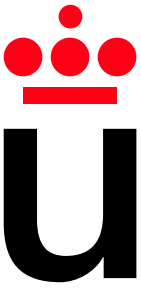

Universidad
Rey Juan Carlos

Escuela Técnica Superior
de Ingeniería Informática

Grado en Ingeniería de Computadores

Curso 2023-2024

Trabajo Fin de Grado

OUTSIDER: UN JUEGO ONLINE EN TIEMPO REAL

Autor: Javier Raúl Alonso Tejera
Tutor: Michel Maes Bermejo

Agradecimientos

Quiero expresar mi más profundo agradecimiento a todas las personas que han hecho viable la realización de este trabajo de fin de grado.

En primer lugar a los docentes de de la Universidad Rey Juan Carlos, en especial a mi tutor, Michel Maes Bermejo, por su apoyo constante a lo largo de todo el proceso de desarrollo y documentación. Sus amplios conocimientos sobre la materia y sus ideas para el avance y mejora del proyecto han sido claves para todo el proceso. Además, siempre ha estado disponible para cualquier duda técnica ofreciendo siempre recursos adicionales y alternativas a diversos obstáculos que hemos ido encontrando. Destaco su genial labor, tanto de docente como de tutor.

Agradezco también a mi familia y amigos por su constante apoyo moral durante estos años de trabajo académico. A mis padres y mi hermana, por su sacrificio personal y por enseñarme a tener paciencia y perseverancia aún estando separados por cientos de kilómetros. A mis amigos, por haberme hecho pasar unos años inolvidables llenos de alegría además de ser los mejores compañeros de carrera que uno pueda desear.

Resumen

En esta memoria se va a documentar todo el proceso de desarrollo de Outsider, un juego multijugador de adivinanza de palabras y roles ocultos accesible a través de la web.

La realización de este proyecto ha estado motivada principalmente por la oportunidad de aprender y experimentar con tecnologías web en tiempo real como websocket. Por ello, el eje central de este trabajo gira en torno al uso de este tipo de tecnologías y su aplicación en un caso concreto.

La aplicación consta de dos partes principales: Un fronted dedicado a la presentación visual de una interfaz web y un servidor backend encargado de la gestión de la lógica del juego, estrechamente vinculada con el uso de websockets y la comunicación entre usuarios.

A lo largo de esta memoria se detallarán las tecnologías empleadas, como Django, Django Channels y Vue, así como detalles sobre su implementación en el contexto de este proyecto.

Se describirán los objetivos y requisitos del proyecto, la arquitectura software de la aplicación y los diseños visuales iniciales. También se explicará el funcionamiento e implementación tanto del backend como del frontend y se especificarán los problemas encontrados a lo largo del desarrollo. Finalmente, se detallará la implementación del testing propuesto y el despliegue final de la aplicación a través de AWS.

Palabras clave:

- Websocket
- Django
- Django Channels
- Testing
- Vue
- Docker
- AWS (Amazon Web Services)

Índice de contenidos

Índice de figuras

Índice de códigos

1 Introducción

Este documento aborda y explica el desarrollo de este proyecto de fin de grado, desde su concepción inicial y las motivaciones que lo impulsaron, hasta los detalles finales de su implementación.

1.1. Descripción general

El objetivo general del proyecto es el desarrollo de un juego multijugador web de adivinanza de palabras y roles ocultos. El principal punto de interés radica en el uso de tecnologías en tiempo real, específicamente de tecnologías websocket, posibilitando el juego online entre los jugadores de forma fiable y óptima.

Las mecánicas base del juego son sencillas y son más una adaptación de un tipo de juego que normalmente se realiza sin el uso de tecnologías informáticas. El concepto más básico del juego se podría resumir mediante los siguientes puntos:

1. En este juego, todos los jugadores juegan como "Inocentes", a excepción de uno de ellos, definido como el "Outsider".
2. Al empezar la partida, los jugadores Inocentes reciben una palabra secreta o contraseña, como por ejemplo, "Hoja".
3. Siguiendo un orden secuencial (con el primer jugador elegido al azar), cada participante debe escribir una palabra relacionada con su palabra clave para indicar a los demás que conocen la palabra asignada.

