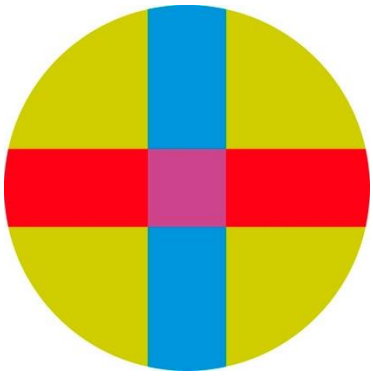
I

UNIVERSIDAD SAN PABLO - CEU

ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA DE SISTEMAS DE INFORMACIÓN



TRABAJO FIN DE GRADO

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. Fernando Ortiz de Pedro la calificación de _____

Resumen

Debido a la pandemia del coronavirus iniciada 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 los organismos públicos, de las cuales destacan las relacionadas con bibliotecas y centros de estudio: control de aforo, ventilación de las estancias, separación entre puestos de estudio con 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 centros sanitarios, o se han mantenido porque han demostrado su utilidad. Es en esta segunda situación en la que surge el tema de este TFG. La Universidad San Pablo CEU ha decidido facilitar la gestión del acceso a sus bibliotecas, y por ello en este TFG se desarrolla un Servicio Web *RESTful* de gestión de reservas de puestos.

Palabras Clave

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

Abstract

Due to the coronavirus pandemic that began in 2020 and the restrictions implemented by institutions to prevent its spread, many everyday situations have been affected worldwide. Universidad San Pablo CEU implemented the necessary measures imposed by public authorities, including those related to libraries and study centers: capacity control, room ventilation, minimum interpersonal distance between study spaces, suspension of book lending systems, and more. Nowadays, most of these measures no longer exist as the situation is less concerning, but there are others that either remain mandatory, such as the use of masks in healthcare centers, or have been maintained because they have proven useful. It is in this second situation that the subject of this Bachelor's Thesis arises. Universidad San Pablo CEU has decided to facilitate access management to its libraries, which is why this thesis focuses on the development of a RESTful Web Service for seat reservations.

Keywords

RESTful web service, library, study post, room, reservation, resource, availability, bookable element.

Índice de contenidos

Índice de ilustraciones

Índice de tablas

Capítulo 1

Introducción

La pandemia mundial que surgió en el año dos mil veinte provocó, y sigue provocando, cambios en todo tipo de aspectos en nuestras vidas y a diferentes escalas. Respecto a la universidad, se tomaron 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). En el momento en que surgió la idea de la aplicación, su objetivo principal era ayudar a evitar la propagación del virus. Sin embargo, debido a la dilatación en la realización del proyecto, el contexto ha cambiado. Afortunadamente el virus ha perdido importancia hoy en día, pero la aplicación sigue siendo una solución a 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, y con la aplicación se solventaría tal situación. Además, post pandemia la sociedad se ha acostumbrado a tener que realizar reservas en su día a día. Ya sea para ir a la oficina, ir a restaurantes, ir a clases colectivas en el gimnasio, o ir a la peluquería. 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 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, …

1.1 Objetivos

Como objetivo general, 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. Así, se podrá controlar el aforo de la estancia y facilitar a los alumnos la gestión y coordinación de su tiempo y espacio de estudio. 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
- El *software* debe tener una arquitectura que permita poder realizar distintas implementaciones de interfaces de usuario (web, móviles, etcétera) y facilite la futura integración dentro del ecosistema de aplicaciones que tengan establecidas (por ejemplo, integración en el portal del alumno)
- Diseñar un Servicio Web que permita la creación de una reserva en la biblioteca, además de altas, bajas y modificaciones de esta

- Creación de representaciones de los diferentes recursos del sistema: en HTML para ser consumidas por usuarios humanos y, por otra parte, en formato JSON para permitir la integración del servicio de otras aplicaciones

Como hemos comentado, fuera ya del ámbito de este TFG, se desea que la aplicación pudiera ser integrada en el sistema de información de la universidad (intranet, portal del alumno). Así, se podrán aprovechar todos los datos que ya se manejan para poder mejorar la biblioteca a nivel sanitario, del servicio, y de la seguridad.

Esta memoria presenta el trabajo realizado durante este TFG 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.

Capítulo 2

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.

2.1 Modelo de ciclo de vida

Para realizar este proyecto se ha utilizado una metodología tradicional, como es el caso de la metodología *waterfall* o en 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.

Esta metodología se compone de varias fases que se ejecutan de manera secuencial en el siguiente orden:

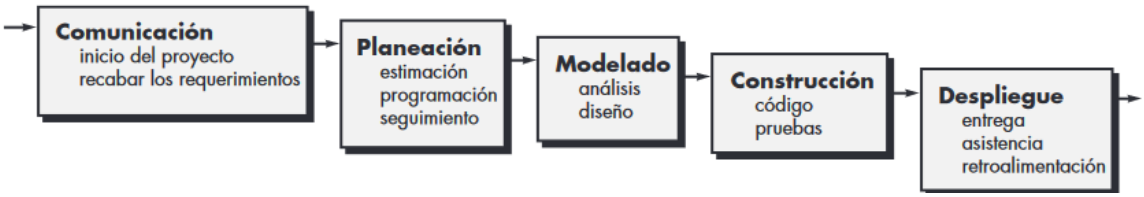


Ilustración 1: Fases de la metodología waterfall (S. Pressman, 1994)

- **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)

2.2 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.

2.3 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:

Nombre	Fecha de inicio	Fecha de fin
Memoria del proyecto	15/2/21	24/6/21
Análisis	17/2/21	25/2/21
Diseño	26/2/21	5/3/21
Tareas parte práctica	8/3/21	24/6/21
Tutoriales API REST	8/3/21	19/3/21
Creación y conexión base de datos	22/3/21	9/4/21
Desarrollo programa	12/4/21	14/6/21
Despliegue	15/6/21	24/6/21

Ilustración 2: Diagrama Gantt de planificación (elaboración propia). Reparto de tareas.

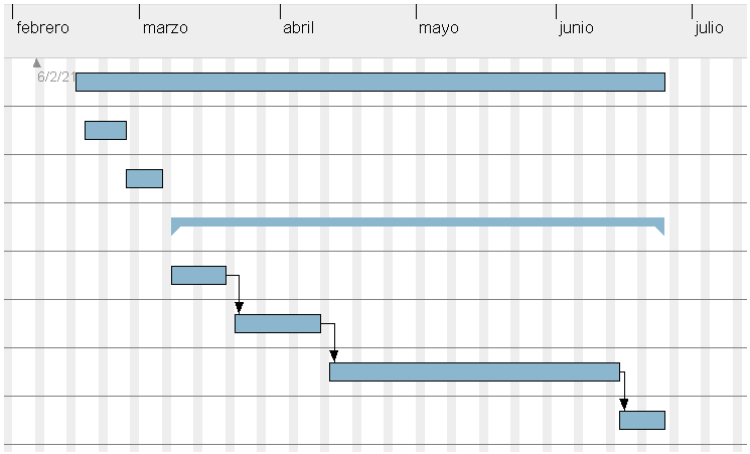


Ilustración 3: Diagrama Gantt de planificación (elaboración propia). Tareas organizadas en cascada.

2.4 Ejecución

Utilizando la metodología *waterfall*, se llevaron a cabo diversas tareas para elaborar este trabajo. Se analizó el dominio de la aplicación, se estudió la situación actual y se identificaron problemas logísticos en la reserva de la biblioteca. Se analizó la solución informática existente y se realizaron encuestas para mejorarla. Con esa información, se establecieron los requisitos del prototipo. Se estudió el estado del arte y se definió la arquitectura del prototipo. Se diseñó la arquitectura y se implementó el software, realizando pruebas y verificaciones. Se redactó la memoria del proyecto, incluyendo introducción, gestión, análisis, diseño e implementación. Se concluyó con líneas futuras, bibliografía y anexos.

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.

Nombre	Fecha de inicio	Fecha de fin
Memoria	25/1/21	29/6/23
Objetivos de la aplicación	25/1/21	16/2/21
Gestión de proyecto	17/2/21	2/3/21
Análisis del dominio de la aplicación	3/3/21	6/4/21
Estado del Arte	7/4/21	20/5/21
Análisis de recursos y procesos del siste...	21/5/21	8/6/21
Especificacion de requisitos	9/6/21	23/6/21
Análisis del dominio Affluences	9/9/21	7/10/21
Affluences investigación app y web	9/9/21	20/9/21
Entrevistas a alumnos y empleados	21/9/21	30/9/21
Especificación requisitos	1/10/21	7/10/21
Recursos y procesos del sistema	24/2/22	9/3/22
Arquitectura y diseño	10/3/22	28/3/22
Tutoriales e investigación REST	29/4/22	20/5/22
Configuración y conexión de base de datos	20/5/22	27/5/22
Desarrollo	30/5/22	28/6/23

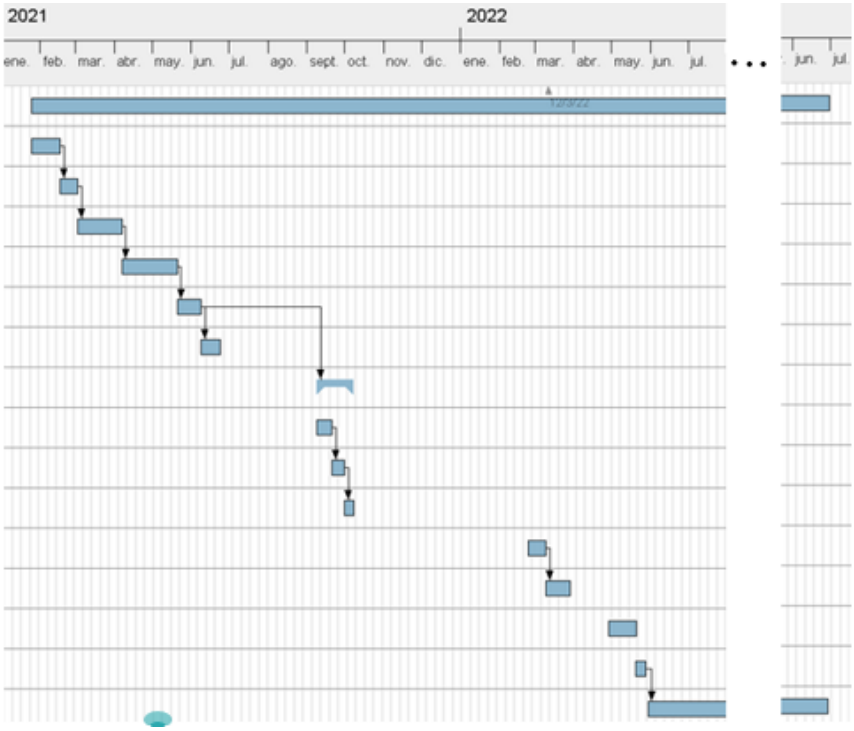


Ilustración 4: Eso Tabla Diagrama Gantt real. Elaboración propia.

Capítulo 3

Estado del Arte

Hoy en día, Internet es esencial en nuestras vidas, marcando un antes y un después en la sociedad. La mayoría de los productos y servicios que usamos actualmente dependen de él, como contactar con alguien, pedir comida a domicilio o hacer reservas en restaurantes.

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.

