UNIVERSIDAD SAN PABLO - CEU

ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA DE SISTEMAS DE INFORMACIÓN



TRABAJO FIN DE GRADO l

**Diseño e Implementación de una aplicación RESTful para la gestión de reservas de puestos bibliotecarios**

**Design and implementation of a RESTful application to manage the reservations of library seats**

Autor: D. Fernando Ortiz de Pedro

Tutor: D. Sergio Saugar García

Junio 2023

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Datos del alumno   |  | | --- | | Nombre: Fernando Ortiz de Pedro |   Datos del Trabajo   |  | | --- | | TÍTULO DEL PROYECTO:  Diseño e Implementación de una aplicación RESTful para la gestión de reservas de puestos bibliotecarios |   Tribunal calificador   |  |  | | --- | --- | | Presidente: | Fdo.: |  |  |  | | --- | --- | | Secretario: | Fdo.: |  |  |  | | --- | --- | | Vocal: | Fdo.: |  |  | | --- | | Reunido este tribunal el \_\_\_ /\_\_\_ /\_\_\_\_\_\_\_, acuerda otorgar al Trabajo Fin de Grado presentado por D./Dña. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ la calificación de \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | |

Resumen

Debido a la pandemia del coronavirus que se inició en el año 2020 y las restricciones tomadas por las instituciones para evitar su propagación, muchas situaciones del día a día se han visto afectadas en todo el mundo. La Universidad San Pablo CEU tomó las medidas necesarias impuestas por las instituciones públicas, de las cuales se destacan las relacionadas con las bibliotecas: control de aforo, ventilación de las salas de estudio, separación entre puestos de estudio con la distancia mínima interpersonal, suspensión del sistema de préstamo de libros, etcétera. Hoy en día, la mayoría de esas medidas ya no existen debido a que la situación es menos preocupante, pero hay otras que, o bien permanecen obligatorias como el uso de mascarillas en transporte público, o se han mantenido voluntariamente para paliar otro tipo de problemas. Es en esta segunda situación en la cual surge el tema de este TFG. La Universidad San Pablo CEU ha decidido mantener un control de acceso a las bibliotecas por seguridad, y por ello se desarrolla en este TFG un Servicio Web *RESTful* de seguimiento.

Palabras Clave

Servicio Web, RESTful, biblioteca, puesto de estudio, sala, reserva, recurso, disponibilidad, elemento reservable.

Abstract

Resumen anterior, en inglés.

Keywords

Índice de contenidos

Contenido

[Capítulo 1 Introducción 13](#_Toc120718549)

[1.1 Objetivos 14](#_Toc120718550)

[Capítulo 2 Gestión del proyecto 15](#_Toc120718551)

[2.1 Modelo de ciclo de vida 15](#_Toc120718552)

[2.2 Papeles desempeñados en el proyecto 16](#_Toc120718553)

[2.3 Planificación 17](#_Toc120718554)

[2.4 Presupuesto 18](#_Toc120718555)

[2.5 Ejecución 18](#_Toc120718556)

[2.6 Recursos utilizados 20](#_Toc120718557)

[*2.6.1* *Hardware* 20](#_Toc120718558)

[*2.6.2* *Software* 20](#_Toc120718559)

[Capítulo 3](#_Toc120718560) [Estado del Arte 22](#_Toc120718561)

[3.1 Servicios Web 22](#_Toc120718562)

[3.1.1 Ventajas y desventajas de Servicios Web 24](#_Toc120718563)

[3.1.2 Arquitectura Orientada a Servicios (SOA) 24](#_Toc120718564)

[3.1.3 Arquitectura Orientada a Recursos (ROA) 27](#_Toc120718565)

[3.2 Tecnologías 34](#_Toc120718566)

[3.2.1 BackEnd 35](#_Toc120718567)

[3.2.2 FrontEnd 35](#_Toc120718568)

[Capítulo 4 Análisis 39](#_Toc120718569)

[4.1 Análisis del dominio de la aplicación 39](#_Toc120718570)

[4.1.1 Solución informal primigenia 41](#_Toc120718571)

[4.1.2 Solución de software *Affluences* 43](#_Toc120718572)

[4.2 Actores 47](#_Toc120718573)

[4.3 Recursos 48](#_Toc120718574)

[4.3.1 Puestos de estudio 48](#_Toc120718575)

[4.3.2 Salas de trabajos 49](#_Toc120718576)

[4.3.3 Libros 49](#_Toc120718577)

[4.3.4 Estanterías 50](#_Toc120718578)

[4.3.5 Ordenadores 50](#_Toc120718579)

[4.3.6 Enchufes 50](#_Toc120718580)

[4.3.7 Ventanas 51](#_Toc120718581)

[4.3.8 Aire acondicionado 51](#_Toc120718582)

[4.4 Procesos 52](#_Toc120718583)

[4.4.1 Reserva en biblioteca 52](#_Toc120718584)

[4.4.2 Cancelar reserva 53](#_Toc120718585)

[4.4.3 Cambio en reserva 53](#_Toc120718586)

[4.4.4 Reserva de sala 53](#_Toc120718587)

[4.4.5 Reserva de puesto 54](#_Toc120718588)

[4.4.6 Reserva de recurso 54](#_Toc120718589)

[4.4.7 Reserva de recurso libro 54](#_Toc120718590)

[4.4.8 Reserva de recurso ordenador 55](#_Toc120718591)

[4.4.9 Casos particulares 55](#_Toc120718592)

[4.5 Especificación de requisitos 55](#_Toc120718593)

[4.5.1 Requisitos funcionales 56](#_Toc120718594)

[4.5.2 Requisitos no funcionales 57](#_Toc120718595)

[4.6 Análisis de seguridad 57](#_Toc120718596)

[4.7 Análisis desde la perspectiva del RGPD 58](#_Toc120718597)

[Capítulo 5 Arquitectura y diseño 59](#_Toc120718598)

[5.1 Arquitectura del sistema 59](#_Toc120718599)

[5.2 Diseño del BackEnd 60](#_Toc120718600)

[5.2.1 Definición de recursos 60](#_Toc120718601)

[5.2.2 Identificación de los recursos Web 61](#_Toc120718603)

[5.2.3 Representaciones utilizadas 63](#_Toc120718604)

[5.2.4 Establecimiento de interfaz 63](#_Toc120718605)

[5.2.5 Base de Datos 70](#_Toc120718606)

[5.3 Diseño del FrontEnd 72](#_Toc120718607)

[Capítulo 6 Implementación y pruebas 73](#_Toc120718608)

[6.1 Implementación *FrontEnd* 73](#_Toc120718609)

[6.2 Implementación *BackEnd* 74](#_Toc120718610)

[6.2.1 Entities 74](#_Toc120718611)

[6.2.2 Services 74](#_Toc120718612)

[6.2.3 Routes 75](#_Toc120718613)

[6.2.4 Controllers 75](#_Toc120718614)

[6.3 Se indicará el IDE, *frameworks*, simuladores, etc. que se han utilizado. 75](#_Toc120718615)

[6.4 Pruebas 75](#_Toc120718616)

[Capítulo 7 Conclusiones y líneas futuras 77](#_Toc120718617)

[7.1 Líneas futuras 78](#_Toc120718618)

[Bibliografía 81](#_Toc120718619)

[Anexo I 83](#_Toc120718620)

[Caso de uso de la aplicación Affluences desde el punto de vista del alumnado 83](#_Toc120718621)

[Anexo II 86](#_Toc120718622)

[Caso de uso de la aplicación Affluences desde el punto de vista del responsable de la biblioteca 86](#_Toc120718623)

[Anexo III 88](#_Toc120718624)

[Tablas API de recursos 88](#_Toc120718625)

Índice de ilustraciones

[Ilustración 1: Fases de la metodología waterfall (S. Pressman, 1994) 15](https://ceu365-my.sharepoint.com/personal/f_ortiz4_usp_ceu_es/Documents/TFG%20ISI/TFG-FernandoOrtiz_2022.docx#_Toc107411892)

[Ilustración 2: Diagrama Gantt de planificación (elaboración propia). Reparto de tareas. 17](https://ceu365-my.sharepoint.com/personal/f_ortiz4_usp_ceu_es/Documents/TFG%20ISI/TFG-FernandoOrtiz_2022.docx#_Toc107411893)

[Ilustración 3: Diagrama Gantt de planificación (elaboración propia). Tareas organizadas en cascada 17](https://ceu365-my.sharepoint.com/personal/f_ortiz4_usp_ceu_es/Documents/TFG%20ISI/TFG-FernandoOrtiz_2022.docx#_Toc107411894)

[Ilustración 4: : Diagrama Gantt de realización de tareas (elaboración propia). (a) Tareas realizadas durante la primera parte del proyecto. (b) Tareas realizadas durante la segunda parte del proyecto 19](https://ceu365-my.sharepoint.com/personal/f_ortiz4_usp_ceu_es/Documents/TFG%20ISI/TFG-FernandoOrtiz_2022.docx#_Toc107411895)

[Ilustración 5: Despliegue de un Servicio Web (Jácome Galarza, 2010) 23](https://ceu365-my.sharepoint.com/personal/f_ortiz4_usp_ceu_es/Documents/TFG%20ISI/TFG-FernandoOrtiz_2022.docx#_Toc107411896)

[Ilustración 6: Diagrama de una Arquitectura Orientada a Servicios (SOA) (V. Issarny et al., 2011 25](https://ceu365-my.sharepoint.com/personal/f_ortiz4_usp_ceu_es/Documents/TFG%20ISI/TFG-FernandoOrtiz_2022.docx#_Toc107411897)

[Ilustración 7: Detalle tecnológico de una Arquitectura Orientada a Servicios (SOA). (Issarny et al., 2011) 26](https://ceu365-my.sharepoint.com/personal/f_ortiz4_usp_ceu_es/Documents/TFG%20ISI/TFG-FernandoOrtiz_2022.docx#_Toc107411898)

[Ilustración 8: Diagrama de una Arquitectura Orientada a Servicios (SOA) (Gitonga, 2021) 27](https://ceu365-my.sharepoint.com/personal/f_ortiz4_usp_ceu_es/Documents/TFG%20ISI/TFG-FernandoOrtiz_2022.docx#_Toc107411899)

[Ilustración 9: Ejemplo nivel 0 Modelo de Madurez de Richardson (Fowler, 2010) 31](https://ceu365-my.sharepoint.com/personal/f_ortiz4_usp_ceu_es/Documents/TFG%20ISI/TFG-FernandoOrtiz_2022.docx#_Toc107411900)

[Ilustración 10: Diagrama de casos de uso. Elaboración propia 52](https://ceu365-my.sharepoint.com/personal/f_ortiz4_usp_ceu_es/Documents/TFG%20ISI/TFG-FernandoOrtiz_2022.docx#_Toc107411901)

[Ilustración 11: Tabla de requisitos funcionales. Elaboración propia 56](https://ceu365-my.sharepoint.com/personal/f_ortiz4_usp_ceu_es/Documents/TFG%20ISI/TFG-FernandoOrtiz_2022.docx#_Toc107411902)

[Ilustración 12: Tabla de requisitos no funcionales. Elaboración propia 56](https://ceu365-my.sharepoint.com/personal/f_ortiz4_usp_ceu_es/Documents/TFG%20ISI/TFG-FernandoOrtiz_2022.docx#_Toc107411903)

[Ilustración 13: Diagrama de la arquitectura del sistema. Elaboración propia 59](https://ceu365-my.sharepoint.com/personal/f_ortiz4_usp_ceu_es/Documents/TFG%20ISI/TFG-FernandoOrtiz_2022.docx#_Toc107411904)

Índice de tablas

[Tabla 1. Recurso usuarios 63](#_Toc107231795)

[Tabla 2. Recurso biblioteca 65](#_Toc107231796)

[Tabla 3. Recurso reserva 68](#_Toc107231797)

[Tabla 4. Recurso usuario 88](#_Toc107231798)

[Tabla 5. Recurso bibliotecas 90](#_Toc107231799)

[Tabla 6. Recurso puestos 92](#_Toc107231800)

[Tabla 7. Recurso puesto 94](#_Toc107231801)

[Tabla 8. Recurso salas 96](#_Toc107231802)

[Tabla 9. Recurso sala 98](#_Toc107231803)

[Tabla 10. Recurso reservas 101](#_Toc107231804)

[Tabla 11. Recurso recursos de puesto 103](#_Toc107231805)

[Tabla 12. Recursos recurso de puesto 105](#_Toc107231806)

[Tabla 13. Recurso recursos de sala 107](#_Toc107231807)

[Tabla 14. Recurso recurso de sala 109](#_Toc107231808)

# Introducción

La pandemia mundial que surgió en el año dos mil veinte ha provocado, y sigue provocando, cambios en todo tipo de aspectos en nuestras vidas y a diferentes escalas. Respecto a la universidad, se han tomado medidas de protección para alumnos y trabajadores frente al Covid-19: distancia de seguridad en la cafetería, control de temperatura corporal al entrar a las facultades, o mayor control en las bibliotecas. De ahí surge este proyecto en cuestión. Debido a la necesidad de tener un mayor control, el personal de la biblioteca comenzó a utilizar una hoja de cálculo para reservar los sitios, un método poco eficiente.

En este Trabajo de Fin de Grado (TFG) se desarrolla un prototipo de un sistema de gestión de reservas para la biblioteca de la Escuela Politécnica Superior (EPS) de la Universidad San Pablo CEU (USP CEU). Para cumplir este objetivo, se analizará el problema desde dos puntos de vista de usuario: alumnado y personal de la biblioteca. Esta aplicación no sólo ayuda a evitar la propagación del virus, sino que también soluciona otros problemas recurrentes en la gestión de las reservas de sitios en la biblioteca. Por ejemplo, hasta antes de la pandemia cualquier alumno que quisiera ir a estudiar a la biblioteca tenía que ir hasta la universidad y comprobar si había algún sitio libre, sin saber a ciencia cierta si lo iba a encontrar o no. Además, aunque este prototipo no tiene como objetivo controlar al alumnado, la trazabilidad de los usuarios de la biblioteca podría ser útil para evitar robos o localizar objetos perdidos. Por tanto, se podría decir que antes de la pandemia estas situaciones eran una necesidad por cubrir y que, en cierta forma gracias ella, se han visto solventadas con medidas como las de este proyecto.

Cabe destacar que este proyecto se podría replicar para las otras dos bibliotecas que dispone la universidad o incluso, se podría extrapolar y aplicar a otras situaciones donde exista la necesidad de control de aforo y reserva de recursos: acceso a laboratorios, cafetería, pistas deportivas, aulas de tutorías, servicio de autobuses, …

## Objetivos

Como objetivo global, este proyecto planea desarrollar un prototipo para satisfacer las necesidades con respecto a la reserva de sitios en la biblioteca de la EPS. Este programa permitirá que los alumnos puedan reservar de forma autónoma, así como posibilitar al personal de la biblioteca el acceso a dichas reservas y estadísticas de aforo. Así, se podrá controlar el aforo de la estancia y garantizar los protocolos de actuación COVID que establecen las autoridades competentes. Los objetivos concretos que se derivan de la problemática del dominio, analizada detalladamente, y los requisitos tanto funcionales como no funcionales a los que da cobertura el *software* desarrollado, son los siguientes:

* Realizar un prototipo que permita la creación de reservas de sitios en la biblioteca de la EPS para gestionar el aforo y los recursos.
* Diseñar un Servicio Web *RESTful* que permita la creación de una reserva en la biblioteca, además de altas, bajas y modificaciones de esta.
* Creación de representaciones *JSON* para los recursos del sistema, para permitir la integración del servicio de otras aplicaciones (escritorio, aplicaciones Web, o aplicaciones móviles).

Fuera del ámbito de este TFG, el objetivo final de la aplicación es su implementación en el sistema de información de la universidad (intranet, portal del alumno). Así, se podrán aprovechar todos los datos del sistema para poder mejorar la biblioteca a nivel sanitario, del servicio, y de la seguridad.

El trabajo realizado durante este TFG se presenta en esta memoria estructurado de la siguiente manera: en el capítulo 2, se estudia la gestión del proyecto que engloba la metodología aplicada, la planificación seguida, y los recursos utilizados. A continuación, el capítulo 3 presenta el estado del arte y el capítulo 4 un análisis exhaustivo para entender la situación inicial en la que surge el proyecto, y también explica cómo el prototipo desarrollado soluciona dicha situación. Los siguientes capítulos de diseño y arquitectura (capítulo 5) e implementación (capítulo 6) se centran en el prototipo desarrollado en este trabajo. Finalmente se exponen las conclusiones y líneas futuras del trabajo en el capítulo 7.

# Gestión del proyecto

En este capítulo se estudiará la planificación del proyecto, explicando el tipo de metodología que se ha llevado a cabo, las diferentes etapas realizadas, los tiempos invertidos en ellas, y los recursos utilizados.

## Modelo de ciclo de vida

Para realizar este proyecto se ha utilizado una metodología tradicional, como es el caso de la metodología *waterfall* o cascada. Este modelo de ciclo de vida clásico “sugiere un enfoque sistemático y secuencial para el desarrollo de *software*, que comienza con la especificación de los requerimientos por parte del cliente y avanza a través de planeación, modelado, construcción y despliegue, para concluir con el apoyo del *software* terminado”. (S. Pressman, 1994)

Es una metodología adecuada por las características del proyecto: está acotado tanto en requisitos como en tiempo, los requisitos a priori están bien definidos y no van a cambiar a lo largo del proyecto. El número de participantes en el proyecto también es fijo, por lo que no es necesario valorar otras metodologías alternativas.

Diagrama

Descripción generada automáticamente con confianza mediaEsta metodología se compone de varias fases que se ejecutan de manera secuencial en el siguiente orden:

Ilustración : Fases de la metodología waterfall (S. Pressman, 1994)

* **Comunicación:** se evalúa la viabilidad, se recoge información, se capturan requisitos, y se hace un análisis de estos.
* **Planificación:** se refiere a la gestión de proyecto. En este apartado se planifican las etapas del trabajo, su duración, y se tiene una visión global para conseguir el objetivo del proyecto.
* **Modelado:** a partir del análisis, se analiza la funcionalidad del proyecto, se estudia qué arquitectura *software* usar, y se diseña la relación entre los componentes y los procesos.
* **Construcción:** se centra en la arquitectura de *software*, precisando las interfaces o entornos y de la aplicación. Se programa el *software*, se hacen pruebas para buscar posibles errores, y solventarlos.
* **Despliegue:** consiste en integrar el *software* desarrollado en los sistemas, y realizar más pruebas de testeo. También, se entrega el *software*, se realiza el mantenimiento ante posibles imprevistos, y realización de cambios o mejoras.

