

UNIVERSIDAD DE ALMERIA

ESCUELA SUPERIOR DE INGENIERIA

“UAL Inventarium  
Una aplicación web  
para la gestión de  
inventario y  
préstamos”

Curso 2021/2022

**Alumno/a:**

Alejo Martín Arias Filippo

**Director/es:**

Juan Francisco Sanjuan Estrada  
Alfonso José Bosch Aran



## Resumen/Abstract

En el Departamento de Informática de la Universidad de Almería surgen problemáticas a la hora de controlar el registro de los equipos que se tiene ya que se desconoce su ubicación o es un proceso complicado el obtenerla. Poder solicitar el material del que disponen se convierte en un proceso tedioso. Todas estas problemáticas ocasionan adquisiciones redundantes en el Departamento y acumulación del inventario. La necesidad de una herramienta que pueda gestionar el inventario y agilizar el proceso de concesión de préstamos en el Departamento de Informática facilitaría en gran medida el trabajo de los técnicos y el control y seguimiento del material del que se dispone. El objetivo principal de este proyecto es la creación de un sistema de gestión de inventario y control de préstamos para el Departamento de Informática de la Universidad de Almería. Este sistema consistiría en el desarrollo de una aplicación web y una interfaz de programación de aplicaciones (API). Este desarrollo sería utilizando tecnologías como Node JS y Angular TypeScript. Nos ayudaremos de Docker para poder poner en funcionamiento el servidor, la API y la base de datos. A su vez iría ligado a una planificación desarrollada y detallada describiendo en profundidad todas las etapas por las que ha ido pasando la creación de la herramienta.

# Índice general

<b>1</b>	<b>Introducción</b>	<b>6</b>
1.1	Motivaciones . . . . .	6
1.2	Objetivos . . . . .	6
1.3	Planificación . . . . .	7
1.4	Estructuración del documento . . . . .	9
<b>2</b>	<b>Herramientas utilizadas</b>	<b>10</b>
2.1	Hardware . . . . .	10
2.2	Software . . . . .	10
2.3	Servicios . . . . .	13
<b>3</b>	<b>Fases previas a la construcción de la aplicación</b>	<b>15</b>
3.1	Reuniones iniciales con los clientes y especificación inicial . . . . .	15
3.1.1	Inicio de sesión y registro . . . . .	16
3.1.2	Barra de navegación . . . . .	17
3.1.3	Añadir objeto . . . . .	17
3.1.4	Grupo de objetos y objetos . . . . .	18
3.1.5	Usuarios . . . . .	18
3.1.6	Préstamos . . . . .	18
3.1.7	Configuraciones . . . . .	19
3.2	Planificación y elaboración del desarrollo del proyecto . . . . .	19
3.2.1	Redacción del proyecto . . . . .	19
3.2.2	Elaboración del desarrollo del proyecto . . . . .	21
3.3	Preparación del entorno de trabajo . . . . .	21
3.3.1	GitHub . . . . .	22
3.3.2	Visual Studio Code y Google Cloud, los mejores amigos . . . . .	22
3.3.3	Instalación de las extensiones de Visual Studio Code dentro de la MV . . . . .	24
3.3.4	Configuración de la máquina virtual . . . . .	24
<b>4</b>	<b>Construcción de la aplicación</b>	<b>30</b>
4.1	Introducción a Docker y Docker Compose . . . . .	30
4.1.1	¿Para qué sirve Docker? . . . . .	30
4.1.2	Docker Compose: el SimCity de los entornos . . . . .	30
4.2	Generación y puesta en marcha de la base de datos . . . . .	30
4.3	Diseño de la arquitectura de la aplicación . . . . .	33
4.3.1	¿Qué es una arquitectura de software? . . . . .	33
4.3.2	Clean Architecture o Arquitectura Limpia . . . . .	33
4.4	Creación y puesta en marcha de la Interfaz de Programación de Aplicaciones (API) . . . . .	34
4.4.1	Creación del sistema de directorios . . . . .	35
4.4.2	Funcionamiento de index.js . . . . .	37
4.4.3	Enrutamiento de la aplicación . . . . .	37
4.4.4	Consultas con la base de datos . . . . .	39
4.4.5	Gestión de archivos con Express . . . . .	39
4.5	Creación de UAL Inventarium . . . . .	41
4.5.1	Estructura de un proyecto Angular . . . . .	41

4.5.2	Disposición y elementos que hay en Inventarium . . . . .	43
4.5.3	Creación de componentes . . . . .	45
4.5.4	Definir el archivo de rutas . . . . .	47
4.5.5	Definir el proxy en nuestra aplicación . . . . .	49
4.5.6	Compilar el sitio Web . . . . .	49
<b>5</b>	<b>Diseño y funcionamiento final</b>	<b>50</b>
5.1	Inicio de sesión y registro de usuario . . . . .	50
5.2	Componentes principales . . . . .	50
5.2.1	Usuario . . . . .	51
5.2.2	Objeto . . . . .	52
5.2.3	Grupo de objetos . . . . .	52
5.2.4	Préstamos . . . . .	52
5.3	Componentes de procesos . . . . .	52
5.3.1	Proceso de creación de grupo de objetos y objetos . . . . .	53
5.3.2	Procesos de creación y modificación . . . . .	54
5.3.3	Proceso de creación, selección y eliminación de ubicaciones . . . . .	54
<b>6</b>	<b>Deploy del sitio web</b>	<b>57</b>
6.1	Levantar el servidor para la API . . . . .	57
6.2	Levantar el servidor Web . . . . .	59
6.2.1	Configuración del ruteo de la web . . . . .	59
6.2.2	Configuración del proxy inverso de la web . . . . .	60
<b>7</b>	<b>Pruebas durante el desarrollo y corrección de errores</b>	<b>61</b>
7.1	Pruebas durante el desarrollo . . . . .	61
7.1.1	Postman . . . . .	61
7.1.2	Selenium . . . . .	61
7.1.3	Herramientas de Desarrollador de Google Chrome . . . . .	62
7.2	Corrección de errores . . . . .	63
7.2.1	Cambio de la base de datos . . . . .	63
7.2.2	Cambios de solicitudes . . . . .	63
7.2.3	Conexión entre máquinas de Docker Compose . . . . .	64
<b>8</b>	<b>Conclusiones y posibles mejoras</b>	<b>65</b>
8.1	Posibles mejoras . . . . .	65
8.2	Conclusiones . . . . .	65
<b>Referencias</b>		<b>67</b>

# Índice de figuras

3.1	Diseño principal de la base de datos . . . . .	16
3.2	Diseño principal del sistema de navegación . . . . .	17
3.3	Diagrama de casos de uso del funcionamiento principal de la herramienta Inventarium . . . . .	18
3.4	Diseño del inicio de sesión y del registro . . . . .	19
3.5	Diseño de la barra lateral de búsqueda . . . . .	20
3.6	Diseño de la característica para poder añadir fungibles e inventarios . . . . .	26
3.7	Jerarquía de elementos que derivan de grupo de objetos . . . . .	27
3.8	Grupo de objetos a la izquierda y objetos que contiene a la derecha . . . . .	27
3.9	Grupo de usuarios a la izquierda y usuario único a la derecha . . . . .	28
3.10	Solicitud del préstamo de un objeto . . . . .	28
3.11	Visualización de los campos de configuración a la izquierda y creación de estos a la derecha . . . . .	28
3.12	Página principal de GitHub . . . . .	29
3.13	Creando un nuevo repositorio dentro de GitHub . . . . .	29
4.1	Diseño final de la base de datos . . . . .	31
4.2	Clean Architecture . . . . .	34
4.3	Interfaces y servicios de UAL Inventarium . . . . .	44
4.4	Rutas disponibles en la aplicación . . . . .	48
5.1	A la izquierda el inicio de sesión y a la derecha el formulario de registro de usuario	50
5.2	Representación de un usuario en la aplicación . . . . .	51
5.3	Representación de un usuario y posibles acciones sobre él en la aplicación . . . . .	51
5.4	Objeto y acciones que podemos hacer sobre él . . . . .	52
5.5	Grupo de objeto y acciones que podemos hacer sobre él . . . . .	53
5.6	Funcionamiento y descomposición de préstamos . . . . .	54
5.7	Funcionamiento y descomposición de préstamos . . . . .	54
5.8	Cuadros para iniciar la búsqueda de grupos de objetos . . . . .	55
5.9	Añadir imagen y datos al grupo de objetos . . . . .	55
5.10	Creación y modificación de elementos . . . . .	55
5.11	Número de objetos y localización . . . . .	56
5.12	Funcionamiento y descomposición de préstamos . . . . .	56
5.13	Selección o creación del edificio . . . . .	56
5.14	Selección o creación de la planta . . . . .	56
5.15	Selección o creación de la ubicación . . . . .	56
7.1	Creando un usuario con Postman . . . . .	62
7.2	Registro de un usuario con Selenium . . . . .	63
7.3	Sección <i>Network</i> dentro de las herramientas de desarrollador de Google Chrome .	64

# 1 Introducción

Son varios los caminos que han llevado al desarrollo de este proyecto. En este capítulo tendré la oportunidad de poder hablar de cada uno de ellos. Desde empezar a estudiar ingeniería informática hasta la mentalidad que he ido adquiriendo a lo largo de estos años gracias a ella.

## 1.1. Motivaciones

La motivación principal ha sido la ingeniería informática y es que sin ella no sería quién soy ahora. Para bien y para mal, ha ido siendo una evolución de pequeños pasos empezando en primero de carrera hasta llegar a saber una pizca de todo este mundo de tecnología ya acabando cuarto. Todo empezó desde pequeño, desde que vi esos monitores gigantes, los discuetes y el Age of Empires 1 a la edad de cuatro años.

Mi madre tuvo un novio, Martín se llamaba, que nos trajo el mundo de la tecnología a la casa. Ella también era otra fan tecnológica, pues al haberme tenido con tan solo 17 años siempre tuvo esa mentalidad más abierta con todo el tema de nuevas herramientas. Creo que esto me acabó ayudando bastante.

Luego de esta pequeña época vino una un poco más oscura que consistía en una vida de cero tecnología. Influía bastante el aspecto de vivir en Argentina y que el precio de los aranceles fuera bastante alto influyendo en los productos tecnológicos en su mayoría. Pasar al lado de los escaparates y ver esas pantallas brillantes con juegos, películas, canciones... era un martirio para mí.

Otro aspecto muy importante y el cual también he de agradecer fue la concesión que hizo la Junta de Andalucía allá por 2009 de aquellos pequeños ordenadores verdes. Esos sí que fueron gasolina para mis aspiraciones y para terminar hoy aquí, dentro del grado de Ingeniería Informática de la Universidad de Almería.

Mi segunda motivación principal para el desarrollo de este proyecto fue la beca extracurricular que publicó el Departamento de Informática de la Universidad de Almería, cuyo director en el momento de la concesión y en la fecha actual de redacción de este documento es Juan Francisco Sanjuan Estrada, el tutor de este proyecto.

Gracias a ella me sentí motivado para poder afianzar la actividad desempeñada en aquel pequeño trayecto de tiempo, unos seis meses, a la redacción de este trabajo.

El trabajo no se basa en lo que exactamente realicé en aquella beca, sino en una refactorización/transformación de aquel proyecto para poder convertirla en algo “fresco” podríamos llamarlo.

Y aquí pasamos al tercer y último punto de este apartado. La transformación del proyecto, el aplicar el concepto de ingeniería para poder transformar algo que era un software cerrado, con poca modularidad, difícil de entender y de ampliar en algo que merezca la pena tener. Un software que combine tecnologías y metodologías de hoy en día. Un ejemplo a seguir.

## 1.2. Objetivos

El objetivo principal de este proyecto es la creación de un sistema de gestión de inventario y préstamos para el Departamento de Informática de la Universidad de Almería.

Este sitio web tiene que cumplir con unos requisitos principales, que son:

- Llevar un registro del inventario del Departamento de Informática ubicado en el edificio Científico Técnico III
- Que el estudiantado y el personal docente e investigador realicen solicitudes de préstamos para los distintos elementos ofertados dentro de la página
- Que los técnicos de servicio puedan gestionar estas solicitudes más el seguimiento del inventario dentro del edificio

Por estos puntos se entiende que la motivación principal de la herramienta es la de gestionar y organizar préstamos.

Luego de los requisitos principales que había que cumplir se presentaban los secundarios.

- La página web tenía que disponer de un modo adaptativo para las versiones móviles
- Había que incorporar la posibilidad de realizar un importado de datos gracias al procesamiento de una tabla de datos que es lo que antiguamente utilizaban los técnicos del departamento
- Había una serie de datos que tenían que almacenar los objetos añadidos a este sistema los cuales podían ser de tres tipos: inventarios, fungibles o kits
- Se permitiría ver un seguimiento de los préstamos realizados sobre los objetos.

En base a estos objetivos principales y secundarios se realizó la construcción de la aplicación.

## 1.3. Planificación

La planificación del proyecto se dividirá en cinco grandes grupos: reuniones iniciales con los clientes, planificación y elaboración del desarrollo del proyecto, preparación del entorno de trabajo, construcción de la aplicación, testeo y comprobación de la aplicación y comprobación de errores.

### 1.3.1. Reuniones iniciales con los clientes y especificación inicial

A pesar de basarse en la reconstrucción de una aplicación las reuniones con los clientes son iniciales. Estas se hicieron en un principio del desarrollo y se mantuvieron unas pocas más a lo largo del mismo.

### 1.3.2. Planificación y elaboración del desarrollo del proyecto

En el desarrollo del proyecto no solo se considera la planificación de las diferentes etapas para poder ver y realizar un seguimiento del proyecto sino que también importa la redacción y especificación de las diferentes actividades realizadas en cada una de estas. Este apartado de la planificación se desarrolla junto a todo el resto de apartados.

### 1.3.3. Preparación del entorno de trabajo

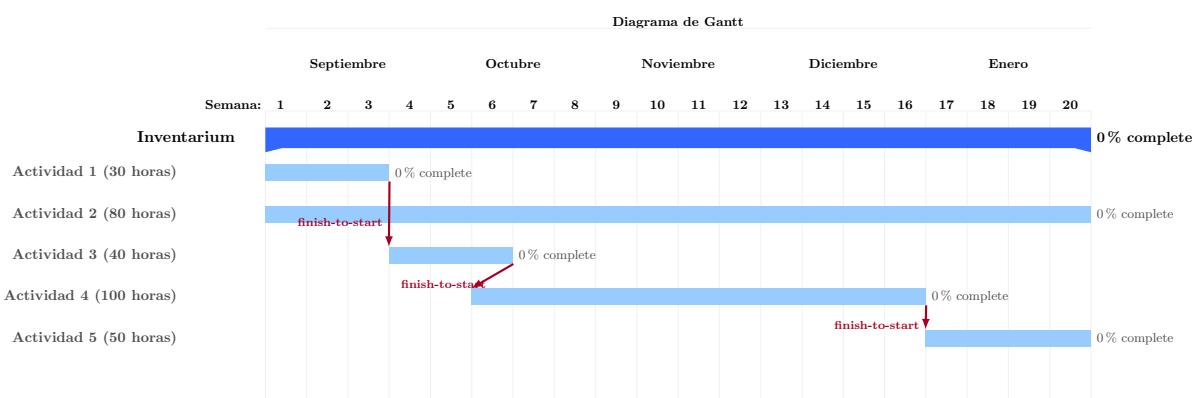
Los mayores casos de éxito tanto en producción como en mejoras de producto de una empresa se debe a la mejora de su entorno de producción. Gracias a algunas asignaturas del grado, a la investigación, a poder haber estado alrededor de un año en el entorno laboral y a la magnífica herramienta que es internet y todo lo que nos brinda se ha podido perfeccionar esta creación de entornos de trabajo.

### 1.3.4. Construcción de la aplicación

Fue uno de los apartados más difíciles en cierto momento, por lo que conllevaba la creación de la lógica de la aplicación, las interacciones que realiza la interfaz con el usuario y las distintas funcionalidades que tiene que presentar la misma. Esta sección unos dos años atrás pudo haber sido de las que más tiempo podrían llevar en realizarse pero gracias a los distintos frameworks que tenemos hoy en día para poder reutilizar componentes software y ahorrar pasos intermedios se ha ido volviendo más pequeña, que no quiere decir menos importante. Gracias a Angular 12 y a algunas tecnologías más que utilizaremos se podrá ver como la construcción no es tan complicada como en un principio parecía.

### 1.3.5. Testeo y comprobación de la aplicación y comprobación de errores

¿Qué es de una aplicación bien planificada, con una interfaz bien elaborada y unos requisitos satisfechos que presente errores? No es nada, y es por ello que un testeo intensivo en el momento que la aplicación finalmente haya sido publicada hará que mejore en esos apartados que anteriormente podría haber presentado fallos.



- **Actividad 1** (30 horas) Reuniones iniciales con los clientes
  - Reunión inicial con el director del proyecto
  - Reunión con los técnicos para investigar el dominio de la aplicación
  - Presentar primer diseño de la aplicación junto a una interpretación del dominio
- **Actividad 2** (80 horas) Planificación y elaboración del desarrollo del proyecto
- **Actividad 3** (40 horas) Preparación del entorno de trabajo
  - Creación del entorno en la nube Google Cloud
  - Creación del repositorio en GitHub
  - Creación del entorno virtual en Visual Studio Code
  - Contenerización de los distintos sistemas que se utilizarán en el desarrollo de la aplicación
- **Actividad 4** (100 horas) Construcción de la aplicación
  - Elaboración y creación de la base de datos
  - Desarrollo de la API
  - Desarrollo del sitio web
- **Actividad 5** (50 horas) Testeo, comprobación de la aplicación y comprobación de errores

## 1.4. Estructuración del documento

La estructura de este documento de trabajo fin de grado es la siguiente:

### 1.4.1. Herramientas utilizadas

Donde se hará un pequeño resumen de todas las herramientas tanto físicas como tecnológicas utilizadas para la construcción de este software.

### 1.4.2. Fases previas a la construcción de la aplicación

Donde se describirán y detallarán todas las fases previas a la construcción de la aplicación, desde las reuniones iniciales con los clientes, la creación de una especificación inicial del proyecto y la planificación y elaboración del desarrollo del proyecto.

### 1.4.3. Construcción de la aplicación

Aquí se tratarán los procesos de construcción por los que ha ido pasando la aplicación.

### 1.4.4. Diseño y funcionamiento final

Se revisará el aspecto final del sitio web y toda la funcionalidad que presenta.

### 1.4.5. Deploy del sitio web

Se explicarán los distintos procesos por los que tiene que pasar el sitio web para poder ser subido a internet.

### 1.4.6. Pruebas durante el desarrollo y corrección de errores

Se expondrán las distintas tecnologías utilizadas para la comprobación del correcto funcionamiento de los componentes desarrollados en este trabajo. También se explicarán problemáticas surgidas durante el desarrollo y su posterior resolución.

### 1.4.7. Conclusiones y posibles mejoras

Se realizará una evaluación de las distintas fases de desarrollo, su desempeño en ellas y posibles mejoras que se hubieran podido realizar.

### 1.4.8. Bibliografía

Se hablará sobre la bibliografía en la que se han fundamentado cada uno de los pasos de elaboración de este proyecto.

## 2 Herramientas utilizadas

Este capítulo tratará sobre las herramientas utilizadas durante el desarrollo del proyecto. Puede resultar de bastante interés debido a la inclusión de nuevas herramientas que irán en conjunción al uso de metodologías de trabajo novedosas.

Al igual que las tecnologías van avanzando las empresas también lo hacen, y la amplia gama de tecnologías gratuitas que están a nuestro alcance conlleva que el conocimiento de las mismas sea de vital importancia en el momento de construcción de una aplicación.

