Universidad de **Valladolid**

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Software

Plataforma web para la gestión de contenedores Docker

Autor:
Jairo José Pérez Calvo

Universidad de **Valladolid**

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Software

Plataforma web para la gestión de contenedores Docker

Autor:
Jairo José Pérez Calvo

Tutores:
Valentín Cardeñoso Payo
Cristian Tejedor García

A mi familia, que ha hecho esto posible.

Agradecimientos

Este proyecto ha salido adelante gracias a muchas personas. En primer lugar, muchas gracias a mis amigos, que me han ayudado a poder desconectar de este proyecto, que en algunas ocasiones parecía que era en lo único que sabía pensar. Por otra parte, dar gracias a mi familia, que me ha dado el apoyo necesario de forma constante. Finalmente, quiero agradecer especialmente toda la ayuda que me han ofrecido mis tutores, Valentín Cardeñoso y Cristian Tejedor, que me han ayudado continuamente en todas las dudas que he tenido, y han revisado mi trabajo periódicamente para que éste haya llegado al estado en el que se encuentra actualmente.

Resumen

En los últimos años, se ha popularizado mucho el uso de contenedores para desplegar aplicaciones debido a la facilidad que proporcionan para gestionar dependencias de software y evitar conflictos entre distintas aplicaciones. Esta tecnología combina lo mejor de desplegar aplicaciones de forma nativa, para obtener un máximo rendimiento de una máquina, y la portabilidad que se obtiene cuando se despliegan en máquinas virtuales. *Docker* proporciona a los desarrolladores las funciones de empaquetar, enviar y ejecutar aplicaciones dentro de un contenedor ligero, portable y autosuficiente, que se puede ejecutar en cualquier máquina. Se pueden desplegar aplicaciones muy sencillas, como un hola mundo; y por otro lado aplicaciones de mayor complejidad, que dependan de una base de datos, utilizando cualquier lenguaje de programación, o conectar varias máquinas con distintos sistemas operativos. La configuración de estos contenedores requiere aprendizaje para poder programar y desplegar estas aplicaciones porque estas tienen multitud de opciones que fijar. En este proyecto se propone una aplicación web que sirva para el despliegue de contenedores *docker*. Esta aplicación tiene como objetivo facilitar a usuarios con poco conocimiento en *docker*, gestionar (crear, borrar, modificar) contenedores.

Abstract

In recent years, the use of containers to deploy applications has become popular due to the facility that it provides to manage software dependencies and to prevent conflicts between applications. This technology combines the best of deploying native applications, which is when it takes advantage of computer performance, and the portability you get when deploying containers on virtual machines. *Docker* enables developers to easily pack, ship, and run application as a lightweight, portable, self-sufficient container, which can run virtualy anywhere. Straightforward applications can be deployed, like a hello world app, and on the other hand you can deploy complex applications which need a database, using another programming language, etc. The containers configuration requires learning to know how to develop and deploy this applications because they have a lot of options to set. In this project a web application which deploys docker containers is developed. The goal of this application is to ease low-level *docker* knowledge to manage (create, delete, modify) *docker* containers.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1 Contexto

Docker [1] es una tecnología de contenedores que sirve para desplegar de manera aislada distintos servicios. Tiene una gran comunidad de soporte detrás [2] y su documentación es más que amplia [3], por lo que es una tecnología muy interesante. Uno de los casos en los que esta tecnología podría ser útil es en un grupo de investigación. La realización de prácticas en empresa en el grupo ECA-SIMM [4] de la Universidad de Valladolid ha servido para aprender sobre esta tecnología. Durante la realización de las prácticas, después realizar una profunda revisión a todas las posibilidades que ofrece *docker* se ha utilizado esta tecnología para poder desplegar múltiples proyectos y se ha estado implementando en múltiples servicios que el grupo ya tenía desarrollados. Para ello se han utilizado multitud de comandos a través de consola para poder lanzar y configurar estos contenedores. Todo esto ha servido para mejorar la gestión de todos los proyectos y para así poder tenerlos desplegados de una manera mucho más ordenada, además de eliminar todos los problemas que pudiera haber con las dependencias y versiones en cada uno de esos proyectos.

La gestión de contenedores es muy importante ya que cada vez más se despliegan servicios utilizando esta tecnología debido a las ventajas que ofrecen. Un contenedor es un paquete formado por un programa o servicio a ejecutar y sus dependencias. Funciona de forma autónoma gracias a que incluye las librerías en su interior. Para orquestar los contenedores se pueden utilizar distintas tecnologías. La más utilizada en las prácticas ha sido *docker-compose* [5], ya que ofrece la posibilidad de crear contenedores independientes pero conectados para un mismo proyecto, así como puede ser un contenedor con la página web y el otro para la base de datos, o incluso podríamos crear uno para el *frontend*, otro para el *backend* y un último para la base de datos.

Recientemente *Docker* se ha utilizado en ciertos trabajos de fin de grado (TFG) con distintas orientaciones. Uno de los usos sería para desplegar una aplicación web, como se indica en este proyecto de la Universidad Politécnica de Madrid [6]. Este caso se parece mucho a lo que se ha realizado en las prácticas en empresa, con la diferencia de que en las prácticas se han utilizado proyectos desarrollados por alumnos en el pasado. Por otro lado, en este TFG [7] de la Universidad de Las Palmas de Gran Canaria

se utiliza *docker* en una *Raspberry Pi* [8] para el despliegue de aplicaciones IOT [9], aprovechando que tiene compatibilidad con este dispositivo. Como último ejemplo, en la Universidad Politécnica de Valencia se ha desarrollado un TFG utilizando *docker* para agilizar el proceso de despliegue de aplicaciones gracias a las ventajas que esta tecnología proporciona [10].

1.2 Motivación

Una vez desplegados servicios del grupo de investigación mediante la tecnología *docker*, el grupo necesita una forma sencilla de poder desplegarlos y gestionarlos para futuros trabajos de fin de grado y aplicaciones para proyectos de investigación, entre otros. Tanto como si se usa *docker-compose* o *docker* directamente, se necesita conocer una serie de comandos y parámetros para poder gestionar todo esto, lo cual hace más complicado aprender a utilizar todo correctamente.

En los TFG revisados en el apartado anterior, ocurre el mismo problema que con los proyectos que se han implementado durante la realización de las prácticas curriculares, y es que se necesitan conocimientos para poder configurar todos los parámetros que *docker* ofrece. Todo esto provoca que surja la necesidad de buscar formas más sencillas de desplegar los contenedores.

Existe una interfaz gráfica en *docker* en la versión de escritorio *Docker-Desktop* [11] en la que podemos ver y monitorizar el estado de los contenedores que tenemos en nuestro sistema en ese momento. Sin embargo, esta plataforma tiene ciertas limitaciones:

- No sirve para crear contenedores avanzados. Solo podremos iniciar, detener o eliminar los que tengamos ya creados y crear contenedores basados en imágenes locales creadas previamente. Por tanto, seguimos necesitando conocer comandos y parámetros para poder crear contenedores.

- No es multiplataforma, por lo que solo nos funcionará si tenemos una máquina con un S.O. que soporte la interfaz. Actualmente solo está disponible para *Mac* y *Windows*, por lo que si utilizamos *Linux* en el servidor de despliegue, que es lo más habitual, no nos servirá.

- No funciona de manera remota, es decir, que solo podremos gestionar los contenedores que tengamos en la máquina desde la que accedemos.

Conociendo las características de *docker* en la versión de escritorio, quedan claras las carencias que tiene y lo que sería necesario para que fuera realmente útil, sin embargo, sí que sirve a modo de inspiración para lo que queremos realizar. La propuesta que surge es realizar una aplicación que sirva para el grupo de investigación ECA-SIMM, y por tanto, el desarrollo de la misma debe estar orientado a proyectos de tamaños similares a los que realiza este grupo.

1.3 Objetivos

En este proyecto se intenta poder gestionar todo el sistema de *docker*, así como todos los distintos proyectos que se tengan *dockerizados* (proyectos que están adaptados para desplegarse con *docker*), de tal forma que desplegarlos o modificarlos sea mucho más sencillo mediante una interfaz que contenga todas las funcionalidades descritas anteriormente.

El objetivo es realizar una aplicación web como ya se ha explicado en el apartado anterior, y debemos utilizar como API (Interfaz de programación de aplicaciones [12]) una tecnología para poder llevar a cabo el despliegue de los contenedores avanzados. Para la gestión más sencilla de contenedores se va a utilizar la tecnología que ya ofrece *docker* mediante *docker-engine*.

Para cumplir estos objetivos, necesitamos cubrir ciertas funcionalidades mínimas con la aplicación, las cuales serán subobjetivos. Inicialmente se han encontrado las siguientes operaciones:

- **Mostrar contenedores**
  Se deberán poder ver de forma visual todos los contenedores del servidor, clasificados según los contenedores que estén activos y los que no. También es interesante que se puedan agrupar los distintos contenedores cuando pertenecen a un mismo servicio, tal y como ocurre cuando se crea una web y una base de datos con *docker-compose*. Desde esa vista deberán poderse realizar las operaciones básicas de los contenedores, como parar, iniciar, borrar o editar.

- **Crear contenedores**
  Crear contenedores puede ser algo muy sencillo o muy complicado, según si simplemente queremos lanzar una imagen que ya tenemos creada o queremos subir un código fuente, que genere una nueva imagen, con una base de datos, etc. Esto nos lleva a que vamos a necesitar dos modos a la hora de realizar esta operación. Con un modo sencillo podemos cubrir la creación de contenedores cuando se parte de una imagen ya existente, y así, tener un modo profesional que se encargue de poder crear esos contenedores más complejos partiendo de un *docker-compose*.

- **Gestión de servidores**
  Al crear esta herramienta, la idea es poder gestionar los contenedores de múltiples servidores a la vez, por ello es necesario implementar una gestión de múltiples servidores para poder operar con varios de ellos al mismo tiempo y por ejemplo, ver todos los contenedores juntos. Esto puede resultar especialmente útil para cuando tengamos dos partes de una misma aplicación desplegadas en diferentes contenedores, aunque esto no vaya a ser lo habitual.

1.4 Tecnologías a utilizar

Una vez tenemos los principales objetivos del sistema, necesitamos ver con qué herramientas vamos a gestionar los contenedores del sistema. Para los contenedores sencillos y la gestión de ellos, vamos a utilizar la herramienta que proporciona *docker*, llamada *Docker-engine* [13], que proporciona

operaciones suficientes para gestionar todo lo que nosotros necesitamos y más. Para los contenedores avanzados, que tienen más opciones, y por tanto, son más complejos, necesitamos un sistema de gestión de contenedores más avanzado, lo que obliga a analizar distintas opciones.

En una primera fase del trabajo, se valoró *docker-swarm* [14], ya que es una opción nativa de *docker*. Sin embargo, después de analizar las opciones que ofrece se ha optado por emplear *docker-compose* en su lugar, dado que también forma parte de *docker* de forma nativa y no se ha visto imprescindible recurrir a la configuración en cluster para desplegar los contenedores, lo cual es el principal añadido que este ofrece. *Kubernetes* [15], por otro lado, es el orquestador de contenedores más utilizado en la actualidad. Está orientado a producción, por lo que se suele utilizar a gran escala. Se va a comparar con *docker-compose* para decidir cual de las dos es mejor para llevar a cabo las tareas de gestión de los contenedores. En la tabla 1.1 se comparan las principales diferencias entre estas tecnologías.

Tecnologías para desplegar contenedores avanzados	
Docker-compose	**Kubernetes**
Servicio creado y mantenido por docker.	Servicio creado y mantenido por *Google*.
Permite crear copias de un contenedor en un solo servidor.	Permite crear copias de un contenedor en distintos servidores.
Solo sirve para contenedores *docker*.	Sirve para múltiples tipos de contenedores que existen, aparte de *docker*.
Configuración relativamente sencilla conociendo ciertos parámetros en el fichero de configuración.	Configuración algo más compleja, ya que no es específica de la tecnología *docker*.

Tabla 1.1: Comparativa de las dos principales alternativas para lanzar contenedores.

Después ver esa tabla se tienen claras las principales diferencias que existen entre estos dos servicios. Si se profundiza más, se pueden encontrar muchas más diferencias, pero aquí se han agrupado las que probablemente sean las características más destacables y las que van a ser decisivas en elegir una u otra para este proyecto.

1.5 Metodología

Una vez revisadas todas las bases del proyecto a realizar, se ha decidido utilizar una metodología iterativa e incremental [16], que permita ir adaptando algunos aspectos conforme se vayan investigando todas las posibilidades que ofrecen las distintas herramientas sobre las que se va a trabajar. No es posible utilizar otro tipo de metodologías, como SCRUM, ya que dicha tecnología no es ideal cuando solo se involucre una persona en el desarrollo.

El uso de la metodología iterativa permite ir especificando más en detalle los requisitos y funcionalidades de la aplicación conforme se van investigando todas las posibilidades que ofrecen todas las herramientas que se van a utilizar. También será posible ir mostrando y probando los avances conforme

se vayan produciendo sin tener que esperar hasta el final.

En la figura 1.1 se puede ver un esquema de la metodología. Más adelante en esta memoria, en el capítulo 3, se podrán ver en detalle todas las iteraciones realizadas.

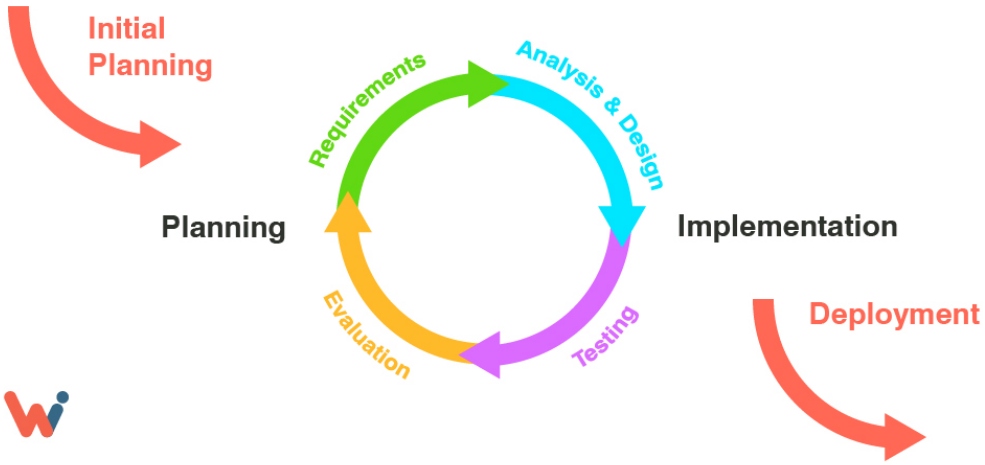


Figura 1.1: Esquema de la metodología. [16]

1.6 Estructura de la memoria

La estructura de esta memoria es la siguiente:

- **Introducción**: Capítulo en el que se explica el contexto, la motivación y los objetivos del proyecto.
- **Planificación**: Se expone la planificación que se va a seguir para desarrollar este proyecto así como las estimaciones de tiempo.
- **Descripción de las iteraciones**: Se detallan todas las iteraciones que se van a realizar explicando todo lo que se va a realizar en cada una.
- **Estado final de la aplicación**: Se explica el proceso seguido, explicando todos los diagramas y decisiones que se han tomado.
- **Pruebas**: Se explican las pruebas realizadas para asegurar el correcto funcionamiento de la aplicación.
- **Conclusiones**: Se explican las conclusiones obtenidas al haber realizado el proyecto.
- **Apéndices**: Documentos adicionales al proyecto, tales como el manual de instalación, manual del usuario, pruebas de usuarios, contenido del proyecto, o aspectos externos al proyecto pero que han sido necesarios para el desarrollo de la aplicación y tienen importancia.
- **Bibliografía**: Mención y referencias a obras y páginas web consultadas durante la realización de este proyecto.

Capítulo 2

Planificación

2.1 Planificación inicial

En este capítulo se va a planificar y organizar todo el tiempo de trabajo que va a ser necesario para realizar todas las iteraciones del proyecto, cumpliendo con todos los requisitos cuando se finalice el proyecto.

La duración del proyecto va a ser entre marzo y octubre, repartiendo de manera proporcional las 300 horas que marca la normativa del Trabajo Fin de Grado y contando con una pausa en los meses de verano. El proyecto se va a iniciar en el mes de marzo del año 2021 y se va a adaptar las horas semanales a la disponibilidad del alumno en cada momento.

El trabajo está dividido en las siguientes iteraciones:

- **1ª Iteración (terminada a finales de marzo) (15 horas)**
  Se realizarán las primeras estimaciones del proyecto, analizando por encima todas las alternativas que hay disponibles para realizarlo. Buscar diferencias de las distintas alternativas que existen y selección de qué servicios usar preferentemente.
- **2ª Iteración (terminada a mediados de mayo) (80 horas)**
  Aprendizaje de la plataforma elegida inicialmente. Realizar un estudio a fondo para ver que realmente con esta opción se pueden desarrollar correctamente todos los objetivos. Si fuera necesario habría que analizar más o menos detenidamente las alternativas disponibles. Realización de los primeros prototipos de la aplicación.
- **3ª Iteración (terminada a finales de septiembre) (160 horas)**
  Implementación de las funcionalidades más importantes hasta el punto de que ya se puedan utilizar. Se desarrollará primero la parte de gestión de contenedores, empezando por mostrarlos y gestionarlos, y se terminará con la creación de nuevos contenedores. Posteriormente se realizará la gestión de conexión a los distintos servidores a los cuales se podrá gestionar con este servicio. La interfaz gráfica debe ser bastante visual para que sea sencilla de utilizar. Se desarrollara de forma simultanea el *frontend* y el *backend* de cada parte. En el tiempo que se tardará en tener realizada esta iteración se está teniendo en cuenta el parón por vacaciones que va a afectar

principalmente a agosto.

- **4ª Iteración (terminada a mediados de octubre) (60 horas)**
  Finalización de la aplicación y arreglo de todos los errores que hayan quedado pendientes en las iteraciones anteriores. Realización de las últimas pruebas para completar el desarrollo.

En la Figura 2.1 se puede ver la distribución gráfica de las semanas.

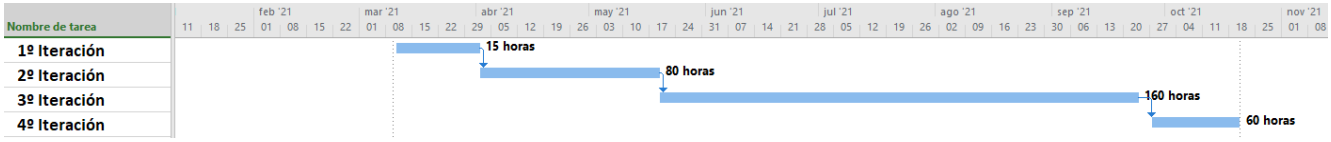


Figura 2.1: Distribución de las iteraciones según sus semanas de desarrollo.

Claramente, la tercera iteración es la que más tiempo consume, ya que es la que mayor número de horas tiene asignada y además coincide con el periodo en el que se va a estar parado durante unas semanas por vacaciones. La suma total de horas es 315 horas, que son las 300 mínimas más la primera iteración, que es dónde se han realizado las primeras estimaciones.

2.2 Entorno tecnológico

Se van a enumerar y describir brevemente todos los elementos del entorno tecnológico que se van a utilizar para el desarrollo de la aplicación.

2.2.1 Aplicaciones software utilizadas

A continuación se incluyen todos los programas utilizados en el desarrollo y despliegue de la aplicación.

- **Angular [17]:** Es la biblioteca de *node* con la que se va a desarrollar el frontend.
- **Astah [18]:** Es la herramienta elegida para hacer los diagramas *UML*.
- **Docker [1]:** Es la tecnología central de todo este proyecto. Se va a utilizar también en múltiples formas, como son *docker-compose* o *docker-engine*.
- **Git [19]:** Se va a usar *git*, de forma nativa por comandos para subir los cambios a *gitlab*.
- **Gitlab [20]:** Se va a utilizar para subir el repositorio del proyecto.
- **Google Meet [21]:** Es la aplicación que se va a utilizar para la realización de las videoconferencias.
- **MongoDB [22]:** Es el tipo de base de datos que se va a utilizar en el *backend* de la aplicación para almacenar todos los datos.
- **Mozilla firefox [23]:** El navegador que se va a utilizar para utilizar la aplicación.

- **Nginx [24]:** La aplicación desarrollada en *Angular* necesita del servidor *nginx* para ser desplegada.
- **Node.js [25]:** El *backend* va a estar desarrollado en *node.js*. Se van a utilizar los siguientes módulos de *npm*:

  dockerode [26]: Es la librería de *node.js* que se conecta con *docker-engine*. Permite realizar las operaciones de *docker* desde *node*.

  express [27]: Es la librería de *node* más habitual para realizar peticiones *http*.

  jsonwebtoken [28]: Es la librería de *node* que gestiona los *Json Web Tokens*, necesarios para la seguridad de la aplicación.

  mongoose [29]: Es la librería de *node* que se encarga de conectarse a la base de datos de tipo *MongoDB*.
- **Overleaf [30]:** La herramienta utilizada para hacer la memoria en *LaTeX*.
- **Postman [31]:** Se va a utilizar para realizar las peticiones *http* que van a servir para probar el *backend*.
- **Telegram [32]:** Es la aplicación que se va a usar para la mensajería.
- **Visual Studio Code [33]:** Es el *IDE* utilizado para desarrollar todo el código de la aplicación.

2.2.2 Equipos utilizados

A continuación se incluyen los equipos utilizados en el desarrollo y despliegue de la aplicación:

Ordenador portátil	
Lenovo G580	
Procesador	Intel Core i7-3520M @ 2.90GHz
RAM	8.0GB DDR3
Almacenamiento	SSD 480GB
Gráfica	NVIDIA GeForce 610M con 1GB dedicado
Sistema operativo	Windows 10
Arquitectura	64 bits

Tabla 2.1: Ordenador portátil utilizado en el desarrollo de la aplicación.

Servidor	
Máquina virtual	
Procesador	Common KVM processor @ 2.2GHz
RAM	3.85GB DDR3
Almacenamiento	10GB
Sistema operativo	Ubuntu 20.04 LTS
Arquitectura	64 bits

Tabla 2.2: Servidor utilizado para desplegar la aplicación

En la tabla 2.1 podemos ver las especificaciones del ordenador portátil en el que se ha realizado todo el desarrollo de la aplicación, incluyendo todas sus partes. En él se han utilizado todos los programas mencionados en el apartado anterior. También se han probado partes de la aplicación en este equipo en las primeras etapas del proyecto.

En la tabla 2.2 podemos ver las especificaciones del servidor. En este caso hay dos máquinas idénticas con estas especificaciones. Las dos se han utilizado para el despliegue de la aplicación. Hay una réplica de la aplicación de *docker* en cada máquina, lo que implica que en ambas se haya instalado *node*. El servidor de autenticación y gestión de servidores, en cambio, está solo en una máquina. Finalmente el *frontend*, que también se encuentra solo en una de las máquinas, necesita de *nginx* para estar desplegado, por lo que solo en esa máquina también se ha instalado el servidor *nginx*.

2.3 Estimación de riesgos

Se describen todos los riesgos que se han encontrado que puedan afectar al desarrollo del proyecto tal y como está planificado. Para realizar esta estimación se va a enumerar la lista de riesgos que se encuentren utilizando un identificador y una frase para nombrar el riesgo, una breve descripción, la probabilidad de que estos ocurran de 0 a 1 (siendo 0 nada probable que ocurra y 1 seguro que va a ocurrir) y cuanto tiempo retrasarían a la planificación. En la tabla 2.3 se encuentran todos los riesgos encontrados descritos mediante los apartados mencionados.

A continuación, en la tabla 2.4 se elabora un plan de acción para saber que hacer si ocurre alguno de todos esos riesgos. Se indica nuevamente el identificador y nombre del riesgo, la exposición al riesgo, que se calcula multiplicando la probabilidad de que ocurra un riesgo por el retraso (en días) que produce ese riesgo cuando se produce y por último el plan de acción a ejecutar con cada uno de ellos.

ID	Riesgo	Probabilidad	Descripción	Retraso
1	Suspender PGP en la convocatoria de fin de carrera.	0.1	El desarrollador no aprueba la asignatura de PGP en la convocatoria extraordinaria de fin de carrera, teniendo que esperar a la siguiente convocatoria.	60 días.
2	Suspender PGP en la convocatoria extraordinaria.	0.01	El desarrollador no aprueba la asignatura de PGP en la convocatoria extraordinaria, teniendo que esperar al siguiente curso.	305 días.
3	Caer enfermo	0.05	El desarrollador cae enfermo, estando varios días sin poder avanzar con el proyecto.	4 días.
4	Falta de experiencia con *Angular*	0.2	El desarrollador tarda más tiempo del esperado en realizar la parte de *Angular* debido a su falta de experiencia, a pesar de que ya se ha tenido en cuenta que se tardará más de lo normal debido a este problema.	7 días.
5	Falta de experiencia con nodejs y dockerode	0.25	El desarrollador tiene problemas con node.js o con la librería dockerode, utilizada en todo el proyecto, provocando retrasos adicionales al desarrollo de la aplicación.	7 días.
6	Problemas con el equipo informático	0.1	Si hay algún problema técnico con los equipos informáticos, pueden producirse retrasos por imposibilitar el avance del proyecto o incluso por la pérdida de datos.	3 días.
7	Fallos en la planificación	0.5	Algunos aspectos de la planificación son difíciles de estimar y puede ocurrir que no se cumplan todos los tiempos planificados.	15 días.

Tabla 2.3: Tabla de riesgos del desarrollo del proyecto.

ID	Riesgo	Exposición al riesgo	Plan de acción
1	Suspender PGP en la convocatoria de fin de carrera.	6	Dedicar muchas más horas a preparar la asignatura, para no sufrir más retrasos en el proyecto.
2	Suspender PGP en la convocatoria extraordinaria.	3.05	Dedicar todo el tiempo a la asignatura, pudiendo retrasar el proyecto pero asegurando aprobar la asignatura para no volver a sufrir un retraso.
3	Caer enfermo	0.2	Intentar recuperar las horas.
4	Falta de experiencia con *Angular*	1.4	Utilizar todas las páginas de documentación y buscar ejemplos de las partes concretas que estén causando especial dificultad.
5	Falta de experiencia con node.js y dockerode	1.75	Utilizar documentación oficial de node.js y dockerode, y apoyarse en la documentación oficial de docker-engine, que es en lo que está basado dockerode y puede ayudar a comprender partes que con dockerode por si solas sean muy complicadas.
6	Problemas con el equipo informático	0.3	Intentar utilizar otro equipo al más mínimo problema y tener subido en el repositorio todo el código de forma actualizada para no depender nunca de un único ordenador.
7	Fallos en la planificación	7.5	Realizar horas extra en horarios no planificados para intentar realizar esas horas sin que retrasen al proyecto.

Tabla 2.4: Tabla de acción riesgos del proyecto.

2.4 Estimación de costes

En este apartado se detalla la estimación de los costes de realizar este proyecto. Se indican todo tipo de costes, incluyendo el salario del desarrollador, de los equipos y programas de pago.

2.4.1 Salario del desarrollador

Al tratarse de un TFG (trabajo de fin de grado), el desarrollador no está contratado por la Universidad de Valladolid y no ha percibido salario. Sin embargo, para realizar los cálculos de los costes se va a tener cuenta el salario de un desarrollador junior. En el momento de realizar este TFG el salario de un desarrollador junior sin experiencia ronda los 20000€ euros brutos anuales aproximadamente [34]. Si se realiza el cálculo al precio por hora da como resultado algo más de 10 euros la hora, por lo que para realizar unos cálculos más sencillos se va a redondear esta cifra a 10 euros la hora exactos. Si se multiplica 335x10 se obtiene el resultado de 3350€ el coste del desarrollador para llevar a cabo este proyecto.

2.4.2 Costes del espacio de trabajo

Al haberse realizado el proyecto mediante teletrabajo, el espacio de trabajo ha sido la casa del desarrollador. Esto implica que no se puede obtener el coste real de este espacio. Se va a tener en cuenta el coste de un alquiler medio en Valladolid de una habitación con los gastos incluidos de unos 350€ al mes para poder realizar el cálculo. Teniendo en cuenta los siete meses que se tarda en desarrollar este proyecto (sin tener en cuenta las vacaciones y contando el tiempo de retraso sufrido), el precio total es de 2450€.

2.4.3 Coste de los equipos utilizados

Se va a tener en cuenta el precio de compra del ordenador portátil en relación con la vida útil del mismo para poder realizar el cálculo de los costes en este proyecto. El equipo utilizado tuvo un coste de 750€, con una vida útil de seis años, nos da un coste de 125€ al año, que equivalen a 10.50€ al mes. Este último dato es el que se va a tener en cuenta para el coste de desarrollar el proyecto. Como el tiempo que se ha tardado en desarrollar el proyecto han sido siete meses, el resultado es 73.50€.

2.4.4 Coste de los servidores

Hay que recordar que para este proyecto se han utilizado dos servidores idénticos con las características ya descritas anteriormente. Al tratarse de máquinas virtuales que proporciona la Universidad de Valladolid, no se tiene un coste real de los mismos, por lo que al igual que se ha hecho en los apartados anteriores, se va a tener en cuenta el coste de servidores equivalentes en la nube. Entre los servidores

virtuales ofrecidos por la compañía OVH [35] el que se llama *Essential* es el más parecido a las máquinas de la universidad y también es el más adecuado a los requisitos del proyecto. Este servidor tiene un precio de 10€ al mes sin IVA, que si se lo sumamos son 12.10€. Debemos multiplicar por 2 este precio ya que disponemos de dos máquinas virtuales, por lo que el precio mensual sería de 24,20€. Si se multiplica el resultado anterior por los siete meses de duración del proyecto se obtienen 169.40€ totales.

2.4.5 Coste de los servicios

A excepción de un programa, todos los servicios y programas utilizados son gratuitos, por lo que no tienen ningún coste en el proyecto. El único programa de pago de los utilizados es el Astah, que en su versión *profesional* tiene un coste de 7.50€ mensuales. Una vez multiplicado el precio mensual por siete se obtiene que el resultado final es de 52.50€.

2.4.6 Coste totales

En la tabla 2.5 se incluyen todos los costes explicados en los apartados anteriores. El total ha ascendido a 6095.40€.

Elemento	**Precio**
Salario del desarrollador	3350€
Costes del espacio de trabajo	2450€
Costes de los equipos utilizados	73.50€
Coste de los 2 servidores	169.40€
Coste de los servicios	52.50€
Total	**6095.40€**

Tabla 2.5: Costes totales del proyecto

2.5 Planificación final

El desarrollo real final del proyecto no ha sido exactamente el planificado inicialmente. El principal problema han sido los riesgos *ID-4*, *ID-5* y *ID-7* (ver tabla 2.3). Adicionalmente ha ocurrido que tras los retrasos se ha acercado la fecha del examen de la asignatura pendiente por el alumno y esto ha provocado que haya sido necesario parar la realización del proyecto hasta después del examen. Teniendo todo esto en cuenta se ha sufrido un retraso de un mes y una semana.

En las primeras etapas del desarrollo del *frontend*, durante el principio de la tercera iteración, se produjo el primer riesgo (*ID-4*), que produjo el retraso esperado (siete días). Este tiempo se utilizó para familiarizarse con *Angular*, herramienta sobre la cual apenas se tenían conocimientos. Una vez se realizó la pantalla principal de la aplicación (donde se pueden ver todos los contenedores), ya se

tenía suficiente conocimiento en la herramienta para que no volviera a producirse otro retraso por este motivo.

Una semanas después, se produjo el segundo riesgo (*ID-5*), cuando se estaba realizando la creación de contenedores de *docker*, que ha sido la parte más complicada de la aplicación, ya que hay que configurar un gran número de opciones. Como se había planteado en el riesgo, se combinó que no se tenía un exceso de experiencia en *node.js* y que la dependencia *dockerode* no se conocía en profundidad. El tener que entender la documentación de *docker*, que no es igual que la sintaxis de los comandos (que es lo que se conocía en ese momento), y la adaptación que *dockerode* hace de la misma, ha hecho que se produjera el riesgo, que ha tenido un retraso de siete días, como se había planificado.

El último riesgo que se ha producido, el *ID-7*. Se empezó a detectar de manera simultánea al primero, y es que la planificación no se estaba cumpliendo porque había desconocimiento de ciertas tecnologías y también porque el desarrollador en algunas ocasiones estaba disponiendo de menos tiempo del esperado, debido a que inició prácticas extracurriculares, las cuales compaginaba con el desarrollo de este proyecto. Para que este problema no se acentuara en exceso se han realizado horas extras en horarios no planificados inicialmente.

Teniendo todo esto en cuenta, las horas totales del proyecto han sido 375, superando las 315 esperadas, que ya eran ligeramente superiores a las 300 mínimas pensadas para este proyecto. Además se han llevado a cabo 16 reuniones con los tutores del TFG. A continuación se incluye un gráfico con las horas empleadas finalmente y la comparación a las planificadas:

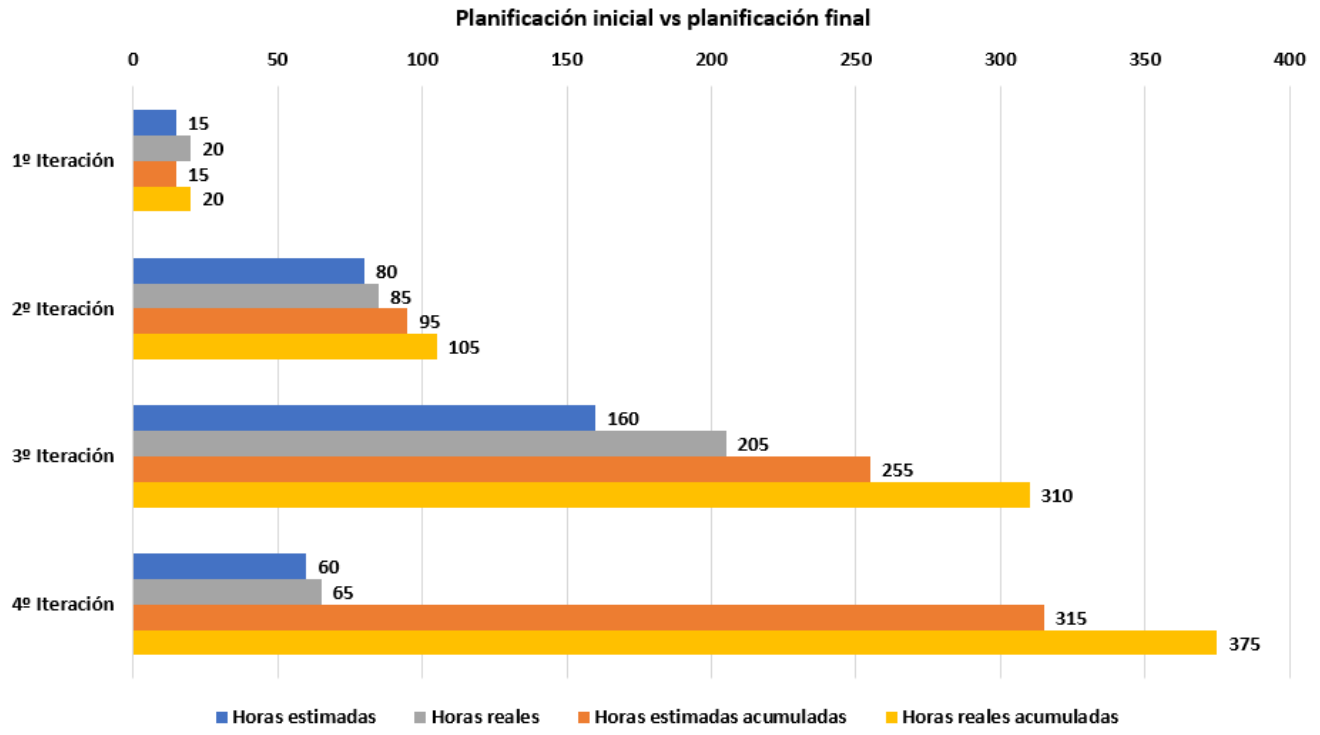


Figura 2.2: Gráfico comparativo de la planificación inicial y la planificación final.

