**GRADO EN INGENIERÍA INFORMÁTICA**

Trabajo de Fin de Grado

DISEÑO DE UNA APLICACIÓN WEB PARA LA DISTRIBUCION DE ESPACIOS

Autor: Alejandro Campos Calvo

Tutor: Nombre y Apellidos del tutor

|  |  |
| --- | --- |
| **UNIVERSIDAD DE LEÓN**  **Escuela de Ingenierías Industrial, Informática y Aeroespacial**  **GRADO EN INGENIERÍA INFORMÁTICA**  **Trabajo de Fin de Grado** | |
| **ALUMNO:** Alejandro Campos Calvo | |
| **TUTOR:** | |
| **TÍTULO:** Diseño de una aplicación web para la distribución de espacios | |
| **TITLE:** | |
| **CONVOCATORIA:** Julio, 2019 | |
| **RESUMEN:**  En este trabajo se realiza el desarrollo de una aplicación web para la distribución, visualización y reserva de espacios. El principal objetivo es automatizar el proceso de reserva de espacios, a través de una interfaz de usuario intuitiva y eficiente, que permita de un solo vistazo comprobar que espacios están liberados para su posible reserva en una fecha determinada.  Se estudiará la posibilidad de utilizar la aplicación para facilitar la distribución de aulas de la facultad de ingenierías de la Universidad de León, por lo que la aplicación contará con una comunicación bidireccional con la API de Google Calendar, dando lugar a la actualización en tiempo real de cada uno de los calendarios correspondientes a las asignaturas de los grados y másteres de esta facultad.  Las reservas realizadas a través de la aplicación se verán reflejadas a su vez en cada uno de los calendarios de Google de la asignatura correspondiente, asegurando que no existirán solapamientos de espacios entre los horarios de las diferentes asignaturas de la facultad. | |
| **ABSTRACT:** | |
| **Palabras clave:** Aplicación Web, Distribución de espacios, Google Calendar, API. | |
| **Firma del alumno:** | **VºBº Tutor/es:** |

Í Índice de contenidos

[Í Índice de contenidos I](#_Toc11844399)

[Índice de figuras II](#_Toc11844400)

[Índice de tablas III](#_Toc11844401)

[Glosario de términos IV](#_Toc11844402)

[Introducción 1](#_Toc11844403)

[Objetivos 2](#_Toc11844404)

[Metodología 2](#_Toc11844405)

[1 Estado del arte 4](#_Toc11844406)

[2 Planificación 5](#_Toc11844407)

[2.1 Tareas a realizar 6](#_Toc11844408)

[2.2 Diagrama de Gantt 7](#_Toc11844409)

[3 Tecnologías utilizadas 8](#_Toc11844410)

[3.1 Servidor Back-End 8](#_Toc11844411)

[3.2 Cliente Front-End 12](#_Toc11844412)

[4 Creación de la aplicación 16](#_Toc11844413)

[4.1 Metodología 16](#_Toc11844414)

[4.2 Primeros pasos 16](#_Toc11844415)

[4.3 Fase de desarrollo 17](#_Toc11844416)

[5 Análisis del prototipo final 24](#_Toc11844417)

[5.1 Estructura del Back-End 25](#_Toc11844418)

[5.2 Estructura del Back-End 31](#_Toc11844419)

[Conclusión 32](#_Toc11844420)

[Aportaciones realizadas 32](#_Toc11844421)

[Trabajos futuros 32](#_Toc11844422)

[Problemas encontrados 32](#_Toc11844423)

[Opiniones personales 32](#_Toc11844424)

[Lista de referencias 32](#_Toc11844425)

[ANEXO A: Control de Versiones 33](#_Toc11844426)

[ANEXO B: Manual de usuario 34](#_Toc11844427)

Índice de figuras

[Figura 3.1 User Repository Interface 10](file:///C:\Users\Alex\Desktop\TFG.docx#_Toc11844019)

[Figura 3.2 Plantilla Thymeleaf 14](file:///C:\Users\Alex\Desktop\TFG.docx#_Toc11844020)

[Figura 5.1 Estructura del prototipo 24](file:///C:\Users\Alex\Desktop\TFG.docx#_Toc11844021)

[Figura 5.2 Configuración Google OAuth 25](#_Toc11844022)

[Figura 5.3 Inicio de la aplicación 26](#_Toc11844023)

[Figura 5.4 Esquema repositorio 26](#_Toc11844024)

Fuente: <https://www.arquitecturajava.com/spring-stereotypes/>

[Figura 5.5 Esquema servicio 27](#_Toc11844025)

Fuente: <https://www.arquitecturajava.com/spring-stereotypes/>

[Figura 5.6 Esquema controlador 27](#_Toc11844026)

Fuente: <https://www.arquitecturajava.com/spring-stereotypes/>

[Figura 5.7 Configuración de seguridad 28](#_Toc11844027)

[Figura 5.8 Página principal 29](#_Toc11844028)

Índice de tablas

[Tabla 4.1 Sprint 1 16](#_Toc11845111)

[Tabla 4.2 Sprint 2 17](#_Toc11845112)

[Tabla 4.3 Sprint 3 17](#_Toc11845113)

[Tabla 4.4 Sprint 4 18](#_Toc11845114)

[Tabla 4.5 Sprint 5 19](#_Toc11845115)

[Tabla 4.6 Sprint 6 19](#_Toc11845116)

[Tabla 4.7 Sprint 7 20](#_Toc11845117)

[Tabla 4.8 Sprint 8 21](#_Toc11845118)

[Tabla 4.9 Sprint 9 21](#_Toc11845119)

[Tabla 4.10 Sprint 10 22](#_Toc11845120)

[Tabla 4.11 Sprint 11 23](#_Toc11845121)

Glosario de términos

Catálogo de términos específicos del contexto del trabajo.

Introducción

Actualmente para la organización de aulas en la facultad de ingenierías, se utilizan alrededor de unos 500 calendarios de Google diferentes, que almacenan las horas de clase de cada una de asignaturas pertenecientes a los distintos grados y másteres que tiene ofertados la facultad. Estos calendarios, se ven reflejados en la página oficial por cursos, utilizando un calendario espejo, que tan solo tiene permisos de visualización sobre las asignaturas que pertenecen al curso que define dicho calendario.

Google Calendar, nos ofrece una herramienta potente y muy eficaz para mostrar eventos independientes, sin embargo, no es tan eficaz si lo que queremos es que no se permita la colisión de horarios según la ubicación de dichos eventos. Por otra parte, esta herramienta nos permite compartir horarios de manera muy sencilla con gran número de usuarios, por lo tanto, el objetivo de este trabajo es desarrollar una aplicación que permita la edición de todos estos horarios en una interfaz centrada en la distribución de espacios por hora y no tanto en los eventos individuales como tal, respetando en última instancia los calendarios actualmente existentes y volcando la información sobre ellos, pero teniendo siempre la posibilidad de realizar el proceso contrario, de importar información actualizada directamente sobre Google Calendar en caso de necesidad.

El objetivo de la aplicación por tanto, es asegurar que en ningún momento se produzca una colisión de dos horas de clase en un mismo espacio y permitiendo corregirla en caso de existir actualmente en los horarios. La visualización de estos se realizará diariamente, mostrando una lista de espacios, y las horas en las cuales están libres para su posible reserva, la cual únicamente se podrá realizar desde determinadas cuentas de usuario.