3.1 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:

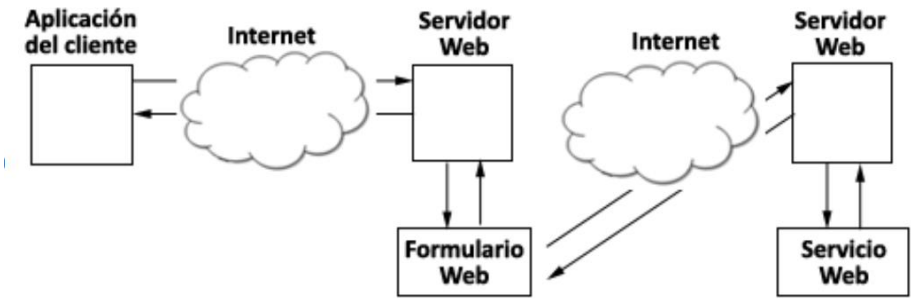


Ilustración 5: Despliegue de un Servicio Web (Jácome Galarza, 2010)

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)

3.1.1 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* cuyas reglas tratan de bloquear o auditar la comunicación entre programas a ambos lados de la barrera. Además, para realizar transacciones, no pueden compararse en su grado de desarrollo con los estándares abiertos de computación distribuida como *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.

3.1.2 Arquitectura Oritentada 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:

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 (RedHat, 2020).

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 (RedHat, 2020).

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" (RedHat, 2020).

*3.1.2.1 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, y como podemos observar en el siguiente diagrama, hay más protocolos que intervienen en las comunicaciones:

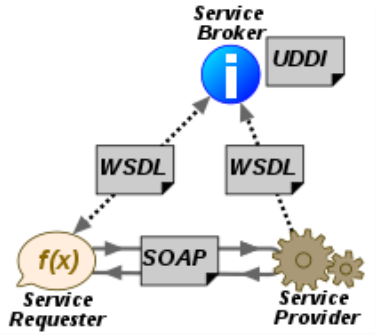


Ilustración 7: Detalle tecnológico de una Arquitectura Orientada a Servicios (SOA). (Issarny et al., 2011)

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.

3.1.3 Arquitectura Orientada a Recursos (ROA)

La 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:

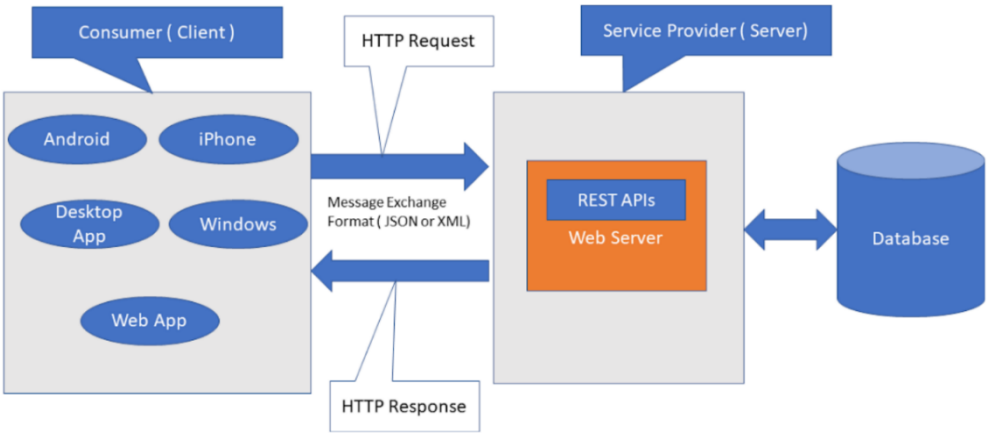


Ilustración 8: Diagrama de una Arquitectura Orientada a Servicios (SOA) (Gitonga, 2021)

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 “sala”, “reserva”, o “puesto”; y sus URIs serían algo parecido a, por ejemplo, “/salas” para identificar al conjunto de salas, o “/salas/:id” para localizar a una sala 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.

3. **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.

4. **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.

3.1.3.1 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 ordenados de menor a mayor: soportes de un servicio para URI, HTTP e Hipermedia (también hay un cuarto nivel sin soporte).

- **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.
- **Nivel 1 - Resources:** En 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 "muchas URI, pero un solo un único verbo HTTP". Además, "las operaciones se tunelizan insertando nombres de operaciones y parámetros en una URI y luego transmitiendo esa *URI* a un servicio remoto, generalmente a través de HTTP GET" (Fowler, 2010) .
- **Nivel 2 - HTTP verbs (métodos http):** En el nivel 2 se usan de diferentes métodos HTTP, ajustándolos a lo que establece el propio estándar (RFC 9110) 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.
- **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.

3.1.3.2 Comparativa entre ambas aproximaciones

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.

Por un lado, REST es fácil de implementar y mantener, sigue una filosofía de Web abierta. Permite una clara separación entre cliente y servidor, y no está controlado por una única entidad. El cliente puede almacenar información para evitar llamadas repetitivas, y REST puede devolver datos en varios formatos. Sin embargo, tiene limitaciones como su dependencia del protocolo HTTP y dificultades en la garantía de autorización y seguridad.

Por otro lado, SOAP sigue un enfoque empresarial formal y es compatible con diversos protocolos de comunicación, incluyendo la comunicación asíncrona. Proporciona información detallada sobre los objetos a los clientes, y la seguridad y autorización forman parte integrante del protocolo, describiéndose mediante WSDL. Sin embargo, SOAP también presenta desventajas como el consumo de ancho de banda debido a la comunicación de metadatos, el uso exclusivo de XML, su dificultad de implementación y su falta de popularidad entre los desarrolladores web y móviles. SOAP se orienta principalmente a operaciones transaccionales y a usuarios satisfechos con esta tecnología. (Fisher, 2015).

3.2 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.

3.2.1 Backend

Play Framework: El *framework* utilizado facilita el desarrollo de Servicios Web con una interfaz HTTP amigable y flexible. La amplia comunidad ofrece abundante documentación y tutoriales. Se utiliza Java, pero también es compatible con Scala. Se explorará su funcionamiento mediante un ejemplo práctico ejemplo más adelante. (Leroux & de Kaper, 2014).

Base de datos: Para el proyecto, se utilizará MySQL, un gestor de base de datos relacional de código abierto. Es fácil de instalar y configurar, con baja demanda de recursos y destacada persistencia de datos. Su elección se basa en su adaptabilidad y cumplimiento de requisitos.

Postman: Esta herramienta permite testear la API permitiendo generar peticiones.

3.2.2 Frontend

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

***Hypertext Markup Language* (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)

CSS: Es un lenguaje que modifica la apariencia de la información en la web, permitiendo cambios en fuentes, colores, tamaños y formatos. Al ser independiente de HTML, garantiza una apariencia uniforme en todas las páginas sin repetir la configuración. Además, estructura la información para adaptarla a diferentes tamaños de pantalla, como dispositivos móviles y monitores. (HTML & CSS.)

JavaScript: Complementa HTML y CSS al agregar funcionalidades dinámicas a las páginas web a través de scripts. Puede actualizar contenido, mostrar tablas interactivas, animar gráficos y más sin necesidad de recargar la página. Las aplicaciones web desarrolladas con JavaScript pueden ser páginas web o widgets distribuidos. Además, JavaScript permite interactuar con el entorno del usuario, como su ubicación, a través de las Interfaces de Programación de Aplicaciones (APIs). Este lenguaje simplifica la implementación de programas complejos y enriquece el desarrollo web, creando páginas más dinámicas.

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.

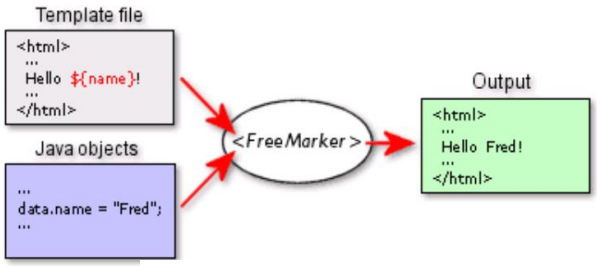


Ilustración 9: Funcionamiento de FreeMarker. (Apache.org, 2015)

Capítulo 4

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. Cabe destacar, que este contexto cambió durante la realización del proyecto. En efecto, la solución primigenia de la universidad fue la analizada en el *Anexo I*, pero más adelante la universidad contrató el *software Affluences*, que también fue analizado en el *Anexo II*. En este mismo apartado también se detallan los actores y recursos que pertenecen al dominio y, más adelante, se detallan los casos de uso de la aplicación. Finalmente, se especifican los requisitos funcionales y no funcionales de la aplicación propuesta en este trabajo.

4.1 Análisis del dominio de la aplicación

Después de la cuarentena nacional, se fue recuperando poco a poco la vida normal y se abrieron de nuevo los centros educativos. La universidad implementó medidas de prevención contra la propagación del virus en la universidad pudiendo contagiar a trabajadores y alumnado. Uno de los muchos espacios que pasaron a tener control es la biblioteca. En efecto, se decidió 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 pasó a tener una mayor vigilancia del aforo

de la biblioteca. Todos los alumnos, profesores, o trabajadores de la biblioteca debían llevar una mascarilla que cubriera nariz y boca durante todo el tiempo que estuvieran en la biblioteca. Había (y sigue estando) un dispensador de gel hidroalcohólico en la entrada de la biblioteca, y toda persona que quisiera entrar debía lavarse las manos con él. Cabe destacar que los puestos de estudio se numeraron y separaron según las restricciones sanitarias para frenar la propagación del virus (1,5 metros). También se prohibió levantarse para interactuar con otro alumno (saludar, preguntar dudas, estudiar juntos, …), sólo se permitía salir a descansar.

Para mayor prevención, se mantenían 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 abandonaba un puesto, éste debía quedar limpio y preparado para que otro alumno lo pudiera ocupar. No sólo actuaba el equipo de limpieza, sino que también se puso a disposición de los alumnos tanto papel como desinfectante para que pudieran limpiar de nuevo su puesto si lo desearan. Esto permitía evitar el contagio entre alumnos que ocuparan el mismo puesto durante el mismo día.

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 pasaron a ser inaccesibles, para evitar el contagio. Lo mismo ocurrió 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 redujo el aforo de la estancia, sino que existía un registro de los alumnos que entraban a estudiar cada día, la franja horaria en la que lo hacían, y el número de puesto en el que se sentaban. Esto ayudaba a tener una trazabilidad en caso de que algún alumno diera positivo en coronavirus.

No solo había que prestar atención a las medidas sanitarias que afectaban directamente a los alumnos, sino también a los procesos que llevaban a cabo en la biblioteca. Por ejemplo, cuando un alumno entraba en la biblioteca, se le asignaba un puesto en el que estudiar. Sin embargo, algunos alumnos tenían preferencias a la hora de estudiar por la ubicación del puesto, la incidencia de luz natural, posición de enchufes, u otros motivos. Por ello, el puesto de estudio no debía ser impuesto, sino que debían tener libertad de elección. Pensando en los alumnos de primeros cursos, que quizás no conocieran del todo la biblioteca, era 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 podía permitir la reserva de puestos, sobre todo teniendo en cuenta que había épocas del año de mayor afluencia a la biblioteca. Y es que, las semanas de vacaciones y días previos a exámenes de convocatoria ordinaria, extraordinaria, o fines de semana, muchos alumnos preferían estudiar en la biblioteca de la universidad. Por ello, era útil que los alumnos pudieran reservar los sitios con antelación, para asegurarse una plaza y evitar ir a la biblioteca si no había plazas libres. A la hora de hacer la reserva, se podía 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 hubiera problemas de aforo en la biblioteca, y que cada alumno pudiera 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 fueron las semanas previas a exámenes parciales y las convocatorias ordinarias y extraordinarias. En el *Anexo I* se puede observar un análisis de la solución informal primigenia, y en el *Anexo II* un análisis del *software Affluences* que se implementó durante el desarrollo de este proyecto. En el siguiente apartado, veremos los actores que intervienen en la aplicación.

