MEMORIA DEL PROYECTO

ARQUITECTURA DE SERVICIOS RESTful PARA UN JUEGO DE TABLERO ONLINE

Trabajo de Fin de Máster

MÁSTER EN INGENIERÍA INFORMÁTICA

VNiVERSiDAD D SALAMANCA

CAMPUS DE EXCELENCIA INTERNACIONAL

Febrero de 2023

AUTOR

Javier Vidal Ruano

TUTOR

Rodrigo Santamaría Vicente

Firma del tutor

D. Rodrigo Santamaría Vicente, profesor del Departamento de Informática y Automática de la Universidad de Salamanca.

CERTIFICA:

Que el trabajo titulado "*Arquitectura de servicios RESTful para un juego de tablero online*" ha sido realizado por D. Javier Vidal Ruano, con DNI 70971324V, bajo su dirección y tutela.

Y para que así conste a todos los efectos oportunos.

En Salamanca, a 27 de febrero de 2023.

D. Rodrigo
Santamaría Vicente

Resumen

Desde hace unos años, se está produciendo un aumento en el número de personas que eligen jugar a juegos de mesa [2], a la vez que un crecimiento en el uso de plataformas online de digitalización de juegos de mesa como Tabletop simulator (2015) o BoardGameArena (2010). Esto, sumado a la inusual situación que recientemente hemos tenido que superar, la pandemia global, que provocó los confinamientos obligatorios e hizo que las personas pasaran dentro de sus casas una cantidad de tiempo mucho mayor a la habitual, viéndose reflejado, entre otros, en un incremento de las ventas de juegos de mesa y cartas en un 18.3% [3], hizo que esta situación se acentuara aún más. Debido a estos datos, se considera que la digitalización de un juego de tablero puede ser una buena idea y que puede tener bastante uso.

El origen del proyecto reside en el tutor de este, Rodrigo Santamaría Vicente, que vio la necesidad de diseñar un juego inspirado en el popular juego de tablero Twilight Struggle, para 2 jugadores, a una cantidad de 4 jugadores, ya que en reuniones familiares o con amigos, solo había posibilidad de que participaran 2 personas. Twilight Struggle es un juego de mesa de estrategia ambientado en la guerra fría, en el cual un jugador representa a los Estados Unidos y el otro a la Unión Soviética. Se decidió rediseñar el juego ambientándolo en la época post-guerra fría, ampliarlo a 4 jugadores, que representan las 4 potencias mundiales (Estados Unidos, Unión Europea, Rusia y China), cambiar las cartas y las reglas de juego, y usar el acontecimiento histórico que marcó el inicio de esta nueva etapa como nombre del juego: The Wall is Down.

El objetivo de este proyecto consiste en crear una infraestructura SOA completa que permita realizar partidas de este juego de tablero, estructurándola en dos capas de manera que la capa básica permita digitalizar todo tipo de juegos de tablero. Para ello se ha elegido el lenguaje de programación Python y el framework de desarrollo de APIs RESTful FastAPI.

El código está disponible en https://github.com/javiervidrua/TWID-SOA y un servidor funcional está disponible en https://prodiasv30.fis.usal.es/docs.

Palabras clave: Juego de mesa, API, RESTful, SOA, Python, FastAPI.

Summary

For the past few years, there has been an increase in the number of people choosing to play board games [2], along with a growth in the use of online platforms for digitizing board games such as Tabletop simulator (2015) or BoardGameArena (2010). This, added to the unusual situation that we have recently had to overcome, the global pandemic, which caused mandatory confinements and made people spend inside their homes a much greater amount of time than usual, being reflected, among others, in an increase in sales of board and card games by 18.3% [3], made this situation even more accentuated. Due to these data, it is considered that the digitization of a board game can be a good idea and that it can be quite useful.

The origin of the project lies in its tutor, Rodrigo Santamaría Vicente, who saw the need to design a game inspired by the popular board game Twilight Struggle, for 2 players, to a number of 4 players, since in family gatherings or with friends, there was only a possibility that only 2 people could participate. Twilight Struggle is a strategy board game set in the cold war, in which one player represents the United States and the other the Soviet Union. It was decided to redesign the game setting it in the post-cold war era, expand it to 4 players, representing the 4 world powers (United States, European Union, Russia and China), change the cards and game rules, and use the historical event that marked the beginning of this new era as the name of the game: The Wall is Down.

The objective of this project is to create a complete SOA infrastructure that allows the creation of games of this board game, structuring it in two layers so that the basic layer allows the digitization of all types of board games. For this purpose, the Python programming language and the FastAPI RESTful API development framework have been chosen.

The code is available at https://github.com/javiervidrua/TWID-SOA and a working server is available at https://prodiasv30.fis.usal.es/docs.

Keywords: Boardgame, API, RESTful, SOA, Python, FastAPI.

Índice de contenido

Índice de Ilustraciones

1 – Introducción

En la actualidad, con todos los juegos en primera persona, con gráficos espectaculares que surgen día tras día, parece que los juegos clásicos o antiguos siguen una tendencia a la desaparición y al desuso, pero esto no podría estar más lejos de la realidad. Encontramos una gran cantidad de foros con personas que juegan activamente, proponen modificaciones de juegos de mesa existentes y juegos de mesa completamente nuevos. Cabe destacar que, a su vez, las tiendas de aplicaciones para móviles han registrado un aumento de juegos de mesa como el ajedrez, el parchís o Scrabble. Si relacionamos estos datos, el resultado es que digitalizar juegos de mesa puede obtener una gran cantidad de usuarios, así como de desarrolladores que se encarguen de realizar las digitalizaciones en sí mismas.

Se pretende crear una infraestructura SOA básica, cuyos servicios se basen y cumplan la especificación RESTful, que permita digitalizar todo tipo de juegos de tablero. Para ello se ha elegido el lenguaje de programación Python y el framework de desarrollo de APIs FastAPI. La elección de este lenguaje de programación se debe a su gran uso y referencias disponibles en internet, guías y problemas ya solucionados, además de que Python incluye una gran cantidad de bibliotecas con gran funcionalidad, evitando tener que implementar funciones comunes y básicas. El framework FastAPI se ha elegido por su increíble rendimiento sobre el resto de las herramientas, además de que incluye integración con otras herramientas de buenas prácticas, como Pydantic, para la validación de datos de entrada y salida, y Swagger, como herramienta de documentación técnica e interfaz de pruebas de usuario.

Una vez desarrollados los servicios, se crean contenedores de Docker, listos para ser desplegados en producción en un servidor del Departamento de Informática y Automática, mediante un archivo Docker Compose, que asegura la disponibilidad del servicio (mediante reinicios automáticos en caso de fallo en algún servicio) y permite gestionar el entorno de forma sencilla, autogestionando las redes de comunicación de servicios y variables de entorno. Se utiliza el proxy inverso Nginx para abrir el servicio a la red interna de la USAL a través del protocolo HTTP(S), de manera que las comunicaciones sean estándar y seguras.

1.1 – Idea del proyecto

El origen del proyecto reside en el tutor de este, Rodrigo Santamaría Vicente, que vio la necesidad de modificar el juego Twilight Struggle, para 2 jugadores, a una cantidad de 4 jugadores, ya que en reuniones familiares o con amigos, solo había posibilidad de que participaran 2 personas. Twilight Struggle

es un juego de mesa temático de guerra y estrategia ambientado en la guerra fría, en el cual un jugador representa a los Estados Unidos y el otro a la Unión Soviética. Se decidió rediseñar el juego ambientándolo en la época post-guerra fría, ampliarlo a 4 jugadores, que representan las 4 potencias mundiales (Estados Unidos, Unión Europea, Rusia y China), ampliar los países del mapa, cambiar las cartas y las reglas de juego, y usar el acontecimiento histórico que marcó el inicio de esta nueva etapa como nombre del juego: The Wall is Down. Toda la información del juego se puede consultar en el enlace a Google Drive https://drive.google.com/file/d/1NQ0MBrjblynNabtVV32vSwMKB69BZAUD/view.

La idea es implementar el juego TWID, pero de una forma extensible a otros juegos de tablero, definiendo una capa de servicios básicos que proporcionen los recursos del juego (servicios de recursos), y otra capa, superior, que se encargue de llevar la lógica y la mecánica del juego (servicios de control), manipulando los recursos usando el servicio de la capa inferior de manera apropiada. La capa de recursos implementa acciones comunes a muchos juegos, como pueden ser subir de ronda, aumentar o disminuir la puntuación de los jugadores o robar cartas del mazo y la capa de control se encarga de las reglas más complejas y específicas del juego, como se puede ver en la Ilustración 1:

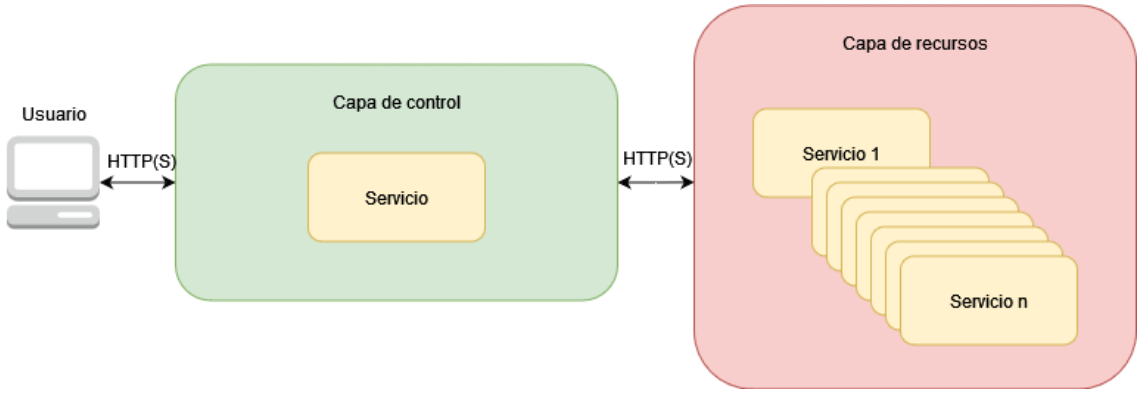

Ilustración 1. Idea de arquitectura del proyecto.

2 – Objetivos

En este apartado se argumentan los objetivos técnicos que el sistema deberá conseguir para cumplir su propósito y tener una buena calidad, y los objetivos personales que se pretenden satisfacer mediante la construcción de este.