El hecho de no pasar a la siguiente fase hasta no haber acabado la actual, facilita el centrarse en la fase actual sin preocuparse por lo demás, tener más control, y adaptarse a las circunstancias que puedan surgir. (S. Pressman, 1994)

## Papeles desempeñados en el proyecto

Este proyecto ha sido realizado por D. Fernando Ortiz de Pedro, estudiante del doble grado en Ingeniería de Sistemas de la Información y Administración y Dirección de Empresas en la Universidad CEU San Pablo. Ha estado guiado por su tutor y *dueño del producto* D. Sergio Saugar García, con quién ha tenido reuniones continuas durante todo el proceso de realización del proyecto, así como del desarrollo del prototipo presentado más adelante.

Además, hay otros actores que han sido de gran ayuda para la elaboración del trabajo. Para entender las necesidades reales tanto de alumnos como de trabajadores y por tanto que el prototipo fuera realista, se realizaron una serie de entrevistas tanto a unos como a otros al ser los principales usuarios del sistema.

## Planificación

Como se ha mencionado anteriormente, en este proyecto se ha utilizado la metodología *waterfall*. Para realizar las etapas de este proyecto con dicha metodología, se han estimado los tiempos empleados en cada una de ellas en el *software* *GanttProject*, y se muestran a continuación:

Ilustración : Diagrama Gantt de planificación (elaboración propia). Reparto de tareas.

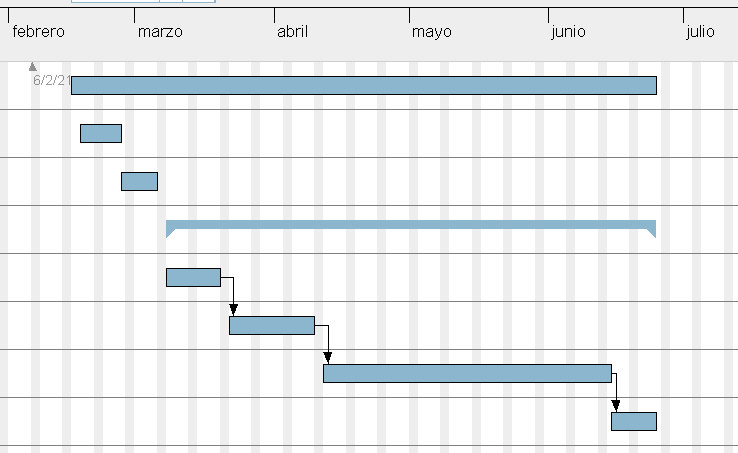


Ilustración : Diagrama Gantt de planificación (elaboración propia). Tareas organizadas en cascada

En el apartado 2.6, se comentarán los recursos utilizados para realizar esas tareas.

## Presupuesto

El prototipo desarrollado en este trabajo sería de gran ayuda si se quisiera realizar e implementar una versión final. Además, USP CEU tiene un departamento de informática que se encargaría del desarrollo. Por tanto, en este caso se podría decir que el presupuesto estaría incluido en los propios de la universidad. Además, el *software* utilizado es libre, por lo que no habría ningún pago de licencias.

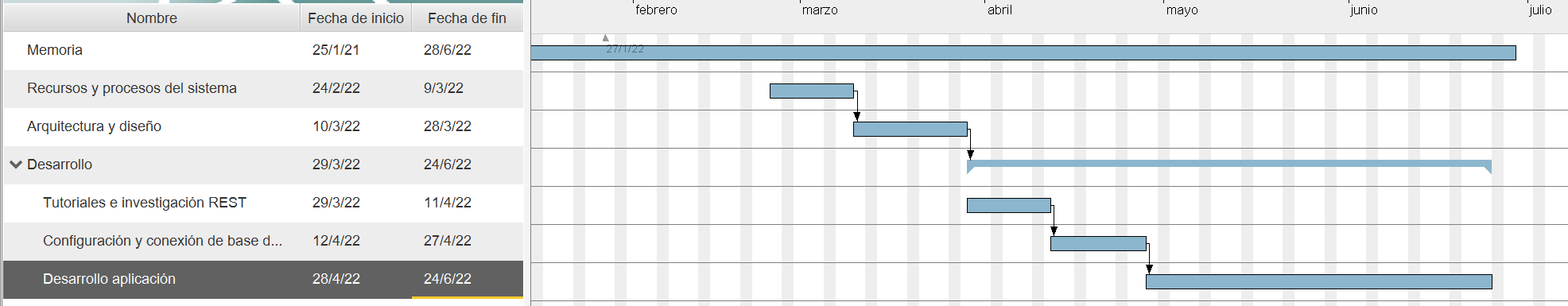
## Ejecución

Siguiendo las etapas de la metodología *waterfall*, se han llevado a cabo las siguientes tareas para la elaboración de este trabajo:

* Análisis del dominio de la aplicación
  + Estudio de la situación de partida y la necesidad de una solución ante los problemas logísticos de la reserva de sitios en la biblioteca
  + Análisis de la solución informatizada utilizada por la universidad
  + Realización de encuestas a alumnado y personal de la biblioteca para mejorar la solución propuesta por la universidad
  + Elaboración de los requisitos del prototipo gracias a la información recabada anteriormente
* Estudio del estado del arte para comprender la tecnología utilizada en el proyecto
* Definición de la arquitectura utilizada en el prototipo para cumplir con los requisitos planteados en fases anteriores
* Diseño del prototipo a nivel de arquitectura
* Implementación del *software* y prueba de errores
* Verificación del prototipo mediante testeos que permitan validar su integración
* Redacción de la memoria del proyecto:
  + Introducción del proyecto explicando el origen de este y sus objetivos
  + Realización del apartado de gestión de proyecto precisando recursos utilizados, diagramas de tiempos, papeles desempeñados, …
  + Elaboración tanto de la sección de análisis, como de diseño e implementación
  + Conclusión del trabajo y líneas futuras
  + Realización de bibliografía y anexos.

Gráfico

Descripción generada automáticamenteComo se ha explicado antes para la planificación del proyecto, el programa GanttProject permite representar visualmente el tiempo invertido en cada una de esas etapas. A continuación, se muestran dos diagramas: uno representa la primera etapa de la realización de este proyecto, que empieza a principios del año 2021 y dura hasta finales de ese curso académico; el segundo, empieza a principios de 2022 y acaba en la finalización del proyecto.

*(a)*

*(b)*

Ilustración 4: Diagrama Gantt de realización de tareas (elaboración propia). (a) Tareas realizadas durante la primera parte del proyecto. (b) Tareas realizadas durante la segunda parte del proyecto

## Recursos utilizados

En este apartado se precisarán todos los recursos utilizados para la realización de este proyecto, tanto a nivel de hardware, como a nivel *software*.

### *Hardware*

#### Ordenador personal

En este dispositivo se han desarrollado todas las tareas del proyecto, tanto de desarrollo de la aplicación web, como de redacción de la memoria y de creación de la presentación del trabajo. Consta de una arquitectura de 64 bits, un procesador Intel(R) Core (TM) i7-10510U y 16GB de memoria RAM.

### *Software*

#### Paquete Office de Microsoft

Las herramientas de ofimática de *Microsoft* han permitido realizar parte del proyecto. Se ha hecho esta documentación en *Word*, y la presentación en *PowerPoint*. Además, se ha utilizado el programa *Excel* para extraer datos a la hora de analizar el programa *Affluences* y su cuenta administrador.

#### GitHub

Se ha utilizado la herramienta *GitHub* para tener una mejor gestión del proyecto usando su sistema de control de versiones. Gracias a ello, se puede tener un seguimiento de los diferentes cambios que se han llevado a cabo tanto a nivel de la memoria como del software del prototipo desarrollado.

#### Play Framework

*Play Framework* permite construir aplicaciones web tanto en el lenguaje de programación *Java* como en *Scala*. En este caso, se ha utilizado *Java* al ser uno de los lenguajes principales del grado.

#### MySQL

Se ha recurrido al gestor de base de datos *MySQL* para almacenar los datos necesarios en el desarrollo y ejecución del prototipo.

#### IntelliJ IDE

Este entorno se ha utilizado para realizar todo desarrollo *software* realizado durante el proyecto.

#### GanttProject

Se ha utilizado este programa para la planificación del proyecto y gestión de recursos, mediante la realización de diagramas de *Gantt*.

#### AWS

En este proyecto se ha utilizado Amazon Web Services (AWS) para el despliegue del prototipo.

# 

# Estado del Arte

Hoy en día, Internet es un elemento imprescindible en nuestras vidas. Su invención y el hecho de hacerse cada vez más popular, marcó un antes y un después en la sociedad. Actualmente la gran mayoría de los productos o servicios que utilizamos, requieren del uso de Internet: contactar con alguien, pedir comida a domicilio, reservar en un restaurante, …

Actualmente, se estima que el 60% de la población mundial tiene acceso a Internet. Para medir este dato se tiene en cuenta estos dos factores: la cantidad de personas conectadas, y el porcentaje de población que tiene acceso a Internet con respecto al total. No sólo hay más personas conectadas en el presente, sino que hemos pasado de transferir datos del orden de bytes, a datos del orden de Gigabytes o incluso Terabytes. Todo esto se debe a la constante evolución de la infraestructura y a la facilidad con la que se pueden desarrollar programas, aplicaciones y Webs hoy en día: nuevas arquitecturas de programación y desarrollo Web, uso de *frameworks*, reutilización de código, o la capacidad de mezclar lenguajes. (El orden mundial, 2021)

A continuación, se analiza desde un punto de vista más teórico los tipos de Servicios Webs que existen. Después, se argumenta cuál se usa en este trabajo y por qué, así como la tecnología que hay detrás de este proyecto.

## Servicios Web

Un Servicio Web es un componente software con el que se interactúa a través de tecnologías propias de la Web (por ejemplo, *HTTP*, *HTML*, *JSON*, …) y es accesible desde cualquier parte de Internet, proporcionando así el intercambio de información entre las aplicaciones utilizando un patrón de comunicación cliente/servidor. El *World Wide Web Consortium* (*WC3*) define un servicio Web como “un sistema *software* diseñado para soportar interacciones máquina a máquina a través de la red”. Se pueden emplear para integrar aplicaciones que estén programadas en distintos lenguajes, o que al ejecutarse utilicen plataformas diferentes. Esto significa que, los servicios Web “proporcionan un estándar de interoperabilidad entre diferentes aplicaciones *software*, ejecutadas en una variedad de plataformas y entornos” (McCabe et al., 2004)

En cuanto al funcionamiento de los Sistemas Web, el programa cliente hace una petición a través de la Web, y el programa servidor la procesa y emite una respuesta. La siguiente figura muestra de manera simplificada el despliegue de un Servicio Web:

Ilustración 5: Despliegue de un Servicio Web (Jácome Galarza, 2010)

Diagrama

Descripción generada automáticamente

Normalmente, las aplicaciones se comunican mediante el lenguaje *XML* y se envían las solicitudes y respuestas mediante el protocolo *HTTP*. Esto se debe a que es fácil de procesar para las máquinas, y además permite adaptarse a las necesidades de cada sistema. Además, da soporte a las bases de datos, lo que agiliza y facilita el proceso de transmisión de datos entre la propia base de datos y el servidor. (Universidad de Alicante, 2014)

### Ventajas y desventajas de Servicios Web

Los Servicios Web tienen una ventaja principal, y es que usan protocolos y estándares para facilitar el acceso al contenido. Esa estandarización y los pocos requisitos para su funcionamiento facilitan su utilización y la escalabilidad. Por lo tanto, su principal característica su gran interoperabilidad y extensibilidad, así como por proporcionar información fácilmente procesable por las máquinas gracias al uso de *XML*. Los Servicios Web pueden combinarse con muy bajo acoplamiento para conseguir la realización de operaciones complejas. De esta forma, las aplicaciones que proporcionan servicios simples pueden interactuar con otras para "entregar" servicios sofisticados añadidos. En añadido, “los Servicios Web se describen dinámicamente, lo que da lugar a sistemas que se pueden actualizar automáticamente” (Doveltech, 2003).

Sin embargo, los Sistemas Web también tienen desventajas. Y es que, tienen un bajo rendimiento si los comparamos con otros modelos de computación como *Java Remote Method Invocation* o *Common Object Request Broker Architecture* (*CORBA*). El motivo principal es que no se encuentran acoplados al *software* que los utiliza. Otra cosa a tener en cuenta es que, al apoyarse en *HTTP*, pueden esquivar medidas de seguridad basadas en [*firewall*](https://es.wikipedia.org/wiki/Firewall) cuyas reglas tratan de bloquear o auditar la comunicación entre programas a ambos lados de la barrera. Además, para realizar [transacciones](https://es.wikipedia.org/wiki/Transacci%C3%B3n_(inform%C3%A1tica)), no pueden compararse en su grado de desarrollo con los estándares abiertos de [computación distribuida](https://es.wikipedia.org/wiki/Computaci%C3%B3n_distribuida) como [*CORBA*](https://es.wikipedia.org/wiki/CORBA). (Wikipedia, 2021)

En cuanto a los tipos de arquitecturas de desarrollo Web, se pueden distinguir dos tipos: los basados en Arquitectura Orientada a Servicios llamados SOAP (*"big" Web Services*), y los basados en Arquitectura Orientada a Recursos (servicios Web *RESTful*). A continuación, se estudiarán ambos tipos y se compararán entre sí.

### Arquitectura Orientada a Servicios (SOA)

Una Arquitectura Orientada a Servicios (SOA) es un modelo de diseño que consiste en “encapsular la lógica de la aplicación dentro de los servicios que interactúan a través de un protocolo de comunicaciones común. Cuando se utilizan Servicios Web para establecer este marco de comunicaciones, básicamente representan una implementación basada en Web de una SOA” (Erl, 2004). Los actores y las interacciones previamente mencionados se representan en el siguiente diagrama:

Diagrama

Descripción generada automáticamente

Ilustración 6: Diagrama de una Arquitectura Orientada a Servicios (SOA) (V. Issarny et al., 2011

Cada servicio tiene un papel o rol concreto. En cuanto al *Service Provider* (*SP*) o proveedor de servicios, “se encarga de crear Servicios Web, ofrecerlos a un registro de servicios disponibles y gestionar sus condiciones de uso”. En otras palabras, si un Servicio Web es un proveedor, este permite a los solicitantes del servicio lo invoquen gracias a una interfaz pública. “Un proveedor de servicios promueve esta interfaz mediante la publicación de una descripción del servicio”, y deberá asegurarse de que es accesible a todo solicitante que requiera ese servicio. En un modelo cliente-servidor, el proveedor de servicios sería el servidor.

El *Service Consumer* (*SC*) o solicitante del servicio, “se encarga de brindar información acerca del servicio a quien lo solicite, y puede ser público o privado”. El solicitante puede ser un programa que interactúe automáticamente, o una persona que controle un navegador. Uno u otro envían un mensaje de solicitud al Servicio Web para invocarlo. En el modelo cliente-servidor, el *SC* sería el cliente.

En tercer lugar, el *Service Registry* (*SR*) o registro de servicios “buscará un servicio en el registro o por medio del agente, y se conectará con el proveedor para recibirlo”. Los solicitantes acceden al *SR* para poder solicitar los servicios, que previamente han sido añadidos con una descripción por los proveedores. Esa descripción incluye información relativas al servicio, como pueden ser las operaciones que se pueden llevar a cabo, la localización, o los tipos de datos. Además, “El servicio Web asume el papel de intermediario cuando recibe un mensaje de un solicitante de servicio y luego reenvía el mensaje a un proveedor de servicio”.

#### Servicios Web basados en la pila de protocolos WS-\*

Este tipo de Servicios Web se basan en el modelo *SOA* anteriormente explicado, y utiliza una serie de protocolos estándar para las comunicaciones entre actores, o *Simple Object Access Protocol* (*SOAP*). Este estándar de protocolos utiliza mensajes encriptados mediante *XML*, ycomo podemos observar en el siguiente diagrama, hay más protocolos que intervienen en las comunicaciones:

Diagrama

Descripción generada automáticamente

Ilustración : Detalle tecnológico de una Arquitectura Orientada a Servicios (SOA). (Issarny et al., 2011)

En efecto, *SOAP* se utiliza para las comunicaciones entre proveedores de servicio y los solicitantes. Sin embargo, el *Web Services Description Language* (*WSDL*) es utilizado para las descripciones del servicio, y el *Universal Description Discovery Integration* (*UDDI*) es utilizado por el registro para verificar los servicios disponibles.

### Arquitectura Orientada a Recursos (ROA)

Diagrama, PowerPoint

Descripción generada automáticamenteLa Arquitectura Orientada a Recursos (ROA) se basa en el estilo arquitectónico *Representational State Transfer* (*REST*) que se puede definir como “una arquitectura de *software* para sistemas hipermedia distribuidos”, definida por Roy Fielding en su tesis doctoral (Thomas Fieldind, 2000). En otras palabras, es una arquitectura de tipo *REST* para Servicios Web. La estructura cliente-servidor en la que se basa el modelo *REST* se muestra en el siguiente diagrama:

Ilustración 8: Diagrama de una Arquitectura Orientada a Servicios (SOA) (Gitonga, 2021)

Como podemos observar, el cliente solicita el servicio deseado accediendo a través de un puerto o *socket*. Para las comunicaciones con el proveedor del servicio, se utiliza el protocolo *HTTP*. Algunas características de este tipo de arquitectura son su simplicidad en cuanto a las interfaces, su escalabilidad en las interacciones, su rendimiento, y su fiabilidad. En los siguientes párrafos, se detallan las restricciones que impone la arquitectura *REST*.

Un recurso es un elemento necesario para el cliente que va a implementar el Servicio Web. Cada uno de los recursos del sistema son únicos, puesto que están identificados por los llamados *Unified Resource Identifiers* (*URIs*). En este proyecto, algunos recursos son “alumno”, “reserva”, o “puesto”; y sus *URIs* serían, por ejemplo, “/alumnos” para identificar al conjunto de alumno, o “/alumnos/:id” para localizar a un alumno en concreto con su número identificador. Y es que, las *URIs* permiten ubicar los recursos, puesto que no sólo tienen un nombre, sino que se puede “navegar” entre los recursos como si de un gestor de archivos con carpetas o base de datos se tratara. Para evitar problemas, existen unas reglas básicas a la hora de crear *URIs*: cada recurso debe tener una *URI* que le identifique, no deben ser acciones (por tanto, hay que evitar los verbos), y no pueden filtrar ninguna información del recurso que identifica.