4.2 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.

En el siguiente diagrama podemos ver los diferentes grupos de interés:

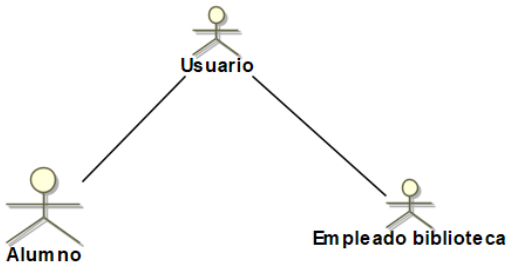


Ilustración 10: Diagrama de actores del sistema. Elaboración propia.

4.3 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.

4.3.1 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).

4.3.2 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.

4.3.3 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.

4.3.4 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.

4.3.5 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*.

4.3.6 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.

4.3.7 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.

4.3.8 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.

4.4 Procesos

En 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 en la Ilustración 11, y más adelante se explica cada situación en detalle.

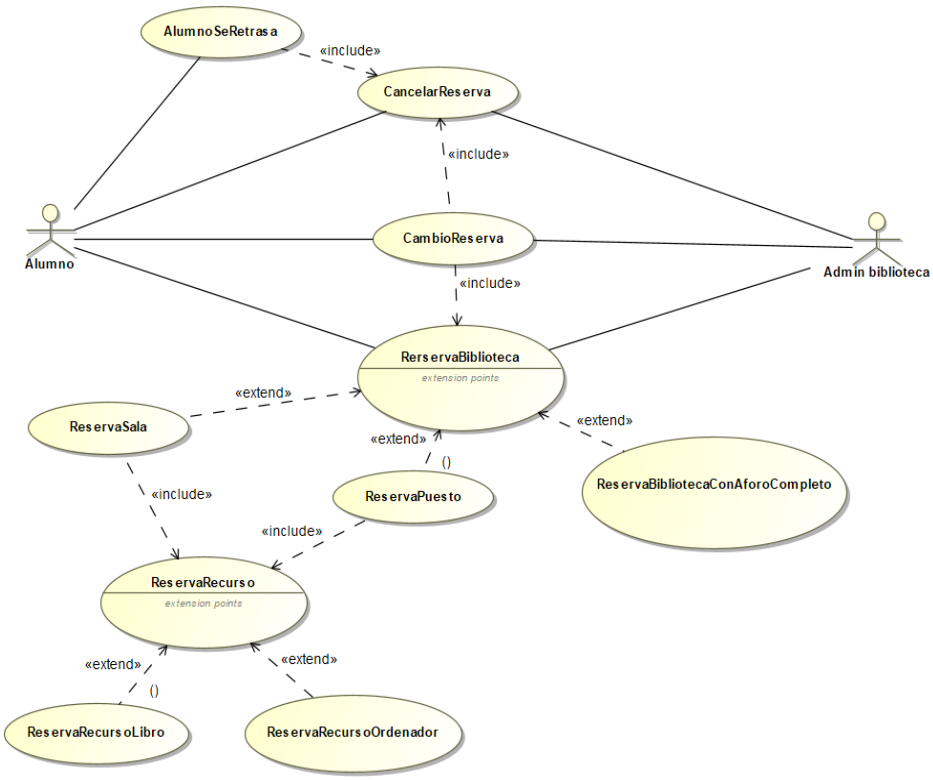


Ilustración 11: Diagrama de casos de uso. Elaboración propia.

4.4.1 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 4.4.9.1*), que le impide realizar la reserva.

4.4.2 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, …).

4.4.3 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*).

4.4.4 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*).

4.4.5 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).

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 4.4.4), 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).

4.4.7 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.

4.4.8 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.

4.4.9 Casos particulares

4.4.9.1 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.

4.4.9.2 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.

4.5 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 y en la reserva de un puesto u otro. Si bien un sistema de gestión de bibliotecas debería ser completo y abarcar

más situaciones, este prototipo desarrollado para verificar la viabilidad del sistema, se centra en la reserva de puestos de estudio y salas, así como de libros u ordenadores. Además, está enfocado al punto de vista del trabajador de la biblioteca, puesto que el usuario puede crear, modificar, y eliminar recursos, permisos que los alumnos no deberían tener. Con esto claro, se detallan a continuación los requisitos funcionales y no funcionales del prototipo a desarrollar.

4.5.1 Requisitos funcionales

Tabla 1: Tabla de requisitos funcionales. Elaboración propia.

Requisitos	Descripción
RF1	Se permitirá reservar puestos y salas de una biblioteca
RF2	Deberá permitir reservar otros recursosde la biblioteca como libros u ordenadores
RF3	Cada reserva realizada por un usuario deberá tener un código asociado
RF4	El usuario podrá realizar reservas
RF5	El usudario podrá cancelar reservas
RF6	El usuario reservará franjas de una hora
RF7	Deberá permitir gestionar las reservas de varias bibliotecas a la vez

4.5.2 Requisitos no funcionales

Tabla 2: Tabla de requisitos no funcionales. Elaboración propia.

Requisitos	Descripción
RNF1	La interfaz deberá ser amigable y fácil de usar
RNF2	Deberá ser escalable, y por tanto soportar picos de mayor uso y afluencia de alumnos por exámenes
RNF3	La plataforma deberá ser fácilmente integrable en otros sistemas legados que tenga la universidad
RNF4	La plataforma deberá ser versátil para poder integrar distintos tipos de interfaces (web, móviles,...)

Capítulo 5

Arquitectura y diseño

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

5.1 Arquitectura del sistema

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:

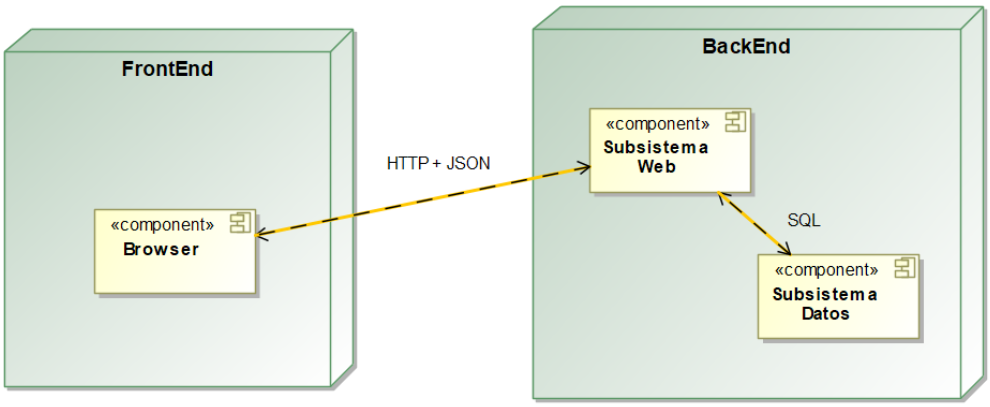


Ilustración 12: Diagrama de despliegue de la arquitectura del sistema. Elaboración propia.

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.

5.2 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 con sus URIs asociadas 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. La URI definida es: **/usuarios/:id**
- **Usuarios:** el recurso en cuestión permite realizar gestiones sobre el conjunto de usuarios. La URI definida es: **/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. La URI definida es: **/bibliotecas/:id/puestos/:idPuesto**
- **Puestos:** permite hacer acciones sobre los diferentes puestos que tiene una biblioteca. La URI definida es: **/bibliotecas/:id/puestos/**
- **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. La URI definida es: **/bibliotecas/:id/salas/:idSala**

- **Salas:** el recurso en cuestión da la posibilidad de interactuar con todas las salas que tiene una biblioteca. La URI definida es: **/bibliotecas/:id/salas/**
- **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, … La URI definida es: **/reservas/:reservaID**
- **Reservas:** este recurso engloba al conjunto de reservas que se llevan a cabo en una biblioteca por parte de los usuarios. La URI definida es: **/reservas**
- **Biblioteca:** al haber varias bibliotecas en el campus de la universidad, este recurso hace referencia a cada una de ellas. La URI definida es: **/bibliotecas/:id**
- **Bibliotecas:** como se ha explicado anteriormente, la universidad tiene diferentes bibliotecas y, este recurso, permite realizar operaciones sobre todas ellas. La URI definida es: **/bibliotecas/**
- **Ordenador:** se refiere al uso de un ordenador prestado por la biblioteca para hacer consultas en un puesto de estudio o sala. La URI definida es: **/bibliotecas/:id/ordenadores/:idOrdenador**
- **Ordenadores:** al haber varias unidades en una biblioteca con identificador ID, este recurso permite realizar modificaciones sobre ellos en conjunto y acceder su información. La URI definida es: **/bibliotecas/:id/ordenadores/**
- **Libro:** este recurso permite interactuar con un libro y consultarlo en la biblioteca. La URI definida es: **/bibliotecas/:id/libros/:idLibro**
- **Libros:** el recurso en cuestión da la posibilidad de interactuar con todos los libros que tiene una biblioteca. La URI definida es: **/bibliotecas/:id/libros/**

5.2.2 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. Se ha utilizado JSON en vez de XML ya que es más ligero y representa la información requerida de una forma más fácil para ser consumida por otros componentes *software*. Para el uso por parte de los usuarios, los recursos también devuelven una representación HTML, fácilmente interpretable por este tipo de usuarios. Como en cualquier aplicación Web, la elección de una u otra representación se lleva a cabo mediante la negociación del tipo de representación que proporciona el protocolo HTTP (donde se indica, explícitamente, el tipo de representaciones que puede consumir el cliente).

5.2.3 Establecimiento de interfaz

En este apartado se muestra, codificado en formato tabla, la definición de la API del sistema que recae en los distintos métodos de HTTP que se utilizan y las respuestas que ofrece cada recurso (definidos en el *Apartado 5.2.1*). Debido a limitaciones de espacio, en este apartado sólo se muestra la tabla de un único recurso. Sin embargo, en el *Anexo IV "Tablas API de recursos"* se puede encontrar la API al completo. En cada una de las tablas, a parte del propio método, se detalla el cuerpo de la solicitud, la semántica, y el código de respuesta. Cabe destacar, que aparte del deseado código 200 o 201 confirmando que se ha hecho correctamente la solicitud, existen otros códigos de excepción que son comunes a todas las tablas API. El código 400 *Bad request* surge cuando se han introducido incorrectamente los datos de la solicitud, el código 404 *Not found* cuando no se encuentra el recurso solicitado, el código 405 *Method not allowed* cuando el método no está disponible en la URI en cuestión, y el código 500 *Internal error* cuando hay un error del servidor.

En este caso, es la tabla API del recurso: **usuario**, cuya URI es **/usuarios/:id**

Tabla 3: Tabla URI recurso usuario

Método HTTP	Cuerpo de la solicitud	Semántica	Código Respuesta	Cuerpo de la respuesta
GET	-	Mostrar la información del usuario	200 – OK	Información del usuario solicitada
PUT	Información completa a actualizar	Actualizar toda la información del usuario	200 - OK	Toda la información del usuario actualizada
DELETE	-	Borrar un usuario	200 - OK	Usuario eliminado
PATCH	Información concreta a actualizar	Modificar parcialmente un usuario	200 - OK	Información actualizada del usuario

El primer método HTTP estudiado es GET. Este método permite al cliente mostrar la información del usuario solicitado. 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 información del usuario en formato JSON para poder ser visualizado por el cliente (o HTML si se invocó el método a través de un navegador).