2.1 – Objetivos técnicos

Este sistema consiste en la creación de la infraestructura SOA básica para las reglas de un juego de tablero online, extensible a otros, que defina de manera completa y eficiente las reglas y mecánicas propias del juego.

2.1.1 – Análisis y diseño de la arquitectura SOA requerida para las reglas del juego

Se deberá analizar y diseñar la arquitectura orientada a servicios (SOA) del sistema, de forma que permita la flexibilidad, extensibilidad, escalabilidad y portabilidad de los servicios, así como un alto rendimiento. También deberá facilitar la implementación de las reglas del juego en la medida de lo posible, permitiendo una comunicación sencilla entre los servicios.

2.1.2 – Creación de los servicios necesarios

Una vez diseñada la arquitectura, se deberá implementar cada uno de los servicios, de modo que cumplan con sus debidas funciones y se comuniquen entre sí de forma óptima, logrando un sistema global exitoso. Para ello se deberá definir estructuras de programación comunes, patrones arquitectónicos y buenas prácticas que disminuyan la probabilidad de cometer errores, así como el uso de formatos estándar para los ficheros de datos.

2.1.3 – Preparación de contenedores de los servicios con Docker

Tras finalizar la creación de los servicios, se deberá preparar imágenes de estos mediante el uso de la herramienta Docker. De esta forma se obtendrán pequeñas piezas autocontenidas, que harán que su despliegue sea organizado, seguro y estable.

2.1.4 – Puesta en producción del sistema en un servidor del departamento

Se deberá poner el sistema completo en producción, usando un servidor del Departamento de Informática y Automática. Para ello, se deberá realizar toda la configuración de un servidor GNU/Linux desde cero, incluyendo la configuración de usuarios, reglas de firewall, instalación de herramientas y puesta

en producción, permitiendo monitorizar los servicios y asegurando la disponibilidad de estos.

2.2 – Objetivos personales

2.2.1 – Planificación temporal

Quizá, el mayor objetivo a cumplir con este trabajo sea conseguir una compatibilización del desarrollo del Trabajo de Fin de Máster con el curso, además de con un trabajo a jornada completa en un puesto de SRE (Site Reliability Engineer). Se trata de una gran meta que requiere una gestión de tareas y disciplina de trabajo notables, ya que requiere el cumplimiento de fechas límite muy cercanas entre sí, contando únicamente con el tiempo restante fuera del trabajo y los fines de semana.

2.2.2 – Gestión del proyecto

Otro gran objetivo consiste en la identificación de las tareas a realizar, así como de su división en subtareas para lograr una forma de trabajo satisfactoria, permitiendo llegar a cumplir todos los objetivos propuestos y teniendo en cuenta las complicaciones que siempre surgen en los peores momentos. El logro de este objetivo supondrá ganar experiencia a nivel de gerencia y permitirá avanzar hacia puestos de gestión de equipos, así como obtener una perspectiva más amplia sobre los proyectos.

2.2.3 – Manejo de código con sistemas de control de versiones

Hay que tener en cuenta que saber manejar correctamente los sistemas de control de versiones se ha convertido en un requisito indispensable en todos los puestos de desarrollo, indiferentemente de la especialización de los profesionales. Un claro ejemplo es el incremento del uso de plataformas como GitHub, GitLab, Bitbucket o Azure DevOps para toda la gestión del código de las empresas. Se ha elegido usar la plataforma GitHub ya que se trata de la más usada en la actualidad y además porque a los estudiantes universitarios les ofrece su plan “Pro” de forma gratuita, obteniendo funcionalidades extra muy útiles, como puede ser la herramienta GitHub Copilot o una cantidad ilimitada de pipelines CI/CD (Continuous Integration/Continuous Deployment).

2.2.4 – Aprendizaje de nuevas tecnologías

Hay una gran variedad de frameworks de desarrollo de APIs para todo tipo de lenguajes de programación. Debido a la experiencia laboral con JavaScript y el framework Express y de Python con el framework Flask, se ha decidido aprender el framework FastAPI para el lenguaje Python, por varias razones:

- Como sugiere el nombre, se trata de un framework que permite obtener un rendimiento superior, ya que usa un servidor asíncrono, al contrario que Flask, siendo este más antiguo.

- Viene con funcionalidad extra ya incorporada, como la gestión de modelos de autenticación o validación de datos de entrada y de salida.

- Soporta la autogeneración de documentación directamente desde el código, proporcionando una interfaz gráfica que permite probar el servicio de forma rápida y sencilla.

Otra tecnología que cada día es más usada tanto en la industria como en el campo de la investigación es la herramienta Docker, la cual permite evitar todos los problemas de portabilidad y aislamiento de servicios que tantos años han causado dolores de cabeza, mediante el uso de contenedores. Se trata de la base que sostiene otras herramientas como Kubernetes, Openshift o Apache Mesos, las cuales se han convertido en un estándar en la industria. Se pretende aumentar los conocimientos sobre esta herramienta, así como del proxy inverso Nginx, para el enrutamiento de peticiones HTTP(S).

2.2.5 – Desarrollo de código limpio

Otro gran objetivo consiste en el desarrollo de código limpio, mantenible, que siga estándares y de ese modo perdure en el tiempo, pudiendo ser reutilizado para implementar otras aplicaciones en el futuro. Se intentará seguir la misma estructura para la implementación de clases y funciones, con nombres claros y usando archivos y formatos que sean estándares en la industria. De este modo las colaboraciones y nuevas implementaciones que usen el código del proyecto como plantilla se encontrarán menos dificultades técnicas.

3 – Conceptos teóricos

3.1 – API

Se trata de la abreviatura de Application Programming Interface. Una API consiste en un conjunto de reglas que una aplicación expone de manera que otras aplicaciones puedan comunicarse con esta. Actúa como una capa intermedia que procesa transferencias de datos entre sistemas. Las definiciones y protocolos de una API permiten conectar todo tipo de aplicaciones, ya que para usarlas los desarrolladores solamente deben cumplir la especificación técnica de la misma. Hay diferentes estándares, pero el más usado en la actualidad es la propuesta OpenAPI 3.0, parte de la OpenAPI Initiative.

Esta propuesta fue realizada por la empresa Swagger como una solución para estandarizar las definiciones de APIs que cumplen con la arquitectura RESTful, y fue donada para actualmente formar parte de la Linux Foundation. Permite que los ordenadores y las personas puedan entender las capacidades del servicio que describe sin necesitar acceso al código fuente, documentación ni inspeccionar el tráfico de red. Si se define correctamente, permite a un consumidor usar el servicio remoto con mínimo esfuerzo y lógica de implementación. Además, permite ser usada por herramientas de generación automática de documentación para generar servidores y clientes de estos en varios lenguajes de programación, herramientas de prueba y más. Un documento OpenAPI 3.0 se representa en formato JSON o YAML, lo cual permite que tenga una alta legibilidad y sea fácilmente modificable. Será el estándar usado para escribir la especificación técnica de los servicios, así como para autogenerar los clientes para la realización de pruebas.

3.2 – REST

Consiste en la abreviatura de REpresentational State Transfer, y se trata de un estilo de arquitectura de software para sistemas de hipermedia distribuidos como la World Wide Web. Se basa en el uso del protocolo HTTP y archivos de cualquier formato (aunque los más usados son XML y JSON) sin abstracciones adicionales como por ejemplo SOAP. Consiste en un protocolo cliente-servidor sin estado, un conjunto de operaciones bien definidas (verbos HTTP), sintaxis universal para la identificación de recursos (URIs) y el uso de hipermedios (HTML, XML, JSON), de manera que se obtenga una interfaz uniforme.

3.3 – RESTful

Se trata de una API que cumple con la arquitectura REST y que además cumple con los siguientes requisitos:

- La URI del servicio es del tipo http://foo.com/resource
- Los datos están un formato estándar (XML, JSON)
- Solamente se usan los siguientes verbos HTTP: GET, POST, PUT, DELETE.
- Accesible a través de un navegador (transferencia de hipertexto).

3.4 – Principios del juego TWID

Se trata de un juego por turnos de 8 rondas. En cada ronda se reparten cartas a los jugadores, y estos las usan sobre el mapa, para tratar de ganar influencia en los países de las distintas regiones, y eventualmente, ganar puntos de victoria si se cumplen ciertas condiciones. Las acciones de unos jugadores afectan al resto, hasta tan punto de poder limitar las acciones de otros jugadores en determinadas zonas del mapa. El objetivo es conseguir la mayor cantidad de puntos de victoria, lo antes posible.

4 – Técnicas y herramientas utilizadas

A continuación, se detallan las técnicas y las herramientas usadas durante el desarrollo del proyecto.

4.1 – Metodología de trabajo

Tras dos años de experiencia laboral trabajando con metodologías ágiles, la elección ha sido obvia. Estas metodologías se basan en la división del proyecto en etapas bien definidas y con una duración temporal estimada, así como la colaboración entre el equipo y el cliente (stakeholder) de forma diaria para lograr una mejora iterativa y continuada en el tiempo. Se ha decidido combinar la metodología SCRUM con Kanban. De SCRUM, se ha usado la clasificación y planificación de tareas, y del método Kanban el sistema de clasificación y gestión de tareas.

4.1.1 – Scrum

Scrum es un marco de trabajo para desarrollo ágil de software que se ha expandido a muchas industrias. Es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo y obtener el mejor resultado posible de proyectos, caracterizado por:

- Adoptar una estrategia de desarrollo incremental, en lugar de la planificación y ejecución completa del producto.
- Basar la calidad del resultado más en el conocimiento tácito de las personas en equipos auto organizados, que en la calidad de los procesos empleados.
- Solapar las diferentes fases del desarrollo, en lugar de realizar una tras otra en un ciclo secuencial o en cascada.

4.1.2 – Kanban

Se trata de un método Lean, muy popular, de gestión del flujo de trabajo para definir, gestionar y mejorar los servicios que proporciona el trabajo de conocimiento. Te ayuda a visualizar el trabajo, maximizar la eficiencia y mejorar continuamente. El trabajo se representa en tableros Kanban, lo que te permite optimizar la entrega de trabajo a través de múltiples equipos y manejar, incluso los proyectos más complejos en un solo entorno.

