica.

1.3 Metodología y organización del proyecto

El trabajo se ha desarrollado en el departamento de energías renovables, junto con un programador y un *team leader*.

El *team leader* es el encargado de gestionar tanto el control de tiempo como de la supervisión de las tareas de desarrollo de la aplicación.

Por otro lado, tanto el programador como yo nos encargamos del desarrollo de la aplicación.

El proyecto se realiza usando metodologías ágiles mediante *Scrum* [1], metodología basada en el uso de *Sprints*.

Estos *Sprints* constan de un número de semanas en las cuales se planifica cierto número de tareas que al final de cada *Sprint* deberían estar acabadas.

Al comienzo de cada *Sprint* se realiza una reunión en la cual se discute y debate el número de tareas que se espera completar y el tiempo estimado que debe tardar en hacerse cada una. En vez de estimar mediante horas, se ha elegido hacerlo mediante puntos de historia, siendo cada uno de estos puntos de historia 8 horas.

El método de los puntos de historia sirve para proporcionar una visión del esfuerzo necesario para cada tarea, otorgando la posibilidad al cliente de decidir la prioridad de realización de dichas tareas según su dificultad.

Los posibles valores otorgados a las tareas siguen la serie de Fibonacci. Esto ayuda a diferenciar las tareas según su dificultad. Si se estimara con una secuencia del 1 al 10 sería difícil diferenciar la dificultad de una tarea de complejidad 6 de una de 7. Usando Fibonacci es más fácil distinguir esto. Por ejemplo, si tenemos tarea de complejidad 8 el siguiente valor posible sería 13 distinguiendo que la dificultad es mayor pero no llegando a ser el doble. Esta separación entre los números es lo que te permite ver la diferencia entre tareas.

Una vez estimadas todas las tareas y comenzado el *Sprint* se realizan reuniones diarias llamadas *dailys* en las cuales cada miembro del equipo, incluido el *team leader*, cuenta su avance en las tareas y posibles complicaciones de manera que se pueda replanificar el tiempo de dichas tareas.

Aparte de estas reuniones, el *team leader* es el encargado de supervisar el trabajo. Si alguno de los plazos marcados no se va a poder cumplir, los miembros del equipo comentan al *team leader* por qué va a ocurrir esto y él valora si la justificación dada por los miembros del equipo es o no suficiente para el atraso. Siendo una justificación correcta, se volverán a estimar las tareas afectadas, incluso podrán ser atrasadas hasta el siguiente *Sprint* si el tiempo restante no permite acabarla en el actual.

1.4 Estructura del documento

El documento está estructurado en distintos capítulos:

- Capítulo 2

Este capítulo trata sobre las tecnologías y herramientas que se utilizan en el proyecto. Se define y explica el uso cada una.

- Capítulo 3

Este capítulo muestra como se capturan los requisitos tanto funcionales como no funcionales de la aplicación. Previamente, se explica la funcionalidad y

estructura de la aplicación de manera que se puedan entender los requisitos pedidos.

- Capítulo 4

Este capítulo explica el diseño de la aplicación. Para ello, se explican las distintas capas de esta y se muestran unos diagramas que ayudan a entender su diseño.

- Capítulo 5

Tras explicar el diseño y los requisitos, este capítulo muestra como se implementan dichos requisitos y como se realiza la conexión de la aplicación para cumplir con el diseño anteriormente detallado.

- Capítulo 6

Este capítulo trata sobre las pruebas realizadas en la aplicación para comprobar su correcto funcionamiento y sobre los despliegues de la aplicación en los distintos entornos.

- Capítulo 7

En este capítulo se detallan las tareas realizadas por mi a lo largo del desarrollo de la aplicación.

- Capítulo 8

Este último capítulo es una conclusión sobre el desarrollo de la aplicación y cómo ha sido de ayuda para mejorar conocimientos y habilidades personales.

2. Tecnologías y herramientas utilizadas

A lo largo de este apartado se introducen sobre las distintas tecnologías y herramientas utilizadas en el proyecto.

Para elegir el uso de estas tecnologías se hicieron reuniones previas a comenzar el proyecto, en las cuales se marcaron cuáles podían ser las mejores para la aplicación a desarrollar.

2.1 Java

Java [14] es un lenguaje de programación muy usado en el ámbito de la informática. Se usa tanto por profesionales como por estudiantes, siendo uno de los lenguajes destacados a la hora de aprender a programar.

Java nos permite el desarrollo de todo tipo de *software* y, en especial, desarrollo de páginas y sitios webs. En este caso se utilizó la versión Java 8 (versión 1.8 según Oracle). Esta versión respecto a las anteriores permite la integración JavaScript [8] con Java.

La mayoría de la capa *back-end* del proyecto ha sido desarrollada en Java.

2.2 Spring

Spring [15] es un *framework* usado para el desarrollo web en Java. Este *framework* tiene la finalidad de estandarizar el trabajo, resolver y agilizar problemas y complejidades que vayan surgiendo. Permite desarrollar aplicaciones de una manera más rápida y eficaz, ahorrando líneas de código para tareas que se repitan.

Spring utiliza los llamados *beans.* Un *bean* se trata de un objeto instanciado, ensamblado y administrado por el contenedor de Spring capaz de realizar la inyección de dependencias sin usar definiciones de Java.

En la configuración del proyecto existe un archivo XML que se encarga de la gestión de Spring mediante estos *beans*.

2.3 Angular

Angular [12] es un *framework* de desarrollo de aplicaciones web que separa completamente la parte *front-end* de la parte *back-end* de la aplicación. Evita código repetido y mantiene todo ordenado con su patrón MVC (Modelo-Vista-Controlador). El lenguaje principal de angular es TypeScript.

2.4 Jasmine

Jasmine [13] es una *suite* de *testing* con metodología *Behavior Driven Development* utilizado para testear código de angular. Tiene una sintaxis muy clara con características que permiten una escritura sencilla, de manera que cada prueba creada es una función.

Para la ejecución de estos tests se utiliza Karma. Karma [13] es un *test-runner*, un módulo que permite automatizar tareas de *suites* de *testing* como Jasmine. Karma permite ver el resultado de la ejecución de los *tests* y así comprobar si funcionan correctamente.

2.5 JQuery

JQuery [16] es una librería desarrollada en 2006 por John Resig que nos permite agregar una capa de interacción AJAX entre la web y las aplicaciones que desarrollamos, controlar eventos y crear animaciones y efectos diversos, enriqueciendo así la experiencia del usuario.

2.6 HTML

HTML [10, 10.1] es el lenguaje en el que se define el contenido de las páginas web. Básicamente es un conjunto de etiquetas que se utilizan para identificar el texto y otros elementos que componen una página web, como imágenes, menús, videos y más.

2.7 Oracle SQL Developer

Oracle SQL Developer [17] es una herramienta que proporciona una interfaz gráfica de usuario que permite a los administradores y usuarios de bases de datos realizar sus tareas con mayor velocidad ahorrando tiempo. Tiene potentes editores que permiten trabajan con SQL, PL/SQL y con procedimientos almacenados de Java y XML.

SQL [11] es un lenguaje de consulta estructurado que ayuda a gestionar la manipulación, definición e integridad de los datos almacenados en bases de datos.