4. Por ejemplo si la palabra clave es la palabra “Hoja”, unas palabras que alienten a esta o palabra clave podrían ser “Árbol”, “Libro” o “Afeitado”.

5. El jugador designado como Outsider, quien no tiene conocimiento de la palabra secreta, debe tratar de deducirla a partir de las palabras previamente mencionadas y decir una palabra que no levante sospechas.

6. Después de que todos los jugadores hayan compartido una palabra, se lleva a cabo una votación simultánea, donde cada jugador vota por la persona que creen que es el Outsider.

7. El Outsider ganará si no es el jugador más votado. Por otra parte, el resto de los jugadores Inocentes ganarán si logran descubrir al Outsider y este llega a ser el jugador más votado.

Dadas estas reglas de funcionamiento del juego, el desarrollo del proyecto consiste principalmente en interpretar y diseñar una aplicación web capaz de poder gestionar todos estos puntos. En el próximo capítulo de Objetivos (2.1), se tratará con más detalle esta definición de reglas.

Además de generar una aplicación/juego que emule estas reglas, es bastante destacable el trabajo adicional relacionado con la gestión informática que implica desarrollar una aplicación web: desde el diseño de interfaces usables hasta su despliegue y testeo, todos estos apartados se trataran en detalle como parte del desarrollo.

1.2. Motivación

La principal motivación para la realización de este proyecto es el aprendizaje propio. La idea del juego surge por parte del tutor del TFG, Michel Maes Bermejo, que ve muy interesante la creación de una aplicación que haga uso de tecnologías en tiempo real. También ha sido una figura clave en el desarrollo iterativo del proyecto, proponiendo mejoras y comentarios a las diferentes versiones de la aplicación.

Dicho esto, a día de hoy hay innumerables proyectos, aplicaciones y juegos que persiguen conseguir un rendimiento económico o lúdico, sin embargo, también es importante destacar el papel de la investigación y el aprendizaje, especialmente en el ámbito académico en el que se sitúa este trabajo de fin grado.

Las empresas y organizaciones buscan desarrolladores software altamente capacitados, pero también especializados en materias en concreto. Destaca más un desarrollador que sepa sobre una tecnología útil y difícil de aprender sobre otro que sepa sobre otra tecnología menos útil o mucho más accesible. Aquí es donde

entran en juego los websockets, cuyas implementaciones suelen dar problemas a los desarrolladores.

Muchos estudiantes hemos realizado pequeños proyectos y pruebas con websockets y tecnologías análogas, pero es muy diferente desarrollar toda una aplicación, realizar testing e incluso llevar a cabo un despliegue en torno a tecnologías en tiempo real. Hay que tener en cuenta muchas variables y por otra parte se tratan de tecnologías útiles con infinidad de aplicaciones. Desde un simple chat en línea al uso de websockets en grandes proyectos, el uso de aplicaciones web en tiempo real es muy práctico.

Desde mi experiencia previa en desarrollo de aplicaciones móviles, un desarrollador no es consciente de la cantidad de lógica que se necesita para manejar una base de datos en lo que a priori se presenta como una sencilla aplicación. Todo el trabajo subyacente es muy poco conocido pero suele ser de gran importancia. En este trabajo previo, se destaca que se planteó varias veces la necesidad de poder crear conexiones websockets entre usuarios de la aplicación para poder compartir datos de forma inmediata.

Las aplicaciones que usamos en nuestros teléfonos y dispositivos, los videojuegos online o los servicios de streaming dependen de una estructura muy bien pensada para cada caso específico. Por ello, es una necesidad por parte de un desarrollador software, al menos entender, como funcionan todos estos sistemas. Mi objetivo es explorar y aprender lo máximo de la mayor cantidad de tecnologías software para desarrollarme como profesional así como para enseñar las posibilidades que ofrecen estas diversas herramientas.