Para cambiar o borrar algún usuario en concreto, se usan los métodos PATCH, PUT y DELETE. El primero sirve para actualizar parcialmente la información

del usuario y, como se puede ver en la tabla, se mostrará un código 200 al realizar una solicitud correcta introduciendo la información que se desea actualizar. El contenido de una petición PATCH no está estandarizado. Pero habitualmente se utiliza un formato en el que se especifique el atributo o atributos a cambiar y el nuevo valor a establecer. El método PUT sirve para actualizar toda la información del usuario en cuestión. Habrá que introducirla antes de realizar la solicitud y, si la solicitud es correcta, aparecerá el código 200 y se devolverá en formato JSON dicha información. Finalmente, el método DELETE permite borrar dicho usuario al realizar la petición. El cliente sabe que se ha borrado correctamente gracias al código de respuesta 200.

5.2.4 Base de datos

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 y URIs",* se ha diseñado el siguiente diagrama entidad-relación disponible en la siguiente página.

De la Ilustración 13 cabe destacar que hay dos herencias: por un lado, la compuesta por las entidades *ElementoReservable* (padre) y sus hijas S*ala* y *Puesto*; por otro lado, la compuesta por *RecursoExtra* (padre) y sus hijas *Libro* y *Ordenador*. La primera herencia forma parte de una *Reserva* que haría un usuario para guardar un puesto de estudio o una sala de trabajo en grupo. La segunda, hace referencia al *RecursoExtra* (*Libro* u *Ordenador*) que se puede asociar a una *Reserva* realizada.

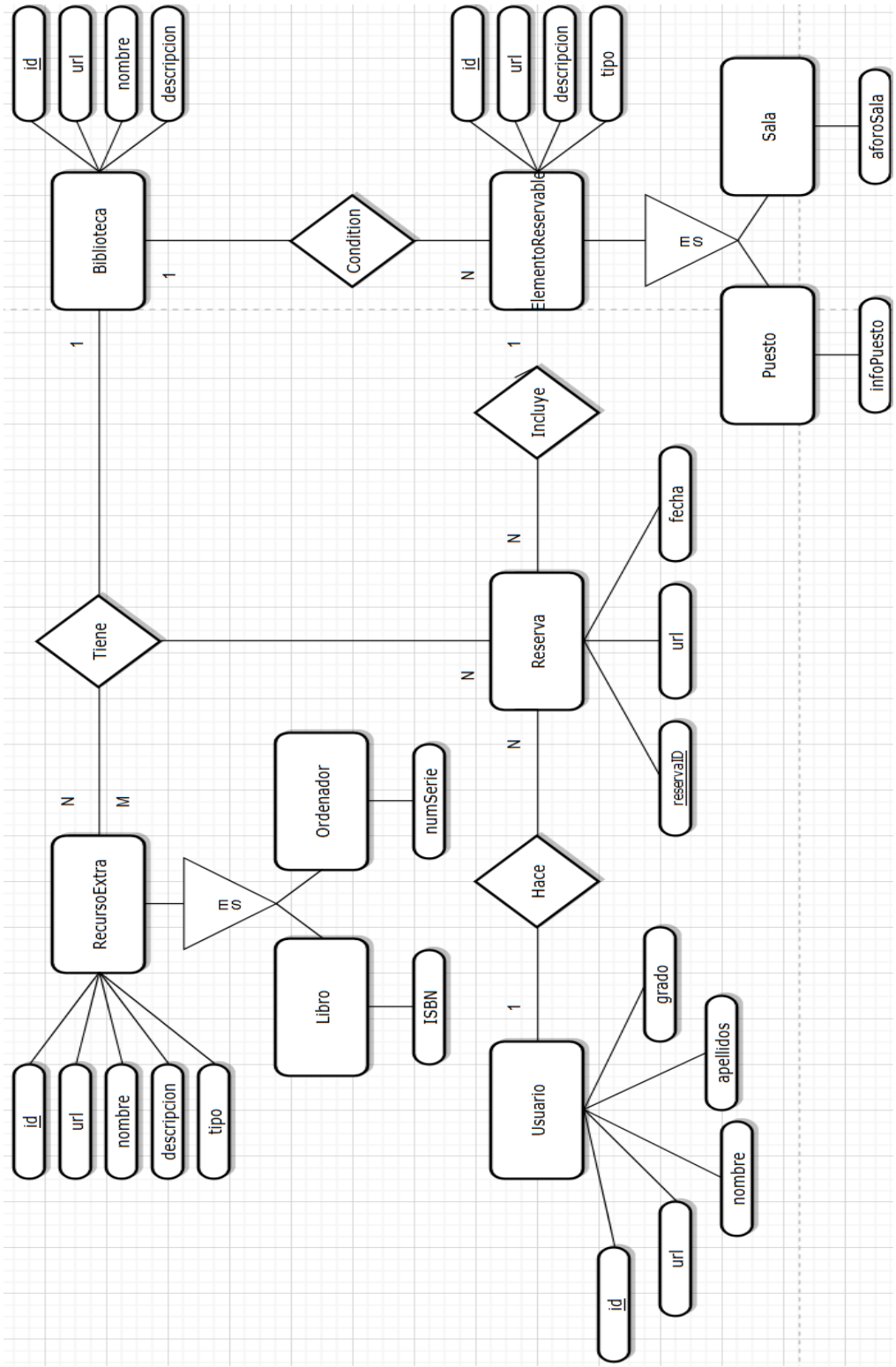


Ilustración 13: Diagrama diseño entidad/relación. Elaboración propia.

5.3 Diseño del Frontend

Para el desarrollo de la interfaz de usuario, se utilizará la herramienta *FreeMarker* para poder realizar una interfaz HTML e interactuar de una forma más sencilla, intruitiva y funcional que con el prototipo. A continuación, se muestra de manera esquemática, cómo se vería la interfaz de la pantalla de Inicio de la aplicación:

Ilustración 14: Esquema página inicio del prototipo

Como se puede ver en el recuadro azul del esquema, la página de inicio tendría una zona de bienvenida al sistema de gestion con una imagen del logo de la universidad. Además, como se ve en la zona verde, tendría unas indicaciones para dejar claro al usuario qué permite hacer el prototipo. Para ello, sería necesario que hubiera unos enlaces directos o botones que permitieran acceder a la información necesaria para realizar reservas. En el esquema, la zona en un recuadro rojo señalan enlaces a las listas de “Usuarios”, “Bibliotecas”, y “Reservas”.

Otro esquema realizado, muestra cómo podría ser una pantalla con información más relevante que la pantalla de bienvenida, la página de “Usuarios”:

Ilustración 15: Esquema página usuarios del prototipo

Como se puede observar en el recuadro rojo del esquema, la pantalla tendría acceso directo a las otras pantallas mediante pulsaciones en “Inicio”, “Bibliotecas” o “Reservas”. Además, como se puede observar en la zona azul, constaría de la lista de todos los identificadores de los usuarios del sistema, así como de sus URIs, en las cuales se podría pinchar para acceder a la información concreta de ese usuario. Finalmente, se permitiría crear un usuario nuevo dando la posibilidad al usuario de introducir la información requerida.

Capítulo 6

Implementación y pruebas

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

6.1 Implementación del Backend

Este apartado sirve para entender mejor los diferentes componentes de la aplicación. El prototipo, como se comentó en el Apartado 3.2 "Tecnologías", se ha realizado utilizando Java y *Play Framework.* En este framework, la estructura de una aplicación se divide en cuatro grandes partes: *Entities*, *Services*, *Controllers*, y *Routes*.

6.1.1 Entities

Estas entidades representan los objetos (junto con sus atributos) que son utilizados en las comunicaciones HTTP. Todas las entidades heredan de la clase *RecursoWeb* puesto que heredan la URI que está en esa clase. Por cada recurso identificado en el Apartado 5.2.1 "Definición de recursos y URIs" (*Libro*, *ElementoReservable*, *Reserva*, …) hay una clase entidad. También destacan las entidades *"Short"* que permiten acceder a una información más resumida (identificador y URI) sin tener que listar todos los atributos del recurso.

Además, otras entidades como *TipoReserva*, *TipoElementoReservable*, o *TipoRecursoExtra*, juegan un papel muy importante puesto que, gracias a un enumerado, permiten distinguir entre *Reserva* y *ReservaExtra*, así como entre *Puesto* y *Sala* o entre *Libro* y *Ordenador* a la hora de realizar peticiones.

Las clases *CambioAforoSala*, *CambioBiblioteca*, *CambioGrado*, … heredan de *Cambio* y son las que se utilizan para generar el JSON en las solicitudes PATCH, e indican el atributo concreto a modificar del recurso.

6.1.2 Routes

El fichero *Routes* relaciona los métodos HTTP que se invocan sobre una URI y la clase/método al que se llama para realizar la atención de la petición sobre dicho recurso.

La siguiente imagen es un ejemplo de una ruta, en la que se puede ver que un método que acepta el recurso *reservas* es POST. Además, no sólo se detalla qué petición es y en qué URI, sino cuál es el controlador al que apuntar para realizar esa petición, en este caso *ReservaController* que ejecutará el método *create(request:Request)* como se puede ver en el rectángulo rojo de la Ilustración 16. Además, cabe destacar toda la API se ha documentado con Swagger (la metainformación que hay sobre la ruta). Gracias a esta documentación y al uso de un plugin de Play Framework se genera, automáticamente, una interfaz que sirve para la realización de las diferentes pruebas de la aplicación (ver *6.4 Pruebas*).

```
###
  #  summary: Crear una reserva. (POST)
  #  consumes:
  #    - application/json
  #  parameters:
  #    - in: body
  #      name: reserva
  #      description: La reserva a crear.
  #      schema:
  #         $ref: '#/definitions/entities.Reserva'
  #  responses:
  #    201:
  #      description: success
  #      content:application/json:
  #      schema:
  #        $ref: '#/definitions/entities.Reserva'
  ###
POST /reservas      controllers.ReservaController.create(request:Request)
```

Ilustración 16: Captura de pantalla de la route POST /reservas

6.1.3 Controllers

Cuando un *controller* recibe la petición gracias a la ruta definida, se encarga de procesar la solicitud del cliente. Siguiendo el ejemplo anterior, el método *create (request:Request)* de *ReservaController* es:

```
33 @    public Result create(Http.Request request) throws SQLException, ClassNotFoundException {
34          JsonNode json = request.body().asJson();
35          System.out.println(json);
36          if (json == null) {
37              return badRequest(ApplicationUtil.createResponse( response: "Expecting JSON data",  ok: false));
38          }
39          logger.debug("In ReservaBD.create(), input is: {}", json.toString());
40          Reserva reserva = ReservaBD.getInstance().addReserva(Json.fromJson(json, Reserva.class));
41
42          JsonNode jsonObject = Json.toJson(reserva);
43          System.out.println("La reserva es: " + reserva);
44          return created(ApplicationUtil.createResponse(jsonObject,  ok: true)).withHeader(LOCATION, reserva.getUrl());
45      }
```

Ilustración 17: Captura de pantalla del método create de ReservaController

En este método, primero se comprueba que los datos introducidos de la reserva que se quiere crear en formato JSON son los correctos ya que, de no ser así, lanza un error *badRequest* con el código 400 correspondiente y el mensaje *"Expecting JSON data"*. Si va todo bien, llama al servicio *ReservaBD* que se encarga de obtener la información requerida desde la capa de persistencia.