Capítulo 3

Descripción de las iteraciones

Se van a detallar todas las iteraciones realizadas en este proyecto, indicando algunos cambios que se han producido en algunos requisitos y funcionalidades que inicialmente estaban previstos para el proyecto.

3.1 Iteración 1

En esta primera iteración, tras reunirse el desarrollador con los tutores de este proyecto y dejando claros los principales objetivos del mismo, se realiza un trabajo de investigación general en el que se van a averiguar que alternativas se tiene para poder cumplir con las funcionalidades básicas y conocer que opciones ofrecen las distintas tecnologías que sirven para cumplir estos requisitos. A continuación se va a exponer en detalle todo el trabajo realizado.

3.1.1 Trabajo realizado

En esta iteración se han investigado las tecnologías existentes para poder tomar la decisión de cuales de ellas son las que se van a utilizar en este proyecto por tener las funcionalidades esperadas y ser sencillas de utilizar y factibles para este proyecto. En la parte de *frontend* se valoraban tecnologías como *Angular* o el uso de alguna tecnología híbrida que incluye *node.js*, para combinar toda la lógica de la aplicación web. También había que decidir qué tecnología utilizar en el *backend*, ya que si se separaba del *frontend* se podía utilizar cualquiera que pueda funcionar como API REST.

Se ha decidido inicialmente utilizar *node.js* para el desarrollo del *backend*, ya que es una tecnología bastante cómoda para lo que queremos realizar. Para el *frontend* se ha decidido utilizar el *framework* de *node.js Angular*, ya que es de las tecnologías más potentes que existen en la actualidad y con mayores opciones y facilidades.

Para hacer la gestión de los contenedores, había que investigar que opciones proporciona el propio *docker* para ello. Se ha encontrado que la herramienta correspondiente es *docker-engine*, que es

una *API* incluida en *docker* que tiene todos los métodos para realizar todas las operaciones sobre contenedores. Para utilizar esta herramienta con múltiples lenguajes de programación, existen diversos proyectos externos al propio *docker* con código abierto que lo que hacen es implementar *docker-engine* desde distintos lenguajes. En este caso, la herramienta que implementa *docker* con *node.js* se llama *dockerode*.

Sin embargo, existe un problema, y es que para para gestionar servicios avanzados, se suele utilizar herramientas como *docker-compose* o *kubernetes*, que son orquestadores de contenedores. Por este motivo, se necesita adicionalmente una gestión avanzada que se ha decidido que va a realizar *docker-compose* por los motivos ya indicados en el primer apartado de este informe.

Teniendo en cuenta estos dos puntos anteriores, se va a aglutinar todo esto en el servicio de *node.js*, que implementa *dockerode* para la gestión general de contenedores y *docker-compose* para la creación de contenedores avanzados. Esta última herramienta solo permite la gestión de contenedores de forma local, por lo que se ha decidido que nunca se va a realizar una comunicación remota directamente con *docker*, sino que se hará a la *API REST*, la cual será la que mediante *dockerode* realice todas las gestiones necesarias de forma local.

A parte de esto, es necesario un servicio *backend* que gestione la lista de servidores con *docker* que existan en el sistema, y también, un servidor de autenticación en el que los usuarios puedan iniciar sesión, ya que la web no va a estar abierta para cualquier persona. Se ha decidido juntar estos dos servicios en un único servicio web que también va a estar desarrollado en *node.js*.

Por tanto vamos a tener el *frontend* en *Angular*, el *backend* de autenticación y gestión de servidores en un *node.js*, y finalmente vamos a tener en todos los servidores en los que queramos gestionar *docker* el *backend* que contiene el *dockerode*, que también está escrito en *node.js*.

Una vez hecho todo este trabajo, se tienen listas todas las tecnologías que se van a utilizar, pero va a ser necesario investigarlas todas más a fondo y de una forma más específica para saber de forma concreta todas las opciones que vamos a desarrollar con esta plataforma. Una vez se haya hecho esto se podrán definir los requisitos concretos del sistema. Esto se realiza en la siguiente iteración, ya que en esta primera el objetivo era fijar las bases y esto ya se ha realizado.

3.2 Iteración 2

3.2.1 Aprendizaje de la plataforma

En esta segunda iteración se ha realizado el aprendizaje de toda la plataforma de *dockerode*, que implementa todo el sistema proporcionado por *docker-engine*. Revisando la documentación oficial [36] se han aprendido todas las opciones de ésta plataforma y qué consultas hay que realizar exactamente para realizar las operaciones básicas, como iniciar o detener un contenedor. Se ha visto que existe la opción de crear un contenedor basándose en una imagen ya existente, borrar contenedores, etc.

Tras ver toda esta documentación se ha vuelto a reafirmar que para crear contenedores avanzados no se tienen herramientas suficientemente potentes para todo lo que se quiere realizar. Esto no es un problema porque como ya se había pensado anteriormente, se puede utilizar *docker-compose* para cubrir esta funcionalidad, y de esta forma la aplicación combinará las funcionalidades de estas distintas herramientas.

Para la gestión de *docker-compose* se van a utilizar directamente los comandos que existen, ya que los archivos se subirán ya creados y simplemente habrá que dar las ordenes de *build* y *up*. Una vez ya están creados los contenedores, se pueden utilizar las operaciones de *docker-engine* para pararlos, volver a iniciarlos, borrarlos, etc. Por tanto, no es necesario el uso de *docker-compose* para ninguna funcionalidad más.

3.2.2 Requisitos de la aplicación

Cuando ya se dispone del análisis de las aplicaciones a utilizar, se ha procedido a elaborar la lista de requisitos funcionales y no funcionales que tiene que tener la aplicación. Este punto es muy importante porque una vez se tengan listos los requisitos se podrá empezar a hacer los prototipos de la aplicación, que se utilizarán posteriormente para iniciar el desarrollo de la aplicación. En la tabla 3.1 se pueden ver todos los requisitos funcionales.

En la siguiente iteración, si es necesario, se puede modificar ligeramente alguno de estos requisitos o incluso añadir alguno más en función de como se avance en todos los puntos. A continuación, en la tabla 3.2 se pueden ver todos los requisitos no funcionales.

Teniendo todos estos requisitos especificados, se puede empezar a plantear una idea de como va a ser la herramienta web que va a gestionar todo ello. En este momento se empieza a pensar en los distintos prototipos de como va a ser la interfaz de la aplicación en relación con todas las funcionalidades que debe tener la misma.

ID	Nombre	Descripción	Prioridad
RF-01	Identificación de usuarios.	El sistema debe permitir que los usuarios inicien sesión para identificarse.	Alta
RF-02	Registro de usuarios.	El sistema debe permitir que los usuarios autorizados registren nuevos usuarios para que puedan acceder al sistema.	Alta
RF-03	Cerrar la sesión de usuario.	El sistema debe permitir que los usuarios que tengan iniciada sesión la cierren.	Alta
RF-04	Iniciar contenedor.	El sistema debe permitir iniciar un contenedor que esté creado pero esté detenido.	Alta
RF-05	Detener contenedor.	El sistema debe permitir detener un contenedor que esté iniciado.	Alta
RF-06	Crear contenedor mediante imagen.	El sistema debe permitir crear un contenedor basándose en una imagen que ya esté creada en el contenedor.	Media
RF-07	Crear contenedor avanzado.	El sistema debe permitir crear un contenedor avanzado subiendo todos los ficheros correspondientes para su funcionamiento.	Alta
RF-08	Borrar contenedor.	El sistema debe permitir borrar contenedores.	Alta
RF-09	Modificar contenedor.	El sistema debe permitir modificar contenedores que ya están creados.	Media
RF-10	Agregar servidores.	El sistema debe permitir agregar servidores sobre los que se van a gestionar los contenedores docker.	Alta
RF-11	Borrar servidores.	El sistema debe permitir borrar servidores sobre los que se van a gestionar los contenedores docker.	Alta

Tabla 3.1: Requisitos funcionales del sistema.

ID	Nombre	Descripción	Prioridad
RNF-01	Plataforma web.	El sistema debe utilizarse desde un navegador web, para que sea multiplataforma y así su acceso se pueda realizar desde cualquier dispositivo, utilizando navegadores como *Firefox*, *Chrome* o *Edge*, todos ellos en la última versión (noviembre 2021) tanto en *windows 10* como *macOS 11* o *ubuntu 20.04*.	Alta
RNF-02	Registro de servidores y usuarios.	El sistema debe guardar los datos de los servidores y usuarios en una base de datos de tipo MongoDB.	Alta
RNF-03	Archivos comprimidos	Se utilizarán formatos de fichero comprimido *.tar* y *.tar.gz* para la subida/bajada de ficheros del servidor.	Media
RNF-04	Facilidad de uso.	El sistema debe ser sencillo de aprender a utilizar para cualquier persona que tenga un conocimiento básico de *docker*.	Media
RNF-05	Uso de *jsonwebtoken* (*JWT*) [37] para comunicaciones.	El sistema utilizará el estándar *JWT* para la seguridad de las comunicaciones de la *API REST*.	Alta
RNF-06	Compatibilidad con *UTF-8* [38]	El sistema utilizará el formato *UTF-8* para codificar los caracteres.	Alta

Tabla 3.2: Requisitos no funcionales del sistema.

3.2.3 Prototipos de la aplicación

En este punto se adjuntan todos los prototipos realizados de la aplicación, los cuales están pensados, como ya se ha indicado, con todos los requisitos especificados en el apartado anterior:

- **Pantalla principal:** En esta primera pantalla (Figura 3.1), la idea es que se muestren en la parte central todos los contenedores que haya en la máquina seleccionada, separando los que están activos de los que no. Debajo de los contenedores hay un botón para crear nuevos contenedores que debe llevar a otra pantalla. En la parte izquierda debe haber un menú que muestre los servidores sobre los que se están gestionando contenedores *docker*, y también deben estar los accesos para modificar esta lista de servidores.

  En esta misma pantalla, cuando se hace click sobre uno de los contenedores, debe desplegarse el detalle del contenedor elegido, como se muestra en la figura 3.2, y deben aparecer los subcontenedores que componen el contenedor elegido, y también deben mostrarse los botones con los cuales se van a realizar las gestiones de los contenedores, tales como iniciar, parar, editar o borrar.

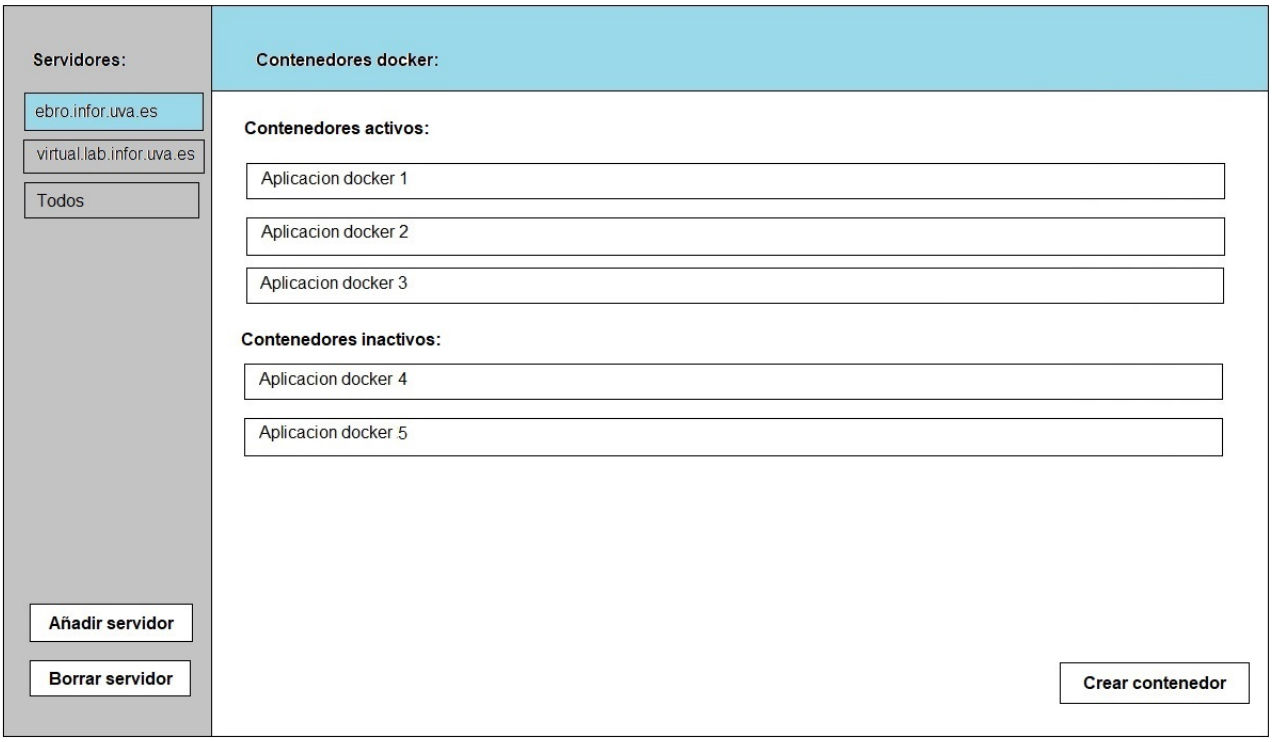


Figura 3.1: Pantalla principal de la aplicación.

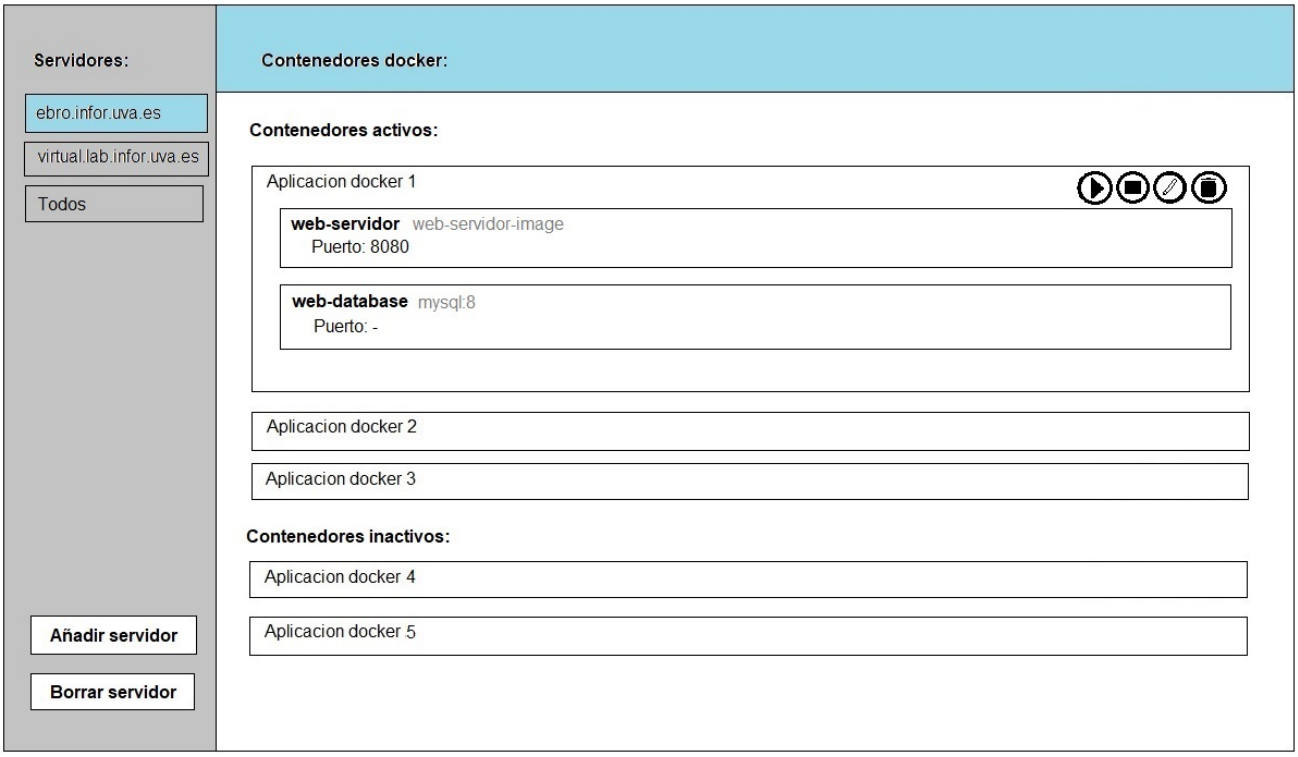


Figura 3.2: Pantalla principal de la aplicación con contenedor desplegado.

- **Crear contenedor sencillo:** En esta pantalla 3.3, se puede ver el diseño para crear un contenedor de forma sencilla, es decir, a partir de una imagen ya existente en la máquina. Como se puede leer, simplemente habría que indicar un nombre para el contenedor, que lo identifique de los demás, el nombre de la imagen a utilizar, y también si se desea, las opciones de puertos y volúmenes deseadas.

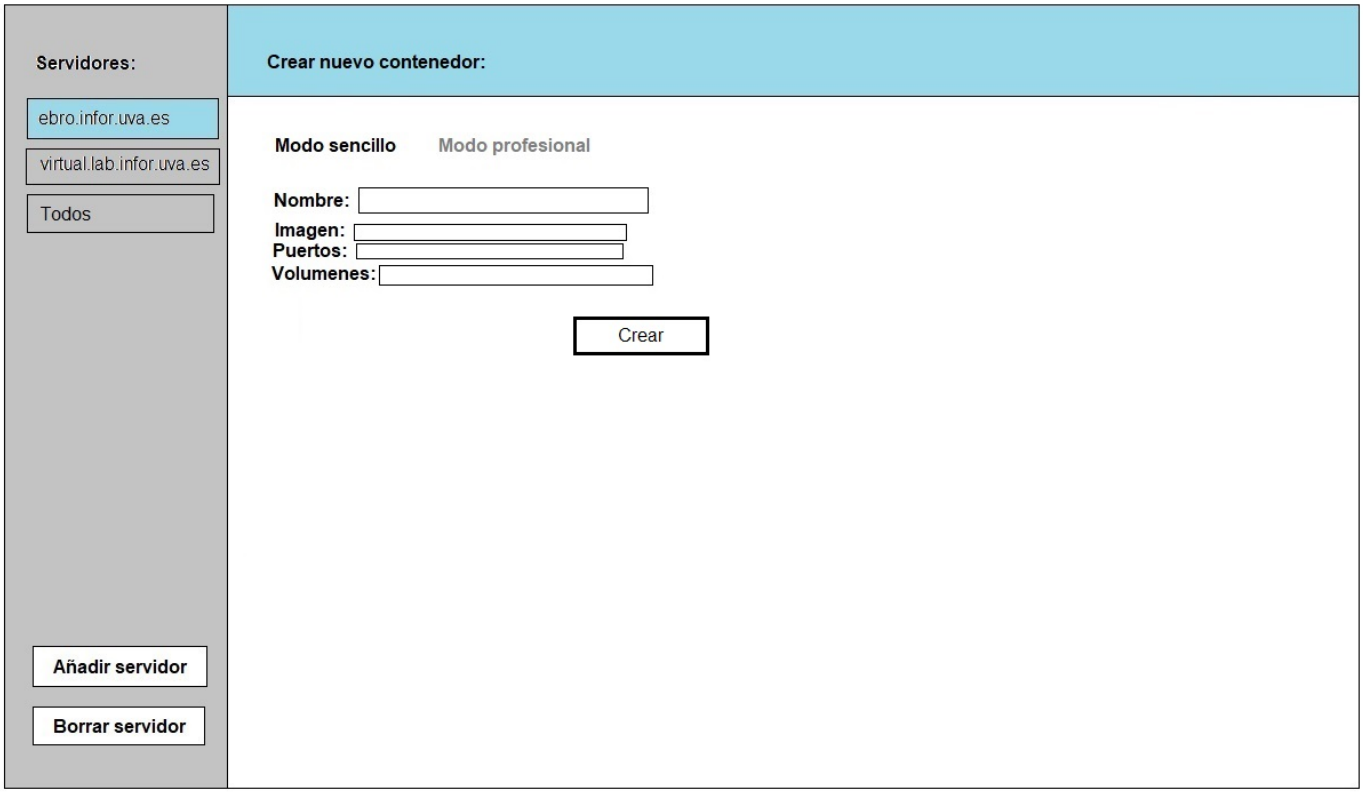


Figura 3.3: Pantalla para crear un contenedor a partir de una imagen.

- **Crear contenedor avanzado:** En esta última pantalla 3.4, se puede ver el diseño para crear un contenedor de forma avanzada, la cual consiste en poner un nombre y subir los ficheros correspondientes al servicio que queremos crear. Este fichero debe contener un *docker-compose.yml* que sea el que posteriormente se lea para crear el contenedor con todos los detalles ahí indicados.

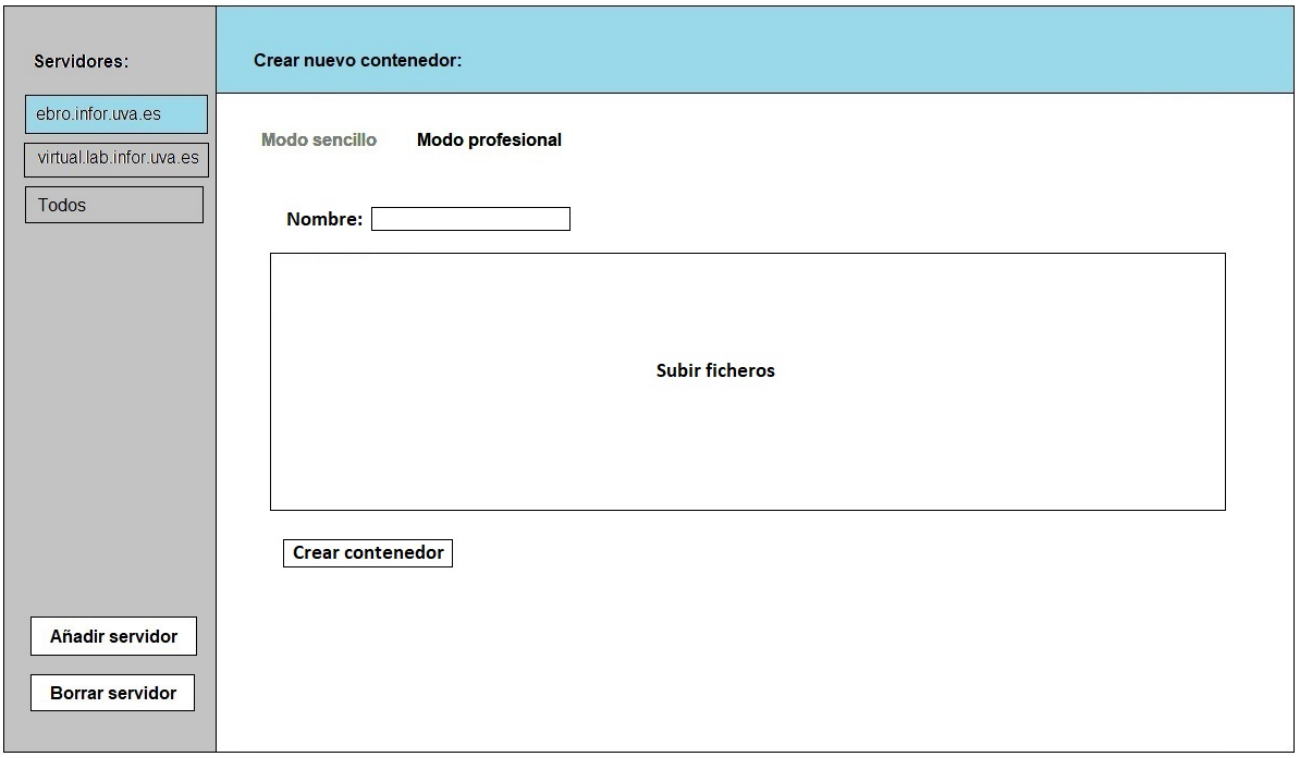


Figura 3.4: Pantalla para crear un contenedor de forma avanzada.

No se ha realizado prototipo de alguna pantalla como la de inicio de sesión, por ejemplo, ya que es algo que se ha visto en prácticamente todos los sitios y no tiene ninguna relevancia hacerlo al ser bien conocido.

Una vez realizado todo lo anterior, se inicia la siguiente iteración, que consiste en realizar el desarrollo de todos los puntos que ya se han descrito.

3.3 Iteración 3

En esta iteración se va a desarrollar toda la aplicación. Se va a ir desarrollando de forma paralela el *frontend* y el *backend* de cada parte, es decir, que para hacer la pantalla principal, donde se muestran todos los contenedores, se hace la función del *backend* que devuelve toda la información necesaria y después se realiza en *Angular* esa pantalla, después las operaciones de parar e iniciar contenedor en el *backend* y seguidamente se implementan los botones que se conectan a esta operación en el *frontend*, etc.

3.3.1 Desarrollo de la pantalla principal

La pantalla principal es la primera que se ha realizado. Se ha iniciado primero el *backend* en *node.js*, donde ha sido necesario crear la aplicación y montar una *API REST*. Una vez hecho esto, solo se ha añadido una operación, que sirve para obtener la información de todos los contenedores. Con este punto listo, se ha iniciado la aplicación *Angular*. Se ha diseñado el diseño que va a seguir la web en todas sus rutas, que es el menú lateral de servidores (inicialmente sin ninguna funcionalidad hasta que se implemente más adelante) y la barra superior, dónde entre otras cosas se situará el botón de cerrar sección. Una vez completado eso, se ha creado la parte central, que es donde se muestran los contenedores. Esto se puede ver en la figura 3.5.

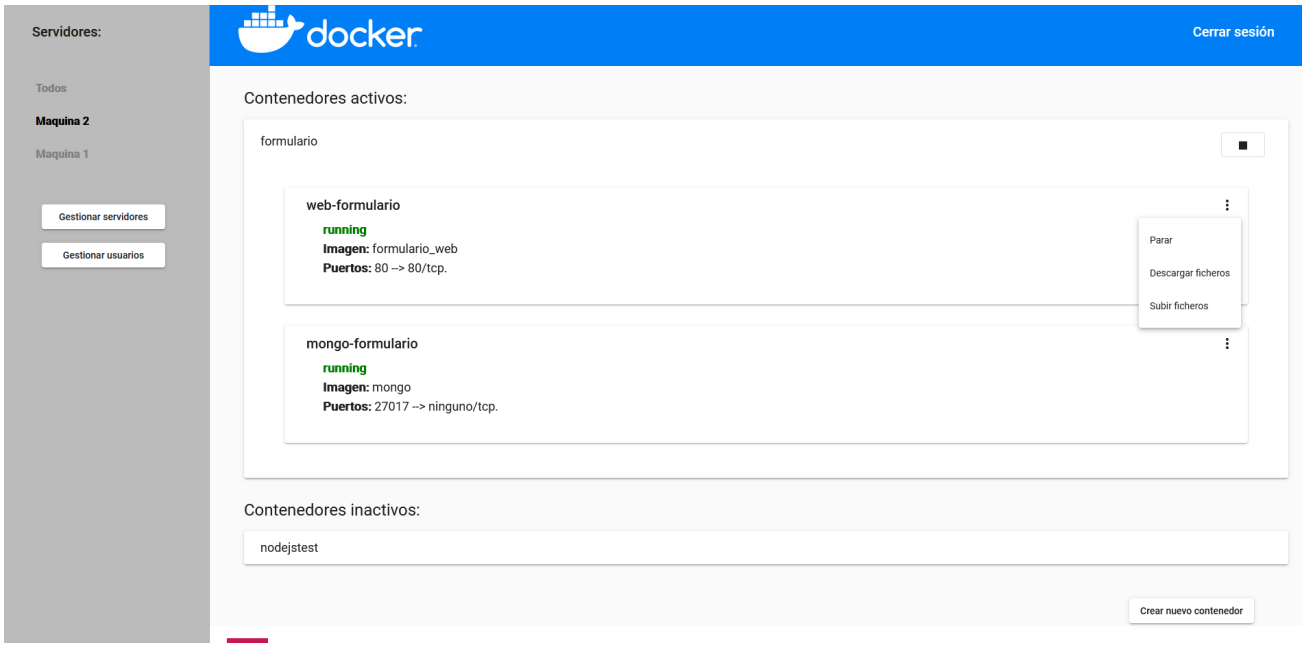


Figura 3.5: Pantalla principal de la aplicación.

Inicialmente se planificó que esta parte llevaría más tiempo del habitual porque no había ningún dominio sobre esta plataforma. Esto se ha cumplido, lo cual hace que el proyecto avance bastante lento en este momento de la iteración.

Con los contenedores mostrados en pantalla, se han añadido las operaciones de start, stop y borrar

para los contenedores, lo que ha implicado tres llamadas más en el *backend*, y simplemente unos botones en los contenedores tal y como se había mostrado en los prototipos.

3.3.2 Desarrollo de la pantalla de crear contenedores sencillos

A continuación se ha desarrollado la pantalla para crear un contenedor utilizando una imagen ya creada como base. Para ello han sido necesarias dos llamadas en el *backend*, una para mostrar la lista de imágenes disponibles y la otra para crear el contenedor en sí. Esta operación no ha tenido especial dificultad en lo que se refiere a crear el contenedor, y a priori parecía que se iba a completar todo en menos tiempo del previsto. Sin embargo, la configuración de puertos y volúmenes, ha dado bastantes problemas en el desarrollo. Posteriormente, para realizar la interfaz más amigable, también se ha empleado algo más tiempo del esperado para que la configuración de puertos y volúmenes sea lo más sencilla posible. En la figura 3.6 se puede ver el resultado de esta iteración.

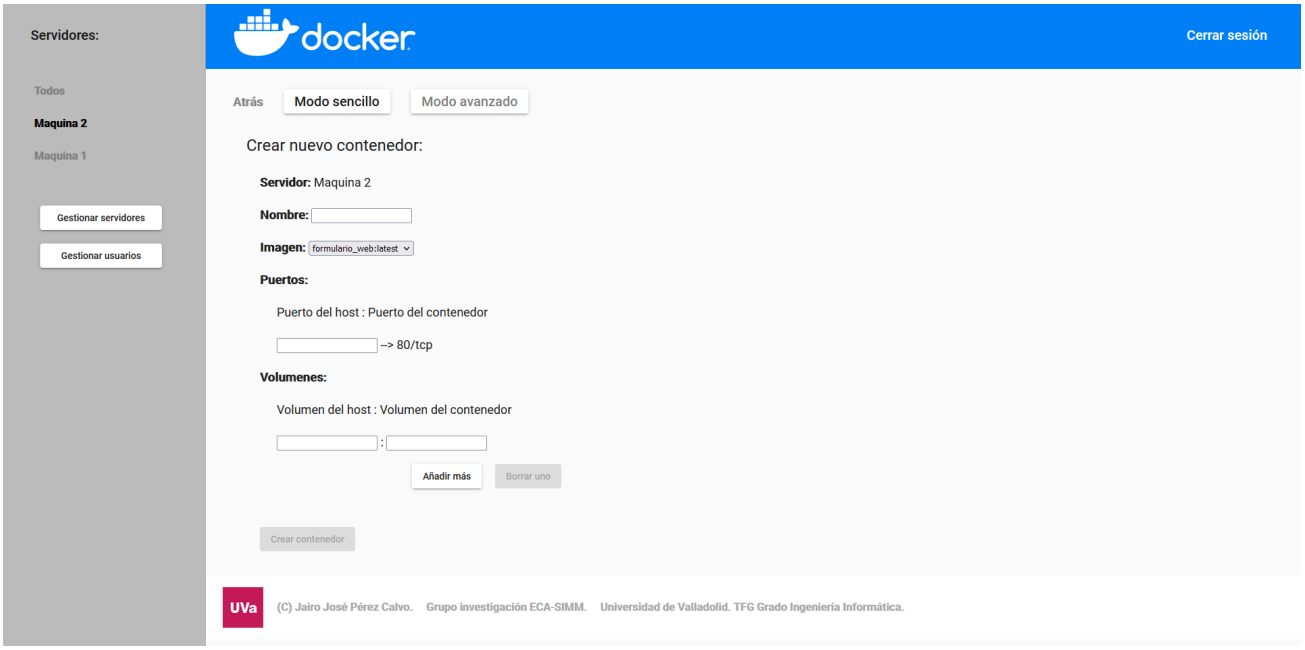


Figura 3.6: Pantalla para crear contenedores sencillos.

Una vez realizado esto, se ha iniciado el despliegue en las máquinas virtuales para tener la posibilidad de ir probando todo en el entorno en el que está pensado que se ejecute, ya que hasta ahora se había probado en el ordenador de desarrollo, y este tiene diferencias con respecto a estas máquinas que pueden provocar que no se tenga el mismo resultado en un ordenador que en otro. Al realizar esto ha habido que solucionar un problema que impide la comunicación entre el *frontend* y el *backend*. Este problema tiene que ver con el *CORS* y se explica en el apéndice A.

3.3.3 Desarrollo de la pantalla de crear contenedores avanzados

Esta es la única operación que no utiliza la API de *dockerode* para completarse. Al usar *docker-compose* con comandos para la ejecución de esta operación, la parte más complicada y que más tiempo ha llevado ha sido la gestión del fichero que contiene los datos para crear el contenedor. Finalmente el funcionamiento de esta operación es con un fichero *zip*, que se va a subir al servidor y ahí se va a descomprimir para poder iniciar el contenedor. Se puede observar en la figura 3.7 el resultado de todo esto.