2 Objetivos

En esta sección se definirán los objetivos principales del proyecto así como de otros objetivos y requisitos secundarios que han ido definiéndose a lo largo del proyecto.

2.1. Objetivos funcionales del juego

En primera instancia destacamos las funcionalidades básicas que debería cumplir el juego como producto software, es decir, relacionados con el usuario de forma directa.

2.1.1. Funcionalidad básica, lobby y conexiones

Además del propio juego, es necesario dar las capacidades y herramientas necesarias a los usuarios para poder organizar partidas de forma sencilla. Para ello se plantea la creación de una página de inicio que permita al usuario poder crear salas de juego o unirse a salas ya creadas.

El objetivo en detalle sería permitir a todos los usuarios de la aplicación poder jugar a diferentes partidas, teniendo en cuenta varias limitaciones como el que un jugador no debería poder acceder a una partida ya empezada o evitar la creación de salas con el mismo nombre. De forma adicional, en esta página se deberían mostrar instrucciones que expliquen el funcionamiento del juego.

También sería esencial tener una pantalla pre-partida en la cual se vayan uniendo los jugadores antes de comenzar el juego. En esta pantalla se propone añadir un chat para los jugadores además de mostrar la información pertinente al estado de la partida: número de jugadores preparados, código de la sala para que se puedan unir más jugadores, etc.

2.1.2. Juego sencillo

En primer lugar, se plantea poder jugar de forma sencilla solo una ronda en la cual se deciden los roles de los jugadores y se les indica una palabra clave a los jugadores Inocentes. Esta palabra clave es desconocida por el Outsider y el resto de jugadores Inocentes deberán demostrar que la conocen mientras evitan desvelarla al verdadero jugador Outsider. Se destaca que ninguno de los jugadores debería conocer a ciencia cierta el rol de otro, por ello, se trata de un juego con un alto factor de interacción social, donde la intuición y la discreción son elementos clave.

Con la palabra clave revelada a los jugadores Inocentes, se procede con la lógica de juego estándar: cada jugador siguiendo un orden secuencial, escribirá una palabra relacionada con su palabra clave para indicar a los demás que conocen la palabra asignada; después de que todos los jugadores hayan compartido una palabra, se lleva a cabo una votación simultánea, donde cada jugador vota por la persona que creen que es el Outsider.

El Outsider ganará si no es el jugador más votado. Por otra parte, el resto de los jugadores Inocentes ganarán si logran descubrir al Outsider y este llega a ser el jugador más votado.

El objetivo es llevar a cabo la implementación de esta lógica de la forma más consistente posible, trabajando en detalle las conexiones websocket.

2.1.3. Última oportunidad

Habiendo implementado el juego sencillo, se quiere añadir nuevas funcionalidades base. En primer lugar, dado el transcurso normal del juego, tras la votación simultánea, en el caso de que el Outsider reciba la mayoría de los votos, se le otorgará una última oportunidad para ganar adivinando la palabra clave.

2.1.4. Múltiples rondas

Cuando existen más de tres jugadores en una partida, se detecta la necesidad de poder jugar varias rondas, ya que, si no se elimina al Outsider inicialmente y

se elimina a un jugador Inocente, podrían seguir jugando el resto de jugadores rondas adicionales.

Los jugadores eliminados pasan a ser espectadores mientras el resto sigue jugando hasta que no se pueda seguir el juego porque solo queden un jugador Outsider y otro Inocente (en este caso ganaría el Outsider de forma definitiva) o se elimine mediante votación al Outsider en cuestión.