Un recurso tiene diferentes estados a lo largo del tiempo. Se denomina “representación de un recurso” un estado dado del mismo, en un momento concreto. Cuando se solicita la representación de ese recurso, se devuelve una “fotografía” suya tomada ese mismo momento, en la cual se especifica cómo tratarlos y otra información adicional. Para solicitar un recurso en la Web, se utiliza *JSON* si se va a utilizar en algún programa, o en *HTML* si es a través del navegador. Además, se usa el *Multipurpose Internet Mail Extensions* (*MIME*) como estándar de codificación de las representaciones de recursos.

Hipermedia hace referencia a los hiperenlaces que componen la representación de los recursos y que los relacionan entre sí. El *Hypermedia as the Engine of the Application State* (*HATEOAS*) es una de las propiedades más relevantes de *REST*: “como este enfoque de diseño de aplicaciones ha de ofrecer una **interfaz universal**, lo que postula *HATEOAS* es que el cliente pueda moverse por la aplicación web únicamente siguiendo a los identificadores únicos *URI* en formato hipermedia. Cuando se aplica este principio, el cliente, aparte de una comprensión básica de los hipermedia, no necesita más información para poder interactuar con la aplicación y el servidor” (Digital Guide, 2018).

Como se ha explicado anteriormente, *REST* se basa en una estructura cliente-servidor, en la cual el primero lanza una petición de servicio al segundo, quien procesa y responde lo requerido. En la Web, esta comunicación síncrona se realiza mediante *HTTP* gracias a métodos que permiten hacer los cambios del estado del recurso, entre los cuales destacan *GET*, *PUT*, *POST*, *DELETE*, *PATCH*.

Como su propio nombre indica, un Servicio Web *RESTful* es aquel que sigue el modelo ROA descrito anteriormente, es decir, en el que hay recursos, interacciones, identificadores, representaciones, etcétera. Además, en él se usan protocolos y estándares para las comunicaciones entre los diferentes actores que lo componen como *HTTP*, *URIs*, *HTML*, o *JSON* entre otros. Para la construcción de un Servicio Web *RESTful*, se deben seguir los siguientes pasos que se ajustan a las restricciones de una arquitectura ROA, tal y como hemos explicado anteriormente.

1. **Definición de un recurso:** Como se ha visto anteriormente, un Sistema Web se basa en la gestión de recursos. Los diferentes actores (*SC*, *SP* y *SR*) navegan por el sistema para acceder a ellos, por lo que es imprescindible definirlos antes de ofertarlos y así poder ser solicitados y tratados.
2. **Identificación de recursos:** Después de definir los recursos, se deben identificar correctamente. El objetivo es facilitar el acceso a los recursos por parte del usuario, y evitar así posibles confusiones, errores humanos, y creando un sistema eficiente en cuanto a tiempos y costes.

Continuando con el objetivo de facilitarle al usuario las interacciones con el sistema y los recursos, las *URIs* siguen unas reglas básicas como se ha explicado anteriormente. Se podrían crear *URIs* con palabras al azar, pero lo que ayuda al usuario es el hecho de tener una semántica que represente los recursos y permita identificarlos fácilmente. Esto ayuda a la hora de navegar entre los recursos del sistema, y llegar al deseado más rápidamente. Y es que, el usuario que utiliza el sistema no tiene por qué saber qué recursos tiene el mismo. Por ello, cuanto más sentido tengan los hiperenlaces, más fácil será interactuar con el sistema.

1. **Representaciones utilizadas:** Algunos tipos de representaciones de recursos que se utilizan en este proyecto son *XML* o *JSON*. Como se ha explicado anteriormente, *HATEOAS* permite crear un sistema hipermedia distribuido. Es importante diseñar las representaciones de los recursos, y las *URIs* que van a ser utilizadas en el sistema.
2. **Establecimiento de la interfaz:** En la Web, se utiliza el protocolo *HTTP* para proporcionar la interfaz y mostrar el estado de un recurso. Para cambiar esos estados, usa los métodos *GET*, *PUT*, *POST*, *PATCH*, y *DELETE*. Además, como hemos visto anteriormente, este protocolo es la puerta de entrada al sistema a través del “*socket*”. Así, el método de la petición al servicio es muy importante puesto que establece la manera en la que cambian los recursos y sus estados.

#### Modelo de Madurez de Richardson

Leonard Richardson propuso una clasificación de los servicios en la Web según sigan las restricciones de una arquitectura *REST*. El modelo se incluye en  
su libro “*RESTful* Web Services” publicado en el año 2007 (Richardson & Ruby, 2007). Este modelo en cuestión presenta tres niveles de madurez de servicio: soportes de un servicio para *URI*, *HTTP* e Hipermedia (también hay un cuarto nivel sin soporte). La siguiente figura  
representa los diferentes niveles del Modelo de Madurez de Richardson:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

Ilustración 8: Modelo de Madurez de Richardson. (Fowler, 2010)

##### Nivel 0 – The Swamp of POX

El nivel 0 o nivel básico del Modelo de Madurez de Richardson hace referencia a los servicios que tienen un solo *URI* y que utilizan un solo método *HTTP* (normalmente *POST*). No llegan a utilizar ningún otro mecanismo de la web. Sería lo equivalente a una tubería para enviar la información solicitada por parte del *Service Consumer.* Los servicios basados en *Web Services* (*WS-\**) se clasificarían en el nivel 0. La siguiente figura muestra un ejemplo práctico de un modelo de nivel 0 según Richardson:

Texto

Descripción generada automáticamenteEscala de tiempo

Descripción generada automáticamente con confianza bajaSe puede observar cómo hay dos actores, un paciente y un doctor. El paciente envía una solicitud al doctor preguntando por la disponibilidad del doctor para un día concreto:

Ilustración 9: Ejemplo nivel 0 Modelo de Madurez de Richardson (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

Ilustración 10: Ejemplo solicitud de servicio nivel 0. (Fowler, 2010)

La respuesta del servidor es la siguiente:

Texto

Descripción generada automáticamente

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Ilustración 11: Ejemplo respuesta del servidor de servicio nivel 0. (Fowler, 2010)

Como se puede observar, la respuesta consiste en dos franjas de horarios en los cuales el doctor está disponible. *POST* no es el método que  
permita alcanzar la representación de un recurso, por lo que esta interacción o comunicación no sigue los principios de *REST*.

##### Nivel 1 - Resources

Diagrama

Descripción generada automáticamenteEn este nivel de madurez, se introduce la gestión de varios recursos individuales en vez de uno grande y complejo como en el nivel 0. En el nivel 1 se emplean “muchos *URI* pero un solo un único verbo *HTTP*”. Además, “las operaciones se tunelizan insertando nombres de operaciones y parámetros en un *URI* y luego transmitiendo ese *URI* a un servicio remoto, generalmente a través de *HTTP GET”* (Fowler, 2010) *.* En la siguiente figura, se muestra el uso de varios recursos y sus identificadores:

Ilustración 12: Ejemplo nivel 1 Modelo de Madurez de Richardson (Álvarez Caules, 2016)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 12: Ejemplo nivel 1 Modelo de Madures de Richarson. (Fowler, 2010)

##### Nivel 2 – HTTP verbs (métodos HTTP)

Diagrama

Descripción generada automáticamenteEn el nivel 2 se usan de diferentes métodos *HTTP*, lo que permite variar los estados de los recursos. Y es que, los servicios *Create*, *Read*, *Update*, y *Delete* (*CRUD*), permiten modificar los estados a través de la red. *La siguiente figura representa esas opciones: para el recurso “Compra”, se crea o inserta si la solicitud se hace con el método POST, se lee o consulta si el método es GET, se actualiza si el método es PUT, y finalmente se borra si el método es DELETE*.

Ilustración 13: Ejemplo nivel 2 Modelo de Madurez de Richardson (Álvarez Caules, 2016)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 13: Ejemplo nivel 2 Modelo de Madures de Richarson. (Fowler, 2010)

##### Nivel 3 – Hypermedia Controls

Este nivel implica el uso de controles hipermedia como motor del estado de la aplicación o Hypermedia as the engine of applicacion state (HATEOAS). Esto ayuda al usuario, puesto que le indica qué puede hacer a continuación: las diferentes representaciones de los recursos contienen enlaces URI a otros recursos que podrían ser de interés para los usuarios. Es el propio sistema el que ayuda al usuario diciéndole cuáles son los siguientes pasos para usar la aplicación. En las respuestas, le enseña nuevos recursos y métodos que podría utilizar con ellos.

Siguiendo el ejemplo del paciente y el doctor, “en lugar de que tengamos que saber dónde publicar nuestra solicitud de cita, los controles hipermedia en la respuesta nos dicen cómo hacerlo”. La siguiente figura muestra otro ejemplo, en el cual un alumno llamado “Fernando” está matriculado en un curso “http://url/cursos”. En el mismo recurso hay una URI que especifica el cuso que corresponde a “Fernando”. En la figura en cuestión, se puede observar cómo se obtendría una representación del alumno al hacer una solicitud GET, pero a su vez la URI del recurso ayudaría a saber el curso de este de forma indirecta.

Ilustración 14: Ejemplo nivel 3 Modelo de Madurez de Richardson (Álvarez Caules, 2016)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

Ilustración 14: Ejemplo nivel 3 Modelo de Madures de Richarson. (Fowler, 2010)

#### Comparativa entre ambas aproximaciones

*Diagrama

Descripción generada automáticamente*En este apartado se van a comparar los modelos *REST* y *SOAP*. El objetivo es identificar ventajas e inconvenientes de ambas aproximaciones, y ver cuál es la más utilizada.

En cuanto a REST, que sigue una filosofía de Web abierta, es relativamente fácil de implementar y mantener. Además, separa claramente las implementaciones de cliente y servidor, y la comunicación no está controlada por una sola entidad. El cliente puede almacenar información para evitar múltiples llamadas, y además REST puede devolver datos en múltiples formatos (JSON, XML, etc.). Sin embargo, sólo funciona sobre el protocolo HTTP y es difícil garantizar la autorización y la seguridad por encima.

Por otro lado, *SOAP* sigue un enfoque empresarial formal, y funciona sobre cualquier protocolo de comunicación, incluso de forma asíncrona. También, hay que tener en cuenta que se comunica a los clientes la información sobre los objetos. Además, la seguridad y la autorización son parte del protocolo, y se pueden describir completamente usando *WSDL*. No obstante, *SOAP* también tiene puntos negativos. En efecto, gasta mucho ancho de banda comunicando metadatos, usa solo *XML*, es difícil de implementar y no es popular entre los desarrolladores Web y móviles. *SOAP* está más enfocado a operaciones transaccionales con usuarios que están satisfechos con la tecnología (Fisher, 2015).

Características como “la aplicación del principio *HATEOAS* permite que la interfaz de un servicio *REST* pueda modificarse siempre que se requiera, lo que constituye una ventaja fundamental de esta arquitectura frente a otras, como las basadas en *SOAP*” (Digital Guide, 2018). Además, si se añade su escalabilidad, simplicidad en cuanto a infraestructura, y rendimiento con interacciones sincronizadas hace que sea la forma más utilizada hoy en día para desarrollar Aplicaciones Web (Fisher, 2015).

## Tecnologías

Este apartado se aborda estudiando las dos perspectivas con las cuales hay que analizar un Servicio Web, es decir, la tecnología relacionada con el *BackEnd*, y la relacionada con el *FrontEnd*. Esta última se refiere a la parte visual de la Web que el cliente percibe, la interfaz con la que interactúa y que por tanto debe ser amigable. El *BackEnd* es la lógica que hay detrás, que incluye la base de datos y la conexión con el servidor.

### BackEnd

**PLAY Framework:** Este *framework* es una herramienta que facilita el desarrollo de Servicios Web. Provee de una interfaz *HTTP* amigable, simple, potente y flexible, algo que facilita el desarrollo del prototipo de este trabajo. Además, gracias a la gran comunidad que utiliza esta herramienta, hay mucha documentación, tutoriales, e información que ayuda al desarrollo de este proyecto. Si bien en este caso se va a desarrollar en el lenguaje de programación *Java*, *PLAY* *framework* también soporta el lenguaje *Scala*. Más adelante, se estudia detalladamente el funcionamiento de *PLAY* mediante un ejemplo (Leroux & de Kaper, 2014).

**Base de datos:** En este proyecto se va a utilizar *MySQL*. Es un gestor de base de datos relacional tradicional, y se considera un software de código abierto, por lo que se adapta a los requerimientos del proyecto. Además, es fácil de instalar y configurar, no requiere mucha capacidad de cómputo, y destaca por su persistencia de datos. Por todo ello se ha decidido utilizar *MySQL***.**

**Postman:** Esta herramienta permite testear la API permitiendo generar peticiones. *Postman facilita el trabajo de desarrolladores y les permite avanzar en el desarrollo de proyectos comprobando que la comunicación entre la API, base de datos, y aplicación funciona correctamente. Además, es fácil instalarlo y utilizarlo, pero incluso se puede usar mediante una extensión en el navegador. También* *permite la automatización de dichos tests de integración de cualquier proyecto.*

### FrontEnd

En el proyecto, se han utilizado los lenguajes HTML, CSS, y JavaScript principalmente, y FreeMaker como motor de plantillas.

***Hypertext Markup Language (HTML):*** *HTML* es un lenguaje de programación que surgió alrededor de los años 90 y asociado a la creación de páginas Web. Mediante etiquetas, (utilizando los símbolos “<” y “>”), este lenguaje permite dar forma a la Web organizando los elementos que la componen, y por tanto establecer cómo se va a ver la información. El orden en que se programan los elementos es muy importante, puesto que será el mismo orden en el que aparecerán visualmente. (*HTML - Standard.*2022)

Los elementos utilizados en *HTML* son muy variados, incluso también se puede precisar en qué zona o sección de la página se situarían. Algunos ejemplos de etiquetas utilizadas son las siguientes: <title> (editar título), <body> (cuerpo de la página), <header> (zona superior), <footer> (zona inferior), <article> (incluir un artículo paginado dentro de la Web), <section> (apartado dentro del cuerpo), <img> (insertar una imagen), <video> (insertar un vídeo), <ul> (lista desordenada), <ol> (lista ordenada), <li> (lista de elementos), etc. (MDN Contributors, 2021)

Una vez establecida la información requerida en la Web gracias a *HTML*, se puede combinar con el lenguaje *Cascading* *Style* *Sheets* (*CSS*) para editar la apariencia de esa información, o incluso el lenguaje JavaScript para incluir otras funcionalidades en la Web. A continuación, se presentan ambos lenguajes.

**CSS:** Este lenguaje permite modificar la apariencia de la información requerida en la Web. En efecto, posibilita cambiar tipos de fuente, colores, tamaños, espaciados, o formatos de los datos representados. Además, al ser independiente de *HTML*, hace que varias páginas de un mismo sistema puedan tener una apariencia homogénea, sin tener que configurar los mismos parámetros en cada una de ellas por separado.

Otra ventaja de *CSS* es, que cumple con una función de estructurar la información. Y es que, este lenguaje adapta los datos a la pantalla que esté mostrando la página. Por tanto, permite que la página Web creada pueda ser visualizada en pantallas de distinto tamaño como por ejemplo dispositivos móviles, monitores, u otros. (*HTML & CSS.*)

**JavaScript:** *JavaScript* es un lenguaje que complementa tanto *HTML* como *CSS*. Permite añadir funcionalidades a la página que se quiere desarrollar gracias a scripts, para así hacerla más dinámica y mostrar ciertos cambios si necesidad de refrescarla. Por ejemplo, actualiza el contenido, muestra tablas o mapas interactivos, o anima gráficos en 2D y 3D.

Las aplicaciones Web desarrolladas con *JavaScript* pueden estar disponibles en el navegador como una página web, o pueden distribuirse como *widgets*. Además, los *scripts* permiten crear un puente entre el navegador que muestra la Web y la plataforma en la se ejecuta, para así incorporar información del entorno del usuario, como por ejemplo la ubicación.

En este proyecto, se utiliza *JavaScript* en las llamadas Interfaces de Programación de Aplicaciones (API). Las APIs son bloques de código listos para ser utilizados por los desarrolladores, y así poder implementar programas que de otro modo serían más complejos o imposibles de implementar. Existen dos tipos de APIs: las que están integradas en el navegador, y las que no.

*JavaScript* es, por tanto, un lenguaje que complementa y aporta valor a los datos programados en *HTML* y *CSS*. Esto permite poder sacarle más partido al desarrollo Web creando páginas más dinámicas. (Contributors, 2022)

**JSON:** *JavaScript* *Object* *Notation* es lenguaje de representación de datos utilizado para *FrontEnd* por su procesamiento fácil con *JavaScript*. Además, es intuitivo y fácil de entender y utilizar.

**FreeMarker:** *Apache* *FreeMarker* es un motor de plantillas programadas en *FreeMarker* *Template Language* (*FTL*), un lenguaje simple y especializado. Se suele utilizar otro lenguaje (*Java* u otro) para preparar los datos, por ejemplo, haciendo consultas a la base de datos, para que luego *FreeMarker* muestre los resultados en las plantillas.

Diagrama

Descripción generada automáticamente

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

Ilustración 15: Funcionamiento de FreeMarker. (Apache.org, 2015)

# Análisis

Al comenzar la pandemia, la universidad empezó a aplicar medidas para proteger a alumnado y trabajadores. Una de esas medidas, consistía en el control de acceso de la biblioteca de la Escuela Politécnica Superior. Al inicio de la realización de este proyecto, y motivo por el cuál surgió la idea, se utilizaba un método demasiado rudimentario para reservar los sitios de la biblioteca. Tras mi experiencia como alumno, y reuniones con los trabajadores de la biblioteca, se recabó información suficiente para poder identificar las necesidades que permitieran agilizar el proceso de reserva. En este capítulo, se analiza el en un primer lugar el dominio de la aplicación para entender el contexto en el que surge la idea del proyecto, así como el método de reserva primigenio propuesto por la universidad, y la implementación de otro *software* llamado *Affluences*. En ese mismo apartado también se detallan los recursos y actores que pertenecen al dominio. En segundo lugar, se detallan situaciones concretas que se pueden dar en el uso de la aplicación y, finalmente, se especifican los requisitos funcionales y no funcionales de la aplicación propuesta en este trabajo.