6.1.4 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 de cada recurso (ReservaBD, BibliotecaBD, UsuarioBD, …). Cabe destacar que no existe una clase tipo *Service* para las clases hijas de las herencias puesto que existen las clases *ElementoReservableBD* y *RecursoExtraBD* de los padres, en las cuales se realiza una consulta u otra a la base de datos en función de si se requiere una petición de un hijo u otro.

En cada una de las clases hay un método que se encarga del método HTTP correspondiente: GET realiza un *Select en la base de datos*, POST hace un *Insert*, PATCH y PUT hacen un *Update*, y DELETE hace un *Delete*.

6.2 Base de datos

Para la persistencia de la información en este proyecto se ha usado MySQL. Después de realizar el diagrama entidad relación de diseño mostrado en el del *Apartado 5.2.4 "Base de datos"*, se muestra el paso a tablas correspondiente:

RecursoExtra (id, url, nombre, descripción, tipo, numSerie, aforoSala, bibliotecaID)
Usuario (id, url, nombre, apellidos, grado)
Biblioteca (id, url, nombre, descripción)
ElementoReservable (id, url, descripción, tipo, infoPuesto, aforoSala, bibliotecaID)
Reserva (reservaID, url, fecha, usuarioID, elementoReservableID)
ReservaExtra (RecursoExtraID, reservaID)

Cabe destacar, que surge la tabla *ReservaExtra* debido a la relación entre las entidades *Reserva* y *RecursoExtra* y su cardinalidad (N:M). Esta tabla almacena el *RecursoExtraID* asociado al identificador *reservaID*Además. Se muestran las tablas del prototipo en la Iliustración 18 mediante un diagrama UML realizado en *Workbench*:

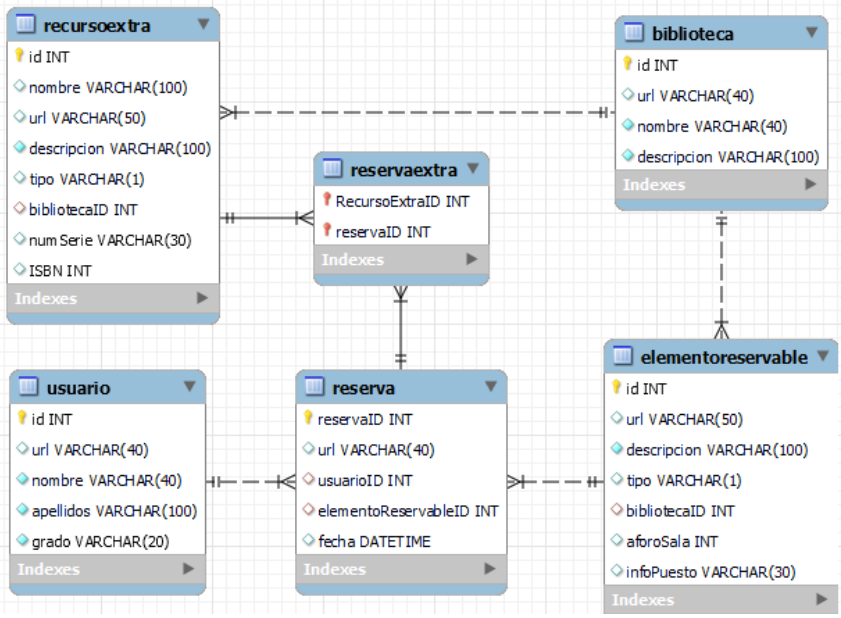


Ilustración 18: Diagrama UML de la base de datos creado en MySQL Workbench. Elaboración propia.

6.3 Implementación del Frontend

Como se ha mencionado anteriormente en el Apartado 3.2.2 "Frontend" en el apartado de tecnologías, *Freemarker* se ha seleccionado como herramienta que permite realizar las representaciones HTML (y, por tanto, la interfaz consumible por humanos de la aplicación). Este *software* es un motor de plantillas integrado en Play Framework y, gracias a CSS y HTML, permite que el usuario pueda interactuar con el prototipo de una forma muy intuitiva.

En efecto, cada recurso requiere su propia plantilla, a la cual se le añade principalmente dos secciones. En la primera se añade el o los *scripts* en JavaScript que permiten lanzar los métodos POST, PATCH, PUT, o DELETE (ya que, los navegadores sólo suelen emitir los métodos GET y POST). En la segunda, el *body* del HTML en el cual se añade toda la información que aparecerá en la página: títulos, frases, formularios para introducir información, botones, …

En el prototipo, el recurso usuario tiene tres plantillas: usuarios.ftl, usuario.ftl, y usuarioMissing.ftl. El primer archivo sería el equivalente a un GET, y muestra la lista de usuarios del sistema, así como un formulario para añadir un puesto como se puede ver a continuación en la Ilustración 19:

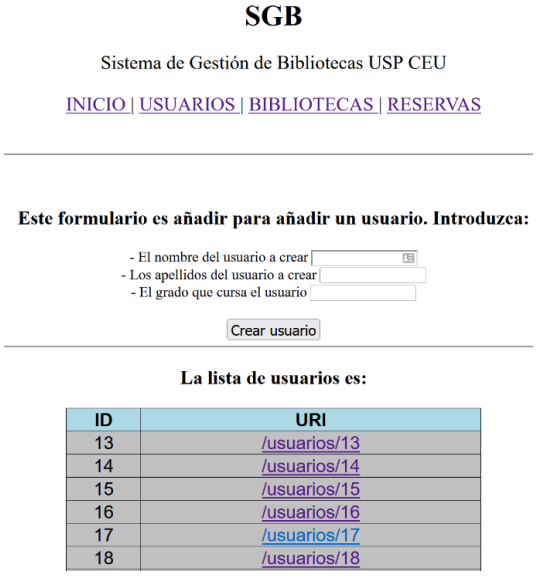


Ilustración 19: Captura de pantalla interfaz prototipo (usuarios.ftl)

Ese formulario es posible gracias al *script* de JavaScript que llama al método POST. Como se puede ver en los rectángulos rojos de la Ilustración 20, primero se especifica qué método se solicita, después se precisa los datos JSON que se va a introducir en el *"formulario",* y finalmente se envía la información mediante *xhr.send(data);*, se detalla el mensaje por pantalla que aparece al realizar el POST,

y un *location.reload();* para recargar la página y que aparezca el nuevo usuario en la lista de usuarios.ftl.

```
<script>
function makePOSTRequest(url){

var xhr = new XMLHttpRequest();
xhr.open("POST", url);

xhr.setRequestHeader("Accept", "application/json");
xhr.setRequestHeader("Content-Type", "application/json");

xhr.onreadystatechange = function () {
   if (xhr.readyState === 4) {
      console.log(xhr.status);
      console.log(xhr.responseText);
   }};


var form = document.querySelector("#formulario");
var data = `{
                    "nombre": "`+form.querySelector('input[name="nombre"]').value+`",
                    "apellidos": "`+form.querySelector('input[name="apellidos"]').value+`",
                    "grado": "`+form.querySelector('input[name="grado"]').value+`"
            }`;

console.log(data)
xhr.send(data);
alert ("Usuario creado correctamente");
location.reload();
}
</script>
```

Ilustración 20: Script método POST plantilla usuarios.ftl

Esa solicitud de POST se lleva a cabo en el *<body>* del HTML a la hora de declarar el formulario como se puede ver en el siguiente recuadro de la Ilustración 21:

```
<form action="#" onSubmit="makePOSTRequest('http://localhost:9000/usuarios'); return false;" id="formulario" >
  <b> <p>Este formulario es añadir para añadir un usuario. Introduzca: </p> </b>
  <div>
    <label for="usuario.nombre">- El nombre del usuario a crear</label>
    <input name="nombre" id="nombre" value="">
  </div>
   <div>
     <label for="usuario.apellidos">- Los apellidos del usuario a crear</label>
     <input name="apellidos" id="apellidos" value="">
    </div>
  <div>
    <label for="usuario.grado">- El grado que cursa el usuario</label>
    <input name="grado" id="grado" value="">
  </div>
  <div style="margin-top:25px;">
    <button id="creacion" class="boton1">Crear usuario</button>
  </div>
</form>
```

Ilustración 21: Script formulario crear usuario plantilla usuarios.ftl

La lista de usuarios se compone por el identificador de este, y su URI en la cual se puede pinchar e ir a la página de información de ese usuario concreto. Esa página es la representada por el archivo usuario.ftl. Dicho archivo representaría un GET, y aparece la información concreta de dicho usuario (id, URI, nombre, apellidos, y grado) y, de igual manera que para el POST, los tres *scripts* en JavaScript de los métodos PATCH, PUT y DELETE y las tres secciones correspondientes para tratar la información (ver siguiente página). Además, cabe destacar que en cada plantilla hay código en CSS para hacer la interfaz más amigable. Para los dos primeros métodos hay dos formularios que permiten introducir la información que se desea cambiar, y para el tercer método hay un botón para eliminar el usuario (ver ilustración 22).

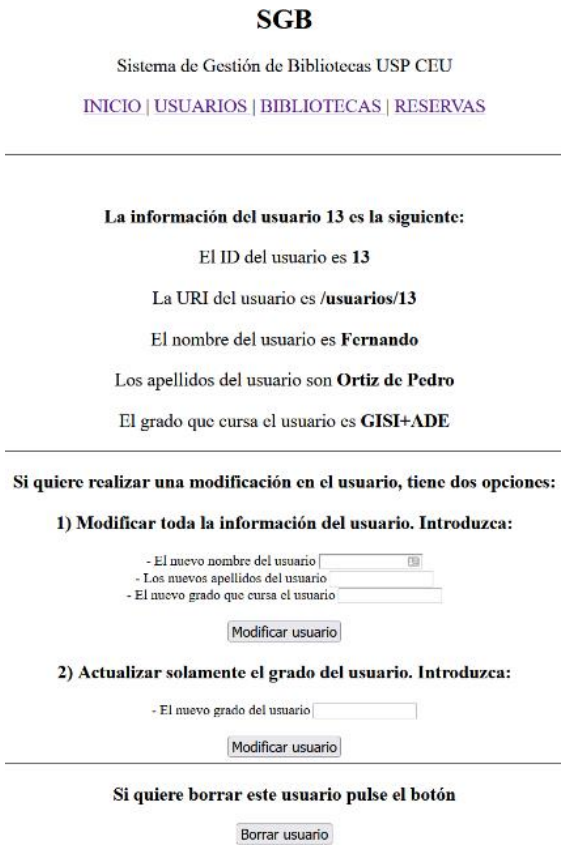


Ilustración 22: Captura de pantalla interfaz plantilla usuario.ftl

En cuanto a usuarioMissing.ftl, es una plantilla para informar de que el usuario al que se quiere acceder con la URI introducida no está disponible porque no existe o ha sido borrado, y por tanto invita al usuario a volver atrás o al inicio.