2.1.5. Varios Outsider

Relacionado con el anterior punto, en partidas con varios jugadores se ve necesario aumentar el número de Outsiders en aras de una jugabilidad más interesante. De esta manera, se plantea que cuando el número de jugadores sea mayor que seis, simplemente sean dos jugadores Outsider en la partida.

En estos casos el Outsider solo tendrá la oportunidad de adivinar la palabra clave si es el último Outsider en la partida. Por contraparte, si todavía hay dos Outsiders jugando y solo quedan otros dos jugadores Inocentes sin eliminar, ganaría el bando Outsider al tener ventaja en los votos.

2.2. Objetivos secundarios y tecnológicos

Dados los objetivos/reglas principales que debería cumplir el juego, se añaden varios puntos adicionales que se quieren tratar como objetivos secundarios del proyecto.

2.2.1. Plataforma de juego

Se quiere aprovechar el uso de un entorno web para hacer más accesible el juego a diferente tipo de usuarios.

El objetivo es hacer usable la aplicación en el mayor número de dispositivos posibles diferenciando principalmente entre teléfonos móviles y equipos de escritorio (ordenadores y portátiles esencialmente). Se diferencia entre estos dos tipos de dispositivos porque también se prevén dos tipos de juego de forma predominante:

- Juego presencial en un mismo espacio físico.
- Juego remoto.

Se entiende que si un grupo quiere jugar en un espacio físico común, por ejemplo, en un cumpleaños en la casa de uno de los jugadores, en la mayoría de casos

todos los jugadores jugarán con sus propios dispositivos móviles, principalmente smartphones.

Por otro lado, también se ve bastante común que otro grupo de jugadores quiera jugar de forma remota, cada uno en una localización diferente. En este caso, se podría asumir que la mayoría de jugadores intentarían jugar con sus dispositivos de escritorio.

2.2.2. Testing

Se propone de forma adicional trabajar ciertos elementos de Integración Continua (CI, por sus siglas en inglés), especialmente el testing del software, ya que, es una práctica esencial en la industria y resulta bastante llamativo trabajar el testing con tecnologías en tiempo real.

2.2.3. Despliegue

Para poder enseñar el resultado del proyecto de la forma más accesible posible, también se quiere trabajar con tecnologías de AWS (Amazon Web Services) para poder tener la aplicación desplegada y totalmente disponible para cualquier usuario a la hora de acceder a esta.

De esta forma se pretende aprender sobre tecnologías de despliegue y de crear una mejor presentación final del proyecto.

2.2.4. Feedback de usuarios

Para finalizar con la descripción inicial de objetivos, se pretende realizar una pequeña prueba con usuarios a mitad de desarrollo con el objetivo de encontrar bugs y pequeñas mejoras para el juego, lo que generará su propia lista de cambios a realizar.

2.3. Resumen de objetivos

1. Gestionar las conexiones entre jugadores mediante salas.
2. Creación del juego básico.
3. Añadir una mecánica adicional a la hora de eliminar al último jugador Outsider.

4. Permitir jugar varias rondas.
5. Añadir más jugadores Outsider al juego si hay muchos jugadores en partida.
6. Gestionar la usabilidad de la aplicación en diferentes dispositivos.
7. Tener en cuenta el feedback de los usuarios. Testing manual.
8. Realización de tests para la lógica en tiempo real. Testing automático.
9. Desplegar la aplicación a través de tecnologías modernas.

3

Herramientas y Metodología

A continuación se expondrá brevemente el uso de tecnologías así como la metodología de trabajo que se ha seguido para la elaboración de la aplicación.

3.1. Herramientas y tecnologías usadas

3.1.1. Django - Django Channels

Las bases del proyecto se fundamentan en el uso de Django [1] como tecnología de backend.

Además de conocer en profundidad cómo funcionan este framework y Python (que es el lenguaje con el que trabaja Django), esta se trata de una herramienta que permite un desarrollo rápido y escalable.