El acceso a la aplicación estará restringido a todas aquellas cuentas pertenecientes a la organización de la universidad de león realizando una identificación a través del proveedor de esta y solo ciertos usuarios tendrán acceso a la realización de reservas, mientras que la mayoría únicamente podrán visualizar los horarios en tiempo real de la escuela.

Objetivos

Por lo tanto, los objetivos de la aplicación a desarrollar están claramente definidos. En un primer lugar, debemos asegurar la consistencia del sistema verificando que en ningún momento se produzca una colisión durante la reserva de espacios y que, en caso de existir, pueda ser resuelta de forma rápida. Por otra parte, la aplicación debe poseer integración de los calendarios de Google, dejando reflejado en ellos toda la información que se encuentra en el sistema, y además, actualizándose a sí mismo desde estos en caso de que suceda algún cambio en los calendarios del propio Google.

En cuanto a la interfaz y el intercambio de información con el usuario, el sistema deberá de ser lo más transparente posible a este, con una interfaz intuitiva y fácil de usar, asegurando que la información de los cambios realizados se vuelca correctamente en los calendarios finales, y permitiendo por lo tanto, la edición de reservas por parte de los administradores.

Como último objetivo opcional, se plantea el desarrollo de un sistema de reservas provisionales, para facilitar el trámite de reserva de espacios que actualmente se realiza en papel.

Metodología

Debido al origen del proyecto, y teniendo en cuenta que este pertenece a la rama de la ingeniería de software, es casi imposible cerrar un diseño a la primera para pasarlo a una fase de desarrollo sin que sufra modificaciones por el camino, bien tanto por parte del cliente, como del propio equipo de desarrollo, por esta razón, la metodología a utilizar será una metodología ágil que acepte cambios en el software y en los objetivos iniciales del proyecto, adaptándose así a unas necesidades del cliente que pueden surgir durante la fase de desarrollo del proyecto y enfocándose en satisfacer las necesidades de este.

Se realizarán reuniones semanales sujetas a la disponibilidad del propio cliente y del equipo de desarrollo, cuyo objetivo es el de dar una visión del avance del proyecto al propio cliente, obteniendo “feedback” y ajustándolo a sus necesidades. Estas reuniones nos ayudarán a la creación de prototipos cambiantes, a los que se les irá añadiendo funcionalidad a lo largo del desarrollo, revisando y mejorando las partes del producto que necesiten algún cambio.

Esta metodología iterativa e incremental nos da una visión de avance del desarrollo, permite manejar posibles riesgos del proyecto y dando una mayor experiencia al equipo de desarrollo aumentando la productividad de este y generando un software de calidad.

# Estado del arte

# Planificación

Se realiza una planificación por objetivos, disponiendo en todo momento de un prototipo funcional, que irá creciendo a lo largo del desarrollo, cuyos cambios se verán reflejados en un repositorio de un controlador de versiones como es GitHub, pudiendo rectificar cualquier cambio en el proyecto gracias a este.

Estos objetivos se irán logrando semanalmente, permitiendo una ampliación de plazo en caso de ser necesario. Un listado de las distintas fases por las que ha pasado el desarrollo, cuyos objetivos están especificados en la parte derecha, sería el siguiente:

* 11 de febrero – 25 de febrero: Planteamiento inicial.
* 25 de febrero – 28 de febrero: Servidor Back-End.
* 28 de febrero – 10 de marzo: Autenticación OAuth2.
* 10 de marzo – 15 de marzo: Primer Front-End.
* 15 de marzo – 24 de marzo: Sistema de Reservas.
* 24 de marzo – 31 de marzo: Funcionalidades Front-End.
* 31 de marzo – 1 de mayo: Reforma del sistema de modelos.
* 1 de mayo – 19 de mayo: Reforma del sistema de IDs.
* 19 de mayo – 26 de mayo: Sistema de auto actualización.
* 26 de mayo – 5 de junio: Sistema de corrección de reservas.
* 5 de junio – 17 de junio: Creación configuración JSON.

Al final de cada una de estas fases, se obtiene un prototipo que con sus limitaciones, es completamente funcional para ser mostrado al cliente y estar sujeto a futuros cambios que este necesite, quedando reflejado en el repositorio online del propio proyecto en todo momento.

## Tareas a realizar

Para describir las tareas listadas anteriormente, se agruparan en tres grandes grupos, que serán resumidos a continuación conteniendo las ideas principales de las tareas que contienen, que serán descritas en partes posteriores del documento más detalladamente.

### Planteamiento Inicial

En esta primera fase, se realizan reuniones para sentar las bases de la aplicación y cuál será el principal problema a solucionar, discutiendo las distintas partes de esta, construyendo al desarrollador una idea de la posible solución que pudiera tener. En ella, el desarrollador decide que tecnologías utilizar durante las siguientes fases del desarrollo.

### Desarrollo de la interfaz

En esta fase intermedia, después de haber desarrollado un primer prototipo básico, y haber sentado un poco la base de lo que será en el futuro el servidor de la aplicación y la autenticación de usuarios que este utiliza, el desarrollador se reúne con el cliente para decidir junto a este cómo será la interfaz de usuario inicial, sujeta a cambios en el futuro, pero que será la base del prototipo final.

### Refactorización

Esta podría ser la fase final del desarrollo del primer prototipo completo y funcional de la aplicación, en ella se ha tenido que realizar una refactorización del código del servidor de back-end casi completa a causa de unos requisitos introducidos por el cliente, que conllevan la utilización de la API de Google Calendar. Si bien en un primer momento era conocida por el desarrollador la necesidad de llegar a utilizar esta API, no es hasta esta fase cuando realmente se crea esa necesidad por parte del cliente.

### Configuración

Una vez finalizada la refactorización, y el desarrollo de las últimas tareas en la aplicación como tal, se pasa a una última fase de configuración, en la cual se creará un archivo de configuración que contiene una lista de las carreras que serán mapeadas en la aplicación, así como las asignaturas que estas contienen y su id de Google Calendar, para futuros accesos de las cuentas robot que la aplicación contiene para la actualización de los calendarios en tiempo real. Además, se añade una lista de espacios que se verán reflejados en las tablas una vez iniciada la aplicación, así como una pequeña lista de configuraciones iniciales, que por ahora, solo contiene la fecha desde la cual se comenzaría a mapear el horario, y que podría ser actualizada anualmente por los administradores del servidor web.

## Diagrama de Gantt

Se adjunta a continuación un diagrama de Gantt que muestra en una línea de tiempo las diferentes tareas realizadas que previamente hemos descrito, indicando cuando se ha iniciado y finalizado cada una de ellas hasta la finalización del proyecto.