PL/SQL es un lenguaje por procedimientos que amplía la capacidad de SQL. Mientras que con SQL solamente se realiza una consulta al mismo tiempo, PL/SQL ejecuta bloques de códigos completos.

2.8 Maven

Maven [18] es un *software* usado en la gestión de proyectos capaz de administrar la construcción, los informes y la documentación del proyecto.

Maven simplifica y estandariza el proceso de construcción del proyecto, aumenta la reutilización de tareas relacionadas con esta construcción reduciendo notablemente la cantidad de pasos del proceso.

2.9 SonarQube

SonarQube [5] es una herramienta que permite medir la calidad del código. Dentro del informe que genera SonarQube sobre calidad encontramos:

- Código duplicado: se muestra un porcentaje y número de líneas duplicadas para arreglar mediante la refactorización del código.

- Código muerto: es código que nunca es utilizado.

- Estándares de codificación: convenciones para el código, nombres de variables, espaciado, indentaciones…

- Bugs: error debido al cual el programa funciona incorrectamente.

- Complejidad ciclomática: métrica de *software* que proporciona una medición cuantitativa de la complejidad lógica de un programa. Se basa en el cálculo de número de caminos independientes del código.

- Comentarios: tener una gran cantidad de comentarios en el código.

- Cobertura y tests: muestra el porcentaje de código testeado y el porcentaje sin testear.

Además, SonarQube divide los problemas en issues, los cuales tienen rangos según su gravedad:

- Bug: fallo en el código.

- *Vulnerability*: problema de seguridad.

- *Code Smells*: problemas de mantenibilidad de código.

2.10 Jenkins

Jenkins [6] se basa en la integración continua. Esto es una práctica en la cual todos los miembros del equipo integran su trabajo de manera frecuente. Cada una de estas integraciones pasa por una serie de procesos como es la verificación de que el código compila (*build*), comprobación de si pasa las

métricas de calidad proporcionadas por la empresa en el sonar (*quality gate*), …

Todos estos procesos van en orden, de manera que si uno de ellos falla nos muestra un error y nos dice qué ha fallado.

2.11 Git

Git [7] es una herramienta que sirve para controlar las versiones del código, pero que no depende de un repositorio central y siendo más potente que otras herramientas como SVN.

Algunas de las características más importantes de Git son:

- Rapidez en la gestión de ramas: Git avisa que un cambio será fusionado con más frecuencia de lo que se escribe originalmente.

- Gestión distribuida: Los cambios se importan como ramas adicionales y pueden ser fusionados de la misma manera que se hace en la rama local.

- Gestión eficiente de proyectos grandes.

2.12 MyBatis

MyBatis [19] es un marco de código abierto que implementa la especificación JPA para conectarse a una base de datos, agregar, eliminar, modificar y verificar operaciones en ella.

JPA significa Java Persistence API. Su propósito es el de ayudar a mejorar la eficiencia del desarrollo.

2.13 JBoss

JBoss [3] es un servidor de aplicaciones que se utiliza para implementar aplicaciones Java. Mediante este servidor se carga la aplicación.

2.14 Documentación i18n

Para gestionar una aplicación que use muchos idiomas distintos se necesita algo que te ayude a realizar las traducciones de una manera óptima respecto a rendimiento, duplicación de código...

Para ello se usa la documentación i18n [2], que sirve para proporcionar una internacionalización de los textos mediante clases en las que configurar el texto en los distintos idiomas de la aplicación.

3. Especificación de requisitos

Este apartado trata de la captura de requisitos, tanto funcionales como no funcionales. Para ello se explica previamente tanto el funcionamiento como la estructura de la aplicación.

3.1 Captura requisitos

La captura de requisitos se realizó mediante reuniones por parte del *team leader* con el cliente en las que se expuso el contexto de la aplicación, funcionamiento, uso…

Estas reuniones fueron todas mediante la herramienta de comunicación Microsoft Teams, ya que debido a la pandemia no se pudo hacer presencialmente.

3.2 Explicación de la aplicación

Para acceder a la aplicación se utilizará una URL propia de la *intranet* de la empresa del cliente. Los usuarios deberán registrarse con un usuario y contraseña dados por la empresa, de manera que solo puedan acceder sus usuarios a la aplicación.

Una vez el usuario accede a la URL aparecerá una página de *login* en la cual deberán introducir el usuario y contraseña para iniciar sesión en la aplicación. Si fallara la introducción de estos campos se notificará, teniendo un límite de oportunidades para ingresar los datos correctos. Si no se accediera correctamente en ese límite de intentos, se bloqueará el acceso a ese usuario durante un tiempo.

Esta pantalla también tendrá opciones de recuperación de datos por si se ha olvidado el usuario o la contraseña.

Una vez ha realizado el proceso de *login*, accederá a la página principal de la aplicación.

En ella se ven:

- **Zona superior**

Aparecerá el logo en la esquina izquierda de la empresa y en la esquina derecha una opción para cambiar el idioma y otra con el nombre de usuario y las opciones de cerrar sesión y de cambiar la contraseña.

- **Zona izquierda**

En esta parte habrá un menú desplegable con todas las funcionalidades de la página. Estas funcionalidades se dividen en dos grupos:

- Gestión de los trabajadores.

Dentro de este grupo hay dos opciones:

- Crear un registro de entrada de un trabajador.
- Finalizar un registro de entrada de un trabajador.

- Gestión de las tareas de los trabajadores.

Dentro de este grupo hay dos opciones:

- Crear un trabajo.
- Finalizar un trabajo.

En esta página principal se accederá a la opción que se desee y se cargará la página propia de la misma.

A continuación, se explica el funcionamiento de cada opción.

- **Crear un registro de asistencia de un trabajador**

Esta pantalla consta de una serie de campos que se deben rellenar para la entrada de un trabajador.

Por defecto, la persona que va a entrar será la asociada al usuario con el que se ha entrado a la aplicación. Al crear un registro de asistencia, esta persona será añadida automáticamente al registro sin posibilidad de ser quitada. Aparte de esto, deberá introducirse la planta a la que se va a acceder. La configuración de las plantas existentes ya está disponible en la base de datos, habiendo sido creada e introducida a partir de otra aplicación de la empresa, y no se modifica en ningún momento

En caso de acceder más personas aparte del usuario que se ha registrado en la aplicación se deberán introducir mediante su usuario o mediante la lectura de un código de barras.

Dependiendo de la planta a la que se acceda puede ser que existan una serie de restricciones, como por ejemplo restricciones de acceso a ciertos espacios de la planta. Estas restricciones deberán ser marcadas como leídas antes de completar el registro de entrada.

Una vez estén todos los datos introducidos, al final de la página habrá dos opciones. La primera opción será la de registrar la entrada y la segunda la de cancelar la operación.

- **Finalizar un registro de asistencia de un trabajador**

Si no hay ningún registro de un trabajador se notifica y si hay, indica las características del registro y da la opción de finalizarlo o de cancelar la operación.

- **Crear un trabajo**

Si no hay ningún registro de trabajador se notifica.

Si hay, aparecerá una serie de formularios en los que introducir datos que se utilizarán para la creación del trabajo.

Lo primero será introducir una orden de trabajo, la cual si es correcta cargará una serie de datos sobre esa orden.

Por otra parte, está la opción de marcar si el trabajo es para un solo trabajador o de añadir más trabajadores mediante su código de usuario o mediante código de barras.