## Análisis del dominio de la aplicación

Después de la cuarentena nacional, se ha ido recuperando poco a poco la vida normal y se han abierto de nuevo los centros educativos, la universidad ha implementado medidas de prevención, para evitar la propagación del virus en la universidad pudiendo contagiar a trabajadores y alumnado. Uno de los muchos espacios que han pasado a tener control es la biblioteca. En efecto, se ha decidido pasar de un total libre acceso a uno de los servicios más utilizados por los alumnos en la universidad, a un acceso más restringido y controlado. Para proteger tanto a alumnos como trabajadores, se ha pasado a tener una mayor vigilancia del aforo de la biblioteca. Todos los alumnos, profesores, o trabajadores de la biblioteca deben

llevar una mascarilla que cubra nariz y boca durante todo el tiempo que estén en la biblioteca. Además, hay un dispensador de gel hidroalcohólico en la entrada de la biblioteca,

y toda persona que quiera entrar debe lavarse las manos con él. Cabe destacar que los puestos de estudio están numerados y separados según las restricciones sanitarias para frenar la propagación del virus (1,5 metros). También se ha prohibido el levantarse para interactuar con otro alumno (saludar, preguntar dudas, estudiar juntos, …), sólo se permite salir a descansar.

Para mayor prevención, se mantienen abiertas las ventanas cada cierto tiempo para favorecer la ventilación, renovar continuamente el aire en el ambiente, y evitar los aerosoles puesto que es una vía de contagio del virus. Además, se debe tener en cuenta la desinfección de los puestos. Así, cuando un alumno abandona un puesto, éste debe quedar limpio y preparado para que otro alumno lo pueda ocupar. No sólo actúa el equipo de limpieza, sino que también hay desinfectante y papel a disposición de los alumnos que quieran limpiar de nuevo su puesto. Esto permite evitar el contagio entre alumnos que ocupen el mismo puesto.

Cabe destacar que, antes de la pandemia, cualquier alumno o profesor que quisiera consultar algún libro tenía libre acceso a ejemplares de las diferentes estanterías presentes en la biblioteca. Sin embargo, las estanterías han pasado a ser inaccesibles, para evitar el contagio. Lo mismo ocurre con otros servicios de la universidad como: ordenadores, alargadores, o las pequeñas salas de trabajos en grupo. Y es que, no sólo hay que pensar en restricciones en cuanto a nivel de aforo, sino también en cuanto a los servicios que ofrece la biblioteca. Para poder ofrecerlos manteniendo un alto nivel de seguridad frente al virus, se centralizaron pidiendo su uso a los responsables de la biblioteca para poder desinfectarlos y tener un registro de dichos préstamos. Conviene poner énfasis en que no sólo se ha reducido el aforo de la estancia, sino que existe un registro de los alumnos que entran a estudiar cada día, la franja horaria en la que lo hacen, y el número de puesto en el que se sientan. Esto ayuda a tener una trazabilidad en caso de que algún alumno diera positivo en coronavirus.

No sólo hay que prestar atención a medidas sanitarias que afecten directamente a los alumnos, sino también a los procesos que llevan a cabo en la biblioteca. Por ejemplo, cuando un alumno entra en la biblioteca, se le asigna un puesto en el que estudiar. Sin embargo, algunos alumnos tienen preferencias a la hora de estudiar por la ubicación del puesto, la incidencia de luz natural, posición de enchufes, u otros motivos. Es por ello que el puesto de estudio no debería ser impuesto, sino que tendrían que tener libertad de elección. Pensando en los alumnos de primeros cursos, que quizás no conozcan del todo la biblioteca, sería conveniente que tuvieran a su disposición un mapa cenital de las estancias con cierta información relevante como la ubicación de enchufes, ventanas, u ordenadores fijos. Incluso, se podría permitir la reserva de puestos, sobre todo teniendo en cuenta que hay épocas del año de mayor afluencia a la biblioteca. Y es que, las semanas de vacaciones y dias previos a exámenes de convocatoria ordinaria, extraordinaria, o fines de semana, muchos alumnos prefieren estudiar en la biblioteca de la universidad. Por ello, sería útil que los alumnos pudieran reservar los sitios con antelación, para asegurarse una plaza y evitar ir a la biblioteca si no hay plazas libres. A la hora de hacer la reserva, se podría asignar un número identificador a cada una de ellas. Por tanto, cada alumno tendría asociado un código con la información relevante: fecha, hora, duración de la reserva, puesto, …

Para evitar que haya problemas de aforo en la biblioteca, y que cada alumno pueda asegurar un puesto de estudio, la universidad implementó finalmente un sistema de reservas a través del correo electrónico de la universidad. Este método era necesario, sobre todo para épocas del año señaladas como son las semanas previas a exámenes parciales, y las convocatorias ordinarias y extraordinarias. En el siguiente apartado veremos en qué consistía y qué procesos cubría. *(en el anexo … se puede ver la solución informal primigenia)*

### Solución informal primigenia

Para intentar controlar mejor el aforo de la biblioteca y evitar que alumnos se desplazaran a estudiar a la biblioteca sin saber si iba a haber sitio o no, la universidad decidió implantar un sistema de reserva de puestos a través del correo electrónico.

Cada alumno que quisiera acudir a la biblioteca debía mandar previamente un correo dando la siguiente información: nombre, apellidos, DNI, horario en el cual estaría en la biblioteca, y el número del sitio en el que deseaba ubicarse en el caso de que tener alguna preferencia. El personal de la biblioteca se encargaría de gestionar el correo y anotar, a mano, en un archivo Excel dichos datos, para dejar constancia de cada una de las reservas que se hacen y tener un seguimiento. Si el alumno que reserva no tiene ninguna preferencia de puesto, el responsable de la biblioteca le asignaría un puesto, notificándoselo por correo. Una vez el alumno accedía a la biblioteca, debía notificar al responsable de la entrada que había llegado a tiempo y se sentaba en el sitio reservado o asignado por el responsable. De no llegar a tiempo, con un margen aproximado de cinco o diez minutos, el responsable de la biblioteca cancelaría su reserva borrándola del Excel. Al abandonar el puesto (acabando así la jornada de estudio), el alumno debía notificarlo al responsable de la entrada. Éste anotaba en el Excel que el sitio en cuestión quedaba libre, y lo marcaba para que fuese limpiado. Cada cierto tiempo, un alumno que estuviera estudiando podía ver cómo entraba el equipo de limpieza y desinfectaba los sitios vacíos anteriormente ocupados y ya liberados. Cabe destacar que, hasta que no se limpiaban esos puestos de trabajo, no se pueda reservar o asignar por lo que, si algún alumno mandaba un correo pidiendo un sitio en esa situación, el propio responsable le contestaba asignándole otro.

El sistema de reservas mediante correo electrónico se empezó a emplear justo antes de los exámenes, principalmente, para evitar aglomeraciones en la puerta de la biblioteca esperando a que hubiera puestos libres. Este método no informatizado permitía que los alumnos fueran a la biblioteca sabiendo que tenían un sitio reservado, y que podrían disfrutar de diferentes estancias para estudiar estando seguros. Sin embargo, rápidamente se observó que no era una solución óptima, puesto que no era eficiente: falta de información al implementarla, requería una persona pendiente de los correos problemas con el Excel y sus versiones, quejas por parte de los alumnos de que se les avisaba de la falta de puestos libres al estar ya de camino a la universidad, …

En aquel momento no se sabía cuánto iba a durar la pandemia, y las restricciones por parte del Gobierno en cuanto a porcentaje de aforos permitidos eran cambiantes, por ello se implementó esa solución momentáneamente. Y es que, de cara al curso siguiente, la universidad invirtió en un nuevo software externo llamado Affluences para la gestión de las reservas de puestos en la biblioteca.

Solución de software Affluences

Durante la realización de este proyecto surgió un cambio importante: la implementación de esta aplicación, de origen francés, que supuso tener que hacer un análisis exhaustivo de la misma para entender qué necesidades cubre y cuáles no.

Los alumnos pueden hacer reservas a través de la aplicación web o móvil, y así garantizar un sitio en la biblioteca. Además, el personal de la biblioteca puede acceder a ellas para tener un mayor control de esta, y un seguimiento sin tener que hacerlo manualmente en Excel (en los Anexo I y Anexo II se pueden encontrar unos análisis de casos de uso desde el punto de vista del alumno y del bibliotecario respectivamente).

Si un alumno desea realizar una reserva, debe asociar su cuenta de correo universitario a la aplicación Affluences, y seleccionar la biblioteca de la Universidad San Pablo CEU. A continuación, elegir el puesto que quiere reservar seleccionando el día, la hora, y la duración de la reserva. El alumno tiene a su disposición un archivo PDF que contiene una foto cenital de los puestos numerados, para poder así elegir el puesto deseado. Se debe seleccionar la fecha en que se quiere realizar dicha reserva, y su duración. Para este último parámetro, el alumno tiene varias opciones pudiendo seleccionar duraciones entre un mínimo de treinta minutos y un máximo de tres horas, en intervalos de media hora (es decir: media hora, una hora, una hora y media, dos horas, dos horas y media o tres horas).

Una vez el alumno llega a la biblioteca, tiene que confirmar su presencialidad escaneando un código QR (impreso y pegado en la puerta de la biblioteca) con la cámara del móvil, o introduciendo manualmente el código de cuatro dígitos (situado en el mismo papel). El alumno puede validar esa presencialidad desde diez minutos antes de la hora de la reserva, y hasta media hora después. Si no lo hace, la reserva se cancela automáticamente quedando libre el sitio y pudiendo ser elegido por otro alumno. En el caso de que un estudiante quiera estar más de tres horas en un mismo puesto, tiene dos opciones: hacer la misma reserva varias veces antes de entrar a la sala de estudio y confirmar su presencialidad cada cambio de reserva, o renovarla cada vez que llegue al límite de las reservas y confirmar su presencialidad en cada momento. Es más recomendable la primera opción, puesto que si llega el límite de una reserva y otro alumno elige el mismo sitio, deberá hacer otra elección y cambiarse de puesto.

A la hora de abandonar el puesto de estudio, el alumno no debe notificárselo a ningún trabajador de la biblioteca como sucedía anteriormente, sino que si ha llegado el final de su reserva puede irse sin problemas. Además, si el alumno quiere abandonar su puesto antes del final de su reserva, tiene la obligación moral de abrir la aplicación para cancelar su reserva. Esto evitaría que ese puesto estuviera vacío durante un tiempo determinado hasta que acabara la reserva y así otro alumno pueda elegirlo cuanto antes. Esto es especialmente relevante en época de exámenes tanto parciales como de convocatoria ordinaria o extraordinaria, ya que hay más afluencia de alumnos a la biblioteca.

En cuanto a los responsables de la biblioteca, tienen acceso a todas las reservas que se hacen durante el día, así como a un historial de reservas anteriores de cada alumno. Además, pueden realizar y cancelar reservas, y poner sanciones a alumnos que incumplan el reglamento de la universidad (ya sea por comportamiento, por no cumplir las medidas sanitarias frente al virus, …). Los responsables tienen dos grandes funcionalidades en su aplicación web. Por una parte, pueden configurar todos los parámetros de las reservas y los recursos de la biblioteca (duración máxima de las reservas, número de puestos, código de confirmación de presencialidad, horario de apertura de la biblioteca, …). Por otro lado, también pueden acceder a datos y estadísticas de la biblioteca. Incluso, pueden exportarlas a Excel, verlas de forma más visual en un dashboard (cuadro de mando) o crear un informe resumido. Algunas de dichas estadísticas son: entradas a la biblioteca, número de reservas, recursos reservados, uso de reservas (aplicación o web), …

Por su parte, el equipo de limpieza desinfecta todos los puestos a primera hora de la mañana, todos los sitios que estén vacíos a la hora de comer, y a media tarde. Por tanto, en épocas de exámenes donde no se suelen liberar muchos asientos, cabe la posibilidad de que un alumno reserve y se sienta en un puesto abandonado por otro alumno sin que haya sido desinfectado. Para ello, la universidad puso a disposición de los alumnos desinfectante y papel.

Como la aplicación Affluences fue implementada mientras se realizaba este trabajo, no sólo se ha analizado usándola, sino que también se han realizado varios días de entrevistas a alumnos y personal de la biblioteca para comprender y contextualizar la nueva aplicación. En cuanto a la versión de alumnado, el autor ha podido probar la aplicación, puesto que la ha usado en su día a día al ir a la biblioteca. Además, se ha entrevistado a un total de diez (10) alumnos para saber su opinión y su experiencia usando el software Affluences. Para analizar la aplicación desde el punto de vista de los responsables de la biblioteca, se han entrevistado a cinco (5) empleados: Itziar, responsable de la biblioteca de la EPS; y otros cuatro (4) ayudantes de biblioteca. También se obtuvo un permiso para poder analizar exhaustivamente la interfaz de usuario de un empleado, y así ver a qué información tienen acceso y qué se puede configurar en el software Affluences. Gracias a esos análisis, se pudieron realizar los Anexos I y II y estudiar los casos de uso del proceso de reserva desde el punto de vista del alumnado y de los empleados de la biblioteca.

En añadido, se presentan en los siguientes subapartados los puntos positivos y negativos de la aplicación Affluences, que surgen de las entrevistas realizadas.

Puntos que destacar del software Affluences

Más fiable: minimiza posibles errores humanos a la hora de seleccionar un sitio u otro, o anotar una hora u otra para cada reserva.

Garantiza integridad de los datos: los datos personales están asociados al correo corporativo con el que se accede a la aplicación.

Solución más eficiente que la situación inicial: minimiza tiempo en los procesos de reserva.

Automatizado: no requiere de un trabajador que gestione todo por email.

Puntos negativos del software Affluences

Poca flexibilidad: la duración de la reserva no se puede introducir manualmente, y es de sólo tres horas como máximo.

Sistema poco ágil: forma colas y cuellos de botella si los alumnos abandonan su puesto y no cancelan su reserva, debería verificar cada cierto tiempo qué sitios hay vacíos.

Interfaz no amigable: requiere un periodo de aprendizaje, poco intuitiva. El alumno debería poder marcar uno o varios sitios favoritos para que no tuviera que buscarlos en una lista de cien puestos, por ejemplo.

Demasiado específica:

No hace una gestión unificada de las diferentes bibliotecas de la universidad. Los campus de la USP CEU tienen diferentes bibliotecas y zonas de estudio. Deberían gestionarse conjuntamente, por tanto, si no hay puestos vacíos en la EPS, el alumno tendría que poder reservar otro de la biblioteca de otra facultad.

El sistema sólo permite la reserva de sitios, no abarca los demás recursos como por ejemplo ordenadores, préstamo de libros, alargadores, …

Incompleta:

Si se reserva una sala en grupo, sólo el alumno que haga la reserva queda registrado en el sistema, sus acompañantes no

Hace lo básico, no tiene ninguna funcionalidad extra que pueda ser de utilidad. Se centra en las reservas de sitios y estaría bien que fuera más global.

Inconsistente:

Pretende identificar a los alumnos, pero el código que verifica la presencialidad es común para todos.

Se pretende corregir los puntos negativos de Affluences que nos han dado los usuarios, pero también añadir funcionalidades nuevas que se han valorado como útiles por ellos mismos.

## Actores

En este apartado se detallan los actores, o personas de interés (*stakeholders*), que intervienen en el dominio de la aplicación propuesta. Estos actores son tanto los alumnos, como los trabajadores de la biblioteca. Los primeros utilizan el *software* para realizar, cambiar, o cancelar reservas; ya sean sitios individuales de estudio, salas de grupo, u otro recurso ofrecido por la biblioteca (por ejemplo, ordenadores de búsqueda, libros del sistema de préstamos, regletas, u otro). Los segundos, utilizan el *software* para administrar la biblioteca. Para ello, tienen la posibilidad de realizar, cambiar, o cancelar reservas como los alumnos, pero también tienen acceso a más información como un historial de reservas de los alumnos, cambiar información de los recursos ofrecidos a los mismos, acceder a la disponibilidad de los puestos para ver el aforo de la biblioteca, etc.

Diagrama

Descripción generada automáticamente con confianza mediaEn el siguiente diagrama podemos ver los diferentes grupos de interés:

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

Ilustración 16: Diagrama de actores del sistema. Elaboración propia

## Recursos

En este apartado, se identifica el conjunto de recursos que intervienen en los procesos de reserva en las bibliotecas de la universidad. Estos recursos son materiales, e intervienen de forma directa o indirecta en los procesos de reserva. En efecto, como se va a ver a continuación, algún recurso que puede parecer secundario puede ser crucial para que un usuario elija un puesto u otro a la hora de realizar una reserva.

### Puestos de estudio

Este recurso representa una mesa que un alumno, profesor, o incluso trabajador de la biblioteca, pueda reservar mediante el software en cuestión. Esta mesa tendría como uso principal el estudio del temario por parte de cualquier alumno, pero también lo puede usar para sentarse a leer. Y es que, las otras situaciones que involucran a profesores o trabajadores no son tan común: un profesor podría reservar un puesto para corregir exámenes o leer, pero normalmente lo hacen en su propio despacho y, los trabajadores de la biblioteca podrían reservar un puesto si algún alumno tuviera algún tipo de problema puntual, pero también tienen un despacho propio donde realizar sus tareas del día a día.

En plenas restricciones de aforo por parte de las instituciones sanitarias, había ciento un (101) puesto de estudio en la biblioteca de la EPS, mientras que actualmente hay doscientos cincuenta y tres (253).

### Salas de trabajos

Como su propio nombre indica, las salas de la biblioteca son pequeñas estancias acristaladas en la cual hay una mesa grande en el interior y sillas para seis (6) personas. Estas salas también pueden ser reservadas tanto por alumnos, como profesores, o empleados. De hecho, es más común que algunos profesores o empleados acudan a alguna sala para tener una reunión, ya sea presencial o telemática.

Durante las olas de la pandemia en las cuales hubo muchos casos y por tanto más restricciones, las seis (6) salas que tiene la biblioteca de la EPS estaban cerradas. Sin embargo, hoy en día están todas abiertas y cualquiera puede acceder con una reserva.