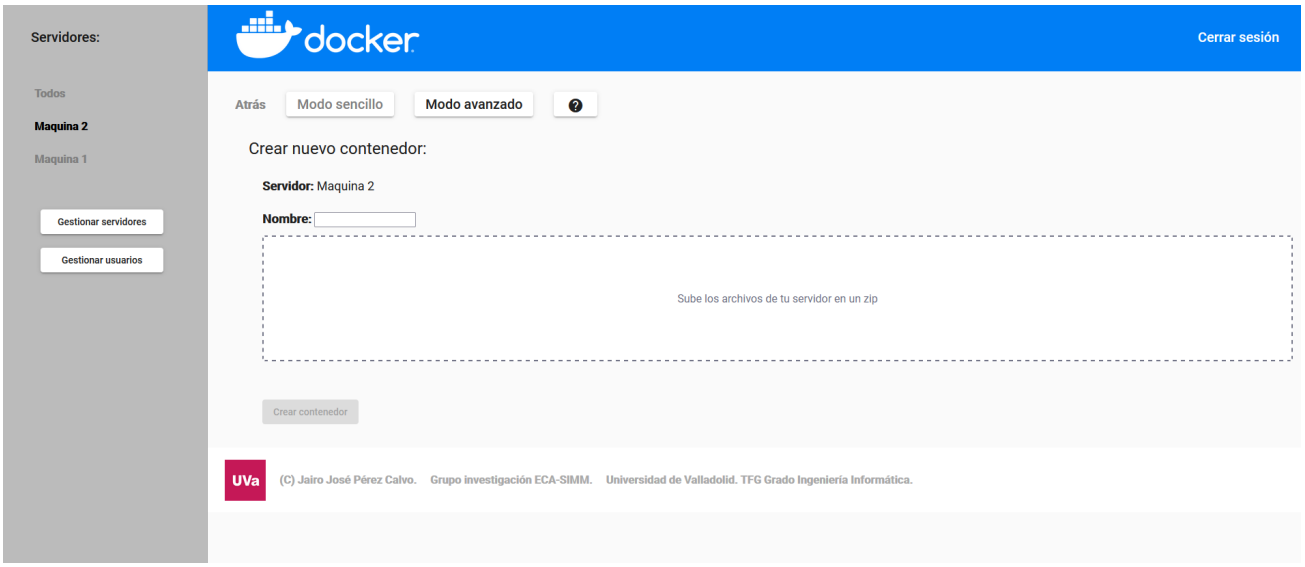


Figura 3.7: Pantalla para crear contenedores avanzados.

Esta operación es la que puede tener como resultado varios contenedores dentro de un mismo proyecto, que es para la que se ha creado el diseño visto en el apartado anterior, en el cual se aglutinan todos los contenedores que pertenecen al proyecto en un mismo desplegable, y se pueden iniciar, detener o borrar juntos.

3.3.4 Desarrollo del servidor de autenticación y gestión de servidores

Con todas las operaciones principales ya funcionando, se ha iniciado el desarrollo del servidor de autenticación. Para el uso del login se ha utilizado el módulo de *nodejs* llamado *jsonwebtoken* [28]. Con esto lo que se va a hacer es generar un *token* [37] con el que identificar al usuario en todas las llamadas que haga para poder verificar que tiene permisos para realizar todas las operaciones correspondientes. Para guardar los datos de los usuarios se ha utilizado una tabla *MongoDB*.

Posteriormente, se ha utilizado también una base de datos *MongoDB* para almacenar los datos de los servidores sobre los que se gestionan contenedores. En este caso ha sido relativamente sencillo hacerlo ya que no había ningún aspecto sobre el que hubiera que investigar para poder completarlo exitosamente. A continuación puede verse la pantalla de inicio de sesión en la figura 3.8.

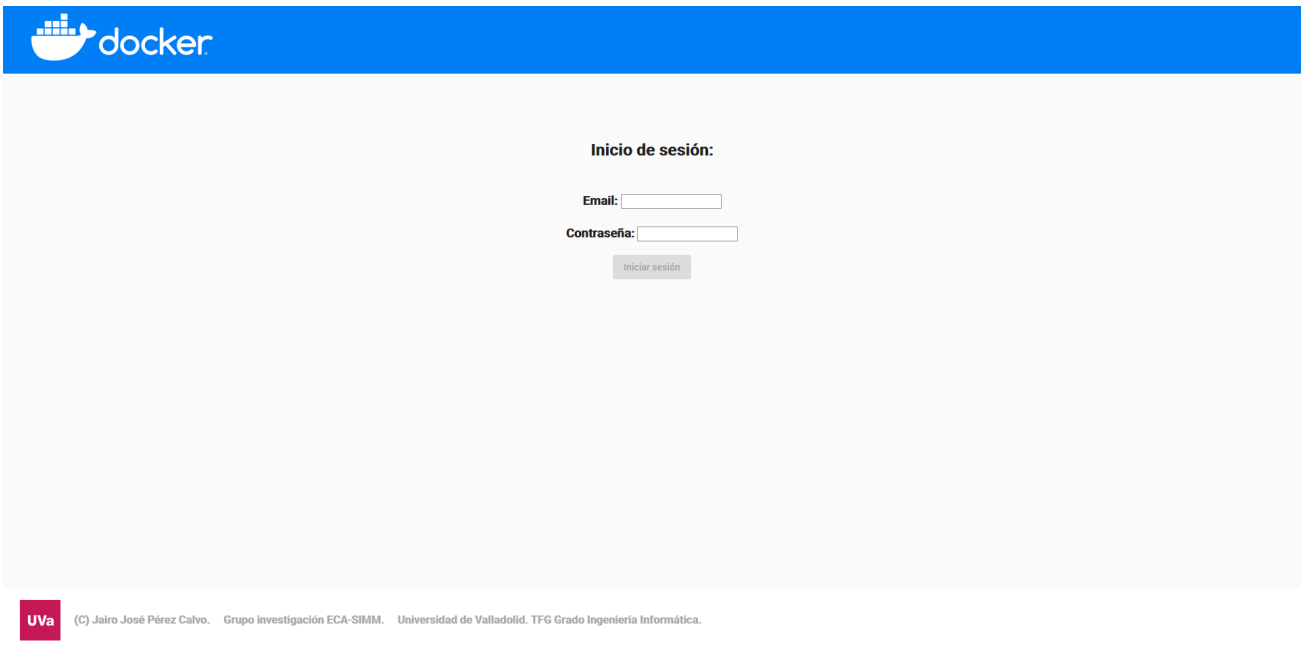


Figura 3.8: Pantalla para crear contenedores avanzados.

Con todo esto completado, se tienen casi todos los requisitos iniciales ya cumplidos. En este momento se han producido las vacaciones del desarrollador, con la idea de acabar de completar todos los requisitos que faltan después.

3.3.5 Desarrollo de las operaciones restantes

Tras la vuelta de vacaciones, se ha analizado que requisitos faltaba por cumplir de los especificados en la iteración anterior. El principal requisito que faltaba por cumplir era el de modificar contenedores.

Tras realizar algunas mejoras menores estéticas, revisando la documentación de *docker-engine* se ha llegado a la conclusión de que la mejor solución para cumplir este requisito es permitir descargar y subir ficheros dentro de un contenedor, de esta forma, se puede actualizar el contenido de cualquier contenedor en cualquier momento, y también se pueden descargar ficheros de su interior si así se desea, por ejemplo, para crear una copia de seguridad de ciertos datos.

Lo último que se ha realizado en esta iteración ha sido utilizar *postman* para crear peticiones de prueba para todas las consultas existentes en los dos servidores escritos con *node.js*. Inicialmente se han creado casos de prueba válidos, y en la última iteración se realizarán más pruebas utilizando también casos de error. En la figura 3.9 puede verse un ejemplo de una llamada.

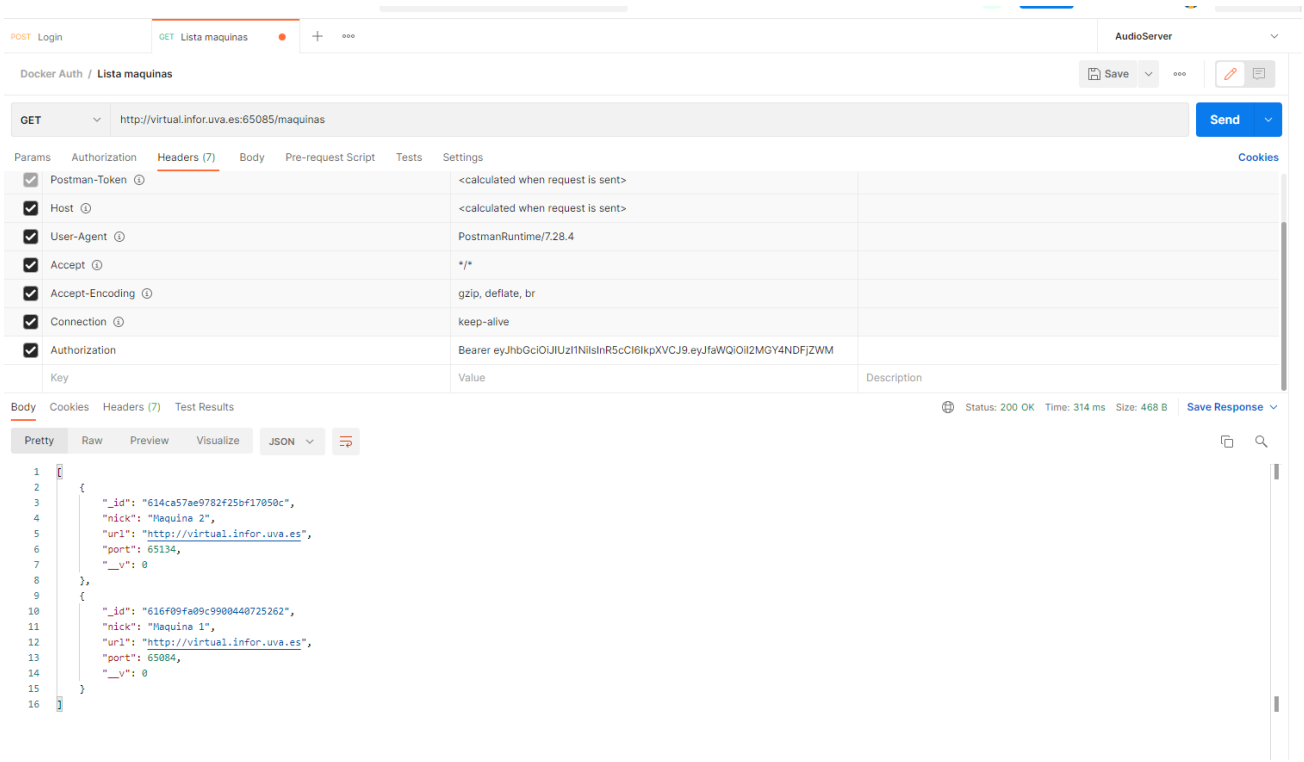


Figura 3.9: Llamada de prueba realizada desde postman.

En este momento, a falta de pulir los detalles, toda la funcionalidad de la aplicación se puede dar por completada.

3.4 Iteración 4

Esta última iteración está enfocada a solventar todos los problemas que puedan surgir en la aplicación, a realizar distintas pruebas, y a redactar todos los detalles de la aplicación.

3.4.1 Elaboración de la memoria

La mayor parte de tiempo de esta iteración se ha dedicado a redactar este informe, que es lo que se ha llevado más de la mitad del tiempo. Se ha realizado en *overleaf* [30], que utiliza el lenguaje *LaTeX* [39]. Para su elaboración se han utilizado todos los documentos, diagramas y esquemas que se han hecho conforme se ha ido realizando el proyecto.

3.4.2 Realización de pruebas

Se han completado todas las pruebas con *postman* creando llamadas con casos de error, y revisando que no se hubiera olvidado ningún caso de los existentes. Una vez hecho esto se han realizado de forma profunda pruebas con la parte del *frontend* para terminar de verificar el correcto funcionamiento de la aplicación. En el capítulo 5 se pueden ver todos los detalles de lo realizado.

3.4.3 Retoques finales

Después de revisar toda la aplicación, ha sido necesario hacer algunos retoques finales para mejorar algunos aspectos principalmente visuales, ya que toda la aplicación se ha probado ya en profundidad. En este momento ya se ha dado el proyecto por finalizado.

3.4.4 Productos finales

Al terminar el TFG se han obtenido tres aplicaciones: una aplicación *Angular*, cuya compilación produce ficheros *html* y *js*, que navegadores como *firefox* pueden ejecutar. Está subida en una máquina virtual de la escuela para poder acceder a ello; una aplicación *node.js* para gestionar los usuarios y los servidores, que también se encuentra desplegada en una máquina de la escuela, y la aplicación que gestiona los contenedores *docker*, que también es una aplicación *node.js*, y está subida en este caso en dos máquinas de la escuela en las que gestiona diferentes contenedores *docker*. Todo el código fuente de estas tres aplicaciones se encuentra en un repositorio privados del grupo ECA-SIMM. El último producto de este TFG es este documento, en el que se explica todo lo relevante sobre el proyecto.

Capítulo 4

Estado final de la aplicación

A lo largo de este capítulo se van a tratar los temas relacionados con el análisis y diseño de la aplicación, exponiendo tanto diagramas como patrones o técnicas utilizadas.

4.1 Diagramas de análisis

Se analizan distintos aspectos de la aplicación, tales como los flujos de la misma, sus historias de usuario, o distintos diagramas.

4.1.1 Historias de usuario

Esta sección está orientada a describir todas las historias de usuario del sistema. Una historia de usuario [40] es una explicación de una función de un software desde la perspectiva del usuario que lo va a utilizar. A continuación se analizan todas las historias de usuario de la aplicación, utilizando los siguientes campos en las tablas:

- **ID**: El identificador de la historia de usuario.
- **Nombre**: El nombre de la historia.
- **Prioridad**: El nivel de prioridad de la historia en comparación al resto.
- **Riesgo**: El nivel de riesgo que supone un fallo en dicha historia.
- **Descripción**: Una descripción de lo que realiza la historia.
- **Prerrequisitos**: Los prerrequisitos de la historia, si los hay.
- **Validación**: Condiciones necesarias para que la historia se de por completada.

ID	**HU01**
Nombre	Iniciar sesión en el sistema.
Prioridad	Alta.
Riesgo	Alto.
Descripción	Iniciar sesión en el sistema a través de una pantalla de inicio de sesión.
Prerrequisitos	Estar registrado en el sistema.
Validación	Iniciar sesión introduciendo el email y la contraseña. Tener un usuario único en el sistema.

Tabla 4.1: Historia de usuario HU01

ID	**HU02**
Nombre	Parar un contenedor del servidor.
Prioridad	Baja.
Riesgo	Bajo.
Descripción	Detener un contenedor de los que están funcionando en el servidor.
Prerrequisitos	Haber iniciado sesión en el sistema. Que el contenedor referido esté iniciado.
Validación	Que el contenedor referido se detenga sin errores.

Tabla 4.2: Historia de usuario HU02

ID	**HU03**
Nombre	Iniciar un contenedor del servidor.
Prioridad	Baja.
Riesgo	Bajo.
Descripción	Iniciar un contenedor de los que están creados en el servidor.
Prerrequisitos	Haber iniciado sesión en el sistema. Que el contenedor referido esté detenido.
Validación	Que el contenedor referido se inicie sin errores.

Tabla 4.3: Historia de usuario HU03

ID	**HU04**
Nombre	Crear un contenedor con el modo sencillo.
Prioridad	Media.
Riesgo	Medio.
Descripción	Crear un contenedor utilizando una imagen de las que ya existen en el servidor.
Prerrequisitos	Haber iniciado sesión en el sistema.
Validación	Que el contenedor referido se cree e inicie sin problemas.

Tabla 4.4: Historia de usuario HU04

ID	HU05
Nombre	Crear un contenedor con el modo avanzado.
Prioridad	Media.
Riesgo	Medio.
Descripción	Crear un contenedor con el modo avanzado, subiendo un fichero zip con los elementos que necesite nuestro servicio incluyendo el fichero *docker-compose.yml*.
Prerrequisitos	Haber iniciado sesión en el sistema.
Validación	Que el contenedor referido se cree e inicie sin problemas.

Tabla 4.5: Historia de usuario HU05

ID	HU06
Nombre	Eliminar un contenedor.
Prioridad	Media.
Riesgo	Medio.
Descripción	Eliminar un contenedor de entre los existentes en el servidor.
Prerrequisitos	Haber iniciado sesión en el sistema. Que el contenedor en cuestión esté detenido.
Validación	Que el contenedor referido se elimine.

Tabla 4.6: Historia de usuario HU06

ID	HU07
Nombre	Seleccionar un servidor.
Prioridad	Media.
Riesgo	Alta.
Descripción	Elegir un contenedor en el menú lateral para así gestionar contenedores en ese servidor en específico.
Prerrequisitos	Haber iniciado sesión en el sistema.
Validación	Que se seleccione ese servidor y a partir de ese momento tanto al gestionar como crear contenedores sean en ese servidor.

Tabla 4.7: Historia de usuario HU07

ID	HU08
Nombre	Borrar un servidor.
Prioridad	Media.
Riesgo	Medio.
Descripción	Borrar uno de los servidores sobre los que está gestionando contenedores.
Prerrequisitos	Haber iniciado sesión en el sistema.
Validación	Que se elimine ese servidor y a partir de ese momento no se puedan gestionar esos contenedores.

Tabla 4.8: Historia de usuario HU08

ID	HU09
Nombre	Agregar un servidor.
Prioridad	Media.
Riesgo	Medio.
Descripción	Agregar un nuevo servidor a la lista de los servidores que se están gestionando.
Prerrequisitos	Haber iniciado sesión en el sistema.
Validación	Que se agregue ese servidor y a partir de ese momento se puedan gestionar sus contenedores.

Tabla 4.9: Historia de usuario HU09

ID	HU10
Nombre	Crear un nuevo usuario.
Prioridad	Media.
Riesgo	Medio.
Descripción	Agregar un nuevo usuario a la lista de los existentes, indicando si tiene permisos para añadir usuarios o no.
Prerrequisitos	Haber iniciado sesión en el sistema con un usuario con permisos de gestión de usuarios.
Validación	Que se agregue ese usuario y a partir de ese momento se puedan iniciar sesión con él.

Tabla 4.10: Historia de usuario HU10

ID	HU11
Nombre	Borrar un usuario.
Prioridad	Media.
Riesgo	Medio.
Descripción	Borrar un usuario de la lista de los existentes. Desde ese momento ya no se podrá iniciar sesión con él.
Prerrequisitos	Haber iniciado sesión en el sistema con un usuario con permisos de gestión de usuarios.
Validación	Que se borre ese usuario y a partir de ese momento no se pueda iniciar sesión con él.

Tabla 4.11: Historia de usuario HU11

ID	HU12
Nombre	Cambiar el permiso de un usuario.
Prioridad	Media.
Riesgo	Medio.
Descripción	Cambiar el permiso de un usuario, para que pueda gestionar usuarios si no podía, o para que ya no pueda si tenía permisoss.
Prerrequisitos	Haber iniciado sesión en el sistema con un usuario con permisos de gestión de usuarios.
Validación	Que se cambie el permiso de ese usuario.

Tabla 4.12: Historia de usuario HU12

ID	HU13
Nombre	Subir un fichero a un contenedor.
Prioridad	Media.
Riesgo	Medio.
Descripción	Poder subir al contenedor el fichero o directorio deseado, comprimido en un .tar, en la ubicación elegida.
Prerrequisitos	Haber iniciado sesión en el sistema.
Validación	Que se suba ese fichero correctamente a la ruta indicada del contenedor.

Tabla 4.13: Historia de usuario HU13

ID	HU14
Nombre	Descargar un fichero de un contenedor.
Prioridad	Media.
Riesgo	Medio.
Descripción	Poder descargar un fichero o directorio del contenedor dada la ruta en la que se encuentra.
Prerrequisitos	Haber iniciado sesión en el sistema.
Validación	Que se descargue ese fichero correctamente.

Tabla 4.14: Historia de usuario HU14

4.1.2 Modelo de dominio

El modelo de dominio de esta aplicación es bastante sencillo, ya que la mayor parte del trabajo se ha empleado en la investigación del funcionamiento de las distintas herramientas y en elaborar la web que lo gestiona todo. En el modelo de dominio se han incluido también las clases de *Container*, *Image* y *Application*, ya que aunque no están hechos en este proyecto, son utilizados por el mismo para su gestión, creación, etc.

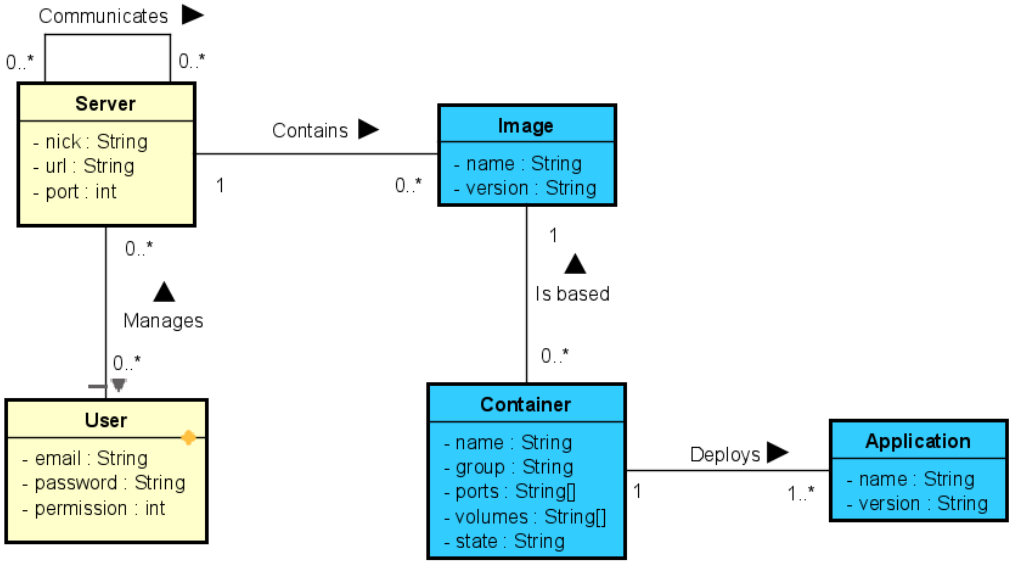


Figura 4.1: Modelo de dominio

Nombre	User
Descripción	Modelo de un usuario que puede acceder al sistema.
Atributos	• email: El email del usuario, el cual se va a usar para iniciar sesión. • password: La contraseña del usuario, la cual se va a usar para iniciar sesión. • permission: Este atributo, que es un número, define si el usuario tiene permisos de gestión de usuarios. El numero 1 significa que no tiene permisos y el 2 significa que sí los tiene. • servers: En esta relación se representan los servidores que puede gestionar un usuario, que puede no haber ninguno (inicialmente) y puede llegar a haber todos los que se quieran.

Tabla 4.15: Usuario

Nombre	Server
Descripción	Modelo de un servidor, que es el que va a alojar los contenedores docker.
Atributos	• nick: El nick con el cual los usuarios van a identificar el servidor. • url: La url en la cual se encuentra la máquina. • port: El puerto en el que está alojado este servidor. • containers: En la relación se puede ver que un servidor puede tener múltiples contenedores. • images: En la relación se puede ver que un servidor puede tener múltiples imágenes. • servers: Esta relación explica la comunicación que hay entre servidores. Esto se explica con más detalle en los siguientes apartados.

Tabla 4.16: Servidor

Nombre	Container
Descripción	Modelo de un contenedor, el cual se gestiona desde nuestro sistema aunque sea ajeno a él.
Atributos	• name: El nombre del contenedor, por el cual se va a identificar. • group: El nombre del grupo al que pertenece, si lo hay. Esto es importante para poder agruparlos cuando todos ellos pertenecen a un mismo proyecto, cuando se han creado con docker-compose. • ports: Este atributo contiene los puertos disponibles del contenedor y a que puerto de la máquina se han redirigido. • volumes: Las rutas del contenedor que están redirigidas a una ruta del servidor. • state: El estado en el que se encuentra este contenedor. • server: En la relación se puede ver un servidor, que es al que pertenece. • image: En la relación se puede ver una imagen, que es en la que se ha basado cuando se creó. • applications: En la relación se puede ver que un contenedor puede tener una o más aplicaciones.

Tabla 4.17: Contenedor

Nombre	Image
Descripción	Modelo de una imagen que existe en el servidor, la cual va a ser utilizada como base para crear contenedores.
Atributos	• name: El nombre de la imagen, con el cual va a ser identificado. • version: La versión de la imagen. • server: En la relación se puede ver que hay un servidor, el cual es en el que está guardada la imagen. • containers: En la relación se puede ver que hay múltiples contenedores, los cuales son los que se han basado en esta imagen cuando se crearon.

Tabla 4.18: Imagen

Nombre	Application
Descripción	Modelo de una aplicación que se ejecuta dentro de un contenedor
Atributos	• name: El nombre de la imagen, con el cual va a ser identificado. • version: La versión de la imagen. • server: En la relación se puede ver que hay un servidor, el cual es en el que está guardada la imagen. • container: En la relación se puede ver el contenedor que contiene esta aplicación.

Tabla 4.19: Aplicación

4.1.3 Diagramas de actividad

En esta sección se incluyen todos los diagramas de actividad, que corresponden a las historias de usuario que se han definido anteriormente. En estos diagramas hay hasta cuatro particiones, que son el usuario, el *frontend*, y los dos tipos de *backend*: el de autenticación y gestión de servidores, y por otro lado el que gestiona los contenedores *docker*.

HU01 - Iniciar sesión en el sistema.

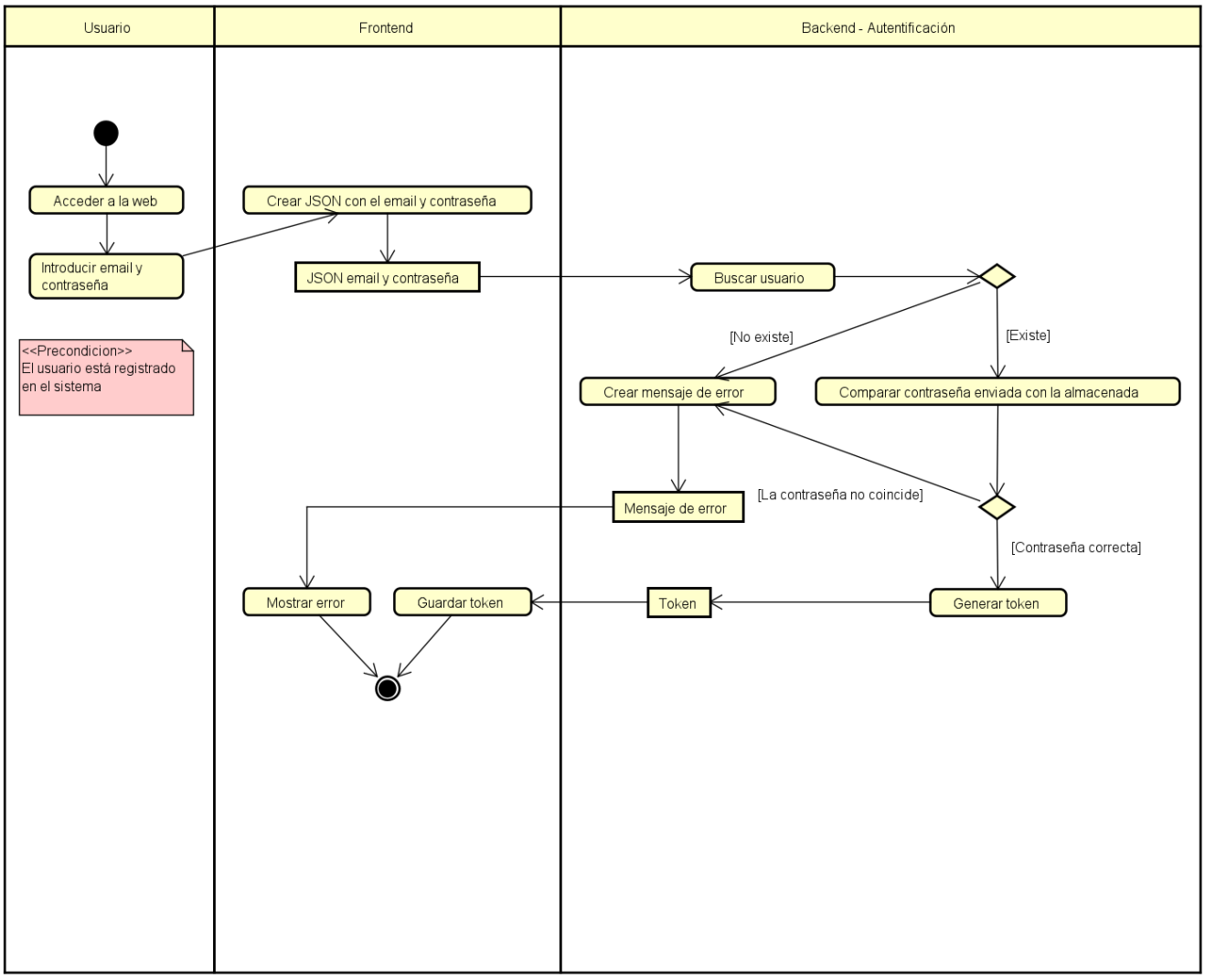


Figura 4.2: Diagrama de actividad de HU01

HU02 - Parar un contenedor del servidor.

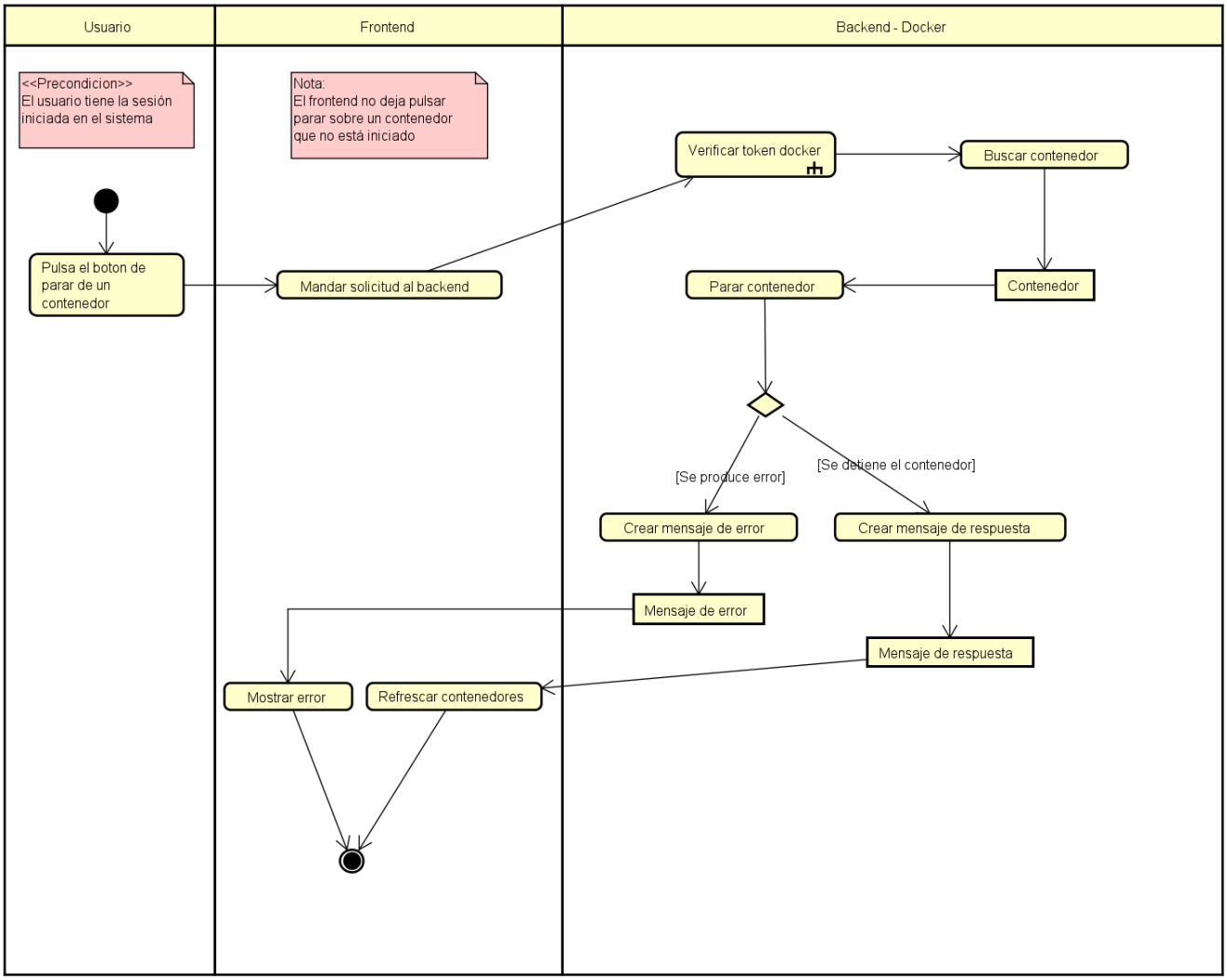


Figura 4.3: Diagrama de actividad de HU02

HU03 - Iniciar un contenedor del servidor.

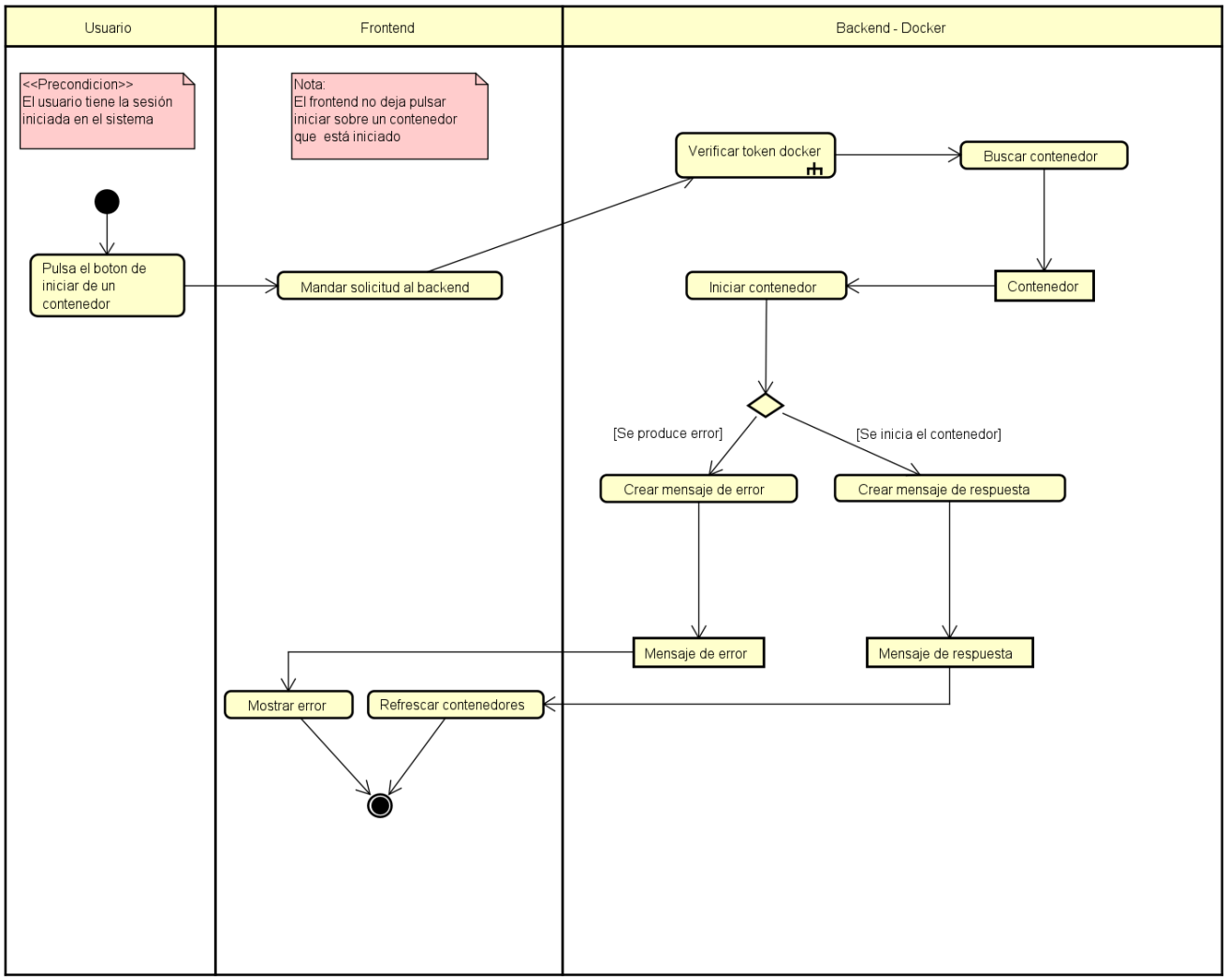


Figura 4.4: Diagrama de actividad de HU03

HU04 - Crear un contenedor con el modo sencillo.