Debajo estará tanto la documentación del trabajo como las restricciones de este. En caso de que existan restricciones la aplicación las cargará en la página y permitirá una opción en la que el trabajador pueda aceptar dichas restricciones.

Por último, una opción de confirmar y crear trabajo y otra de cancelar. Si se crea el trabajo y te faltan datos de introducir, la aplicación avisará mediante una ventana que indica que datos faltan.

- **Finalizar un trabajo**

Si no hay ningún trabajo se notifica y si hay, se muestran los datos del trabajo y da la opción de finalizarlo o de cancelar la operación.

- **Cambiar idioma**

Al pulsar esta opción se desplegarán los idiomas disponibles, una vez pulsado uno, la aplicación se cargará cambiando el idioma al seleccionado.

- **Cambiar contraseña**

Al pulsar esta opción se cargará una página en la que introducir la contraseña antigua, la nueva contraseña y una confirmación de esta nueva contraseña. De esta manera se comprueba que el valor introducido en la nueva y la confirmación son los mismos.

Una vez hecho esto, el botón de cambiar contraseña se activará y habrá que pulsarlo para confirmar la operación.

- **Cerrar sesión**

Al pulsar en esta opción se saldrá de la sesión del usuario y se redirigirá a la pantalla de *login*.

3.3 Requisitos funcionales

Los requisitos funcionales [20] de una aplicación son los servicios que prestará y la forma en la que reaccionará en determinadas situaciones.

La siguiente tabla indica los requisitos funcionales de la aplicación:

Identificador	Descripción corta	Descripción del requisito
FUN001	Gestión de usuarios	Todos los datos de los usuarios y su acceso a la aplicación deben realizarse según los protocolos propios de la intranet corporativa.
FUN002	Creación de trabajos	El usuario dispondrá de una página con un resumen de los datos introducidos para la creación de los trabajos.
FUN003	Finalización de trabajos	El usuario podrá ver un resumen de los datos del trabajo a finalizar en el que tendrá entre otros datos el lugar del trabajo, la fecha y los usuarios que lo realizan.
FUN004	Creación de asistencia	El usuario podrá crear una asistencia a una planta de trabajo en la que podrá añadir más usuarios aparte del suyo propio y leer las posibles restricciones de la planta.
FUN005	Finalización de asistencia	El usuario verá un resumen de la asistencia activa en el que aparecerán los datos de los usuarios pertenecientes a la asistencia y la fecha y lugar de la planta.
FUN006	Control de idioma	El usuario dispondrá de una opción en la aplicación para gestionar el idioma de esta.
FUN007	Control de usuario	El usuario dispondrá de una opción en la aplicación para tanto cambiar su contraseña actual por otra como para cerrar su sesión.

Tabla 1: Requisitos Funcionales

3.4 Requisitos no funcionales

Los requisitos no funcionales [20] de una aplicación son aquellos que no se refieren a las funciones del sistema. Estos requisitos se refieren a las propiedades como el rendimiento o seguridad de la aplicación.

La siguiente tabla indica algunos de los requisitos no funcionales de la aplicación:

Identificador	Descripción corta	Descripción del requisito
NFUN001	Seguridad	La aplicación no deberá presentar vulnerabilidades que afecten a la seguridad de los datos de esta. Se utilizarán herramientas de análisis de vulnerabilidades para evitar esto.
NFUN002	Visualización	La aplicación deberá diseñarse de manera que sea correctamente visible en navegadores como Internet Explorer, Google Chrome y Mozilla Firefox.
NFUN003	Expiración de Sesión	La aplicación detectará si un usuario deja de usarla por un periodo de tiempo determinado. Si es así, se cerrará la sesión del usuario de manera que cuando vuelva a utilizarla deba acceder otra vez con su usuario y contraseña.
NFUN004	Calidad de Código	La aplicación deberá pasar por herramientas de análisis de la calidad de código como SonarQube. En estas mediciones, la calidad de código deberá superar unos valores mínimos que serán marcados por la empresa.

Tabla 2: Requisitos No Funcionales

4. Diseño

Tras obtener los requisitos tanto funcionales como no funcionales, se realizará el diseño del sistema. Este diseño tiene por una parte el diseño arquitectónico y por otra el diseño detallado.

La aplicación no tiene roles distintos para los usuarios, todos tienen acceso a todas las funcionalidades ya que tanto trabajadores como administradores de los parques tienen que usar dichas funcionalidades para las actividades en el parque.

4.1 Diseño arquitectónico

El diseño arquitectónico [21] nos permite definir la organización del sistema y cómo debe organizarse la estructura global. Identifica los principales componentes estructurales y la relación entre ellos.

En este caso, el sistema que se desarrolla es una aplicación web con acceso mediante *intranet* corporativa.

Respecto a la capa *front-end* de la aplicación, esta es desarrollada en Angular y, por otra parte, la capa *back-end* desarrollada en java y conectada a una base de datos que tiene todos los datos de esta aplicación y de otras de la misma empresa.

Este sistema utilizará servicios REST. REST es una API de transferencia para conectar diferentes sistemas basados en HTTP. Esta API sirve para a obtener, crear, ejecutar datos y devolver estos datos en diferentes formatos como XML y JSON.

El formato más utilizado hoy en día es JSON, ya que es más ligero y fácil de leer que XML. REST utiliza las primitivas *GET, POST, PUT* y *DELETE*.

En este caso, el desarrollo de las capas *front-end* y *back-end* se realiza en el entorno de Eclipse, el cual posee la capacidad de arrancar la aplicación mediante la configuración servidores JBoss.

Respecto a las capas de la aplicación:

- **Capa *Front-End***

Es la parte visible de la aplicación, consta de todas las opciones de menú, todas las tablas y elementos visuales. Está desarrollada con Angular.

- **Capa de Lógica de Negocio**

Es la parte interna de la aplicación, lee, graba y trata la información con la base de datos. Esta desarrollada con Java y Spring principalmente.

- **Capa de datos**

Se trata de la base de datos donde están todos los datos de la aplicación.

Estas dos últimas capas forman la capa *Back-End*.

4.2 Diseño detallado

En este apartado se muestra un diagrama de despliegue y el modelo de datos del sistema.

El diagrama de despliegue es un tipo de diagrama UML que muestra la arquitectura de la ejecución de la aplicación. Son utilizados para visualizar el *hardware* y *software* de la aplicación.