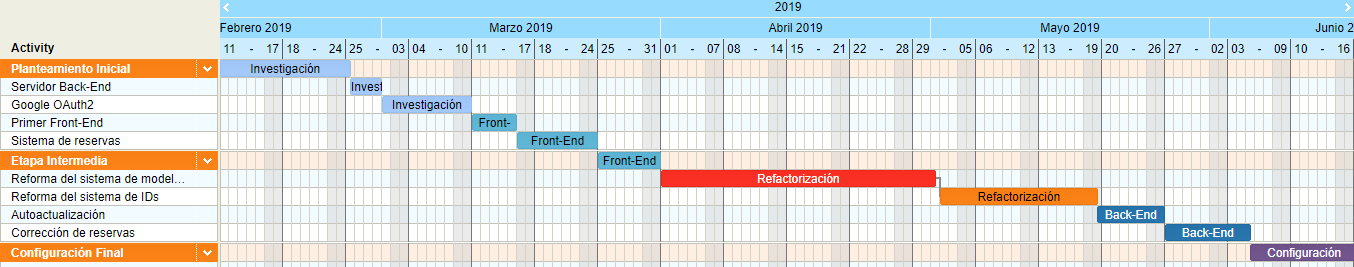


Figura 2.1 Diagrama de Gantt

Se podrá consultar una versión completa y de mayor resolución en el siguiente enlace: [Diagrama de Gantt](https://github.com/acampc01/TFG/blob/master/Gantt%20Completo.png). Las tareas mostradas en el diagrama, se verán reflejadas a los largo de los sprints llevados a cabo, en el capítulo sobre la metodología en este mismo documento.

# Tecnologías utilizadas

A continuación, se realizará una descripción de las tecnologías utilizadas durante el desarrollo de la aplicación comenzando por las diferentes tecnologías que conforman desde servidor de back-end hasta el front-end de la aplicación, y una pequeña parte final de configuración.

## Servidor Back-End

Como servidor web utilizaremos un Tomcat incluido en un proyecto Java Maven configurado gracias al Framework Spring, que junto a un conjunto de dependencias, proporcionan una herramienta completa para el desarrollo de nuestra aplicación web. Para la creación de nuestro proyecto Maven, utilizaremos “*Spring Initializr*”, disponible en este [enlace](https://start.spring.io/), que genera automáticamente el proyecto Maven con las dependencias especificadas, en nuestro caso, las listadas a continuación:

1. Spring Web Starter
2. Spring Security
3. Spring Data JPA
4. Spring DevTools
5. Thymeleaf
6. MySQL Driver
7. OAuth2 Client

Si bien vamos a explicar las más importantes en los apartados siguientes, hay que hacer mención a dependencias como “*Spring DevTools”*, que a pesar de no aportar nada al proyecto en sí, mejoran increíblemente la experiencia de desarrollo del mismo, permitiendo un reinicio y re compilación rápido de nuestro proyecto, durante la fase de codificación del mismo. Además, una última mención a “*MySQL Driver”*, que permite la interacción entre “*Spring Data JPA*” y la base de datos MariaDB utilizada en este proyecto.

### Spring Boot

Es el punto de partida para cualquier aplicación basada en Spring Framework, diseñado para acelerar el proceso de lanzamiento de aplicaciones web en fase de producción, permitiendo una construcción en segundos de un servidor web acompañado de distintas características añadidas por las dependencias adicionales. Es la base de nuestro proyecto, y la parte más importante, junto a las dependencias de seguridad de autenticación.

#### Spring Boot Security y OAuth2

Ambos forman nuestro framework de seguridad, potente y personalizable, que permite añadir un control de acceso y de autenticación a nuestro servidor web, si bien solo “*Spring Boot Security*” ya permite estas dos cosas, la dependencia de OAuth2, nos permite realizar la autenticación desde el servidor de una tercera entidad, en este caso, pasamos por Google OAuth2.0, que realizará una autenticación de los usuarios que accedan a nuestro servidor, asegurando que pertenecen a la organización de Google correspondiente con la Universidad de León. Volviendo a la dependencia de seguridad, esta nos permite realizar un control de acceso por rol de usuario, limitando las características del front-end según el tipo de usuario identificado en nuestro servidor.

Hasta ahora, tenemos un sistema que nos autentica la identidad del usuario que accede a la aplicación (Google OAuth2.0) y un sistema que controla qué podemos ver una vez que nuestro servidor ya tiene la confirmación de Google. En la parte perteneciente al Front-End, veremos una herramienta que en parte, acompaña a estas limitaciones de acceso por rol, llamada “*Thymeleaf*”.

#### Spring Data JPA

Antes de explicar de qué trata la dependencia de Data JPA, debemos explicar que nos aporta su antecesor Spring Data. Spring Data nos da un modelo de programación basado en Spring para el acceso a datos, permitiendo el intercambio de información con base de datos de cualquier tipo, así como la creación de aplicaciones para servicios en la nube, etc. Es un framework que contiene muchísimos subproyectos, según el tipo de intercambio que deseemos, en nuestro caso, un intercambio con una base de datos relacional de tipo MySQL.

Spring Data JPA es uno de los subproyectos más grandes pertenecientes a Spring Data, y nos facilita la implementación de repositorios JPA de una forma increíble, permitiendo la creación de peticiones a base de datos con tan solo la implementación de la cabecera de los métodos en una clase interfaz java, reduciendo el esfuerzo y el tiempo que esto conlleva. Para ilustrar esta explicación, se adjunta una captura de la clase repositorio perteneciente al modelo Usuario en nuestro proyecto:

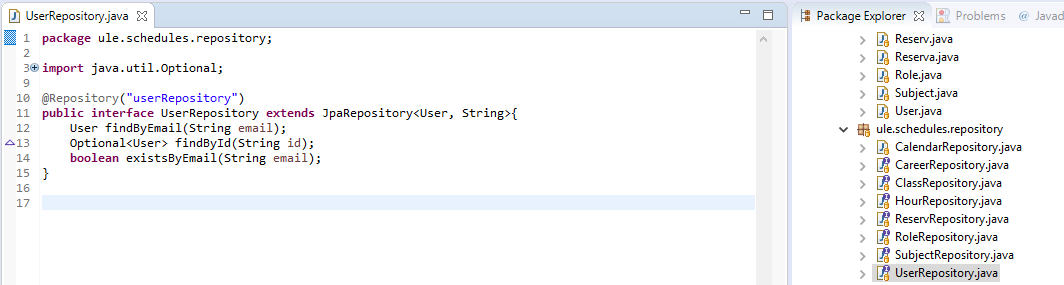


Figura 3.1 User Repository Interface

Como se puede observar, con tan solo una sintaxis específica en los nombres de los métodos, se puede obtener un intercambio satisfactorio con nuestra base de datos, gracias a que es Spring Data JPA el encargado de la creación de las peticiones.

### Google Calendar API

Otro de los grandes pilares sobre los que trabaja nuestra aplicación, permitiéndonos intercambiar información con los calendarios de google específicos de cada asignatura de la facultad, esto logra la API de Google Calendar. Si bien gracias a Spring Boot y sus dependencias, podríamos crear una aplicación web completamente independiente, en nuestro caso y a causa del diseño elegido, utilizaremos esta API para aportar información a nuestra aplicación.

La API de Google Calendar, permite obtener tanto los calendarios como los eventos de estos a los que se tiene acceso desde una cuenta de google, bien sea una personal, o una cuenta robot, como vendría a ser en nuestro caso, integrando Google Calendar en nuestra aplicación.