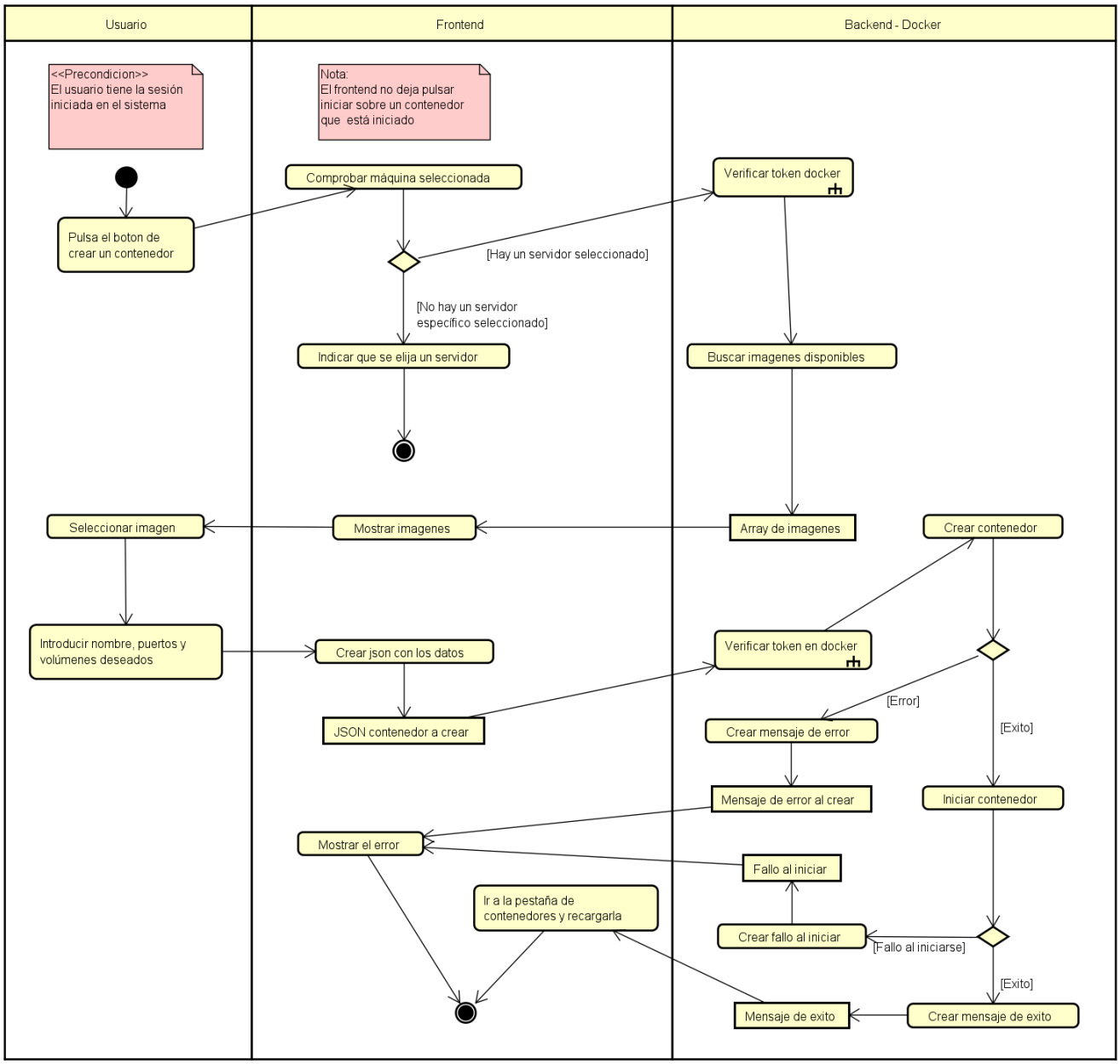


Figura 4.5: Diagrama de actividad de HU04

HU05 - Crear un contenedor con el modo avanzado.

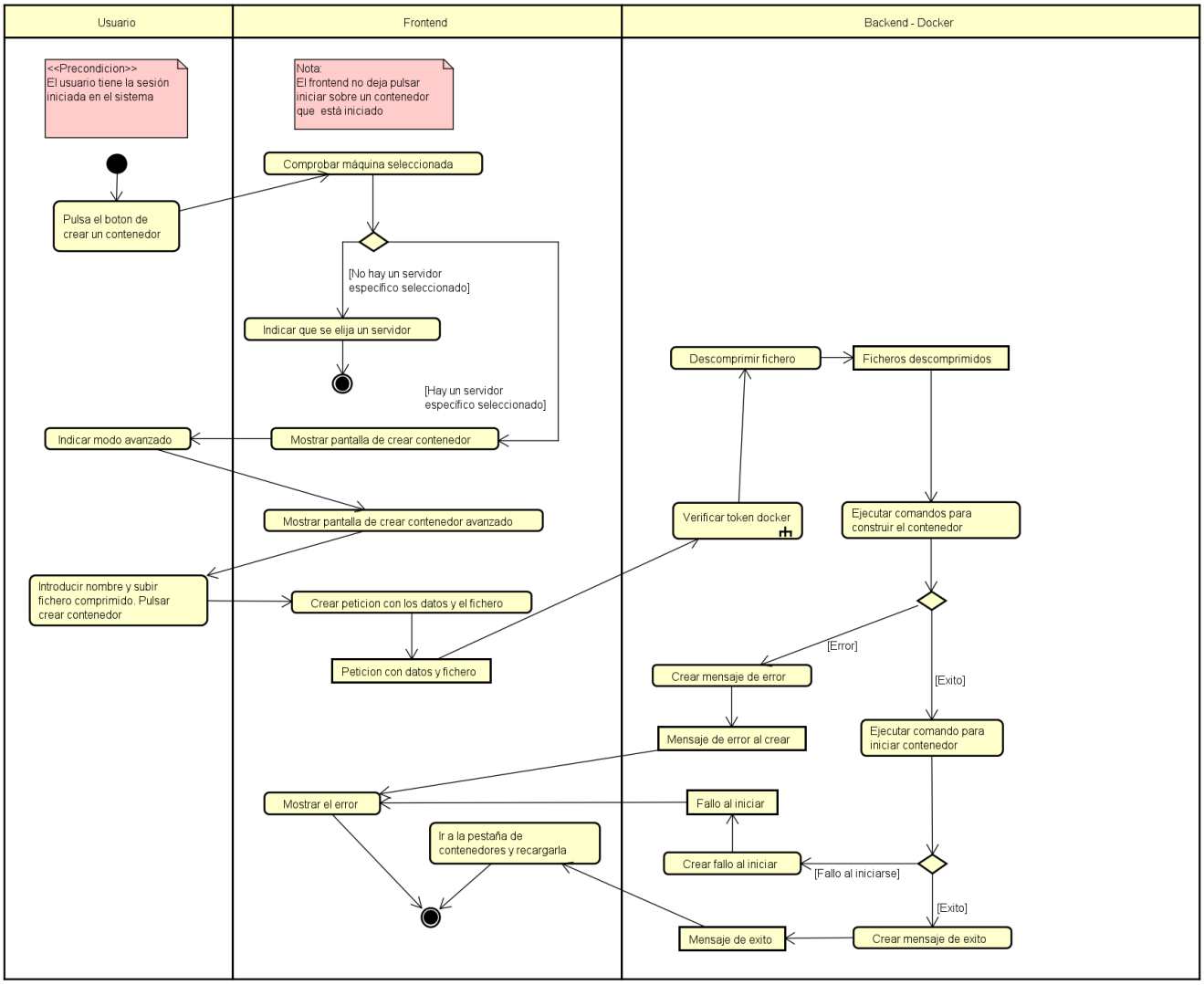


Figura 4.6: Diagrama de actividad de HU05

HU06 - Eliminar un contenedor.

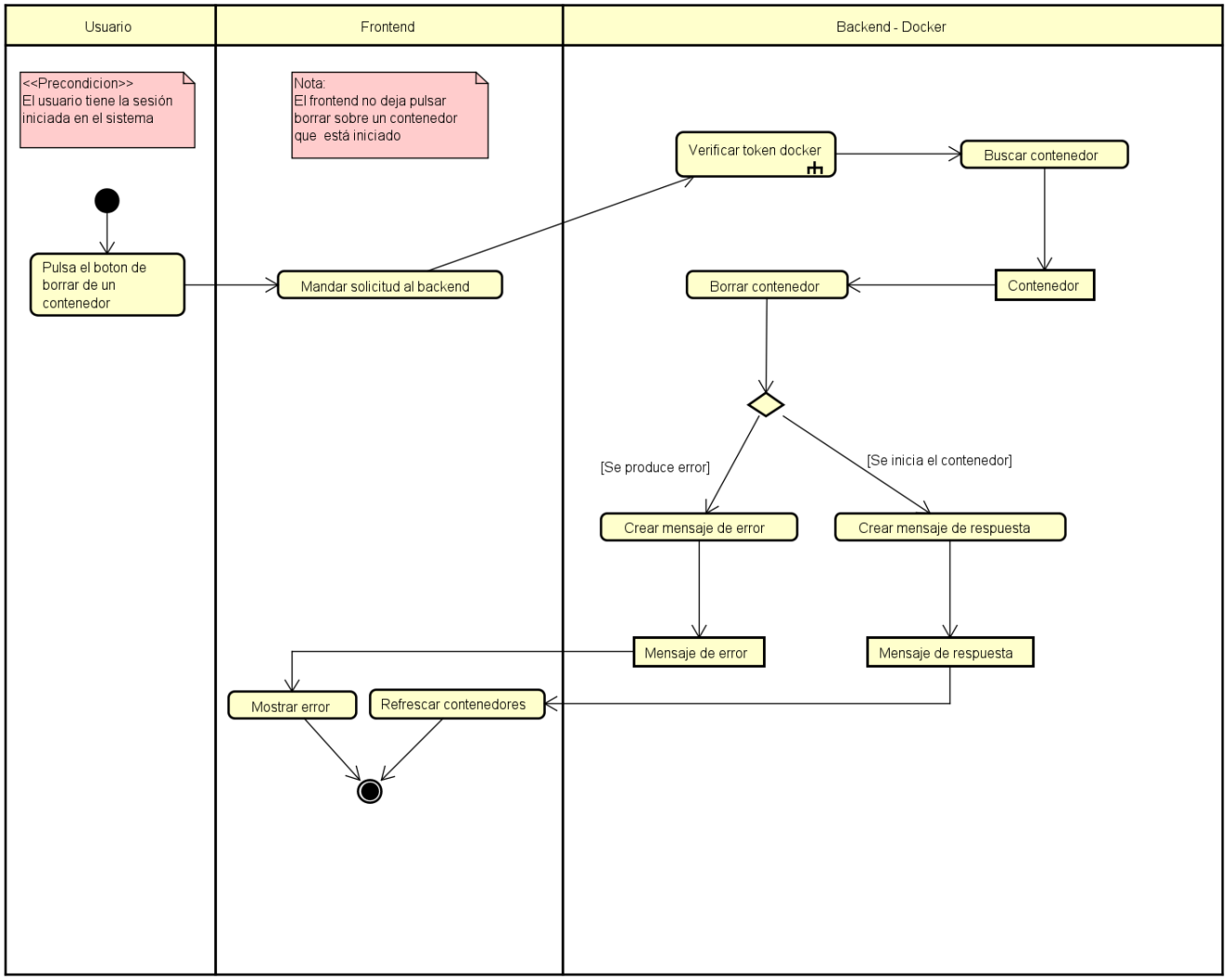


Figura 4.7: Diagrama de actividad de HU06

HU07 - Seleccionar un servidor.

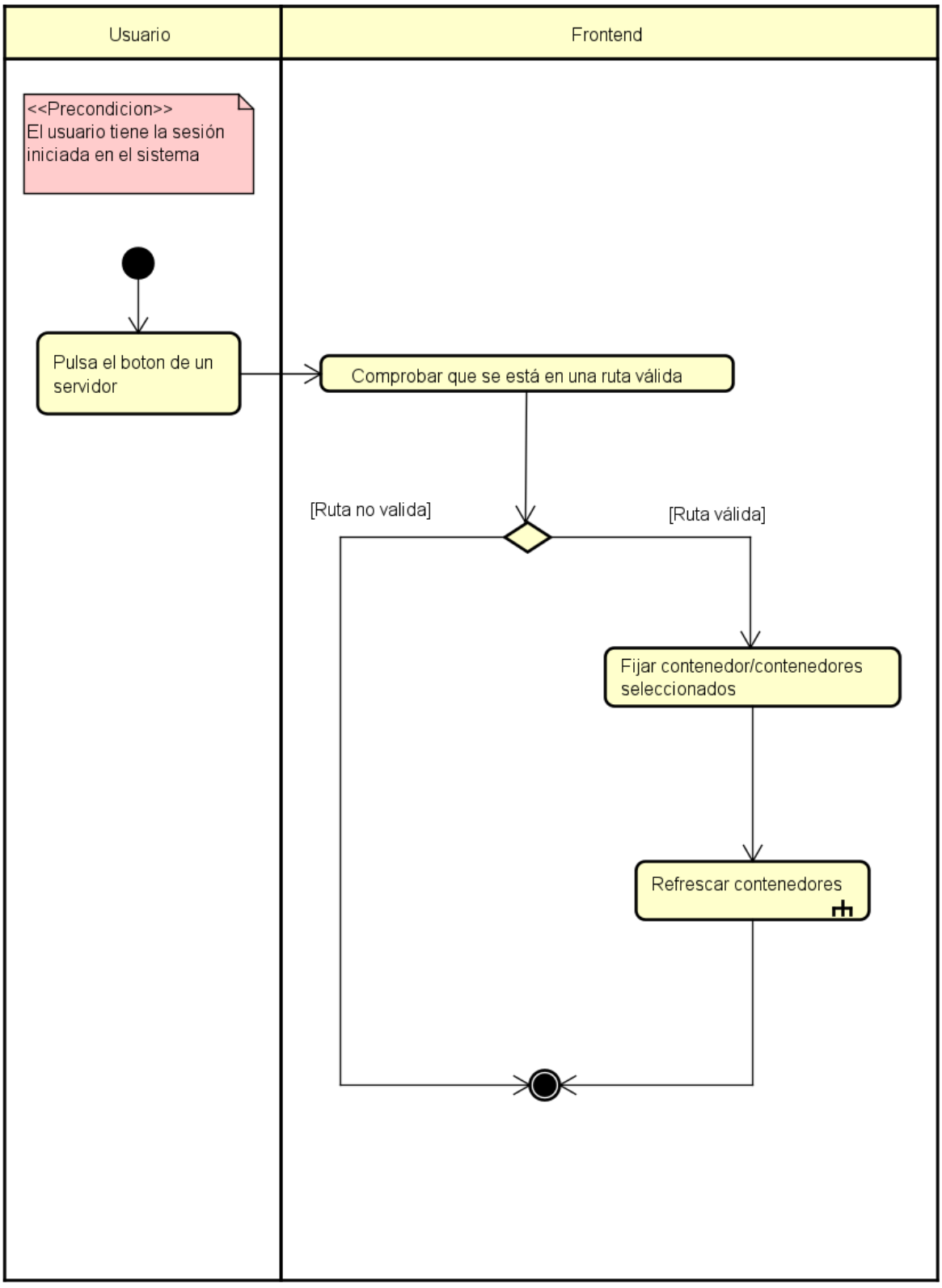


Figura 4.8: Diagrama de actividad de HU07

HU08 - Borrar un servidor.

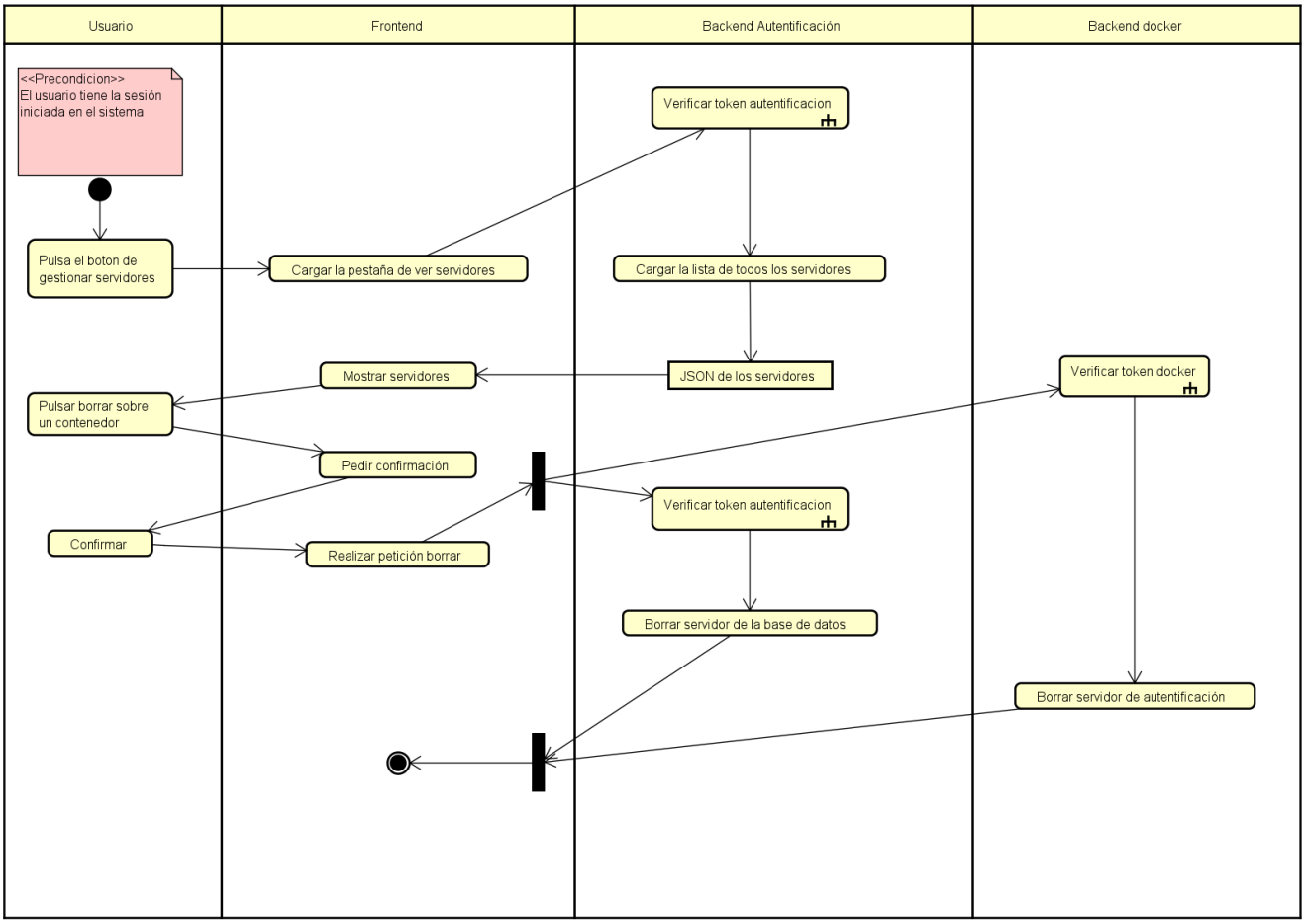


Figura 4.9: Diagrama de actividad de HU08

HU09 - Agregar un servidor.

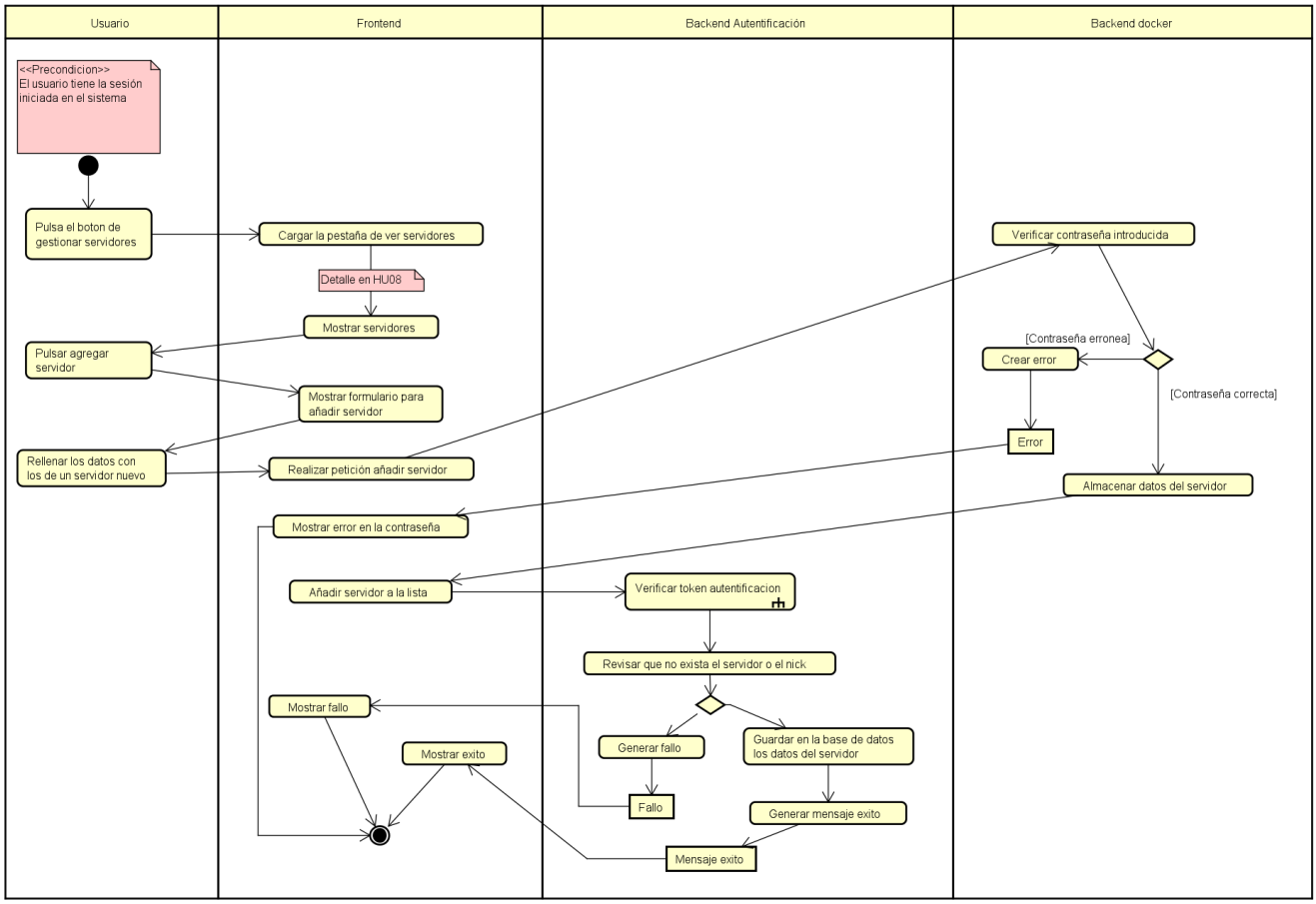


Figura 4.10: Diagrama de actividad de HU09

HU10 - Crear un nuevo usuario.

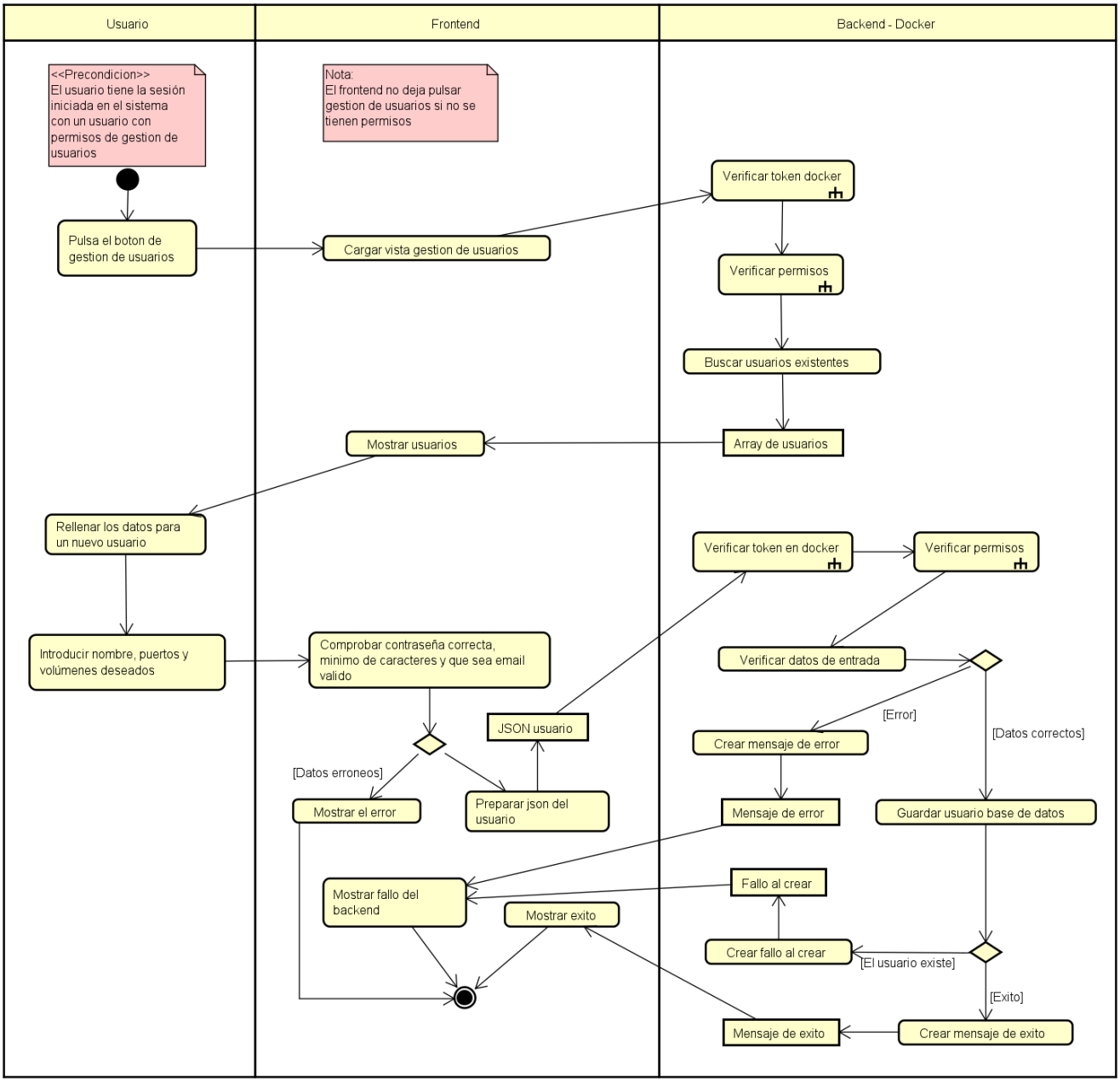


Figura 4.11: Diagrama de actividad de HU10

HU11 - Iniciar sesión en el sistema.

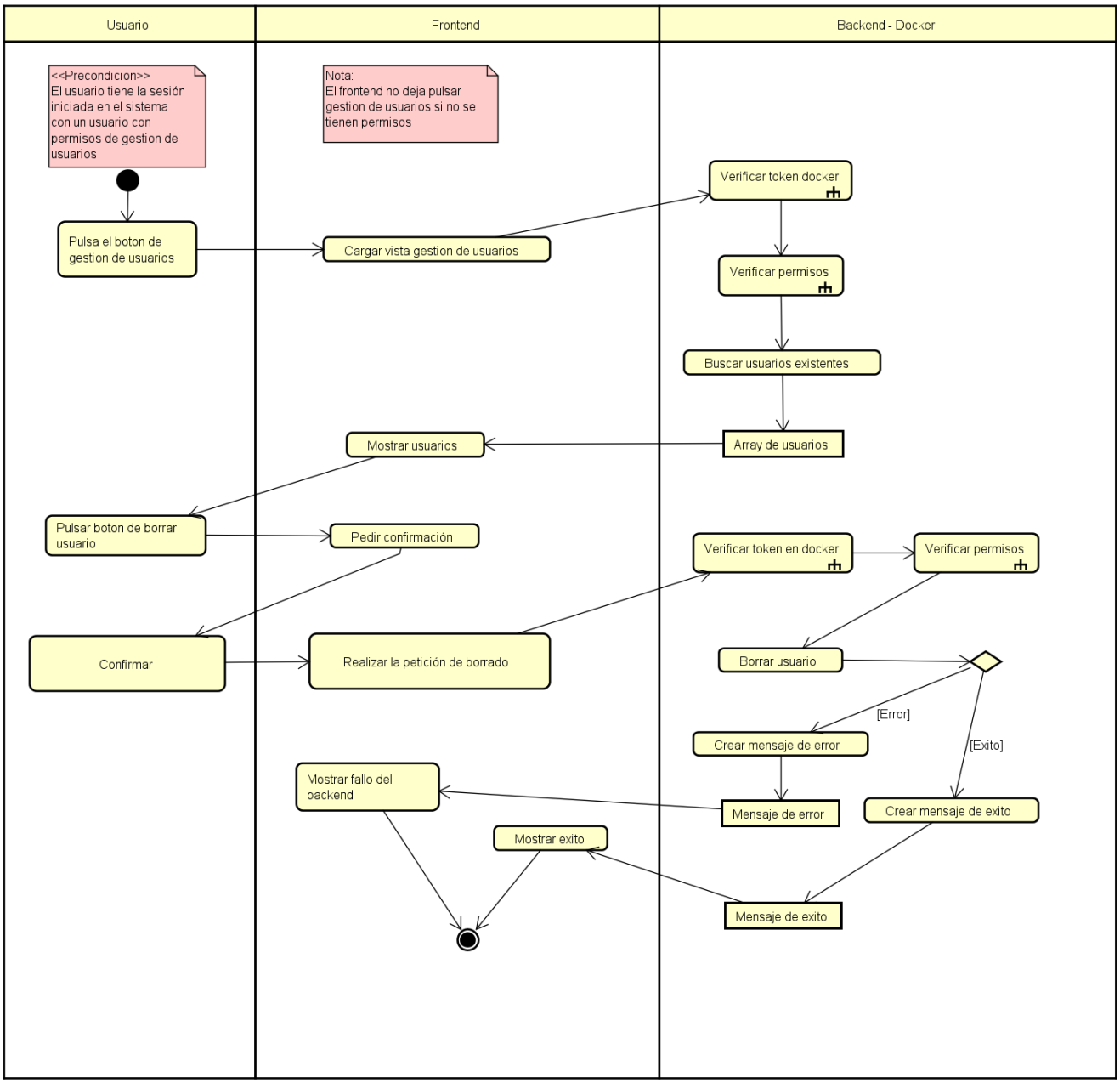


Figura 4.12: Diagrama de actividad de HU11

HU12 - Iniciar sesión en el sistema.

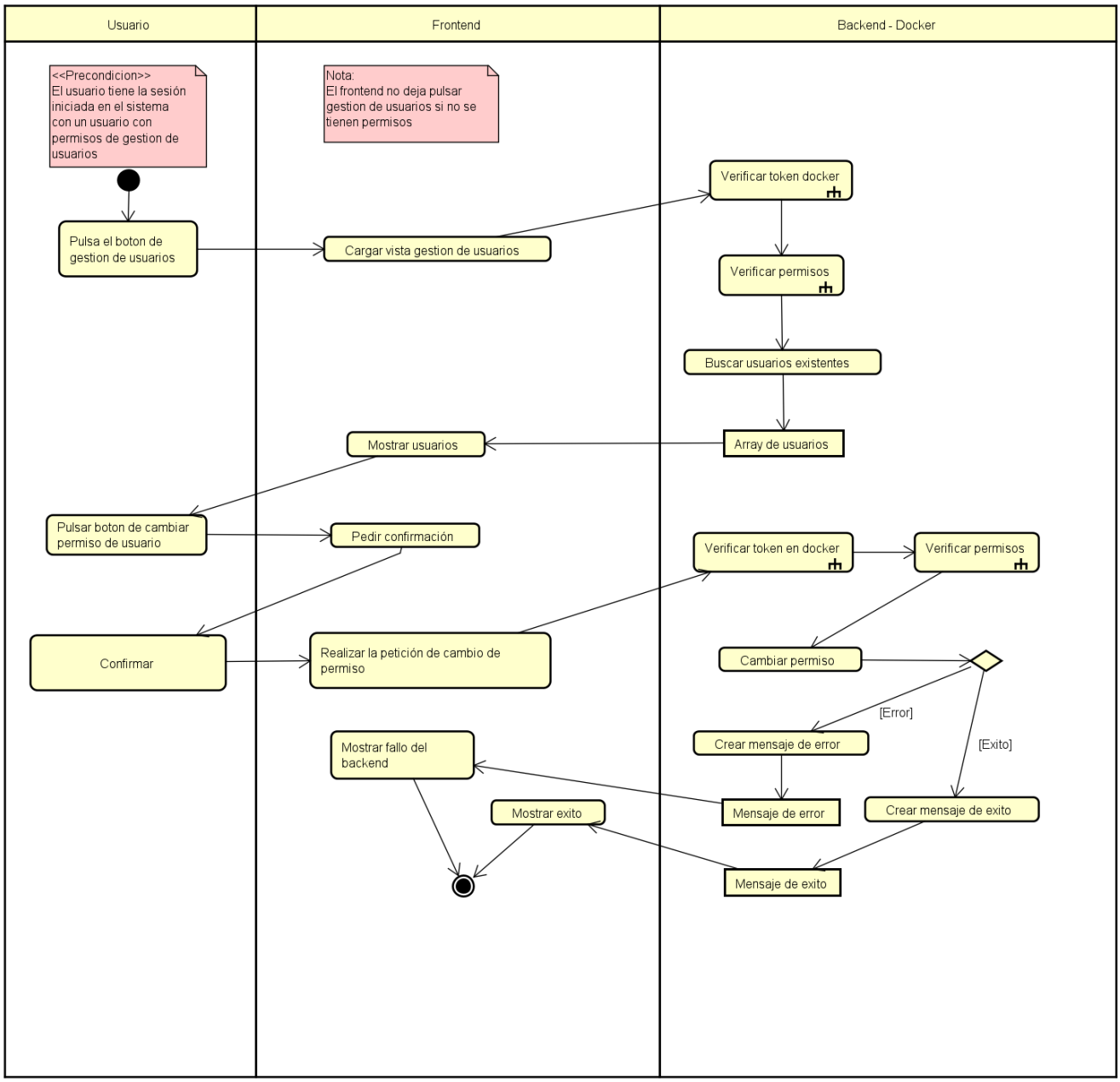


Figura 4.13: Diagrama de actividad de HU12

HU13 - Iniciar sesión en el sistema.