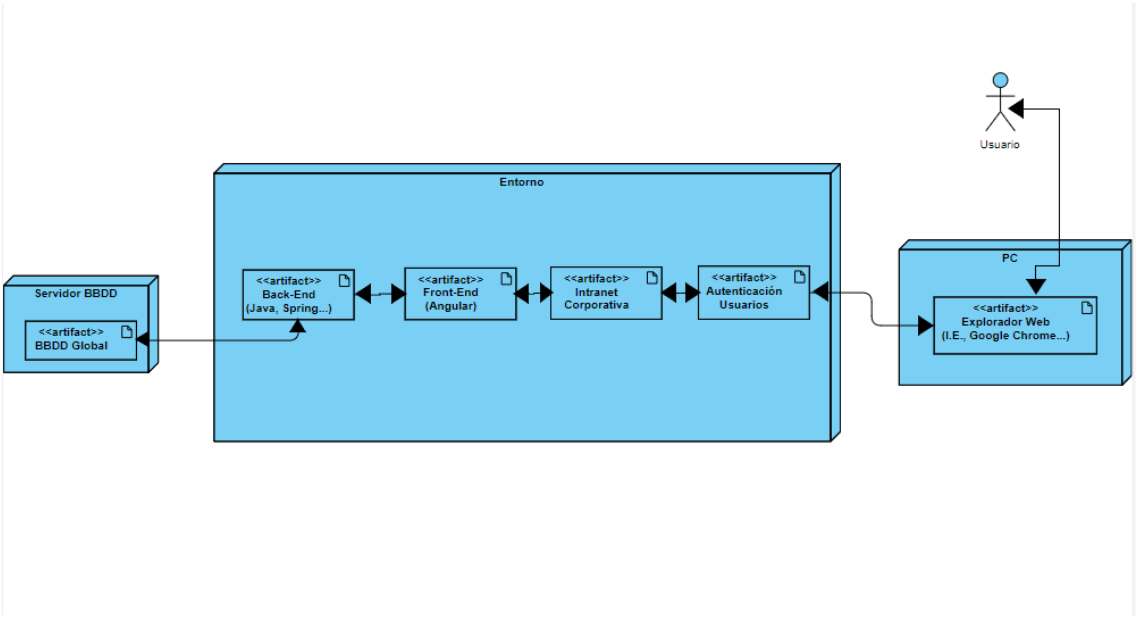

Figura 1: Diagrama de Despliegue

Observando este diagrama se observan tres nodos principales:

- **Primer nodo:** Correspondiente al servidor donde se encuentra la base de datos. Esta base de datos es global para tanto esta como otras aplicaciones.

- **Segundo nodo:** Correspondiente a las capas de la aplicación *back-end* y *front-end* y a la *intranet* corporativa y el protocolo de autenticación para que los usuarios accedan a esta.

- **Tercer nodo:** Ordenador del usuario el cual usando el navegador que elija el usuario accederá a la aplicación.

Respecto a la base de datos utilizada, ya se ha comentado que es utilizada por otras aplicaciones propias de la misma empresa. Esto provoca que dicha base

de datos contenga mucha más información que la necesitada para esta aplicación. Sin embargo, habrá otras tablas que compartirá con otras aplicaciones, como por ejemplo la tabla que contenga los usuarios de la empresa o los países, ciudades… de sus plantas.

A continuación, se muestra una imagen de diagrama entidad-relación [27] de la base de datos. Estos diagramas ilustran cómo las distintas tablas se relacionan entre sí dentro de la base de datos.

Las imágenes están deliberadamente difuminadas, ya que el proyecto tiene una política de privacidad que no permite mostrar datos de la aplicación.

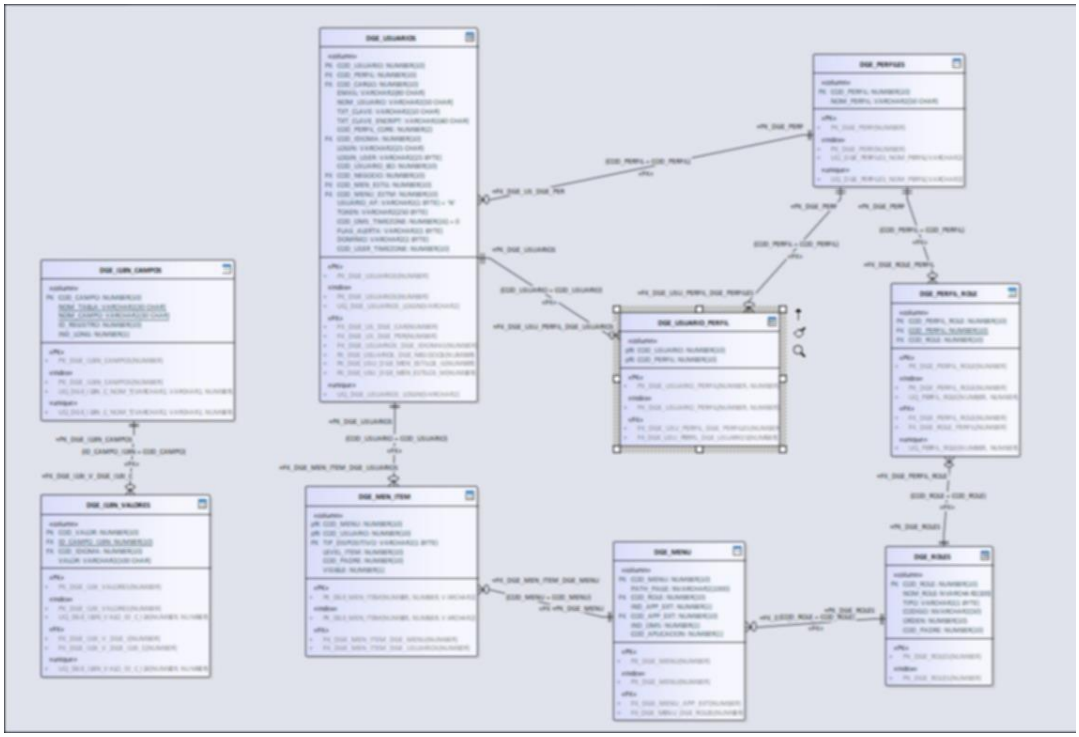

Figura 2: Diagrama Relacional de la Base de Datos

Al ser una base de datos global y que usa otras aplicaciones el número de tablas es mucho más grande de lo que sería si fuera exclusiva para esta aplicación. Sin embargo, en esta captura se han intentado mostrar solo unas pocas para poder ver un ejemplo de la estructura y las uniones entre tablas.

Además de esto, se realizaron muchos otros tipos de diagramas para el diseño la aplicación. Por ejemplo, los diagramas de flujos.

Los diagramas de flujo [28] son un tipo de diagramas cuyo objetivo es el de describir un proceso, sistema o algoritmo matemático. La siguiente figura muestra un ejemplo censurado del esquema de estos tipos de diagramas.

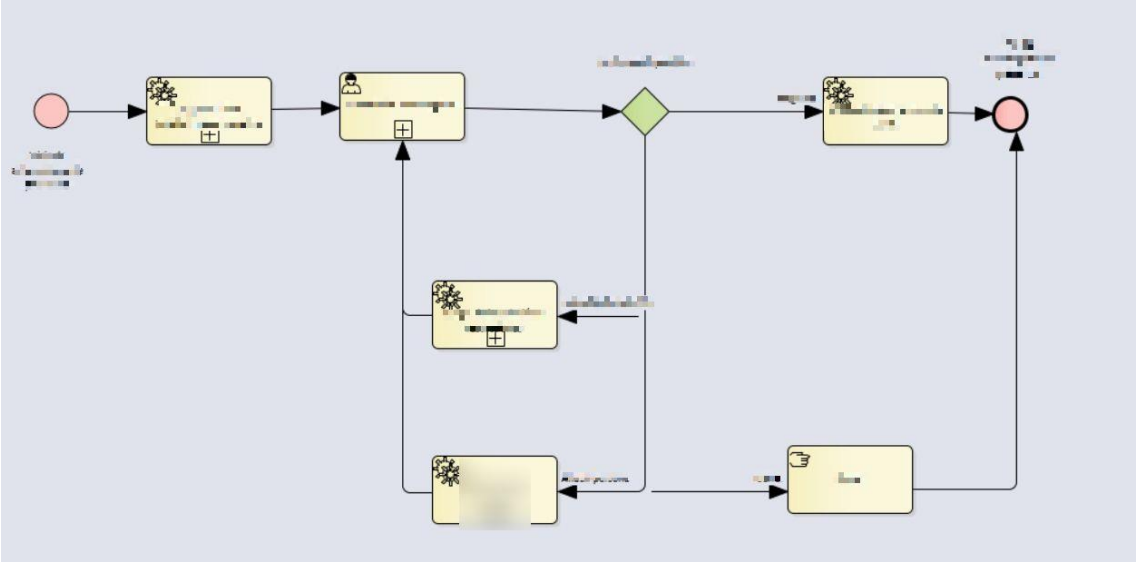

Figura 3: Diagrama de Flujo

5. Implementación

En este apartado se va a detallar cómo se realizó la implementación del *software* de la aplicación.