La herramienta que se ha usado para la gestión de tareas (“4.3.3 – GitHub projects“) sigue una forma de trabajo fiel al método Kanban.

4.2 – Herramientas de desarrollo

4.2.1 – Visual Studio Code

Se trata de un entorno de desarrollo integrado (IDE) multiplataforma desarrollado por Microsoft, el cual tiene una gran cantidad de funcionalidades que lo hacen muy atractivo, consiguiendo que más de la mitad de los desarrolladores de todo el mundo lo usen como su editor predeterminado. Posee control de versiones con Git integrado, resaltado de sintaxis y lo más importante, soporte de extensiones personalizadas, de modo que constantemente se publican nuevas extensiones que amplían su funcionalidad.

4.2.2 – Python

Es un lenguaje de programación de propósito general y de alto nivel. Su filosofía de diseño enfatiza la legibilidad de código con el uso de la sangría en lugar de las llaves, clásicamente usadas en la mayoría de los lenguajes de programación anteriores. Se trata de un lenguaje de tipificación dinámica (en tiempo de ejecución) ya que se trata de un lenguaje interpretado. Eso ocasiona que no sea tan rápido como los lenguajes compilados, pero lo compensa con cantidad de tiempo invertido en los desarrollos, que suele ser mucho menor. Viene con una gran cantidad de módulos o paquetes que implementan funcionalidad básica, como servidores HTTP, Websockets, módulo de logging, manipulación de archivos estándar como XML, CSV, JSON.

Actualmente, es el lenguaje por excelencia para la inteligencia artificial, el análisis de grandes cantidades de datos (Big Data), el minado de datos y hasta otras áreas como el IoT (Internet of Things).

4.2.3 – FastAPI

Es un framework web de desarrollo de APIs RESTful para el lenguaje de programación Python. Es moderno, muy rápido (alto rendimiento) basado en la sugerencia estándar de tipos de datos de Python. Sus principales ventajas sobre otros frameworks son las siguientes:

- Muy alto rendimiento, ya que es asíncrono.
- Intuitivo y sencillo, rápido de desarrollar.
- Induce a menos errores, gracias a la integración con Pydantic.
- Robusto.
- Totalmente compatible con la especificación OpenAPI 3.0 y otros como puede ser JSON Schema.

A continuación, en la Ilustración 2, se presenta una comparativa de rendimientos donde se puede apreciar que FastAPI es muy superior a de Flask, otro framework de desarrollo de APIs para Python (último de la lista). Esto es debido a que se trata de un framework que trabaja con funciones asíncronas, al contrario que Flask, que trabaja de forma síncrona.

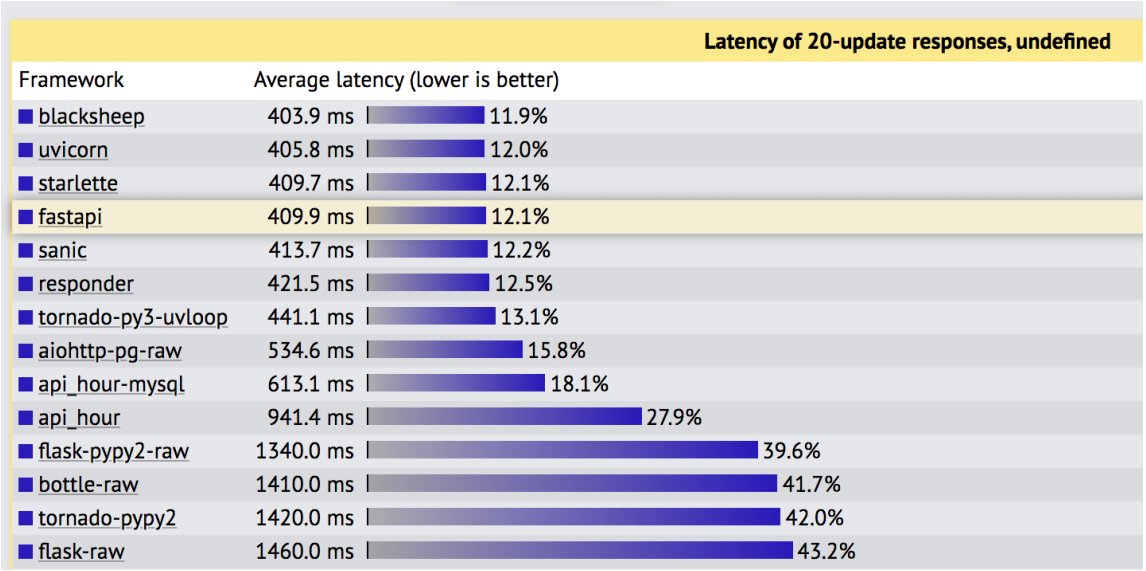

Ilustración 2. Comparativa de rendimiento entre FastAPI y Flask. [https://testdriven.io/blog/fastapi-streamlit/]

4.2.4 – Pydantic

Es un framework para el lenguaje de programación Python, que permite realizar validaciones de datos y configuraciones usando anotaciones estándar de tipos de datos de Python. Comprueba los tipos en tiempo de ejecución y proporciona errores legibles por los usuarios cuando los datos son inválidos.

4.2.5 – Swagger

Es un conjunto de herramientas de software de código abierto para diseñar, construir, documentar, y utilizar servicios web RESTful. Fue desarrollado por SmartBear Software e incluye documentación automatizada, generación de código, y generación de casos de prueba. Se basa en la especificación OpenAPI 3.0 para ofrecer una interfaz gráfica de usuario en la que se muestra la documentación del servicio y que proporciona la posibilidad de probar el servicio de forma rápida, sencilla y eficaz.

4.2.6 – JSON

Abreviatura de JavaScript Object Notation, es el formato de datos del lenguaje JavaScript, altamente usado para el almacenamiento y transporte de información, debido a su ligereza y legibilidad.

4.2.7 – Docker

Docker es una plataforma de código abierto que permite a los desarrolladores crear, desplegar, ejecutar, actualizar y gestionar contenedores, que son componentes ejecutables estandarizados que combinan el código fuente de las aplicaciones con las bibliotecas y dependencias del sistema operativo (SO) necesarias para ejecutar ese código en cualquier entorno.

De la misma forma en que los objetos son instancias de clases, en términos de la programación orientada a objetos (POO), con Docker, los contenedores son instancias de imágenes. Las imágenes se pueden crear de cero, o a partir de una imagen base, añadiendo funcionalidad. La segunda opción es la más usada, ya que las empresas crean imágenes base de sus productos como puede ser Python, Node.js o PostgreSQL, por nombrar algunos, y los desarrolladores que usan dichas herramientas añaden su funcionalidad. Este proceso se puede entender mejor con el diagrama de la Ilustración 3, que muestra el proceso de creación una imagen utilizando otra de base.

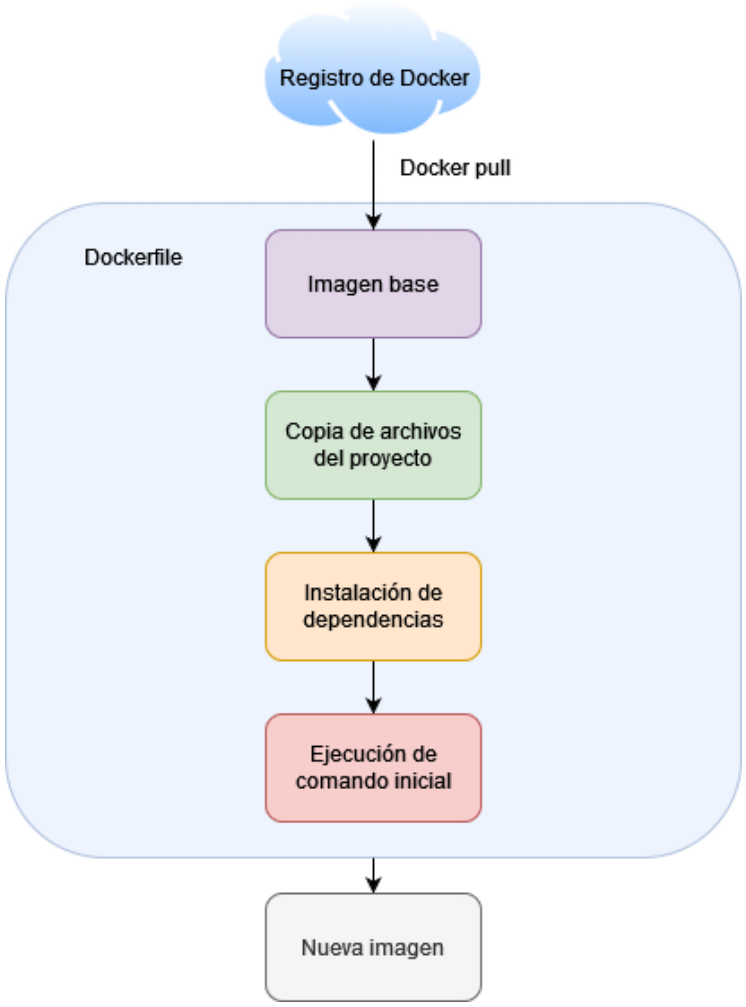

Ilustración 3. Proceso de creación de una imagen de Docker a partir de un archivo Dockerfile.

En la Ilustración 4 se puede ver el funcionamiento de la herramienta. El cliente, que puede estar en el mismo equipo o en otro remoto, ejecuta comandos contra el servicio de Docker, el cual se ejecuta en la máquina anfitrión. El servicio, descarga las imágenes necesarias del registro, y las usa para crear contenedores que usen los recursos del equipo anfitrión.