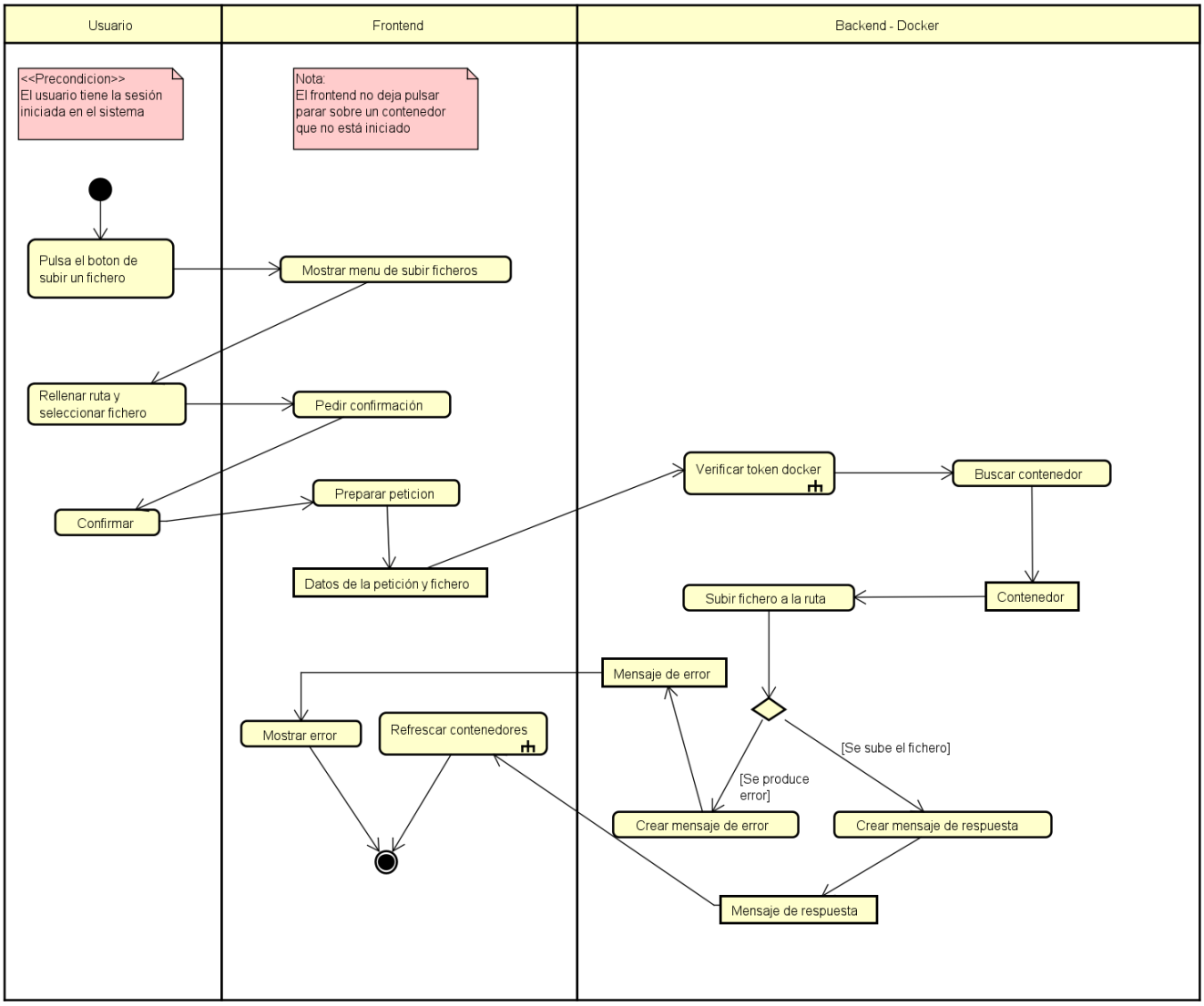


Figura 4.14: Diagrama de actividad de HU13

HU14 - Iniciar sesión en el sistema.

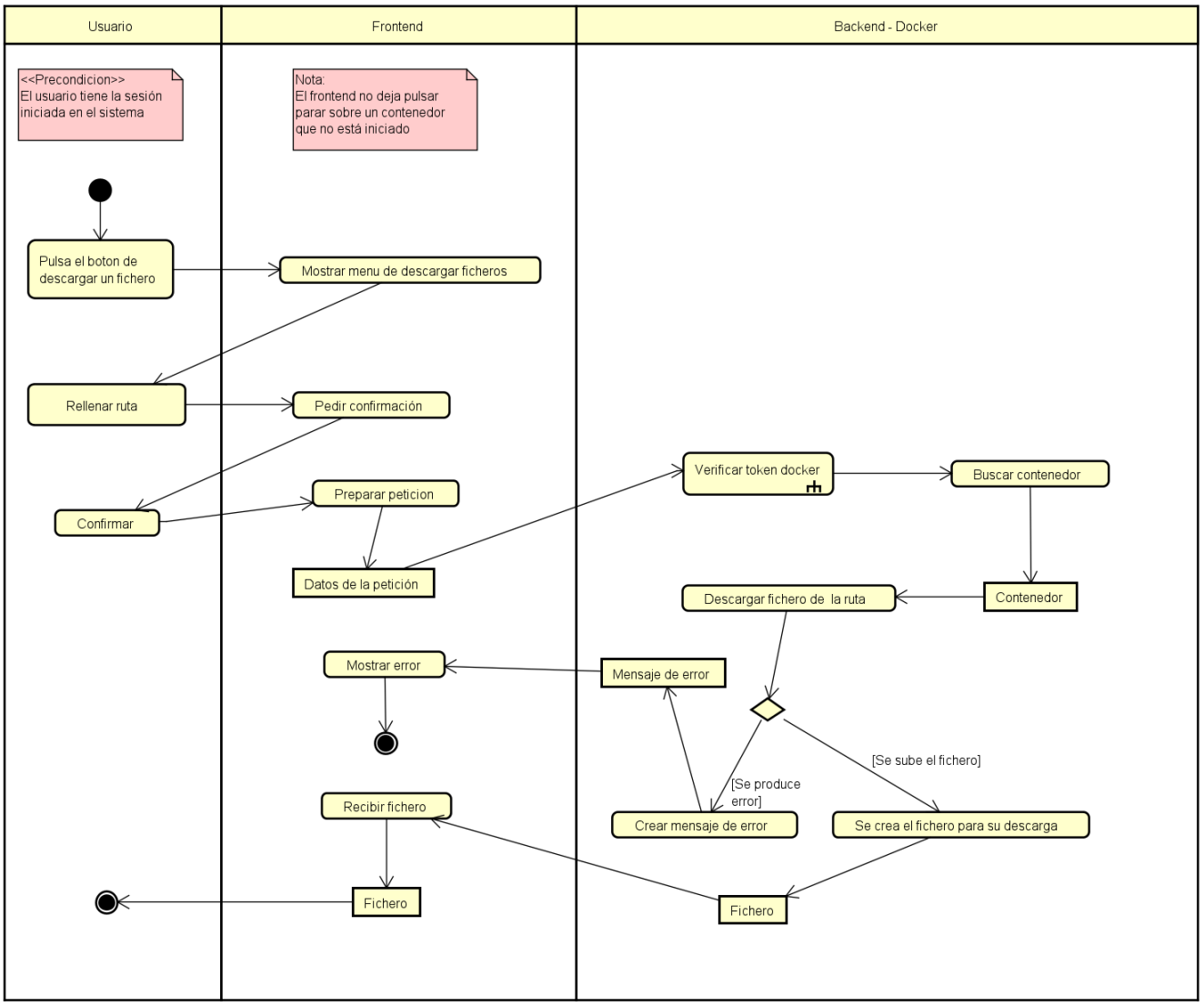


Figura 4.15: Diagrama de actividad de HU14

A continuación se incluyen otros diagramas que corresponden a las acciones repetitivas que se han realizado en los distintos diagramas de las historias de usuario.

Refrescar contenedores.

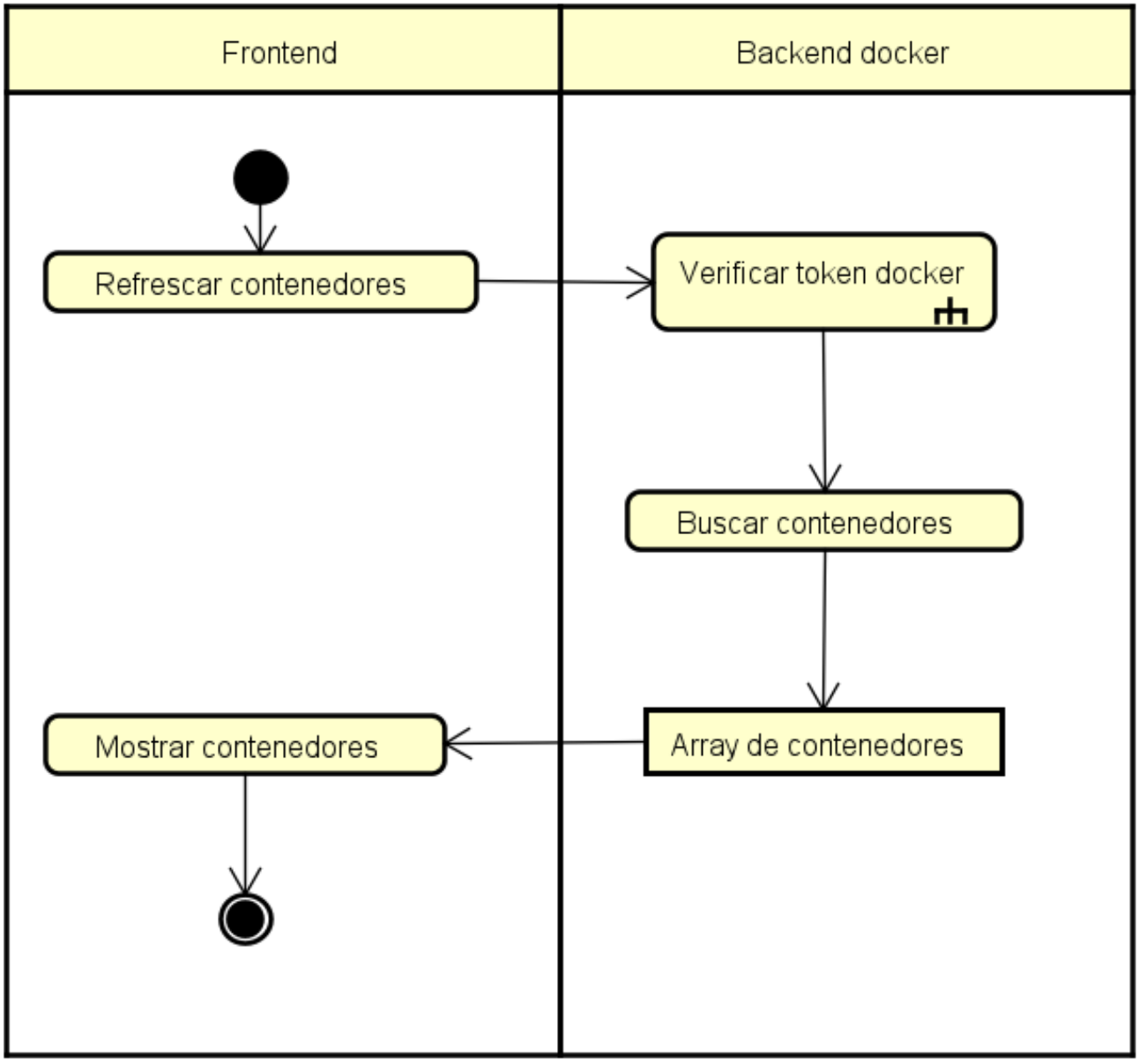


Figura 4.16: Diagrama de actividad de la operación refrescar contenedores

Verificar permisos.

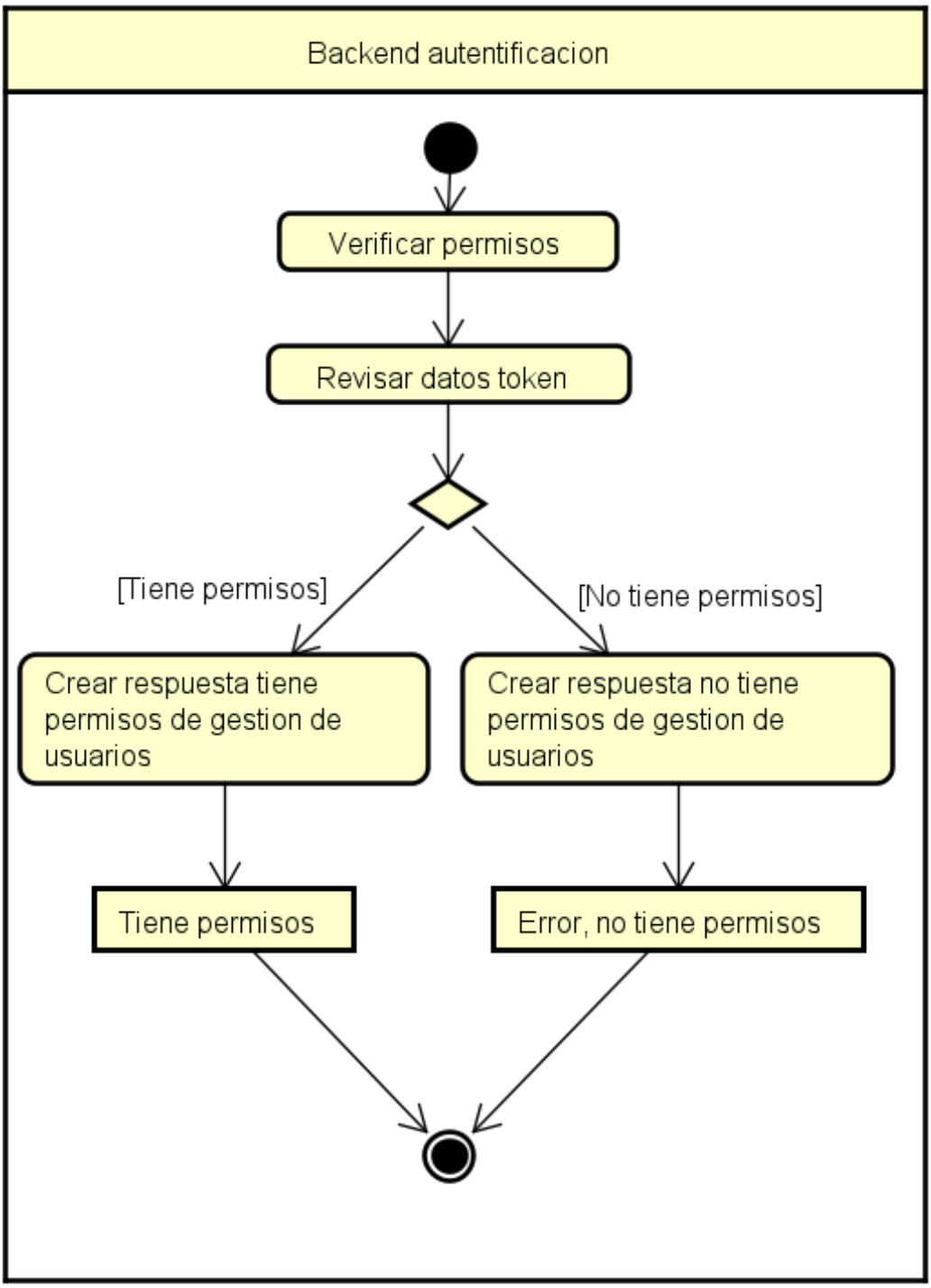


Figura 4.17: Diagrama de actividad de la operación verificar permisos

Verificar token en el servidor de autenticación.

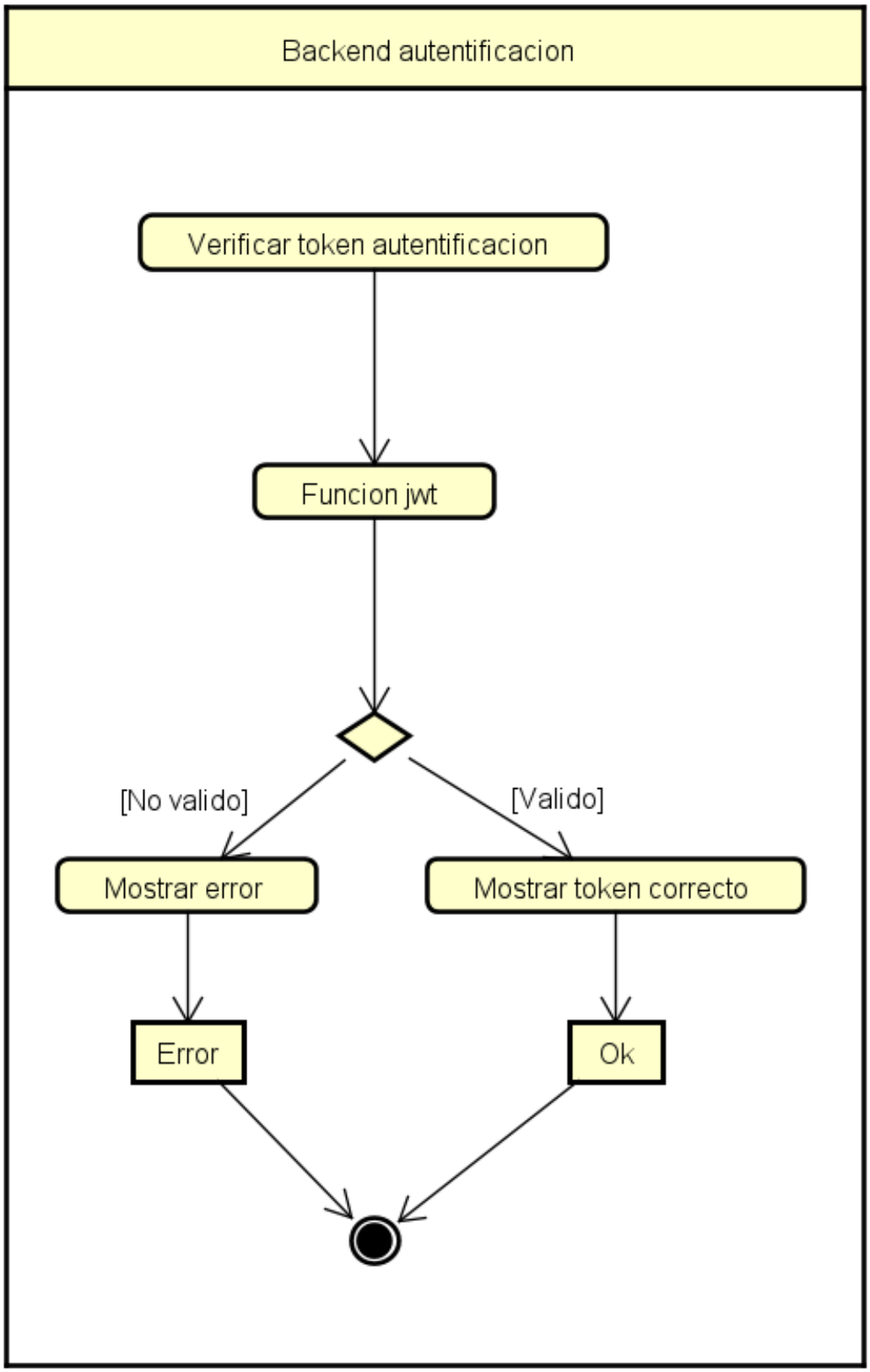


Figura 4.18: Diagrama de actividad de la operación verificar token autenticación

Verificar token en el servidor de docker.

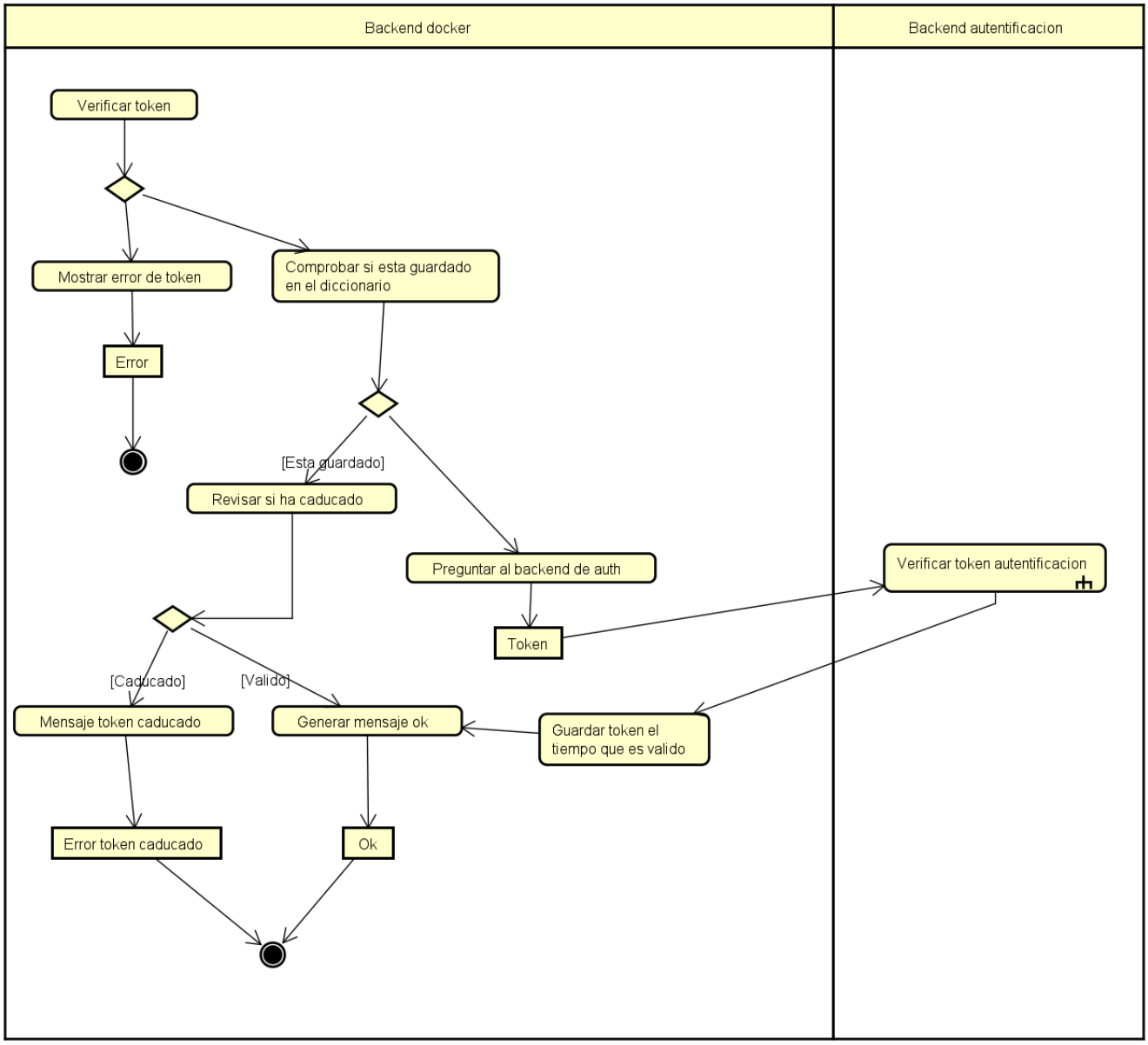
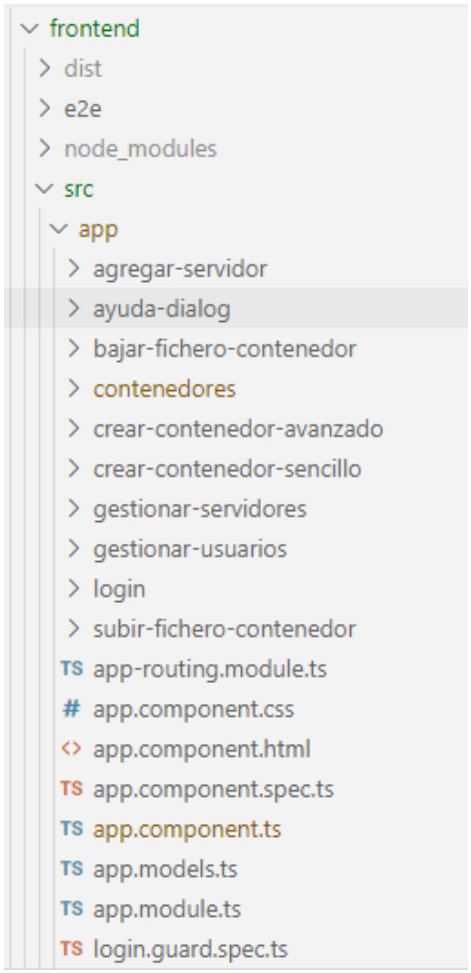


Figura 4.19: Diagrama de actividad de la operación verificar token docker

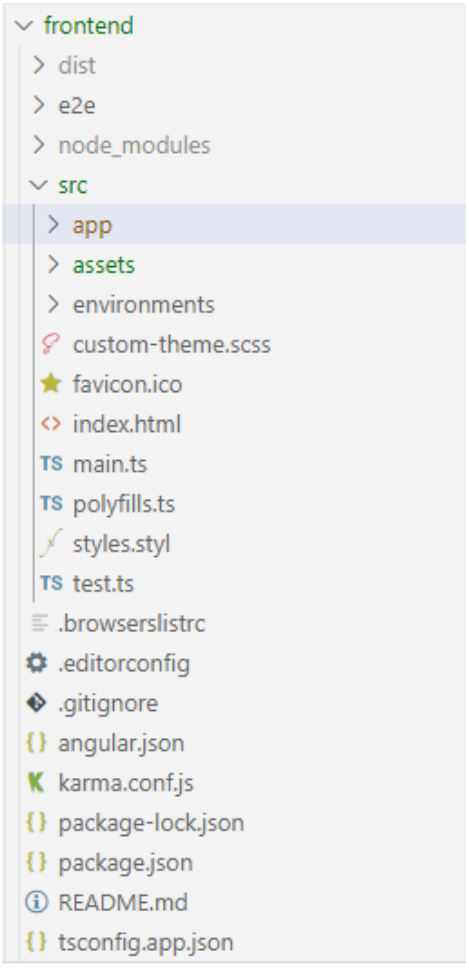
4.2 Estructura del proyecto

4.2.1 Estructura del *frontend*

En la figura 4.20 podemos ver qué estructura tiene el *frontend* del proyecto. Al haber usado el *framework Angular*, se puede ver como la estructura es la habitual en este tipo de proyectos.



Desplegado



Sin desplegar

Figura 4.20: Estructura del *frontend*

Cualquier persona que haya visto anteriormente proyectos *Angular* se puede dar cuenta rápido que se encuentra delante de uno. Se van a analizar los directorios y ficheros:

- El directorio *dist* contiene los ficheros resultantes de la compilación del proyecto.
- Los directorios *e2e* y *node_modules* son ficheros de *Angular* y librerías que no se han manipulado.
- El directorio *src* es el que contiene todo el código del proyecto. Posteriormente se detallará todo el detalle de esta ruta, que es la relevante.

- Finalmente, los archivos *.browserslistrc*, *.editorconfig* y todos los que se pueden ver debajo en la imagen sin desplegar de la figura 4.20, contienen distintos ficheros de configuración que no tienen ninguna relevancia ya que no hay que modificarlos y loas genera *Angular* automáticamente.

A continuación se va a ver el detalle del directorio *src*:

- El directorio *app* contiene todas nuestros componentes, que son todas las vistas de la aplicación. En la vista desplegada se pueden ver todos los componentes. Cada componente contiene:
  - Fichero *.css*. Es el fichero de diseño, que define aspectos como el tamaño del texto, el color, etc.
  - Fichero *.html*. Es el fichero de la web, que tiene distintos parámetros para que se adapte de forma dinámica a lo necesario en cada momento.
  - Fichero *.spec.ts*. Es el fichero de pruebas.
  - Fichero *.ts*. Es el fichero que contiene el código de los métodos que se ejecutan, y que van a definir, entre otras cosas, las variables que leen los ficheros *html*. Desde estos ficheros es en los que se realizan las llamadas al *backend*.

  Este directorio también contiene los archivos raíz de los componentes, que son:

  - El archivo *app-routing.module*, que se encarga de gestionar rutas.
  - Los cuatro ficheros que empiezan por *app.component* son el componente principal, que tienen configuraciones generales y son los que redirigen a cada uno de los componentes existentes. En este componente es donde se escribe el diseño del menú lateral y la barra superior, que son comunes a todas las pantallas del proyecto.
  - El fichero *app.models.ts* es el que define los modelos de objetos que se van a usar en los distintos componentes.
  - El fichero *app.module.ts* se encarga principalmente de gestionar de forma general todas las dependencias del proyecto. También es donde se asigna qué componente corresponde a cada ruta del navegador.
  - Finalmente, los dos ficheros que comienzan por *login.guard*, son los encargados de gestionar el *token* que se usa para verificar la sesión del usuario.
- El directorio *assets* contiene todos los ficheros multimedia que son accesibles desde la web, tales como las imágenes, por ejemplo.
- El directorio *environments* sirve para definir los entornos en los que se va a ejecutar la aplicación, para entre otras cosas, definir variables específicas a cada entorno.
- El resto de ficheros, contienen distintas configuraciones generales de *Angular*, como puede ser el estilo visual, no hay que modificar salvo que queramos cambiar alguna configuración generales y por ello no se van a detallar.

4.2.2 Estructura del *backend de autenticación*

En la figura 4.21 podemos ver qué estructura tiene el *backend* de autenticación del proyecto. Está escrito en *node.js*, el cual no tiene una estructura definida para sus proyectos, lo que quiere decir que esta es libre.

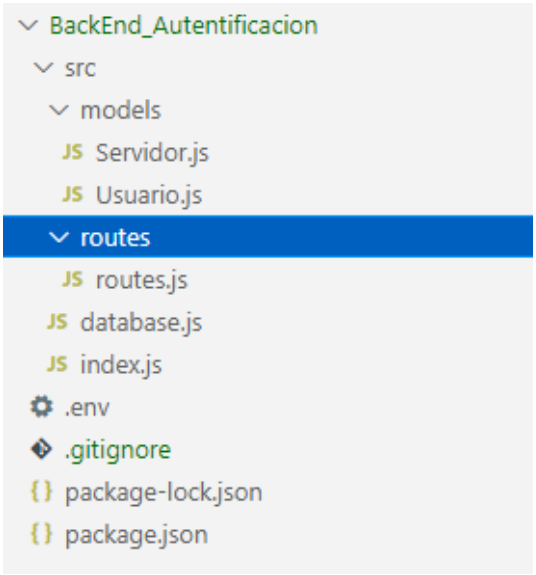


Figura 4.21: Estructura del backend de autenticación.

En este caso, se ha guardado el código en la carpeta *src*, para que esté separado de los ficheros de *node.js*. La estructura general es la siguiente:

- El directorio *src* contiene los ficheros del proyecto. La estructura interna es la siguiente:
  - El directorio *models* contiene los modelos de objeto que existen en la aplicación. Estos son únicamente *Servidor* y *Usuario*. Ambos utilizan la dependencia *mongoose* y tienen el formato adecuado para usarse al subir y descargar datos de la base de datos.
  - El otro directorio, *routes*, contiene el fichero con todas las rutas http. En la tabla 4.20 se muestran todas las llamadas que hay en este fichero.
  - Ya en la raíz del directorio, el fichero *database.js* es el que realiza la conexión a *MongoDB* cuando se inicia la aplicación.
  - El fichero *index.js* es el principal, y es por el que se empieza leyendo la aplicación. Dentro de él se hacen las llamadas a las demás clases.
- El fichero *.env* se utiliza para definir parámetros/variables, como las credenciales de la base de datos.
- El fichero *.gitignore* se usa para excluir ficheros y directorios de *git*.
- Finalmente, los archivos *package.json* y *package-lock.json* son los que usa *node.js* para leer las dependencias del proyecto y ejecutarlo. Nosotros definiremos todo en el primero, y posteriormente, *node.js* utilizará el lock para guardar el estado de las dependencias, etc.

Ruta	Método	Descripción
/login	POST	Ruta para iniciar sesión.
/registerServer	POST	Ruta para iniciar sesión.
/deleteServer	DELETE	Ruta para registrar un nuevo servidor de *docker* en el sistema.
/listaUsuarios	GET	Ruta para obtener una ruta de todos los usuarios y los permisos de cada uno. Restringida solo a usuarios con permisos de gestión de usuarios.
/cambiarPermiso	PUT	Ruta para cambiar el permiso de los usuarios. También está restringida a los usuarios que tengan permisos.
/crearUsuario	POST	Ruta para introducir nuevos usuarios en el sistema. También está restringida a los usuarios que tengan permisos.
/borrarUsuario	DELETE	Ruta para eliminar usuarios del sistema. También está restringida a los usuarios que tengan permisos.
/maquinas	GET	Ruta para obtener la lista de las máquinas existentes en el sistema.
/checktoken	GET	Ruta para comprobar la validez de un token.

Tabla 4.20: Rutas de la aplicación de autenticación y gestión de servidores

4.2.3 Estructura del *backend de docker*

En la figura 4.22 podemos ver la estructura que tiene el *backend* de *docker* del proyecto. Está escrito en *node.js*, al igual que el *backend* de autenticación.

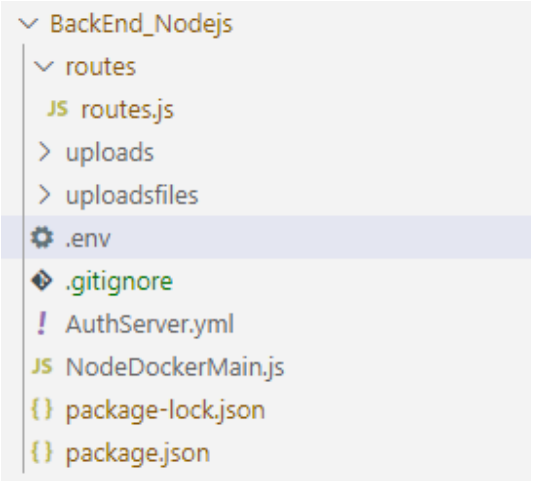


Figura 4.22: Estructura del backend de autenticación.

En este caso, se ha guardado todo el código en la carpeta raíz. La estructura general es la siguiente:

- El directorio *routes* contiene contiene el fichero con todas las rutas http. En la tabla 4.21 se mues-

tran todas las llamadas que hay en este fichero.

- Los directorios *uploads* y *uploadfiles* están vacíos. Es necesario que existan para que este servidor pueda recibir ciertos ficheros cuando se suben.

- El fichero *.env* se utiliza para definir parámetros/variables, como las credenciales de la base de datos.

- El fichero *.gitignore* se usa para excluir ficheros y directorios de *git*.

- El fichero *AuthServer.yml* se utiliza para guardar el servidor de autenticación al que está conectado. Al ser solo una variable no se ha utilizado ninguna base de datos y en su lugar se guarda en este fichero de formato *YAML*.