5.1 Implementación requisitos

Para la implementación de los requisitos detallados en el apartado 3.3 se ha dividido el código desarrollado en distintos paquetes:

- Usuario: en este paquete está el código relacionado con los requisitos de gestión y de control de usuarios.

- Asistencia: en este paquete está el código relacionado con la creación y finalización de asistencias.

- Trabajos: en este paquete está el código relacionado con la creación y finalización de los trabajos.

En estos paquetes están las validaciones, funciones, traducciones y controladores que conectan la parte *front-end* de la *back-end,* conexiones con la base de datos, …

5.2 Conexión con la base de datos

Esta conexión se realiza mediante la configuración de archivos XML y el servidor JBoss con el que se arranca la aplicación.

Por una parte, en la configuración del servidor JBoss se pone la dirección de la base de datos y las credenciales de usuario para acceder a ella.

Y, por otra parte, en los archivos de configuración del proyecto se tiene que poner la relación entre la configuración del JBoss y el proyecto.

A continuación, se muestra una serie de archivos que hay que configurar tanto en el servidor como en el proyecto. Por motivos de confidencialidad se tapan los nombres de los datos.

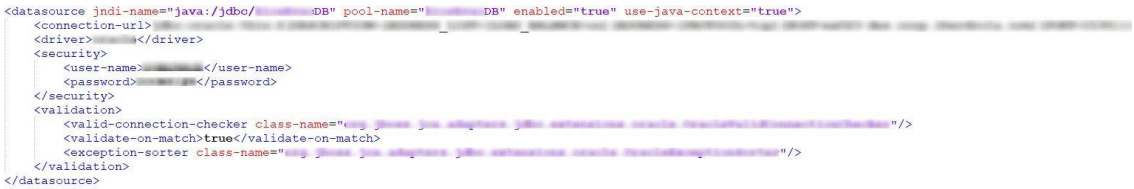

Figura 4: Archivo Configuración Servidor "StandAlone.xml"

Esta captura muestra cómo se configura en el servidor el acceso a la base de datos. Para esta configuración se añaden parámetros como *connection-url*

donde está la *url* de acceso a la misma o "*security*" donde está el nombre y contraseña del usuario.

Tras configurar este archivo del servidor JBoss, para enlazarlo con la aplicación se configuran otra serie de archivos. Entre ellos tenemos estos dos:

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-web>
    <resource-ref>
        <res-ref-name>jdbc/[redacted]DB</res-ref-name>
        <jndi-name>java:/jdbc/[redacted]DB</jndi-name>
    </resource-ref>
```

Figura 5: Archivo Configuración "JBoss-Web.xml"

```
<!-- Application Resources -->
<resource-ref>
    <description>application datasource</description>
    <res-ref-name>jdbc/[redacted]DB</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
    <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```

Figura 6: Archivo Configuración "Web.xml"

En primer lugar, se introduce una explicación de los parámetros mostrados en las figuras:

- **<jboss-web>:** marca el inicio del archivo de configuración "JBoss-Web.xml".

- **<resource-ref>:** marca el inicio de los recursos usados, dentro de este parámetro se añaden otros para configurar el recurso:

  - **<res-ref-name>:** indica el nombre referente al recurso.

  - **<jndi-name>:** Java Naming and Directory Interface, permite que las aplicaciones distribuidas busquen servicios de forma abstracta e independiente de los recursos.

  - **<description>:** sirve para poner una descripción del recurso que se va a crear.

  - **<res-type>:** indica el tipo de recurso, en este caso es de tipo "javax.sql.datasource" que es el que proporciona las conexiones con la base de datos.

  - **<res-auth>:** indica si el código del componente realiza el inicio de sesión mediante programación o si el contenedor inicia sesión. En este caso es el contenedor.

- **<res-sharing-scope>:** poner este parámetro como “Shareable” indica que este recurso puede ser utilizado por otros de la misma aplicación.

Para obtener los datos de la configuración del archivo “standalone.xml” de la configuración de JBoss se emplea un archivo "JBoss-Web.xml" en nuestro proyecto. Este archivo recoge el recurso de la base de datos usando en el parámetro *<res-ref-name>* el nombre con el que guardamos anteriormente la conexión a la base de datos en el archivo “StandAlone.xml”. A su vez, en el archivo “web.xml” de la aplicación tenemos enlazado el “JBoss-Web.xml” para configurarlo para su uso en la aplicación, usando el mismo nombre en el parámetro *<res-ref-name>*.

5.3 Utilización de la base de datos

Tras configurar el acceso a la base de datos, todas las consultas se realizarán mediante ficheros XML de MyBatis llamados *mappers* [23]. En estos *mappers* se encuentran todas las llamas a bases de datos.

A continuación, se muestra un ejemplo de un *mapper*:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.1//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="">

    <resultMap id="" type="">
        <id column="" property="" />
        <result column="" property="" />
        <result column="" property="" />
        <result column="" property="" />
        <result column="" property="" />
        <result column="" property="" />
        <result column="" property="" />
    </resultMap>

    <select id="" parameterType="java.lang.Long" resultType="java.lang.Long">
            SELECT  FROM
            INNER JOIN
            ON  AND =#{}
    </select>
```

Figura 7: “Mapper.xml”

Al comienzo de los *mapper* se introduce un *nameSpace* con la dirección del *mapper* en el proyecto. A partir de ahí se escriben todas las funciones.

Los *mapper* permiten usar cualquier tipo de llamada a base de datos, *select, insert, update…*

Para ello se hace de la manera que se observa en la figura 7: un “<” para abrir la función seguido del tipo de función con un identificador y los tipos de parámetros y resultados que se van a usar, finalizando con un “>”.

Una de las ventajas de MyBatis es poder pasar a las funciones parámetros que usar en ellas y el tipo de resultado que se espera para guardarlo a la hora de llamar a la función desde el código.

Dentro está la consulta escrita tal y como se haría desde un entorno como Oracle SQL Developer, a diferencia de los parámetros que le lleguen que se usaran mediante “#{nombreParámetro}”. Para acabar la consulta se añade “</nombreTipoFuncion>”.

Otra ventaja de MyBatis es que permite usar condiciones para las funciones. Para usar los parámetros en estas condiciones no es necesario usar “#{nombreParámetro}”, sino que se puede usar directamente con el nombre. En la siguiente figura se muestra un ejemplo de esto:

```
SELECT * FROM TABLA
    WHERE ID_TABLA=1
    <if test="null != nombreVariable and !nombreVariableLista.isEmpty()">
        AND VALOR=#{nombreVariable}
    </if>
```

Figura 8: Condición “Mapper.xml”

Esto permite que según los valores de los parámetros se puedan añadir o quitar líneas a función final que se manda a base de datos en función de si se cumple o no la condición que se declare.

5.4 Servicios de la aplicación

La parte de los servicios de la aplicación gestiona toda la lógica interna de la aplicación.

Las clases que gestionan estos servicios tienen de cabecera la notación *@Service*. Esta notación es propia de Spring y sirve para indicar que la clase es un *bean* de la capa de negocio. Además de esto, los atributos son marcados como *@Autowired* para inyectar automáticamente las dependencias del tipo especificado.