Como en la facultad de ingenierías, a la que está enfocado este proyecto, posee alrededor de 550 asignaturas con su correspondiente calendario, y el conjunto de estos calendarios tiene alrededor de 13000 eventos, y debido a las limitaciones de la API, que nos limita a 250 calendarios por cuenta, y a 2500 eventos por calendario, nos veremos obligados a crear dos cuentas robot, una para las asignaturas correspondientes a los diferentes grados, y otra para las asignaturas correspondientes a los másteres que desea mapear la facultad, a las cuáles se les otorgará permiso de edición en los calendarios.

Una vez otorgados estos permisos, la API nos permitirá obtener un listado de calendarios y un listado de eventos por calendario, recolectando la información necesaria para la creación de las reservas en nuestra aplicación. Este recolectado, se realiza distinguiendo dos campos dentro de un evento cualquiera de la API, la fecha de inicio y finalización del evento, y su campo de localización. Por especificaciones del cliente, los eventos deben comenzar y finalizar en horas a en punto o y media, así como tener en el apartado de ubicación una cadena de caracteres perteneciente a un listado predefinido de aulas con nombre y numeración o simplemente tener este apartado vacío, en cualquier otro caso, el evento cuya ubicación sea desconocida, será ignorado por el sistema de reservas. Todo esto, da lugar a un filtrado de eventos, que nos deja alrededor de unos 9800 eventos con los que trabajar dentro del sistema de reservas, todos ellos identificados por la API de calendar con una ID única para cada uno permitiendo la modificación de los mismos.

### Google OAuth2.0

Las autorizaciones OAuth (Open Authorization) son una forma sencilla de autenticar usuarios para sitios web o aplicaciones, permitiendo a un servidor A identificar usuarios y compartir esa identificación con otros servidores B que necesiten esa autorización, además de utilizar esa autenticación para dar servicios propios. En nuestra aplicación, podríamos hablar de dos autenticaciones diferentes, una para los usuarios que acceden a través del front-end pasando por el sistema de autenticación de google y por el de SSO (Single Sign-On) de la Universidad de León, y otra autenticación para la utilización de la API de Google Calendar, que se realizará gracias a la creación de cuentas de servicio a través de la consola de Google perteneciente al proyecto.

Volviendo al primer tipo de autenticación, cuando un usuario cualquiera se conecte a nuestra aplicación a través de un navegador, entrará en la página de bienvenida. Tras pulsar en el botón de acceso, nuestro servidor recibirá una petición de login, y redireccionará al usuario a la página de inicio de sesión de Google, tras introducir un correo, que obligatoriamente deberá pertenecer a la organización de la Universidad de León, se le redireccionará al servicio de Single Sign-On de autenticación centralizada de la universidad. Una vez realizado todo el proceso de autenticación, el servidor dejará pasar al usuario a la página principal en la que puede consultar las reservas de aulas de los distintos espacios que tiene la facultad de ingenierías.

Para la autenticación de las cuentas de servicio, parte de del servidor de back-end, y más exactamente del repositorio encargado del intercambio de información con Google Calendar, se utiliza la librería de autenticación OAuth de Google, que permite crear una credencial utilizando los datos de las cuentas de servicio guardadas en un JSON y con esa credencial construir un cliente con el que realizar intercambios de información en los diferentes calendarios sobre los que tenga acceso dicha cuenta.

## Cliente Front-End

Como estamos construyendo una aplicación web nuestro front-end tendrá como base HTML, JS y CSS, acompañados de diferentes dependencias que facilitarán en mayor o menor medida el trabajo de implementación. Todas las dependencias al igual que en el caso del servidor back-end serán incluidas a través del archivo de dependencias de Maven, pero en este caso, al ser para utilizarlas en la parte del cliente, se debe añadir una dependencia a mayores para localizar todas las librerías de front-end dentro de nuestro proyecto, desde los diferentes HTML, llamada “*Webjars*”. Si bien esto no es necesario, en caso de que alguna de las librerías utilizadas deje de funcionar a través de CDN, nosotros las tendríamos disponibles debido a que se las tenemos disponibles en nuestro servidor.

### Thymeleaf

Sin duda, es la dependencia para el desarrollo de front-end de nuestro proyecto, permitiendo la creación de plantillas HTML de forma muy sencilla e intuitiva. Trabaja en el servidor de back-end pero debido a su objetivo, se ha incluido en esta parte del capítulo, y junto al framework Spring, permite una integración de los modelos java dentro de nuestras plantillas front-end.

Todo esto, Thymeleaf lo consigue a través de una sintaxis propia, que unida al HTML estándar, permite la implementación de estructuras iterables que utilizan los datos sacados directamente del back-end. La sintaxis de Thymeleaf en el caso de nuestra aplicación será la estándar de Thymeleaf, que teniendo en cuenta que es una integración con Spring se podría utilizar la llamada “SpringStandard Dialect”, pero debido a la complejidad de la plantilla principal, se optó por la integración HTML.

Un ejemplo de la estructura que es capaz de lograr este framework, lo encontraríamos en la página principal de muestra de reservas de nuestra aplicación, cuyo aspecto es el mostrado en la figura 3.2.

Si nos fijamos, podemos observar diferentes etiquetas correspondientes con thymeleaf, como por ejemplo *“th:”* en cuyo caso, está introduciendo un elemento en el HTML que forma parte del framework que estamos explicando, asi como *“${…}"* o *“@{…}”* que introducirían objetos java y direcciones web respectivamente en nuestra plantilla. Volviendo al caso de la primera etiqueta, en la primera línea se puede encontrar *“th:fragment”*, los fragmentos de thymeleaf son una herramienta muy potente para la reutilización de código HTML, permitiendo crear patrones de código que se repite a lo largo de las peticiones, así como partes de la página dinámicas que podrán ser suministradas a peticiones Ajax si fuera necesario. Un ejemplo de esta reutilización de código, podría ser el caso de los Navbar de cualquier página web.



Figura 3.2 Plantilla Thymeleaf

### Bootstrap y Material Design

Una de las herramientas de código abierto más populares para desarrollo de front-end con HTML, CSS y JS, permite realizar rápidamente prototipos estructurales para la aplicación, adaptables a los distintos tamaños de pantalla y pantallas móviles, además provee de un extenso catálogo de componentes precompilados creados con la librería JQuery.

En nuestro caso estamos utilizando la última versión hasta la fecha, Bootstrap 4.3, que incluye todas las nuevas características de las nuevas versiones de CSS.

Respecto al uso, al igual que Thymeleaf, permite una implementación intuitiva de interfaces a través de los nombres de las clases HTML, de forma que, con tan solo un nombre, nos permite añadir ciertas características a nuestros elementos. Volviendo a la figura 3.2, encontramos nombres de clases pertenecientes a diferentes partes de esta librería, las cuales se dividen en 4 grandes grupos:

1. Layout: Principalmente encontramos los distintos tipos de contenedores para los elementos de nuestra web, y las características que estos ofrecen.
2. Content: Contiene los elementos básicos para la creación de una página cualquiera, como por ejemplo cabeceras, formularios, etc.
3. Components: Agrupa todos los componentes específicos de Bootstrap, con sus estilos y utilidades específicas, ya listos para su uso.
4. Utilities: En este último grupo, encontramos un conjunto de herramientas para realizar pequeños cambios de aspecto en elementos de nuestra página, tanto si son elementos estándar, como si son de grupos anteriores de la propia librería.