### 2.1. Hardware

Dentro del apartado de hardware se dispone de dos ordenadores. Su procesador no conlleva relevancia en el desarrollo de la aplicación debido a la utilización de servicios en la nube.

#### 2.1.1. Ordenador de sobremesa

El cual contiene como procesador un Xeon E5-2620 V3. 16GB de RAM DDR3. Una tarjeta gráfica RTX 570 de 4GB DDR5. Tiene 256GB de memoria SSD y 1TB de memoria HDD.

#### 2.1.2. Ordenador portátil

Un MacBook Air M1 de 2020 con 8GB de RAM y 256GB de almacenamiento.

### 2.2. Software

#### 2.2.1. Entorno de desarrollo

El entorno de desarrollo y desde donde se hará casi absolutamente todo será Visual Studio Code. Un editor de código desarrollado por Microsoft que soporta varias distribuciones de sistemas operativos, entre ellas: Windows, Mac Os y Ubuntu.

Una de las características de esta herramienta que la hacen la predilecta de varios desarrolladores es el gran soporte que tiene por parte de la comunidad. Tiene un mercado de plugins bastante grande que apoya la creación continua de código para todos los desarrolladores.

Dispone de integraciones con Git, resaltado en errores de sintaxis, finalización de código y hasta conexión remota a otros entornos de trabajo mediante SSH.

Otra enorme ventaja que presenta es el consumo de memoria que tiene, bastante pequeño. Es un programa para ordenadores de todos los tamaños y precios, un software gratuito y una herramienta increíblemente potente al alcance de todos.

#### 2.2.2. Redacción del documento

Estas líneas están siendo escritas ahora mismo desde LaTeX. LaTeX es un sistema de composición de textos que está formado mayoritariamente por órdenes construidas a partir de comandos TeX. En un principio no estaba seguro de qué herramienta utilizar, ya que la posición de varios profesores respecto a esta herramienta era bastante férrea pero Word siempre había ido agarrado a mi mano desde comienzos del instituto.

Luego de pasar de Word a LaTeX y de LaTeX a Word bastantes veces no fue hasta que mi

profesora Rosa, en una de mis visitas matinales a su despacho me dijo: Yo hice mi TFG en LaTeX.

No me lo podía creer y al comprobar la fecha de publicación de este programa de procesado de textos me sorprendí al ver que su lanzamiento oficial fue en 1980. “Si el programa ha durado tanto es que algo de importante tendrá” pensé. Y aquí me hallo redactando este documento con un programa que facilita el control de versiones de Git de una manera asombrosa. Facilita también los procesos de documentación y disposición de las diferentes subsecciones. Y, lo que más me gusta sin lugar a dudas, que puedo realizar una separación de cada capítulo por documentos separados y es que a mí, el tener las cosas modularizadas, me puede.

### **2.2.3. Diseño y creación de la base de datos**

La base de datos ha sido diseñada con MySQL Workbench. Esta es una herramienta visual de diseño de bases de datos, capaz de administrar, diseñar, gestionar y mantener bases de datos. Su primera versión fue publicada en 2005.

En un principio la base de datos fue exportada para que el deploy también se realizará en un servidor MySQL pero el versionado de estos scripts para la creación de las bases de datos ocasionan bastantes problemas. MariaDB ofrece una compatibilidad perfecta con MySQL, además, es software libre.

MariaDB es un sistema de gestión de bases de datos derivado de MySQL con licencia GPL. Fue escrito por el mismo creador que MySQL, debido a que este vendió su producto (MySQL) a Oracle, dejando este de ser software libre.

Para la gestión de la base de datos se ha utilizado PHPMyAdmin. Un gestor de bases de datos que se utiliza desde páginas web. Facilita cualquier tipo de inspección que un técnico tenga que realizar por no poder hacerlo desde la aplicación web de inventarium.

### **2.2.4. Diseño del sitio web**

Para realizar el diseño de la aplicación se ha utilizado Adobe XD. Adobe XD es un editor de gráficos vectoriales desarrollado y publicado por Adobe Inc para diseñar un prototipo de la experiencia del usuario para páginas web y aplicaciones móviles.

Adobe XD apoya a los diseños vectoriales y a los sitios web wireframe creando prototipos simples e interactivos con un solo click.

### **2.2.5. Diseño de diagramas**

El diseño del funcionamiento de la aplicación y el análisis de requisitos fue hecho mediante Visual Paradigm Professional. Visual Paradigm es una herramienta CASE, Ingeniería de Software Asistida por Computación, que brinda un gran abanico de herramientas para la ayuda del diseño, creación, planificación, análisis, documentación y un largo etcétera más de proyectos informáticos.

Esta herramienta ha sido concebida para soportar el ciclo de vida completo del proceso de desarrollo de un software a través de la representación de todo tipo de diagramas.

### **2.2.6. Diseño y modificación de imágenes**

Para el diseño y modificación de imágenes se utilizará Adobe Photoshop 2021. Adobe Photoshop es un editor de fotografías desarrollado por Adobe Systems Incorporated. Usado principalmente para el retoque de fotografías y gráficos.

### 2.2.7. Contenerización del sistema

¿Qué significa contenerización del sistema? Contenerización es la práctica de transportar mercancías en contenedores con forma y tamaño uniformes. Y con la contenerización del sistema se pretende hacer lo mismo pero para sistemas operativos informáticos.

Esto se puede desarrollar gracias a Docker. Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos.

Simplificando la definición, Docker permite desplegar y borrar máquinas de forma fácil y eficiente. Modularizando su usabilidad y evitando que si ocurre algún error en una de ellas no se rompan las demás.

#### Docker Compose

Docker Compose es una herramienta de Docker que sirve para poder levantar varios contenedores a la vez y de forma simultánea. Gracias a esta utilidad podemos desplegar entornos web en tiempos cortos. Además brinda utilidades haciendo, por ejemplo, que una máquina virtual se reinicie al detenerse.

### 2.2.8. NodeJS

NodeJS es un entorno de tiempo de ejecución de JavaScript. Este entorno de tiempo de ejecución en tiempo real incluye todo lo que se necesita para ejecutar un programa escrito en JavaScript. Gracias a él se pueden utilizar varias herramientas explicadas a continuación.

### 2.2.9. Deploy de la Interfaz de Programación de Aplicaciones(API)

El servicio Web se levanta desde Express. Express es un framework back-end para NodeJS que está diseñado para levantar sitios webs y APIs. Es la herramienta predilecta para levantar servicios web dentro del entorno de Node.

### 2.2.10. Deploy del sitio web

Nginx es un servidor web/proxy de código abierto. Su primer lanzamiento fue en 2004. Presenta ventajas en el momento de recibir un número elevado de solicitudes concurrentes y es utilizado por compañías de alto perfil tecnológico como Google, Facebook o Twitter.

### 2.2.11. Desarrollo del sitio web

El desarrollo del sitio web se realizará con Angular 12. Este es el punto más importante de esta subsección.

Angular es un framework para aplicaciones web desarrollado en TypeScript, de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador (MVC), en un esfuerzo para hacer que el desarrollo y las pruebas sean más fáciles. Angular se basa en clases de tipo Componentes, que facilitan el “binding” de los datos, es decir, disponer de una variable en el controlador cuyo valor sea visualizado en la vista.

Angular es la evolución de AngularJS (la versión de Angular que usaba JavaScript) aunque incompatible con su predecesor.

Este framework ha facilitado el desarrollo potencialmente. Y es que en la anterior versión de la aplicación se habían utilizado únicamente PHP, Javascript y CSS. Es más, se había conseguido desarrollar un modelo basado en jQuery para que el sitio web se ubicara en una única página y fuera refrescando los datos de forma interactiva y fluida.

Angular fue un cambio total, la gestión de componentes es lo que le da la vida en su totalidad y es que, ¿qué sería un desarrollo software sin que presente una modularidad en sus componentes? Angular te lo ofrece, y te da más. Se puede definir todo el dominio de la aplicación dentro de ella y crear los servicios para contactar con la API. Luego se puede definir el sistema de rutas que deba tener la aplicación junto al resto de componentes que tienen que interactuar con el usuario.

No sería posible o sería muy complicado poder realizar un desarrollo planificado: con sus diagramas de requisitos, de base de datos, diseño de componentes y estructuración de vistas sin este framework.

### 2.2.12. Pruebas de UAL Inventarium

Para realizar las pruebas se han utilizado tres herramientas:

- **Postman:** Aporta una serie de utilidades para poder mandar solicitudes HTTP a las direcciones que queramos. Es ideal para poder probar APIs o aplicaciones que utilicen proxy para sus consultas con la base de datos.
- **Selenium:** Esta herramienta permite crear pruebas de software en nuestro sitio web. Provee de una herramienta de gran utilidad que es la de poder grabar y reproducir secuencias de prueba con tan solo un botón. Esta utilidad ayuda en gran medida a la hora de tener que ir probando campos de formulario y no estar todo el rato ingresando los datos mecánicamente.
- **Herramientas de Desarrollador de Google Chrome:** Esta columna derecha que incorpora Google Chrome es de gran utilidad para ver la consola, modificar estilos, ver los datos que estamos almacenando en caché y capturar tráfico web de las páginas.

## 2.3. Servicios

### 2.3.1. Sistema de control de versiones

El sistema de control de versiones se realizará gracias a GitHub. GitHub es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git. Se utiliza principalmente para la creación de aplicaciones.

El software que opera GitHub fue escrito en Ruby on Rails. Desde enero de 2010, GitHub opera bajo el nombre de GitHub, Inc. Anteriormente era conocida como Logical Awesome LLC. El código de los proyectos alojados en GitHub se almacena normalmente de forma pública.

El 4 de junio de 2018 Microsoft compró GitHub por 7500 millones de dólares. Al principio, el cambio de propietario generó preocupaciones y la salida de algunos proyectos de este repositorio, sin embargo no fueron representativos. GitHub continúa siendo la plataforma más importante de colaboración para proyectos Open Source.

### 2.3.2. Gestión y creación de máquinas virtuales

La Universidad de Almería posee OpenStack. Un servicio que permite poder crear y destruir máquinas virtuales además de poder ubicarlas en una infraestructura de red. El problema que ocurrió OpenStack fue la causado por la configuración de red de la universidad. Y es que había unos determinados puertos que por mucho que se abriesen en la máquina virtual el cortafuegos de la Universidad impedía acceder a ellos. Como se verá más adelante a lo largo de estas páginas esto no tuvo por qué haber sido un problema utilizando la tunelización.

Debido a los problemas anteriormente mencionados con OpenStack se empezaron a buscar soluciones que brindaban las distintas “empresas top” del sector.

Se irán evaluando por precio, rendimiento e interfaz. Se han probado cada una de ellas.

### **Amazon Web Services**

En precio se encuentra en un rango medio. Disponía de buena documentación y la interfaz era compleja pero fácil de usar. La conectividad y el acceso a las máquinas era rápido pero los planes y servidores eran algo limitados.

### **Microsoft Azure**

El precio es alto. El manejo y creación de base de datos se complica por el plan de precios que presentan que están almacenados en unos tipos de monederos. Da el parecer de que estas funcionalidades están mal implementadas dificultando el aprendizaje inicial. Los servidores y la configuración de los mismos son más limitados que los de Amazon Web Services.

### **Google Cloud**

Ofrecen el mejor precio de la competencia. El problema es que la primera vez que entras te encuentras con muchas soluciones que ofrece Google con servicios en la nube. Es una sobreinformación de cosas que realmente no necesitas para desplegar una simple máquina virtual. Una vez te ubicas que tienes que moverte en el entorno de “Compute Engine” todo se vuelve mucho más fácil. Las máquinas virtuales son fáciles de desplegar y de personalizar y presentan una alta gama de servidores disponibles, al fin y al cabo Google está en todo el mundo.

La elección para la realización de este proyecto fue Google Cloud.

---

### **3 Fases previas a la construcción de la aplicación**

La temporalidad del proyecto no es exacta del todo ya que los puntos 3.1 y 3.2 se hicieron aquellos dos primeros meses de prácticas extracurriculares mientras que los otros son totalmente nuevos.

Para el seguimiento de las fases y tareas se han estado utilizando la herramienta Trello y las nuevas funcionalidades de GitHub que incorpora tableros parecidos a los de Trello.

#### **3.1. Reuniones iniciales con los clientes y especificación inicial**

La reunión inicial fue con Juan Francisco Sanjuan Estrada quien explicó los distintos objetivos que se tenía pensado para la aplicación y los puntos de vista de los técnicos. Quienes utilizaban hojas de cálculo para gestionar el inventario y registrar los préstamos.

Dentro de las hojas de cálculo se detallaban exhaustivamente los objetos y también se indicaban sus ubicaciones actuales dentro del Edificio Científico Técnico III.

El problema es que esto no estaba tan organizado como parecía en un principio debido a que la gestión no la realizaba únicamente un técnico sino que eran tres. Es decir, había tres filosofías distintas a la hora de ir gestionando una parte del inventariado del Departamento de Informática.

Después de estas aclaraciones Juan comentó que no tenían un diseño en mente en el Departamento por lo que las propuestas iniciales de diseño iban a ser libres siempre y cuando se satisfacieran los requisitos principales de esta como la adición y eliminación del inventario y la concesión de préstamos.

Luego de esta primera reunión se acordó en que podrían visualizarse los archivos Excel de los técnicos. Estos compartieron sus ficheros y desde ahí se pudo empezar a realizar la definición de los diferentes campos que irían ligados a cada tabla en la base de datos. Desde este punto se haría una propuesta inicial.

La propuesta inicial fue la de la figura 3.1.

La lógica de la aplicación consistiría en generar un grupo de objetos de un determinado tipo, inventario o fungible, 1 o 0. Dentro de este se contendrían objetos que irían vinculados a una ubicación. También los objetos podrían contener una posible configuración.

Esta configuración ¿para qué sirviría? Resulta que los técnicos aparte de manejar el inventariado disponible en la universidad también manejan las claves y accesos a esos determinados objetos, como puede ser el armario donde se ubiquen, sus direcciones mac o ip, las bocas de conexión con las regletas y más.

Este objeto tendría la posibilidad de tener varios préstamos, aunque es verdad que sería uno activo por persona, siempre dejando un registro de los anteriores usos que se hayan realizado del mismo. Este préstamo tendría que ir ligado obligatoriamente a un usuario. Contiene un campo de “retiradoPor” en el caso de que un docente o investigador quiera realizar un préstamo para su alumno.

La propuesta inicial fue aceptada por el grupo de los técnicos y por el director del proyecto así que se pudo continuar a las siguientes etapas. Más adelante se realizaron unas modificaciones en los datos que manejarían los objetos por lo que las columnas de las tablas expuestas no serían

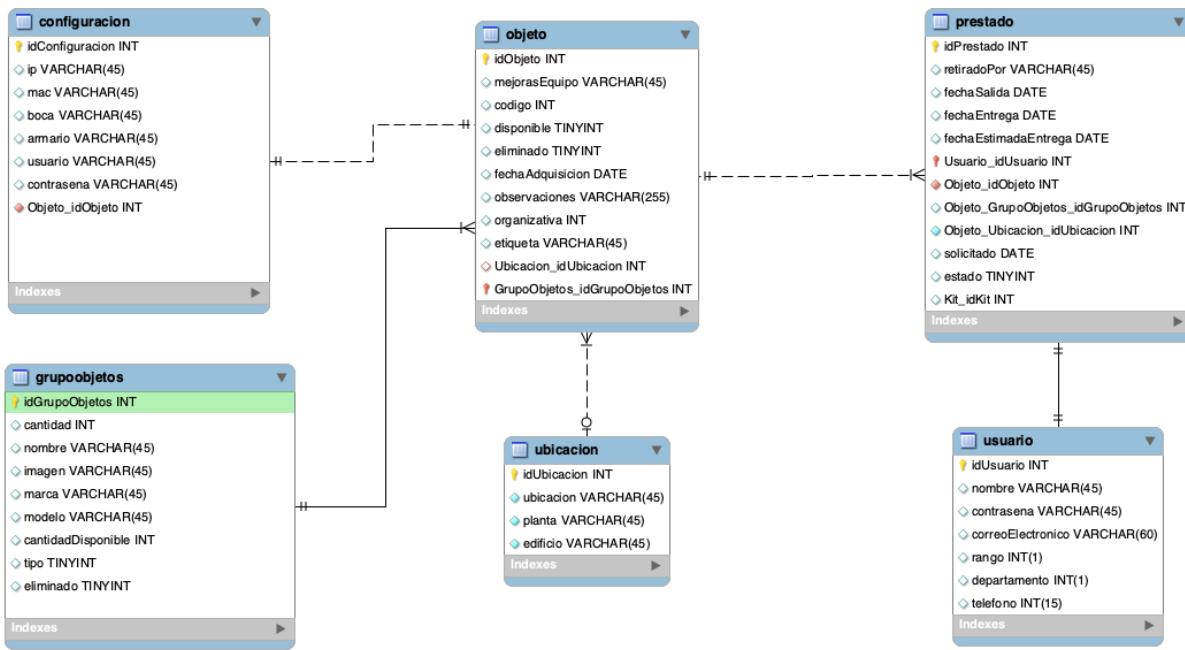


Figura 3.1: Diseño principal de la base de datos

las definitivas.

Se prepararon las propuestas iniciales de diseño mediante la utilización de la herramienta de Adobe XD. El aprendizaje de la herramienta es rápido y fácil de usar. Adobe XD le dió un toque de frescura y profesionalidad a las propuestas de diseño.

Las propuestas iniciales fueron ligadas a un diseño móvil ya que desde el primer momento se buscó que la aplicación fuera responsive, es decir, que desde un dispositivo móvil fuera fácil poder utilizarla. Esto influyó luego a la hora de realizar el diseño ya que se basó en un funcionamiento de “cards”, es decir, componentes web con forma de carta y con fácil adaptabilidad a diseños móviles.

El esquema de navegación de la propuesta inicial quedaría tal como se puede observar en la Figura 3.2.

Donde se tienen dos roles de usuario, un personal docente o investigador y un técnico. Después del análisis hecho en la entrevista y correos mantenidos más tarde con el cliente se obtuvo el siguiente resultado de la Figura 3.3.

### 3.1.1. Inicio de sesión y registro

El registro del usuario no conllevaba un registro instantáneo del mismo ya que este tiene que ser dado de alta por un técnico del sistema. Pueden observarse los diseños en la figura 3.4.

Dentro del inicio de sesión se puede ver un campo interesante a considerar y es el de departamento. Una de las solicitudes que hizo el director del proyecto era poder agrupar a los usuarios dentro de departamentos. En este caso refiriéndose a dentro del personal docente e investigador que pertenecen al Departamento de Informática.

Los valores del departamento en la base de datos pueden ser los siguientes:

- 0 = Departamento de informática (valor por defecto).
- 1 = Ingeniería de sistemas y automática.
- 2 = Lenguaje y sistemas informáticos.
- 3 = Ciencias de la computación e inteligencia artificial.