- El fichero *NodeDockerMain.js* es el principal, y es por el que se empieza leyendo la aplicación. Dentro de él se hacen las llamadas a las demás clases.

- Finalmente, los archivos *package.json* y *package-lock.json* nuevamente son los que usa *node.js* para leer las dependencias del proyecto y ejecutarlo. Nosotros definiremos todo en el primero, como la dependencia de *dockerode*, y posteriormente, *node.js* utilizará el *lock* para guardar el estado de las dependencias, etc.

Ruta	**Método**	**Descripción**
/register	POST	Ruta para registrar la máquina a un sistema.
/unregister	DELETE	Ruta para eliminar el registro de la máquina de un sistema.
/contenedores	GET	Ruta para obtener todos los contenedores creados de la máquina, su estado y sus características.
/activar	POST	Ruta para iniciar un contenedor que esté detenido en ese momento.
/desactivar	POST	Ruta para detener un contenedor que está ejecutándose en la máquina.
/borrar	DELETE	Ruta para eliminar un contenedor de los existentes.
/imagenes	GET	Ruta para ver todas las imágenes existentes en la máquina.
/detalleimagen	GET	Ruta para ver todos los detalles y configuraciones posibles de una imagen.
/crearcontenedor	POST	Ruta para crear un contenedor de forma sencilla.
/subir-contenedor	POST	Ruta para crear un contenedor de forma avanzada subiendo un fichero comprimido.
/descargar-ficheros	GET	Ruta para descargar ficheros de un contenedor.
/subir-ficheros	POST	Ruta para subir ficheros a un contenedor.
/prune	POST	Ruta para ejecutar una limpieza de los contenedores inactivos del servidor.

Tabla 4.21: Rutas de la aplicación de *docker*

4.2.4 Operaciones y comandos *docker* utilizados

A continuación se enumeran todos los métodos utilizados de *docker* mediante *dockerode* explicando para qué se usa cada uno y que funcionalidad tiene.

Inicialmente se ha declarado la variable *docker*, que está instanciada con la API de *dockerode*:

var Docker = require('dockerode');

var docker = new Docker();

Dado que se está trabajando siempre en local, no hay que indicar ningún argumento al crear la instancia *docker*. A continuación se incluyen todas las operaciones generales:

- *docker.listContainers({all: true})* Esta operación es la que se usa para ver todos los contenedores creados en una máquina. Se puede introducir sin ningún argumento, pero en ese caso solo nos mostraría aquellos que estén iniciados en ese momento. Añadiendo el argumento *{all: true}* indicamos que también debe devolver la información de aquellos contenedores que estén detenidos.
- *docker.getContainer(id)* Esta operación nos devuelve un contenedor a partir de su identificador, que es el valor que tendrá la variable *id*.
- *docker.listImages({all: true})* Esta operación es la que se usa para ver todas las imagenes que existen en una máquina. Introduciendo el argumento *{all: true}* se asegura que se devuelva información de todas las que haya creadas incluyendo imágenes intermedias, por si se quisieran utilizar para algún propósito.
- *docker.getImage(image)* Esta operación nos devuelve una imagen a partir de su identificador, que es el valor que tendrá la variable *image*. Este valor será uno de los que se hayan obtenido en la consulta anterior, donde se obtendrán todas las imágenes existentes y sus identificadores.
- *docker.createContainer(configcreate)* Esta operación crea un contenedor en base a la configuración indicada en la variable *configcreate*. Esta variable es un fichero *JSON* con los siguientes parámetros:
  - *Image*: El nombre de la imagen en la que se va a basar el contenedor.
  - *name*: El nombre que va a tener el contenedor.
  - *HostConfig*: Incluye varios parámetros:
    - *PortBindings*: Es un *JSON* que incluye un parámetro por cada puerto redirigible. Por ejemplo ”80/tcp”. Dentro de ese apartado, se incluiría esto: *[{”HostPort”:”puerto”}]* Introduciendo el puerto de la máquina donde dice *puerto*.
    - *Binds*: Es un array de *String*, en el que cada cual redirecciona una ruta de la máquina con una ruta del contenedor: [”/rutamaquina:/rutacontenedor”, ”/rutamaquina2:/rutacontenedor2”]
- *docker.pruneContainers()* Esta operación es equivalente a ejecutar el comandono *docker prune*. Va a limpiar todos los contenedores que no se estén usando en ese momento.

A continuación se incluyen todas las operaciones que pueden ejecutarse a un contenedor (que estará guardado en la variable *container*) obtenido de la forma *docker.getContainer(id)*:

- *container.start()*: Inicia un contenedor. Dará error si no es posible ejecutarlo (por ejemplo, porque ya esté iniciado).
- *container.stop()*: Detiene un contenedor. Dará error si no es posible detenerlo (por ejemplo, porque ya esté detenido).
- *container.remove()*: Elimina un contenedor. Dará error si encuentra algún problema.
- *container.getArchive(path: path)*: Obtiene el fichero que haya en la ruta *path* de ese contenedor. Dará error si no se encuentra el fichero.
- *container.putArchive(file, path: path)*: Sube el fichero que se pase en la variable *file* a la ruta *path* de ese contenedor.

Por último, se incluye la operación utilizada con las imágenes:

- *image.inspect()*: Devuelve todos los detalles existentes de la imagen referida.

A continuación se incluyen los comandos ejecutados para crear contenedores avanzados, utilizando *docker-compose*:

- *docker-compose build*: Se ejecuta sobre la ruta en la que se haya creado el proyecto, donde se encontrarán todos los ficheros, incluyendo el *docker-compose.yml*. Este comando construirá el contenedor correspondiente, guardando las imágenes utilizadas en la máquina para poder utilizarlas después en el modo sencillo.
- *docker-compose up -d*: Se ejecuta sobre la ruta en la que se haya creado el proyecto. Iniciará el contenedor manteniéndolo ejecutado en segundo plano.

4.2.5 Estructura de las aplicaciones avanzadas

El proyecto está pensado entre otras funciones para poder desplegar aplicaciones avanzadas utilizando la interfaz web. Para poder realizar esta tarea es para la que hay que tener mayores conocimientos sobre *docker* y más en específico *docker-compose*. El contenido del fichero subido a la máquina mantendrá el formato inicial tras ser descomprimido del fichero *.zip* en el que deberá ser subido.

El formato de estas aplicaciones lo decide cada desarrollador a excepción de un fichero, el *docker-compose.yml*. Este fichero deberá encontrarse en la raíz del fichero *.zip* subido tal y como se muestra en la figura 4.23, ya que será el que se ejecute para poder desplegar la aplicación deseada con todos los servicios que lo compongan.

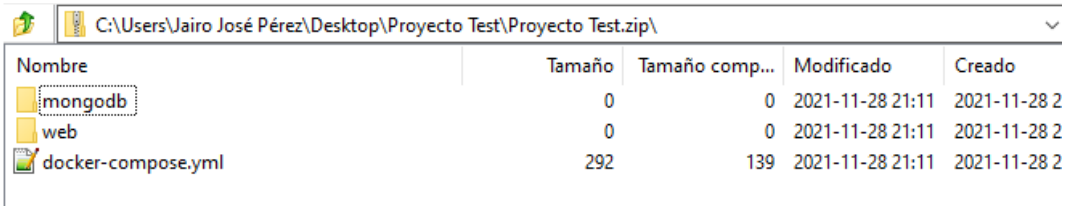


Figura 4.23: Estructura del fichero *.zip*

A partir de ahí, cada desarrollador puede organizar todos los componentes de la forma deseada, ya que en el fichero *docker-compose.yml* estarán las referencias de todos los ficheros necesarios, y esta al ser configurable permite absoluta libertad en la elección, siempre y cuando todo esté dentro del directorio inicial, lógicamente. No obstante, es recomendable separar en distintos directorios cada componente de la aplicación para una mayor organización. En la figura 4.24 se muestra como ejemplo el contenido de un fichero *docker-compose.yml*.

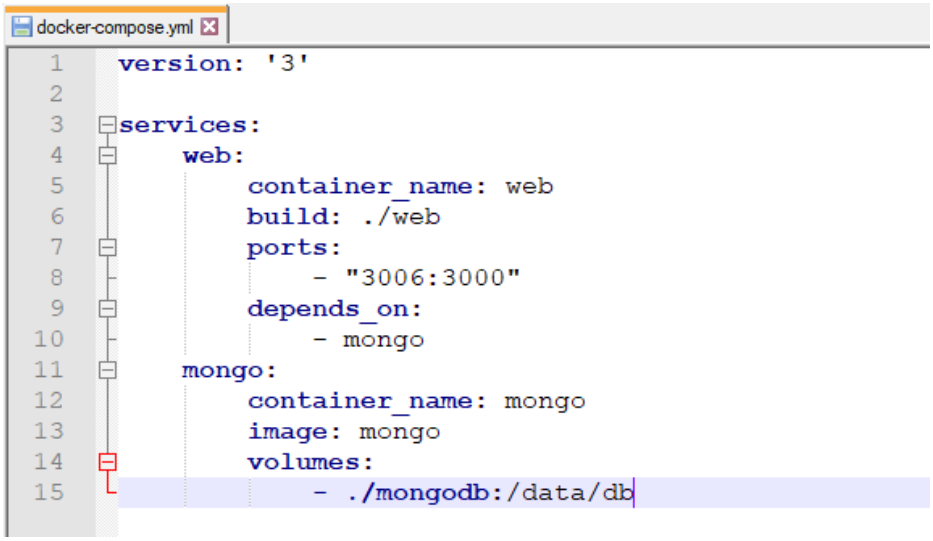


Figura 4.24: Estructura del fichero *docker-compose.yml*

En la figura se observa como se definen los nombres de los contenedores, la redirección de puertos, y la referencia a ficheros que hay en las carpetas que se encuentran en la ruta, como es el caso del directorio *web*, donde se encuentra el código de la aplicación web a desplegar, y por otro lado en la carpeta *mongodb*, que va a utilizarse para guardar los contenidos del contenedor de base de datos.

4.3 Diagramas de diseño

4.3.1 Componentes de la aplicación

En la figura 4.25 se puede ver un esquema general de todos los componentes con la aplicación desplegada. En este esquema hay dos servidores con *docker*, aunque podría haber solo uno o un número ilimitado de los mismos.

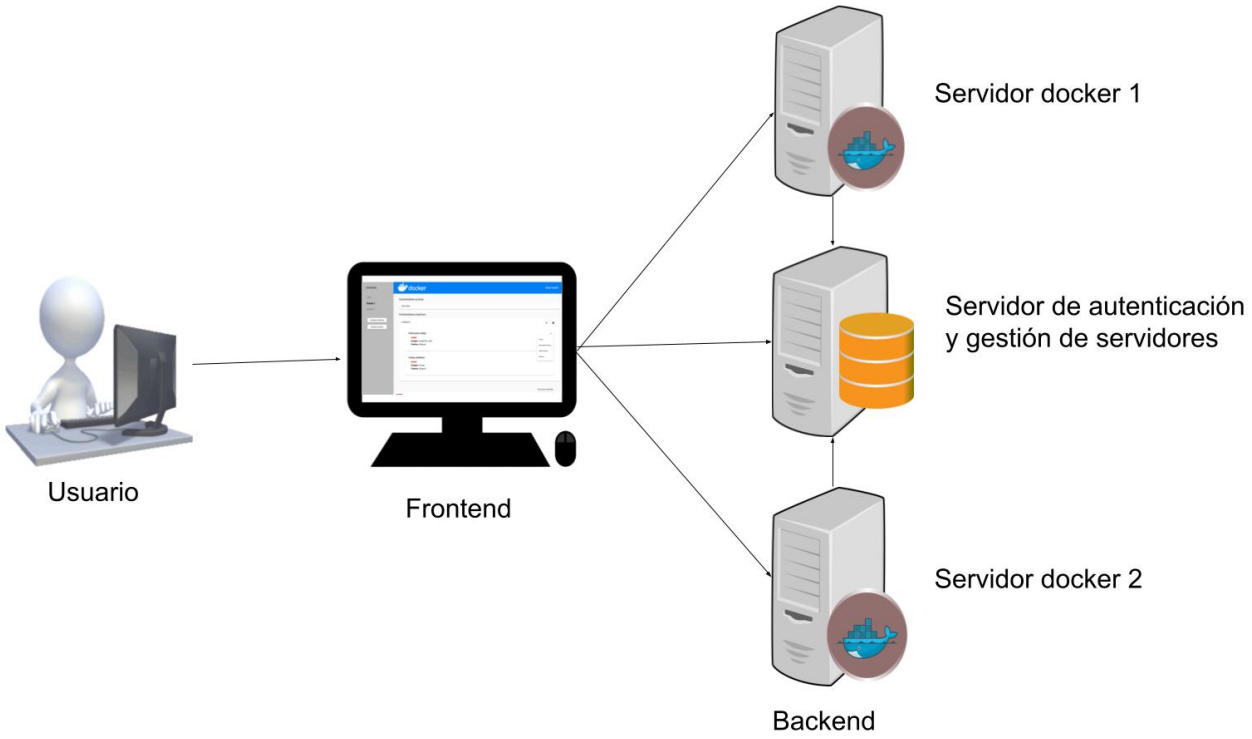


Figura 4.25: Esquema de los componentes

Se puede observar que la comunicación del usuario se realiza a través del *frontend* con el servidor de *backend* que requiere en cada momento, la cual va a utilizar siempre el protocolo *HTTP* [41]. En la parte central se encuentra el servidor de autenticación y gestión de servidores, que contiene una base de datos de tipo *MongoDB* que almacena todos los datos de esta aplicación. Por otro lado, los servidores de *docker*, que en este esquema se encuentran arriba y abajo a la derecha, dependen del servidor de autenticación para poder verificar las sesiones de los clientes que les lleguen, lo que provoca que tengan que comunicarse con él, también vía *HTTP*.

4.3.2 Arquitectura y patrones de diseño

En este apartado se explican los distintos patrones utilizados en la realización del proyecto.

Arquitectura de capas

El patrón de arquitectura de capas [42] es un patrón que distribuye la aplicación en distintas partes de tal forma que cada una se encargue de una función. La primera capa es la de presentación, que en la aplicación corresponde con el *frontend* realizado en *Angular*. La segunda capa, que se encarga de la lógica de negocio, en esta aplicación corresponde con los *backend*, tanto de *docker* como de gestión de servidores y autenticación. La última capa es la de persistencia, que se corresponde con la base de datos *MongoDB* en la parte de gestión de usuarios y servidores. *Docker* por su lado también tiene esta capa de forma interna para almacenar todos us datos y contenedores que tiene creados.

Módulos

Este patrón se ha utilizado mayormente en *Angular*, aunque también existe en el *backend*, ya que se utiliza con *JavaScript*. Este patrón se utiliza en la creación de objetos, para poder simplificar la forma de acceder a ellos y poder modificarlos, enviarlos, etc. En el *frontend* se ha utilizado con los *json* que se reciben al comunicarse con los *backend*, para poder acceder a los puntos relevantes de forma más cómoda y rápida. En el *backend* también se ha utilizado para los objetos de la base de datos *MongoDB*, que utiliza el módulo *mongoose* para realizar las peticiones, tanto de descarga como de subida de datos.

Singleton

Este patrón sirve para asegurar instancias únicas en las clases, de tal forma que solo se pueden realizar desde ahí las operaciones [43]. *Angular* utiliza este patrón para asegurarse de que siempre se llama a la misma instancia en clases clave, como puede ser la instancia principal y ciertas configuraciones. Algunos casos en los que se usa singleton en *Angular* es en *forRoot()* o *Router* [44].

4.3.3 Modelo de datos

Figura 4.26: Modelo de datos

En la figura 4.26 se pueden observar los dos objetos que hay en el modelo de datos, que son los que se van a guardar en nuestra base de datos de tipo *MongoDB*. Estos dos objetos son los mismos que se han visto en el modelo de dominio (figura 4.1), excluyendo lógicamente los elementos que no

pertenecen a este sistema. Adicionalmente, *MongoDB*, al guardar objetos, siempre añade un campo más que se llama *_id*, el cual va a servirle a la base de datos para poder identificar cada elemento.

Los elementos mencionados en el modelo de dominio que no pertenecen a este sistema no se encuentran en este modelo de datos. Todos ellos los guarda *docker* de forma autónoma, y este sistema no tiene que gestionar en ningún momento ningún apartado de ellos. Estarán guardados en las máquinas en las que se encuentren cada uno de los contenedores, imágenes, aplicaciones, etc. Adicionalmente *docker* almacena mucha más información y configuraciones que en este proyecto no van a ser utilizadas.

4.3.4 Diagrama de paquetes

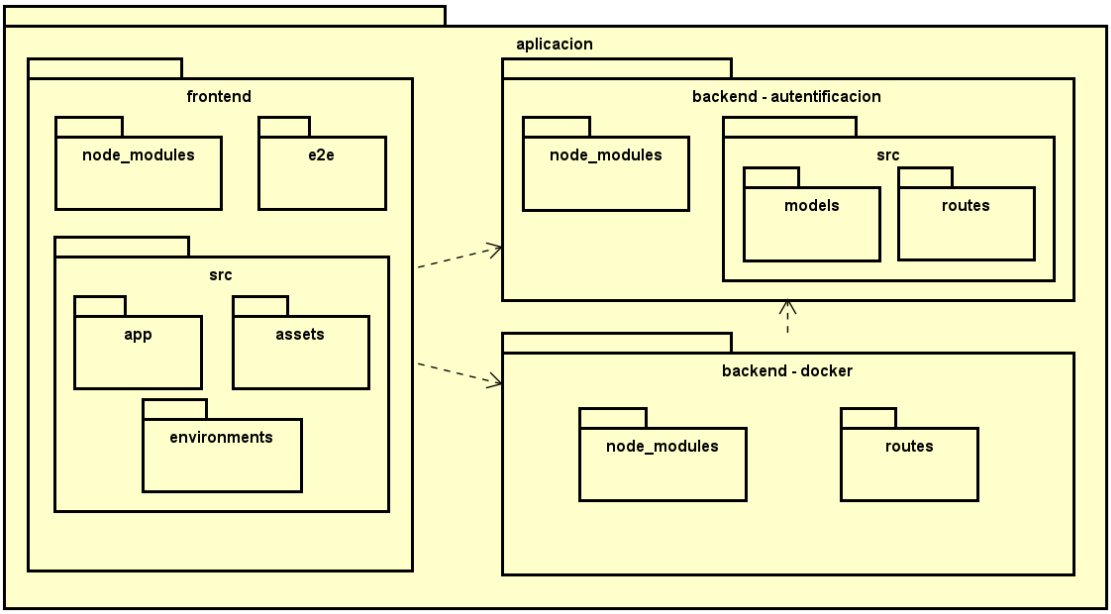


Figura 4.27: Diagrama de paquetes

Una vez se ha visto la estructura del proyecto en el apartado anterior, el diagrama de paquetes se puede comprobar que es bastante similar. Las tres partes de esta aplicación contienen el paquete *node_modules*, ya que los dos backend están hechos en *node.js* y el *frontend*, *Angular*, es un *framework* también de *node.js*.

En la parte del *frontend* se pueden observar todos los paquetes que se habían comentado anteriormente. El directorio *src* contiene el código de la aplicación, donde se incluye el paquete *app*, en el cual estarían todos los componentes. Los dos paquetes externos al *src* contienen ficheros de *Angular* que no hay que manipular.

Por otro lado, los dos *backend* tienen los paquetes que ya se explicaron en el apartado anterior. Estos paquetes son los que necesita cualquier aplicación escrita en *node.js*, que incluyen los *node_modules* y luego aparte los que creemos nosotros en el código.

4.3.5 Diagrama de despliegue

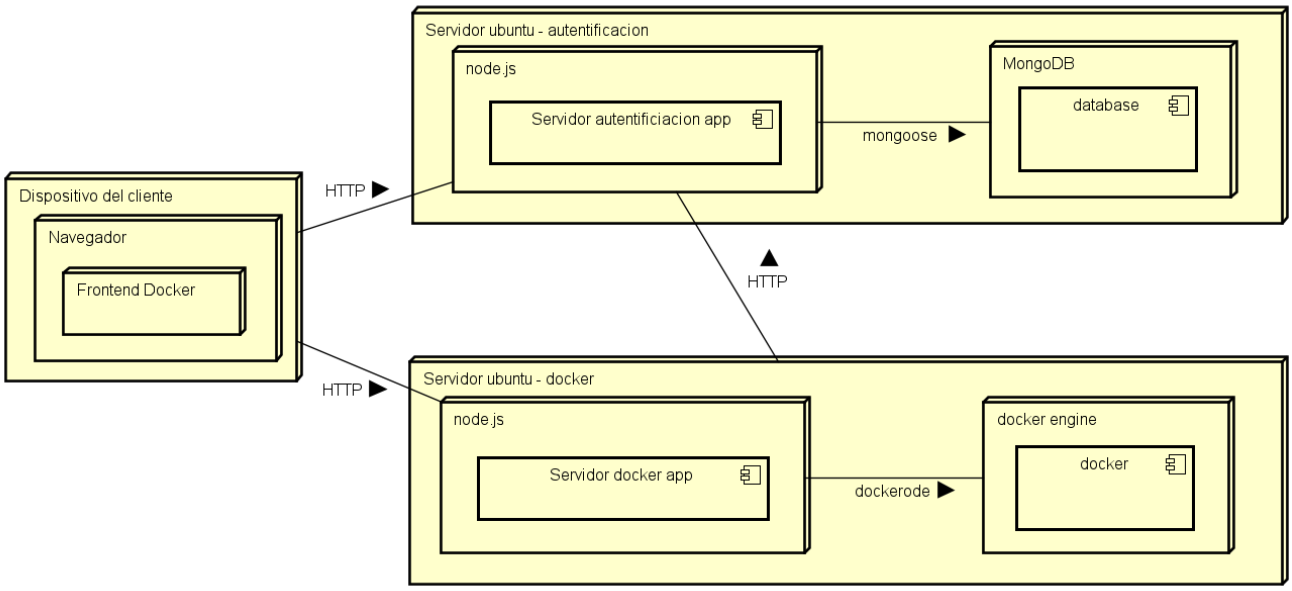


Figura 4.28: Diagrama de despliegue

El diagrama de despliegue explica de forma visual cómo se han organizado todas las partes de la aplicación. Se puede apreciar que el *frontend*, que se ejecuta en el navegador, se conecta directamente a los dos *backend* que existen. El servidor de autenticación contiene la base de datos en la que va a guardar los datos explicados anteriormente. Por otro lado, los servidores de *docker* se conectan mediante *dockerode* a la plataforma, para poder gestionar los contenedores. La conexión *http* entre los servidores de *docker* y el de autenticación se produce para verificar los *tokens JWT* que se envíen en las peticiones, ya que los servidores de *docker* dependen del servidor de autenticación para ello.

4.3.6 Seguridad del proyecto

La seguridad en este proyecto es un punto muy importante ya que nadie que no esté autorizado puede acceder a ninguna de las partes del sistema, así como ejecutar ninguna operación de ningún tipo de las que permite el sistema desarrollado. Para abordar el tema de la seguridad se ha utilizado el método de seguridad JWT [37], que es un estándar abierto que define una forma para transmitir información segura entre dos partes utilizando un objeto JSON. Esta información es segura y confiable ya que está firmada digitalmente y esto se puede hacer de dos formas que son utilizando un secreto en la firma (que es la forma utilizada en este proyecto) o utilizando un par de claves públicas y privadas.

Respecto a todos ficheros de la aplicación y de *docker*, se almacenan en la máquina donde se despliega. Ninguno de esos ficheros es accesible desde el exterior y solo alguien con acceso a esa máquina puede acceder a esos ficheros o manipularlos.

Capítulo 5

Pruebas

5.1 Pruebas del backend

Para realizar la prueba de los servicios del *backend*, se ha utilizado la aplicación *postman* [31], que sirve para realizar llamadas a *API REST*. Las pruebas que se han realizado sobre la aplicación que gestiona los *docker* han ido evolucionando conforme se ha desarrollando la aplicación y se han implementando todas las funciones. Por ejemplo, el servidor que gestiona los contenedores ha sido desarrollado anteriormente al de autenticación, y por ello, la seguridad, entre otras cosas, no estaba incluida en las primeras pruebas que se han realizado. En la colección de pruebas finales, se prueban todas las llamadas que existen en la aplicación, y están pensadas en relación a las llamadas que se van a realizar desde el navegador por parte del *frontend*.

Las pruebas de la aplicación de autenticación y gestión de servidores también se han realizado con *postman*. Esta aplicación tiene todas las llamadas pensadas para que se realicen desde el *frontend* excepto una, que es la que está pensada para verificar *tokens*. Con el uso de *postman* se pueden probar todas las llamadas seguidas con indiferencia de si se van a llamar desde un navegador o desde otro servidor. En el apéndice D pueden verse con detalle todas las pruebas realizadas para verificar el correcto funcionamiento tanto de la aplicación de autenticación y gestión de servidores como de la aplicación que gestiona los *docker*.

5.2 Pruebas del frontend

Las pruebas que se han realizado sobre el *frontend* están divididas en dos apartados, el primero, para comprobar el correcto funcionamiento de todas las operaciones; y el segundo, para probar la usabilidad de la aplicación.

5.2.1 Pruebas de funcionamiento

Las pruebas de funcionamiento se han realizado de forma manual, probando a mano las opciones de la aplicación. Todas las pruebas realizadas coinciden con las historias de usuario ya descritas en el apartado 4.1.1 de este mismo informe. Para cada historia de usuario, se ha realizado con un caso correcto, y con los casos erróneos posibles. Por ejemplo, en la historia de usuario HU01, se ha iniciado sesión con un usuario y contraseña válidos, pero también se ha probado con un usuario que no existe, y con uno que sí existe pero introduciendo mal la contraseña.

5.2.2 Pruebas de usabilidad

Para poder comprobar que la interfaz tiene una usabilidad sencilla de aprender e intuitiva se tienen que realizar pruebas que consiste en hacer utilizar la aplicación a usuarios reales. En función del resultado de estas pruebas se podrán extraer conclusiones sobre si la aplicación tiene la interfaz adecuada y si es necesario realizar algún cambio en concreto.

Pruebas realizadas por los expertos

Inicialmente, debido a la limitación de tiempo de la que se dispone, se han realizado pruebas de usabilidad realizadas por los expertos. La primera prueba realizada por los expertos fue tras la creación inicial de los prototipos, en la que se extrajo como conclusión la forma en la que había que disponer los iconos de los contenedores con los cuales se iban a realizar las operaciones principales, y también se modificó ligeramente las pantallas de creación de nuevos contenedores. En la segunda prueba realizada, ya se había implementado en la aplicación casi todas las funcionalidades. Dado que se habían seguido los prototipos, mayormente no hubo que realizar ningún cambio de usabilidad, pero se hizo una aportación muy interesante, ponerles apodos a los distintos servidores que se gestionaban para así poder identificarles con mayor facilidad. En la última prueba realizada por los expertos, ya estaba completada toda la funcionalidad, y la única aportación que se realizó fue la de añadir botones en las pantallas para poder volver a la pantalla anterior sin tener que recurrir a la barra superior que tiene un acceso a la pantalla inicial.

Pruebas realizadas por usuarios

En la etapa final del proyecto se realiza una prueba de usabilidad con dos usuarios reales, en la cual se les va a proponer que realicen una serie de acciones para poder comprobar si la aplicación es fácil de utilizar. En la tabla 5.1 se muestra el formato que van a tener las anotaciones que se van a tomar sobre estas pruebas.

Tarea	**Tarea X**
Completado	Si/No
Valoración	De 0 a 10.
Descripción	Se describirá el procedimiento a seguir.
Métricas	• Tiempo dedicado a realizar la tarea. • Toques necesarios para realizar la tarea. • Errores que se han producido al realizar la tarea.
Observaciones	Comentarios adicionales.

Tabla 5.1: Plantilla para las pruebas de usabilidad con usuarios.

Las tareas que se van a realizar son las siguientes:

- Acceder a la aplicación. Iniciar sesión. Crear un servidor *MongoDB* en la Máquina 1 con el modo sencillo redirigiendo su puerto (27017) al 47047 al que se le dará por nombre *MongoPrueba*. Comprobar en la pantalla principal que todo se ha creado correctamente.
- Acceder a la aplicación. Iniciar sesión. Detener el contenedor *MongoPrueba*. Descargarse el contenido del contenedor alojado en */data/db*. Borrar el contenedor.

Una vez se tienen las tareas, se crean los escenarios de prueba que se van a enviar a los usuarios que realizan la prueba. Estos escenarios contienen el contexto de la aplicación y la tarea a realizar. Son los siguientes:

- Te han pedido que crees un nuevo contenedor de base de datos MongoDB. Para ello, debes acceder a la aplicación web con el usuario y contraseña que te han proporcionado, y crear ese contenedor en la máquina 1 utilizando el modo sencillo y la imagen de mongo en la versión 4.4.10. Para identificarlo del resto el contenedor debe llamarse *MongoPrueba*, y para poder acceder a la base de datos se necesita que el puerto de MongoDB (27017) se redirija al puerto 47047 de la máquina 1. No será necesario crear ningún volumen. Una vez se haya creado se debe asegurar que se ha realizado todo correctamente revisando la información del contenedor desde la pantalla principal.
- Han pasado dos meses desde la anterior tarea. Ya no se necesita la base de datos, pero por si fuera necesario en un futuro se quieren descargar los datos almacenados de la base de datos por si en un futuro se necesitaran para algo. Se debe acceder de nuevo al sistema. Se debe detener el contenedor que se creó en la tarea anterior. Se deben descargar los datos de la base de datos, que MongoDB almacena en la ruta */data/db*. Una vez descargados los datos, se debe borrar el contenedor.

Al finalizar estas dos tareas se va a preguntar al usuario lo siguiente, para conocer información adicional:

- ¿Te ha resultado intuitiva la interfaz?
- ¿En algún punto de las tareas no sabías como continuar?
- ¿El diseño de la aplicación, donde se muestran los contenedores te ha resultado adecuado?
- ¿Algo más que quieras añadir?

Para realizar la prueba se han escogido dos usuarios estudiantes de Ingeniería Informática, el primero con muy bajos conocimientos sobre *docker*, y el segundo con unos conocimientos entre bajos y medios.

Antes de iniciar la prueba, se ha explicado brevemente el contexto de la aplicación, para que sirve, etc. A continuación se ha iniciado la prueba. En el apéndice E se pueden ver las tablas con los resultados obtenidos.

Las conclusiones obtenidas de la prueba han sido que en general el diseño de la interfaz es el adecuado para usuarios que no tienen una amplia experiencia en *docker*. En parte, esto es así porque los expertos ya habían hecho sus comentarios en las etapas anteriores y se habían aplicado todos los cambios que habían propuesto. Ambos usuarios han sido capaces de completar con éxito las pruebas con bastante facilidad. Las principales conclusiones obtenidas han sido las dos siguientes:

- El botón de crear contenedores quizás debería revisarse. Ninguno ha tenido problema a la hora de usarlo, pero ambos han coincidido en que quizás debería revisarse.
- En general el diseño de la interfaz es bastante intuitivo, y se encuentran las opciones más o menos donde se esperaría.

Una vez realizada esta prueba se ha conseguido comprobar que la usabilidad de la aplicación es la correcta para los usuarios.

Capítulo 6

Conclusiones

En este capítulo final se exponen las ideas que se han recopilado de la realización de este proyecto a modo de conclusión. Adicionalmente se van a apuntar varias ideas de cara a mejorar el proyecto en un futuro.

Lo realizado en este proyecto ha sido una aplicación web (*frontend*) que permite gestionar contenedores *docker* instalando una aplicación (*backend* de docker) en todos los servidores sobre los que se quiera trabajar, y registrándolos en una aplicación principal (*backend* de autenticación) que tiene el registro de todos los usuarios y servidores del sistema. De esta forma se consigue que los usuarios con poco conocimiento de *docker* puedan trabajar con él. En concreto, todo el sistema se ha puesto a prueba en los servidores del grupo ECA-SIMM de la Universidad de Valladolid, desplegando varias aplicaciones de prueba.

Al iniciar este proyecto se partía de la idea de querer realizar una aplicación web para facilitar la gestión de contenedores *docker*. Para realizar esto, por un lado, se aprovecha que al realizar una aplicación web, se puede permitir gestionar contenedores de forma remota y desde cualquier dispositivo que tenga la capacidad de navegar en internet. Por otro lado, es necesario que la interfaz sea sencilla para usuarios que se estén iniciando en la gestión de contenedores y no tengan mucha idea.

Para la realización de la aplicación, se ha aprendido mucho sobre las tecnologías utilizadas en el proyecto, que han sido fundamentalmente tres: *Angular*, *node.js* y *docker*. La mayor parte del desarrollo de la aplicación se ha empleado en el aprendizaje de uso de estas tres plataformas (parte también con el aprendizaje de *dockerode*, que combina *node.js* y *docker*).

En el caso de *Angular*, el desarrollador solo había trabajado con esta tecnología una vez, y solo conocía muy por encima el funcionamiento de la misma. Durante las primeras etapas del desarrollo del *frontend* se avanzó lento, pero se aprendió mucho sobre la tecnología, lo que permitió que en las últimas etapas del desarrollo del proyecto esta parte no requirió de demasiado tiempo para ser realizada.

La tecnología de *node.js*, al igual que la anterior, apenas había sido utilizada por el desarrollador. A diferencia de *Angular*, este lenguaje es bastante sencillo de aprender para poder desarrollar cualquier cosa con él. Sin embargo, tiene un inconveniente, y es que dado que el desarrollador no era experimen-

tado, cuando se han producido errores en el desarrollo de la aplicación con esta tecnología, ha costado relativamente más de lo esperado solucionarlos.

La última tecnología de las mencionadas, *docker*, ya se conocía de las prácticas curriculares realizadas con el grupo ECA-SIMM. Sin embargo, la utilización de la misma mediante *dockerode* ha supuesto un aprendizaje casi completo de *docker-engine*, que es lo que permite manipular los contenedores desde fuera. Durante las prácticas se habían gestionado los contenedores mediante comandos, lo que ha hecho que se conociera bien el qué se quería hacer, pero ha necesitado de investigación para conocer el cómo.

Sobre el resto de tecnologías utilizadas, mayormente ya se conocía su funcionamiento y utilizado, como por ejemplo la base de datos *MongoDB*, o la aplicación *postman*, lo que ha permitido que cuando ha llegado el momento de utilizarlas no se ha tenido que perder tiempo en aprender el funcionamiento de las mismas.

Otro aspecto a destacar, es la planificación del proyecto. Este apartado en especial, sin ninguna duda, es de los más difíciles de realizar con exactitud, ya que calcular el tiempo que se va a tardar en desarrollar un proyecto del cual no se conoce bien ni la tecnología sobre la que se trabaja ni las posibilidades de cada una de ellas, provoca que haya que tener mucha flexibilidad con los tiempos para ir variando muchas partes según se van desarrollando.

En la parte de las pruebas, se puede observar como estas permiten obtener información de la aplicación que no es capaz de ver a simple vista el desarrollador. Esto incluye tanto los posibles errores que pueda tener la aplicación por errores de código o por simplemente no haber tenido en cuenta todas las posibilidades de parámetros de entrada o salida en una llamada. A mayores, las pruebas realizadas con usuarios permiten conocer puntos de vista externos al proyecto que permiten adecuar la interfaz a usuarios que no tienen mucho conocimiento sobre el sistema.