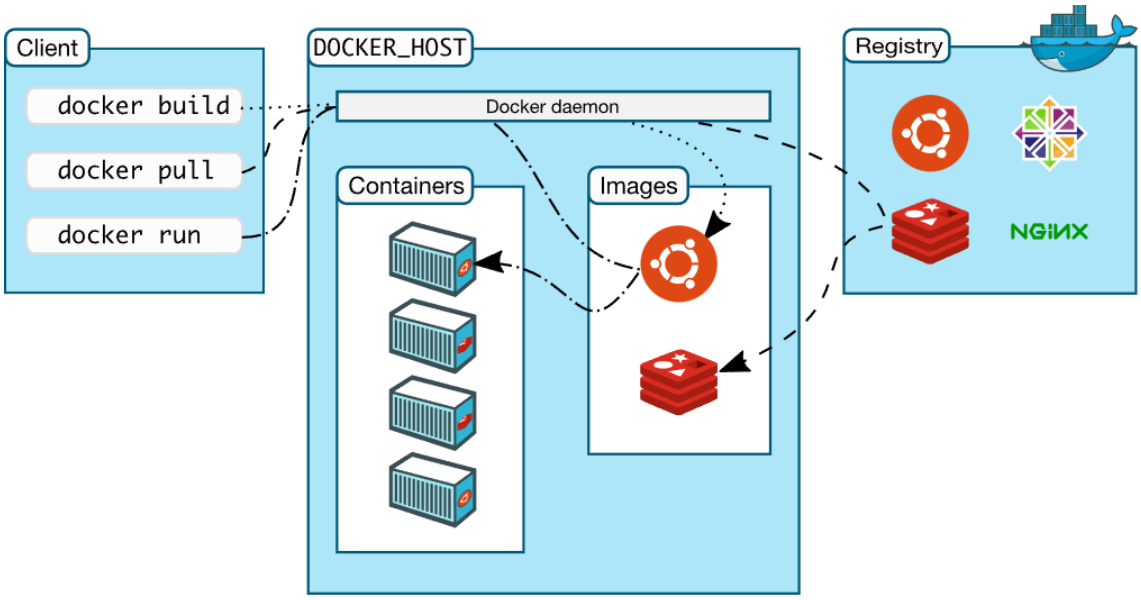

Ilustración 4. Diagrama de arquitectura de Docker. [https://docs.docker.com/get-started/overview/]

Los contenedores simplifican el desarrollo y la entrega de aplicaciones distribuidas. Se han hecho cada vez más populares a medida que las organizaciones adoptan el desarrollo nativo en la nube y los entornos híbridos. Los desarrolladores pueden crear contenedores sin Docker, trabajando directamente con las capacidades integradas en Linux y otros sistemas operativos. Pero Docker hace que la creación de contenedores sea más rápida, sencilla y segura. En el momento de escribir estas líneas, Docker informaba de que más de 18 millones de desarrolladores utilizaban la plataforma, y tenían una cantidad de más de 13000 millones de descargas de imágenes cada mes.

4.2.8 – Docker Compose

Se trata de una herramienta para definir y ejecutar aplicaciones de varios contenedores Docker. Permite, mediante un archivo YAML, especificar todos los parámetros necesarios para lanzar los servicios, desde la red de Docker en la que deben estar, hasta el nombre los contenedores que se creen. Todo ello de manera concisa y sencilla, permitiendo ahorrar mucho tiempo respecto a si se realizara el mismo procedimiento con comandos básicos de Docker.

4.2.9 – Postman

Se trata de una herramienta destinada al desarrollo de APIs REST, que tiene varias herramientas que permiten acelerar el desarrollo y las pruebas sobre estas.

Su funcionalidad se basa en la habilidad de realizar peticiones HTTP personalizables de forma completa y minuciosa, de modo que permite comprobar el funcionamiento de las URIs objetivo. Además, permite realizar monitorizaciones de APIs, automatizar pruebas y escribir documentación, entre otras funciones.

Se ha usado para realizar pruebas de los endpoints de los dos servicios implementados, de forma que se pudiera comprobar el correcto funcionamiento, lanzándole todo tipo de peticiones y viendo la respuesta del sistema.

4.2.10 – Nginx

Consiste en un servidor proxy inverso, ligero y de alto rendimiento que soporta varios protocolos (HTTP, IMAP, POP3). Es software de código abierto, y se encuentra disponible para sistemas tipo UNIX y Windows.

Se ha decidido usar esta herramienta en lugar de otras también muy conocidas como Apache dada la gran disponibilidad de información y guías de uso junto con la herramienta Uvicorn, que se describe a continuación. También, desde Docker recomiendan su uso sobre el servidor Apache, por su simpleza de configuración y modularidad, entre otros.

4.2.11 – Uvicorn

Uvicorn es una implementación de servidor web ASGI (Asynchronous Server Gateway Interface) para Python. Hasta hace poco, Python carecía de una interfaz mínima de servidor/aplicación de bajo nivel para funcionalidad asíncrona. La especificación ASGI llena este vacío, y significa que ahora somos capaces de empezar a construir un conjunto común de herramientas utilizables a través de todos los marcos async. Actualmente, Uvicorn es compatible con HTTP/1.1 y WebSockets.

4.3 – Herramientas CASE

4.3.1 – Git

Es un sistema de control de versiones creado por Linus Torvalds. Se encarga de llevar un registro de los cambios efectuados en un proyecto en el que

trabajan varias personas, permitiendo la colaboración y la resolución de conflictos de una forma rápida y sencilla. En la Ilustración 5 se presenta una de las más utilizadas propuestas como modelo de ramas y versionado con Git. La rama principal es “master”, y de ella nacen las ramas “develop” y “hotfixes”. La idea es tener una para los arreglos, y otra para las nuevas funcionalidades. De esta forma se permite el trabajo paralelo y se disminuye el número de conflictos.

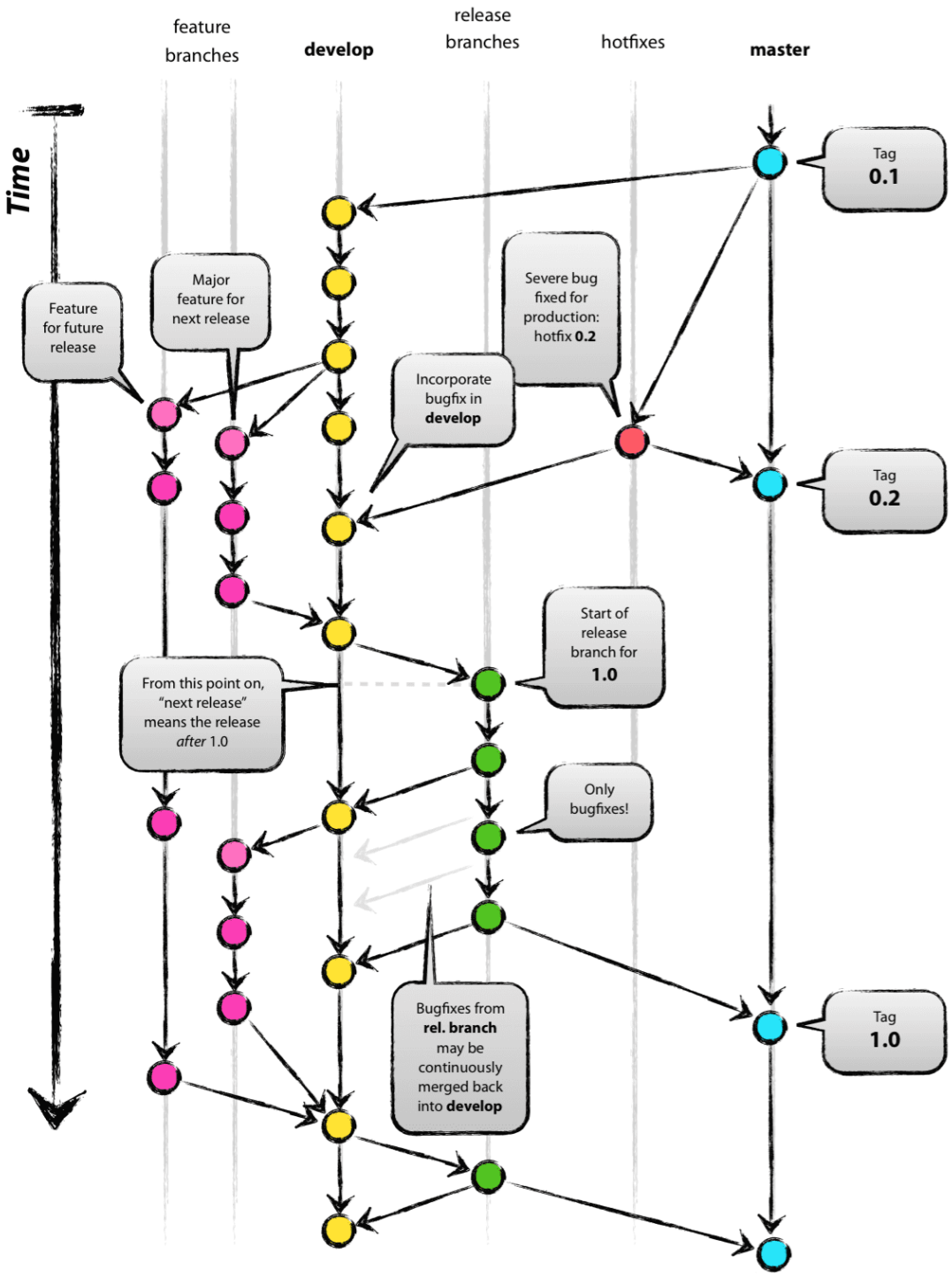

Ilustración 5. Una de las propuestas más usadas como modelo de ramas y versionado con Git. [https://nvie.com/posts/a-successful-git-branching-model/]

Se ha usado como una herramienta de control de versiones del proyecto. Resulta increíblemente útil para gestionar cambios en archivos, ver cuándo se han introducido bugs en el servicio, o simplemente si se desea recobrar un fragmento de código que fue borrado.

4.3.2 – GitHub

Se trata de un servicio web de control de versiones basado en Git. Además de almacenamiento de repositorios, también ofrece un alojamiento de páginas web, wikis, seguimiento de errores, CI/CD y otras herramientas como puede ser GitHub Copilot.

Esta herramienta se usará como almacenamiento seguro del repositorio del proyecto, evitando los problemas que surgen cuando no se hacen copias de seguridad.

4.3.3 – GitHub projects

Se trata de un servicio web que ofrece GitHub para la gestión de tareas en los proyectos. Está basado en la metodología Kanban, agrupando tareas en bloques, que generalmente son: “Propuestas”, “Listo para hacer”, “En desarrollo” y “Finalizado”.

Esta herramienta permite tener una mejor organización, evitando perderse entre las tareas ya que permite definir la estimación de dificultad y tiempo que se deberá dedicar a cada tarea. Dada su total integración con el repositorio de código y su fácil acceso, ha sido la elección obvia. El uso de la herramienta consiste en mover las tarjetas de izquierda a derecha, según se vaya realizando avances en la tarea.