6.4 Pruebas

Se han realizado numerosas pruebas del prototipo. Para ello, se ha utilizado Postman durante el desarrollo de este y así poder testear cada método de una forma correcta. Además, una vez documentada la API, se ha utilizado la interfaz generada por Swagger, probando de una manera sencilla la totalidad de esta una vez acabada. En la Ilustración 23 se pueden ver algunos de los métodos implementados:

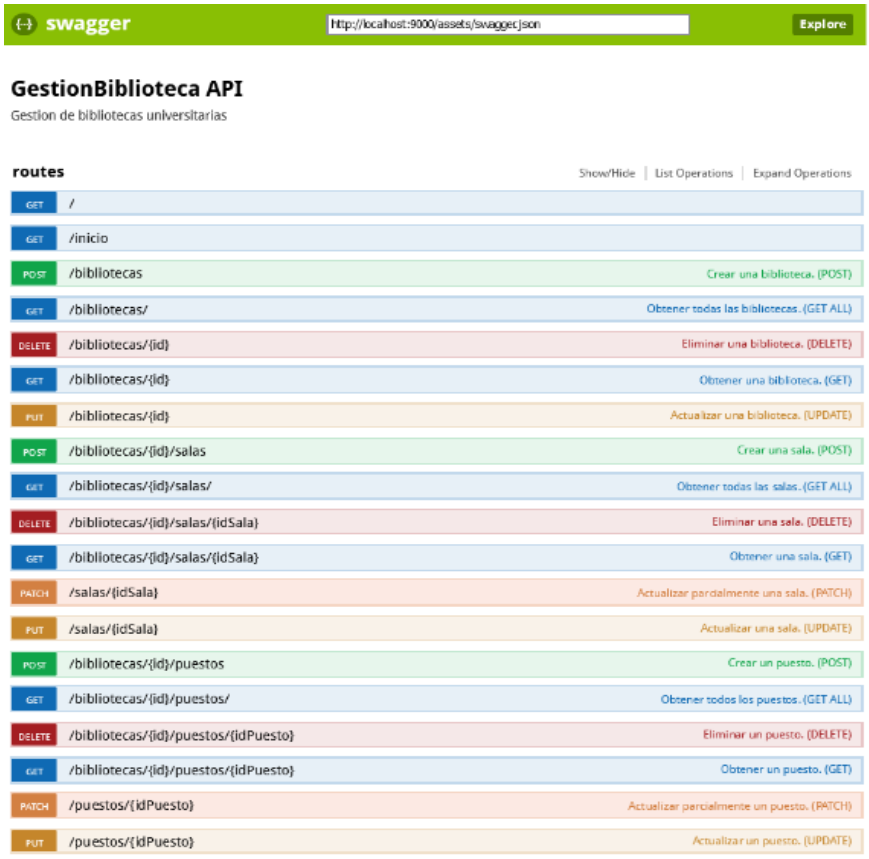


Ilustración 23: Captura métodos Swagger

6.5 Referencia al repositorio de software

Respecto al *software* de este proyecto, toda la implementación del prototipo, así como la memoria de este proyecto, se encuentran accesibles en el siguiente repositorio: https://github.com/nandortiz/tfgisi

6.6 Despliegue aplicación

El prototipo en cuestión se encuentra desplegado en una instancia de Amazon Web Services (AWS), accesible a través en http://34.244.157.254:9000 (no disponible a través de Eduroam).

6.7 Manuales

Para que pueda probarse la aplicación, se ha proporcionado en el Anexo V "Manual: creación de reservas en prototipo desarrollado", un manual paso a paso de cómo realizar una reserva completa sobre el prototipo desarrollado.

Capítulo 7

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 inicial 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 adquirió y puso en producción un nuevo *software* de gestión que también tuvo que ser analizado, para intentar detectar sus debilidades y aporta, en nuestra propuesta, 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 de la aplicación, se comenzó con la implementación del prototipo. Para ello se programó un Servicio Web *RESTful* utilizando *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. En este TFG se ha definido la API de una aplicación (que recae en el uso de los métodos del protocolo HTTP): HTTP GET, POST, PATCH y DELETE para poder interactuar con los recursos identificados 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
- El *software* debe tener una arquitectura que permita poder realizar diferentes implementaciones de interfaces de usuario (web, móviles etcétera) y facilite la futura integración dentro del exosistema de aplicaciones que tengan establecidas (por ejemplo, integración en el portal del alumno)
- Diseñar un Servicio Web que permita la creación de una reserva en la biblioteca, además de altas, bajas y modificaciones de esta
- Creación de representaciones de los diferentes recursos del sistema: en HTML para ser consumidas por usuarios humanos y, por otra parte, en formato JSON para permitir la integración del servicio de otras aplicaciones.

Además, cabe destacar que se han cumplido todos los requisitos funcionales propuestos en el *Apartado 4.5.1 Requisitos funcionales*, así como los no funcionales propuestos en el *Apartado 4.5.2 Requisitos no funcionales*.

Siguiendo el Modelo de Madurez de Richardson explicado en el Apartado 3.1.3.1 Modelo de Richardson se puede clasificar el prototipo creado en el nivel 3 de madurez, ya que se dispone de varios recursos (uno por cada abstracción importante del dominio de la aplicación), cada recurso responde a una interfaz homogénea haciendo uso de los distintos métodos HTTP de manera consistente con la definición del estándar y también se cumple la restricción HATEOAS impuesta por REST, ya que, en las representaciones JSON que se proporcionan de los diferentes recursos, se incluyen enlaces a otros recursos permitiendo que la interfaz pueda facilitar la navegación entre distintos estados de la aplicación.

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.

7.1 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 hubiera un proceso de verificación de la presencialidad del usuario al llegar a la biblioteca
- Que hubiera un *logout* y los alumnos tuvieran que realizarlo al salir de la biblioteca y el sistema verificara cada cierto tiempo qué sitios reservados se hubieran quedado vacíos antes de que acabara la reserva realizada, para evitar cuellos de botella y esperas innecesarias si un alumno se fuera antes de que acabe su reserva
- Que hiciera 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 asistieran más o menos alumnos que el aforo de la 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
- Mejorar la aplicación añadiendo un *login* de tal manera que, por ejemplo, una vez el usuario introduce sus credenciales, no tenga que meter su identificador para realizar una reserva
- Que la aplicación estuviera integrada con el ecosistema de la universidad, y por tanto los usuarios tuvieran acceso a otros servicios en la misma aplicación, como por ejemplo acceso a sus asignaturas.

Bibliografía

- Apache.org. (2015). *What is Apache FreeMarker™?* https://freemarker.apache.org/ (Último acceso, junio 2023)
- Digital Guide, I. (2018). *HATEOAS: ¿cuál es el principio que oculta este acrónimo?* https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/hateoas-que-es-y-cual-es-su-funcion-en-las-api-rest/ (Último acceso, junio 2023)
- Doveltech. (2003). *The Pros and Cons of Web Services.* https://doveltech.com/innovation/the-pros-and-cons-of-web-services/ (Último acceso, junio 2023)
- El orden mundial. (2021). El mapa del acceso a internet en el mundo. https://elordenmundial.com/mapas-y-graficos/mapa-acceso-internet-mundo/ (Último acceso, junio 2023)
- Erl, T. (2004). *Introduction to Web Services Technologies: SOA, SOAP, WSDL and UDDI*
- Fisher, R. (2015). *REST vs SOAP: Nordic APIs Infographic comparison* https://nordicapis.com/rest-vs-soap-nordic-apis-infographic-comparison/ (Último acceso, junio 2023)
- Fowler, M. (2010). Richardson Maturity Model,

  steps toward the glory of REST.

https://martinfowler.com/articles/richardsonMaturityModel.html (Último acceso, junio 2023)

- *HTML – Standard* (2022) https://html.spec.whatwg.org/multipage/introduction.html#background (Último acceso, junio 2023)
- Leroux, N., & de Kaper, S. (2014). *Play for Java*
- McCabe, F., Michael Champion, I., Ferris, C. & Orchard, D. (2004). *Web Services Architecture.* https://www.w3.org/TR/ws-arch/ (Último acceso, junio 2023)
- Richardson, L., & Ruby, S. (2007). *RESTful Web Services*
- S. Pressman, R. (1994). *Ingeniería del software. Un enfoque práctico.* (Séptima ed.)
- Thomas Fieldind, R. (2000). *Architectural Styles and the Design of Network-based Software Architectures* https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (Último acceso, junio 2023)
- Universidad de Alicante. (2014). *Introducción a los Servicios Web. Invocación de servicios web SOAP. Índice*
- RedHat (2020), *¿Qué es la arquitectura orientada a los servicios (SOA)?* https://www.redhat.com/es/topics/cloud-native-apps/what-is-service-oriented-architecture#resumen (Último acceso, junio 2023)

- Wikipedia. (2021). *Servicio web.* https://es.wikipedia.org/wiki/Servicio_web#Inconvenientes_de_los_servicios_web (Último acceso, junio 2023)

Anexo I
Análisis de la 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.

Anexo II

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 *Apartado I.1* y *Apartado I.2* 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: n***o 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.

I.1 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.

En 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:

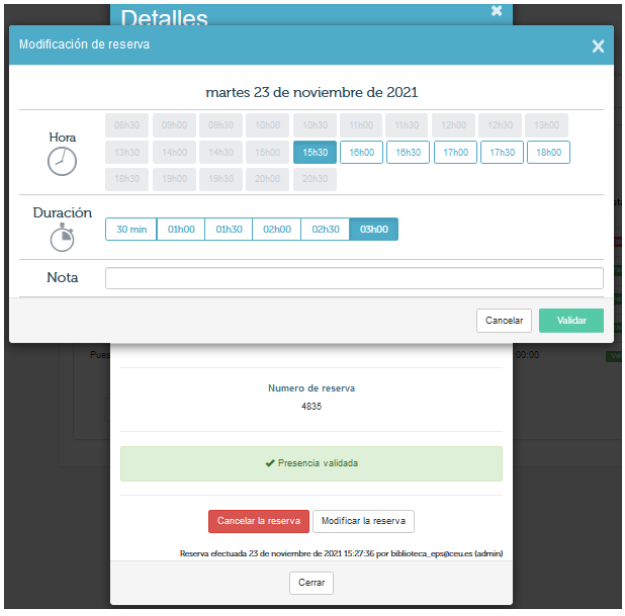


Ilustración 24: Proceso reserva puesto en Affluences por parte de un alumno. Elaboración propia.

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.

Suponiendo 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:

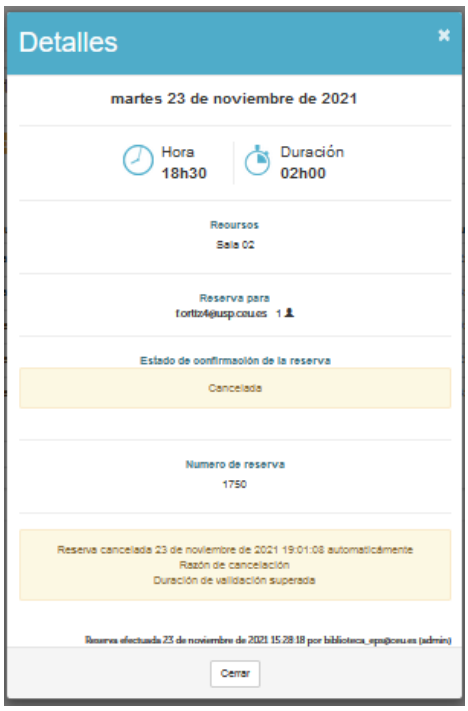


Ilustración 25: Reserva puesto en Affluences cancelada. Elaboración propia.

Ilustración 26: Reserva puesto Affluences confirmada. 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.