El conjunto de estos grupos, permiten crear interfaces que sean capaces de adaptarse al tamaño de pantalla de los dispositivos del cliente, intuitivos y rápidos.

En conjunto con Bootstrap, tenemos la librería Material Design, que sin cambiar nada de los implementado en la librería explicada con anterioridad, le profiere un aspecto más moderno y elaborado, añadiendo a los diferentes elementos creados con Bootstrap características visuales como serían sombras, animaciones, contorno, etc.

### Otros

A pesar de las dependencias para el desarrollo de front-end explicadas anteriormente, no puede faltar una mención a librerías como JQuery, que permite la implementación de código JS de forma increíblemente sencilla y Font-Awesome que provee de una gran cantidad de iconos para darle forma a nuestra interfaz y hacerla más comprensible para el usuario final, con tan solo añadir el nombre en la clase del elemento que quieras que tenga el icono, al igual que con Bootstrap.

# Creación de la aplicación

En este capítulo, vamos a describir los diferentes puntos que hemos seguido para la creación de la aplicación, desde cero hasta el primer prototipo funcional que será publicado en la dirección correspondiente, la metodología que hemos seguido y una descripción de los puntos más relevantes encontrados a lo largo del desarrollo del proyecto.

## Metodología

## Primeros pasos

En una fase temprana del proyecto, el objetivo es dejar claro las necesidades a cubrir especificadas por el cliente, para ello se realizará un conjunto de reuniones a lo largo del primer mes de implementación en las que el desarrollador le realizará preguntas al cliente.

### SPRINT 1

|  |  |
| --- | --- |
| **Planteamiento Inicial** | |
| Fecha | 11 de febrero |
| Objetivo | Aclaración de conceptos |
| Descripción | El cliente explica un concepto de su idea sobre la aplicación, la disponibilidad que esta debe tener, y los usuarios que tendrán acceso a esta.  Debido a las necesidades del cliente, el desarrollador decide que la aplicación será una aplicación web. |

Tabla 4.1 Sprint 1

### SPRINT 2

|  |  |
| --- | --- |
| **Servidor Back-End** | |
| Fecha | 25 de febrero |
| Objetivo | Tipo de Autenticación |
| Descripción | Una vez implementado un primer servidor inicial, con los repositorios y servicios correspondientes pero sin front-end, el desarrollador concierta una reunión en la que se barajarán las diferentes opciones de autenticación para la aplicación.  Debido a la necesidad de limitar el acceso a los miembros de la universidad, la autenticación a implementar será de tipo OAuth. |

Tabla 4.2 Sprint 2

### SPRINT 3

|  |  |
| --- | --- |
| **Google OAuth2** | |
| Fecha | 1 de marzo |
| Objetivo | Presentación del sistema de autenticación |
| Descripción | El desarrollador le muestra al cliente el sistema de autenticación disponible en la aplicación, y este da el visto bueno.  En una segunda parte de la reunión se plantea una primera idea de lo que sería el front-end de los horarios, utilizando una página ya existente como plantilla de lo que el cliente quiere.  El formato de lo que el cliente, es parecido a la página disponible en el siguiente enlace:  <http://reservasaulas.unileon.es/Web/view-schedule.php> |

Tabla 4.3 Sprint 3

## Fase de desarrollo

Durante todas las reuniones anteriores, el desarrollador consiguió sentar una base de lo que será la aplicación a lo largo de los próximos meses, teniendo en este punto, un servidor capaz de recibir peticiones de entrada, delegando el servidor de Google para la autenticación de los usuarios y redirigiendo a estos a una interfaz aún por implementar. A partir de ahora se realizan reuniones que influyen en lo que será el diseño de la interfaz del prototipo final y de su funcionamiento.

Esta fase se podría dividir en dos partes, una para la implementación del front-end y otra perteneciente únicamente a todo lo referido con la API de Google Calendar, cuya integración hará rediseñar al desarrollador el sistema de reservas.

### SPRINT 4

|  |  |
| --- | --- |
| **Primer Front-End** | |
| Fecha | 11 de marzo |
| Objetivo | Bases para la interfaz de usuario |
| Descripción | Cliente y desarrollador ponen en común una pila de ideas para poder implementar una interfaz para el primer prototipo. El desarrollador además, da el visto bueno al diseño de la página que el cliente le mostró en la reunión anterior y explica las limitaciones que cree que tiene.  Por lo tanto, el front-end terminará siendo algo parecido a la página de muestra con las adaptaciones pertinentes debido a las necesidades del cliente a lo largo del desarrollo. |

Tabla 4.4 Sprint 4

### SPRINT 5

|  |  |
| --- | --- |
| **Sistema de Reservas** | |
| Fecha | 18 de marzo |
| Objetivo | Pequeña demostración del prototipo |
| Descripción | El desarrollador le muestra al cliente una muestra del front-end disponible en el prototipo actualizado a la fecha de la reunión, así como un primitivo sistema de reservas de espacios que será la base del sistema de reservas definitivo. Si bien este crea reservas de espacios, no vuelva esa información en los calendarios de google de las asignaturas correspondientes. |

Tabla 4.5 Sprint 5

### SPRINT 6

|  |  |
| --- | --- |
| **Etapa Intermedia** | |
| Fecha | 25 de marzo |
| Objetivo | Definición de objetivos |
| Descripción | En esta reunión, ambos miembros ponen en común la situación en la que se encuentra el desarrollo actual de la aplicación. Se trata de una reunión informativa, para dejar claros conceptos que el cliente o el desarrollador tengan dubitativos. En ella surgen ideas como el sistema de popups para reservas y el cómo será la interfaz de edición de estas. |

Tabla 4.6 Sprint 6

### SPRINT 7

|  |  |
| --- | --- |
| **Reforma del sistema de modelos** | |
| Fecha | 1 de abril |
| Objetivo | Integración de Google Calendar |
| Descripción | En todas las reuniones anteriores, se trabajó sobre lo que sería un sistema primitivo de reservas en la propia aplicación con una interfaz limitada y cuyo objetivo era el de ser no más que una muestra para el cliente.  Durante esta reunión, surgió la idea de desarrollar una integración con la API de Google Calendar y los calendarios de cada una de las asignaturas disponibles en los grados y másteres de la universidad.  El cliente, tiene acceso a todas las cuentas y calendarios pertinentes asique el desarrollador decide que se podría llevar a cabo algo así, pero este cambio conlleva una refactorización de la mayor parte del proyecto, ya que afecta a la parte de back-end, modelos, servicios y repositorios, y para realizar el cambio se debe asegurar de que la aplicación mantiene la consistencia. |