### Libros

El *software* que se presenta en este trabajo tiene como principal objetivo el controlar el acceso a la biblioteca centrándose en las reservas de puestos de estudio y salas. Hay que tener en cuenta que los usuarios pueden acceder a la biblioteca para utilizar el servicio de préstamo de libros. En efecto, un usuario puede pedir un libro y tenerlo en el puesto o sala que haya reservado. Sería necesario tener un control sobre quién accede a un libro, y para ello se utiliza la reserva que haya hecho el usuario en el *software*. Es por ello, que se deben considerar como recursos del sistema.

### Estanterías

Los libros están ubicados en diferentes estanterías a lo largo y ancho de las bibliotecas de la universidad. Cabe la posibilidad de que un alumno, o profesor, quiera reservar un puesto cerca de una estantería o sección de la biblioteca donde sepa que están los libros que vaya a necesitar esa jornada. Por ello, se considerarían las estanterías como un recurso.

### Ordenadores

En las bibliotecas de la universidad, no sólo hay un servicio de préstamo de libros, sino también de ordenadores. Si bien los ordenadores no pueden salir de la biblioteca, no dejan de ser un recurso que cualquier usuario (alumno, empleado o profesor) pueden reservar o pedir, y por tanto tener un seguimiento de control gracias al *software*.

### Enchufes

Hay enchufes en cada una de las bibliotecas de la universidad, pero al haber más puestos que enchufes, en épocas de exámenes parciales y convocatorias ordinarias y/o extraordinarias suelen estar ocupados. Al estudiar con dispositivos electrónicos, los enchufes suelen ser un gran aliciente para reservar un puesto u otro en la biblioteca. Por ello, se consideran un recurso del sistema.

Una solución al problema de falta de enchufes en épocas de gran afluencia de alumnos a la biblioteca es el uso de regletas. Es un servicio que ofrecen los empleados de la biblioteca a estudiantes que lo necesiten. Como se ha explicado anteriormente, en algunas épocas del curso hay más aforo y, por tanto, los enchufes suelen estar ocupados. En vez de apuntar el nombre del alumno que solicita y recoge la regleta en el mostrador de la biblioteca, se podría realizar un seguimiento de esa solicitud, entrega, y devolución gracias al *software*. Es por ello, que la regleta se podría considerar un recurso.

### Ventanas

En todas las bibliotecas de la universidad, las ventanas o ventanales son accesibles a los alumnos, o profesores que estén en las diferentes zonas. Es decir, pueden abrirlas y cerrarlas a su gusto, lo que puede ser un aliciente para reservar un puesto u otro, tanto en épocas de frío como sobre todo en épocas de mucho calor. Por ello, las ventanas también son un recurso del sistema.

### Aire acondicionado

Siguiendo el mismo argumento que para las ventanas, si bien las bibliotecas tienen una climatización regulada por la calefacción central, hay unos mandos de control que permiten apagar o encender el aire acondicionado. El hecho de tener el control del aire acondicionado o de estar justo debajo o lejos de la rejilla por donde sale el aire, es un aliciente para elegir un sitio u otro, por tanto, se considera un recurso del sistema.

## Procesos

Diagrama

Descripción generada automáticamenteEn este apartado se describen los diferentes casos de uso que se pueden dar cuando los usuarios realicen reservas a través del sistema. Primero se muestra un diagrama de casos de uso, y más adelante se explica cada situación en detalle:

Ilustración 10: Diagrama de casos de uso. Elaboración propia

### Reserva en biblioteca

Para este proceso, como podemos ver en la figura anterior, cualquier usuario puede realizar una reserva a través del sistema. A la hora de hacerla, debería identificarse con sus datos personales, deberá elegir un sitio al que querrá acudir, así como el día, la hora, y la duración de la reserva. Si un usuario quiere modificar su reserva, puede hacerlo dando paso al proceso “*Cambio* *de* *reserva*” (*Apartado 4.4.3*). Cabe destacar que una vez se ejecute el proceso de reserva, el usuario tiene que elegir qué tipo de reserva quiere hacer ejecutando cualquiera de los procesos: “*Reserva de sala*” (*Apartado 4.4.4*), o “*Reserva de puesto*” (*Apartado 4.4.5*). Ahí, se aportarán los datos necesarios (día, hora, duración, recurso a reservar) y por tanto altera su disponibilidad para otros usuarios. Así, si el usuario pretende hacer una reserva, pero la biblioteca está en aforo completo, no habrá disponibilidad y se ejecuta el proceso “*Reserva en biblioteca con aforo completo*” (*Apartado 4.4.7.1*), que le impide realizar la reserva.

### Cancelar reserva

Una reserva puede ser cancelada por el mismo alumno que la ha realizado, o por un empleado de la biblioteca por algún motivo de peso (reglamento de la biblioteca quebrantado, necesidad de reservar una sala para una reunión importante, por tener que realizar alguna tarea de mantenimiento en la zona de la biblioteca en la que se encuentra un puesto concreto, …).

### Cambio en reserva

Un cambio en una reserva puede hacerlo tanto un empleado como un alumno que quiera cambiar la duración, o la hora de llegada, por ejemplo. En ese caso, el cambio en la reserva implica cancelar la reserva actual, y repetir los pasos para realizar una nueva reserva (aportando los datos necesarios). Un cambio en la reserva implica por tanto el proceso “*Cancelar reserva*” existente (*Apartado 4.4.2*), y una nueva “*Reserva en biblioteca*” (*Apartado 4.4.1*).

### Reserva de sala

Como se puede observar en el diagrama anterior, la “*Reserva de sala*” implica una “*Reserva en biblioteca*” (*Apartado 4.4.1*). Por ello, el alumno que requiera la sala en cuestión deberá aportar la misma información: datos personales, qué sala quiere reservar, el día, la hora, y la duración de la reserva. Además, cada uno de los usuarios que quieran estar en esa sala, deberán hacer la misma reserva, para tener así un mejor seguimiento de los usuarios que disfrutan de la estancia. Una vez se ejecute el proceso, el usuario tiene la posibilidad de ejecutar la “*Reserva de recurso*” (*Apartado 4.4.6*).

### Reserva de puesto

Siguiendo la misma lógica que para la “*Reserva de sala*” (*Apartado 4.4.4*), cualquier usuario que quiera hacer una reserva de un puesto bibliotecario deberá realizar una “*Reserva en biblioteca*” (*Apartado 4.4.1*) aportando sus datos personales, qué puesto quiere reservar, cuando (día y hora), y la duración. Al igual que con la “*Reserva de sala*” (*Apartado 4.4.4*), si ha elegido la opción de reservar un puesto de estudio y ejecuta el proceso, el usuario tiene la posibilidad de ejecutar la “*Reserva de recurso*” (*Apartado 4.4.6*).

### Reserva de recurso

Como se ha explicado anteriormente, las bibliotecas de la universidad ofrecen diferentes servicios que son atractivos para los usuarios del sistema. No sólo pueden reservar puestos o salas para trabajar, sino también hacer uso de los préstamos de libros y ordenadores. Asociada a la “*Reserva de puesto*” (*Apartado 4.4.5*) o a la “*Reserva de sala*” (*Apartado 4.5.5*), el usuario puede añadir una “*Reserva de recurso*” (libro para consultar en el puesto o sala, ordenador, etc.). Para realizar este tipo de reservas, el usuario deberá aportar su información persona, el día y la hora a la que recoge el recurso, e información sobre qué recurso ejecutando el o los procesos “*Reserva de recurso libro*” (*Apartado 4.4.7*) y/o “*Reserva de recurso ordenador*” (*Apartado 4.4.8*).

### Reserva de recurso libro

Como se ha explicado anteriormente, el proceso “*Reserva de recurso libro*” implica una “*Reserva de recurso*” (*Apartado 4.4.6*). Así, cualquier usuario puede elegir reservar un libro tras hacer una “*Reserva de puesto*” (*Apartado 4.4.5*) o “*Reserva de sala*” (*Apartado 4.4.4*) y disfrutar de la lectura en su sitio reservado. Al hacer la reserva del recurso, el usuario debe aportar su información personal, el ordenador que coge prestado, cuando, y la duración.

### Reserva de recurso ordenador

Siguiendo la misma lógica que para la “*Reserva de recurso libro*” (*Apartado 4.4.7*), el usuario que quiera utilizar un ordenador debe ejecutar el proceso “*Reserva de recurso ordenador*” no sin antes haber ejecutado el proceso “*Reserva de puesto*” (*Apartado 4.4.5*) o “*Reserva de sala*” (*Apartado 4.4.4*) para disfrutar del mismo. En este caso, debe aportar su información personal, el ordenador que coge prestado, cuando, y la duración.

### Casos particulares

#### Reserva biblioteca con aforo completo

Por motivos obvios, en épocas de exámenes (parciales o de convocatorias ordinaria y extraordinaria), cabe la posibilidad de que haya ningún sitio libre en la biblioteca. Por ello, si un usuario pretende realizar una “*Reserva en biblioteca*” (*Apartado 4.4.1*), el sistema no le deja al ejecutar el proceso “*Reserva en biblioteca con aforo completo*”. En ese caso, sabiendo que el alumno solicita un puesto o sala con ciertos recursos, el sistema valorará la posibilidad de ofrecerle un puesto o sala con características similares en otra biblioteca de la universidad en función de la disponibilidad.

#### Alumno se retrasa

Pongamos que se da la situación en la que un alumno tiene una “*Reserva en biblioteca*” (*Apartado 4.4.1*) a una hora determinada. Si el alumno llega más tarde de la hora indicada, el sistema cancela la reserva ejecutando el proceso “*AlumnoSeRetrasa*”. Por tanto, el alumno tendría que realizar de nuevo otra “*Reserva en biblioteca*” (*Apartado 4.4.1*) aportando los datos pertinentes.

## Especificación de requisitos

En este apartado se listarán los requisitos funcionales y no funcionales de la solución propuesta en este proyecto, y que se verán reflejados en el prototipo desarrollado. Durante el análisis hecho previamente en el trabajo, y específicamente en el *4.3 Recursos,* se han detallado situaciones o recursos que pueden intervenir en la gestión de una biblioteca. Sin embargo, el prototipo desarrollado se centra en la reserva de puestos de estudio y salas por parte de los alumnos. Con este objetivo, se detallan a continuación los requisitos funcionales y no funcionales del prototipo a desarrollar:

### Requisitos funcionales

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Ilustración 11: Tabla de requisitos funcionales. Elaboración propia

### Requisitos no funcionales

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Ilustración 12: Tabla de requisitos no funcionales. Elaboración propia

## Análisis de seguridad 🡪 ANEXO

Este proyecto contiene una serie de comunicaciones entre elementos de la arquitectura desarrollada en Internet. Por ello, es importante que satisfagan las dimensiones de seguridad:

* **Autenticidad:** garantiza que el usuario autorizado es quien dice ser. En este proyecto, se cumple puesto que los usuarios acreditan quiénes son mediante el identificador del carné de la universidad.
* **Confidencialidad:** la información debe ser accesible a los usuarios autorizados. En este caso, todo usuario registrado en el sistema tiene acceso a la información que le dejan sus permisos (reservas, datos personales, entre otros).
* **Integridad:** los datos deben mantenerse tal y como los introduce un usuario autorizado en el sistema. En un cambio de reserva de este proyecto, el sistema varía la disponibilidad de los recursos. Sin embargo, ningún dato personal o información que no tenga relación directa con una reserva no es modificado por el sistema.
* **Disponibilidad:** la información debe estar siempre disponible para los usuarios autorizados. En este caso, se garantiza que el sistema permite realizar reservas sin ningún problema, y acceder datos relativos a esas reservas.
* **Trazabilidad:** permite registrar e identificar las acciones que se hacen sobre los datos. Uno de los objetivos principales de este *software* es la trazabilidad y seguimiento que permite tener sobre los diferentes recursos del sistema y los actores que intervienen.

En este proyecto de fin de carrera se desarrolla un Servicio Web. Esto implica que hay comunicaciones entre *Browser* (cliente) y servidor mediante protocolo *HTTP*. Una medida de protección puede ser su cambio, por el uso del protocolo por *HTTPS*, para evitar que otros usuarios puedan interceptar la información confidencial en esa comunicación en la relación cliente-servidor. Además, si se quisiera implementar el *software* integrándolo en el ecosistema de programas y aplicaciones de la universidad, se añadiría más seguridad al requerir que todos los usuarios del sistema tuvieran que hacer *login* con las credenciales de la institución. Incluso, se aplicarían todas las medidas de seguridad informática que tiene la universidad en sus redes, como *firewalls* y otros filtros.

Análisis desde la perspectiva del RGPD

Este prototipo se ha desarrollado utilizando información ficticia, por tanto, no habría ningún problema de tratamiento de datos personales. Si finalmente la universidad estuviera interesada en implementar una versión final, tampoco debería tener ningún problema puesto que la información personal que manejaría la aplicación de alumnos y personal de la biblioteca es la misma a la que ya tiene acceso actualmente.

# Arquitectura y diseño

En este capítulo, como su propio nombre indica, se precisa cómo se diseña e implementa el *software* en cuestión. Se presenta en una primera aproximación la arquitectura del sistema y, en un segundo lugar, se analiza con detalle el *BackEnd* por un lado, y el *FrontEnd* por otro.

## Arquitectura del sistema

Diagrama

Descripción generada automáticamente El sistema está formado por dos grandes elementos: un *BackEnd* o lógica de negocio, y por un *FrontEnd* o interfaz de usuario. El primero, está formado por un servidor gracias y una base de datos de donde se obtienen los datos procesados. El segundo, permite al usuario interactuar con el sistema. El siguiente diagrama muestra la estructura de la arquitectura:

Ilustración 13: Diagrama de despliegue de la arquitectura del sistema. Elaboración propia

En el diagrama anterior también se pueden ver los flujos de solicitudes que realiza el cliente mediante *HTTP*, así como las respuestas del servidor. Tras las solicitudes del cliente, se muestra la información requerida gracias a la interfaz de usuario, para que pueda interactuar de forma más sencilla. Además, se puede observar como el servidor de *Play* *Framework* accede a la base de datos y solicita datos mediante *queries*, que le son devueltos.

## Diseño del BackEnd

Como se ha explicado anteriormente, el software de este proyecto se basa en un Servicio Web *RESTful* que sigue una arquitectura estilo ROA. Por ello, en este apartado se siguen los pasos enumerados en la sección *3.1.1.4 Arquitectura Orientada a Recursos (ROA).*

### 5.2.1 Definición de recursos Y URIS

En la aplicación desarrollada en este proyecto, se tienen en cuenta una serie de recursos que intervienen en los procesos de reserva de un puesto bibliotecario. Dichos recursos Web son los siguientes:

* **Usuario:** este recurso hace referencia a cualquier persona que utilice el *software* de reservas, ya sea un alumno, un profesor o un trabajador de la biblioteca
* **Usuarios:** el recurso en cuestión permite realizar gestiones sobre el conjunto de usuarios
* **Puesto:** se refiere a una mesa de estudio que puede ser reservada tanto por cualquier usuario para leer, estudiar, u otra actividad que esté dentro del reglamento de la biblioteca
* **Puestos:** permite hacer acciones sobre los diferentes puestos que tiene una biblioteca
* **Sala:** este recurso es una mesa de estudio más amplia que se encuentra cerrada dentro de cuatro paredes de cristal. Se utilizan para trabajos en grupo, y también puede ser reservada por cualquier usuario
* **Salas:** el recurso en cuestión da la posibilidad de interactuar con todas las salas que tiene una biblioteca
* **Reserva:** se refiere al hecho de guardar un puesto o sala en concreto y a la información relevante de la misma como es su hora, el usuario involucrado, puesto o sala reservado o reservada, …
* **Reservas:** este recurso engloba al conjunto de reservas que se llevan a cabo en una biblioteca por parte de los usuarios
* **Biblioteca:** al haber varias bibliotecas en el campus de la universidad, este recurso hace referencia a cada una de ellas
* **Bibliotecas:** como se ha explicado anteriormente, la universidad tiene diferentes bibliotecas y, este recurso, permite realizar operaciones sobre todas ellas
* **Recurso:** se refiere al conjunto de otros atributos que ofrece la biblioteca, como pueden ser los ordenadores, las regletas, libros de préstamo, o características del puesto (enchufes cercanos, ventanas, aire acondicionado, …)
* **Recursos:** al haber diferentes tipos de recursos y varias unidades de los mismos, este recurso permite realizar modificaciones sobre ellos.

### Identificación de los recursos Web 🡪 juntar con apartado anterior

En este apartado se detallan las diferentes *URIs* que se utilizan en el sistema para identificar a los recursos inequívocamente:

* **/usuarios**: recibe todas las solicitudes *HTTP* del conjunto de usuarios del sistema.
* **/usuarios/:usuarioID**: recibe todas las solicitudes *HTTP* de un usuario en concreto, que se identifica con su identificador o *ID.*
* **/reservas**: recibe todas las solicitudes *HTTP* del conjunto de reservas del sistema.
* **/reservas/:reservaID**: recibe todas las solicitudes *HTTP* de una reserva concreta, que se identifica con su identificador o *ID.*
* **/bibliotecas**: recibe todas las solicitudes *HTTP* del conjunto de bibliotecas de la universidad que están en el sistema.
* **/bibliotecas/:bibliotecaID**: recibe todas las solicitudes *HTTP* de una biblioteca concreta de la universidad, que se identifica con su identificador o *ID.*
* **/bibliotecas/:bibliotecaID/puestos**:recibe todas las solicitudes *HTTP* del conjunto de puestos de una biblioteca concreta, que se identifica con su identificador o *ID.*
* **/bibliotecas/:bibliotecaID/puestos/:puestoID**: recibe todas las solicitudes *HTTP* de un puesto en concreto identificado con su identificador o *ID*, que pertenece a una biblioteca concreta, identificada a su vez con su identificador o *ID.*
* **/bibliotecas/:bibliotecaID/puestos/:puestoID/recursos**: recibe todas las solicitudes *HTTP* del conjunto de recursos de un puesto en concreto identificado con su identificador o *ID*, que pertenece a una biblioteca concreta, identificada a su vez con su identificador o *ID.*
* **/bibliotecas/:bibliotecaID/puestos/:puestoID/recursos/:recursoPuestoID**: recibe todas las solicitudes *HTTP* de un recurso identificado por su identificador o *ID*, de un puesto en concreto identificado con su identificador o *ID*, que pertenece a su vez a una biblioteca concreta, que se identifica con su identificador o *ID.*
* **/bibliotecas/:bibliotecaID/salas**:recibe todas las solicitudes *HTTP* del conjunto de salas de una biblioteca concreta, que se identifica con su identificador o *ID.*
* **/bibliotecas/:bibliotecaID/salas/:salaID**: recibe todas las solicitudes *HTTP* de una sala en concreto identificado con su identificador o *ID*, que pertenece a una biblioteca concreta, identificada a su vez con su identificador o *ID.*
* **/bibliotecas/:bibliotecaID/salas/:salaID/recursos**: recibe todas las solicitudes *HTTP* del conjunto de recursos de una sala en concreto identificado con su identificador o *ID*, que pertenece a una biblioteca concreta, identificada a su vez con su identificador o *ID.*
* **/bibliotecas/:bibliotecaID/salas/:salaID/recursos/:recursoSalaID**: recibe todas las solicitudes *HTTP* de un recurso identificado por su identificador o *ID*, de una sala en concreto identificado con su identificador o *ID*, que pertenece a su vez a una biblioteca concreta, que se identifica con su identificador o *ID.*

### Representaciones utilizadas

En este proyecto se utiliza *JSON* para la representación de recursos a través del sistema. Se integran la referencia a las *URIs*, los recursos apuntan los unos a los otros, etc … Además, se ha utilizado *JSON* en vez de *XML* ya que es más intuitivo y representa la información requerida de una forma más fácil para el usuario. Así, dinamiza el uso del sistema por parte de los usuarios.