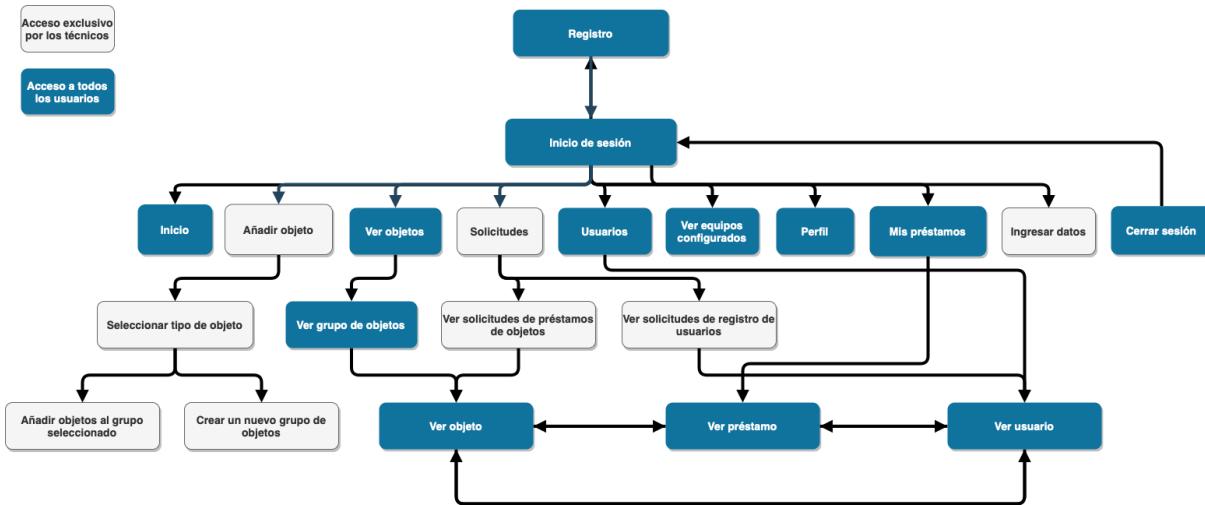


Figura 3.2: Diseño principal del sistema de navegación

- 4 = Arquitectura y tecnología de computadores.

Otro campo para comprobar es el del correo electrónica institucional teniendo que ser este con la extensión **@inlumine.ual.es** o **@ual.es**.

### 3.1.2. Barra de navegación

La barra de navegación vertical tuvo una pequeña modificación que era incorporar una pequeña sección de inicio donde aparecieran información pertinente y relevante para el usuario. Esto será explicado más adelante en la parte de desarrollo. Puede observarse en la figura 3.5.

Dentro de ella se pueden ver cinco apartados que derivan en varios componentes visuales:

- Añadir objeto: un acceso rápido donde poder añadir objetos dentro de grupos de objetos. Sección de menú solo visible para los técnicos.
- Ver objetos: apartado donde se visualizan los grupos de objetos, que contienen tanto el inventariado, fungibles y kits.
- Solicitudes: un menú donde se visualizan las solicitudes tanto de alta de usuarios como de objetos que le puedan llegar a los técnicos.
- Usuarios: componente visual donde cargan los usuarios registrados que hay dentro de la aplicación.
- Ver equipos configurados: aquí cargan únicamente objetos a los que se les haya aplicado una configuración. En caso de que no, no aparecerían.

### 3.1.3. Añadir objeto

La única diferencia entre añadir un objeto de tipo inventario y otro fungible es que en el de inventario hay que registrar su código. Siempre que se vaya a añadir uno de estos elementos tiene que ser dentro de un **grupo de objetos** que debe haberse creado anteriormente. Por el resto, los dos son idénticos y se le puede añadir el mismo tipo de información. Estas diferencias se pueden observar en la figura 3.6.

No solo existen los fungibles e inventarios sino que también se solicitó más tarde poder crear kits.

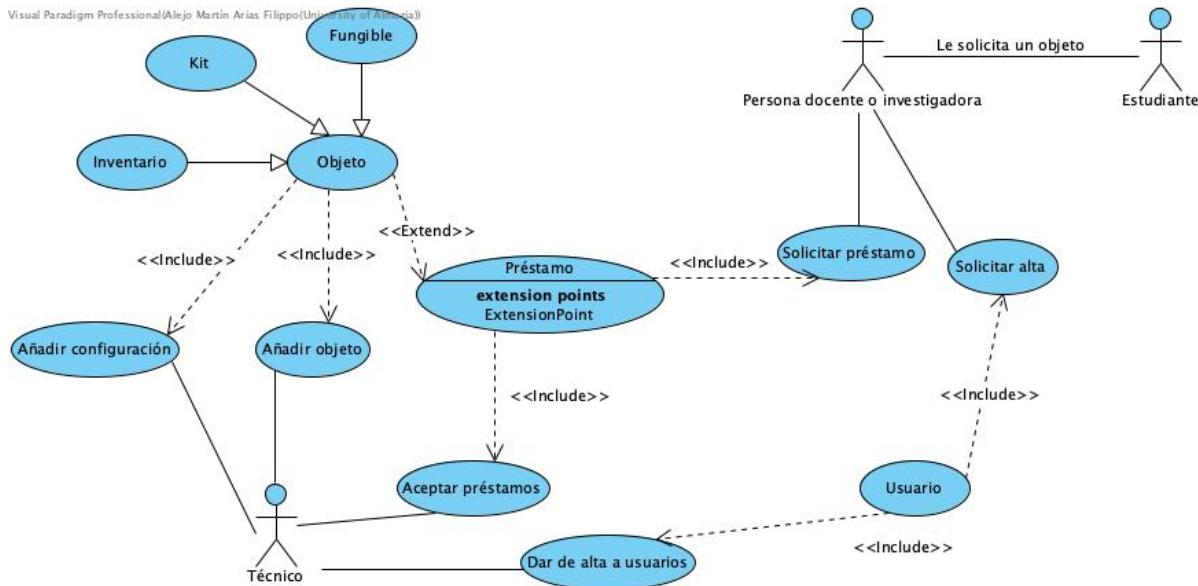


Figura 3.3: Diagrama de casos de uso del funcionamiento principal de la herramienta Inventarium

Un **kit** puede considerarse como otro nivel de agrupación. Ya que un kit contiene más objetos dentro suyo que llamaremos “objetos kit”. Para entender los niveles de agrupación puede observarse la figura 3.7.

### 3.1.4. Grupo de objetos y objetos

Dentro de la sección de “ver objetos” de la barra de navegación se podrá observar la visualización de los grupos de objetos de la aplicación. En las cuales habrá también objetos. Estos dos conjuntos se pueden modificar y además se pueden pedir solicitudes para el inventariado, fungibles y kits. Pueden observarse en la figura 3.8.

### 3.1.5. Usuarios

En este apartado se podrán visualizar los usuarios que están registrados en la aplicación. Permitiéndose poder acceder a cada uno de los perfiles para ver sus respectivos datos y sus solicitudes o historial de préstamos. Tal como se observa en la figura 3.9.

### 3.1.6. Préstamos

Los préstamos se pueden encontrar en tres diferentes estados:

- Préstamo en curso: una solicitud de préstamo aprobada por un técnico y que está en vigencia.
- Préstamos solicitado: donde aparecerá una solicitud en espera a ser aprobada por un técnico, en el caso de que el objeto solicitado esté actualmente con un préstamo en curso no se permitirá conceder el préstamo solicitado hasta que finalice el actual.
- Préstamo finalizado: un préstamo en curso que ha sido devuelto, la aprobación de devolución la tiene que conceder un técnico.
- Préstamo vencido: un préstamo en curso que ha sobrepasado su fecha máxima que se había comprometido a devolver. En caso de que quiera solicitar una ampliación tiene que hacerlo nuevamente generando una nueva petición.

Este diagrama esquemático ilustra el diseño de un formulario web para iniciar sesión y registrarse en la aplicación UAL-Inventarium. El formulario se divide en dos secciones principales: "Solicitar acceso" (Acceso) y "UAL-Inventarium".

- Solicitar acceso:** Sección que incluye campos para "Nombre completo" (campo vacío), "Correo electrónico institucional" (campo vacío), "Contraseña" (campo vacío) y "Repetir contraseña" (campo vacío). Hay un botón "Solicitar acceso" en este apartado.
- UAL-Inventarium:** Sección que incluye campos para "Correo electrónico" (campo vacío), "Contraseña" (campo vacío), "Departamento" (campo con lista desplegable que muestra "...") y "Teléfono" (campo vacío). Hay un botón "Iniciar sesión" en este apartado.

Figura 3.4: Diseño del inicio de sesión y del registro

Cada solicitud irá con un color representativo dependiendo de cuál sea su estado, una utilidad tanto para el usuario como para el técnico que van a realizar las acciones de ir solicitando y concediendo préstamos. Tal como muestra la figura 3.10.

### 3.1.7. Configuraciones

Un objeto puede ir ligado a una información extra o no. Esta información no es algo que vuelva más descriptivo el objeto sino que es una utilidad que se solicitó en el momento del desarrollo de la aplicación. Su objetivo es poder acceder a información a la que suelen recurrir los técnicos de forma bastante frecuente. Como la IP o dirección MAC de un ordenador o las credenciales para poder acceder a él. Gracias a esta distinción se permite que la información de los equipos que disponen de una configuración en particular estén en una sección para ellos solos facilitándoles el acceso a ese recurso a los técnicos. Tal como se ve en la figura 3.11.

## 3.2. Planificación y elaboración del desarrollo del proyecto

Esta sección implica dos apartados de relevante importancia.

### 3.2.1. Redacción del proyecto

Para la redacción del proyecto como se ha explicado anteriormente está utilizando la herramienta LaTeX. Esta serie de archivos viene incorporado al repositorio principal del proyecto dentro de una carpeta llamada documentación.

Nuestra carpeta de documentación a la vez está subdividida en varios apartados:



Figura 3.5: Diseño de la barra lateral de búsqueda

## Build

Carpeta que viene por defecto incorporada dentro de la “compilación” del proyecto que realiza LaTeX. Dentro de ella se genera el documento PDF que gracias a Visual Studio Code se va generando cada vez que se guarda un archivo del proyecto que vaya ligado o afecte al main.tex

## Bibliografía

En este apartado se añadirá la bibliografía utilizada durante la realización del proyecto.

## Capítulos

Dentro de esta carpeta se guardarán cada uno de los capítulos del documento. En el caso de que un capítulo presente apartados demasiados extensos se generará una carpeta con el nombre del capítulo y se meterán las secciones dentro de este. Gracias a hacer esto la modificación de cada zona del documento se hace de una forma mucho más cómoda.

## Diagramas

Aquí se almacenan los diagramas del proyecto. Estos pueden estar en archivos de imágenes aunque también pueden estar generados mediante LaTeX. Debido a la unicidad que presentan los diagramas estos no están separados en diferentes carpetas por cada capítulo que haya.

## Imágenes

Como su propio nombre indica se almacenan las imágenes del proyecto. Estas generalmente están divididas por carpetas con el nombre de los capítulos. En el caso de que sea demasiado extensa también se contempla la posibilidad de realizar el almacenaje mediante secciones.

## Include

Aquí se añadirán los archivos que generen inclusiones dentro de nuestro documento. Dependiendo de para qué sean estos añadidos irán con unas determinadas agrupaciones u otras.

## main.tex

Esto no es un directorio pero es un archivo de relevancia. Su contenido es escaso debido a la generalización que se presenta en la disposición de la documentación. Dentro de él se pueden encontrar las inclusiones al principio del documento. La adición del índice, los capítulos y de la bibliografía al final de la página. Es el archivo que utiliza LaTeX para generar nuestro documento PDF.

### 3.2.2. Elaboración del desarrollo del proyecto

Esta sección consiste en la explicación y descomposición de pasos que se realizará para la elaboración de la aplicación. Una vez hecha la lógica de la aplicación, su dominio y sus casos de uso se procederá a la maquetación del sistema.

#### Generación de la base de datos

Generación de la base de datos, para ello se utilizará MariaDB.

#### Construcción de la Interfaz de Programación de Aplicaciones (API)

La creación de la API debe realizarse de forma cuidadosa, ya que tiene que incorporar todas las reglas de negocio y cualquier paso adicional que haya que añadir en cada una de las peticiones. Estos pueden consistir, por ejemplo, en que cuando un objeto es creado hay que sumar una unidad dentro del campo de “objetos” y “objetosDisponibles” de su respectivo grupo de objetos.

#### Construcción de la aplicación

Se detallarán cada uno de los pasos más adelante debido a que la construcción de esta depende en pequeña medida al framework utilizado que en este caso es Angular. En todo caso el orden de creación de ficheros sería:

1. Creación de interfaces.
2. Creación de servicios.
3. Creación de componentes visuales.

## 3.3. Preparación del entorno de trabajo

Como decía Robert Owen, un reconocido empresario galés, en el siglo XVIII:

Mejorando el entorno se mejora al hombre.

El desarrollo de esta preparación previa ha supuesto un ahorro de tiempo bastante grande para el proyecto.

Esta sección irá estructurada en consonancia a la cronología de creación de cada parte del entorno. Empezando por la semilla de todo: Github.

### 3.3.1. GitHub

El proyecto se realizará en GitHub. Esto se hace de forma muy rápida y fácil.

1. Hay que dirigirse a la dirección de [github.com](https://github.com). Donde podrá observarse algo parecido a la figura 3.12.
2. En la parte superior derecha aparecerá “Sign up” donde se clicará para poder registrarse dentro de la web.
3. Una vez se haya realizado el registro se volverá a [github.com](https://github.com) y se clicará a la izquierda de “Sign up” que pone “Sign in” donde podrá iniciarse sesión.
4. Cuando se haya iniciado sesión hay que dirigirse a la parte superior izquierda clicando en “new” para crear un nuevo repositorio.
5. Se llenarán los campos que se solicitan para crear un nuevo repositorio. Se le dará el nombre de Inventarium al repositorio y se marcará en la casilla para que sea privado. También se procederá a añadirle un archivo del tipo Readme para poder describir partes del proyecto dentro de este. Por último se pulsará sobre el botón “Create repository” como puede verse en la figura 3.13.

Con estos sencillos pasos se ha realizado la creación del repositorio del proyecto.

### 3.3.2. Visual Studio Code y Google Cloud, los mejores amigos

Hoy en día lo nuevo y a lo que la sociedad se dirige es la nube. Dentro de ella se pretenden que se realicen todos los procesos. La mayoría de herramientas y servicios que se nos ofrece cumplen un modelo de caja negra. Es decir, se puede interactuar con ellos y recibir respuestas pero no puede verse qué procesos ocurren dentro de aquella caja.

Por esta razón cada vez se empieza a disponer menos de un software como tal y se empiezan a pasar a servicios en línea. Esto no quiere decir que se dejen de usar programas o aplicaciones móviles. Pero la realidad es que sin internet; la mayoría no funcionaría.

Esto presenta desventajas siendo la principal que se depende constantemente de una conexión en línea que puede parecer que está presente en todos sitios pero en lugares remotos de la ciudad, como son los pueblos de montaña, el internet no es el mejor compañero, por decirlo, de una manera.

Las ventajas son enormes aunque solo se verán las que implican mayor relevancia para el desarrollo del proyecto.

El tener ya un repositorio generado con el proyecto subido dentro de él aporta bastante autonomía ya que puede accederse a él y descargar los datos desde cualquier lugar. Pero, ¿y configurarlo?

Aquí viene una problemática que no resuelve un entorno de repositorios. El tener que configurarlo todo cada vez que se quiera trabajar desde un sitio distinto. Esto lo resuelven las máquinas virtuales en la nube. El generar un directorio de trabajo donde poder conectar el Visual Studio Code y olvidarse de preocupaciones.

Lo segundo que se ha considerado más importante es el ahorro de memoria tanto de RAM como de disco y de procesos que se origina al hacer esto. No es lo mismo tener que trabajar con un portátil conectado todo el día a un enchufe que poder ir llevándolo contigo durante todo el día

por el escaso consumo de batería que tiene. Peor si es una torre, solo puedes trabajar desde un único sitio, con la problemática también de que siempre se te puede ir la luz.

Estas ventajas expuestas son las que favorecen al desarrollo del proyecto que es realizado por una única persona, pero ¿y si este se hiciera en un equipo? El poder tener varias personas trabajando en el mismo proyecto, tocando los distintos componentes que se están desarrollando dentro de la aplicación es fantástico.

### **Crear una máquina virtual en Google Cloud**

Los pasos para la creación de una máquina virtual son:

1. El registro es fácil de realizar ya que la herramienta es propiedad de Google. Se omitirá este paso.
2. Google Cloud se organiza mediante proyectos. Esto brinda facilidades a la hora de calcular el gasto que se tendrá por el uso de la computación en la nube. Aparte de poder facilitar la búsqueda por cada proyecto que se tenga activo. Se procederá a crear un nuevo proyecto clicando arriba a la izquierda a la derecha del título de la página y posteriormente clicando en nuevo proyecto.
3. Luego se desplegará la barra lateral de la izquierda y se clicará en el apartado de Compute Engine.
4. Se pulsará el botón de crear instancia. A la hora de llenar los parámetros se ha comprobado que no es necesario darle demasiada importancia a la zona o región donde se ubique tu máquina. Esta se conectará con una velocidad bastante decente. En el apartado de configuración de máquina con 2GB de RAM y 1VCPU se tendrán suficientes recursos. El tamaño del disco elegido será de 20GB. Dentro de la sección Firewall se habilitará el tráfico HTTP y HTTPS. Y posteriormente se pulsará en crear.
5. Después de haber creado la máquina se le asignará una IP estática pública que se usará para conectarse a ella mediante SSH.
6. En el momento de haber creado la máquina esta ha dado una clave privada SSH para poder conectarse a ella. Es necesario guardarla.

### **Configurar claves SSH**

Teniendo ya la clave privada SSH de la máquina virtual se utilizará para realizar la conexión con ella. Pero para ello es necesario configurarla. Hay que acceder al fichero usuario/.ssh/config y añadir tres nuevas líneas:

```
Host "Dirección IP"
HostName "Nombre del host"
User "Nombre de usuario"
```

Lo último que nos queda es realizar la conexión mediante SSH a la máquina con Visual Studio Code.

### **Realizar conexión SSH mediante Visual Studio Code**

Dentro del apartado de extensiones de Visual Studio Code se localizarán los siguientes plugins para instalar:

- Remote - SSH

- Remote - SSH: Editing Configuration Files

Luego de la instalación de estas dos extensiones se procederá a realizar la conexión. Abajo a la izquierda aparecerá la opción de poder conectarse mediante SSH. Hay que clicar sobre ella y luego se pulsará en “Connect to Host” añadiendo la dirección IP estática pública que dio Google Cloud.

Ya con esto estaría la masa del entorno hecha, solo hace falta darle un poco de forma y calor para poder finalizarla.

### **3.3.3. Instalación de las extensiones de Visual Studio Code dentro de la MV**

Esta es la lista de componentes con la que se trabajará en todas las fases del proyecto. Hay que acceder a las extensiones dentro de Visual Studio Code y proceder a instalarlas dentro de la máquina virtual.

- Docker
- LaTeX
- LaTeX Workshop

Los complementos de Latex ayudarán más tarde con la redacción del Trabajo de Fin de Grado. En Visual Studio Code también se instalará:

- Bootstrap 4 snippets
- HTML snippets
- CSS snippets

### **3.3.4. Configuración de la máquina virtual**

Desde Visual Studio Code se accederá a la terminal de la máquina virtual. Se pinchará en “Ver” en la sección superior y luego en Terminal. Desde ahí se podrá controlar la máquina.