```
@Service
public class [redacted] implements [redacted] {

    @Autowired
    private static final String [redacted];

    @Autowired
    private [redacted] mapper;
```

Figura 9: Ejemplo Clase *Service*

5.5 Controladores de la aplicación

Estos controladores sirven para conectar la capa *front-end* con la capa *back-end* de la aplicación.

Mediante los servicios API REST [22] se crean clases con la notación *@RestController* y *@RequestMapping*. Esta última indica la dirección a la que se deberá llamar desde la capa *front-end* para acceder al controlador.

Dentro de esta clase están los métodos a los que se accederá dependiendo de la dirección después de la indicada en el *@RequestMapping.*

A continuación, se muestra un ejemplo de este tipo de clases:

Figura 10: Creación Clase Controlador

Dentro de *@RequestMapping* se introduce una constante con la dirección del controlador (/nombreControlador).

Figura 11: Método Interno Controlador

Este método tiene una cabecera *@PostMapping* que sirve para procesar solicitudes de publicación. Dentro de esta cabecera se indicará la dirección del método (/nombreMétodo) y lo que va a producir, en el caso de la figura 11 un JSON.

También existen *@GetMapping* para solicitudes de obtención, *@PutMapping* que es equivalente al *@PostMapping* y *@DeleteMapping* para eliminar la asignación de *URL*.

Estos métodos pueden tener parámetros que se manden desde el *front*, para ello se añade la notación *@RequestBody* junto al nombre del parámetro.

Todo esto permite la conexión entre la parte *back-end* y la parte *front-end* de la aplicación.

Para realizar estas llamadas la parte *front-end* deberá ser configurada para que apunte a las direcciones puestas en los controladores a los que se desea acceder en cada caso.

Figura 12: Ejemplo Llamada Controlador

Los métodos *front* que llaman a estos controladores tienen este estilo:

- Nombre método y parámetro.
- Preparación llamada *controller.*
- Llamada con la dirección /direcciónMapper/direcciónMétodo y con el parámetro.

5.6 Clases adicionales

A parte de todo esto, existen clases que permiten facilitar la programación de la aplicación, reutilizando código y teniendo todos los datos más ordenados.
Entre ellas:

- **Data Transfer Objects (DTO):** Los DTO [29] tienen la finalidad de crear un objeto con una serie de atributos que puedan ser enviados o recuperados por el servidor con una sola invocación. De esta manera, se almacena información en una sola clase.

- **Clases con constantes:** Estas clases sirven para almacenar atributos con valores fijos que se van a reutilizar en el código de la aplicación. Por ejemplo, direcciones de controladores.

- **Clases de excepciones:** Clases con las que crear excepciones propias para distintas situaciones que puedan ocurrir en el uso de la aplicación.

6. Pruebas y despliegue

Este apartado trata sobre las pruebas realizadas y los despliegues de la aplicación.

6.1 Pruebas unitarias

Las pruebas unitarias se realizaron con las librerías, Mockito, PowerMockito y JUnit.

JUnit [4] es un *framework* de java para la realización de tests de clases que permite comprobar que sus métodos realizan su cometido de forma correcta. Dentro de JUnit se puede usar el *framework* de Mockito para crear instancias de una clase que testear. Este *framework* es muy útil debido a que te permite definir el comportamiento de los métodos que testear.

Además de Mockito, existen variantes como PowerMockito que permiten ampliar las funcionalidades de Mockito como, por ejemplo, crear instancias de métodos estáticos, privados...

La cobertura mínima de código con estos tests debía ser de un 70%. Esta medida la marcó el cliente Debido a esto se fueron realizando tests de cada implementación de código que se realizaba para así mantener un alto porcentaje de cobertura y que no hubiera ningún problema al finalizar la aplicación.

Por otra parte, se realizaron pruebas del código Angular de la parte *front-end*. Para estas pruebas se utilizó la *suite* de *testing* Jasmine y el *test-runner* Karma para ejecutar las pruebas.

6.2 Pruebas de Integración

Las pruebas de integración [24] sirven para examinar si el ensamblaje entre los componentes que han sido probados unitariamente interactúa correctamente a través de sus interfaces tanto internas como externas.

Para este tipo de pruebas se utilizaron herramientas como PostMan o SoapUI que permiten llamar a los controladores de la aplicación para probar su correcto funcionamiento con los parámetros que se introduzcan en la herramienta.

- **PostMan [25]:** esta herramienta permite sobre todo el *testing* de API REST. Con Postman se puede monitorizar, escribir, documentar, *mockear* y simular pruebas para comprobar el correcto funcionamiento en la aplicación.

- **SoapUI [26]:** es una herramienta similar a PostMan que permite estableces conexiones a bases de datos mediante JDBC.

6.3 Pruebas de aceptación

Estas pruebas se realizan tras el desarrollo de funcionalidades del código.

Tras desarrollar una nueva funcionalidad, el código se sube primero al entorno de desarrollo en el cual el team leader comprueba que su funcionamiento sea correcto. Si este funcionamiento es erróneo o hace que otros fallen, el team leader notifica esto para que se corrija dicho fallo.

Una vez el team leader da el visto bueno se suben los cambios al entorno de integración en el cual el cliente prueba la nueva funcionalidad.

Si el cliente da el visto bueno, esta nueva funcionalidad se sube a producción donde se utiliza por los trabajadores.

6.4 Pruebas de seguridad

Estas pruebas se realizan para comprobar las vulnerabilidades de la aplicación.

En este caso, la herramienta utilizada para esto fue HCL AppScan. Esta herramienta sirve para monitorear y realizar pruebas de seguridad web. Prueba las aplicaciones en busca de vulnerabilidades de seguridad y muestra un informe con todas las vulnerabilidades encontradas junto a posibles soluciones.

En el caso de este proyecto hubo bastantes vulnerabilidades que se tuvieron que corregir antes de su entrega final. Algunas de ellas:

Falta atributo de seguridad en una cookie de sesión cifrada (SSL)

Falta atributo de seguridad en una cookie de sesión cifrada (SSL)	
Gravedad:	Media
Puntuación de CVSS:	6,4
URL:	[redacted]
Entidad:	XSRF-TOKEN (Cookie)
Riesgo:	Es posible robar la información de usuario y de sesión (cookies) que se enviaron durante una sesión cifrada
Causas:	La aplicación web envía cookies no seguras sobre SSL
Arreglo:	Añada el atributo 'Secure' a todas las cookies sensibles

Esta vulnerabilidad provoca que se pueda robar información de los usuarios que van a usar la aplicación. El problema se encuentra en la configuración de la *cookie* de sesión. Para solucionar esto y que la información de los usuarios no peligre se ha configurado el atributo *secure* en la *cookie* que indica que debe haber una seguridad sobre esta. Añadiendo la siguiente línea a la configuración dentro del archivo “Web.xml” se soluciona esta vulnerabilidad:

```
<session-config> -->
    <session-timeout>10</session-timeout>
    <cookie-config>
        <http-only>true</http-only>
        <secure>true</secure>
    </cookie-config>
    <tracking-mode>COOKIE</tracking-mode>
</session-config>
```

Parámetro de consulta en solicitud SSL

Parámetro de consulta en solicitud SSL	
Gravedad:	Baja
Puntuación de CVSS:	5,0
URL:	
Entidad:	(Parameter)
Riesgo:	Es posible robar datos sensibles como, por ejemplo, los números de tarjeta de crédito, los números de la seguridad social, etc. que se envían sin cifrar
Causas:	Los parámetros de consulta pasaron sobre SSL y pueden contener información sensible
Arreglo:	Utilice siempre los parámetros SSL y POST (cuerpo) al enviar información sensible.

Esta es otra vulnerabilidad encontrada en el desarrollo de la aplicación. En este caso el problema se encuentra en la manera en la que los datos que introduce el usuario desde la parte *front* de la aplicación se envían a la parte *back*. La manera de realizar esta transferencia era mediante los métodos de petición GET. Estos métodos tienen una gran desventaja. Los parámetros que se introducen en la página se mandan en la dirección URL. Esto permite que sea muy fácil robar los datos que usa un usuario.

Para evitar este posible robo se usan métodos de petición tipo POST. Estos métodos permiten que los datos introducidos no se muestren en la URL, en el caché o en el historial.

6.5 Despliegue

Para realizar los despliegues de la aplicación, tanto en el entorno de desarrollo como en integración o producción, el código debía pasar antes por la herramienta Jenkins. Esta herramienta comprueba que la aplicación es correcta, compilando el código, creando una imagen de SonarQube y mirando que pase los valores mínimos de cobertura…

Una vez pasado el código por Jenkins, mediante la herramienta MobaXterm [30] se realizan los despliegues de las aplicaciones. A esta herramienta se entra con un usuario que tenga permisos para realizar los despliegues. Además de realizar estos despliegues se presentan más opciones. Entre ellas:

- **Despliegue de aplicaciones**

  Con esta opción se despliega la aplicación en el entorno de desarrollo, integración y producción según se elija.

- **Arranque y parada de servidores**

  Esta opción sirve para arrancar o parar los servidores en los que probar la aplicación.

- **Estado de servidores**

  Con esta opción compruebas el estado de los servidores. Esto es útil para saber si los servidores se han caído o hay un error en la aplicación que haya provocado que la aplicación no funcione.

7. Tareas realizadas

Este proyecto ha sido desarrollado en grupo en la empresa, sin embargo, en este apartado se detallan las tareas concretas realizadas por mí. Además, junto a estas tareas se detallan ejemplos con funcionalidades reales de la aplicación.

7.1 Desarrollo de las pantallas

La creación de nuevas pantallas para cada función de la aplicación ha sido una de mis tareas a lo largo del proyecto. Para ello, se utilizan tecnologías como Angular con JavaScript, HTML, CSS, …

Por una parte, se crea la estructura de la pantalla mediante HTML. Aquí se divide la cabecera de la pantalla, el logo de la aplicación, las opciones de *login* y cambio de idioma, del cuerpo de la aplicación en el que están las tablas, campos de entrada de datos, formularios, …
Además, en todos estos elementos de la pantalla se pueden añadir funciones de código desarrolladas, en este caso, en JavaScript. Estas funciones se pueden usar para hacer llamadas a la parte *Back-End* de la aplicación y que se obtengan resultados sobre acciones que se hagan en estos elementos de la parte *Front-End*. Por ejemplo, si se registra una asistencia a una planta y se añade un usuario a dicha asistencia se comprueba con una función en el campo de entrada del nuevo usuario que se quiere añadir que dicho usuario existe. Esto hace una llamada al *Back-End* que a su vez comprueba con una llamada a la base de datos si el usuario existe y después envía una respuesta de vuelta a la parte *Front-End* para que según si existe o no se muestre un error avisando que no existe o se añada correctamente ese usuario.

En este caso, yo me he encargado de realizar la parte HTML de todas las pantallas y, además, de realizar también funciones JavaScript para el funcionamiento de los elementos de estas pantallas y modificar los estilos mediante CSS [9].

Muchas de las funciones que he realizado en JavaScript están relacionadas con llamas a la parte *Back-End* para realizar tanto consultas en la base de datos como para pasar validaciones. Estas tareas son explicadas en el capítulo 5.

Por último, muchos de los estilos de los elementos de la pantalla se han modificado mediante CSS. Tanto márgenes como tamaños o estilos de tablas, botones, formularios, … han sido modificados.

7.2 Desarrollo de consultas de base de datos

Como se ha comentado en el apartado 7.1, hay situaciones en las que se deben hacer consultas a la base de datos. La creación de estas consultas ha sido otra de mis funciones.

Siguiendo el ejemplo del apartado 7.1, cuando se quiere añadir un usuario adicional a una asistencia se debe comprobar que ese usuario exista. Para ello se hace una llamada a la base de datos con el identificador del usuario que se quiere añadir para comprobar si en la tabla donde se almacenan los usuarios existe dicho usuario.

Otro caso es cuando se crea una asistencia, esta asistencia nueva se debe almacenar en la base de datos junto a sus detalles, usuario que entra en la planta, usuarios adicionales si los hay, fecha de la entrada, nombre de la planta, …

Una vez la asistencia finalice ya se hará otra llamada a la base de datos para borrar la asistencia que se ha almacenado anteriormente.

La explicación de este tipo de tareas es detallada en el capítulo 5.3.

7.3 Creación de validaciones

Las validaciones sirven para comprobar que los datos que se buscan o que se introducen tienen los formatos o valores que deben ser. Por ejemplo, si se quiere crear un trabajo y se indica que la fecha de finalización va a ser antes de la de creación debería mostrarse un error indicando que esto no es posible. Para comprobar si la fecha de finalización indicada es correcta se pasa por una validación que se encargue de comprobar esto. Si la fecha es correcta la validación simplemente deja que continue el funcionamiento de la aplicación y que se cree el trabajo, pero si es incorrecta se detendrá el funcionamiento y se mostrará un error que indique que los datos introducidos no son correctos.

Otra validación comprueba que las restricciones de una planta han sido aceptadas. Esto se hace comprobando el valor de un elemento *checkbox* de HTML. Al estar marcado se han aceptado las restricciones. Para ello se envía el valor del *checkbox* y la validación comprueba que esté marcado. Si no es así se muestra un mensaje diciendo que es necesario aceptar las restricciones antes de seguir.