I.2 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

Análisis de seguridad

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.

Anexo IV

Tablas API de recursos

Cabe recordar, que existen otros códigos de excepción que son comunes a todas las tablas API aparte de los códigos *200 OK* o *201 Created*. El código 400 *Bad request* surge cuando se han introducido incorrectamente los datos de la solicitud, el código 404 *Not found* cuando no se encuentra el recurso solicitado, el código 405 *Method not allowed* cuando el método no está disponible en la URI en cuestión, y el código 500 *Internal error* cuando hay un error del servidor. Sólo se representarán los métodos utilizados, los métodos *not allowed* no se mostrarán en las tablas.

La siguiente tabla API es del recurso **usuarios**, con URI **/usuarios**

Tabla 4: Tabla URI recurso usuarios

Método HTTP	Cuerpo de la solicitud	Semántica	Código Respuesta	Cuerpo de la respuesta
GET	-	Mostrar todos los usuarios	200 – OK	IDs y URIs de todos los usuarios del sistema
POST	Información del usuario a crear	Añadir un usuario	201 - Created	La información del usuario creado

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 IDs y URIs de los usuarios en formato JSON. 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 y su información introducida.

La siguiente tabla API de los recursos es común para los recursos **puesto**, **sala**, **ordenador** y **libro**, ya que comparten métodos. Sus URI son respectivamente: **/bibliotecas/:id/puestos/:idPuesto**, **/bibliotecas/:id/salas/:idSala**, **/bibliotecas/:id/orednadores/:idOrdenador**, **/bibliotecas/:id/libros/:idLibro**

Tabla 5: Tabla URI recursos puesto, sala, ordenador y libro

Método HTTP	Cuerpo de la solicitud	Semántica	Código Respuesta	Cuerpo de la respuesta
GET	-	Mostrar la información del recurso	200 – OK	Información del recurso requerido
PUT	Información completa que actualizar	Actualizar toda la información del recurso	200 - OK	Toda la información del recurso actualizada
DELETE	-	Borrar un recurso	200 - OK	Recurso eliminado

PATCH	Información concreta que actualizar	Modificar parcialmente un recurso	200 - OK	Información modificada del recurso

El primer método *HTTP* estudiado es *GET*. Este método permite al cliente mostrar la información del recurso solicitado (aquí puesto, sala, ordenador o libro). 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 información del recurso en formato *JSON* para poder ser visualizado por el cliente.

Para cambiar o borrar algún recurso en concreto, se usan los métodos *PATCH*, *PUT* y *DELETE*. El primero sirve para actualizar parcialmente la información del recurso y, como se puede ver en la tabla, se mostrará un código 200 al realizar una solicitud correcta introduciendo la información que se desea actualizar. El método *PUT* sirve para actualizar toda la información del usuario en cuestión. Habrá que introducirla antes de realizar la solicitud y, si está bien hecha, aparecerá el código 200 y se devolverá en formato *JSON* dicha información. Finalmente, el método *DELETE* permite borrar dicho usuario al realizar la petición. El cliente sabe que se ha borrado correctamente gracias al código de respuesta 200.

La siguiente tabla API de los recursos es común para los recursos **puestos**, **salas**, **ordenadores** y **libros**, ya que comparten métodos. Sus URI son respectivamente: **/bibliotecas/:id/puestos/**, **/bibliotecas/:id/salas/**, **/bibliotecas/:id/orednadores/**, **/bibliotecas/:id/libros/**

Tabla 6: Tabla URI recursos puestos, salas, ordenadores y libros

Método HTTP	**Cuerpo de la solicitud**	**Semántica**	**Código Respuesta**	**Cuerpo de la respuesta**
GET	-	Mostrar la lista de todos los recursos en cuestión	200 – OK	IDs y URIs de todos los recursos solicitados del sistema
POST	Información del recurso a crear	Añadir un recurso	201 - Created	La información del recurso creado

El primer método HTTP estudiado es GET. Este método permite al cliente listar todos los puestos, salas, ordenadores o libros 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 IDs y URIs de los recursos mencionados anteriormente en formato JSON. 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 recurso, y devuelve la URI que enlaza con el nuevo recurso y su información introducida.

La siguiente tabla API es del recurso **reserva**, y su URI es **/reservas/:reservaID**

Tabla 7: Tabla URI recurso reserva

Método HTTP	Cuerpo de la solicitud	Semántica	Código Respuesta	Cuerpo de la respuesta
GET	-	Mostrar la información de la reserva	200 – OK	La información de la reserva solicitada
DELETE	-	Borrar una reserva concreta	200 – OK	Reserva eliminada

El primer método HTTP estudiado es GET. Este método permite al cliente mostrar la información de la reserva solicitada. 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 información de la reserva en formato JSON para poder ser visualizado por el cliente. Para borrar alguna reserva en concreto, se usa DELETE. Este método permite borrar la reserva concreta al realizar la petición. El cliente sabe que se ha borrado correctamente gracias al código de respuesta 200.

La siguiente tabla API es del recurso **reservas**, y su URI es **/reservas**

Tabla 8: Tabla URI recurso reservas

Método HTTP	Cuerpo de la solicitud	Semántica	Código Respuesta	Cuerpo de la respuesta
GET	-	Mostrar la lista de las reservas del sistema	200 – OK	La lista de IDs y URIs de reservas que hay en el sistema
POST	Información de la reserva a crear	Añadir una reserva	201 - Created	La información la reserva realizada

El primer método HTTP estudiado es GET. Este método permite al cliente listar todas las reservas 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 IDs y URIs de las reservas en formato JSON. El método POST permite crear una reserva 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 reserva, y devuelve la URI que enlaza con la nueva reserva y su información introducida.

La siguiente tabla API es del recurso **biblioteca**, y su URI es **/bibliotecas/:id**

Tabla 9: Tabla URI recurso biblioteca

Método HTTP	Cuerpo de la solicitud	Semántica	Código Respuesta	Cuerpo de la respuesta
GET	-	Mostrar la información de la biblioteca	200 – OK	Información de la biblioteca requerida
PUT	Información completa que actualizar	Actualizar toda la información de la biblioteca	200 - OK	Toda la información de la biblioteca actualizada
DELETE	-	Borrar una biblioteca	200 - OK	Biblioteca eliminada

El primer método HTTP estudiado es GET. Este método permite al cliente mostrar la información de la biblioteca solicitada. 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 información de la biblioteca deseada en formato JSON para poder ser visualizado por el cliente.

Para cambiarla información de la biblioteca o borrar alguna biblioteca en concreto, se usan los métodos PUT y DELETE. El método PUT sirve para actualizar toda la información de la biblioteca en cuestión. Habrá que introducirla antes de realizar la solicitud y, si está bien hecha, aparecerá el código 200 y se devolverá en formato JSON dicha información. Finalmente, el método DELETE permite borrar dicha biblioteca al realizar la petición. El cliente sabe que se ha borrado correctamente gracias al código de respuesta 200.

La siguiente tabla API es del recurso **bibliotecas**, y su URI es **/bibliotecas/**

Tabla 10: Tabla URI recurso bibliotecas

Método HTTP	Cuerpo de la solicitud	Semántica	Código Respuesta	Cuerpo de la respuesta
GET	-	Mostrar la lista de las bibliotecas	200 – OK	IDs y URIs de todas las bibliotecas del sistema
POST	Información de la biblioteca a crear	Añadir una biblioteca	201 - Created	La información la biblioteca creada

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 IDs y URIs de las bibliotecas en formato JSON. 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 y su información introducida.

Anexo V

Manual: creación de reservas en prototipo desarrollado

- Acceder a http://34.244.157.254:9000 en el navegador. A continuación, se muestra la pantalla de inicio, y pincharemos en el acceso directo "Usuarios" marcado en amarillo en la siguiente imagen:

Ilustración 27: Captura interfaz /inicio prototipo. Elaboración propia.

- En la Ilustración 28, aparece el listado de usuarios del sistema, así como un formulario para crear un usuario. Proceda a introducir los datos en el formulario, y pulse el botón "Crear usuario" marcado en amarillo en la siguiente imagen. Al pulsar el botón, la página se recarga automáticamente y su usuario aparecerá en la tabla de usuarios. Si

quiere acceder a su usuario para modificarlo o eliminarlo, pulse sobre su URI de la lista de usuarios. Si no, proceda a pulsar el acceso directo a las bibliotecas, que también aparece en amarillo en la siguiente foto:

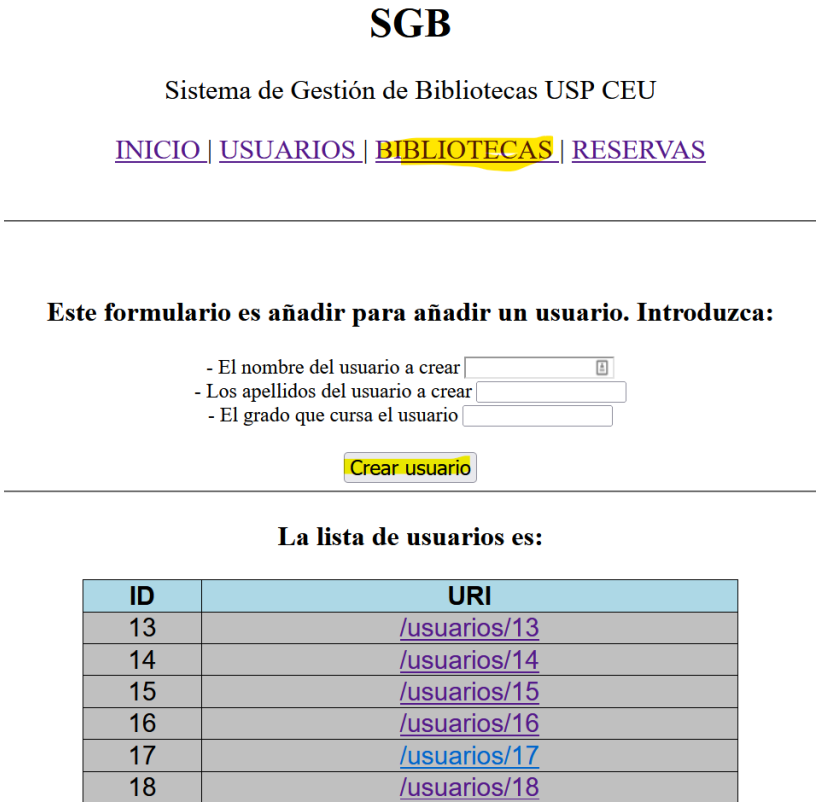
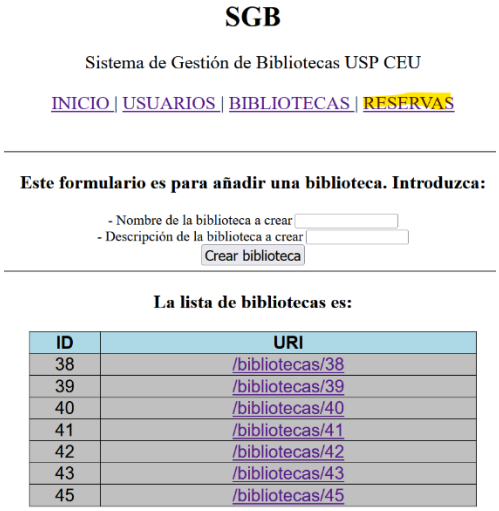


Ilustración 28: Captura interfaz /usuarios prototipo. Elaboración propia.