#### **Actualización del sistema**

Se realizará una actualización del sistema para evitar posibles fallos en un futuro:

```
sudo apt-get update  
sudo apt-get upgrade
```

#### **Instalación de Node JS**

```
sudo apt install nodejs
```

#### **Instalación de Angular 12**

```
sudo npm install npm@latest -g  
sudo npm install -g @angular/cli
```

### Instalación de Docker y Docker Compose

```
\\"Instalacion de Docker

sudo apt install apt-transport-https ca-certificates
curl software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg
sudo apt-key add -
sudo add-apt-repository
"deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"
sudo apt install docker-ce

\\\"Instalacion de Docker Compose

sudo curl -L "https://github.com/docker/compose/releases/download/1.26.0
/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

### Configuración de credenciales de Git e instalación del repositorio

```
//Configuración de credenciales

git config --global user.name "tu nombre de usuario"
git config --global user.email "tu correo electrónico"
git config --global user.password "tu contraseña"

//El repositorio se ubicará dentro de la carpeta raíz del usuario

git clone https://github.com/alejomaf/Inventarium.git
```

Añadir objeto	
	<input type="button" value="Inventario"/> <input type="button" value="Fungible"/>
Nombre *	
<input type="text"/>	
Imagen *	
<input type="button" value="Cámara"/>	<input type="button" value="Cámara"/>
Ubicación *	
<input type="button" value="Ubicación"/>	<input type="button" value="Ubicación"/>
Marca	
<input type="text"/>	
Nombre *	
<input type="text"/>	
Imagen *	
<input type="button" value="Cámara"/>	<input type="button" value="Cámara"/>
Marca	
<input type="text"/>	
Modelo	
<input type="text"/>	
Mejoras en el equipo	
<input type="text"/>	
Cantidad	
<span style="font-size: 2em;">2</span>	<input type="button" value="+"/> <input type="button" value="-"/>
Código del inventario 1 *	
<input type="text"/>	
Código del inventario 2 *	
<input type="text"/>	
Cantidad	
<span style="font-size: 2em;">1</span>	<input type="button" value="+"/> <input type="button" value="-"/>
Configuración	
<input type="button" value="Aplicar"/>	<input type="button" value="Aplicar"/>
Añadir	
Añadir	

Figura 3.6: Diseño de la característica para poder añadir fungibles e inventarios

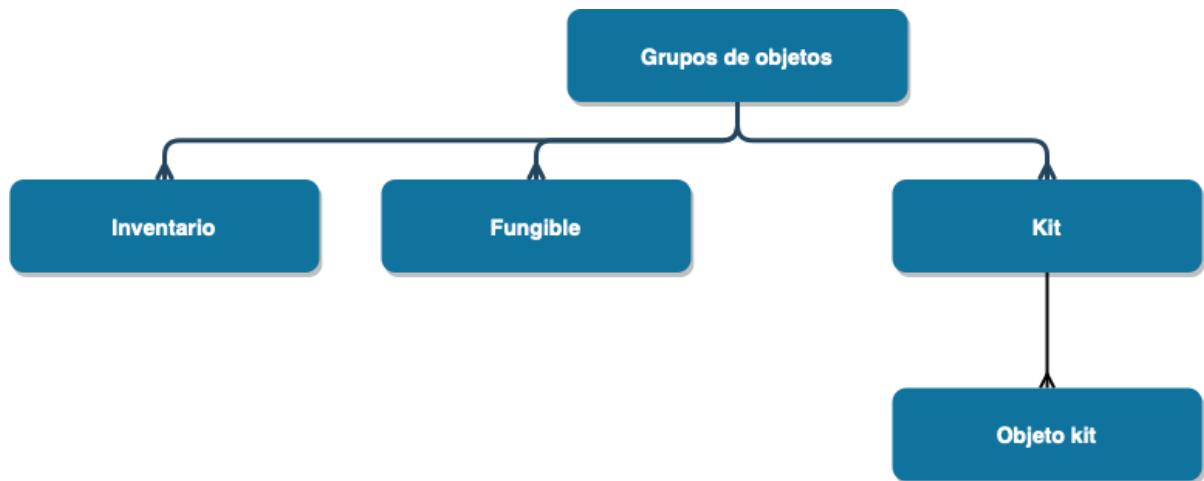


Figura 3.7: Jerarquía de elementos que derivan de grupo de objetos



Figura 3.8: Grupo de objetos a la izquierda y objetos que contiene a la derecha

Usuarios

Juan Francisco Sanjuan Estrada
José Andrés Moreno Ruiz

20 usuarios máximos por página

1    2    Siguiente

Usuario

Nombre: Juan Francisco Sanjuan Estrada
Correo electrónico: jsanjuan@ual.es
Departamento: Informática
Teléfono: 950 214017

Objetos prestados ▾

Objeto
Objeto

Nombre completo del destinatario\*

Cantidad

1	▼
---	---

Se podrá seleccionar para que sea indefinida  
Fecha estimada de entrega \*

Solicitar      Cancelar

Figura 3.9: Grupo de usuarios a la izquierda y usuario único a la derecha

Figura 3.10: Solicitud del préstamo de un objeto

Crear configuración

IP:	192.168.0.0
MAC:	XX-XX-XX-XX-XX-XX
Boca:	Boca
Armario:	Oscuro
Usuario:	Alejo
Contraseña:	1234

Modificar      Cancelar      Crear

Figura 3.11: Visualización de los campos de configuración a la izquierda y creación de estos a la derecha

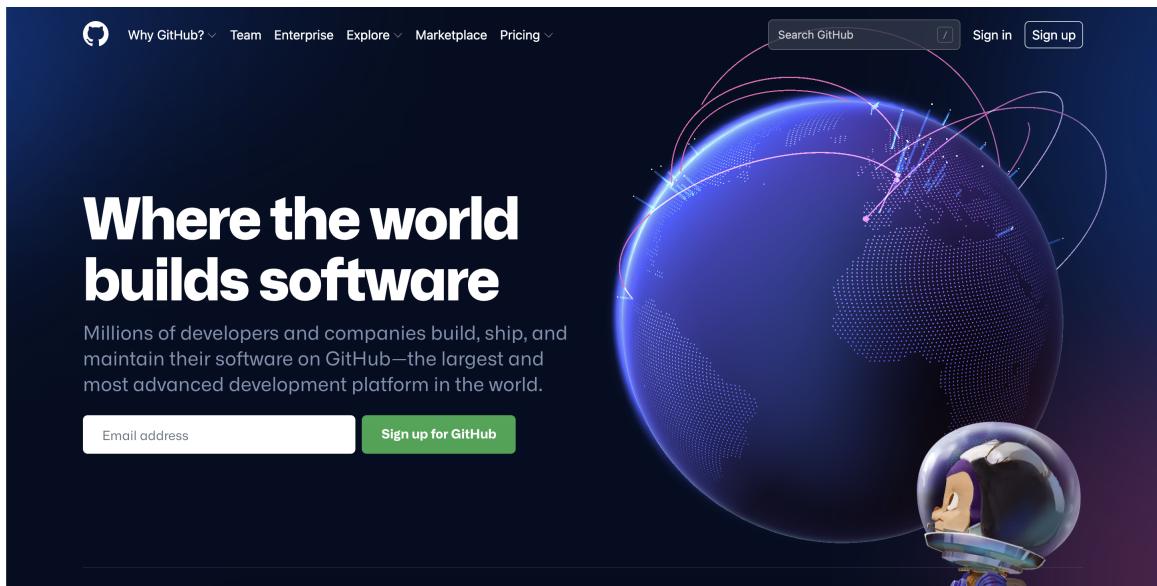


Figura 3.12: Página principal de GitHub

**Create a new repository**

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

---

**Repository template**  
Start your repository with a template repository's contents.

[No template ▾](#)

---

<b>Owner *</b>	<b>Repository name *</b>
 alejomaf ▾	/ <input type="text"/>

Great repository names are short and memorable. Need inspiration? How about [glowing-barnacle](#)?

**Description (optional)**

---

 **Public**  
Anyone on the internet can see this repository. You choose who can commit.

 **Private**  
You choose who can see and commit to this repository.

---

**Initialize this repository with:**  
Skip this step if you're importing an existing repository.

**Add a README file**  
This is where you can write a long description for your project. [Learn more.](#)

**Add .gitignore**  
Choose which files not to track from a list of templates. [Learn more.](#)

**Choose a license**  
A license tells others what they can and can't do with your code. [Learn more.](#)

---

[Create repository](#)

Figura 3.13: Creando un nuevo repositorio dentro de GitHub

## 4 Construcción de la aplicación

Dentro de este capítulo explicaré los distintos procesos por los que ha pasado la construcción de la aplicación. Todos tienen un punto común y es Docker Compose.

### 4.1. Introducción a Docker y Docker Compose

¿Cómo funciona Docker Compose? O mejor dicho ¿para qué es necesario Docker?

#### 4.1.1. ¿Para qué sirve Docker?

Gracias a Docker en vez de máquinas virtuales pueden utilizarse contenedores. Los contenedores presentan diferentes ventajas:

- El entorno de pruebas donde se ejecutarán los distintos componentes de la aplicación son idénticos al de un servidor.
- Se obtiene una mayor modularidad. Gracias a esto se tiene un entorno ideal donde poder trabajar con microservicios.
- Puede ejecutarse la aplicación en un entorno donde en caso de fallo puede reiniciarse fácilmente.

Docker es un avance para la informática. Ayuda a ahorrar recursos en los sistemas y a dedicar el tiempo invertido en la configuración de entornos de servidor a otras actividades de desarrollo.

#### 4.1.2. Docker Compose: el SimCity de los entornos

¿Por qué SimCity? Al igual que en el famoso juego SimCity se levantan ciudades en cuestión de segundos, gracias a Docker Compose, se levantan entornos en el mismo tiempo.

Docker Compose ayuda a definir un entorno de contenedores con la posibilidad de poder conectarlos entre ellos. También ofrece la posibilidad de generar volúmenes de almacenamiento donde ir guardando la información generada dentro de ellos.

Docker Compose funciona sobre un archivo llamado docker-compose.yml dentro del cual se irán añadiendo los componentes que quieran configurarse. Todo esto se irá viendo a lo largo de las siguientes secciones.

### 4.2. Generación y puesta en marcha de la base de datos

El diagrama final de la base de datos quedaría como puede observarse en la figura 4.1. Se generará el script “.sql” para poder crear la base de datos.

Esta configuración se basará únicamente en la sección de código que se añadirá dentro del docker-compose.yml.

Se añade la versión de docker-compose a utilizar:

```
version: "3.9"
```

Se creará el apartado “services”:

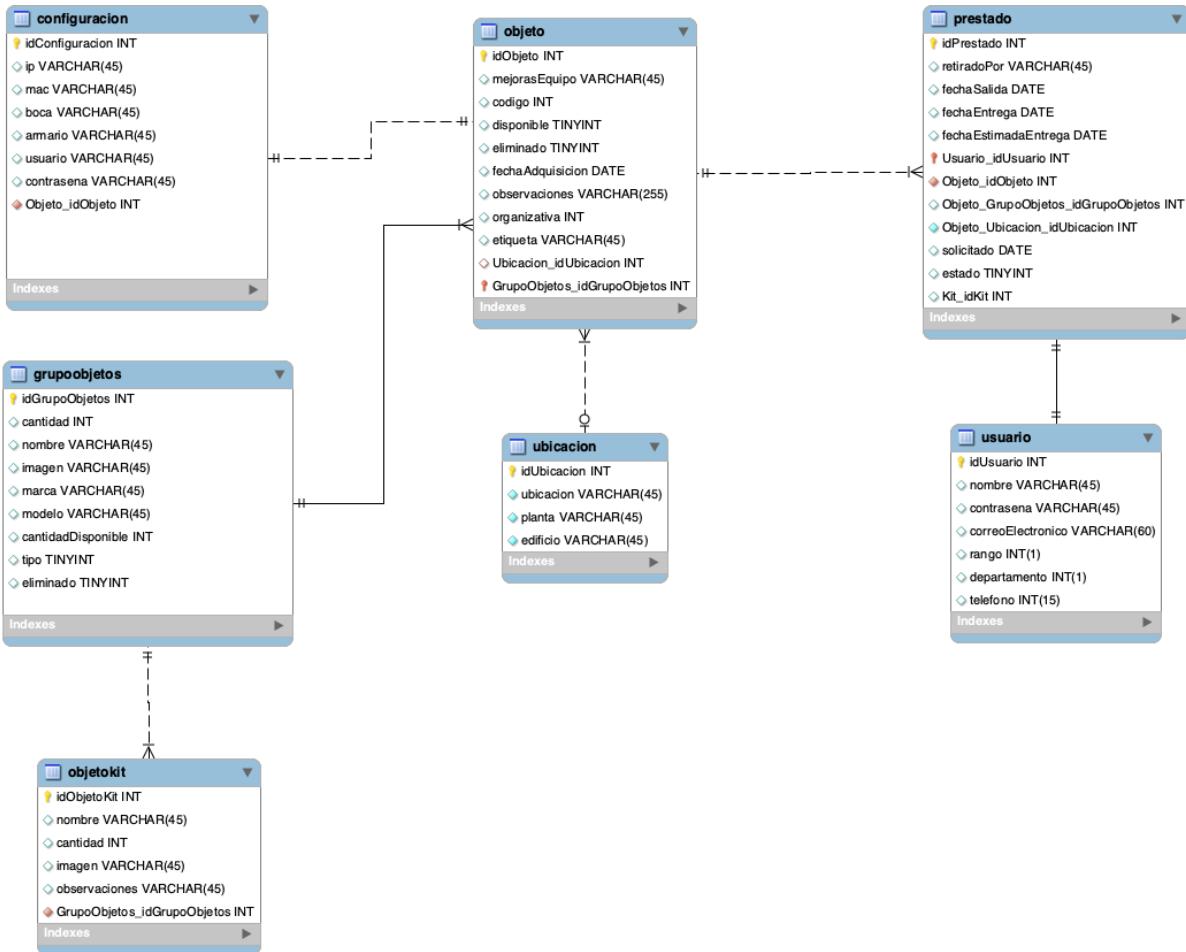


Figura 4.1: Diseño final de la base de datos

services:

Se añade el primer contenedor, en este caso, el de la base de datos:

```
db:
  container_name: inventarium_sql
  image: mariadb:10.7.1-focal
  ports:
    - "3306:3306"
  volumes:
    - ./dump:/docker-entrypoint-initdb.d
    - persistent:/var/lib/mysql
  networks:
    - default
environment:
  MYSQL_DATABASE: ualinventario
  MYSQL_USER: ualinventario
  MYSQL_PASSWORD: contraseñasecreta
  MYSQL_ROOT_PASSWORD: contraseñasecreta
```

- **db**: Este es el nombre que se le ha puesto como referencia para docker-compose.
- **container\_name**: Es el nombre que tendrá el contenedor que se vaya a desplegar.

- **image:** La imagen utilizada, es decir, la “máquina virtual” que se irá a levantar. En este caso es una máquina virtual de MariaDB.
- **ports:** Los puertos que utilizará la máquina, a la izquierda son los que le llegan a la máquina y a la derecha los que salen.
- **volumes:** Esta sección es muy importante ya que aquí se podrán importar archivos en el momento del despliegue del entorno. En este caso se está importando la carpeta “dump” donde está el archivo de generación de la base de datos. También se le enlazará el volumen “persistent” que se creará más adelante. Esto es para que no se pierdan datos cuando se detenga contenedor ya que los datos de la base de datos, se almacenarán en un sitio externo.
- **networks:** Aquí pueden añadirse las redes que manejará el contenedor. En este caso es la default.
- **environment:** Dentro de environments se podrán definir variables que se necesiten en el momento del despliegue del contenedor. En este caso se definirá el nombre de la base de datos, el usuario junto a su contraseña y la contraseña del usuario root.

```
phpmyadmin:  
  container_name: inventarium_php_adm  
  image: phpmyadmin:5.1  
  links:  
    - db:db  
  ports:  
    - 8000:80  
  environment:  
    MYSQL_USER: user  
    MYSQL_PASSWORD: user  
    MYSQL_ROOT_PASSWORD: user  
  networks:  
    - default
```

Este despliegue es para phpmyadmin, el gestor de la base de datos. Como añadido al punto anterior está el apartado **links** que ayuda a crear una vinculación de un contenedor con otro. En este caso se le está pasando la información de la base de datos para que trabaje sobre ella.

```
volumes:  
  persistent: {}
```

Dentro de la sección **volumes** se pueden generar volúmenes que se van a utilizar dentro de los contenedores. En este caso se ha creado el volumen “persistent” para poder guardar la información que se almacene dentro de nuestra base de datos.

Para levantar la base de datos se ejecutará dentro del directorio donde se ha creado el docker-compose.yml el siguiente comando:

```
sudo docker compose up
```

Ya con esto estaría la base de datos en funcionamiento.

### ¿Cómo comprobamos los resultados?

Visual Studio Code realiza una **tunelización de los puertos** a partir de la conexión SSH. Es decir, habilita redireccionamientos de puertos para poder ir viendo los resultados en la máquina. Esto resulta de gran ayuda ya que no hay que realizar una configuración en la máquina de apertura de puertos para poder ir trabajando sobre cada una de las aplicaciones existentes ya que el funcionamiento del sistema será interno. Con poder disponer de los puertos HTTP Y HTTPS abiertos no se necesitará más.

## 4.3. Diseño de la arquitectura de la aplicación

Antes de empezar con el desarrollo de la API y del sitio web se expondrá la arquitectura que se ha utilizado para el desarrollo de estos elementos.

### 4.3.1. ¿Qué es una arquitectura de software?

Según el IEEE la arquitectura es la organización fundamental de un sistema compuesto por sus componentes, las relaciones que tienen unos con otros y con el entorno y los principios que guían su diseño y evolución.

Entendiendo sistema como un conjunto de componentes que se organizan para cumplir una determinada función o conjunto de funciones.

Un sistema existe para cumplir una o más misiones en su entorno.

El entorno o contexto determina la configuración y circunstancias en el desarrollo, las operaciones, la política y demás aspectos que puedan influenciar a un sistema.

La arquitectura de software es de vital importancia en el desarrollo de un sistema ya que determina su estructura conllevando un aspecto directo sobre la capacidad de este para satisfacer los requisitos y evolución de un proyecto.

### ¿Por qué es importante definir una arquitectura software?

Definir una arquitectura de software presenta varias ventajas. Entre ellas se encuentran las siguientes:

- **Independencia de los frameworks:** Una arquitectura puede ser definida en múltiples frameworks y lenguajes pues no depende de ellos.
- **Independencia de las reglas de negocio:** Las reglas de negocio no se verán alteradas por el cambio de arquitectura software.

### 4.3.2. Clean Architecture o Arquitectura Limpia

Es un conjunto de principios cuya finalidad principal es ocultar los detalles de implementación a la lógica de dominio de la aplicación.

Gracias a esto puede mantenerse la lógica de la aplicación aislada de forma que sea más mantenible a lo largo del tiempo.

Para explicar qué es la arquitectura limpia se dispone del gráfico 4.2.

Puede observarse que la estructura del diagrama está formada por capas. Las capas representan distintos conjuntos que componen el sistema.

Dentro de la figura puede verse una flecha en la que aparece escrito “Dependency Rule”, en español, regla de dependencia. Significa el sentido que tomarán los distintos componentes de la aplicación. En este caso de afuera hacia dentro. Los componentes más externos dependen de los más internos y las entidades o el dominio de la aplicación depende de ella misma.

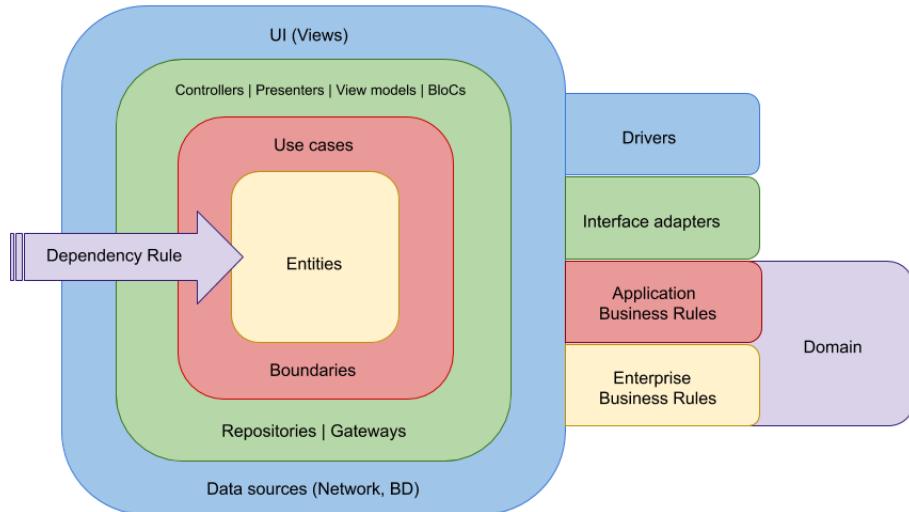


Figura 4.2: Clean Architecture

En el momento de la separación de funcionalidades hay que tener en cuenta que el sistema se divide en tres partes. La web, la API y la base de datos.

### Entidades

Las entidades representan las piezas básicas de la aplicación. Estas serían las tablas del diagrama de la base de datos. Cada una de las tablas conforma un tipo de datos. Pueden tener dependencias entre ellos pero nunca van a tener dependencias externas. Por eso es lo primero que se define ya que si hay que cambiarla habría que cambiar los elementos que dependieran de ella.

### Casos de uso

Los casos de uso están definidos en la API, que es la que gestiona todos los procesos internos que pueden realizarse con los datos, desde la solicitud de préstamos hasta dar de alta a un usuario.

### Controladores, presentadores y vistas de los modelos

Este conjunto está definido por las herramientas que ayudan a solicitar y adaptar la información más interna de la aplicación. También en esta parte tenemos las interfaces las cuales ayudan a adaptar los datos que llegan de los casos de usos para poder interpretarlos.

### Vistas y fuentes externas

Aquí están los frameworks, los adaptadores de red, el servidor de la base de datos y las vistas que es la forma en que se estructura y diseña la información para que se muestre al usuario.

## 4.4. Creación y puesta en marcha de la Interfaz de Programación de Aplicaciones (API)

Las APIs son una de esas pequeñas implementaciones que han cambiado enormemente los desarrollos en la informática, hoy en día si tu aplicación no dispone de una API implica varios

aspectos:

- La expansión de tu aplicación a otras tecnologías conllevará un esfuerzo mayor que si tuviera una API.
- Tu servidor requiere más servicios y por tanto recursos para funcionar.
- La lógica de negocio de la aplicación ha sido integrada en conjunto con la web presentando una alto nivel de acoplamiento y dificultando la modificación de la misma.

Una API ofrece una capa de abstracción para la aplicación. El famoso modelo de caja negra del que se ha hablado con anterioridad. Esto permite a distintas tecnologías orientadas a desarrollo web, móvil y de programas de ordenador a tener un punto en común entre todas. A que cada una no necesite personalizar sus comunicaciones con la base de datos y pueda usar un intermediario. Así es el funcionamiento de las APIs.

Para implementar la API se ha utilizado Node junto a Express. Gracias a Node puede levantarse un servidor web que no necesite de demasiados recursos para su funcionamiento. Esto se haría con las siguientes líneas de código:

```
app.listen(port, () => {
  console.log('Inventarium API listening at http://api:3000')
});
```

Otra de las partes que aporta Node es su gestor de paquetes, Node Package Manager (NPM), gracias a esto la implementación de funcionalidades más complejas se pueden realizar en un tramo de tiempo muy pequeño.

Para inicializar un proyecto de NPM se utiliza el siguiente comando dentro de un directorio:

```
npm init
```

Express es unos de los frameworks principales que presenta Node. Es un marco de desarrollo minimalista para Node que nos permite estructurar aplicaciones, crear enrutamientos y muchos más aspectos relacionados con lo que sería un entorno web.

Para instalar Express en un proyecto NPM y guardarlo en la lista de dependencias hay que escribir el siguiente comando:

```
npm install express --save
```

#### 4.4.1. Creación del sistema de directorios

Un apartado a tratar en este capítulo es la gestión de directorios y los distintos ficheros que hay en su ubicación interactúan entre ellos.

Ya habiendo creado el proyecto Node y habiéndole añadido Express esta sería la estructura de directorios que se usará:

- **images:** Dentro de este directorio se almacenan las imágenes que se van subiendo al sitio web. Estas imágenes van ligadas a un “grupo de objetos”.
- **node\_modules:** La carpeta de node\_modules es donde se gestionan todos los paquetes del NPM.
- **routes:** Dentro de la carpeta routers se creará todo el enrutamiento de la aplicación. En ella se ubica un archivo por cada tabla que se presenta en la base de datos.
- **services:** El objetivo de la carpeta services es el de poder realizar todos los tipos de consultas que requiera la aplicación.

- *.dockerignore*: La funcionalidad de este fichero es el de poder ignorar un directorio o fichero en el momento de la creación de una imagen de un contenedor docker. En este caso el único directorio que se está ignorando es node\_modules. ¿Por qué? Porque en el momento de la generación de una imagen docker suele dar problemas el importar directamente las dependencias que se iban a utilizar a mano. Es mejor que desde el propio sistema de gestor de paquetes, después de leer el packagelock.json, sea quien instale las dependencias nuevamente.
- *.env*: Aquí se ubicarán las variables con las que se trabajará en el entorno. Es una forma fácil y sencilla de poder generalizar secciones del código. El contenido del archivo es el siguiente:

```
DB_HOST='localhost',
DB_USER='user',
DB_PASSWORD='secretpassword',
DB_NAME='ualinventarium',
```

- *config.js*: Dentro del fichero de configuración se añadirán los parámetros que va a coger la variable “db” para conectarse a la base de datos.

```
const config = {
  db: {
    host: env.DB_HOST,
    user: env.DB_USER,
    password: env.DB_PASSWORD,
    database: env.DB_NAME
  },
  listPerPage: env.LIST_PER_PAGE || 10,
};
```

#### Utilización del archivo .env

Podemos ver cómo nuestro archivo que genera la configuración para la base de datos solicita la información dentro de nuestro fichero *env*. El último atributo que intenta solicitar es “LIST\_PER\_PAGE” en el caso que no lo pueda obtener, que será lo que va a ocurrir, pondrá como valor predeterminado 10.

- *Dockerfile*: Este archivo funciona para la generación de una imagen de un contenedor docker.
- *helper.js*: Un pequeño fichero que brinda un par de funciones a la hora de manejar consultas con la base de datos.
- *index.js*: El archivo principal de la API, dentro de él se inicializará todo.
- *packagelock.json*: El fichero principal de NPM que ayuda a gestionar todas las dependencias con los paquetes que se tengan instalados en el proyecto.
- *package.json*: El paquete json es el corazón de cualquier proyecto de Node. Registra metadatos importantes necesarios para la publicación de la aplicación, y también define atributos funcionales de un proyecto que NPM usa para instalar dependencias, ejecutar scripts e identificar el punto de entrada al paquete.

#### 4.4.2. Funcionamiento de index.js

Dentro de index.js se gestionarán todas las llamadas a los diferentes componentes de la API. Desde añadir el ruteo hasta gestionar la utilización de diferentes librerías que se hayan instalado dentro del proyecto.

Una variable importante a recalcar es la de Express. La cual se define así:

```
const app = express();
```

Gracias a la variable “app” se puede enlazar toda la configuración principal del ruteo que va a tener la API.

Por ejemplo, de la ruta */api/grupoobjetos* se quiere que el usuario pueda hacer interacciones con la tabla de grupo de objetos. Para gestionarlo se enlaza ese punto al archivo de ruteo que se explicará en el siguiente apartado:

```
//La variable grupoobjetos es la referenciación del archivo de ruteo  
app.use('/api/grupoobjetos', grupoobjetos);
```

Se realizaron otras gestiones en el componente de Express entre las que se encuentran:

- Aumentar el tamaño de las peticiones que lleguen para la carga de imágenes.
- Configurar un endpoint en la raíz del servidor que tenga como finalidad comprobar que la API funciona correctamente.

Por último se configuró el puerto de entrada del servidor:

```
app.listen(port, () => {  
    console.log('Inventarium API listening at http://api:3000')  
});
```

Esto permite que la aplicación se encuentre “escuchando” cada petición de entrada que le envíe la página web.

#### 4.4.3. Enrutamiento de la aplicación

El objetivo del directorio “routes” es la correcta gestión de todas las peticiones para el funcionamiento de la página web.

Cuando se crea una herramienta para la gestión de los recursos de una base de datos lo más normal es que se necesite implementar el repertorio de herramientas llamadas CRUD, create, read, update and, delete, es decir: crear, leer, actualizar y eliminar.

Al inicio del archivo de enrutamiento se llama a la funcionalidad de Router que incopora Express:

```
const router = express.Router();
```

#### Métodos de petición HTTP

Los métodos de petición HTTP es la definición de un conjunto de elementos que tiene como objetivo realizar diferentes acciones para la gestión de un recurso determinado.

Estos métodos serán los que ayuden en la implementación del repertorio de herramientas CRUD.

#### GET (leer)

El método GET es el encargado de solicitar un recurso en específico. Estas peticiones, por norma general, deben tener como único objetivo la recopilación de datos.

Por ejemplo, en el archivo de grupoobjetos se implementa el siguiente método:

```
/* GET group_of_objects. */
router.get('/', async function (req, res, next) {
  try {
    res.json(await group_of_objects.getMultiple(req, req.query.page));
  } catch (err) {
    console.error('Error while getting group_of_objects ', err.message);
    next(err);
  }
});
```

Dentro de la llamada al método “get” de “router” se realiza un “try catch” en el que se llama al método “getMultiple” de la clase que se ha importado con anterioridad de group\_of\_objects con el siguiente método:

```
const group_of_objects = require('../services/grupo_objetos');
```

El método “getMultiple” devuelve una lista con los diferentes grupos de objetos que hay en el servidor. Se ahondará más en ese método en la siguiente sección.

Dentro del “catch” se hace que la API devuelva un mensaje de error como respuesta en caso de fallo.

### Personalización de peticiones

El endpoint que gestiona “grupoobjetos” es `/api/grupoobjetos` por lo que al estar definiendo dentro del `router.get` el parámetro ‘/’ se pretende que la API ofrezca esa petición desde la raíz del endpoint. Por ejemplo, si se quisiera personalizar más la petición podría modificarse este apartado y en vez de usar ‘/’ se usaría `'/fungibles'`. Para poder acceder a esa consulta GET habría que llamar a `/api/grupoobjetos/fungibles`.

No hace falta que se le pase ningún parámetro a la petición GET pero se ha dejado en caso de realizar alguna implementación extra.

### POST (escribir)

El método POST se utilizará para el envío de una entidad a un determinado recurso. La API hará uso de este método para la adición de elementos a la base de datos.

### PUT (actualizar)

Este método se encarga del reemplazo de una determinada entidad. Se utilizará para la actualización de las filas de la base de datos.

### DELETE (eliminar)

Como su propio nombre indica en inglés el método DELETE se utilizará para borrar un recurso en específico.

Se ha añadido solamente un ejemplo dentro del método GET porque el resto de llamadas se realiza de forma muy parecida. En el POST y PUT se pasarán objetos como parámetros y en el PUT se especificaría una ID para la actualización del objeto. En DELETE solamente se especificaría la ID al igual que dentro del método PUT. Por ejemplo, si se quiere realizar una llamada eliminando el grupo de objetos con id 4 se llamaría al método DELETE de la API con la siguiente dirección `/api/grupoobjetos/4`.

#### 4.4.4. Consultas con la base de datos

Dentro de la carpeta “services” están las consultas con la base de datos. En ella se definen todas las posibles interacciones que querría realizar el usuario con la base de datos.

Las funciones coincidentes en todos los ficheros que se encuentran ubicados en este directorio son las siguientes:

- **getMultiple**: dentro de ella se llama al método SELECT de MariaDB.
- **create**: se llama al método INSERT INTO.
- **update**: se llama al método UPDATE.
- **remove**: se llama al método DELETE.

Para explicar la estructura de estas funciones se cogerá como ejemplo “getMultiple”:

```
const offset = helper.getOffset(page, config.listPerPage);
const rows = await db.query(
  'SELECT idGrupoObjetos, cantidad, nombre, imagen, marca,
    modelo, cantidadDisponible, tipo, eliminado
    FROM grupoobjetos WHERE eliminado = 0 LIMIT ?,?',
  [offset, config.listPerPage]
);
const data = helper.emptyOrRows(rows);
const meta = { page };

return {
  data,
  meta
}
}
```

Como puede comprobarse se llama al método offset que ayudará a realizar la paginación del sitio web.

Después de la consulta se llama al método de la clase definida anteriormente “helper” llamado “emptyOrRows”. Este método ayuda a evitar cualquier problemática que pueda causar la API si la base de datos devuelve el array vacío.

Por último en el “return” de la función se devuelve un JSON que tiene como primer elemento la respuesta de la petición y como segundo la página.

#### 4.4.5. Gestión de archivos con Express

Para poder realizar la subida de imágenes a la base de datos se hará uso de la librería “formidable”.

Esta se encontrará importada en el fichero “index.js” de la siguiente forma:

```
const formidable = require('express-formidable');
```

Se procederá luego a incorporarla a Express de la siguiente forma:

```
app.use(formidable());
```

Gracias a esto ya puede analizarse el campo de archivos de las peticiones que lleguen.

## Descomposición de la petición

Dentro de la petición que llega a la API se coge el campo de archivos que va vinculada a ella gracias a la utilización de *formidable*.

```
files = req.files
```

## Análisis del archivo

Luego se procede a analizar el archivo para que cumpla los siguientes requisitos:

- Que sea únicamente un solo archivo.
- El método “isValid” comprueba que el archivo tenga la extensión *jpg*, *jpeg*, *png*. Es decir, que sea una imagen. Se ha seleccionado únicamente esta extensión de archivos porque luego si se cambia la extensión a *jpg* se pueden visualizar sin problemas.

```
// Se comprueba que el archivo sea uno o más de uno
if (!files.length) {
    //En el caso de que sea solo un archivo
    const file = files.image;
    // Se comprueba que el archivo sea válido
    const isValid = isValid(file);
    // Se crea un nombre en base al momento actual en que se
    // está subiendo el archivo
    const fileName = time + ".jpg";

    if (!isValid) {
        // Si el archivo no es válido se lanza un error
        return res.status(400).json({
            status: "Fail",
            message: "The file type is not a valid type",
        });
    }
}
```

## Subida de la imagen

Gracias a esto puede realizarse la subida de la imagen sin problemas. El directorio de subida se será “images”.

```
const uploadFolder = path.join(__dirname, "..", "images",
    "group_of_objects");
try {
    // Se cambia el nombre del archivo en el directorio
    fs.renameSync(file.path, path.join(uploadFolder, fileName));
} catch (error) {
    console.log(error);
}
} else return;
```

### Envío de la consulta a la base de datos

Se envía una solicitud a la base de datos para finalizar con la creación del grupo de objeto. En caso que la subida diera error este proceso se pararía. Como puede comprobarse, al llamar al método de creación de grupo de objetos se le pasa como parámetro el campo *fields*, donde va el cuerpo de la petición.

```
//Consulta post en la base de datos
res.json(await group_of_objects.create(req.fields, time));

} catch (err) {
  console.error('Error while creating group of objects', err.message);
  next(err);
}
});
```

Ya con esto se tendría la imagen subida con el nombre del archivo actualizado dentro de la base de datos.

## 4.5. Creación de UAL Inventarium

Este es la sección donde se juntan todos los conocimientos y herramientas que se han ido exponiendo a lo largo de este documentos y se cohesionan para crear UAL Inventarium.

UAL Inventarium es una herramienta web diseñada para la gestión del inventario y préstamos del Departamento de Informática de la Universidad de Almería.

La página web está hecha en Angular 12. Angular es una plataforma de desarrollo compuesta por un framework y librerías. Angular brinda todas las herramientas necesarias para la creación de un sitio web.

### 4.5.1. Estructura de un proyecto Angular

Para poder desplegar un entorno donde trabajar en Angular primero hay que tenerlo instalado en Node Package Manager (NPM) con el siguiente comando:

```
npm i -g @angular/cli
```

Al hacer esto se desplegará el entorno de desarrollo para Angular. El directorio **node\_modules** es donde se almacena el Framework de Angular, el CLI y los distintos componentes que se vayan instalando con el NPM.

#### ¿Qué diferencias hay entre Angular CLI y Angular Framework?

Angular CLI es la Command Line Interface la cual permite poder crear proyectos Angular, añadir componentes, servicios o directivas desde una línea de comandos. Angular CLI se encarga de la gestión de las distintas posibilidades que puede ofrecer el framework de Angular.

Los ficheros que se despliegan sobre el directorio raíz al crear un proyecto Angular son:

- **.editorconfig**: Un archivo de configuración para editores de código.
- **README.MD**: Archivo de texto que procesa GitHub en sus repositorios. El contenido inicial del fichero en el momento de la creación del proyecto trata sobre documentación acerca del Framework.

- **angular.json:** Esta es la configuración predeterminada que aporta el CLI de Angular para poder construir la aplicación, generar el servicio y testear los diferentes componentes.
- **package.json:** Este fichero se encarga de manejar las dependencias de NPM, precisamente las que están habilitadas dentro del espacio de trabajo.
- **package-lock.json:** Aporta información del versionado de los distintos paquetes que están en node\_modules.
- **tsconfig.json:** Esta es la configuración básica de TypeScript para el proyecto.
- **proxy.conf.json:** Este es el fichero que va a ayudar a poder consumir la API. Reenvía las peticiones que llegan a la aplicación al puerto 3000 que es donde se encuentra ubicada.

En la misma carpeta raíz se tiene un directorio llamado *src*, su descomposición es la siguiente:

- **app:** Directorio que contiene todos los distintos componentes de los que está compuesta la aplicación.
- **assets:** Contiene imágenes y otros recursos para ser copiados en el momento que se construya la aplicación.
- **environments:** Gracias a este fichero puede configurarse una opción en particular de construcción de la aplicación.
- **favicon.ico:** El ícono que sale en la parte superior de la pestaña de la página web.
- **index.html:** La página principal que tiene cualquier web. El CLI se dedica a añadir automáticamente todo el JavaScript y el CSS cuando construye la aplicación. No es un fichero que se use.
- **main.ts:** Este fichero es el punto de entrada principal de la aplicación. Compila la aplicación y arranca el módulo raíz de la aplicación ( AppModule) para que se ejecute en el navegador.
- **polyfills.ts:** Provee de adaptaciones para distintos navegadores.
- **styles.css:** Es un archivo de configuración global de estilos para todos los componentes de la aplicación.
- **test.ts:** El punto de entrada principal para los test que se realicen en la aplicación.

El contenido del directorio *app* en el momento de la creación del proyecto es el siguiente:

- **app.component.ts:** Define la lógica para la aplicación raíz.
- **app.component.html:** Define el diseño HTML asociado con el elemento raíz.
- **app.component.css:** Define el elemento de diseño para el elemento raíz.
- **app.component.spec.ts:** Define el conjunto de pruebas asociado con el elemento raíz.
- **app.module.ts:** Define el módulo raíz, este fichero le comunica a Angular cómo se tiene que realizar el ensamblaje de la aplicación. Inicialmente está declarado dentro de él el propio módulo raíz, pero a medida que se vayan añadiendo elementos a la aplicación se irán incluyendo más módulos.

Dentro de la carpeta *src* hay tres directorios más:

- *components.*
- *interfaces.*
- *services.*

## Componente

Los componentes son las estructuras principales de construcción que hay en Angular. Para poder generarlos se utiliza el siguiente comando:

```
ng g c dirección_y_nombre_del_componente
```

Cuando se ejecute se generará un nuevo directorio con el nombre del componente. Dentro de él se habrán creado cuatro ficheros diferentes:

- **componente.html**: Aquí irá ubicado el diseño html que tendrá el componente.
- **componente.css**: Este documento de estilos se aplicará únicamente al componente.
- **componente.ts**: En el fichero TypeScript está la lógica del componente y cualquier tipo de procesado de datos que haya que realizar.
- **componente.spects.ts**: Este será el fichero de pruebas unitarias para el componente. Durante el desarrollo del proyecto no se ha utilizado ya que el testing del proyecto se ha hecho de otra forma.

## Servicio

Los servicios sirven para aislar más el modelo de vista controlador que presentan los componentes. Estos serán los encargados de comunicarse con nuestra API.

Para generar un servicio se utiliza el siguiente comando:

```
ng g s directorio_y_nombre_del_servicio
```

Se generarán dos ficheros TypeScript, uno para el conjunto de pruebas y otro para el servicio.

## Interfaz

La interfaz sirve para definir los elementos con los que se va a trabajar. Estos elementos corresponden a los que están en la base de datos.

Para poder generar una interfaz se escribe lo siguiente en la terminal dentro del proyecto:

```
ng g i directorio_y_nombre_de_la_interfaz
```

Un ejemplo de interfaz sería por ejemplo la de un objeto del tipo *Configuración*:

```
export interface Configuracion {
    idConfiguracion: number,
    ip: string,
    mac: string,
    boca: string,
    armario: string,
    usuario: string,
    contrasena: string,
    Objeto_idObjeto?: number
}
```

Estos elementos también ayudarán a procesar las respuestas que lleguen desde la API y para que puedan manipularse en los componentes de Angular sin problemas.

### 4.5.2. Disposición y elementos que hay en Inventarium

Se empezarán tratando los servicios y las interfaces y cómo se presenta su contenido en el caso de los primeros.

## Interfaces

Se tiene una interfaz por cada elemento que se encuentra en la base de datos.

## Services

Los servicios pueden ser los que hayan dado más tipos de problemas a lo largo del desarrollo. Son los encargados de generar las consultas HTTP de las que se han hablado en la sección anterior en el desarrollo de la API.

Se tienen tantos servicios como tablas en la base de datos quitando el de *user* que sirve para iniciar sesión dentro de la aplicación.

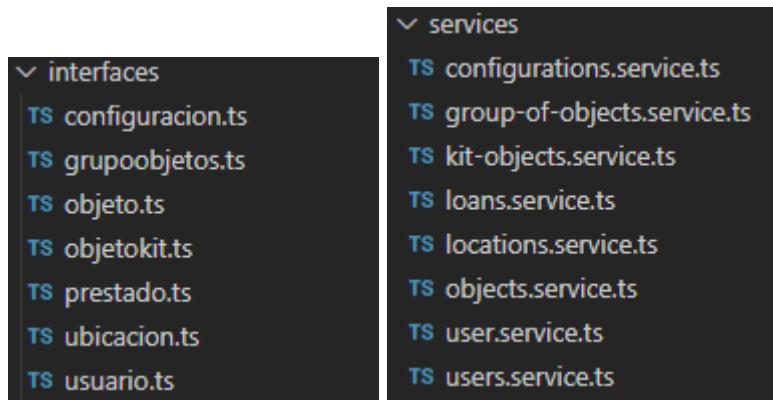


Figura 4.3: Interfaces y servicios de UAL Inventarium

Para mostrar la estructura de uno de los servicios de la aplicación se verá como ejemplo el fichero *group-of-objects.service.ts*. Al principio del documento están las importaciones de código y más adelante se declara lo siguiente:

```

@Injectable({
    providedIn: 'root'
})

```

El *@Injectable* junto al *provideIn: 'root'* sirve para que el servicio pueda utilizarse en todo el proyecto Angular.

Avanzando un poco más por el código puede observarse la declaración de la variable *\_url* a la cual se le indica la dirección a la que tiene que mandar la solicitud.

```
_url = "api/grupoobjetos"
```

Como puede verse esa es la ruta a la que se accederá a la API. La dirección es así porque se ha habilitado un proxy que redirige las peticiones. De tal proxy se hablará más adelante.

En el constructor de la aplicación se inicializará la siguiente variable:

```
constructor(private http: HttpClient) {}
```

*HttpClient* es el servicio encargado de mandar las solicitudes HTTP a la API.

Ahora ya pueden empezarse a crear las funciones que se utilizarán a lo largo del desarrollo:

```
getGroup0f0bjects() {
    return this.http.get(this._url);
}
```

En este caso la petición que se manda es del tipo *get* si se quiere crear un grupo de objetos sería con una petición *post*:

```
addGroupOfObject(objectGroup: FormData) {
    return this.http.post(this._url, objectGroup);
}
```

Donde como parámetro se pasa un campo del tipo formulario.

Para el resto de peticiones *put* y *delete* solamente se le añade un campo numérico que sea la id del objeto que se va a modificar */:id* y en el caso de *put* también se le pasa otro tipo de objeto formulario.

Con esto ya estaría definido el servicio listo para funcionar.

#### 4.5.3. Creación de componentes

Sabiendo cómo crear un componente se procederá a explicar un poco de forma más detallada su funcionamiento.

Como ejemplo se hablará del componente *groups-of-objects* que es donde se visualizan todos los grupos de objetos.

##### Preparar el apartado TypeScript

Se declaran las variables que se necesitarán, en este caso solo es:

```
group_of_objects: GrupoObjetos[] = [];
```

Dentro del código también se inicializa el array del grupo de objetos. Como se puede ver al declarar la variable se ha llamado a su interfaz para que su utilización sea mucho más cómoda. Para poder cargar los grupos de objetos que hay definidos en Inventarium hay que importar su servicio:

```
constructor(private group_of_objects_service: GroupOfObjectsService)
```

Y para que este cargue los respectivos grupos de objetos, dentro del constructor habrá que llamar al método para que lo haga:

```
this.group_of_objects_service.getGroupOfObjects().subscribe(
  (data : any) => {
    this.group_of_objects = res.data;
  },err => console.log('Error', err));
```

El método *getGroupOfObjects()* es el método que se ha definido anteriormente, el *.subscribe* es para poder realizar la consulta. Dentro de él puede decidirse cómo manipular los elementos que devuelva la petición.

Este elemento es un objeto del *json* pero Angular lo interpreta perfectamente. Objeto que puede tener uno de estos dos atributos, *data* que es un array de grupo de objetos, significando que la consulta no ha tenido errores y *err* que devuelve una cadena de caracteres en las que sale el tipo de error que ha ocurrido.

Con esto ya estaría el objeto cargado, pero ahora hay que mostrarlo.

##### Preparar el archivo HTML

El fichero HTML será el siguiente:

```
<div class="row d-flex-inline justify-content-center">
  <!--Contenido-->
  <div *ngFor="let go of group_of_objects" style="width: fit-content;">
    <app-group-of-object>
      
        <h5 nombre [routerLink]="/group-of-object",
        go.idGrupoObjetos" style="cursor:pointer"
        class="card-title text-dark text-center">
            {{go.nombre}}
        </h5>
        <span cantidad>{{go.cantidad}}</span>
        <span marca>{{go.marca}}</span>
        <span modelo>{{go.modelo}}</span>
        <span cantidadDisponible>{{go.cantidadDisponible}}</span>
        <span *ngIf="go.tipo==0" tipo>Inventario</span>
        <span *ngIf="go.tipo==1" tipo>Fungible</span>
        <span *ngIf="go.tipo==2" tipo>Kit</span>
    </app-group-of-object>
</div>
</div>
```

El componente es bastante sencillo. Primero se define un contenedor donde se irán insertando los componentes que haya devuelto la petición mandada anteriormente en el constructor.

Ahora se definirá otro contenedor donde se iterará sobre el array de grupos de objetos asignándolo a una variable auxiliar *go*. Esto se hará utilizando *\*ngFor*.

Luego de iterar se llamará al componente hijo que será el que se irá generando por cada grupo de objeto que cargue.

Dentro del componente hijo se definirán los componentes HTML que se quieran pasar. Para poder definir esto basta con añadirle un nombre que después se referenciará en el otro componente. Puede verse al final del código que se define un atributo dentro del componente HTML *span* llamado *\*ngIf*. Este atributo es un condicional, en caso de que sea *true* se cargará el componente. Si es *false* no lo hará.

Ahora se definirá componente hijo que es al que se está llamando:

```
<div class="card border rounded p-3 m-2"
style="width: 22rem;background-color:#FDF7FF;">
<div style="width: 100%; height: 230px;">
    <ng-content select="[imagen]"></ng-content>
</div>
<div style="width: 100%;" class="card-body list-group-item-dark border">
    <ng-content select="[nombre]"></ng-content>
</div>
<ul class="list-group list-group-flush">
    <li class="list-group-item bg-light">
        <b>Marca</b>
        <a>:<ng-content select="[marca]"></ng-content></a>
    </li>
    <li class="list-group-item bg-light">
        <b>Modelo</b>
        <a>:<ng-content select="[modelo]"></ng-content></a>
    </li>
    <li class="list-group-item bg-light">
        <b>Cantidad</b>
        <a>:<ng-content select="[cantidad]"></ng-content></a>
    </li>
    <li class="list-group-item bg-light">
        <b>Cantidad disponible</b>
```

```

<a>:<ng-content select="[cantidadDisponible]"></ng-content></a>
</li>
</ul>
<li class="list-group-item list-group-item-dark
font-weight-bold text-center">
    <ng-content select="[tipo]"></ng-content>
</li>
    <ng-content select="[botones]"></ng-content>
</div>

```

Este es el modelado que tiene el objeto. El componente HTML llamado *ng-content* será el encargado de tomar los objetos que le está pasando el componente padre. Estos los referencia con el atributo *select* y le indica el nombre del tipo de componente que quiere coger.

Para poder ir revisando cómo quedan los componentes se utilizará una funcionalidad que incorpora Angular. Dentro del archivo *package.json* NPM define una serie de scripts que pueden utilizarse. Uno de ellos es el siguiente:

```
"start": "ng serve"
```

Para poder ejecutar este script se ejecutará el siguiente comando:

```
npm run start
```

Que sería lo mismo que utilizar:

```
ng serve
```

Esto desplegará un servidor de la aplicación de Angular en el puerto 4200. Una de las ventajas que ofrece esto es una compilación continua del proyecto. Es decir, a medida que se vaya realizando la construcción de la aplicación, la generación de componentes y la creación de rutas la web se irá actualizando en cada guardado y será de gran utilidad poder ir viendo estos cambios al momento.

#### 4.5.4. Definir el archivo de rutas

Definir el archivo de rutas de la aplicación sirve para saber qué direcciones y qué comportamientos tendrá esta en su uso.

Las rutas de la aplicación son las indicadas en la figura 4.4.

Las rutas de la aplicación tienen una correlación directa con el manejo de los componentes. Estas rutas se definen dentro del fichero *app-routing.module.ts* que está dentro de *app*.

Este archivo importa todos los componentes que vayan a utilizarse en las rutas. El comienzo del archivo consiste en definir una constante que se llamará *routes* y en la que irá toda la lógica de la aplicación:

```

const routes: Routes = [
{
  path: '', component: MainComponent,
  children: [
    { path: 'dashboard', component: DashboardComponent },
    {
      path: 'add-object', component: AddObjectComponent,
      children: [
        { path: 'select-type', component: SelectObjectToCreateComponent },
        .
        .
        .
      ]
    }
  ]
}

```

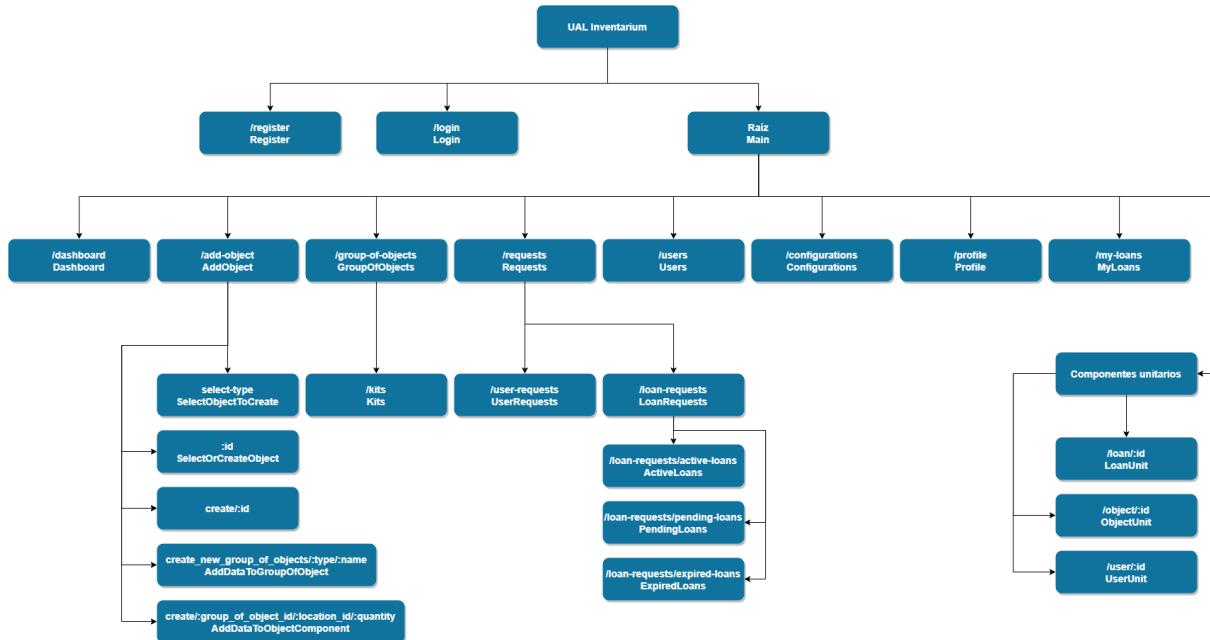


Figura 4.4: Rutas disponibles en la aplicación

El archivo es bastante más extenso pero con este pequeño trozo pueden verse y explicarse los diferentes componentes que contiene.

Para definir una ruta de forma básica la estructura será la siguiente *path: nombre\_de\_la\_ruta, component: nombre\_del\_componente* con esto al acceder a la ruta ya se mostraría dicho contenido.

Lo interesante que aporta Angular es la posibilidad de añadir hijos a estas rutas. Se hará llamando al atributo *children: [array\_de\_rutas]*.

Esto es muy importante ya que el uso de hijos en la ruta hace que los componentes de rutas superiores no se descarguen. Se explicará mejor con el archivo *MainComponent.html*:

```

<app-vertical-navbar></app-vertical-navbar>
<div class="page-content p-5" id="content">
    <!-- Toggle button -->
    <button id="sidebarCollapse" (click)="sidebarCollapse()" type="button"
    class="btn btn-light bg-white rounded-pill shadow-sm px-4 mb-4">
        <i class="fa fa-bars mr-2"></i>
        <small class="text-uppercase font-weight-bold">Toggle</small>
    </button>
    <router-outlet></router-outlet>
</div>

```

Dentro del HTML se puede ver un componente llamativo: *router-outlet*. Este está relacionado directamente con el routing que se tenga en la aplicación ya que es desde ahí donde se cargarán los hijos. La barra de navegación, por ejemplo, no desaparecería al acceder a */dashboard*.

Para realizar el importado de rutas dentro de Angular se llamará a *@NgModule* al final del documento para que las exporte al componente principal.

```

@ NgModule({
    imports: [RouterModule.forRoot(routes)],
    exports: [RouterModule]
})

```

#### 4.5.5. Definir el proxy en nuestra aplicación

La finalidad de un proxy inverso es reenviar las solicitudes que realiza la aplicación a la API, esta por políticas de CORS no puede reenviar una solicitud que entra al puerto 80 al puerto 3000 del mismo sitio. Tampoco se puede hacer que la API funcione sobre el puerto 80 ya que este puerto está siendo utilizado por la aplicación web. Por ello se tiene que hacer que este proceso se haga de forma interna en el servidor y para ello se utilizará un proxy inverso.

Un proxy inverso se encarga de reenviar las solicitudes que llegan a unas determinadas rutas del sitio web a otro puerto del entorno que no está alojado desde donde se envía. En este caso el proxy sería el encargado de que todas las solicitudes que entrasen por el puerto 80 de las direcciones definidas en la sección anterior sean redirigidas a la API que trabaja sobre el puerto 3000.

Este proxy inverso se definirá en el fichero *proxy.conf.json* de la siguiente forma:

```
"/api/configuracion": {
  "target": "http://localhost:3000",
  "changeOrigin": true
}
```

Y para poder utilizarlo durante el desarrollo de la aplicación se modificará el comando que se utiliza para inicializar el entorno de pruebas por el siguiente:

```
"start": "ng serve -o --proxy-config proxy.conf.json --host 0.0.0.0"
```

*/api/configuracion\** es para indicar que todas las solicitudes que llegan a la web con esa dirección se reenvien a *"target": "http://localhost:3000"* y que se cambie el origen de la solicitud *changeOrigin": true*.

#### ¿Qué son las políticas de CORS?

CORS (Cross-Origin Resource Sharing) es un mecanismo o política de seguridad que permite controlar las peticiones HTTP asíncronas que se pueden realizar desde un navegador a un servidor con un dominio diferente de la página cargada originalmente. Este tipo de peticiones se llaman peticiones de origen cruzado (cross-origin). Estas peticiones no están permitidas por ley porque suelen ser utilizadas para la piratería informática.

#### 4.5.6. Compilar el sitio Web

Para compilar el sitio web es necesario ejecutar el siguiente comando dentro de la raíz del proyecto:

```
ng build
```

Esto generará un directorio *dist* con otro directorio dentro con el nombre del proyecto. En este caso *Inventarium*. Este contendrá todos los ficheros para poder añadirlo directamente a un servidor web.

## 5 Diseño y funcionamiento final

Luego de todo el desarrollo de la aplicación se mostrará el diseño y funcionamiento final que tiene la aplicación.

### 5.1. Inicio de sesión y registro de usuario

The figure displays two user interface screens. On the left is the 'Iniciar sesión' (Login) screen, which has input fields for 'Correo electrónico' (Email) containing 'aaf842@inlumine.ual.es' and 'Contraseña' (Password) consisting of several dots. Below these fields is a blue link '¿Has olvidado la contraseña?'. At the bottom are two buttons: a purple one labeled 'Iniciar sesión' and a light blue one labeled 'Registrarse'. On the right is the 'Registrarse' (Register) screen, which contains six input fields: 'Nombre completo' (Full name) with placeholder 'Ingrese su nombre completo', 'Contraseña' (Password) with placeholder 'Ingrese su contraseña', 'Repita la contraseña' (Repeat password) with placeholder 'Repita su contraseña', 'Correo electrónico' (Email) with placeholder 'Ingrese su correo electrónico', 'Departamento' (Department) with placeholder 'Departamento de informática', and 'Número de teléfono' (Phone number) with placeholder 'Ingrese su número de teléfono'. At the bottom are three buttons: a purple one labeled 'Registrarse', a light blue one labeled 'Volver' (Back), and a dark blue one labeled 'Volver'.

Figura 5.1: A la izquierda el inicio de sesión y a la derecha el formulario de registro de usuario

En la figura 5.1 puede verse cómo ha quedado el resultado final del registro e inicio de sesión de Inventarium. Dentro del cuadro de inicio de sesión se añade un enlace posibilitando una recuperación de la contraseña. Dicha recuperación tendría que ser facilitada por un técnico. En el complemento del trabajo de fin de grado junto a la implementación del gestor de correos se completará la funcionalidad para que un usuario pueda recuperar sus credenciales.

Dentro del formulario de registro todos los campos deben ser rellenados de forma obligatoria. Además se han implementado comprobaciones para que estas variables cumplan con unas determinadas propiedades:

- El campo del nombre completo debe contener más de ocho caracteres
- La contraseña debe tener más de ocho caracteres
- La entrada de “Contraseña” y la de “Repita contraseña” deben ser iguales
- El correo electrónico tiene que tener la extensión @ual.es o @inlumine.ual.es
- El apartado para ingresar el número de teléfono debe contener 9 dígitos exactos.

### 5.2. Componentes principales

Estos componentes reciben su nombre debido a que son la estructura principal de la aplicación. Son los átomos que conforman los distintos componentes.

Un ejemplo sería la figura 5.2. En ella puede verse cómo se muestran los campos que puede contener un usuario, el correo electrónico, el departamento al que pertenece, su número de teléfono y qué tipo de usuario es. Este componente ha sido creado en base a uno principal con

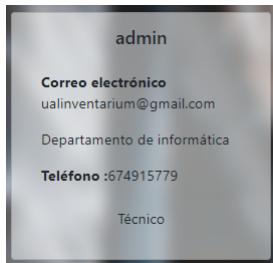


Figura 5.2: Representación de un usuario en la aplicación

el cual también podría crearse el de la figura 5.3. ¿Qué objetivo se pretenden conseguir con



Figura 5.3: Representación de un usuario y posibles acciones sobre él en la aplicación

esta diferenciación? La modularización del código, es decir, una separación entre visualización y acciones posibles sobre el objeto.

Esta vista donde se proyecta la visualización del usuario puede ponerse en varias secciones del código pero, dónde pueden modificarse su campos. No se pretende possibilitar la modificación de los campos en todos los sitios donde se llame al componente. Por ello se crea el concepto de vistas unitarias.

En una vista unitaria se van a poder realizar todas las acciones posibles sobre un objeto. Esto ayudará a poder hacer una separación de permisos, dependiendo de si es un usuario o un técnico y a implementar modificaciones únicamente en un sitio en caso de realizar algún cambio. Por ejemplo el usuario de la figura 5.3 es de la vista unitaria del usuario. Dentro de ella pueden realizarse diferentes acciones como convertirlo en técnico o eliminarlo.

Esta diferencia de vistas permite crear relaciones que ayudarán en la gestión de la herramienta. Las vistas unitarias junto a sus respectivas relaciones son las siguientes:

### 5.2.1. Usuario

Dentro de un usuario se dispondrá de un campo donde se podrá acceder para visualizar sus préstamos. También se podrán realizar diferentes acciones sobre él en caso de que ser un *técnico*:

- Dar de alta.
- Eliminar.
- Convertir en técnico.

### 5.2.2. Objeto

En el objeto se puede acceder a un apartado donde se verán sus préstamos. También se podrá ver el grupo de objetos al que pertenece y por último hay un campo donde se le puede añadir una configuración.

Para poder hacer una distinción de unos objetos de otros se les ha asociado a su nombre la id del elemento. Esto ayudará a los usuarios a distinguir posibles objetos que no tienen un código de inventario asociado a su nombre.

Puede verse cómo queda la vista de los objetos en la figura 5.4.



Figura 5.4: Objeto y acciones que podemos hacer sobre él

### 5.2.3. Grupo de objetos

Dentro de los grupos de objetos pueden visualizarse los datos principales del grupo como puede ser la *marca*, *modelo* y *nombre* y debajo de ella se verán los objetos que pertenecen a ese grupo.

Adicionalmente en el caso de que un objeto pertenezca a la categoría de *kit* se mostrará un botón que pondrá *¿Qué contiene el kit?* y que clicándolo mostrará los elementos del tipo *objetokit* que contiene. Todo esto puede verse en la figura 5.5.

### 5.2.4. Préstamos

Al haber distintos campos de préstamos cada uno dispone de diferentes casos de usos. Estos son los de la figura 5.6.

## 5.3. Componentes de procesos

Habiendo visto los componentes principales que conforman la aplicación otro aspecto a destacar es el del desarrollo de los procesos. Se distinguirán cuatro tipos de procesos:

- Proceso de creación de grupo de objetos y objetos.
- Procesos de creación.
- Procesos de modificación.
- Proceso de creación, selección y eliminación de ubicaciones.



Figura 5.5: Grupo de objeto y acciones que podemos hacer sobre él

### 5.3.1. Proceso de creación de grupo de objetos y objetos

Los procesos de creación de estos dos componentes tenían que llevar un aspecto de personalización debido a la interrelación que hay entre ellos.

Es decir, si un técnico quiere crear un objeto al pasar por este proceso va a saber si este ya ha sido creado con anterioridad para poder añadirlo al grupo de objetos.

#### **Escoger el tipo**

El primer paso es escoger el tipo de objeto que se desea crear, tal como puede verse en la figura 5.7

#### **Buscar o añadir el grupo de objeto**

Este proceso se encargará de que el usuario pueda ver si su grupo de objetos ya ha sido creado. En caso de que no lo sea puede pulsar en el botón de crear grupo de objetos para realizar una creación. Puede observarse en la figura 5.8.

En este ejemplo se supondrá que se ha iniciado un proceso de creación de grupo de objetos.

#### **Añadir imagen y datos**

Uno de los requisitos obtenido en las entrevistas iniciales era que cada grupo de objeto tenía que ir asociado obligatoriamente con una imagen. Por lo que el subir una imagen es un proceso obligatorio en el proceso de creación de un grupo de objetos tal como se ve en la figura 5.9.

#### **Creación de objetos**

Luego de haber generado el grupo de objetos, el sistema se dirigirá a la ventana de adición de objetos. Esta vista es la misma a la que hubiera ido en caso de haber seleccionado un grupo de objetos del cuadro de búsqueda.

Para empezar con la creación de elementos hay que elegir cuántos elementos se quieren crear y dónde se ubicarán. El proceso de gestión de ubicaciones se tratará más adelante. De momento se verá el contenido de la figura 5.11.

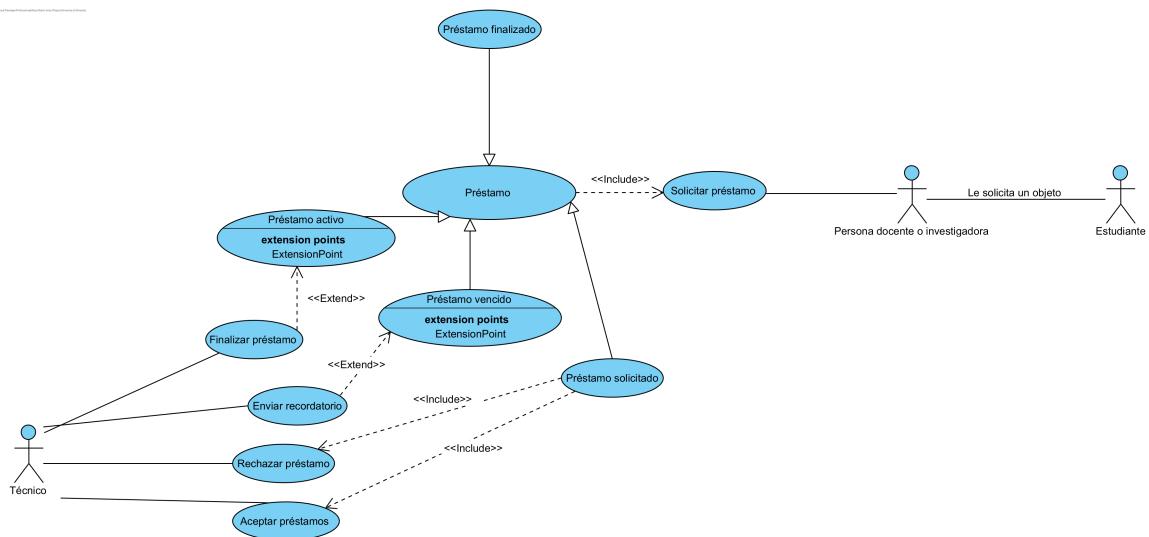


Figura 5.6: Funcionamiento y descomposición de préstamos

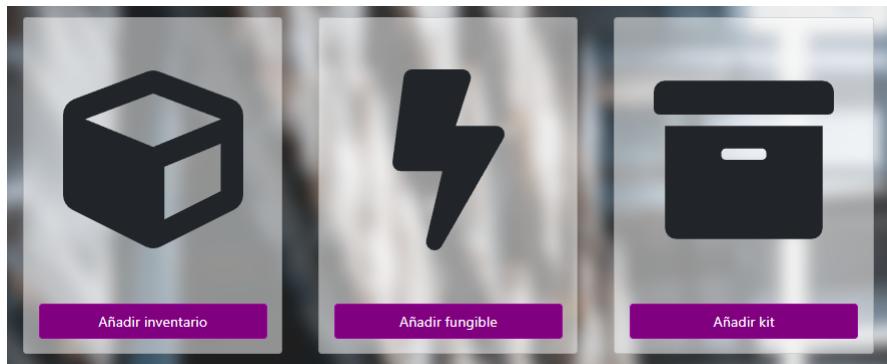


Figura 5.7: Funcionamiento y descomposición de préstamos

### Añadir datos a los objetos

Al realizar el paso anterior se proyectará una vista donde se irán añadiendo los datos de los objetos que quieran crearse. En caso de que no quieran añadirse los datos objeto por objeto, hay un botón habilitado que realiza la creación de estos de forma automática. Esto puede verse en la figura 5.12.

#### 5.3.2. Procesos de creación y modificación

Estos procesos utilizan el mismo componente ya que la acción de modificación permite cambiar los mismos atributos que la acción de creación. Puede verse un ejemplo en la figura 5.10.

#### 5.3.3. Proceso de creación, selección y eliminación de ubicaciones

Una ubicación en la Universidad de Almería consta de tres atributos. Un edificio, una planta y una localización dentro de esa planta.

En base a estos distintos atributos se realizó el componente encargado de gestionar la ubicación de un objeto.

Este componente consta de tres etapas de selección: la primera donde empiezas seleccionando el edificio, figura 5.13, luego podrás ver las plantas, figura 5.14, y luego las localizaciones dentro de esas plantas, figura 5.15.



Figura 5.8: Cuadros para iniciar la búsqueda de grupos de objetos

Figura 5.9: Añadir imagen y datos al grupo de objetos

Figura 5.10: Creación y modificación de elementos

En este proceso está presente en la parte superior una barra de búsqueda que también hace la función de creación. Es decir, en el caso de que no se encuentre un edificio, se escribe el nombre y se pulsa el botón de creación. A partir de ahí se pasa a la siguiente etapa que es crear una planta para ese edificio. Hasta que no se llega a la etapa final de la creación de la localización esta ubicación no se genera.

Para poder eliminar las ubicaciones basta con seleccionar la localización que se quiera eliminar y en caso de que esta esté enlazada con un objeto se pedirá que se eliminen dichos objetos o se les cambie la ubicación. Si no se hace, el sistema impedirá eliminar la ubicación.

Número de objetos  
1

Ubicación del/los objeto/s\*  
Debes añadir una ubicación al objeto obligatoriamente.  
Seleccionar ubicación

**Crear objetos**

Figura 5.11: Número de objetos y localización

Código  
Código del inventario  
dd/mm/aaaa

Etiqueta  
Etiqueta del equipo

Área  
Departamento de informática

Mejoras en el equipo  
Mejoras en el equipo

Observaciones  
Observaciones del equipo

**Crear objeto**  
Saltar mejoras y crear objetos restantes

Figura 5.12: Funcionamiento y descomposición de préstamos

Busca el edificio o crea uno nuevo  
Escribe tu búsqueda o el nombre del nuevo objeto  
Escribe el nombre del objeto antes de crearlo

**Crea un edificio**

CITE III  
CITE IV

**Cerrar**

Figura 5.13: Selección o creación del edificio

Busca la planta o crea una nueva  
Escribe tu búsqueda o el nombre del nuevo objeto  
Escribe el nombre del objeto antes de crearlo

**Crea una planta**

Planta 1

**Atrás**

Figura 5.14: Selección o creación de la planta

Busca la ubicación o crea una nueva  
Escribe tu búsqueda o el nombre del nuevo objeto  
Escribe el nombre del objeto antes de crearlo

**Crea una ubicación**

Despacho 13

**Atrás**

Figura 5.15: Selección o creación de la ubicación

---

# 6 Deploy del sitio web

Al terminar la construcción de la página lo que se hará será preparar los archivos para añadirlos al servidor web.

Uno de los objetivos de la preparación del deploy del sitio es que estuviera todo en un archivo docker-compose. Esto en vistas a una mejor configuración y a poder asegurar el entorno antes de llevarlo a producción.

Por lo que el fichero docker-compose se tendría que encargar de levantar los siguientes elementos:

- La base de datos de MariaDB.
- El servidor Node que manejará la API.
- El servicio de PHPMyAdmin para manejar la base de datos ante cualquier problemática.
- El servidor Web con el Deploy de Angular que irá con Nginx.

Más adelante en un futuro se añadiría otro bloque más que sería el gestor de copias de seguridad de la base de datos.

## 6.1. Levantar el servidor para la API

Para poder levantar el servidor con la API era necesario sí o sí realizar una configuración de una imagen docker Node para poder inicializar el Node Package Manager.

También, con el objetivo de ahorrar intermediarios, se quería poder generar la imagen Docker desde Docker Compose.

Un fichero Dockerfile sirve para generar una imagen docker (no un contenedor) que se podrá ir manipulando para que tenga la configuración que sea necesaria para que provea de los servicios necesarios para UAL Inventarium.

La estructura de un archivo Dockerfile es la siguiente:

```
FROM node:16.4
```

Primero se define la imagen desde la que se partirá. Este es un proceso obligatorio. Se generará la imagen para la API desde la versión de node 16.4 que es con la que se ha llevado a cabo el desarrollo.

```
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
```

El segundo paso será definir la dirección del directorio de trabajo. Esta dirección */app* será donde se ubicará el directorio raíz de la API.

Luego se copiarán todos los archivos que empiecen por *package* que en este caso son dos: *package.json* y *package-lock.json*. Dentro de estos ficheros se encuentra toda el versionado y la

configuración de los plugins que se utilizarán en el sistema.

Luego de copiar la configuración del versionado se procederá a iniciar Node Package Manager. Este, en base a los paquetes que se han declarado anteriormente, será el encargado de generar todos los directorios que necesita Node para funcionar.

Después de haber instalado NPM el siguiente paso es copiar todo el directorio raíz de la API dentro de la imagen que está siendo creada, eso se hará con “*COPY . . .*”.

### ¿Por qué no se copia directamente el directorio raíz con los plugins y demás?

Lo que ocurre cuando haces una copia exacta es que tu sistema ha adaptado las librerías y alguna configuración extras a tu entorno. Por lo que luego al contenerizarlo da error. Lo bueno que permite Docker es que esté donde esté el sistema que se vaya a utilizar, mientras esté en un contenedor, esté será el mismo para todos los usuarios.

```
EXPOSE 3000  
CMD ["npm", "run", "dev"]
```

Lo siguiente sería abrir el puerto 3000 para que el usuario pueda mandar solicitudes a la API y ejecutar el comando *npm run dev* para que esta empiece a funcionar.

Junto a este Dockerfile también hay que generar un *.dockerignore* que al igual que *.gitignore* sirve para que en el momento de la creación de la imagen que se hace con el Dockerfile el sistema Docker no coja ni los ficheros ni directorios que se encuentren declarados dentro del documento. El archivo contiene los siguientes campos de texto:

```
node_modules  
Dockerfile  
.git
```

Se ignorará el directorio *node\_modules* que es donde se descargan los paquetes. El *Dockerfile* y el *.git* es para que la imagen no tenga contenido extra que no se vaya a utilizar.

Con todo esto ya definido se pasaría a añadir el apartado de la API en el Docker Compose.

```
api:  
  build:  
    context: ../API  
    dockerfile: ../API/Dockerfile
```

Dentro de la sección *build* se define el sitio donde Docker Compose generará la imagen que se vaya a utilizar para el contenedor. En el apartado *context* se define el directorio que será la raíz de la generación y en el apartado *dockerfile* como su propio nombre indica se seleccionará el archivo Dockerfile a utilizar.

```
image: inventarium_api:1.0  
restart: always
```

Aquí se define el nombre de la imagen que se esté creando y *restart: always* servirá para que en el momento en el que el contenedor se pare este se vuelva a iniciar.

### ¿Cuándo se para un contenedor?

Un contenedor puede dejar de funcionar por varias razones pero, la principal, es porque se ha quedado sin tareas que realizar. En el caso de que ocurriese eso en el sistema, significaría que la API ha dado un error y se ha parado. Al no querer que esta deje de funcionar se programa para que se vuelva a encender.

```

container_name: api
ports:
  - "3000:3000"
volumes:
  - ./API:/app
networks:
  - default

```

Por último se define el nombre del contenedor, para que redirija del puerto 3000 a el puerto 3000 del sistema, que realice un copiado de archivos del contenido de la carpeta API dentro de la raíz del sistema en caso de que haya nuevos archivos para subir y por último que esté funcionando sobre la red con el nombre *default*.

## 6.2. Levantar el servidor Web

Han surgido dos problemas principales al intentar realizar esto:

1. Configurar las rutas de la aplicación desde el servidor web.
2. Configurar un proxy reverso desde el servidor web.

El servidor web que se utilizó fue **Nginx**. La configuración inicial fue bastante rápida. Quedando el fichero *docker-compose.yml* así en un principio:

```

client:
  image: nginx:latest
  ports:
    - 80:80
  volumes:
    - ./inventarium/dist/inventarium:/usr/share/nginx/html

```

Se importa la última imagen de nginx, se redirige el puerto 80 al puerto 80 y por último se importa el contenido del sitio web dentro de el directorio */usr/share/nginx/html* de nginx.

En un principio parecía que estaba todo bien pero al acceder al sitio web ocurrieron dos cosas: que no llegaban las solicitudes a la API y que tampoco podía accederse al archivo de rutas más allá que la raíz de Angular. Es decir, no podía accederse a la ruta */group-of-objects* por ejemplo. Con *ng build* no se había importado el proxy que se había creado en el sitio web y tampoco funcionaba la configuración de rutas, aunque, sí se podía navegar dentro de la aplicación.

Esto se debía a la configuración de rutas que estaban configuradas en nginx. Rutas que se modificaban desde */etc/nginx/conf.d/default.conf*.

Por lo tanto se creó un archivo llamado *default.conf* y se añadió una línea más en la sección de *volumes* del fichero del Compose. Esta línea era:

```
- ./web/default.conf:/etc/nginx/conf.d/default.conf
```

Que ayudaba a configurar las rutas en *default.conf*.

### 6.2.1. Configuración del ruteo de la web

El objetivo de configurar las rutas de la web es que todas las direcciones apuntásen al mismo archivo *index.html*. Para eso se añadieron dentro del fichero las siguientes líneas:

```

location / {
  root /usr/share/nginx/html;
  try_files $uri $uri/ /index.html;
}

```

Esta configuración define las rutas entrantes a la dirección raíz “/”, en el caso de que sean allí apuntará al directorio *html* que ha sido indicado anteriormente y si no es el caso lo hará sobre *try\_files \$uri \$uri/ /index.html;*. Sobre *index.html*. Es decir, lo mismo que si apuntara a la raíz del servidor.

### 6.2.2. Configuración del proxy inverso de la web

Como se explicó en la sección 4.5, para poder conseguir que la aplicación se comunique con la API esta tiene que disponer de un proxy que rediriga unas determinadas rutas dentro del puerto 80 a otras del puerto 3000.

Para poder hacer esto en Nginx hay que añadir las siguientes líneas de código en el archivo anterior:

```
location /api/users/login {  
    proxy_pass http://api:3000;  
    proxy_pass_request_headers on;  
}
```

Sigue casi la misma estructura que en el apartado anterior siendo *proxy\_pass* la dirección donde se reenviarán las peticiones. El dominio donde las reenvia se llama *api*, esto es debido a que dentro del entorno que genera Docker Compose pueden llamarse a las máquinas creadas en base a su referencia.

La siguiente línea sirve para poder reenviar el encabezado de las solicitudes para que en la ampliación del proyecto pueda generarse un sistema de tokens con el objetivo de querer aumentar la seguridad.

Con esto ya estarí creado el servidor web donde se localizaría la aplicación.

---

# 7 Pruebas durante el desarrollo y corrección de errores

Una de las etapas más importantes durante el desarrollo de una aplicación es la de la creación y ejecución de pruebas. Desde ir probando determinadas configuraciones en la creación de formularios, comprobaciones de la navegabilidad e interacción de componentes hasta comprobar la fluidez y los tiempos de carga.

Antes de sacar un proyecto a producción han de pasar por un testeo y eso sobre lo que tratará este último capítulo.

## 7.1. Pruebas durante el desarrollo

Las herramientas que han utilizado para la realización de pruebas durante el desarrollo del proyecto han sido tres:

- **Postman:** Para poder realizar pruebas sobre la API.
- **Selenium:** Para poder realizar pruebas más que nada orientadas al envío de formularios.
- **Herramientas de Desarrollador de Google Chrome:** Para poder controlar la consola, algunos problemas de estilos y el envío y recibo de paquetes en la red.

### 7.1.1. Postman

Gracias a Postman se ha podido probar en este proyecto todas las interacciones que se pueden realizar con la API y el proxy configurado en Angular. Como puede comprobarse en la figura 7.1 se ha generado una solicitud POST para mandarla a la dirección `/api/users` de la página web. Que a su vez la redirigirá al registro probando el proxy.

Se añaden los parámetros dentro del campo *form-data* ubicado dentro de *Body* que es la forma en la que se ha planificado el procesado de las peticiones.

Luego de pulsar en el botón *Send* puede comprobarse que se manda la solicitud porque llega una respuesta que de la API, en este caso: '*message*': '*usuario created successfully*'

Un punto bastante a favor que presenta Postman es que el conjunto de solicitudes que tengas se almacena en tu usuario por lo que al cambiar de dispositivo estas solicitudes se siguen manteniendo.

Otra herramienta que tiene incorporada esta aplicación te permite capturar solicitudes que salgan de un determinado puerto. Aparte de capturarlas, interpreta en su totalidad las estructuras de estas y puede comprobarse si se ha cometido algún fallo al mandar cualquier tipo de solicitud desde la página.

### 7.1.2. Selenium

Gracias a la herramienta de grabación que aporta Selenium se han realizado bastantes pruebas en la página web.

Para poder utilizar la herramienta se ha hecho desde Google Chrome instalando la siguiente extensión.

Luego de instalarla hay que dirigirse a la lista de complementos del navegador y seleccionar la

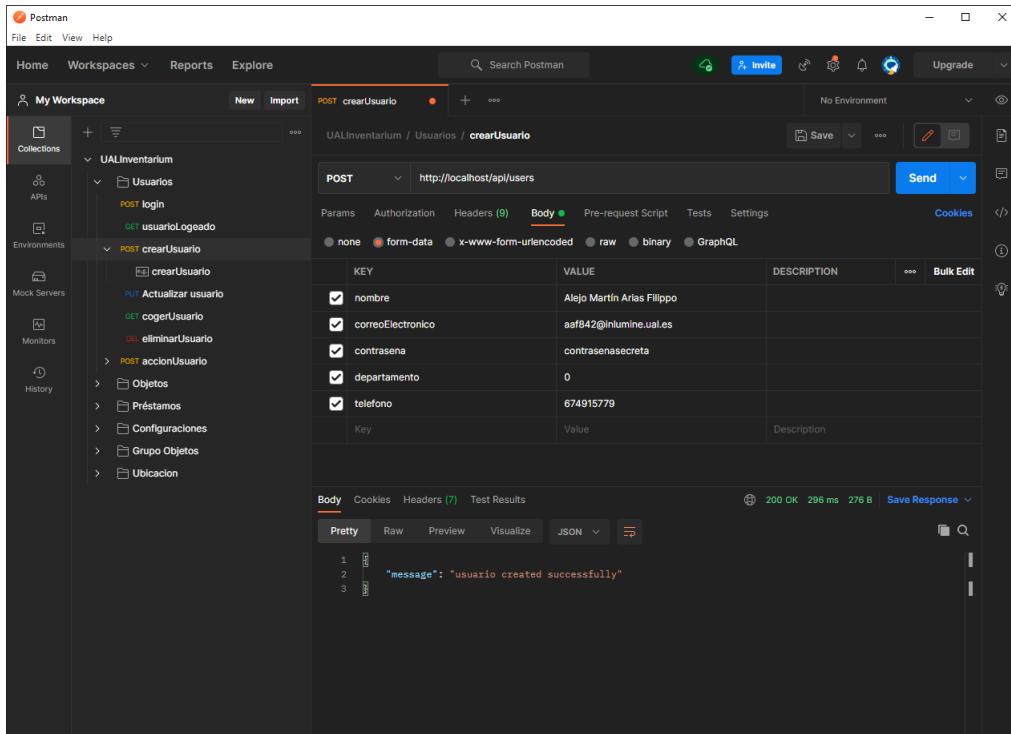


Figura 7.1: Creando un usuario con Postman

aplicación. Se pulsará en crear nuevo proyecto y la aplicación pedirá una *url* para empezar a grabar pruebas. Se añadirá la url *http://localhost/register* para poder automatizar el proceso de registro.

Al empezar la grabación se abrirá una nueva ventana de Google Chrome y se completarán los datos como un proceso normal de registro pulsando finalmente en *Registrarse*.

La prueba que se ha generado puede verse en la figura 7.2

### 7.1.3. Herramientas de Desarrollador de Google Chrome

Las Herramientas de Desarrollador de Google han sido utilizadas en todas las fases del desarrollo de la aplicación. Desde las más tempranas hasta las más tardías.

El navegador Google Chrome brinda un conjunto de herramientas muy completo. Las tres que más se han utilizado han sido:

- **Elements:** Aquí pueden verse los elementos presentes en pantalla. Disponem tambié de un inspector que permite pulsar sobre un elemento y localizarlo dentro de la estructura HTML que presenta el documento. También permite poder editar los estilos con los que se esté trabajando y ver los cambios en tiempo real. Estos no se aplican al documento pero ayudan en gran medida a realizar arreglos de diseño.
- **Console:** Desde aquí pueden verse las salidas de consola que da la aplicación. En Angular para poder emitir señales en la consola se utiliza *console.log("Elemento que quiera emitirse")*. Desde aquí pueden verse fallos que haya devuelto la aplicación y poder actuar en medida.
- **Network:** Esta sección es de gran ayuda porque desde aquí se pueden ver todos los paquetes entrantes y salientes desde el punto de vista del cliente.

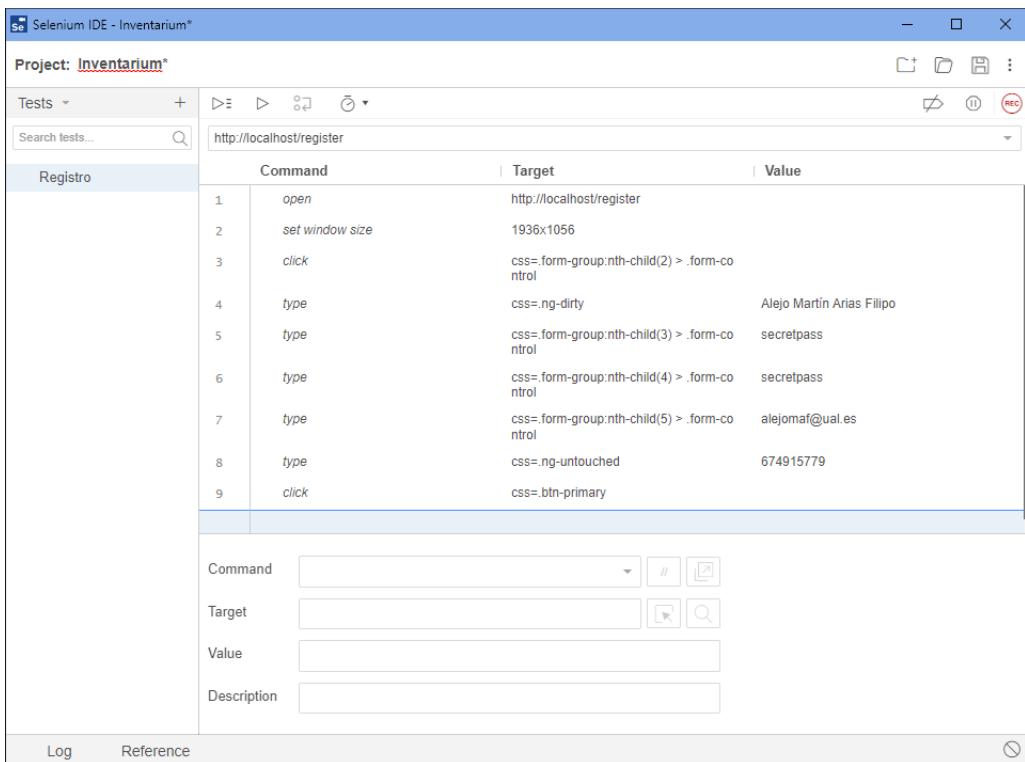


Figura 7.2: Registro de un usuario con Selenium

## 7.2. Corrección de errores

Para poder realizar esta sección se han ido recopilando las correcciones más significativas que ha sufrido el proyecto.

### 7.2.1. Cambio de la base de datos

Cuando se estaba realizando la construcción del sitio web con la base de datos y la API ya programadas se pidió crear un nuevo tipo de objeto. Este objeto serían los kits.

Los kits consistían en un grupo de objetos que en realidad eran un conjunto de otros objetos. Poder remodelar la página para adaptar la lógica de los kits fue bastante complicado pero la solución bastante buena.

En un principio se iban a implementar dos nuevas tablas a la base de datos, una que se llamaría **Kit** y otra **ObjetoKit**. Esto hubiera modificado la mayoría de consultas de generación de préstamos y suponía un aumento de la complejidad de la aplicación bastante grande. Había que distinguir en cada consulta si un elemento era un grupo de objetos o un kit.

Un tiempo después se comprobó que no hacía falta implementar un elemento del tipo Kit, ya que esta funcionalidad la cubría grupo de objeto.

El formato terminó consistiendo en que un grupo de objetos contenía dos tipos de parámetros diferentes: objetos u objeto kit. Los objeto kit eran los objetos de los que estaba compuesto el kit, en caso de serlo. Mientras que los objetos era el número de elementos con sus distintos atributos que podía tener.

### 7.2.2. Cambios de solicitudes

En las etapas intermedias del proyecto cuando estaba siendo realizado el proceso de creación de un grupo de objetos se tuvo que hacer una readaptación.

El problema ocurría a la hora de estar subiendo imágenes a la base de datos y es que para poder

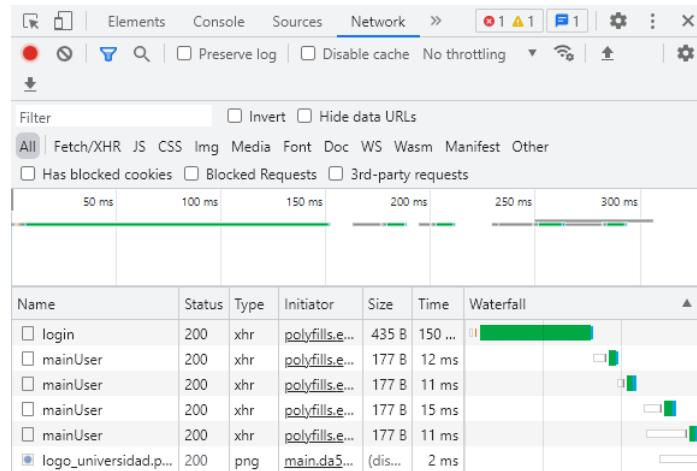


Figura 7.3: Sección *Network* dentro de las herramientas de desarrollador de Google Chrome

procesar y enviar estas imágenes se tuvieron que usar campos de formularios.

No había problema hasta que los plugins que había de procesamiento de formularios no eran compatibles con el que se tenía para poder procesar solicitudes en formato json.

Así que por las imágenes tanto de los grupos de objetos como las de los objetos de kit se tuvo que cambiar todo el repertorio de solicitudes que había sido creado para la creación, modificación, actualización y eliminación de elementos para adaptarlos con un formato de formularios.

### 7.2.3. Conexión entre máquinas de Docker Compose

Resulta que al configurar la API con la base de datos había que esperar a que la base de datos se encendiera, como se ha explicado en la configuración del documento de docker compose.

Esta estaba mandando constatemente solicitudes TCP al puerto 3306 que es el que usaba la base de datos y posteriormente cuando se intentaba conectar daba un error de conexión.

Al analizar los errores en la terminal se vió que la base de datos había mandado un mensaje que decía que no había sido posible conectar con ella porque las credenciales eran del estilo ‘user’:‘unauthenticated’ y ‘password’:‘unset’.

La solución fue, como se ha señalado anteriormente en este documento: que, para poder realizar una conexión con una máquina en un entorno local de Docker desplegado con Docker Compose hay que referenciar la máquina con el nombre con la que se había creado. Quedando así:

```
const connection = await mysql.createConnection({
  host: 'db',
  user: 'ualinventarium',
  password: 'secretpassword',
  database: 'ualinventarium',
  port: '3306'
});
```

---

# **8 Conclusiones y posibles mejoras**

## **8.1. Posibles mejoras**

### **Adición de capa superior**

Una de las posibles mejoras que se contempló al inicio del desarrollo del proyecto era la factorización de la aplicación. Es decir, poder añadirle una capa extra de generalización a todos los componentes utilizados.

Estas modificaciones conllevaría que podría registrarse un sistema de gestión de inventarios y que cada uno de estos gestionaría sus propios usuarios. Esto implicaría tener que habilitar nuevos end-points al sitio web y más complicado aún, habría que pensar en una forma de generalizar la información que se almacenara en cada uno de los componentes.

### **Gestor de notificaciones**

La siguiente mejora era más viable: implementar un gestor de notificaciones. Un gestor de notificaciones hubiera ayudado en el aviso de acciones de concesión, rechazo, creación y caducidad de préstamos tanto para los usuarios como para los técnicos.

También se hubiera podido utilizar en el momento de generación y creación de nuevos objetos para que los usuarios vieran la disponibilidad y adquisiciones de inventario del departamento.

### **Generalización de tipos de objetos**

La tercera y última modificación va muy relacionada con la primera. Esta sería la de poder generalizar la creación y manipulación de subcomponentes en un grupo de objetos. Consistiría en permitir la creación de una capa de aislamiento de cada objeto. Con esto no haría falta tener que diferenciar únicamente entre inventario, fungible o kit sino que de esto ya se encargaría Inventarium.

### **Inclusión de horarios de laboratorio**

Una información presente en las hojas de cálculo es la de los horarios de los distintos laboratorios del CITE III. Esto, a pesar de no ser una funcionalidad en sí a lo que respecta al manejo del inventario y los préstamos, puede ser una nueva funcionalidad a implementar con el objetivo de facilitar el trabajo de los técnicos.

## **8.2. Conclusiones**

La conclusión de un trabajo a la que le he puesto tanta dedicación se me hace un poco complicado.

Durante el desarrollo de este proyecto he podido ir aprendiendo aspectos muy importantes dentro del desarrollo de aplicaciones. Además he podido unir los distintos aprendizajes que he obtenido en el Grado y utilizarlos en gran medida en cada una de las distintas tecnologías y apartados de la aplicación.

En un principio deseaba poder tener una herramienta que con tan solo escribir una línea en la terminal:

`docker compose up`

Pudieramos tener una aplicación en su totalidad, con base de datos, API, servidor web y más adiciones que he podido realizar posteriormente en el desarrollo del complemento de este trabajo. Creo que el resultado final no difiere mucho de lo que en un momento quería tener. Un sistema que es capaz de implementar una persona que utilice el ordenador para ofimática y navegación web. Una herramienta fácil y accesible por todos.

Una bonita definición de lo que es la ingeniería.

---

# Referencias

- Angular. (s.f.-a). *Common routing tasks*. Descargado de <https://angular.io/guide/router>
- Angular. (s.f.-b). *Introduction to services and dependency injection*. Descargado de <https://angular.io/guide/architecture-services>
- Angular. (s.f.-c). *Workspace and project file structure*. Descargado de <https://angular.io/guide/file-structure>
- Angular. (18 de marzo de 2021). *Angular components overview*. Descargado de <https://angular.io/guide/component-overview>
- Bob, U. (13 de agosto de 2013). *The clean architecture*. Descargado de <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>
- Docs, D. (s.f.). *Use docker compose*. Descargado de [https://docs.docker.com/get-started/08\\_using\\_compose/](https://docs.docker.com/get-started/08_using_compose/)
- Docs, N. (s.f.). *Nginx reverse proxy*. Descargado de <https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/>
- Ellingwood, J. (2 de diciembre de 2020). *Cómo usar ssh para conectarse a un servidor remoto*. Descargado de <https://www.digitalocean.com/community/tutorials/how-to-use-ssh-to-connect-to-a-remote-server-es>
- Google. (s.f.-a). *Chrome devtools*. Descargado de <https://developer.chrome.com/docs/devtools/>
- Google. (s.f.-b). *Desarrollar soluciones en google cloud*. Descargado de <https://cloud.google.com/docs>
- Izquierda, C. (1 de diciembre de 2018). *Cómo se hace: Api testing con postman*. Descargado de <https://medium.com/@cesiztel/c%C3%B3mo-se-hace-api-testing-con-postman-978a521552f4>
- Josh. (1 de abril de 2020). *How to use mysql or mariadb with node.js and express*. Descargado de <https://forum.codeselfstudy.com/t/tutorial-how-to-use-mysql-or-mariadb-with-node-js-and-express/2260>
- Juell, K. (5 de diciembre de 2019). *Cómo crear una aplicación node.js con docker*. Descargado de <https://www.digitalocean.com/community/tutorials/como-crear-una-aplicacion-node-js-con-docker-es>
- McCormick, M. (8 de octubre de 2012). Waterfall vs. agile methodology. *MPCS, Inc.*. Descargado de [http://www.mccormickpcs.com/images/Waterfall\\_vs\\_Agile\\_Methodology.pdf](http://www.mccormickpcs.com/images/Waterfall_vs_Agile_Methodology.pdf)
- Mouat, A. (25 de diciembre de 2015). *Using docker: Developing and deploying software with containers*. O'Reilly Media.
- M.W. Maier, R. H., D. Emery. (abril de 2001). *Software architecture: introducing ieee standard 1471*. Descargado de <https://ieeexplore.ieee.org/document/917550>

## Referencias

---

- Orcero, D. S. (13 de enero de 2020). *La biblia de latex*. Lulu Press.
- Rainville, S. (s.f.). *How to configure nginx for angular and reactjs*. Descargado de <https://www.serverlab.ca/tutorials/linux/web-servers-linux/how-to-configure-nginx-for-angular-and-reactjs/>
- Rangle.io. (s.f.). *Interfaces*. Descargado de <https://angular-training-guide.rangle.io/features/typescript/interfaces>
- RedHat. (31 de octubre de 2017). *¿qué es una api?* Descargado de <https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces>
- Rungta, K. (11 de diciembre de 2021). *What is selenium? introduction to selenium automation testing*. Descargado de <https://www.guru99.com/introduction-to-selenium.html>
- Series, R. C. M. (17 de septiembre de 2017). *Clean architecture: A craftsman's guide to software structure and design*. Addison-Wesley.
- Yakov Fain, A. M. (21 de marzo de 2019). *Angular development with typescript*. Manning Publications.