Están validaciones están organizadas en paquetes como se explica en el capítulo 5.1.

7.4 Realización de pruebas unitarias y de integración

Otra de mis tareas ha sido la realización de pruebas unitarias y de integración de las funciones creadas. Estas pruebas se realizan en la parte *front-end* y *back-end* como se explica en el capítulo 6.1.

7.5 Traducciones mediante documentación i18n

Al ser una aplicación con varios idiomas, otra de mis tareas ha sido la creación de traducciones para los textos usados en la aplicación. Según el idioma seleccionado en la aplicación y mediante las documentaciones i18n se cargaban los textos, nombres, … en un idioma u otro.

7.6 Corrección de vulnerabilidades

Las vulnerabilidades son fallos en el código que permiten a personas externas obtener información de usuarios de la aplicación. Por ejemplo, obtener el usuario y contraseña de un trabajador que inicie sesión. Para solucionar esto, otra de mis funciones ha sido la corrección de estos fallos que provocan un funcionamiento no seguro de la aplicación.

En el capítulo 6.4 se muestran algunos ejemplos de vulnerabilidades.

8. Conclusión

El objetivo del proyecto ha sido la creación de una nueva aplicación que cumpla todos los requisitos marcados por el cliente. Al finalizar la aplicación, todos estos requisitos han sido cumplidos. Por ello, el cliente quedó satisfecho con el equipo por el trabajo realizado. Además, se terminó con tiempo suficiente para poder realizar múltiples comprobaciones sobre el correcto funcionamiento de todas las funcionalidades de la aplicación, corrigiendo pequeños fallos o realizando alguna corrección de estilos.

A nivel personal, este proyecto me ha servido para obtener experiencia sobre cómo es el desarrollo de un proyecto de principio a fin. Gracias a este proyecto he podido afianzar muchos conocimientos y técnicas de desarrollo web. Además, he adquirido nuevos conocimientos que de cara al futuro serán muy útiles para el desarrollo de nuevos proyectos.

Por otra parte, trabajar con un equipo me ha ayudado a mejorar habilidades de trabajo en equipo y comunicación. Tratar con el cliente y aprender cómo gestionar sus posibles dudas o inquietudes sobre funcionalidades de la aplicación. Aprender a gestionar problemas que puedan surgir y que obliguen a tener que resolverlos lo más rápido posible para hacer las entregas en el tiempo estimado.

Actualmente, este es uno de los proyectos de los cuales estoy a cargo de su mantenimiento. Haber trabajado en su desarrollo me facilita el trabajo actual de mantenimiento ya que me permite conocer la estructura y código del proyecto de manera que si surge algún error o se necesita realizar algún cambio puedo hacerlo con mayor rapidez.

Bibliografía

[1] Xavier Albaladejo "Reunión diaria de sincronización del equipo (Scrum daily meeting)"
https://proyectosagiles.org/reunion-diaria-de-sincronizacion-scrum-daily-meeting/#:~:text=Scrum%20daily%20meeting)-,Reuni%C3%B3n%20diaria%20de%20sincronizaci%C3%B3n%20del%20equipo%20(Scrum%20daily%20meeting),pueden%20ayudar%20unos%20a%20otros.

[2] Picodotdev "Internacionalización (i18n) en JavaScript"
https://picodotdev.github.io/blog-bitix/2015/01/internacionalizacion-i18n-en-javascript/

[3] Osmosis Latina "¿Qué es JBoss? ¿Qué hace JBoss?"
https://www.osmosislatina.com/jboss/basico.htm

[4] Aj Castillo "¿Qué es jUnit?"
https://elrincondeaj.wordpress.com/2012/07/09/junit/

[5] Rafael Márquez "Evalúa la calidad de tu código con Sonarqube"
https://www.paradigmadigital.com/dev/evalua-la-calidad-de-tu-codigo-con-sonarqube/

[6] Javier Garzas "¿Qué es Jenkins?"
https://www.javiergarzas.com/2014/05/jenkins.html

[7] Juan Carlos Rubio "Qué es GIT y para qué sirve"
https://openwebinars.net/blog/que-es-git-y-para-que-sirve/

[8] MDN contributors "¿Qué es JavaScript?"
https://developer.mozilla.org/es/docs/Learn/JavaScript/First_steps/What_is_JavaScript

[9] Gustavo B "¿Qué es CSS?"
https://www.hostinger.es/tutoriales/que-es-css

[10] David Perálvarez "¿Qué es el HTML y el CSS?"
https://silicodevalley.com/que-es-el-html-y-el-css/

[10.1] Miguel Ángel Álvarez "Qué es HTML"
https://desarrolloweb.com/articulos/que-es-html.html

[11] Pastor Ramos "Qué es y para qué sirve SQL"
https://styde.net/que-es-y-para-que-sirve-sql/

[12] QualityDevs "¿Qué es Angular y para qué sirve?"
https://www.qualitydevs.com/2019/09/16/que-es-angular-y-para-que-sirve/

[13] Antonio Pérez "Cómo usar Testing en Angular con Jasmine y Karma"
https://www.digital55.com/desarrollo-tecnologia/como-usar-testing-angular-jasmine-karma/

[14] Formatalent "Definición de Java y usos
https://formatalent.com/definicion-de-java-y-usos/

[15] Curiotek "Spring Framework ¿Qué es y para qué sirve? – Java"
https://curiotek.com/java-que-es-spring/

[16] Miguel Parada "Qué es jQuery"
https://openwebinars.net/blog/que-es-jquery/

[17] Oracle "Qué es SQL Developer?"
https://www.oracle.com/es/tools/technologies/whatis-sql-developer.html

[18] Editorial de Geekflare "Introducción a Maven: una herramienta sencilla de gestión de proyectos"
https://geekflare.com/es/apache-maven-for-beginners/

[19] Programmerclick "¿Qué es mybatis?"
https://programmerclick.com/article/6228796541/

[20] Requeridos Blog "Requerimientos Funcionales y No Funcionales, ejemplos y tips"
https://medium.com/@requeridosblog/requerimientos-funcionales-y-no-funcionales-ejemplos-y-tips-aa31cb59b22a

[21] Instituto Politécnico Nacional UPIICSA "Diseño Arquitectónico"
http://upiicsa.tecnologia-educativa.com.mx/docs/u2/s3/DISENO%20ARQUITECTONICO.pdf

[22] José Manuel Rosa Moncayo "¿Qué es REST? Conoce su potencia"
https://openwebinars.net/blog/que-es-rest-conoce-su-potencia/

[23] Developpaper "Spring Boot Integrated MyBatis and SQL Server Practice"
https://developpaper.com/spring-boot-integrated-mybatis-and-sql-server-practice/

[24] Manuel Cillero "Pruebas de Integración"
https://manuel.cillero.es/doc/metodologia/metrica-3/tecnicas/pruebas/integracion/

[25] Alejandro López "Qué es Postman y para qué sirve"
https://openwebinars.net/blog/que-es-postman/

[26] Pablo Fonseca "Pruebas API con SoapUI"
https://www.sngular.com/es/pruebas-api-con-soapui/

[27] Lucidchart "Qué es un diagrama entidad-relación"
https://www.lucidchart.com/pages/es/que-es-un-diagrama-entidad-relacion

[28] Lucidchart "Qué es un diagrama de flujo"
https://www.lucidchart.com/pages/es/que-es-un-diagrama-de-flujo

[29] Oblancarte "Data Transfer Object (DTO) – Patrón de diseño"
https://www.oscarblancarteblog.com/2018/11/30/data-transfer-object-dto-patron-diseno/

[30] Sergio De Luz "MobaXterm: el mejor terminal para Windows con cliente SSH y SFTP"
https://www.redeszone.net/analisis/software/mobaxterm-terminal-windows/