Tabla 4.7 Sprint 7

### SPRINT 8

|  |  |
| --- | --- |
| **Reforma del sistema de IDs** | |
| Fecha | 2 de mayo |
| Objetivo | Actualización al cliente de los avances |
| Descripción | Tras un mes de implementación, el desarrollador tiene disponible un prototipo con integración con calendar para la reserva de espacios, pero con serias limitaciones que serán descubiertas durante esta reunión.  En ella, el cliente le muestra qué hay dentro de los calendarios de las asignaturas, para que el desarrollador sepa a lo que se enfrenta.  Los calendarios están disponibles en la página oficial de ingenierías. En ellos, se dispone de un conjunto de eventos, cuya cabecera y ubicación le muestran a los usuarios donde y cuando se realizarán las clases. Estos eventos, poseen una ID única definida por Google Calendar que permite la modificación y eliminación de estos.  Por lo tanto durante esta fase del desarrollo se buscará la modificación del sistema de IDs actual del prototipo para adaptarse a las IDs de Google Calendar. |

Tabla 4.8 Sprint 8

### SPRINT 9

|  |  |
| --- | --- |
| **Actualización Automática** | |
| Fecha | 20 de mayo |
| Objetivo | Actualización al cliente de los avances |
| Descripción | Una vez más, cliente y desarrollador se reúnen para poner una puesta en común de las capacidades del prototipo, pero esta vez, el desarrollador le trae al cliente una lista de posibles vías de enfoque para los siguientes pasos tras la investigación realizada en el último sprint.  Ambos deciden que el siguiente paso será la implementación de un sistema de actualización automática o manual para la aplicación, que reconstruirá por completo la pila de reservas que tiene la aplicación en el momento de la actualización, sacando la información pertinente de los calendarios de google que tengan disponibles las cuentas de servicio. |

Tabla 4.9 Sprint 9

### SPRINT 10

|  |  |
| --- | --- |
| **Corrección de reservas** | |
| Fecha | 27 de mayo |
| Objetivo | Muestra del sistema de actualización y próximo paso |
| Descripción | Junto a la reunión de integración de la API de Google Calendar, esta es una de las más importantes para el proyecto. En ella se decide que es posible realizar una implementación para la corrección de reservas ya existentes en los calendarios. Por lo tanto, el sistema además de importar todos los eventos que cumplan con las características predefinidas en esos calendarios, será capaz de importar aquellos que no las cumplan y ofrecer una posible corrección sobre la mayoría de ellos.  Un ejemplo de estas correcciones viene dado en los eventos de los exámenes que aún no tienen un aula asignada en la ubicación, y a través del prototipo el cliente podría aplicar una corrección. |

Tabla 4.10 Sprint 10

### SPRINT 11

|  |  |
| --- | --- |
| **Configuración** | |
| Fecha | 5 de junio |
| Objetivo | Crear la configuración necesaria para el prototipo |
| Descripción | Una vez finalizada toda la implementación, el desarrollador y el cliente se reúnen para poner a punto la configuración de la aplicación, que constará de un archivo en formato JSON, en el que se incluirán 3 listas.  En la primera, se encontrará listado todas las carreras y másteres que oferta la facultad de ingenierías, en cada una de estas entradas, habrá una lista de asignaturas con sus correspondientes nombres e IDs de los calendarios de Google a los que nuestras cuentas de servicio tienen acceso.  En la segunda, una lista en la que se encuentran todos los espacios mapeados por nuestra aplicación con su correspondiente nombre, numeración y número de plazas.  Por último, un pequeño listado de pequeñas configuraciones de inicio para la aplicación, en el cuál actualmente solo se encuentra la fecha desde la cual la aplicación realizará importaciones desde los calendarios, siendo esta el 1 de agosto de 2019. |

Tabla 4.11 Sprint 11

Además de las reuniones antes de cada uno de los sprints, se han realizado reuniones menores para resolver dudas, tanto por parte del cliente como por parte del desarrollador. En ellos se tomaron decisiones principalmente de aspecto o limitación de utilidad, así como para comprobar ciertos funcionamientos de la aplicación o para utilizar esta misma para corregir algunos horarios ya existentes con un prototipo no terminado.

Por otro lado, pese a que durante las pertinentes explicaciones de las reuniones llevadas a cabo, la figura del cliente se muestra completamente separada del equipo de desarrollo, esto no es más que una forma de expresión, ya que en la realidad, el cliente a formado parte en gran medida del propio equipo de desarrollo, siendo él quien ha dado permisos de edición en todos los calendarios a las cuentas de servicio, conseguido una cuenta perteneciente a la organización de la universidad para el desarrollo de este proyecto y tramitado los permisos pertinentes para la obtención de un servidor virtual para alojar la aplicación.

# Análisis del prototipo final

Tras toda la fase de desarrollo explicada anteriormente, nos encontramos con un prototipo funcional terminado. La idea de este proyecto, no es sino la de sentar unas bases para el posible desarrollo de una aplicación de reservas de espacios con funcionalidad completa, por lo tanto se procederá en este capítulo a realizar un análisis de la estructura de código que actualmente existe en el proyecto, analizando las principales partes de cada uno de los paquetes y clases que en ellos se localizan.

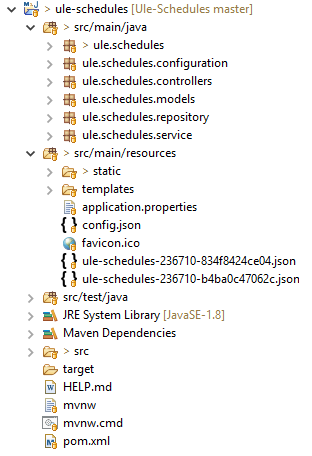
 Como bien se ha explicado en capítulos anteriores, se ha utilizado una versión actual de Spring Boot para el desarrollo de la aplicación, y Java Maven como herramienta de importación de dependencias, por lo tanto, estamos ante una estructura de aplicación web de modelo, vista y controlador.

Figura 5.1 Estructura del prototipo

En la estructura principal se pueden observar dos paquetes claramente diferenciados, uno para guardar todos los paquetes y código del back-end (src/main/java), y un segundo paquete para almacenar la parte del prototipo que pertenece al front-end (src/main/resources).

Además de la parte perteneciente al front-end, en la carpeta de resources, encontramos archivos de pura configuración del servidor Tomcat, que Spring utiliza en el momento de inicio de la aplicación.

Dentro de estos archivos de configuración, cabe destacar el archivo “*application.properties*”, el cuál almacena toda la configuración pertinente y necesaria para el servidor Tomcat, a través de comando de Spring.

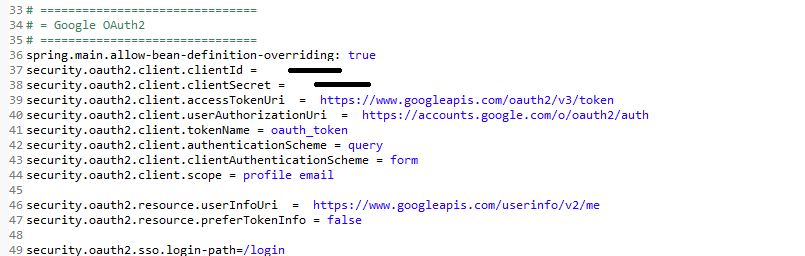


Figura 5.2 Configuración Google OAuth

En dicho archivo, además de configuraciones estándar que hacen referencia a la parte de base de datos, tipos de cifrado e Hibernate, encontramos una parte exclusivamente dedicada al servidor de autenticación de google, tal y como se muestra en la figura 5.2, en la cual se han omitido los campos pertinentes por razones de seguridad.