5 – Aspectos relevantes en el desarrollo del proyecto

En este apartado se comentarán los detalles más importantes de las fases que conforman el desarrollo del proyecto.

5.1 – Planificación de tareas

Como se ha comentado en el apartado "4.1 – Metodología de trabajo", se ha utilizado una metodología ágil para la planificación de tareas basada en Scrum, con reuniones aproximadamente mensuales con el tutor donde se establecían los hitos del proyecto (denominados "features") y se analizaba el cumplimiento del hito anterior. Una vez definidos los hitos, la creación y gestión de tareas ha sido responsabilidad del alumno. A continuación, se presenta la lista de hitos del proyecto, junto con su temporización:

- Comprensión y análisis de las reglas de juego.
  - Inicio: 18-10-2022
  - Fin: 27-10-2022
- Diseño y creación de la infraestructura básica.
  - Inicio: 28-10-2022
  - Fin: 24-11-2022
- Diseño y creación de la infraestructura específica.
  - Inicio: 25-11-2022
  - Fin: 15-12-2022
- Empaquetamiento y despliegue en producción del sistema.
  - Inicio: 16-12-2022
  - Fin: 29-12-2022
- Documentación.
  - Inicio: 30-12-2022
  - Fin: 18-01-2023

5.2 – Franjas y horario de trabajo

Dado que el alumno debe compatibilizar el máster con un trabajo a jornada completa, en un principio se planteó la idea de realizar todo el trabajo relativo al Trabajo de Fin de Máster en la franja de tiempo correspondiente a los viernes de 16:00 a 20:00, los sábados de 08:30 a 17:00, y los domingos de 10:00 a 14:00. Esto es equivalente a dos días de trabajo en una jornada de 40 horas semanales, lo cual, en un principio, parecía un buen plan. Tras un par de semanas siguiendo este modelo de trabajo, se descubrió que presentaba varios inconvenientes:

- El tutor se encontraba disponible entre semana (su jornada laboral), mientras que el alumno realizaba los avances en fin de semana. Esto llevó a que la comunicación no fuese fluida y que se avanzase de forma lenta.
- Tras la primera semana, el alumno descubrió que después de trabajar 40 horas semanales en su puesto profesional, trabajar un día completo más (el sábado) era demasiada carga de trabajo continuada y muy duro a nivel psicológico.

Por estos motivos, se tomó la decisión de cambiar el horario de trabajo al siguiente, obteniendo una cantidad de horas de trabajo similar:

- Desde las 18:00 a las 20:00 horas de lunes a viernes.
- De 10:00 a 14:00 los sábados.
- Excepcionalmente, de 10:00 a 14:00 los domingos.

Este cambio permitió mejorar la comunicación entre el tutor y el alumno, ya que los avances eran más pequeños y rápidos, lo cual conllevó un incremento en la productividad. Por otro lado, al no tener tantas horas seguidas de trabajo tras la jornada laboral, el alumno comenzó a encontrarse mucho más cómodo a nivel psicológico con la realización del proyecto.

5.3 – Análisis de casos de uso

Se ha realizado un análisis de los casos de uso que surgen de la interacción entre un usuario que quiere jugar una partida, y el sistema. A continuación, se presenta la lista de casos de uso del sistema, que se pueden ver reflejados en el diagrama de casos de uso, mostrado en la Ilustración 6:

- Iniciar sesión
- Cerrar sesión
- Ver partidas
- Crear partida
- Eliminar partida
- Ver detalle partida
- Seleccionar jugador
- Empezar partida
- Establecer carta como cabecera
- Jugar carta
- Jugar carta por influencia
- Jugar carta por desestabilización

- Jugar carta por texto
- Jugar carta por puntuación
- Jugar carta por New World Oder

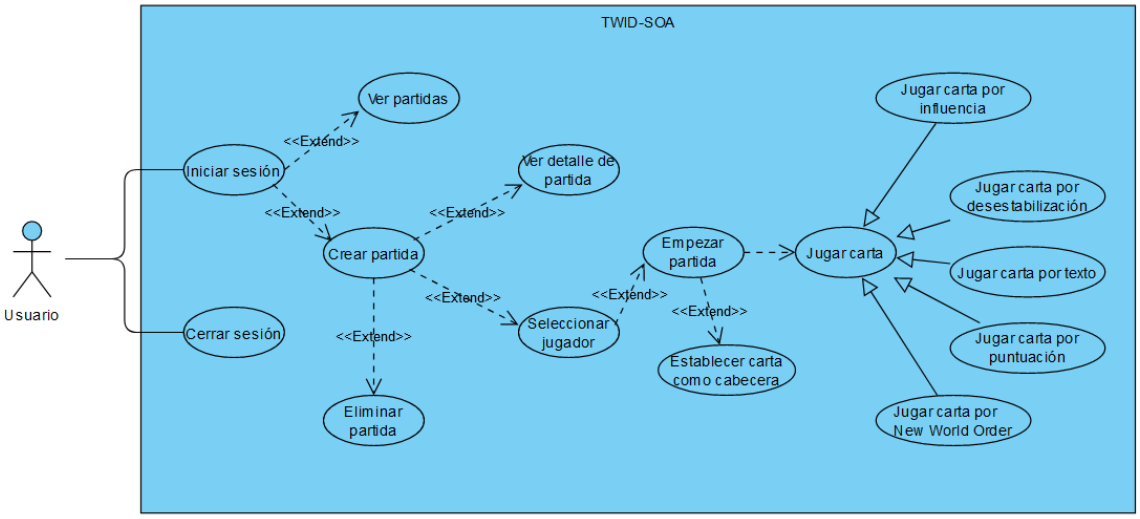

Ilustración 6. Diagrama de casos de uso.

5.4 – Análisis y diseño de la arquitectura SOA

Como se explicó en la sección "1.1 – Idea del proyecto", la idea consiste en dividir la funcionalidad en dos capas:

- La capa de recursos, que contendrá servicios básicos que se encargarán de manejar y permitir realizar operaciones sobre los recursos del juego, como pueden ser los países, las regiones, la puntuación de los jugadores o las rondas. Esta capa no lleva incluida nada más que la lógica necesaria para asegurarse de que las operaciones se realizan con valores válidos. Por ejemplo, si hablamos de las operaciones sobre el número de ronda, únicamente se comprueba si el número de ronda es válido (valor entre 1 y 8, ambos incluidos), no si la operación tiene sentido según las reglas del juego (como puede ser bajar el número de ronda, algo que va contra las reglas). Esta capa solamente será accesible desde la capa de control, de modo que el cliente no pueda modificar los recursos del juego.
- La capa de control, que almacenará los servicios encargados de llevar la lógica de negocio y de realizar las modificaciones oportunas sobre los recursos del juego. Deberá implementar las reglas del juego y las acciones que realizarán los jugadores, de modo que cuando un jugador realice una jugada, se desencadenen las operaciones pertinentes sobre la capa de recursos, siguiendo en todo momento las reglas del juego.

5.5 – Diseño de los servicios necesarios

Para realizar el diseño de los servicios, se ha seguido fielmente las normas del juego y se ha tenido en cuenta los recursos y las acciones de los jugadores. Estas normas se encuentran disponibles en el "Anexo I – The Wall is Down", el cual ha sido provisto en su totalidad por el tutor [1].

5.5.1 – Capa de recursos

En cuanto a la capa de recursos, los principales recursos son el tablero y las cartas. Respecto al tablero, contiene la ronda, la puntuación de los jugadores y el mapa del mundo. El mapa se divide en regiones, las cuales contienen países, que pueden ser influenciados por cada uno de los jugadores, y otro recurso denominado "New World Order", que permite a los jugadores tomar el control de ámbitos de la realidad que no se pueden reflejar en el mapa, como pueden ser la influencia sobre los medios de comunicación o el control de los mercados financieros.

Las cartas se dividen en, por un lado, los mazos de cartas, y por otro, las cartas de los jugadores. Hay 3 mazos de cartas, el principal, de donde se roban las cartas de juego, el mazo de descartes, a donde van a parar las cartas que se pueden volver a usar, y el mazo de cartas eliminadas, para las cartas de un solo uso. Las cartas de los jugadores se dividen en cartas en juego, y cartas en la mano de cada jugador. De las últimas, cada jugador puede tener o no una carta de cabecera, dependiendo de la fase del juego en la que se encuentre la ronda.

En la Ilustración 7 se puede observar el tablero de juego, con las zonas más importantes marcadas con una pequeña explicación.