### Establecimiento de interfaz 🡪 poner la más completa, y las demás a anexo

En este apartado se muestran, a modo de tablas, las comunicaciones entre cliente y servidor mediante métodos de *HTTP* y las respuestas de cada uno de los recursos definidos en el *Apartado 5.2.1*. En esas tablas, a parte del propio método, se detalla el cuerpo de la solicitud, la semántica, el código de respuesta, así como su cuerpo. Se presentan unos ejemplos más representativos, el resto de las tablas se pueden encontrar en el *Anexo III* “*Tablas API de recursos*”.

#### Usuarios

**Recurso:** usuarios

**URI:** /usuarios

Tabla . Recurso usuarios

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método HTTP | Cuerpo de la solicitud | Semántica | Código Respuesta | Cuerpo de la respuesta |
| GET | - | Listado de los usuarios | 200 – OK | Listado de usuarios |
| 400 – Bad request | - |
| 404 – Not Found | - |
| 500 – Internal Server Error | - |
| POST | Información del usuario | Crear un usuario | 201 – Created  (El ID se devuelve en la cabecera Location) | El usuario creado con todos sus datos |
| 400 – Bad request | - |
| 500 – Internal Server Error | - |
| PUT | - | N/A | 405 – Method not allowed | - |
| DELETE | - | N/A | 405 – Method not allowed | - |
| PATCH | - | N/A | 405 – Method not allowed | - |

El primer método *HTTP* estudiado es *GET*. Este método permite al cliente listar todos los usuarios del sistema. Como se puede ver en la tabla, si todo sale bien la respuesta del servidor será el código “*200-OK*”, y el cuerpo de la respuesta será precisamente la lista de usuarios en formato *JSON* para poder ser visualizado por el cliente. Si nos fijamos en las siguientes líneas, vemos como pueden surgir algunos errores. En esos casos, se mostrarían los códigos “*400-Bad Request*” si la solicitud se hace de manera incorrecta, “*404-Not Found*” si no encuentra los recursos solicitados, o “*500-Internal Server Error*” si hay un error del servidor.

El método *POST* permite crear un usuario nuevo adjuntando en el cuerpo de la solicitud del *JSON* la información de este (nombre, apellidos, grado). En los métodos *POST*, los *ID* y *URL* son asignados automáticamente por el servidor, quien responde el código “*201-Created*” si ha ido todo bien y se ha creado dicho usuario, y devuelve la *URI* que enlaza con el nuevo usuario. Si ha habido algún error al ahora de realizar la solicitud o hay algún fallo del servidor, se muestran los códigos “*400-Bad Request*” o “*500-Internal Server* *Error*” respectivamente, como se ha explicado anteriormente para el método *GET*.

Para cambiar o borrar algún usuario en concreto, se ha ir a la URI del usuario en cuestión, por ello no se permiten los métodos *PUT*, *DELETE*, o *PATCH* para este recurso.

#### Biblioteca

**Recurso:** biblioteca

**URI:** /bibliotecas/:bibliotecaID

Tabla . Recurso biblioteca

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método HTTP | Cuerpo de la solicitud | Semántica | Código Respuesta | Cuerpo de la respuesta |
| GET | - | Toda la información de una biblioteca concreta | 200 - OK | Información de la biblioteca en cuestión con sus puestos, salas y su disponibilidad |
| 400 – Bad request | - |
| 404 – Not Found | - |
| 500 – Internal Server Error | - |
| POST | - | N/A | 405 – Method not allowed | - |
| PUT | Información para modificar de la biblioteca específica | Modificar una biblioteca | 200 - OK | La biblioteca con los cambios realizados |
| 400 – Bad request | - |
| 500 – Internal Server Error | - |
| DELETE | - | Borrado de una biblioteca concreta | 200 - OK | - |
| 400 – Bad request |
| 500 – Internal Server Error |
| PATCH | Información para modificar la disponibilidad de la biblioteca | Añadir o eliminar la disponibilidad de una biblioteca | 200 - OK | La disponibilidad añadida o borrada |
| 400 – Bad request | - |
| 500 – Internal Server Error | - |

El recurso “*Biblioteca*” es un poco diferente al explicado anteriormente. El primer método *GET* hace lo mismo que para el recurso anterior: lista la información de una biblioteca (ID, URI, nombre, descripción, disponibilidad, puestos de estudio, salas). Además, también muestra los códigos “*200-OK*” si va todo bien y devuelve la información de la biblioteca en el objeto *JSON*, “*400-Bad Request*” si se ha solicitado el recurso de forma incorrecta, “*404-Not Found*” si no se encuentra, y “*500-Internal Server Error*” si surge un error en el servidor.

Sin embargo, la cosa cambia con otros métodos. En este caso, el método *PUT* permite cambios en la información de la biblioteca específica. Por ejemplo, permite cambiar el nombre o la descripción si se añade al cuerpo de la solicitud. Como anteriormente, si sale todo bien se devolverá el código “*200-OK*” y la información modificada de la biblioteca en cuestión, y si sale algo más, surgirán los códigos “*400-Bad Request*”, o “*500-Internal Server Error*” en función del error.

El método *DELETE* permite el borrado de una biblioteca específica de la universidad del sistema. Una vez más, si todo sale bien se devuelve el código “*200-OK*”, y si no, los errores “*400-Bad Request*”, o “*500-Internal Server Error*” en función del error.

El método *PATCH* se encarga de modificar la disponibilidad que tiene una biblioteca concreta en función del aforo. Por ejemplo, si se desea añadir una hora de disponibilidad, se debe adjuntar ese cambio que se requiere en el cuerpo de la solicitud, y un método *ADD* de añadido. Si se desea borrar una cierta disponibilidad de una hora, en el objeto *JSON* debe aparecer esa disponibilidad de una hora y un método *REMOVE* para eliminarla. De nuevo, si todo sale bien se devuelve el código “*200-OK*”, y si no, los errores “*400-Bad Request*”, o “*500-Internal Server Error*” en función del error.

#### Reserva 🡪 **QUITAR UPDATE, Y MODIFY, DIRECTAMETNE BORRAR RESERVA Y CREAR NUEVA**

Recurso: Reserva

URI: /reservas/:reservaID

Tabla . Recurso reserva

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método HTTP | Cuerpo de la solicitud | Semántica | Código Respuesta | Cuerpo de la respuesta |
| GET | - | Toda la información de una reserva | 200 - OK | Información de una reserva |
| 400 – Bad request | - |
| 404 – Not Found | - |
| 500 – Internal Server Error | - |
| POST | - | N/A | 405 – Method not allowed | - |
| PUT | - | No se permiten las modificaciones | 405 – Method not allowed | - |
| DELETE | - | Borrado de una reserva específica y devolución de las disponibilidades a los recursos reservados | 200 - OK | - |
| 400 – Bad request |
| 500 – Internal Server Error |
| PATCH | - | N/A | 405 – Method not allowed |  |

Para el recurso “Reserva”, el método *GET* ayuda a ver la información relevante a dicha reserva concreta (ID, URI, disponibilidad, ID del usuario que realiza la reserva, ID de la biblioteca en la que se ha hecho la reserva, puesto o sala en cuestión, y recursos que se han guardado). Esa información se mostrará si va todo bien y se devuelve el código “*200-OK*”, como de costumbre. Si no, los errores “*400-Bad Request*”, “*404-Not Found*”, o “*500-Internal Server Error*” aparecerán en función del error.

*DELETE* es el método mediante el cual se puede borrar una reserva específica de recurso perteneciente a un puesto o sala. Nuevamente, se devuelve el código “*200-OK*” si va todo bien y, los errores “*400-Bad Request*”, “*404-Not Found*”, o “*500-Internal Server Error*” si no.

Este recurso no permite ni *PUT*, ni *PATCH*, puesto que al ser reservas de disponibilidad reducida (cada media hora), el cliente deberá borrar una reserva y hacer otra nuevamente, en vez de modificar la actual.

### Base de Datos

E/R 🡪 Recordar subrayar los IDs como PK

SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS

La base de datos permite acceder a los datos que son procesados por el sistema para poder tanto realizar peticiones como generar las respuestas. Gracias al listado de recursos desarrollado en el apartado *5.2.1 Definición de recursos*, se han creado un conjunto de tablas con los atributos necesarios para almacenar la información de cada recurso. Las tablas creadas surgen del siguiente diagrama entidad-relación:

E/R

Las tablas creadas son las siguientes: **🡪Las tablas normales nada, sóo explicar las de relación, y las herencias**

* **Usuario:** almacena el conjunto de usuarios del sistema con su información y un identificador único.
* **Biblioteca:** representa las bibliotecas que tiene la universidad con sus atributos y un identificador.
* **disponibilidadBiblioteca:** tabla que engloba las franjas de disponibilidad de cada biblioteca de la universidad para poder hacer reservas.
* **Reserva:** representa el conjunto de reservas que se han llevado a cabo.
* **Puesto:** conjunto de puestos de estudio que tiene la universidad, con atributos e identificadores.
* **Sala:** esta tabla recopila todas las salas que tiene la universidad junto con sus atributos e identificadores.
* **disponibilidadSala:** representa las franjas de disponibilidad de las salas de la universidad.
* **disponibilidadPuesto:** en este caso engloba las franjas de disponibilidad de los puestos de estudio.
* **reservaRecurso:** esta tabla recopila las reservas de recursos.
* **recursoPuesto:** representa los recursos que tienen los puestos de las bibliotecas..
* **recursoSala:** recoge el conjunto de recursos que tienen las salas.
* **disponibilidadRecursoPuesto:** recopila franjas de disponibilidad de los recursos que tienen los puestos de las bibliotecas.
* **disponibilidadRecursoSala:** recoge las franjas de disponibilidad de los recursos de las salas.

## Diseño del FrontEnd

Para el desarrollo de la interfaz de usuario, se utiliza la herramienta *FreeMarker*.

# Implementación y pruebas

**Referencia al repositorio de software**

Se indicará la URL del repositorio en que está el software para, si procede, que el tribunal pueda probarlo en sus propias máquinas.

**Manuales 6.4 🡪 pantallazos de los pasos para hacer una reserva 🡪 anexo.** FIJARME EN ARRADI PARA VER UN EJEMPLO DE RESERVA Y QUE SE VEA UN HILO CONDUCTOR

En caso de que no tenga sentido incluir en el fichero README.md del repositorio alguno de los siguientes manuales: instalación, operación o usuario, se incluirá en la memoria del TFG.

CAPITULO 6: BACKEND Y FRONTEND

SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS

En este capítulo se presenta con más detalles la implementación del prototipo, explicando tanto la parte del *FrontEnd* y del *BackEnd*, así como las herramientas que se han utilizado.

## Implementación *FrontEnd*

Como se ha descrito anteriormente, el *FrontEnd* se ha implementado…..

## Implementación *BackEnd*

Este apartado sirve para entender mejor los diferentes componentes de la aplicación. La estructura de la aplicación se conforma de cuatro grandes partes: *Entities*, *Services*, *Controllers*, y *Routes*.

### Entities 🡪 resaltar clases importantes CambioTipo, Shorts, Herencias

Hay una clase “entidad” por cada recurso que crea objetos con sus atributos para poder ser utilizados en las comunicaciones *HTTP*: Reserva, Usuario, Biblioteca, Puesto, Sala, RecursoPuesto, RecursoSala, CambioHorario, CambioTipo, ReservaPuesto, ReservaSala.

Además, también están las entidades “*Short*”: BibliotecaShort, PuestoShort, SalaShort, RecursoPuestoShort, RecursoSalaShort, RecursoShort. Estas clases permiten mostrar una información más resumida, sin tener que listar todos los atributos de cada recurso.

Cabe destacar que todas las “entidades” heredan de la clase “RecursoWeb” puesto que heredan la *URI* que está en esa clase. Además, las clases *CambioHorario* y *CambioTipo* se encargan de gestionar las disponibilidades. *CambioHorario* se encarga de almacenar la información de los cambios que hay que hacer, y *CambioTipo* sirve para saber si el cambio es de añadido de disponibilidad o de borrado de disponibilidad.

### Services

Estas clases sirven para gestionar los accesos a base de datos *MySQL*. Por ello, hay una clase principal *ConexionBD* (tiene el *login*, la ubicación de la base de datos, y realiza la conexión), y las clases para cada recurso: *UsuarioBD*, *ReservaBD*, *BibliotecaBD*, *PuestoBD*, *SalaBD*, *RecursoPuestoBD*, y *RecursoSalaBD*. En cada una de esas clases, hay un método que se encarga del método *HTTP* correspondiente: para el método *GET* hace un *Select*, para el método *POST* hace un *Insert* en la base de datos, para los métodos *PATCH* y *PUT* hace un *Update* de la base de datos, y para el método *DELETE* hace un *Delete* en la base de datos.

### Routes

La clase *Routes* lista los métodos *HTTP*, la clase a la que llama cada uno de los métodos, y la ruta de acceso al recurso en cuestión.

### Controllers

En esta clase también hay un “controlador” por recurso: *BibliotecaController*, *UsuarioController*, *ReservaController*, *RecursoPuestoController*, *RecursoSalaController*, *PuestoController*, *SalaController*. Juegan el papel de intermediarios entre la clase *Routes* y el usuario. Cuando recibe la petición desde *Routes*, se encarga de procesar la información introducida por el usuario, y ejecuta el método correspondiente al método *HTTP* que le ha pasado *Routes*.

## Se indicará el IDE, *frameworks*, simuladores, etc. que se han utilizado.

Como se ha detallado anteriormente, se ha utilizado un entorno formado por los siguientes programas y tecnologías: Play Framework para Servicio Web, MySQL para base de datos, Freemarker para interfaz de usuario. Además, se utiliza el IDE IntelliJ para el desarrollo de la aplicación por su facilidad de uso y ayuda al code develpment.

## Pruebas 🡪 reflejo de que se han probado toda la API (delete create etc) con Psotman, y también se ha documentado con Swagger

En el Servicio Web de este trabajo se ha usado *Postman* para realizar pruebas durante y después del desarrollo del prototipo. El objetivo era ir verificando las entradas de la API, que funciona la conexión con la base de datos, y que se muestra la información requerida en cada solicitud realizada. Por ejemplo, se ha realizado la siguiente prueba:

**POST para que se vea el json pasado y el código de respuesta OK 200… (CAPTURAS)**

# Conclusiones y líneas futuras

Este proyecto surgió de una situación prácticamente nueva para todo el mundo. La pandemia del coronavirus azotó el mundo y los gobiernos impusieron unas medidas de control de aforos en espacios cerrados para evitar la propagación del virus. En este proyecto tenía como objetivo el desarrollo de un *software* de gestión de una biblioteca.

Con el objetivo de mejorar el método de reserva primigenio rudimentario y poco eficiente que implantó la universidad, se realizó un análisis del dominio y estudio de requisitos que debía cumplir la aplicación analizando las necesidades de los actores del proceso de reserva. En pleno desarrollo de ese análisis, la universidad implementó un *software* de gestión que también tuvo que ser analizado para intentar corregir sus debilidades y aportar funcionalidades extra. Para ello, se realizaron una serie de entrevistas tanto a alumnos como a empleados de la biblioteca para identificar problemas recurrentes en el proceso de reserva, analizando el *software* junto a ellos para destacar los puntos fuertes y los mejorables.

Una vez hecho el análisis y el diseño, se implementó el prototipo. Para ello se utilizó un Servicio Web *RESTful* gracias a *Play* *Framework*. *También se ha contado con una base de datos MySQL para almacenar datos y se ha utilizado el programa Postman para probar los diferentes procesos de la aplicación desarrollada.* Se han implementado los métodos *HTTP* *GET*, *POST*, *PATCH* y *DELETE* para poder interactuar con los recursos del sistema. De forma que se han cumplido los objetivos establecidos al inicio del proyecto:

* Realizar un prototipo que permita la creación de reservas de sitios en la biblioteca de la EPS para gestionar el aforo y los recursos
* Diseñar un Servicio Web *RESTful* que permita la creación de una reserva en la biblioteca, además de altas, bajas y modificaciones de la misma
* Creación de representaciones *JSON* para los recursos del sistema, para permitir la integración del servicio de otras aplicaciones (escritorio, aplicaciones Web, o aplicaciones móviles).

Siguiendo el Modelo de Madurez de Richardson explicado en el Apartado XXXXXX “xxxxxxx” se puede clasificar el prototipo creado en el nivel XXXX (2 sin interfaz, 3 con interfaz)

Durante el desarrollo de este proyecto, han surgido problemas que han lo han obstaculizado. Por ejemplo, hubo fallos técnicos en la conexión a la base de datos o a la hora de desarrollar el código del prototipo, pero también problemas en la parte lógica del proyecto como por ejemplo la inclusión del software de gestión por parte de la universidad que provocó un cambio de rumbo en la realización del proyecto.