Por último, hablando de los archivos de configuración, se pueden observar dos archivos en formato JSON, que almacenarían los datos pertenecientes a las cuentas de servicio activadas a través de la consola de desarrollador de Google, y descargados directamente de esta para su posterior uso a la hora de crear las credenciales en el repositorio correspondiente al intercambio de información con Google Calendar.

## Estructura del Back-End

Para realizar el análisis del back-end del servidor, haremos una revisión de cada uno de los paquetes que lo forman, analizando las clases más relevantes dentro de estos. Dicha estructura MVC es obligatoria a causa del uso del framework, que busca dentro de la lista de paquetes, aquellos cuyo nombre sea configuration, controllers, models, repository y service. Dentro de estos analiza las clases individualmente en busca de anotaciones para la configuración del servidor y base de datos durante la fase de inicio de la aplicación, tal y como se observa en la figura 5.3.



Figura 5.3 Inicio de la aplicación

### Paquete básico

Dentro de este paquete genérico, encontramos dos clases, una dedicada al inicio de la aplicación y otra para rellenar las tablas de la base de datos en la que el dato más relevante que podemos encontrar es la primera anotación de Spring Framework, “*@Component*”. Las anotaciones sirven para especificarle a Spring Boot las funciones de los componentes asociándoles una responsabilidad en concreto. Estas anotaciones se llaman Spring Stereotypes, existiendo únicamente 4 tipos distintos de Stereotypes:

* @Component: Es un estereotipo general y permite anotar un bean para que Spring lo considere en su configuración.
* @Repository: Es el estereotipo encargado de dar de alta un bean que implemente el patrón repositorio encargado de intercambiar información con la base de datos.

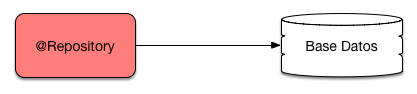


Figura 5.4 Esquema repositorio  
Fuente: <https://www.arquitecturajava.com/spring-stereotypes/>

* @Service: Este estereotipo es el encargado de gestionar el intercambio de información entre los controladores y los distintos repositorios. Agrupando llamadas desde un mismo servicio a uno o varios repositorios.

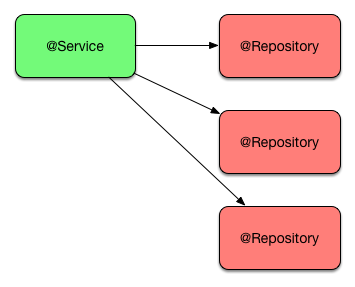


Figura 5.5 Esquema servicio  
Fuente: <https://www.arquitecturajava.com/spring-stereotypes/>

* @Controller: El cuarto estereotipo, es el encargado de realizar las tareas de comunicación entre cliente y aplicación, para ello, en caso de nuestra aplicación se utiliza el Framework de plantillas Thymeleaf para facilitar la implementación del front-end.

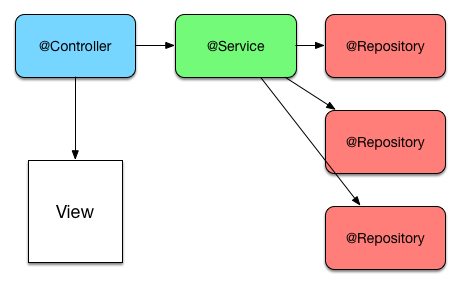


Figura 5.6 Esquema controlador   
Fuente: <https://www.arquitecturajava.com/spring-stereotypes/>

### Paquete de configuración

En este paquete, únicamente encontramos una clase java dedicada a la capa de seguridad de nuestro servidor web, en ella tal y como se puede observar en la figura 5.7, se limita el acceso a todos aquellos usuarios que no estén identificados más allá de la página de bienvenida, liberando por otra parte componentes como las librerías y dependencias de terceros utilizadas en el front-end, las cuáles se han explicado en un [capítulo](#_Cliente_Front-End) anterior.

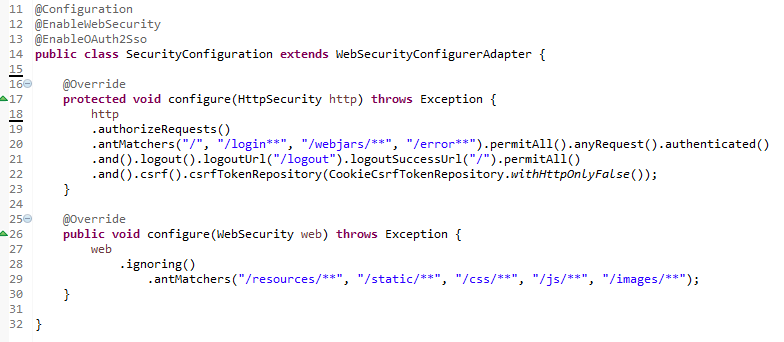


Figura 5.7 Configuración de seguridad

Además de esta configuración, en la clase SecurityConfiguration, encontramos tres nuevas anotaciones no pertenecientes a Spring Stereotypes como serían @Configuration, @EnableWebSecurity y @EnableOAuth2Sso. Por orden de aparición, estas anotaciones nos especifican lo siguiente:

* @Configuration: Le específica a Spring que la clase a continuación está dedicada a configuración, y será del tipo de la clase de la cual herede, en este caso WebSecurityConfigurerAdapter.
* @EnableWebSecurity: Aplica las configuraciones de seguridad implementadas en los métodos que contiene la clase como tal.
* @EnableOAuth2Sso: Activa la autenticación de Google en este caso, sacando la configuración del archivo “application.properties” explicado anteriormente en este mismo capítulo y redireccionando al servicio pertinente en caso de acceder a una de las páginas restringidas de la aplicación.

### Paquete de controladores

Para explicar el paquete controladores, debemos explicar para que y como se utilizan las clases que este contiene. Por un lado nos encontramos con un controlador cuya anotación sí está en Spring Stereotypes, como sería el caso de la clase FragmentController.

Esta clase está dedicada exclusivamente a la actualización dinámica de partes del front-end gracias a los fragmentos de Thymeleaf y a peticiones Ajax. Si bien podríamos explicar cada uno de los métodos que esta clase contiene individualmente, lo mejor es utilizar un ejemplo de uno de ellos.

Al entrar a la página principal por primera vez, nos encontramos con la interfaz de usuario mostrada en la siguiente figura.