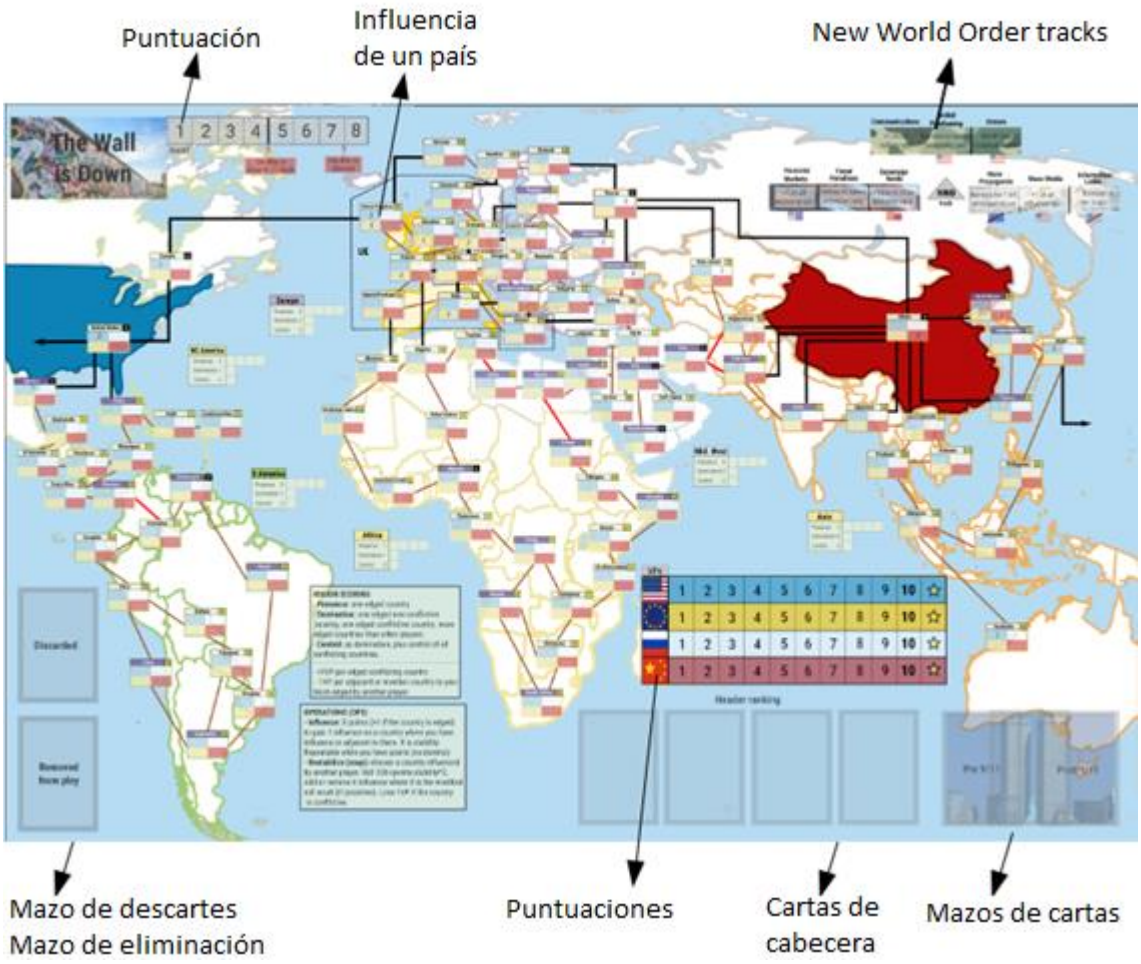

Ilustración 7. Tablero de juego. [https://drive.google.com/file/d/1NQ0MBrjblynNabtVV32vSwMKB69BZAUD/view]

5.5.2 – Capa de control

En cuanto a la capa de control, por un lado, se tiene el recurso de autenticación de jugadores, que permite iniciar sesión como invitado (controlando el máximo de sesiones abiertas por dirección IP) y que tiene una validez de 24 horas, además de permitir cerrar la sesión, y por otro lado el recurso de juego.

El recurso de juego permite obtener los juegos de los que se forma parte, crear nuevos juegos (controlando el número máximo de juegos por jugador), eliminar juegos y comenzar un juego recién creado. Una vez se ha creado un juego permite seleccionar el jugador que se desea utilizar, y cuando se comience el juego, todas las acciones relativas a las cartas. Estas acciones son ver la información de una carta, ver la mano de cartas del jugador, ver las cartas en juego, establecer una carta como carta de cabecera, modificar el recurso "New World Order" o jugar una carta de las diferentes formas posibles existentes.

5.5.3 – Resumen del diseño

De forma esquemática, el diseño obtenido es el que se presenta desglosado en la Ilustración 8:

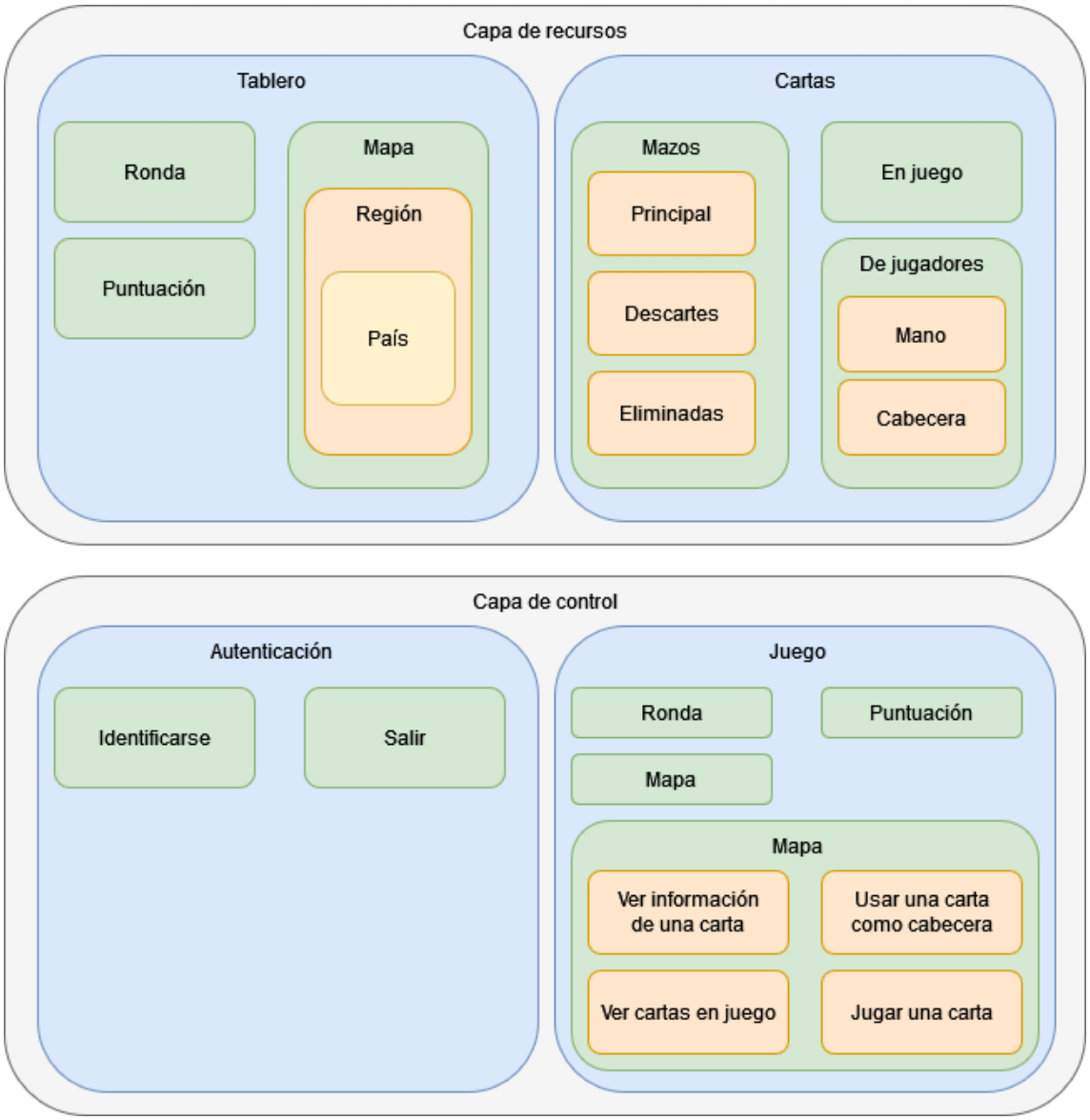

Ilustración 8. Esquema del diseño de los servicios.

5.5.4 – Implementación del diseño

Para dejar plasmado el diseño explicado anteriormente, se han creado dos ficheros que contienen la especificación del diseño de los servicios en términos técnicos, usando el estándar OpenAPI 3.0, explicado en el apartado "3.1 – API". Estos archivos se encuentran bajo la carpeta "specification", y pueden ser visualizados si se abren en la página https://editor.swagger.io, pudiendo ver el resultado final que generará la herramienta Swagger, explicada en el apartado "4.2.5 – Swagger".

5.6 – Creación de los servicios necesarios

Una vez creado el diseño de los servicios, se ha usado las especificaciones técnicas para implementarlos. Se ha utilizado el framework FastAPI sobre el lenguaje Python. Se ha seguido la guía proporcionada en la página oficial de la herramienta, disponible en las referencias [7]. Cabe destacar que no se poseía experiencia previa con este marco de trabajo, pero el desarrollo ha sido sencillo y sin grandes complicaciones, debido a la gran calidad de la documentación, que explica en profundidad cada funcionalidad que proporciona la herramienta.

En el "Anexo II – Documentación de usuario" se explica paso a paso y con ejemplos cómo un usuario (o sistema, como una interfaz gráfica) puede y debe comunicarse con el sistema, y en el "Anexo III – Documentación técnica" se detalla la estructura del código, las funciones de cada archivo y el flujo de llamadas entre los ficheros.

5.6.1 – Creación del servicio de recursos

Para implementar el servicio se ha seguido el enfoque de la POO, y se ha creado una clase para el recurso "tablero", y otra para el recurso "cartas". Ambas clases deben tomar unos valores iniciales, que serán los que se muestren al iniciar la partida. Para ello, se ha decidido crear un fichero JSON que contenga los valores iniciales de los recursos. De esta forma, al crear una instancia de alguna de las dos clases, el objeto lee el fichero JSON de memoria y lo convierte en un objeto diccionario, nativo del lenguaje Python, almacenándolo en memoria principal. De esta forma se cargan los valores predeterminados, y la instancia queda lista para ser usada.

También se ha implementado una clase "partida", que es la que contiene las clases "tablero" y "cartas", y es creada cuando un usuario decide empezar un juego. De esta forma se logra que pueda haber varias partidas de forma concurrente y totalmente transparente a los usuarios. Esta clase es la que permite al servicio de control manipular los recursos del juego, y esto es posible debido a que expone una API que cumple con los requisitos de la implementación del diseño, explicado en la sección "5.2.4 – Implementación del diseño". En la Ilustración 9 se muestra un diagrama de la arquitectura anteriormente descrita.