## Líneas futuras

El *software* desarrollado es un prototipo para demostrar la viabilidad de la propuesta (diseño de un servicio RESTful que, mejorando los procesos anteriores, pudiera ser integrable con el *software* de la universidad). Por lo tanto, queda trabajo por delante para mejorar completamente todo lo que ha sido analizado en los capítulos anteriores. Por ejemplo, algunas de las funcionalidades más interesantes que podrían incluirse son:

* Que la aplicación tuviera en cuenta los horarios de los equipos de limpieza, para que los recursos estuvieran higienizados y listos para ser reservados y usados por los usuarios.
* Que los alumnos pudieran escribir manualmente la franja deseada para la reserva en vez de hacer varias reservas de franjas concretas.
* Que el proceso de verificación de presencialidad fuera dinámico, siendo un código asociado a cada reserva y no un código escaneable común.
* Que haya un *logout* y los alumnos tengan que realizarlo al salir de la biblioteca y el sistema verifique cada cierto tiempo qué sitios reservados se han quedado vacíos antes de que acabe la reserva realizada, para evitar cuellos de botella y esperas innecesarias si un alumno se va antes de que acabe su reserva.
* Que haga un seguimiento de todos los alumnos que están en una sala en grupo, en vez de registrar sólo al alumno que hace la reserva de la sala y por tanto haya penalizaciones si asisten menos alumnos que los mínimos para poder tener una sala.
* Que la aplicación fuera más global, y se aplicara a otro tipo de reservas como reserva de instalaciones deportivas del campus, sitios en los autobuses, o en laboratorios.
* Que los alumnos pudieran marcar ciertos sitios en favoritos para acceder a ellos más fácilmente, y no tener que recorrer la larga lista de recursos disponibles hasta encontrar el deseado.
* Que la aplicación estuviera integrada con el ecosistema de la universidad, y por tanto los usuarios tengan acceso a otros servicios en la misma aplicación, como por ejemplo acceso a sus asignaturas.

# Bibliografía

Bibliografía citada en la memoria. Seguirá el [formato APA](https://apastyle.apa.org/).

Ejemplos de referencias:

1. Grady, J. S., Her, M., Moreno, G., Perez, C., & Yelinek, J. (2019). Emotions in storybooks: A comparison of storybooks that represent ethnic and racial groups in the United States. Psychology of Popular Media Culture, 8(3), 207–217. <https://doi.org/10.1037/ppm0000185>
2. Jerrentrup, A., Mueller, T., Glowalla, U., Herder, M., Henrichs, N., Neubauer, A., & Schaefer, J. R. (2018). Teaching medicine with the help of “Dr. House”. PLoS ONE, 13(3), Article e0193972. <https://doi.org/10.1371/journal.pone.0193972>
3. Schaefer, N. K., & Shapiro, B. (2019, September 6). New middle chapter in the story of human evolution. Science, 365(6457), 981–982. <https://doi.org/10.1126/science.aay3550>
4. Carey, B. (2019, March 22). Can we get better at forgetting? The New York Times. <https://www.nytimes.com/2019/03/22/health/memory-forgetting-psychology.html>
5. Rabinowitz, F. E. (2019). Deepening group psychotherapy with men: Stories and insights for the journey. American Psychological Association. <https://doi.org/10.1037/0000132-000>
6. Aron, L., Botella, M., & Lubart, T. (2019). Culinary arts: Talent and their development. In R. F. Subotnik, P. Olszewski-Kubilius, & F. C. Worrell (Eds.), The psychology of high performance: Developing human potential into domain-specific talent (pp. 345–359). American Psychological Association. <https://doi.org/10.1037/0000120-016>
7. Harvard University. (2019, August 28). Soft robotic gripper for jellyfish [Video]. YouTube. <https://www.youtube.com/watch?v=guRoWTYfxMs>
8. APA Databases [@APA\_Databases]. (2019, September 5). Help students avoid plagiarismWeb emoji of crossing hands and researchers navigate the publication process. More details available in the 7th edition @APA\_Style table [Tweet]. Twitter. <https://twitter.com/APA_Databases/status/1169644365452578823>
9. Fagan, J. (2019, March 25). Nursing clinical brain. OER Commons. Retrieved September 17, 2019, from <https://www.oercommons.org/authoring/53029-nursing-clinical-brain/view>
10. National Institute of Mental Health. (2018, July). Anxiety disorders. U.S. Department of Health and Human Services, National Institutes of Health. <https://www.nimh.nih.gov/health/topics/anxiety-disorders/index.shtml>
11. Woodyatt, A. (2019, September 10). Daytime naps once or twice a week may be linked to a healthy heart, researchers say. CNN. <https://www.cnn.com/2019/09/10/health/nap-heart-health-wellness-intl-scli/index.html>
12. World Health Organization. (2018, May 24). The top 10 causes of death. <https://www.who.int/news-room/fact-sheets/detail/the-top-10-causes-of-death>

# Anexo I

# Caso de uso de la aplicación Affluences desde el punto de vista del alumnado

En este anexo se estudiará cómo ve un alumno la aplicación *Affluences* y los procesos que debe hacer para poder acudir a un puesto de la biblioteca.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamenteEn cuanto a la funcionalidad ofrecida al alumnado, una vez accede a la cuenta con el correo corporativo y seleccionado la biblioteca de la Escuela Politécnica Superior, el usuario puede reservar el sitio deseado. Para ello, debe seleccionar el día en cuestión, la hora a la que quiere hacer la reserva, su duración (las opciones son: 30mins, 1h, 1h30, 2h, 2h30, 3h), y por supuesto la sala de estudio o sitio requerido. Para dar más facilidades al alumno, la aplicación tiene adjunta una foto de un esquema hecho a mano de las diferentes zonas de la biblioteca donde aparecen numerados los sitios. La siguiente imagen muestra el proceso de modificar una reserva ya hecha, que es idéntico al de hacer una nueva:

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Figura X: Proceso de reservar un puesto en *Affluences* por un alumno

Fuente: Elaboración propia

Una vez se ha hecho la reserva en la aplicación, se recibe un mail que debe ser confirmado para finalizar el proceso de la reserva. Si no se ha confirmado la reserva en un plazo de media hora, el sitio o sala de estudio en cuestión vuelve a estar disponible.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamenteInterfaz de usuario gráfica, Aplicación, Sitio web

Descripción generada automáticamenteSuponiendo que se ha confirmado y que la hora de la reserva se acerca, el alumno debe validar su presencia desde diez minutos antes de ésta a través de un código *QR*. Éste está impreso y situado en la puerta de la biblioteca, y la validación de presencialidad consiste en que el alumno en cuestión debe acceder a la aplicación *Affluences* y escanearlo con la cámara del móvil. Así, el sitio aparecerá como ocupado. Si no se valida la presencia en esos diez minutos de cortesía, el sitio volvería a aparecer disponible para otros alumnos. Las siguientes imágenes muestran dos reservas hechas por un alumno. Una de ellas ha sido cancelada por no confirmar la presencialidad en el margen que tiene el alumnado, y la otra fue confirmada:

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* confirmada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

Figura X: Reservar de un puesto en *Affluences* cancelada

Fuente: Elaboración propia

En esta versión, el alumno puede abandonar el puesto de trabajo antes de que acabe la reserva y no lo notificaría en la aplicación, por tanto, ese sitio se queda libre e inutilizado. Además, si quiere quedarse más de tres horas, tendría que renovar la reserva confirmando de nuevo su presencialidad en la puerta.

En cuanto al equipo de limpieza, desinfectan todos los sitios vacíos en los siguientes turnos establecidos: primera hora de la mañana, después de comer, y a media tarde. Esto significa que se puede dar la situación en que un alumno abandone su puesto, y otro se ponga en su lugar sin haberlo desinfectado previamente.

# Anexo II

# Caso de uso de la aplicación Affluences desde el punto de vista del responsable de la biblioteca

El anexo II se centra en la perspectiva del bibliotecario. Se analizarán las funcionalidades que puede modificar en la aplicación *Affluences* y cómo gestionar las reservas llevadas a cabo por el alumnado.

La interfaz de administrador tiene varias funcionalidades añadidas para poder configurar ciertos parámetros de la aplicación. El personal de la biblioteca puede acceder a más o menos parámetros en función de su puesto de trabajo y por tanto de sus permisos. Primero veremos a los responsables que están gestionando el día a día en la biblioteca de la universidad, y más adelante la interfaz del administrador jefe que controla que todo vaya como es debido.

En la pantalla de inicio no aparece ninguna información específica, sólo un menú con diferentes opciones: “Reservas”, “Parámetros” y “Cuenta”. El primer menú tiene los dos submenús “Planning” y “Búsqueda de usuario”. El primero tiene una vista en modo lista y otra en modo calendario para ver todas las reservas en curso de forma visual según la hora y el día. Estas reservas pueden ser modificadas o añadidas por el administrador, pudiendo modificar los siguientes datos: el puesto, el correo electrónico corporativo del alumno, la hora de la reserva, y su duración. Además, el administrador puede acceder a cualquier reserva y ver esos datos para identificar quién reserva qué puesto. En añadido, El administrador tiene un apartado en el que buscar a un alumno o alumna por nombre o correo corporativo, y así tener un historial de todas sus reservas, así como algún comentario adjunto o incluso si ha sido bloqueado alguna vez. Éstos son penalizaciones que ponen los encargados de la biblioteca manualmente a alguien que no cumple las normas. En efecto, ya sea por incumplir el reglamento de la biblioteca o la normativa Covid-19 de la institución, el alumno puede ser bloqueado por un tiempo definido por el responsable.

En cuanto al administrador jefe, tiene más funcionalidades bajo su control. A parte de lo mencionado anteriormente, también puede acceder a información más relevante en cuanto a la gestión del servicio de biblioteca. Así, tiene un apartado de configuración en el cual puede modificar información como los horarios y días de apertura, el código de validación, crear una alerta si se llega a un porcentaje de aforo específico, mandar un mensaje a todos los alumnos como por ejemplo “Por causas ajenas a la universidad, permanecerá cerrada la biblioteca”, … También, puede modificar los recursos (añadir, borrar, cambiar sus características)

Además, también tiene un apartado de estadísticas donde puede visualizar de una forma muy sencilla, tanto información de análisis como de historial de reservas. En cuanto a los datos de análisis, podemos encontrar: el tiempo espera máximo de validación, número máximo de visitantes, número máximo de entradas a la biblioteca, número total de reservas, top de recursos reservados (salas, puestos), número total de consultas, día más consultado, … En cuanto a la estadística, se permite acceder a un historial de reservas, pudiendo filtrar por puesto o sala, alumno, … Y exportar esos datos a un archivo de hoja de cálculo, e incluso hacer gráficos visuales.

# Anexo III

# Tablas API de recursos

**Recurso:** usuario

**URI:** /usuarios/:usuarioID

Tabla . Recurso usuario

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método HTTP | Cuerpo de la solicitud | Semántica | Código Respuesta | Cuerpo de la respuesta |
| GET | - | Toda la información de un usuario concreto | 200 - OK | Información del usuario concreto y sus reservas |
| 400 – Bad request | - |
| 404 – Not Found | - |
| 500 – Internal Server Error | - |
| POST | - | N/A | 405 – Method not allowed | - |
| PUT | Información para cambiar del usuario específico | Modificar un usuario | 200 - OK | El usuario con los cambios realizados |
| 400 – Bad request | - |
| 500 – Internal Server Error | - |
| DELETE | - | Borrado de usuario específico | 200 - OK | - |
| 400 – Bad request |
| 500 – Internal Server Error |
| PATCH | - | N/A | 405 – Method not allowed | - |

El método *GET* permite al cliente ver la información relativa a un único usuario (id, URI nombre, apellidos, grado, reservas). Como se ha explicado en el apartado *5.2.4 Establecimiento de interfaz*, el código “*200-OK*” se muestra si todo ha ido correctamente, y la respuesta contendrá la información del usuario en formato *JSON*. Si hay algún error, aparecerán los códigos “*400-Bad Request*”, “*404-Not Found*”, o “*500-Internal Server Error*” en función de su naturaleza.

El método *PUT* facilita el cambio de datos con respecto a un usuario. Se puede modificar por ejemplo el nombre, el apellido, o su grado. Si ha ido todo bien, aparecerá el código “*200-OK*” y la respuesta será la información actualizada del usuario en formato *JSON*. Sino, aparecerán los errores con código “*400-Bad Request*” o “*500-Internal Server Error*” en función de dónde surja, si en el lado del cliente o en el del servidor.

*DELETE* permite al cliente borrar un usuario en especial. “*200-OK*” es el mensaje que aparece si todo sale correctamente, y si no sale uno de los dos errores habituales: “*400-Bad Request*” o “*500-Internal Server Error*”.

En esta ocasión no hay método *PATCH* ya que cambia la disponibilidad de algún recurso, y un usuario no tiene.

**Recurso:** bibliotecas

**URI:** /bibliotecas

Tabla 5. Recurso bibliotecas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método HTTP | Cuerpo de la solicitud | Semántica | Código Respuesta | Cuerpo de la respuesta |
| GET | - | Listado de las bibliotecas | 200 – OK | Listado de bibliotecas |
| 400 – Bad request | - |
| 404 – Not Found | - |
| 500 – Internal Server Error | - |
| POST | Información de la biblioteca | Crear una biblioteca | 201 – Created  (El ID se devuelve en la cabecera Location) | Biblioteca creada con todos sus datos |
| 400 – Bad request | - |
| 500 – Internal Server Error | - |
| PUT | - | N/A | 405 – Method not allowed | - |
| DELETE | - | N/A | 405 – Method not allowed | - |
| PATCH | - | N/A | 405 – Method not allowed | - |

El primer método *HTTP* estudiado es *GET*. Este método permite al cliente listar todas las bibliotecas del sistema. Como se puede ver en la tabla, si todo sale bien la respuesta del servidor será el código “*200-OK*”, y el cuerpo de la respuesta será precisamente la lista de bibliotecas en formato *JSON* para poder ser visualizado por el cliente. Si nos fijamos en las siguientes líneas, vemos como pueden surgir algunos errores. En esos casos, se mostrarían los códigos “*400-Bad Request*” si la solicitud se hace de manera incorrecta, “*404-Not Found*” si no encuentra los recursos solicitados, o “*500-Internal Server Error*” si hay un error del servidor.

El método *POST* permite crear una biblioteca nueva adjuntando en el cuerpo de la solicitud del *JSON* la información de esta. En los métodos *POST*, los *ID* y *URL* son asignados automáticamente por el servidor, quien responde el código “*201-Created*” si ha ido todo bien y se ha creado dicha biblioteca, y devuelve la *URI* que enlaza con la nueva biblioteca. Si ha habido algún error al ahora de realizar la solicitud o hay algún fallo del servidor, se muestran los códigos “*400-Bad Request*” o “*500-Internal Server* *Error*” respectivamente, como se ha explicado anteriormente para el método *GET*.

Para cambiar o borrar alguna biblioteca en concreto, se ha ir a la URI de la biblioteca en cuestión, por ello no se permiten los métodos *PUT*, *DELETE*, o *PATCH* para este recurso.

**Recurso:** puestos

**URI:** /bibliotecas/:bibliotecaID/puestos

Tabla . Recurso puesto

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método HTTP | Cuerpo de la solicitud | Semántica | Código Respuesta | Cuerpo de la respuesta |
| GET | - | Listado de los puestos de estudio | 200 - OK | Listado de los puestos de estudio |
| 400 – Bad request | - |
| 404 – Not Found | - |
| 500 – Internal Server Error | - |
| POST | Información del puesto de estudio | Crear un banco de trabajo | 201 – Created  (El ID se devuelve en la cabecera Location) | El puesto de estudio creado con todos sus datos |
| 400 – Bad request | - |
| 500 – Internal Server Error | - |
| PUT | - | N/A | 405 – Method not allowed | - |
| DELETE | - | N/A | 405 – Method not allowed | - |
| PATCH | - | N/A | 405 – Method not allowed | - |

El primer método *HTTP* estudiado es *GET*. Este método permite al cliente listar todos los puestos del sistema. Como se puede ver en la tabla, si todo sale bien la respuesta del servidor será el código “*200-OK*”, y el cuerpo de la respuesta será precisamente la lista de puestos en formato JSON para poder ser visualizado por el cliente. Si nos fijamos en las siguientes líneas, vemos como pueden surgir algunos errores. En esos casos, se mostrarían los códigos “*400-Bad Request*” si la solicitud se hace de manera incorrecta, “*404-Not Found*” si no encuentra los recursos solicitados, o “*500-Internal Server Error*” si hay un error del servidor.

El método *POST* permite crear un puesto nuevo adjuntando en el cuerpo de la solicitud del *JSON* la información de este. En los métodos *POST*, los *ID* y *URL* son asignados automáticamente por el servidor, quien responde el código “*201-Created*” si ha ido todo bien y se ha creado dicho puesto, y devuelve la *URI* que enlaza con el nuevo puesto. Si ha habido algún error al ahora de realizar la solicitud o hay algún fallo del servidor, se muestran los códigos “*400-Bad Request*” o “*500-Internal Server* *Error*” respectivamente, como se ha explicado anteriormente para el método *GET*.

Para cambiar o borrar algún puesto en concreto, se ha ir a la URI del puesto en cuestión, por ello no se permiten los métodos *PUT*, *DELETE*, o *PATCH* para este recurso.

**Recurso:** puesto

**URI:** /bibliotecas/:bibliotecaID/puestos/:puestoID

Tabla . Recurso puesto

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método HTTP | Cuerpo de la solicitud | Semántica | Código Respuesta | Cuerpo de la respuesta |
| GET | - | Toda la información de un puesto de estudio concreto | 200 - OK | Información del puesto de estudio con sus recursos y su disponibilidad |
| 400 – Bad request | - |
| 404 – Not Found | - |
| 500 – Internal Server Error | - |
| POST | - | N/A | 405 – Method not allowed | - |
| PUT | Información para cambiar el puesto de estudio | Modificar un puesto de estudio | 200 - OK | El puesto de estudio con los cambios realizados |
| 400 – Bad request | - |
| 500 – Internal Server Error | - |
| DELETE | - | Borrado de un puesto de estudio específico | 200 - OK | - |
| 400 – Bad request |
| 500 – Internal Server Error |
| PATCH | Información para cambiar la disponibilidad del puesto de estudio | Añadir o eliminar la disponibilidad de puesto de estudio | 200 - OK | La disponibilidad añadida o borrada |
| 400 – Bad request | - |
| 500 – Internal Server Error | - |

El método *GET* facilita obtener toda la información sobre un puesto en específico de una biblioteca. Como es habitual, el código “200-OK” aparece si va todo bien, y la respuesta es la información del puesto en formato *JSON.* Si no, aparecen los errores “404-Bad Request” o “500-Internal Server Error” según dónde surja (en cliente o servidor).