Por último, respecto a la parte personal de realizar este proyecto, la valoración del alumno es que la realización de este trabajo ha sido satisfactoria. Antes de la realización de las prácticas, sobre las que se ha realizado este TFG, el alumno no tenía conocimiento sobre *docker*, y apenas conocía aspectos muy básicos de *node.js* y *Angular*. Este TFG ha servido para adquirir un amplio conocimiento sobre tecnologías que son punteras y estas van a servir de gran utilidad en la vida laboral del alumno. Este proyecto también ha servido al alumno para aprender a gestionar situaciones de bloqueo en las cuales los problemas parecen no tener solución. Los tutores del proyecto han servido de gran ayuda para ayudar y dar solución a estos momentos. Sobre la utilización de la tecnología *docker* para desplegar aplicaciones, el alumno cree que la utilizará con frecuencia en el futuro, principalmente por la gestión de dependencias tan sencilla que ofrece. También es importante la portabilidad que proporciona *docker*, ya que elimina el temor a que una aplicación deje de funcionar al tener que migrarla de un ordenador a otro, lo que es otro argumento muy fuerte para decidir utilizarla.

6.1 Trabajo futuro

En este proyecto se han abordado todos los temas principales y más importantes de la gestión de contenedores *docker*. Sin embargo, siempre se pueden añadir funcionalidades, ya que se pueden cubrir muchas más funciones no tan principales. Además, muchas de ellas con la tecnología ya utilizada se podrían implementar sin añadir más dependencias ni requisitos. A continuación se listan una serie de ideas:

- Acceder de forma online a los ficheros de cada contenedor, para poder visualizarlos sin tener que descargarlos (al estilo *filezilla* [45]).
- Poder acceder a la consola de los contenedores en ejecución desde la propia web, sin tener que instalar ningún programa ni tener que conectarse directamente a la máquina donde están alojados.
- Mejorar de distintas formas la edición de algunas configuraciones de los contenedores, permitiendo modificar ciertos parámetros que actualmente solo son configurables en el momento de la creación de los mismos.
- Elaborar una interfaz online para poder crear contenedores complejos sin tener que subir directamente un fichero comprimido. Un modo 'intermedio' entre el modo sencillo y el avanzado a la hora de crear contenedores.
- Tener un acceso vía web a los *logs* de los contenedores, para poder conocer si está funcionando todo correctamente.

Apéndices

Apéndice A

El problema del CORS

Cuando se desarrolla una aplicación que tiene separado el *frontend* del *backend* nos va a ocurrir un problema que va a provocar que el *frontend* no pueda comunicarse con el *backend*. Si abrimos la consola del navegador podremos ver que el error tiene que ver con *CORS*. Esto ocurre tanto si ambas partes están desplegadas en una máquina como si lo están en una distinta.

El nombre de *CORS* significa *Cross-Origin Resource Sharing*, origen de recursos cruzado. Esto implica que cuando una petición llegue desde un *frontend* en lugar que desde un navegador directamente, se va a rechazar la petición y no se le va a dar respuesta. Esto es así por una política que han de seguir los servidores, lo cual afecta a este tipo de aplicaciones.

A causa de esta política, por defecto todas las peticiones que se realicen al *backend* quedarán bloqueadas, y por tanto, nuestra aplicación web quedará inservible. Después de realizar una investigación se ha encontrado como solucionar esto para servidores que utilizan *express* en *node.js* [46]. A continuación se indican los pasos:

- En primer lugar, necesitamos importar *CORS*. Esto se puede hacer editando el fichero *package.json* o ejecutando el siguiente comando: *npm i –save cors*.
- Una vez hecho esto, tenemos que irnos a la clase de nuestro programa que inicie el servidor, donde tenemos el *express*. En esta clase importamos *cors*.
- Finalmente solo tenemos que escribir la siguiente linea: *app.use(cors())*, siendo *app* la variable que contiene nuestro *express*.

Una vez hecho eso, habríamos solucionado este problema y a partir de este momento nuestro servidor responderá las peticiones vengan o no de un *frontend*.

Apéndice B

Manual de instalación

Se va a explicar como instalar todo el sistema que compone la aplicación web. Esto se compone de tres elementos:

- La aplicación *Angular*, que es la que se encarga del *frontend*.
- La aplicación de autenticación y gestión de servidores, que está escrita en *nodejs*.
- La aplicación que gestiona *docker*, que también está escrita en *nodejs*.

B.1 Instalación del frontend

Vamos a necesitar el proyecto del *frontend*, que es el que se encuentra en la carpeta *frontend* de la raíz del repositorio. Para que todo funcione van a ser necesarios una serie de requisitos.

B.1.1 Requisitos del entorno

Se va a trabajar sobre el S.O. *Ubuntu* 20.04, por lo que todas las rutas y referencias concretas podrán no coincidir en otro sistema operativo. En el sistema operativo sobre el que vamos a instalar el *frontend* tiene que tener instalado lo siguiente:

- Servidor *nginx* en la versión 1.18.0 o superior.
- *Node.js* en la versión 10.19.0.*

* Nota: Si ya disponemos del código compilado no es necesario este requisito.

B.1.2 Compilación del código

Si ya se dispone del código compilado se puede saltar este punto. En caso de disponer del código fuente, a continuación se detalla como se compila.

- Se debe abrir una terminal dentro del directorio *frontend*, que es la raíz de nuestro proyecto y donde se encuentran ficheros como *package.json* o directorios como *src* o *assets*.
- Se debe ejecutar el comando *ng build*.

Una vez se hayan completado los pasos anteriores, podremos ver que existe una carpeta llamada *dist* dentro de nuestra carpeta del proyecto.

B.1.3 Configuración del frontend

En este caso solo hay que configurar la url del servidor de autenticación que se indica como instalar en la siguiente sección.

- Dentro de la carpeta *assets*, que hay dentro de los ficheros compilados (carpeta *dist*), se debe abrir el fichero *urls.json*. Una vez dentro, en el único parámetro que hay se tiene que escribir la url del servidor de autenticación, incluyendo el puerto.

B.1.4 Despliegue del frontend

Finalmente, tenemos que copiar todos los ficheros que hay en la carpeta *dist* dentro de la ruta que tenga asignada nuestro *nginx*. Habitualmente esta ruta se encontrará en */var/www/html*. Si este es el caso podemos ejecutar los siguientes comandos:

- Nos situamos en la raíz de nuestro proyecto ejecutando *cd <ruta del proyecto>* (que es donde está nuestra carpeta *dist* y todo el código).
- Ejecutamos el comando *rm -rf /var/www/html* para borrar lo que hubiera en ese directorio.
- Y copiamos el contenido de angular en su lugar ejecutando *cp -r dist /var/www/html*.

Ahora simplemente necesitaremos iniciar el servicio de *nginx* (si no lo estaba ya) e ir en el navegador a la ruta que le tengamos configurada.

En este punto puede ocurrir que si recargamos la página desde una ruta que no es la raíz (como puede ser /login), el servicio nos devuelva un 404 diciendo que no se ha encontrado la ruta. Si esto nos sucede hay que hacer una última configuración para que todo funcione correctamente:

- Debemos abrir un fichero de configuración de *nginx*. Por defecto, este será el encontrado en */etc/nginx/sites-enabled/default*.
- Una vez dentro del fichero, deberemos buscar donde dice *location / {*. La siguiente línea no comentada debería empezar por *try_files*. Por defecto ésta es la línea 51 del archivo. Debe quedar escrito lo siguiente: *try_files $uri $uri/ /index.html?$query_string;*. Con esto se está indicando que si no encuentra la ruta, devuelva el fichero *index* con la ruta referida, que es lo que queremos que haga, ya que ese fichero tiene todas las rutas del proyecto.

En este punto toda la configuración del *frontend* se habría completado.

B.2 Instalación del backend de autenticación y gestión de servidores

El proyecto necesario esta vez es el *backend* de autenticación, que es el que se encuentra en la carpeta *Backend_Autenticacion* de la raíz del repositorio.

B.2.1 Requisitos del entorno

Se va a trabajar sobre el S.O. *Ubuntu* 20.04, por lo que todas las rutas y referencias concretas podrán no coincidir en otro sistema operativo. Las dependencias necesarias son las siguientes:

- *Node.js* en la versión 10.19.0.
- *MongoDB*, en este caso en la versión 3.6.8. Debe tener creado el usuario a utilizar en caso de que quiera proteger el acceso con una contraseña.

B.2.2 Compilación del código

Al tratarse de una aplicación escrita en *node.js* no requiere compilación.

B.2.3 Configuración del backend de autenticación

En este caso, todos los parámetros configurables se encuentran en el fichero *.env*. Tiene las siguientes variables para configurar:

- *PORT*: El puerto en el que va a ejecutarse este servidor. En el desarrollo del proyecto se ha utilizado el 3000.
- *SECRET*: La clave secreta que va a utilizarse para crear los *tokens* que se utilizan para la seguridad del sistema.

- *USER*: El usuario por defecto para acceder a la aplicación. Este usuario solo es válido cuando no existe ninguno en la base de datos.
- *PASS*: La contraseña del usuario por defecto de la aplicación.
- *MONGO*: La ruta donde se encuentra la base de datos de tipo *MongoDB*. Tiene el formato *mongodb://<servidor>/<database>*. Nota: En el servidor es necesario indicar "*:<puerto>*" en caso de que el servidor de la base de datos esté en un puerto distinto al por defecto (27017).

B.2.4 Despliegue del backend de autenticación

Finalmente, tenemos que copiar todos los ficheros del proyecto y guardarlos donde vayamos a desplegarlo. A continuación habría que ejecutar los siguientes comandos:

- Nos situamos en la raíz de nuestro proyecto abriendo la consola directamente en ese directorio o ejecutando *cd <ruta>*.
- Ejecutamos el comando *npm install* para instalar todas las dependencias. Este comando solo será necesario la primera vez, ya que una vez estén instaladas todas las dependencias no tendrá efecto.
- Ejecutamos el comando *npm start* para iniciar la aplicación. En este momento, la consola indicará que se está ejecutando en el puerto que se configuró anteriormente. Si se produce algún fallo al conectarse a la base de datos se notificará en este momento.

Ya tenemos funcionando la aplicación. Si configuramos la ruta de esta aplicación en el *frontend* ya podremos iniciar sesión en la aplicación y añadir servidores.

B.3 Instalación del backend de docker

Esta vez el proyecto necesario es el *backend* de docker, que es el que se encuentra en la carpeta *Backend_Docker* de la raíz del repositorio.

B.3.1 Requisitos del entorno

Se va a trabajar sobre el S.O. Ubuntu 20.04, por lo que todas las rutas y referencias concretas podrán no coincidir en otro sistema operativo. Las dependencias necesarias son las siguientes:

- *Node.js* en la versión 10.19.0.
- *Docker*, en la versión 20.10.2.
- *Docker-compose*, en la versión 1.25.0.

B.3.2 Compilación del código

Al tratarse de una aplicación escrita en *node.js* no requiere compilación.

B.3.3 Configuración del backend de docker

En este caso, todos los parámetros configurables se encuentran en el fichero *.env*. Tiene las siguientes variables para configurar:

- *PASS*: La contraseña de este servidor. Va a ser necesaria en el momento que se añada desde la aplicación la primera vez.
- *PORT*: El puerto en el que va a ejecutarse este servidor.

Debemos asegurar que existen las carpetas *uploads* y *uploadsfiles*, ya que son necesarias para el correcto funcionamiento de este servidor.

B.3.4 Despliegue del backend de docker

Finalmente, tenemos que copiar todos los ficheros del proyecto y guardarlos donde vayamos a desplegarlo. A continuación habría que ejecutar los siguientes comandos:

- Nos situamos en la raíz de nuestro proyecto abriendo la consola directamente en ese directorio o ejecutando *cd <ruta>*.
- Ejecutamos el comando *npm install* para instalar todas las dependencias. Este comando solo será necesario la primera vez, ya que una vez estén instaladas todas las dependencias no tendrá efecto.
- Ejecutamos el comando *npm start* para iniciar la aplicación. En este momento, la consola indicará que se está ejecutando en el puerto que se configuró anteriormente. Si se produce algún fallo al conectarse a la base de datos se notificará en este momento.

Ya tenemos funcionando la aplicación. A partir de este momento, se podrá añadir este servidor para gestionar contenedores desde la aplicación. Esta aplicación puede instalarse en todas las máquinas que se desee. En la web posteriormente, habría que añadirlas todas.

Apéndice C

Manual de usuario

En esta sección se va a explicar como utilizar la aplicación de forma general. Se detalla cada una de las pantallas.

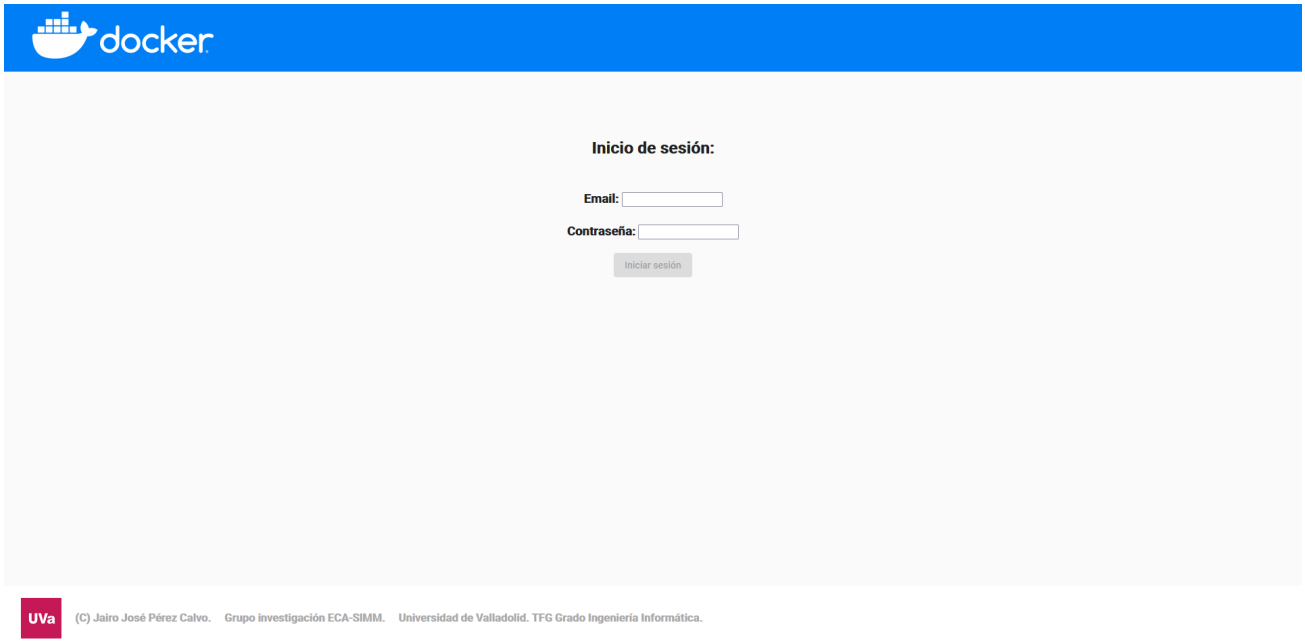


Figura C.1: Pantalla de login de la aplicación.

En la figura C.1 puede verse la pantalla de inicio de sesión de la aplicación. En esta pantalla lo único que se puede hacer es introducir el email y contraseña y darle al botón de iniciar sesión.

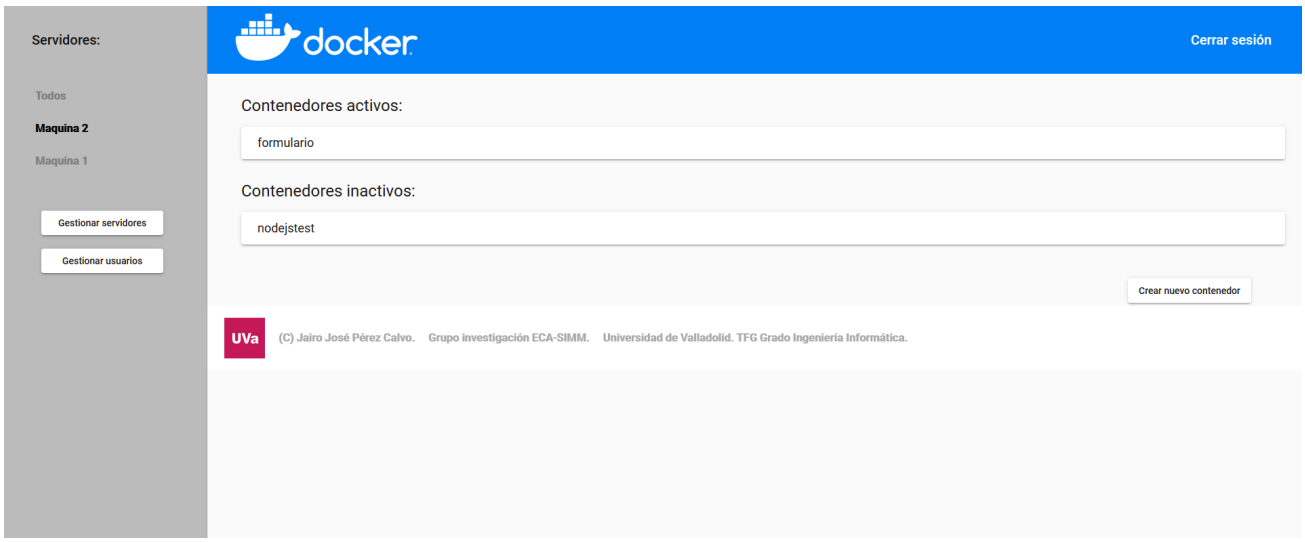


Figura C.2: Pantalla principal de la aplicación.

En la figura C.2 puede verse la pantalla principal de la aplicación. Se pueden ver los contenedores que existen, en la parte central y los servidores sobre los que se está gestionando, en el menú lateral izquierdo. En todo momento se encuentra arriba a la derecha el botón para cerrar sesión. Debajo de los contenedores está el botón para crear un nuevo contenedor, que para poder pulsar deberemos haber elegido un servidor concreto previamente. En el menú lateral, a la izquierda, se pueden ver los servidores existentes. Se puede hacer *click* en cualquiera de las opciones para seleccionarla. La opción seleccionada tiene el color e negro, mientras que el resto aparecerán en gris. Debajo se ven los botones de gestionar servidores y usuarios, en caso de que el usuario tenga los correspondientes permisos.

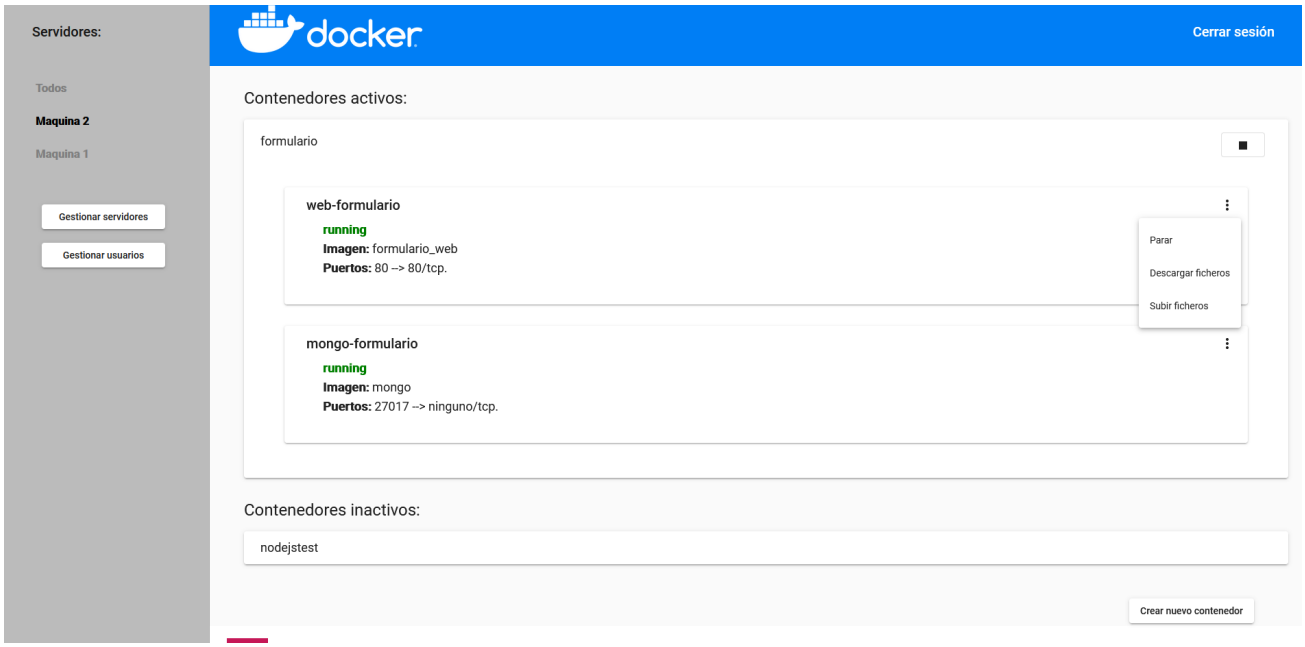


Figura C.3: Pantalla principal de la aplicación 2.

En la figura C.3 puede verse la pantalla principal de la aplicación con un contenedor activo desple-

gado. Se puede ver el botón de parar el conjunto de contenedores, con el icono de un cuadrado, arriba a la derecha del desplegable. Adicionalmente, en cada servicio que compone el contenedor, se puede detener de forma individual y se pueden subir y descargar ficheros de él.

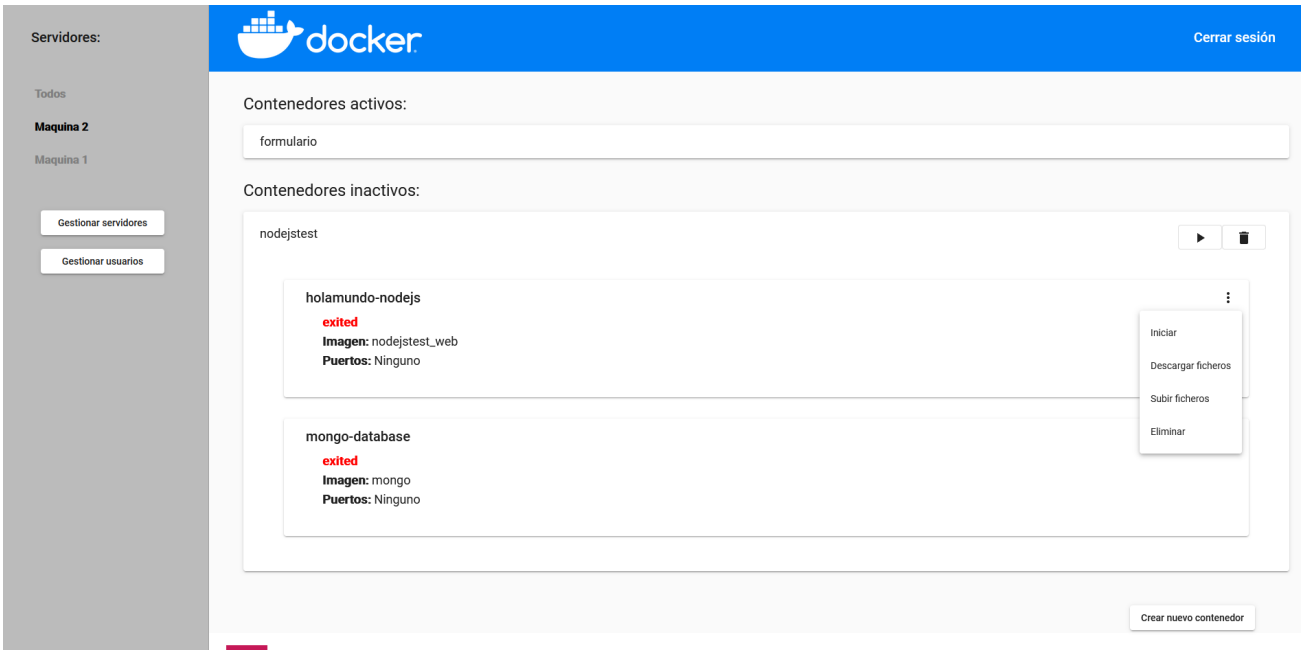


Figura C.4: Pantalla principal de la aplicación 3.

En la figura C.4 puede verse la pantalla principal de la aplicación con un contenedor inactivo desplegado. Se puede ver el botón de iniciar y borrar el conjunto de contenedores, con el icono de un triangulo y una papelera, respectivamente. Adicionalmente, en cada servicio que compone el contenedor, se puede iniciar de forma individual, borrar, y nuevamente se pueden subir y descargar ficheros de él.

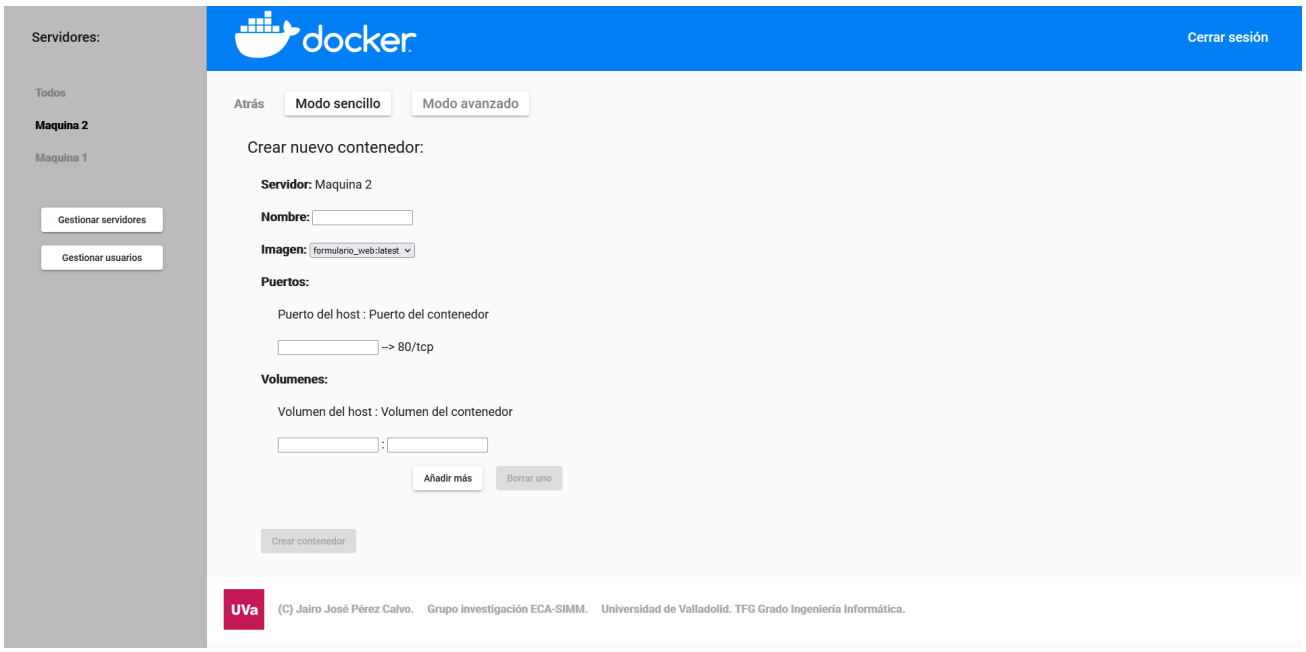


Figura C.5: Pantalla de agregar contenedores en modo sencillo.

En la figura C.5 puede verse la pantalla para agregar contenedores en modo sencillo. En esta pantalla, arriba se tiene acceso al modo avanzado y también se puede volver a la pantalla anterior, que es la principal. Para crear un contenedor se le debe poner nombre. Posteriormente se puede elegir una imagen del desplegable, momento a partir del cual se mostrarán los puertos disponibles para configurar. Finalmente se podrán configurar tantos volúmenes se quiera utilizando los botones de añadir más o borrar uno si se desean eliminar. Una vez se hayan rellenado todos los datos deseados se puede pulsar el botón de crear para finalizar la acción.

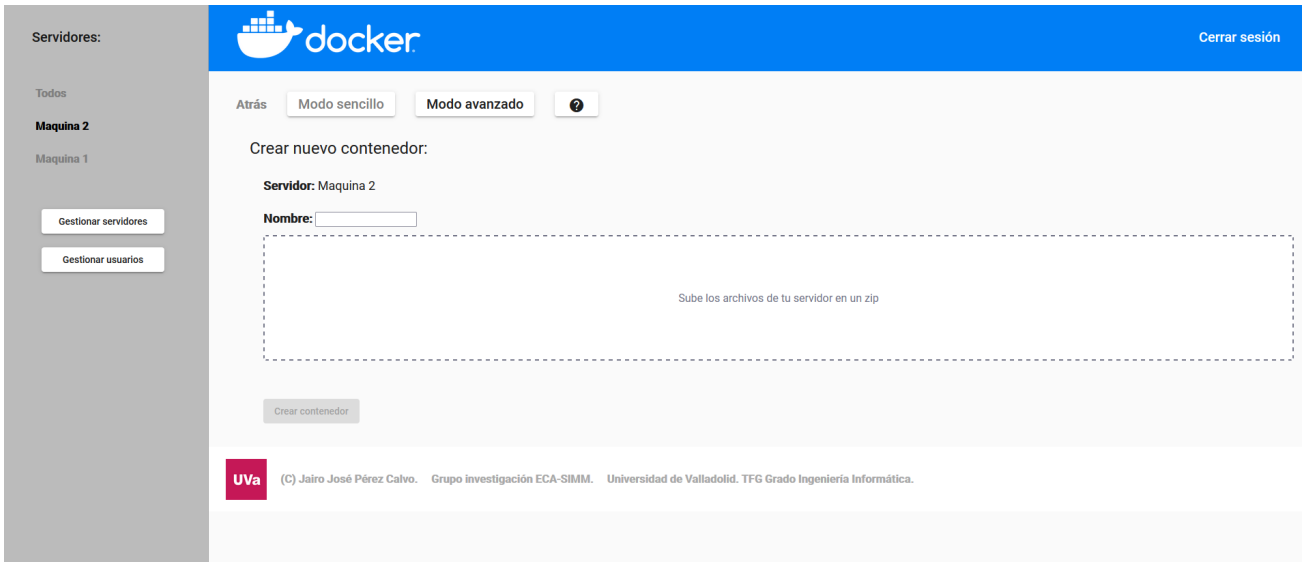


Figura C.6: Pantalla de agregar contenedores en modo avanzado.

En la figura C.6 puede verse la pantalla para agregar contenedores en modo avanzado. En esta pantalla también se tiene acceso al modo sencillo y se puede volver a la pantalla principal mediante

el botón de atrás. Adicionalmente, en el botón de al lado, se puede ver una ayuda respecto a como usar este modo. Para crear un contenedor se le debe poner nombre. Posteriormente se tiene que subir un fichero *zip* que contenga en la raíz el fichero *docker-compose.yml* en el que se indique toda la configuración deseada. Una vez se haya puesto nombre y subido el fichero se puede pulsar el botón de crear para finalizar la acción.

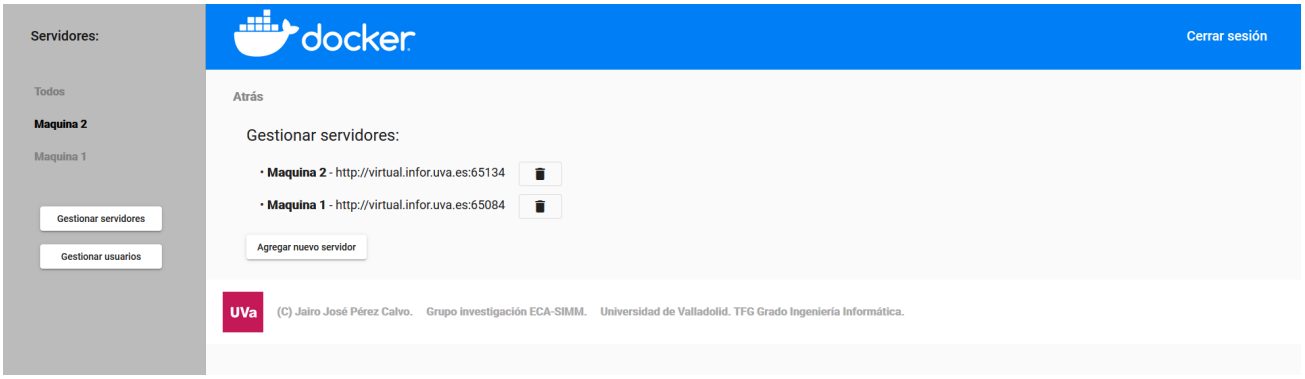


Figura C.7: Pantalla de gestión de servidores.

En la figura C.7 se encuentra la gestión de servidores. En la parte superior se muestran los servidores existentes, encabezados por su *nick*, y seguidos por su *url* y puerto. Se puede borrar uno pulsando el botón de la papelera de su derecha, y se pueden añadir más pulsando el botón de abajo.

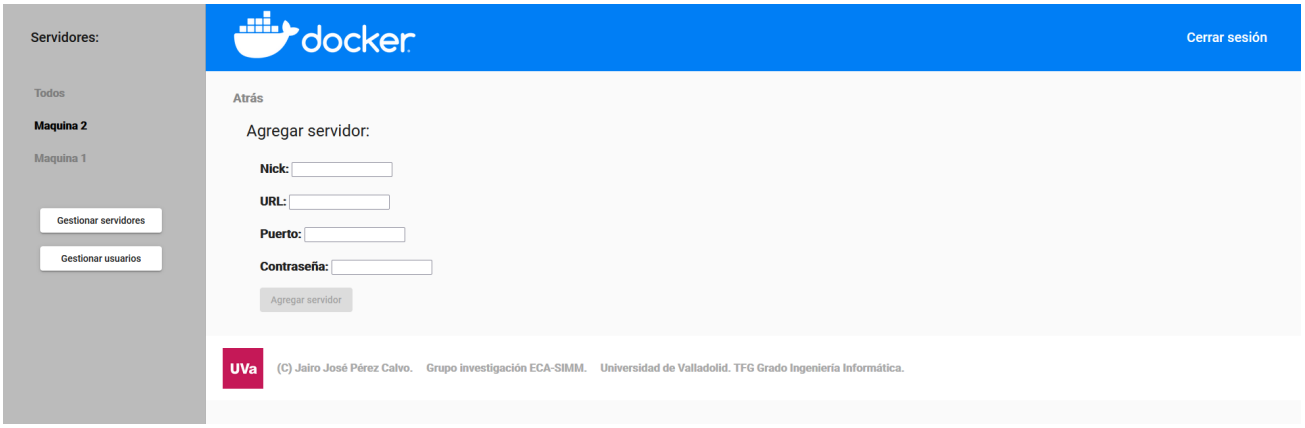


Figura C.8: Pantalla de agregar servidores.

En la figura C.8 se pueden añadir servidores. Se debe indicar un *nick*, que tiene que ser único en el sistema, una *url* y puerto (que corresponden a un servidor que también debe ser único en el sistema) y finalmente la contraseña que tiene ese servidor para poder conectarnos a él. Una vez se tengan todos los datos se puede dar al botón de agregar servidor para completar la acción.