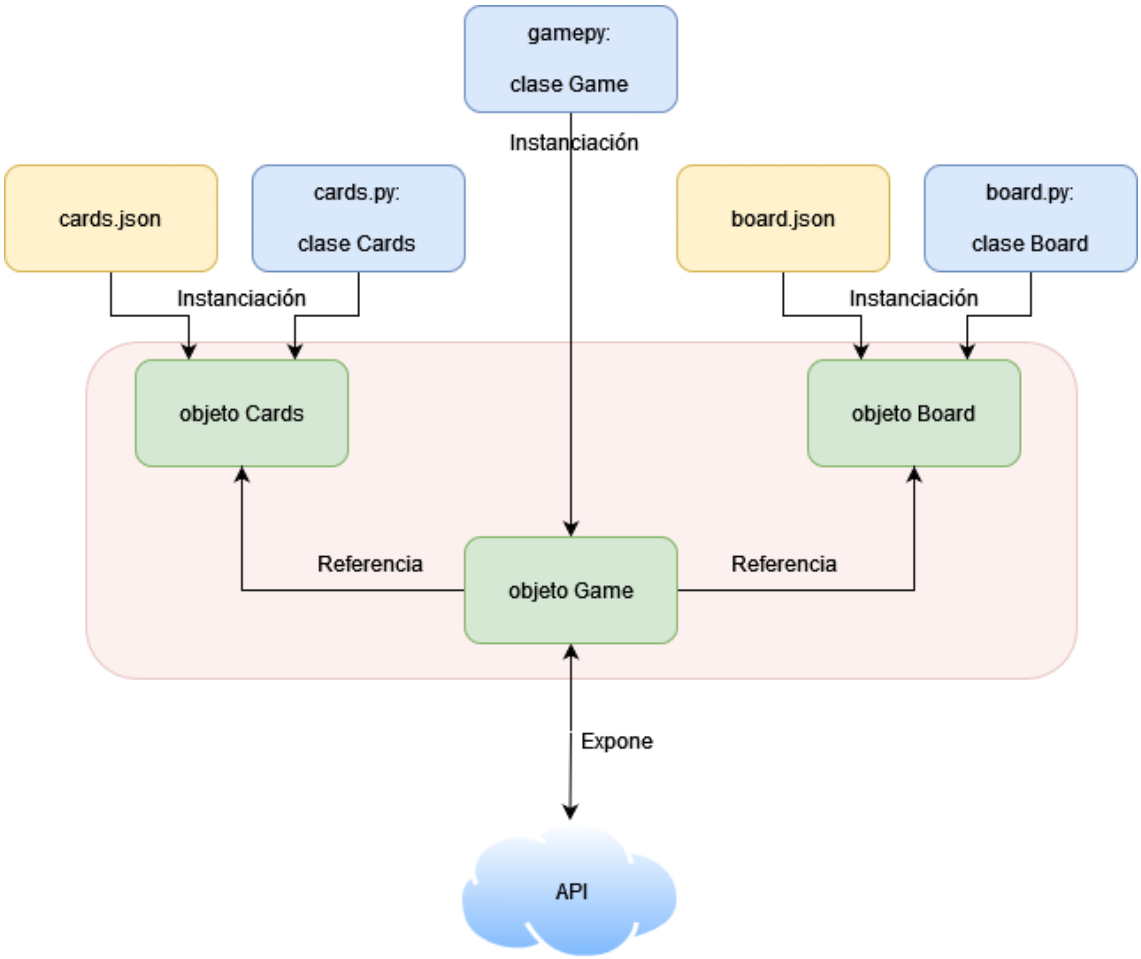

Ilustración 9. Diagrama de arquitectura del servicio de recursos.

5.6.2 – Creación del servicio de control

Para la implementación de este servicio se ha usado el enfoque de la POO, al igual que con el servicio de recursos, pero en este servicio se ha creado una clase para el recurso "partida". De esta forma, cada vez que un usuario crea una partida, se crea una instancia de este objeto, el cual contiene los métodos que permiten realizar acciones a los usuarios.

Para la creación de los endpoints se ha usado el framework Pydantic, descrito en la sección "4.2.4 – Pydantic". Se han creado clases que permiten validar los tipos de datos de entrada, asegurándose de que la entrada del usuario no tiene valores erróneos. Con esto, aparte de conseguir devolver al usuario un error claro en caso de que realice una entrada de datos incorrecta, se aumenta la seguridad del servicio, evitando ataques de inyección, entre otros. Esta clase es la que permite al usuario comunicarse con el propio servicio, ya que expone una API que cumple con los requisitos de la implementación del diseño, explicado en la sección "5.2.4 – Implementación del diseño". Para poder manipular los recursos del juego, esta clase hace uso de la API que expone el servicio de

recursos. En la Ilustración 10 se puede ver un diagrama de la arquitectura anteriormente descrita.

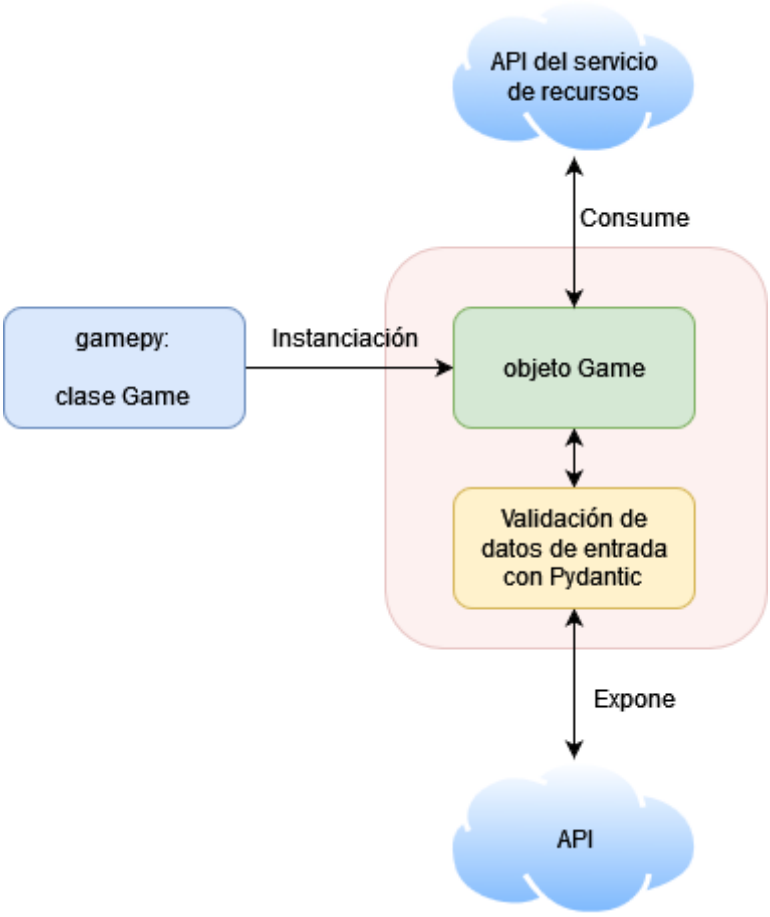

Ilustración 10. Diagrama de arquitectura del servicio de control.

5.7 – Preparación de los contenedores de los servicios con la herramienta Docker

Una vez se han implementado los servicios del juego, se crean los contenedores de Docker. Para ello se ha seguido la guía de la documentación oficial de FastAPI, disponible en las referencias [8]. Primero se ha generado el archivo de dependencias para cada servicio, llamado "requirements.txt". En este archivo se almacenan los paquetes de Python que cada servicio necesita para funcionar, junto con sus versiones. Una vez hecho eso, se puede usar el archivo "Dockerfile" de ejemplo disponible en la página, modificando la versión de la imagen base de Python, el nombre del directorio de trabajo, y el puerto a exponer por el servicio para construir nuestra imagen del servicio. Esto se ha realizado para los dos servicios, de manera que se pongan a la escucha puertos diferentes. Se han elegido los puertos 8000 y 8001, ya que son puertos altos que generalmente están libres.

5.8 – Puesta en producción de los servicios en un servidor del departamento

Tras contactar con José Andrés Vicente Lober, administrador de sistemas del Departamento de Informática y Automática, se ha proporcionado el servidor prodiasv30.fis.usal.es como plataforma para realizar la puesta en producción del sistema. Este servidor ejecuta un sistema operativo Ubuntu 22.04 LTS, perfecto para el propósito que se le va a encomendar, y se encuentra disponible en https://prodiasv30.fis.usal.es/docs.

El servidor ha sido proporcionado con una instalación limpia del sistema, de modo que cae sobre el alumno la responsabilidad de realizar la correspondiente configuración de las herramientas y del propio sistema operativo. Para ello, se han seguido los pasos que se describen a continuación.

5.8.1 – Aseguración de cuentas de usuario

El primer paso consiste en asegurar las cuentas de usuario. Para ello, dado que para el primer acceso al servidor se ha proporcionado un usuario con una contraseña insegura, se cambia la contraseña de este usuario por una con una longitud de 20 caracteres alfanuméricos y con símbolos (“#$~%&/()-_”, entre otros), y eliminamos el acceso de cualquier otro usuario al sistema (con la excepción del usuario de José Andrés, en caso de que ocurra una incidencia y se necesite de su ayuda). También se ha reservado el acceso de superusuario para el usuario del alumno y José Andrés.

El siguiente paso es eliminar la posibilidad de acceso por SSH para el usuario root, modificando el fichero de configuración del servicio SSH “/etc/ssh/sshd_config”.

5.8.2 – Protección del sistema

Después de asegurar las cuentas de usuario, se procede a proteger el sistema, usando las herramientas Fail2ban y ufw.

La herramienta Fail2ban consiste en un servicio de escucha de intentos de conexión por SSH, y si encuentra que una IP ha realizado demasiados intentos fallidos, la bloquea durante un tiempo determinado. Todos los parámetros se pueden personalizar en el archivo de configuración del servicio “/etc/fail2ban/jail.conf”.

La herramienta ufw permite manejar el servicio iptables del kernel de Linux de forma sencilla, de manera que se puedan manipular reglas de firewall rápida

y eficazmente. Se han creado reglas para permitir exclusivamente los puertos 22 (SSH), 80 (HTTP, redireccionará al 443) y 443 (HTTPS).

5.8.3 – Instalación de herramientas

El siguiente paso consiste en realizar la instalación de la herramienta Docker, y para ello se ha seguido las instrucciones de la documentación oficial, disponible en la página https://docs.docker.com/engine/install/ubuntu/.

5.8.4 – Despliegue del servicio

Para realizar el despliegue del servicio se ha utilizado la herramienta Docker Compose, explicada en la sección "4.2.8 – Docker Compose". Se ha creado una configuración de despliegue con 3 contenedores: Uno para el servicio de recursos, otro para el servicio de control, y otro con un servidor Nginx (explicado en la sección "4.2.10 – Nginx"), para el enrutamiento de peticiones. Se ha decidido hacerlo de esta forma para que no haya que modificar los puertos de los contenedores de los servicios y para poder tener comunicación HTTPS en lugar de HTTP.