PUT cambia la descripción del puesto de la biblioteca específico, incluyendo esa información en el *body* del JSON. Si todo ha salido bien, el servidor devolverá el código “200-OK”, y en el *body* de la respuesta estará el puesto en cuestión con la información actualizada. Si no, saldrán los errores ya explicados.

Con DELETE el cliente puede borrar un puesto en concreto de la biblioteca. Si todo ha salido de forma correcta, el servidor devolverá el código “200-OKy si no, aparecen los errores ya explicados.

En esta URI se permite el método PATCH, ya que, en esta aplicación el método PATCH se usa para modificar sólo la disponibilidad de un recurso. En este caso si el cliente quiere añadir una franja de una hora de disponibilidad en el body del JSON deberá constar el método ADD y la franja de gque se quiere añadir.

Si por el contrario se quiere eliminar una franja de disponibilidad en el body del JSON deberá constar el método REMOVE y la franja de media hora que se quiere eliminar. Si todo ha salido de forma correcta, el servidor devolverá el código “200-OK”, que informa que la petición se ha realizado correctamente. Sino saldrán los mensaje de error vistos hasta ahora, según sean de cliente o de servidor.

**Recurso:** salas

**URI:** /bibliotecas/:bibliotecaID/salas

Tabla . Recurso salas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método HTTP | Cuerpo de la solicitud | Semántica | Código Respuesta | Cuerpo de la respuesta |
| GET | - | Listado de las salas de grupo | 200 - OK | Listado de las salas de grupo |
| 400 – Bad request | - |
| 404 – Not Found | - |
| 500 – Internal Server Error | - |
| POST | Información de las salas de grupo | Crear una sala de trabajos en grupo | 201 – Created  (El ID se devuelve en la cabecera Location) | La sala de grupo creado con todos sus datos |
| 400 – Bad request | - |
| 500 – Internal Server Error | - |
| PUT | - | N/A | 405 – Method not allowed | - |
| DELETE | - | N/A | 405 – Method not allowed | - |
| PATCH | - | N/A | 405 – Method not allowed | - |

El primer método *HTTP* estudiado es *GET*. Este método permite al cliente listar todas las salas del sistema. Como se puede ver en la tabla, si todo sale bien la respuesta del servidor será el código “*200-OK*”, y el cuerpo de la respuesta será precisamente la lista de salas en formato *JSON* para poder ser visualizado por el cliente. Si nos fijamos en las siguientes líneas, vemos como pueden surgir algunos errores. En esos casos, se mostrarían los códigos “*400-Bad Request*” si la solicitud se hace de manera incorrecta, “*404-Not Found*” si no encuentra los recursos solicitados, o “*500-Internal Server Error*” si hay un error del servidor.

El método *POST* permite crear una sala nueva adjuntando en el cuerpo de la solicitud del *JSON* su información. En los métodos *POST*, los *ID* y *URL* son asignados automáticamente por el servidor, quien responde el código “*201-Created*” si ha ido todo bien y se ha creado dicha sala, y devuelve la *URI* que enlaza con la nueva sala. Si ha habido algún error al ahora de realizar la solicitud o hay algún fallo del servidor, se muestran los códigos “*400-Bad Request*” o “*500-Internal Server* *Error*” respectivamente, como se ha explicado anteriormente para el método *GET*.

Para cambiar o borrar alguna sala en concreto, se ha ir a la URI de la sala en cuestión, por ello no se permiten los métodos *PUT*, *DELETE*, o *PATCH* para este recurso.

**Recurso:** salas

**URI:** /bibliotecas/:bibliotecaID/salas/:salaID

Tabla . Recurso sala

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método HTTP | Cuerpo de la solicitud | Semántica | Código Respuesta | Cuerpo de la respuesta |
| GET | - | Toda la información de una sala de grupo en concreto | 200 - OK | Información de la sala de grupo concreta con sus recursos y su disponibilidad |
| 400 – Bad request | - |
| 404 – Not Found | - |
| 500 – Internal Server Error | - |
| POST | - | N/A | 405 – Method not allowed | - |
| PUT | Información para cambiar la sala de trabajo en grupo | Cambiar una sala de grupo | 200 - OK | La sala de grupo con los cambios hechos |
| 400 – Bad request | - |
| 500 – Internal Server Error | - |
| DELETE | - | Borrado de una sala de grupo específica | 200 - OK | - |
| 400 – Bad request |
| 500 – Internal Server Error |
| PATCH | Información para cambiar la disponibilidad de la sala de grupo concreta | Añadir o borrar la disponibilidad de una sala de grupo | 200 - OK | La disponibilidad añadida o borrada |
| 400 – Bad request | - |
| 500 – Internal Server Error | - |

*GET* es el método para poder ver la información de un puesto concreto: id, URI, descripción, disponibilidad y recursos del puesto. Como ya es habitual, el código “*200-OK*” aparece si no ha ocurrido ningún problema y la respuesta es la información del puesto en formado *JSON*. Si surge algún error, aparecen los códigos “*404- Not Found*” (si no se encuentra lo solicitado), “*400-Bad-Request*” (si la solicitud está mal hecha), “*500-Internal Server Error*” (si falla el servidor).

El método *PUT* el cliente puede cambiar la información del puesto. “*200-OK*” aparece si va todo bien, mostrando la información actualizada en formato *JSON*. Si no, aparecen los códigos “*400-Bad Request*” o “*500-Server Internal Error*”.

*DELETE* permite al cliente borrar un puesto concreto del sistema. “200-OK” si va todo bien. Si no, saldrán los códigos de error vistos anteriormente.

En esta URI se permite el método PATCH, ya que, en esta aplicación el método PATCH se usa para modificar sólo la disponibilidad de un recurso. En este caso si el cliente quiere añadir una franja de media hora de disponibilidad en el body del JSON deberá constar el método ADD y la franja de media hora que se quiere añadir.

Si por el contrario se quiere eliminar una franja de disponibilidad en el body del JSON deberá constar el método REMOVE y la franja de media hora que se quiere eliminar. Si todo ha salido de forma correcta, el servidor devolverá el código “200-OK”, que informa que la petición se ha realizado correctamente. Sino saldrán los mensaje de error vistos hasta ahora, según sean de cliente o de servidor.

**Recurso:** reservas

**URI:** /reservas

Tabla . Recurso reservas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método HTTP | Cuerpo de la solicitud | Semántica | Código Respuesta | Cuerpo de la respuesta |
| GET | - | Listado de todas las reservas | 200 - OK | Listado de las reservas |
| 400 – Bad request | - |
| 404 – Not Found | - |
| 500 – Internal Server Error | - |
| POST | Información la reserva | Crear una reserva | 201 – Created  (El ID se devuelve en la cabecera Location) | La reserva creada con todos sus datos |
| 400 – Bad request | - |
| 500 – Internal Server Error | - |
| PUT | - | N/A | 405 – Method not allowed | - |
| DELETE | - | N/A | 405 – Method not allowed | - |
| PATCH | - | N/A | 405 – Method not allowed | - |

*GET* permite al cliente listar todas las reservas del sistema. Si todo sale bien la respuesta del servidor será el código “*200-OK*”, la respuesta será una lista de las reservas en formato *JSON* para poder ser visualizada. Sin embargo, pueden surgir algunos errores. “*400-Bad Request*” aparece si la solicitud se hace de manera incorrecta, “*404-Not Found” si*  los recursos solicitados no se encuentran, o “*500-Internal Server Error*” si hay un error del servidor.

*POST* permite crear una reserva nueva adjuntando en el cuerpo de la solicitud del *JSON* su información. En los métodos *POST*, los *ID* y *URL* son asignados automáticamente por el servidor, quien responde el código “*201-Created*” si ha ido todo bien y se ha creado dicha reserva, y devuelve la *URI* que enlaza con la nueva reserva. Si ha habido algún error al ahora de realizar la solicitud o hay algún fallo del servidor, se muestran los códigos “*400-Bad Request*” o “*500-Internal Server* *Error*” respectivamente, como se ha explicado anteriormente para el método *GET*.

Para cambiar o borrar alguna reserva en específico, se ha ir a la URI de la reserva en cuestión, por ello no se permiten los métodos *PUT*, *DELETE*, o *PATCH* para este recurso.

**Recurso:** recursos de puesto

**URI:** /bibliotecas/:bibliotecaID/puestos/:puestoID/recursos

Tabla . Recurso recursos de puesto

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método HTTP | Cuerpo de la solicitud | Semántica | Código Respuesta | Cuerpo de la respuesta |
| GET | - | Listado de los recursos del puesto de estudio | 200 - OK | Listado de los recursos del puesto de estudio |
| 400 – Bad request | - |
| 404 – Not Found | - |
| 500 – Internal Server Error | - |
| POST | Información del recurso del puesto de estudio | Crear un recurso del puesto de estudio | 201 – Created  (El ID se devuelve en la cabecera Location) | El recurso del puesto de estudio creado con todos sus datos |
| 400 – Bad request | - |
| 500 – Internal Server Error | - |
| PUT | - | N/A | 405 – Method not allowed | - |
| DELETE | - | N/A | 405 – Method not allowed | - |
| PATCH | - | N/A | 405 – Method not allowed | - |

El primer método *HTTP* estudiado es *GET*. Este método permite al cliente listar todos los recursos de los puestos del sistema. Como se puede ver en la tabla, si todo sale bien la respuesta del servidor será el código “*200-OK*”, y el cuerpo de la respuesta será precisamente la lista de recursos en formato JSON para poder ser visualizado por el cliente. Si nos fijamos en las siguientes líneas, vemos como pueden surgir algunos errores. En esos casos, se mostrarían los códigos “*400-Bad Request*” si la solicitud se hace de manera incorrecta, “*404-Not Found*” si no encuentra los recursos solicitados, o “*500-Internal Server Error*” si hay un error del servidor.

El método *POST* permite crear un recurso nuevo adjuntando en el cuerpo de la solicitud del *JSON* la información de este. En los métodos *POST*, los *ID* y *URL* son asignados automáticamente por el servidor, quien responde el código “*201-Created*” si ha ido todo bien y se ha creado dicho puesto, y devuelve la *URI* que enlaza con el nuevo recurso del puesto. Si ha habido algún error al ahora de realizar la solicitud o hay algún fallo del servidor, se muestran los códigos “*400-Bad Request*” o “*500-Internal Server* *Error*” respectivamente, como se ha explicado anteriormente para el método *GET*.

Para cambiar o borrar algún recurso en concreto, se ha ir a la URI del puesto en cuestión, por ello no se permiten los métodos *PUT*, *DELETE*, o *PATCH* para este recurso.

**Recurso:** recurso de puesto

**URI:** /bibliotecas/:bibliotecaID/puestos/:puestoID/recursos/:recursoPuestoID

Tabla . Recursos recurso de puesto

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método HTTP | Cuerpo de la solicitud | Semántica | Código Respuesta | Cuerpo de la respuesta |
| GET | - | Toda la información de un recurso del puesto de trabajo | 200 – OK | Información del recurso del puesto de estudio con su disponibilidad |
| 400 – Bad request | - |
| 404 – Not Found | - |
| 500 – Internal Server Error | - |
| POST | - | N/A | 405 – Method not allowed | - |
| PUT | Información para cambiar el recurso del del puesto de trabajo | Cambiar un recurso del puesto de trabajo | 200 - OK | El recurso del puesto de estudio con los cambios realizados |
| 400 – Bad request | - |
| 500 – Internal Server Error | - |
| DELETE |  | Borrado de un recurso específico del puesto de estudio | 200 - OK | - |
| 400 – Bad request |
| 500 – Internal Server Error |
| PATCH | Información para cambiar la disponibilidad del recurso del puesto de estudio | Añadir o borrar la disponibilidad de un recurso del puesto de estudio | 200 - OK | La disponibilidad añadida o borrada |
| 400 – Bad request | - |
| 500 – Internal Server Error | - |

*GET* es el método para poder ver la información de un recurso concreto de un puesto. Como ya es habitual, el código “*200-OK*” aparece si no ha ocurrido ningún problema y la respuesta es la información del recurso del puesto en formado *JSON*. Si surge algún error, aparecen los códigos “*404- Not Found*” (si no se encuentra lo solicitado), “*400-Bad-Request*” (si la solicitud está mal hecha), “*500-Internal Server Error*” (si falla el servidor).

El método *PUT* el cliente puede cambiar la información del recurso del puesto. “*200-OK*” aparece si va todo bien, mostrando la información actualizada en formato *JSON*. Si no, aparecen los códigos “*400-Bad Request*” o “*500-Server Internal Error*”.

*DELETE* permite al cliente borrar un recurso de puesto concreto del sistema. “200-OK” si va todo bien. Si no, saldrán los códigos de error vistos anteriormente.

En esta URI se permite el método PATCH, ya que, en esta aplicación el método PATCH se usa para modificar sólo la disponibilidad de un recurso. En este caso si el cliente quiere añadir una franja de media hora de disponibilidad en el body del JSON deberá constar el método ADD y la franja de media hora que se quiere añadir.

Si por el contrario se quiere eliminar una franja de disponibilidad en el body del JSON deberá constar el método REMOVE y la franja de media hora que se quiere eliminar. Si todo ha salido de forma correcta, el servidor devolverá el código “200-OK”, que informa que la petición se ha realizado correctamente. Sino saldrán los mensaje de error vistos hasta ahora, según sean de cliente o de servidor.

**Recurso:** recursos de sala

**URI:** /bibliotecas/:bibliotecaID/salas/:salaID/recursos

Tabla . Recurso recursos de sala

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método HTTP | Cuerpo de la solicitud | Semántica | Código Respuesta | Cuerpo de la respuesta |
| GET | - | Listado de los recursos de la sala en grupo | 200 - OK | Listado de los recursos de la sala de trabajo en grupo |
| 400 – Bad request | - |
| 404 – Not Found | - |
| 500 – Internal Server Error | - |
| POST | Información del recurso de la sala de grupo | Crear un recurso de la sala | 201 – Created  (El ID se devuelve en la cabecera Location) | El recurso la sala de grupo creado con todos sus datos |
| 400 – Bad request | - |
| 500 – Internal Server Error | - |
| PUT | - | N/A | 405 – Method not allowed | - |
| DELETE | - | N/A | 405 – Method not allowed | - |
| PATCH | - | N/A | 405 – Method not allowed | - |

El método *GET* permite al cliente listar todos los recursos de una sala del sistema. Como se puede ver en la tabla, si todo sale bien la respuesta del servidor será el código “*200-OK*”, y el cuerpo de la respuesta será precisamente la lista de recursos de la sala en formato *JSON* para poder ser visualizado por el cliente. Si nos fijamos en las siguientes líneas, vemos como pueden surgir algunos errores. En esos casos, se mostrarían los códigos “*400-Bad Request*” si la solicitud se hace de manera incorrecta, “*404-Not Found*” si no encuentra los recursos solicitados, o “*500-Internal Server Error*” si hay un error del servidor.

El método *POST* permite crear recursos de una sala nueva proporcionando información. En los métodos *POST*, los *ID* y *URL* son asignados automáticamente por el servidor, quien responde el código “*201-Created*” si ha ido todo bien y se ha creado dicha sala, y devuelve la *URI* que enlaza con los nuevos recursos de sala. Si ha habido algún error al ahora de realizar la solicitud o hay algún fallo del servidor, se muestran los códigos “*400-Bad Request*” o “*500-Internal Server* *Error*” respectivamente, como se ha explicado anteriormente para el método *GET*.

Para cambiar o borrar algun recurso de sala en concreto, se ha ir a la URI del recursos de sala en cuestión, por ello no se permiten los métodos *PUT*, *DELETE*, o *PATCH* para este recurso.

**Recurso:** recurso de sala

**URI:** /bibliotecas/:bibliotecaID/salas/:salaID/recursos/:recursoSalaID

Tabla . Recurso recurso de sala

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método HTTP | Cuerpo de la solicitud | Semántica | Código Respuesta | Cuerpo de la respuesta |
| GET | - | Toda la información de un recurso de la sala de grupo | 200 – OK | Información la sala en grupo con su disponibilidad |
| 400 – Bad request | - |
| 404 – Not Found | - |
| 500 – Internal Server Error | - |
| POST | - | N/A | 405 – Method not allowed | - |
| PUT | Información para modificar el recurso del de la sala de trabajo en grupo | Cambiar un recurso de la sala | 200 - OK | El recurso de la sala con los cambios realizados |
| 400 – Bad request | - |
| 500 – Internal Server Error | - |
| DELETE | - | Borrado de un recurso específico de la sala de trabajo en grupo | 200 - OK | - |
| 400 – Bad request |
| 500 – Internal Server Error |
| PATCH | Información para cambiar la disponibilidad del recurso de la sala | Añadir o borrar la disponibilidad de un recurso de la sala de grupo | 200 - OK | La disponibilidad añadida o borrada |
| 400 – Bad request | - |
| 500 – Internal Server Error | - |

*GET* es el método para poder ver la información de un recurso concreto de un puesto. Como ya es habitual, el código “*200-OK*” aparece si no ha ocurrido ningún problema y la respuesta es la información del recurso del puesto en formado *JSON*. Si surge algún error, aparecen los códigos “*404- Not Found*” (si no se encuentra lo solicitado), “*400-Bad-Request*” (si la solicitud está mal hecha), “*500-Internal Server Error*” (si falla el servidor).

El método *PUT* el cliente puede cambiar la información del recurso del puesto. “*200-OK*” aparece si va todo bien, mostrando la información actualizada en formato *JSON*. Si no, aparecen los códigos “*400-Bad Request*” o “*500-Server Internal Error*”.

*DELETE* permite al cliente borrar un recurso de puesto concreto del sistema. “200-OK” si va todo bien. Si no, saldrán los códigos de error vistos anteriormente.

En esta URI se permite el método PATCH, ya que, en esta aplicación el método PATCH se usa para modificar sólo la disponibilidad de un recurso. En este caso si el cliente quiere añadir una franja de media hora de disponibilidad en el body del JSON deberá constar el método ADD y la franja de media hora que se quiere añadir.

Si por el contrario se quiere eliminar una franja de disponibilidad en el body del JSON deberá constar el método REMOVE y la franja de media hora que se quiere eliminar. Si todo ha salido de forma correcta, el servidor devolverá el código “200-OK”, que informa que la petición se ha realizado correctamente. Sino saldrán los mensaje de error vistos hasta ahora, según sean de cliente o de servidor.