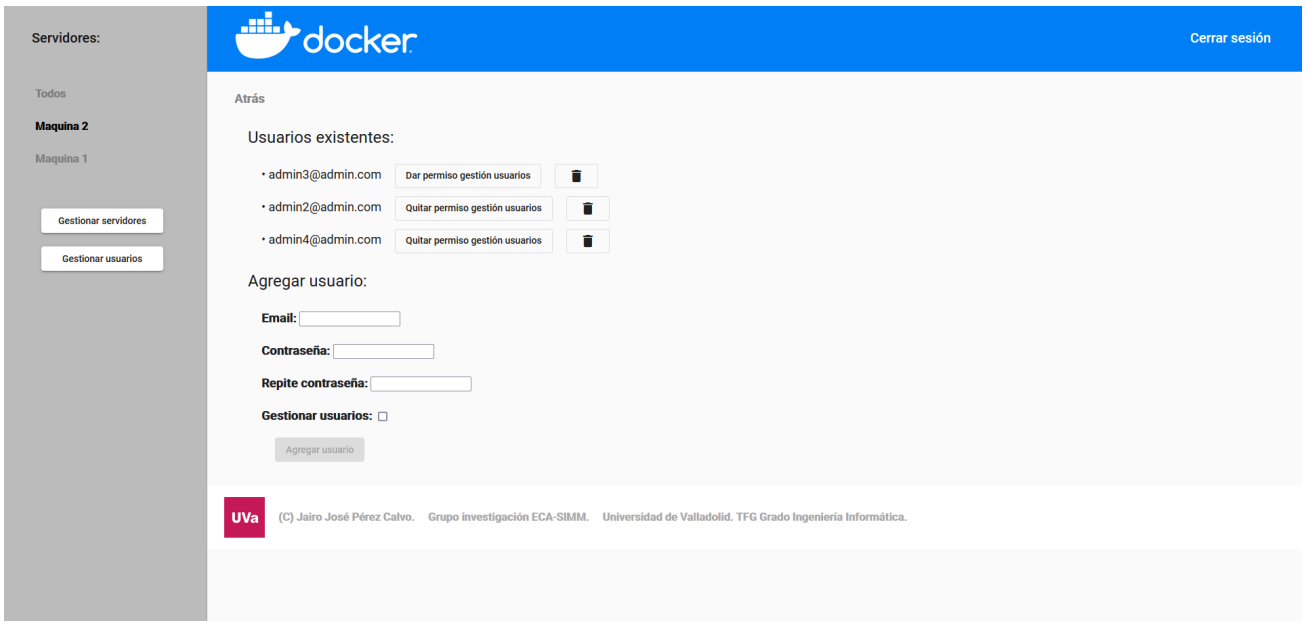


Figura C.9: Pantalla de gestionar usuarios.

En la figura C.9 se puede ver la pantalla de gestión de usuarios, la cual solo es accesible por los usuarios que tengan el permiso correspondiente. En la parte superior se ve la lista de usuarios existentes (sin incluir el de la sesión), y se puede cambiar el permiso de los usuarios deseados. También se puede borrar cualquiera de ellos. En la parte inferior se puede agregar un nuevo usuario, introduciendo su *email*, una contraseña válida que hay que repetir dos veces, y finalmente si ese usuario va a tener permisos de gestión de usuarios. Pulsando el botón de abajo se completaría la acción.

Apéndice D

Pruebas realizadas en el backend

Este apéndice está dedicado a mostrar todas las pruebas que se han realizado para comprobar el correcto funcionamiento del *backend* mediante la aplicación *postman*.

A continuación se incluyen todas las pruebas realizadas relativas al servidor de autenticación y gestión de servidores. Todas las llamadas excepto la de inicio de sesión (correspondiente a las tres primeras pruebas) contienen una cabecera que se llama *Authorization*, y que incluye el token del usuario para cada llamada, para así verificar que tiene permisos para realizarlo.

Id	PB-01
Descripción	Iniciar sesión en el sistema correctamente.
Url	/login
Método	POST

Entrada	{
	”email”: ”admin2@admin.com”,
	”pass”: ”******”
	}

Código de respuesta	200

Resultado	{
	”token”: ”eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2MGY4NDFjZWMwZGRkMzViM2M5ZjBkYTkiLCJwZXJtaXNvIjoyLCJlbWFpbCI6ImFkbWluMkBhZG1pbi5jb20iLCJpYXQiOjE2MzcwODMyNTcsImV4cCI6MTYzNzA4Njg1N30.7pJN-fb9zV8Qqfjl_lNHmHJ61nL4CJrsi9s-1a_Omdw”,
	”permiso”: 2
	}

Tabla D.1: PB-01

Id	PB-02
Descripción	Iniciar sesión en el sistema con un email incorrecto.
Url	/login
Método	POST

Entrada	{
	”email”: ”admininexistente@admin.com”,
	”pass”: ”**********”
	}

Código de respuesta	401
Resultado	Ese usuario no existe.

Tabla D.2: PB-02

Id	PB-03
Descripción	Iniciar sesión en el sistema con una contraseña erronea.
Url	/login
Método	POST

Entrada	{
	”email”: ”admin2@admin.com”,
	”pass”: ”**********”
	}

Código de respuesta	401
Resultado	Contraseña erronea.

Tabla D.3: PB-03

Id	PB-04
Descripción	Comprobar token correcto.
Url	/checktoken
Método	GET
Entrada	-
Código de respuesta	200
Resultado	Ok

Tabla D.4: PB-04

Id	PB-05
Descripción	Comprobar token incorrecto.
Url	/checktoken
Método	GET
Entrada	-
Código de respuesta	401
Resultado	Error con el token.

Tabla D.5: PB-05

Id	PB-06
Descripción	Ver la lista de servidores del sistema.
Url	/maquinas
Método	GET
Entrada	-
Código de respuesta	200

Resultado	[
	{
	”_id”: ”614ca57ae9782f25bf17050c”,
	”nick”: ”Maquina 2”,
	”url”: ”http://virtual.infor.uva.es”,
	”port”: 65134,
	”__v”: 0
	},
	{
	”_id”: ”616f09fa09c9900440725262”,
	”nick”: ”Maquina 1”,
	”url”: ”http://virtual.infor.uva.es”,
	”port”: 65084,
	”__v”: 0
	},
	{
	”_id”: ”6193f9f26cb287e938b1f0f2”,
	”nick”: ”Maquina 11”,
	”url”: ”virtual.infor.uva.es”,
	”port”: 65084,
	”__v”: 0
	}
	]

Tabla D.6: PB-06

Id	PB-07
Descripción	Registrar servidor.
Url	/registerServer
Método	POST

Entrada	{
	”nick”: ”Maquina Nueva”,
	”url”: ”virtual.infor.uva.es”,
	”port”: ”61084”
	}

Código de respuesta	200
Resultado	Servidor registrado.

Tabla D.7: PB-07

Id	PB-08
Descripción	Registrar servidor existente en nick.
Url	/registerServer
Método	POST

Entrada	{
	”nick”: ”Maquina 1”,
	”url”: ”virtual.infor.uva.es”,
	”port”: ”65084”
	}

Código de respuesta	400
Resultado	Ya existe ese nick.

Tabla D.8: PB-08

Id	PB-09
Descripción	Registrar servidor ya existente por url y puerto.
Url	/registerServer
Método	POST

Entrada	{
	”nick”: ”Maquina Nueva”,
	”url”: ”virtual.infor.uva.es”,
	”port”: ”65084”
	}

Código de respuesta	400
Resultado	Ya se ha agregado ese servidor.

Tabla D.9: PB-09

Id	PB-10
Descripción	Borrar servidor.
Url	/deleteServer
Método	DELETE

Entrada	{
	”url”: ”virtual.infor.uva.es”,
	”port”: ”65084”
	}

Código de respuesta	200
Resultado	Servidor borrado.

Tabla D.10: PB-10

Id	PB-11
Descripción	Ver la lista de usuarios.
Url	/listaUsuarios
Método	GET
Entrada	-
Código de respuesta	200

Resultado	[
	{
	”email”: ”admin3@admin.com”,
	”permiso”: 1,
	},
	{
	”email”: ”admin2@admin.com”,
	”permiso”: 2,
	},
	{
	”email”: ”admin@admin.com”,
	”permiso”: 2,
	}
	]

Tabla D.11: PB-11

Id	PB-12
Descripción	Ver la lista de usuarios sin tener permiso.
Url	/listaUsuarios
Método	GET
Entrada	-
Código de respuesta	401
Resultado	Ese usuario no está autorizado para hacer eso.

Tabla D.12: PB-12

Id	PB-13
Descripción	Crear usuario.
Url	/crearUsuario
Método	POST

Entrada	{
	”email”: ”admin44@admin.com”,
	”pass”: ”asfafasf”,
	”permiso”: 1
	}

Código de respuesta	201
Resultado	Usuario creado.

Tabla D.13: PB-13

Id	PB-14
Descripción	Crear usuario ya existente.
Url	/crearUsuario
Método	POST

Entrada	{
	”email”: ”admin3@admin.com”,
	”pass”: ”asfafasf”,
	”permiso”: 1
	}

Código de respuesta	409
Resultado	Ese email ya existe.

Tabla D.14: PB-14

Id	PB-15
Descripción	Crear usuario sin permisos.
Url	/crearUsuario
Método	POST

Entrada	{
	”email”: ”admin45@admin.com”,
	”pass”: ”asfafasf”,
	”permiso”: 1
	}

Código de respuesta	401
Resultado	Ese usuario no está autorizado para hacer eso.

Tabla D.15: PB-15

Id	PB-16
Descripción	Cambiar permiso de usuario.
Url	/cambiarPermiso
Método	PUT

Entrada	{
	”email”: ”admin44@admin.com”,
	}

Código de respuesta	200
Resultado	Permiso editado.

Tabla D.16: PB-16

Id	PB-17
Descripción	Cambiar permiso de usuario email erroneo.
Url	/cambiarPermiso
Método	PUT

Entrada	{
	”email”: ”adminnoexistente@admin.com”,
	}

Código de respuesta	406
Resultado	Ese usuario no existe.

Tabla D.17: PB-17

Id	PB-18
Descripción	Cambiar permiso de usuario sin tener permisos.
Url	/cambiarPermiso
Método	PUT

Entrada	{
	”email”: ”admin44@admin.com”,
	}

Código de respuesta	200
Resultado	Ese usuario no está autorizado para hacer eso.

Tabla D.18: PB-18

Id	PB-19
Descripción	Borrar usuario.
Url	/borrarUsuario
Método	DELETE

Entrada	{
	”email”: ”admin44@admin.com”,
	}

Código de respuesta	200
Resultado	Usuario borrado.

Tabla D.19: PB-19

Id	PB-20
Descripción	Borrar usuario sin tener permisos.
Url	/borrarUsuario
Método	DELETE

Entrada	{
	”email”: ”admin44@admin.com”,
	}

Código de respuesta	401
Resultado	Ese usuario no está autorizado para hacer eso.

Tabla D.20: PB-20

A continuación se incluyen todas las pruebas realizadas a la aplicación *backend* de gestión de contenedores *docker*. Al igual que todas las pruebas anteriores, todas las llamadas contienen una cabecera que se llama *Authorization*, y que incluye el token del usuario para cada llamada, para así verificar que tiene permisos para realizarlo. En este caso, la única excepción es el registro del servidor al *backend* de gestión de servidores y autenticación, que corresponde a las pruebas 21 y 22.

Id	PB-21
Descripción	Registrar servidor en el servidor de gestión de servidores y autenticación.
Url	/register
Método	POST

Entrada	{
	”host”: ”http://virtual.infor.uva.es”,
	”port”: ”65085”,
	”pass”: ”*****”,
	}

Código de respuesta	200

Resultado	{
	”Operacion”: ”Aceptada”,
	}

Tabla D.21: PB-21

Id	PB-22
Descripción	Registrar servidor con contraseña errónea.
Url	/register
Método	POST

Entrada	{
	”host”: ”http://virtual.infor.uva.es”,
	”port”: ”65085”,
	”pass”: ”**********”,
	}

Código de respuesta	200
Resultado	La contraseña no es válida.

Tabla D.22: PB-22

Id	PB-23
Descripción	Borrar el servidor de gestión de servidores en el que se encuentra.
Url	/unregister
Método	DELETE
Entrada	-
Código de respuesta	200

Resultado	{
	”Operacion”: ”Aceptada”,
	}

Tabla D.23: PB-23

Id	PB-24
Descripción	Obtener la lista de contenedores de una máquina.
Url	/
Método	GET
Entrada	-
Código de respuesta	200

Resultado	[
	{
	”Id”: ”66a63b4a1ed91bb2a10f0d202a526858bdc76c...”,
	”Names”: [
	”/Mongo”
	],
	”Image”: ”sha256:ca8e14b1fda68aedb435fec2a6eaa...”,
	”ImageID”: ”sha256:ca8e14b1fda68aedb435fec2a6...”,
	”Command”: ”docker-entrypoint.sh mongod”,
	”Created”: 1633369729,
	”Ports”: [
	{
	”PrivatePort”: 27017,
	”Type”: ”tcp”
	}
	],
	”Labels”: {},
	”State”: ”running”,
	”Status”: ”Up 3 weeks”,
	”HostConfig”: {
	”NetworkMode”: ”default”
	},
	”NetworkSettings”: {
	”Networks”: {
	”bridge”: {
	”NetworkID”: ”bffe2ae62ba9bfc4184f3def...”,
	”EndpointID”: ”a7a69622457fb3147462a8a...”,
	”Gateway”: ”172.17.0.1”,
	”IPAddress”: ”172.17.0.3”,
	”IPPrefixLen”: 16
	}
	}
	},
	”Mounts”: [
	{
	”Type”: ”volume”,
	”Name”: ”12a953a71ef7bb5282f35570611d6bfa...”,
	”Source”: ””,
	”Destination”: ”/data/configdb”,
	”Driver”: ”local”,
	”RW”: true
	}
	]
	}
	]

Tabla D.24: PB-24

Id	PB-25
Descripción	Encender contenedor.
Url	/activar
Método	POST
Entrada	Query param: id=ContenedorName
Código de respuesta	200

Resultado	{
	”Operacion”: ”Aceptada”
	}

Tabla D.25: PB-25

Id	PB-26
Descripción	Encender contenedor no existente.
Url	/activar
Método	POST
Entrada	Query param: id=ContenedorName
Código de respuesta	200

Resultado	{
	”Operacion”: ”Rechazada”,
	”Mensaje”: ”No such container: ContenedorName”,
	}

Tabla D.26: PB-26

Id	PB-27
Descripción	Encender contenedor que ya está encendido.
Url	/activar
Método	POST
Entrada	Query param: id=ContenedorName
Código de respuesta	200

Resultado	{
	”Operacion”: ”Rechazada”
	}

Tabla D.27: PB-27

Id	PB-28
Descripción	Detener contenedor.
Url	/desactivar
Método	POST
Entrada	Query param: id=ContenedorName
Código de respuesta	200

Resultado	{
	”Operacion”: ”Aceptada”
	}

Tabla D.28: PB-28

Id	PB-29
Descripción	Detener contenedor no existente.
Url	/desactivar
Método	POST
Entrada	Query param: id=ContenedorName
Código de respuesta	200

Resultado	{
	”Operacion”: ”Rechazada”,
	”Mensaje”: ”No such container: ContenedorName”,
	}

Tabla D.29: PB-29

Id	PB-30
Descripción	Detener contenedor que no está encendido.
Url	/desactivar
Método	POST
Entrada	Query param: id=ContenedorName
Código de respuesta	200

Resultado	{
	”Operacion”: ”Rechazada”
	}

Tabla D.30: PB-30

Id	PB-31
Descripción	Crear nuevo contenedor sencillo.
Url	/crearContenedor
Método	POST
Entrada	Query params: imagen=mongo nombre=Puerto3000 puertos1=3000 puertos2=27017/tcp volumenes=-
Código de respuesta	200

Resultado	{
	”Operacion”: ”Aceptada”
	}

Tabla D.31: PB-31

Id	PB-32
Descripción	Crear nuevo contenedor sencillo con parámetros erróneos.
Url	/crearContenedor
Método	POST
Entrada	Query params: imagen=mongonoexiste nombre=Puerto3000 puertos1=2223000 puertos2=27017/tcp volumenes=-
Código de respuesta	200

Resultado	{
	”Operacion”: ”Rechazada”
	}

Tabla D.32: PB-32

Id	PB-33
Descripción	Crear nuevo contenedor avanzado.
Url	/subir-contenedor
Método	POST
Entrada	Body form data: nombre=Puerto3000 file=Zip del servidor a crear.
Código de respuesta	200

Resultado	{
	”Operacion”: ”Aceptada”
	}

Tabla D.33: PB-33

Id	PB-34
Descripción	Crear nuevo contenedor avanzado con error.
Url	/subir-contenedor
Método	POST
Entrada	Body form data: nombre=Puerto3000 file=Zip erróneo del servidor a crear.
Código de respuesta	200

Resultado	{
	”Operacion”: ”Rechazada”
	}

Tabla D.34: PB-34

Id	PB-35
Descripción	Borrar contenedor.
Url	/borrar
Método	DELETE
Entrada	Query param: id=ContenedorName
Código de respuesta	200

Resultado	{
	”Operacion”: ”Aceptada”
	}

Tabla D.35: PB-35

Id	PB-36
Descripción	Borrar contenedor no existente.
Url	/borrar
Método	DELETE
Entrada	Query param: id=ContenedorName
Código de respuesta	200

Resultado	{
	”Operacion”: ”Rechazada”,
	”Mensaje”: ”No such container: ContenedorName”,
	}

Tabla D.36: PB-36

Id	PB-37
Descripción	Ver las imagenes existentes en un servidor.
Url	/imagenes
Método	GET
Entrada	-
Código de respuesta	200

Resultado	[
	{
	”Containers”: -1,
	”Created”: 1612438493,
	”Id”: ”sha256:54da9a218a9848403cbd5a326c7bc...”,
	”Labels”: null,
	”ParentId”: ”sha256:0257a9cb76e5f9434c2410c...”,
	”RepoDigests”: null
	”RepoTags”: [
	”formulario_web:latest”
	],
	”SharedSize”: -1,
	”Size”: 199201931,
	”VirtualSize”: 199201931
	}
	]

Tabla D.37: PB-37

Id	PB-38
Descripción	Ver los detalles de una imagen.
Url	/detalleimagen
Método	GET
Entrada	Query param: imagen=mongo
Código de respuesta	200

Resultado	{
	 ”Id”: ”sha256:54da9a218a9848403cbd5a326c7bc...”,
	 ”RepoTags”: [
	 ”formulario_web:latest”
	],
	 ”RepoDigests”: [],
	 ”ParentId”: ”sha256:0257a9cb76e5f9434c2410c...”,
	 ”Created”: ”2021-02-04T11:34:53.670416852Z”
	 ”Container”: ”d29108082ce5b2f329fc1e1dd9fe5...”,
	 ”ContainerConfig”: {
	 ”Hostname”: ”d29108082ce5”
	 ”ExposedPorts”: {
	 ”80/tcp”: {}
	 }
	 ”Env”: [
	 ”PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin”,
	 ”NODE_VERSION=9.11.2”,
	 ”YARN_VERSION=1.5.1”
	]
	 ”Cmd”: [
	 ”/bin/sh”,
	 ”-c”,
	 ”#(nop) ”,
	 ”CMD [\”npm\” \”start\”]”
	]
	 ”Image”: ”sha256:0257a76e5f9434c2410c940c2e5...”
	 ”WorkingDir”: ”/app”
	 }
	 ”DockerVersion”: 19.03.8
	 ...
	 }

Tabla D.38: PB-38

Id	PB-39
Descripción	Ver los detalles de una imagen inexistente.
Url	/detalleimagen
Método	GET
Entrada	Query param: imagen=mongonoexiste
Código de respuesta	200
Resultado	null

Tabla D.39: PB-39

Id	PB-40
Descripción	Descargar ficheros.
Url	/descargar-ficheros
Método	GET
Entrada	Query params: imagen=web imagen=/a.txt
Código de respuesta	200
Resultado	.tar file

Tabla D.40: PB-40

Id	PB-41
Descripción	Descargar ficheros contenedor no existe.
Url	/descargar-ficheros
Método	GET
Entrada	Query params: id=webnoexiste path=/a.txt
Código de respuesta	404
Resultado	no such container

Tabla D.41: PB-41

Id	PB-42
Descripción	Descargar ficheros no existentes.
Url	/descargar-ficheros
Método	GET
Entrada	Query params: id=web path=/noexiste.txt
Código de respuesta	404
Resultado	no such file

Tabla D.42: PB-42

Id	PB-43
Descripción	Subir fichero.
Url	/subir-ficheros
Método	POST
Entrada	Body as form data: file=El fichero .tar nombre=a.tar id=web path=/
Código de respuesta	200

Resultado	{
	”Operacion”: ”Aceptada”
	}

Tabla D.43: PB-43

Id	PB-44
Descripción	Subir fichero a contenedor inexistente.
Url	/subir-ficheros
Método	POST
Entrada	Body as form data: file=El fichero .tar nombre=a.tar id=webnoexiste path=/
Código de respuesta	404

Resultado	{
	”Operacion”: ”Rechazada”
	”Mensaje”: ”no such container”
	}

Tabla D.44: PB-44

Id	PB-45
Descripción	Ejecutar prune para limpiar los contenedores no utilizados.
Url	/prune
Método	POST
Entrada	-
Código de respuesta	200

Resultado	{
	”Operacion”: ”Aceptada”
	”Mensaje”: null
	}

Tabla D.45: PB-45

Apéndice E

Pruebas de usabilidad con usuarios

En este apéndice se muestran los resultados obtenidos por los dos usuarios que han realizado las pruebas de usabilidad.

E.1 Primer usuario

En las tablas E.1, E.2 y E.3 se incluyen las tareas asociadas a la primera prueba que se ha realizado.

Tarea	**Tarea 1**
Completado	Si
Valoración	10
Descripción	Inicio de sesión en la aplicación.
Métricas	• Tiempo dedicado a realizar la tarea: 10 segundos. • Toques necesarios para realizar la tarea (mínimos 3): 3. • Errores que se han producido al realizar la tarea: Ninguno.
Observaciones	Ninguna.

Tabla E.1: Primera tarea del primer usuario.

Tarea	**Tarea 2**
Completado	Si
Valoración	8
Descripción	Creación del contenedor de MongoDB.
Métricas	• Tiempo dedicado a realizar la tarea: 1 minuto y 30 segundos. • Toques necesarios para realizar la tarea (mínimos 6): 6. • Errores que se han producido al realizar la tarea: 0.
Observaciones	Al ser la primera vez que se realiza la tarea se ha realizado un poco mas despacio de lo esperado.

Tabla E.2: Segunda tarea del primer usuario.

Tarea	**Tarea 3**
Completado	Si
Valoración	10
Descripción	Comprobación de que se ha creado correctamente el contenedor con los parámetros indicados.
Métricas	• Tiempo dedicado a realizar la tarea: 10 segundos. • Toques necesarios para realizar la tarea (mínimos 1): 1. • Errores que se han producido al realizar la tarea: Ninguno.
Observaciones	Ninguna.

Tabla E.3: Tercera tarea del primer usuario.

En las tablas E.4, E.5 y E.6 se incluyen las tareas asociadas a la segunda prueba que se ha realizado.

Tarea	**Tarea 4**
Completado	Si
Valoración	9
Descripción	Detención del contenedor creado en la tarea 2.
Métricas	• Tiempo dedicado a realizar la tarea: 5 segundos. • Toques necesarios para realizar la tarea (mínimos 2): 3. • Errores que se han producido al realizar la tarea: Ninguno.
Observaciones	Se ha realizado desde el botón que hay dentro del menú en lugar de utilizar el que hay fuera.

Tabla E.4: Cuarta tarea del primer usuario.

Tarea	**Tarea 5**
Completado	Si
Valoración	10
Descripción	Descarga de los ficheros de la ruta */data/db* del contenedor.
Métricas	• Tiempo dedicado a realizar la tarea: 20 segundos. • Toques necesarios para realizar la tarea (mínimos 5): 5. • Errores que se han producido al realizar la tarea: Ninguno.
Observaciones	Ninguna.

Tabla E.5: Quinta tarea del primer usuario.

Tarea	**Tarea 6**
Completado	Si
Valoración	9
Descripción	Borrado del contenedor creado en la tarea 2.
Métricas	• Tiempo dedicado a realizar la tarea: 10 segundos. • Toques necesarios para realizar la tarea (mínimos 3): 4. • Errores que se han producido al realizar la tarea: Ninguno.
Observaciones	Se ha vuelto a utilizar el botón del menú en lugar del que se encuentra fuera.

Tabla E.6: Sexta tarea del primer usuario.

Tras realizar las tareas se realizaron las preguntas. Las respuestas fueron:

- ¿Te ha resultado intuitiva la interfaz? Sí.
- ¿En algún punto de las tareas no sabías como continuar? No.
- ¿El diseño de la aplicación, donde se muestran los contenedores te ha resultado adecuado? Sí.
- ¿Algo más que quieras añadir? Quizás el botón de crear contenedores estaría más visible más a la izquierda.

E.2 Segundo usuario

En las tablas E.7, E.8 y E.9 se incluyen las tareas asociadas a la primera prueba que se ha realizado.

Tarea	**Tarea 1**
Completado	Si
Valoración	10
Descripción	Inicio de sesión en la aplicación.
Métricas	• Tiempo dedicado a realizar la tarea: 10 segundos. • Toques necesarios para realizar la tarea (mínimos 3): 3. • Errores que se han producido al realizar la tarea: Ninguno.
Observaciones	Ninguna.

Tabla E.7: Primera tarea del segundo usuario.

Tarea	**Tarea 2**
Completado	Si
Valoración	8
Descripción	Creación del contenedor de MongoDB.
Métricas	• Tiempo dedicado a realizar la tarea: 1 minuto. • Toques necesarios para realizar la tarea (mínimos 8): 6. • Errores que se han producido al realizar la tarea: 0.
Observaciones	Ha pulsado el botón de crear contenedor sin elegir servidor primero, pero ha entendido el mensaje que sale al pulsarlo y ha sabido realizarlo de forma correcta con bastante agilidad.

Tabla E.8: Segunda tarea del segundo usuario.

Tarea	**Tarea 3**
Completado	Si
Valoración	10
Descripción	Comprobación de que se ha creado correctamente el contenedor con los parámetros indicados.
Métricas	• Tiempo dedicado a realizar la tarea: 10 segundos. • Toques necesarios para realizar la tarea (mínimos 1): 1. • Errores que se han producido al realizar la tarea: Ninguno.
Observaciones	Ninguna.

Tabla E.9: Tercera tarea del segundo usuario.

En las tablas E.10, E.11 y E.12 se incluyen las tareas asociadas a la segunda prueba que se ha realizado.

Tarea	**Tarea 4**
Completado	Si
Valoración	9
Descripción	Detención del contenedor creado en la tarea 2.
Métricas	• Tiempo dedicado a realizar la tarea: 5 segundos. • Toques necesarios para realizar la tarea (mínimos 2): 3. • Errores que se han producido al realizar la tarea: Ninguno.
Observaciones	Se ha realizado desde el botón que hay dentro del menú en lugar de utilizar el que hay fuera.

Tabla E.10: Cuarta tarea del segundo usuario.

Tarea	**Tarea 5**
Completado	Si
Valoración	10
Descripción	Descarga de los ficheros de la ruta */data/db* del contenedor.
Métricas	• Tiempo dedicado a realizar la tarea: 15 segundos. • Toques necesarios para realizar la tarea (mínimos 5): 5. • Errores que se han producido al realizar la tarea: Ninguno.
Observaciones	Ninguna.

Tabla E.11: Quinta tarea del segundo usuario.

Tarea	**Tarea 6**
Completado	Si
Valoración	9
Descripción	Borrado del contenedor creado en la tarea 2.
Métricas	• Tiempo dedicado a realizar la tarea: 10 segundos. • Toques necesarios para realizar la tarea (mínimos 3): 4. • Errores que se han producido al realizar la tarea: Ninguno.
Observaciones	Se ha vuelto a utilizar el botón del menú en lugar del que se encuentra fuera.

Tabla E.12: Sexta tarea del segundo usuario.

Tras realizar las tareas se realizaron las preguntas. Las respuestas fueron:

- ¿Te ha resultado intuitiva la interfaz? Sí.
- ¿En algún punto de las tareas no sabías como continuar? No.
- ¿El diseño de la aplicación, donde se muestran los contenedores te ha resultado adecuado? Sí.
- ¿Algo más que quieras añadir? Se podría poner un pop-up para elegir máquina cuando pulsas el botón de crear contenedor sin haber elegido una máquina.

Apéndice F

Acceso a los contenidos del TFG

Tanto el software desarrollado, como la memoria y resto de documentación técnica asociada a este Trabajo de Fin de Grado, se ha **depositado en un repositorio GitLab de acceso público, cuya URL se proporciona a continuación**.

`https://gitlab.inf.uva.es/eca-simm/dock4all`

De dicho repositorio se pueden descargar:

- **FrontEnd**: Ficheros de *Angular* que forman el *frontend*. Una vez compilado, los ficheros resultantes deben colocarse en el servidor *nginx* que vaya a alojar esta parte de la aplicación.
- **BackEnd_Docker**: Ficheros de *node.js* que forman el servidor de *backend* que se debe ejecutar en todas las máquinas en las que se gestionen contenedores.
- **BackEnd_Autenticacion**: Ficheros de *node.js* que forman el servidor de *backend* de autenticación y gestión de servidores.

Si se precisa más información o desea acceder a ejemplos adicionales de contenedores, contacte con `gir.ecasimm@uva.es`.

Bibliografía

[1] Docker. Visitado: 2021-04-05. [Online]. Available: https://www.docker.com/

[2] Docker hub. Visitado: 2021-04-05. [Online]. Available: https://hub.docker.com/

[3] Docker docs. Visitado: 2021-04-05. [Online]. Available: https://docs.docker.com/

[4] Eca-simm. Visitado: 2021-04-15. [Online]. Available: https://eca-simm.uva.es/

[5] Docker-compose. Visitado: 2021-04-06. [Online]. Available: https://docs.docker.com/compose/

[6] Tfg de la universidad politécnica de madrid. Visitado: 2021-10-19. [Online]. Available: https://oa.upm.es/64214/

[7] Tfg de la universidad de las palmas de gran canaria. Visitado: 2021-10-19. [Online]. Available: https://accedacris.ulpgc.es/handle/10553/76812

[8] Raspberry pi. Visitado: 2021-10-19. [Online]. Available: https://www.raspberrypi.org/

[9] Internet of things. Visitado: 2021-10-19. [Online]. Available: https://www2.deloitte.com/es/es/pages/technology/articles/IoT-internet-of-things.html

[10] Tfg de la universidad politécnica de valencia. Visitado: 2021-10-19. [Online]. Available: https://riunet.upv.es/handle/10251/71386

[11] Docker-desktop. Visitado: 2021-04-06. [Online]. Available: https://www.docker.com/products/docker-desktop

[12] Api. Visitado: 2021-10-12. [Online]. Available: https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces

[13] Docker-engine. Visitado: 2021-04-07. [Online]. Available: https://docs.docker.com/engine/

[14] Docker-swarm. Visitado: 2021-04-07. [Online]. Available: https://docs.docker.com/engine/swarm/swarm-mode/

[15] Kubernetes. Visitado: 2021-04-08. [Online]. Available: https://kubernetes.io/es/

[16] What is the iterative design approach? Visitado: 2021-10-06. [Online]. Available: https://wishdesk.com/blog/what-is-iterative-design-approach

[17] Angular. Visitado: 2021-10-12. [Online]. Available: https://angular.io/

[18] Astah. Visitado: 2021-10-12. [Online]. Available: https://astah.net/

[19] Git. Visitado: 2021-10-12. [Online]. Available: https://git-scm.com/

[20] Gitlab. Visitado: 2021-10-12. [Online]. Available: https://about.gitlab.com/

[21] Google meet. Visitado: 2021-10-12. [Online]. Available: https://apps.google.com/meet/

[22] Mongodb. Visitado: 2021-10-12. [Online]. Available: https://www.mongodb.com/es

[23] Firefox. Visitado: 2021-10-12. [Online]. Available: https://www.mozilla.org/es-ES/firefox/

[24] Nginx. Visitado: 2021-10-12. [Online]. Available: https://www.nginx.com/

[25] node.js. Visitado: 2021-10-12. [Online]. Available: https://nodejs.org/es/

[26] Dockerode. Visitado: 2021-05-04. [Online]. Available: https://github.com/apocas/dockerode

[27] Expressjs. Visitado: 2021-10-12. [Online]. Available: https://expressjs.com/es/

[28] jsonwebtoken para node. Visitado: 2021-06-28. [Online]. Available: https://www.npmjs.com/package/jsonwebtoken

[29] Mongoose. Visitado: 2021-10-12. [Online]. Available: https://mongoosejs.com/

[30] Overleaf. Visitado: 2021-07-01. [Online]. Available: https://es.overleaf.com/

[31] Postman. Visitado: 2021-10-16. [Online]. Available: https://www.postman.com/

[32] Telegram. Visitado: 2021-10-12. [Online]. Available: https://telegram.org/

[33] Visual studio code. Visitado: 2021-10-13. [Online]. Available: https://code.visualstudio.com/

[34] Salario ingeniero informático. Visitado: 2021-10-11. [Online]. Available: https://www.jobted.es/salario/ingeniero-inform%C3%A1tico

[35] Servidores vps en ovh cloud. Visitado: 2021-10-11. [Online]. Available: https://www.ovhcloud.com/es-es/vps/compare/

[36] Documentación oficial de docker engine. Visitado: 2021-04-16. [Online]. Available: https://docs.docker.com/engine/api/v1.41/

[37] Json web token. Visitado: 2021-10-12. [Online]. Available: https://es.wikipedia.org/wiki/JSON_Web_Token

[38] Utf-8. Visitado: 2021-05-04. [Online]. Available: https://es.wikipedia.org/wiki/UTF-8

[39] Latex. Visitado: 2021-10-12. [Online]. Available: https://es.wikipedia.org/wiki/LaTeX

[40] Historias de usuario. Visitado: 2021-05-12. [Online]. Available: https://www.atlassian.com/es/agile/project-management/user-stories

[41] Protocolo http. Visitado: 2021-10-13. [Online]. Available: https://es.wikipedia.org/wiki/Protocolo_de_transferencia_de_hipertexto

[42] Patrón de arquitectura de capas. Visitado: 2021-10-13. [Online]. Available: https://platzi.com/tutoriales/1248-pro-arquitectura/5439-patron-arquitectonico-de-capas-layers

[43] Patrón de singleton. Visitado: 2021-10-13. [Online]. Available: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/patron-singleton/

[44] Patrón de singleton en angular. Visitado: 2021-10-13. [Online]. Available: https://angular.io/guide/singleton-services

[45] Filezilla. Visitado: 2021-10-13. [Online]. Available: https://filezilla-project.org/

[46] Handling cors with node.js. Visitado: 2021-06-08. [Online]. Available: https://stackabuse.com/handling-cors-with-node-js