Se ha configurado el servidor Nginx para redireccionar todas las peticiones que vayan por el puerto 80 (HTTP) al puerto 443 (HTTPS), y dado que las conexiones HTTPS necesitan un certificado, se ha valorado la opción de usar los certificados gratuitos y mundialmente aceptados de Let's Encrypt, pero dado que el servidor se encuentra dentro de la red de la USAL y no hay conexión directa con Internet, la única opción posible ha sido la de generar un certificado auto firmado. Esto hará que salga una advertencia en el navegador de los clientes al acceder al servicio, pero la conexión será segura.

A continuación, se presenta la Ilustración 11, donde se puede ver la arquitectura anteriormente descrita:

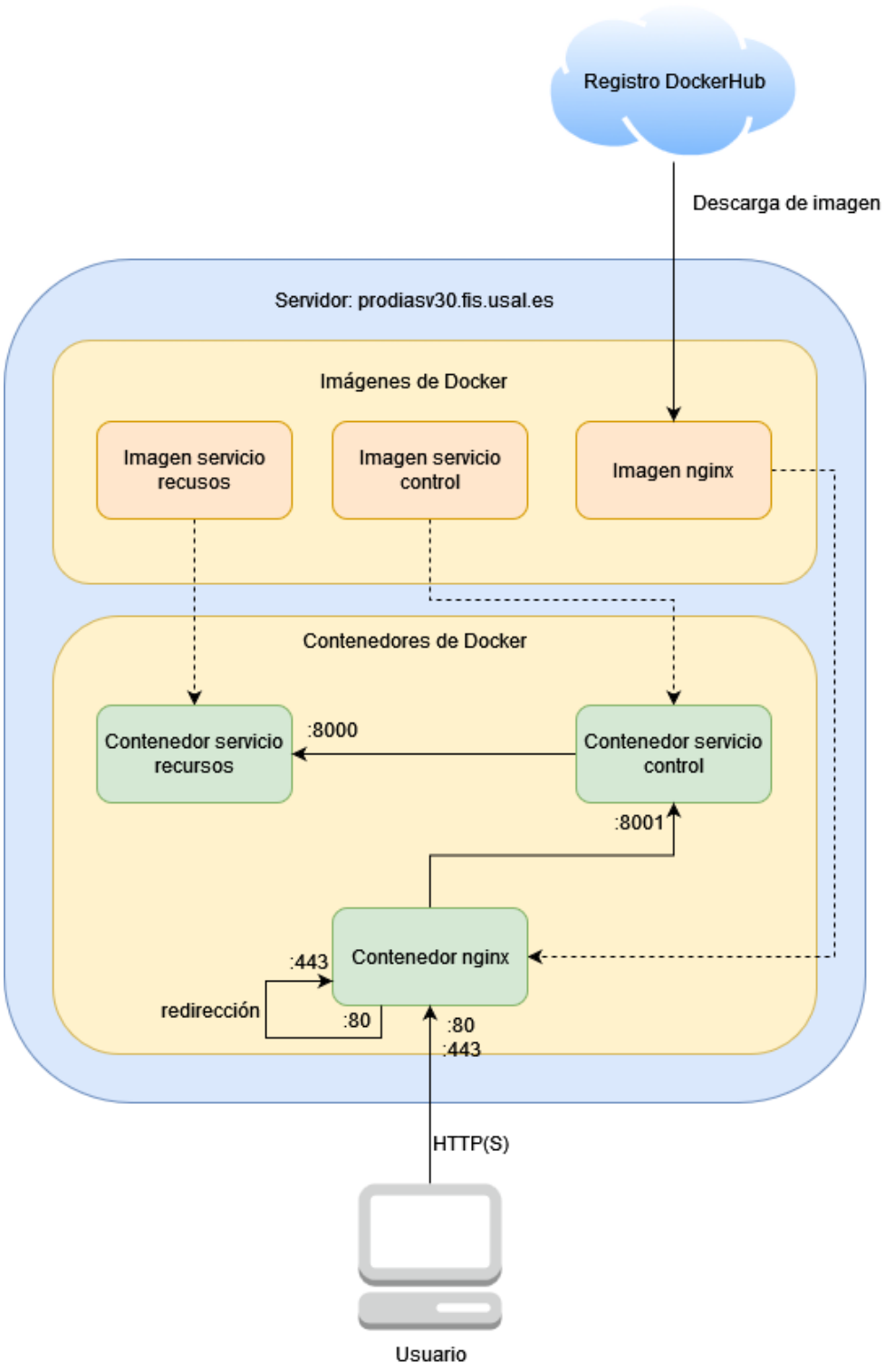

Ilustración 11. Arquitectura del despliegue con Docker Compose.

6 – Conclusiones

Una vez acabado el proyecto, resulta interesante echar la vista atrás y analizar el proceso seguido y las dificultades encontradas, así como el resultado final obtenido. De esta forma se pueden extraer una serie de conclusiones, tanto a nivel personal, como técnico y profesional. Dichas conclusiones se presentan a continuación.

6.1 – Producto final

Se ha obtenido un servidor funcional que se encuentra disponible por comunicación segura en https://prodiasv30.fis.usal.es/docs y el repositorio de código está disponible en https://github.com/javiervidrua/TWID-SOA bajo la licencia MIT, que permite a cualquier persona modificar libremente el código, bajo su propia responsabilidad.

6.2 – Franjas y horario de trabajo

Como se explicó en la sección "5.2 – Franjas y horario de trabajo", la idea inicial del horario de trabajo tuvo que ser modificada debido a los impedimentos que se encontraron, según se fue avanzando en el proyecto. Tras adaptar mejor el horario, se notó una gran mejoría en el desarrollo del proyecto.

La conclusión obtenida es que tener un buen horario de trabajo, que permita tener descansos y una buena comunicación de manera frecuente entre las partes implicadas en el proyecto, es clave para asegurar el éxito de este.

6.3 – Cumplimiento de los objetivos inicialmente propuestos

Se ha cumplido con los objetivos propuestos inicialmente:

- Se ha analizado y diseñado la arquitectura SOA del sistema, se ha diseñado la especificación técnica de los servicios usando el estándar OpenAPI 3.0 y se han implementado acorde a dicha especificación.
- Se han creado imágenes de Docker de los servicios, además de un fichero de configuración para la herramienta Docker Compose para tener una forma rápida y sencilla de manejar el despliegue de los servicios, así como de asegurarse su disponibilidad.
- Se ha configurado un servidor GNU/Linux desde cero, pasando por la aseguración de las cuentas de usuario, la protección del sistema y la instalación de las herramientas necesarias.

- Se ha puesto el sistema en producción en el servidor configurado, funcionando de manera segura (HTTPS).
- Se ha conseguido realizar una planificación temporal que ha permitido compatibilizar el trabajo a jornada completa con el curso y con la realización del Trabajo de Fin de Máster.
- Se ha conseguido identificar, planear y realizar las tareas de forma satisfactoria, así como aprender nuevas tecnologías como FastAPI.
- Se ha conseguido llevar el control de versiones del proyecto con GitHub, a la vez que se implementaba código limpio, mantenible y que sigue estándares (como REST) y buenas prácticas de programación.

7 – Líneas futuras de trabajo

Así como se han obtenido las conclusiones comentadas en el anterior apartado, también se han encontrado puntos de mejora, causados tanto por falta de tiempo como por falta de conocimientos técnicos, que deben ser anotados para poder perfeccionar el sistema en un futuro.

Algo que ha causado bastantes contratiempos durante la implementación de los servicios ha sido la realización de las pruebas de la funcionalidad que se crea. Estas pruebas se han realizado a mano, haciendo flujos completos que imitan las acciones del usuario, llamando a los endpoints correspondientes de cada servicio. Esto ha ocasionado que las pruebas de una funcionalidad sean excesivamente largas, especialmente en los casos en los que se producían errores, y había que tratar de reproducirlos para encontrar el fallo.

La mejor solución es la de adoptar la metodología Test Driven Design (TDD), que consiste en escribir las pruebas de la funcionalidad antes de implementar dicha funcionalidad. De esta forma siempre va a haber la forma de probar una funcionalidad, y tiene una fácil integración en los flujos CI/CD.

Realizar esto sería posible sin mucha complicación, dado que FastAPI tiene integración total con el framework de pruebas Pytest, como viene explicado en la documentación oficial https://fastapi.tiangolo.com/tutorial/testing/.

Otro tema pendiente, debido a la falta de tiempo, consiste en la finalización de la lógica correspondiente al endpoint de jugar una carta por su texto, ya que, por el momento, no todas las cartas tienen esta funcionalidad implementada. Para ello, habría que implementar, haciendo uso del servicio de recursos, las acciones correspondientes al texto de cada carta. Esta tarea no es complicada, pero conlleva una larga cantidad de tiempo, principalmente debido a la cantidad de pruebas a realizar. Por eso, lo ideal sería realizarla una vez se hayan escrito las pruebas para el resto de las funcionalidades.

8 – Referencias

1. Rodrigo Santamaría Vicente – "*The Wall is Down*" [https://drive.google.com/file/d/1NQ0MBrjblynNabtVV32vSwMKB69BZAUD/view]. Accedido: 17-01-2023.
2. Statista – "*Global board games market value from 2017 to 2023*" [https://www.statista.com/statistics/829285/global-board-games-market-value/]. Accedido: 17-01-2023.
3. 20 minutos – "*La venta de juegos de mesa se disparó un 18.3% el año de la pandemia: Virus y el clásico Monopoly triunfan en España*" [https://www.20minutos.es/noticia/4674789/0/la-venta-de-juegos-de-mesa-se-disparo-un-18-el-ano-de-la-pandemia-virus-y-monopoly-lideraron-la-demanda-nacional/]Accedido: 17-01-2023.
4. Testdrive.io – "*FastAPI streamlit*" [https://testdriven.io/blog/fastapi-streamlit/]. Accedido: 17-01-2023.
5. Docker – "*Overview*" [https://docs.docker.com/get-started/overview/]. Accedido: 17-01-2023.
6. Nvie – "*A successful git branching model*" [https://nvie.com/posts/a-successful-git-branching-model/]. Accedido: 17-01-2023.
7. FastAPI – "*First steps*" [https://fastapi.tiangolo.com/tutorial/first-steps/]. Accedido: 17-01-2023.
8. FastAPI – "*FastAPI in containers - Docker*" [https://fastapi.tiangolo.com/deployment/docker/]. Accedido: 17-01-2023.