- En esta página se muestra un listado de las bibliotecas del sistema como se puede ver en la Ilustración 29. Si no quiere probar a añadir una biblioteca nueva, pulse el acceso directo a la biblioteca en la que quiere realizar la reserva (supongamos que la 38):



SGB

Sistema de Gestión de Bibliotecas USP CEU

INICIO | USUARIOS | BIBLIOTECAS | RESERVAS

Este formulario es para añadir una biblioteca. Introduzca:

- Nombre de la biblioteca a crear
- Descripción de la biblioteca a crear

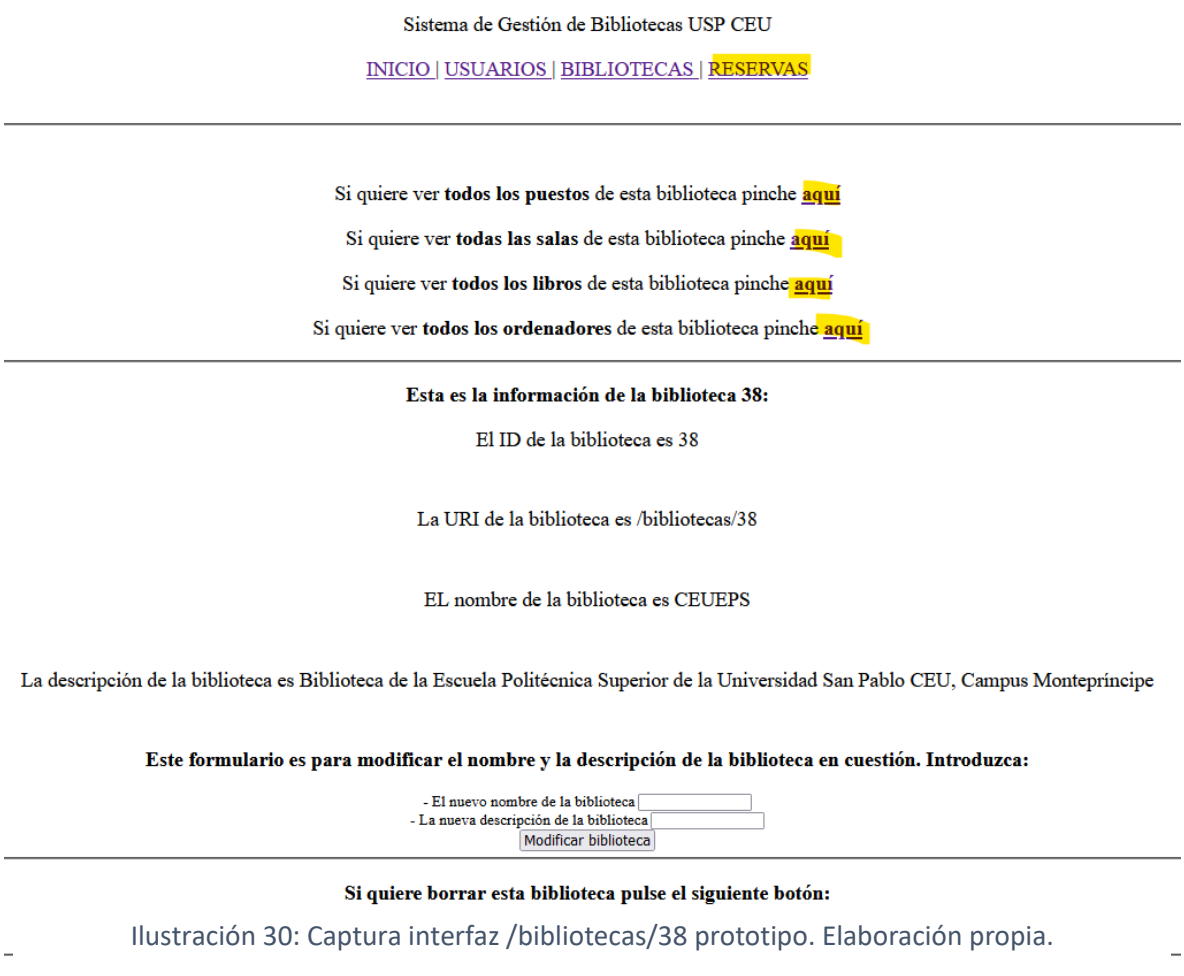
Crear biblioteca

La lista de bibliotecas es:

ID	URI
38	/bibliotecas/38
39	/bibliotecas/39
40	/bibliotecas/40
41	/bibliotecas/41
42	/bibliotecas/42
43	/bibliotecas/43
45	/bibliotecas/45

Ilustración 29: Captura interfaz /bibliotecas/ prototipo. Elaboración propia.

- En esta página puede ver la información de esta biblioteca (su identificador, su URI, su nombre, y su descripción), así como un



SGB

Sistema de Gestión de Bibliotecas USP CEU

INICIO | USUARIOS | BIBLIOTECAS | RESERVAS

Si quiere ver **todos los puestos** de esta biblioteca pinche **aquí**

Si quiere ver **todas las salas** de esta biblioteca pinche **aquí**

Si quiere ver **todos los libros** de esta biblioteca pinche **aquí**

Si quiere ver **todos los ordenadores** de esta biblioteca pinche **aquí**

Esta es la información de la biblioteca 38:

El ID de la biblioteca es 38

La URI de la biblioteca es /bibliotecas/38

EL nombre de la biblioteca es CEUEPS

La descripción de la biblioteca es Biblioteca de la Escuela Politécnica Superior de la Universidad San Pablo CEU, Campus Montepríncipe

Este formulario es para modificar el nombre y la descripción de la biblioteca en cuestión. Introduzca:

- El nuevo nombre de la biblioteca
- La nueva descripción de la biblioteca

Modificar biblioteca

Si quiere borrar esta biblioteca pulse el siguiente botón:

Ilustración 30: Captura interfaz /bibliotecas/38 prototipo. Elaboración propia.

formulario por si quiere actualizar la información de esta, y otro botón para borrarla. Cabe destacar, que, en la parte superior, se puede acceder a todos los puestos, salas, libros, y ordenadores de la biblioteca. Acceda a esos listados gracias a los accesos directos que aparecen en amarillo en la Ilustración 30, y escoja el puesto o sala que va a querer reservar, así como algún libro y/u ordenador si va a desear reservarlo también. Una vez memorice los recursos a reservar, acceda al apartado reservas pulsando en el menú superior (en amarillo en la Ilustración 30):

- Una vez accedido a /reservas , deberá rellenar el formulario añadiendo el identificador de usuario, el del elemento reservable que desea reservar, y la fecha en un formato concreto y pulsar el botón que aparece en amarillo en la Ilustración 31. Al pulsarlo, la página se recarga automáticamente y se añade la nueva reserva a la lista.

Ilustración 31: Captura interfaz /reservas prototipo. Elaboración propia.

- Si pulsa en la URI de su reserva, como por ejemplo en /reservas/63, se muestra en una primera instancia la información de la reserva:

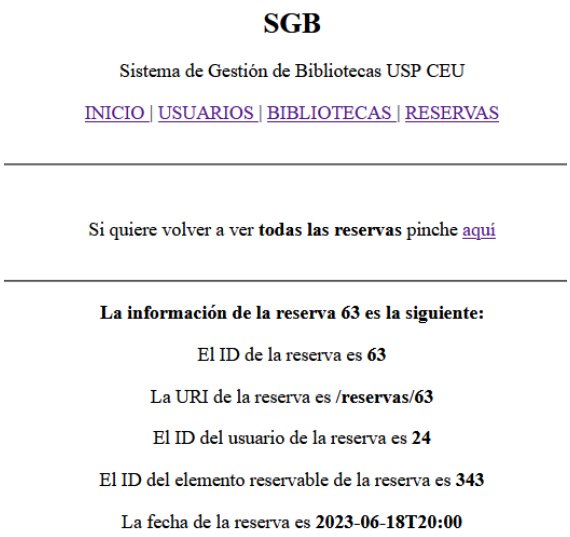


Ilustración 32: Captura interfaz /reservas/63 prototipo. Elaboración propia.

- Más abajo, aparecen los recursos extras asociados a la reserva normal del puesto o sala, como se muestra a continuación:

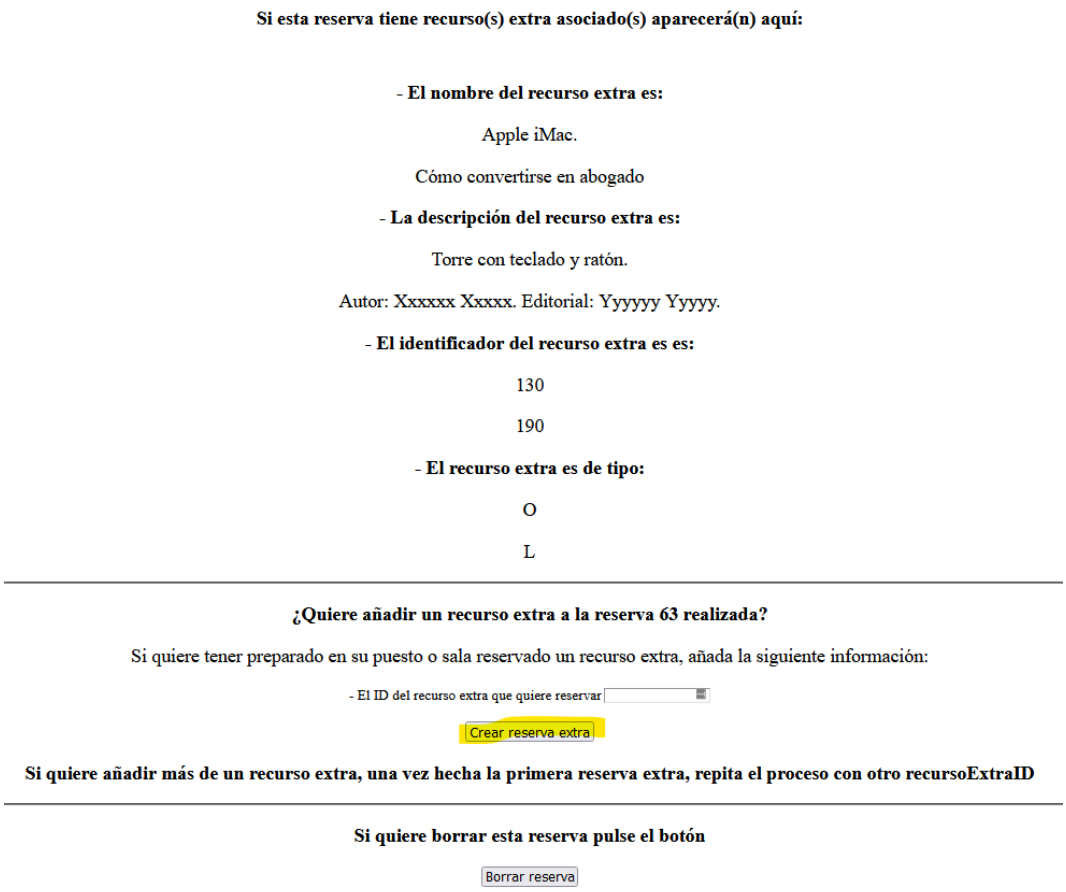


Ilustración 33: Captura 2 interfaz /reservas/63 prototipo. Elaboración propia.

- Para añadir un recurso extra, sólo necesita introducir su identificador y pulsar el botón amarillo de la Ilustración 33.