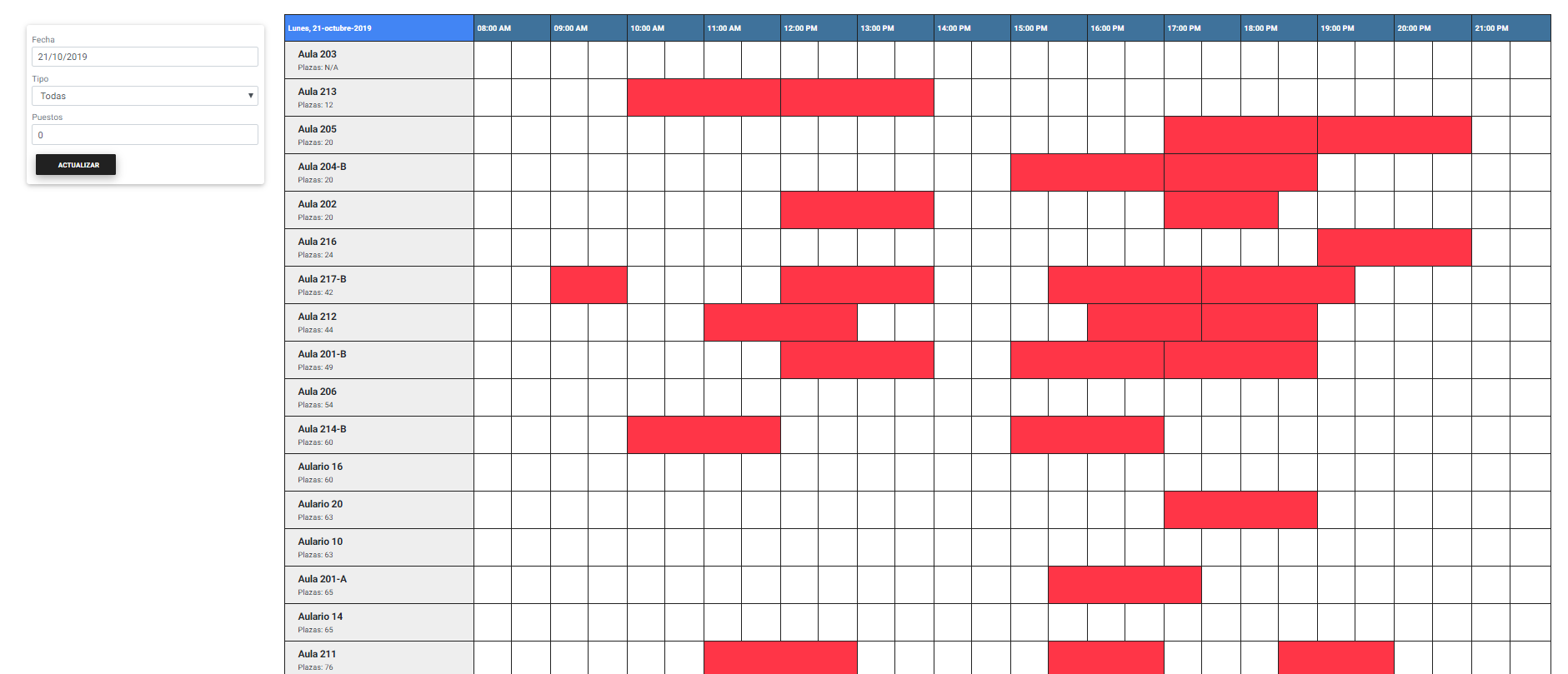


Figura 5.8 Página principal

En la figura, se han omitido las partes irrelevantes que no tienen que ver con esta explicación para que el ejemplo sea más clarificador. Tal y como se observa, existen dos partes claramente diferenciadas, una tabla con las reservas pertinentes de un día elegido al azar y una pequeña lista de opciones a su izquierda. En el momento de cambiar esas opciones y hacer click en el botón “Actualizar”, la tabla de la izquierda cambiará sus datos, sin que se actualice en ningún momento el navegador del cliente. Este proceso consta de dos partes, una relacionada con el front-end que será explicada en esta [parte](#_Estructura_del_Back-End) de este mismo capítulo, y una parte de back-end, la cual se realiza en la clase que estamos explicando. FragmentController recibe esa petición de actualización, y gracias a los fragmentos de Thymeleaf, construye únicamente la parte del template que sea necesario refrescar, en este caso, la tabla de reservas, exactamente reconstruye y reenvía el código de la figura 3.2. Este proceso se repite con todas aquellas partes que deban ser actualizadas en el front-end, bien sean datos de reservas individuales o tablas completas.

Por otro lado, y volviendo al análisis del paquete en el que nos encontramos, tenemos una segunda clase llamada SchedulesController. En este caso sin embargo, no se trata de una clase cuya anotación pertenece a Spring Stereotypes, sino que es un tipo de anotación no estándar, @RestController. Este controlador, además de contener las recepciones de las peticiones GET para las páginas principales en nuestra aplicación, contiene peticiones que trabajan con estructuras de datos en formato JSON, que es el formato de trabajo de clásico para este tipo de controladores. En él, se administra la creación, modificación, corrección y eliminación de reservas, delegando a los servicios o repositorios que correspondan en cada caso.

### Paquete de modelos

En este paquete encontramos todos los modelos de datos utilizados durante la implementación de la aplicación, en cada uno de ellos se han incluido anotaciones pertenecientes a Spring, sirviendo estas para que sea el propio framework quien construya y pueda realizar intercambios con la base de datos y no nosotros manualmente.

Esta sería una de las ventajas de la utilización de JPA Repository en nuestra aplicación Spring Boot, permitiéndonos manipular la base de datos a través de objetos Java conocidos como entidades. Estas entidades no son más que clases comunes y corrientes a las que también se les llama POJO’s (Plain Old Java Objetcs), con la particularidad de que están mapeadas en cada una de las tablas de la base de datos que se le da a conocer a nuestra aplicación en el archivo application.properties. Dicho mapeo, se realiza mediante las anotaciones explicadas con anterioridad, que nos brindan suficiente información como para poder relacionar las clases contra las tablas y las propiedades de estas clases contra las columnas de las tablas de la base de datos.

Así bien, para dar a conocer estas clases Java y distinguirlas de cualquier otra, utilizaremos la anotación @Entity, que única y exclusivamente está dedicada a convertir una clase Java estándar en entidad. Sin embargo, esta anotación no consigue que JPA almacene la información de nuestra entidad en la base de datos, por lo que debemos utilizar una nueva anotación (@Table), en la que además especificamos el nombre que va a tomar la tabla que refleje los datos de nuestra entidad en la base datos, así como indicarle a JPA el tipo de índices que tienen que tener nuestras propiedades dentro de las columnas de la base de datos.

En nuestro caso, y como última aportación a este pequeño análisis, las entidades de Reserv, User y Subject sacan sus IDs de la API de Google correspondiente, por lo que nuestra aplicación no tiene que generarlas automáticamente como en el caso del resto de entidades, pero sí asegurarse de que estas IDs sean únicas dentro de su tabla.

### Paquete de repositorios

### Paquete de servicios

## Estructura del Back-End

### Paquete de plantillas

### Paquete de archivos estáticos

Conclusión

Expresión personal del conjunto de conclusiones que, a juicio del autor, se derivan de los resultados expuestos en el trabajo. Deberá tener una extensión entre cinco y diez páginas.

Aportaciones realizadas

Trabajos futuros

Problemas encontrados

## Opiniones personales

# Lista de referencias

Información bibliográfica citada en el texto del trabajo. Otras lecturas recomendadas o consultadas, de figurar, aparecerán en anexos.

Se debe seguir la norma ISO 690 (buscar en google ISO 690 ugr)

ANEXO A: Control de Versiones

**Obligatorio.** Seguimiento del trabajo real.

### Forma de seguimiento

### Planificación inicial

### Planificación final

ANEXO B: Manual de usuario