Por otra parte se trabajará con Django Channels [2] para gestionar el uso de websockets. Django Channels permite el uso de websockets y tecnologías análogas mediante varios paquetes que se integran dentro del framework de Django. A través del uso de "consumers" o consumidores, abstracciones propias de Channels, se desarrollará la mayor parte de la lógica de la aplicación.

Posteriormente se hará mención a Pytest [3] a la hora de implementar el testing automático.

3.1.2. Vue3

Por otro lado, para el desarrollo frontend se propone el uso de Vue [4] debido a su popularidad, versatilidad y a su uso personal en otros proyectos. De esta forma se creará un frontend sencillo pero bastante personalizable.

Se hace referencia a Vue3 por ser la versión más moderna de Vue, la cual tiene cambios destacables en comparación a Vue2 [5]. Todos los paquetes y librerías adicionales se incorporarán teniendo en cuenta que se está haciendo uso de Vue3. En especial, se destaca la librería de componentes Vuetify [6], la cual es un gran añadido a la hora de ofrecer una gran diversidad de componentes para la construcción de una interfaz vistosa y dinámica.

3.1.3. Herramientas de diseño

Figma [7] es un editor de gráficos que se usa principalmente para generar prototipos. Se usará principalmente para el diseño y experimentación inicial de la aplicación web. Además de Figma, se harán uso de otras herramientas menores de diseño como draw.io [8] para la creación de diagramas, Pictogrammers [9] para hacer uso de iconos de forma sencilla o Contrast Finder [10] para comparar contrastes de color.

3.1.4. Docker

Docker [11] es una herramienta de código abierto que facilita un despliegue sencillo y portable utilizando contenedores. Un contenedor es una unidad estándar de software que agrupa el código y todas sus dependencias para que la aplicación se ejecute de manera rápida y fiable en diferentes entornos de computación [12]. Con el uso de Docker, se preparará la aplicación en diferentes contenedores para su despliegue de forma sencilla.

3.1.5. AWS

Amazon Web Services (AWS) [13] es una plataforma de servicios en la nube ofrecida por Amazon. Proporciona una amplísima gama de servicios que incluyen computación, almacenamiento, bases de datos, etc. Estas soluciones permitirán alojar la aplicación final de forma flexible.

Se detallará el uso tanto de Docker como de AWS en el proceso de preparación y despliegue de la aplicación.

3.1.6. GitHub

GitHub [14] es una plataforma de desarrollo colaborativo basada en la web que utiliza Git, un sistema de control de versiones distribuido. Permite a los desarrolladores alojar, gestionar y compartir proyectos de software, facilitando la accesibilidad del código.

Todo el código del proyecto será accesible mediante un repositorio público en GitHub [15] .

3.1.7. WSL

Se destaca de forma adicional el uso del subsistema de Windows para Linux (WSL) [16] para poder ejecutar de forma sencilla todo el entorno de la aplicación así como para la realización de pruebas de despliegue sin dejar de hacer uso directo de Windows como sistema operativo.

3.1.8. Visual Code

En último lugar se encuentra la herramienta más básica y a la vez más importante para el proyecto, Visual Studio Code [17]. Visual Studio Code, comúnmente conocido como Visual Code o VS Code, es un editor de código fuente desarrollado por Microsoft.

Todo el código se ha tratado mediante Visual Code, desde la creación y edición del código hasta la ejecución de comandos en la terminal del sistema.

3.2. Metodología

Este proyecto se ha guiado por el uso de una metodología de trabajo iterativa-incremental enfocada en la revisión de objetivos y avances entre el tutor y el alumno. Durante cada encuentro, se han ido discutiendo propuestas de mejora que se han convertido en objetivos para la siguiente evaluación. En primera instancia, después de plantear los objetivos principales del desarrollo y la motivación general del proyecto, se han propuesto tres diferentes fases de progreso con fechas estimadas:

- Fase 1: Septiembre 2023 - Diciembre 2023
- Fase 2: Enero 2024 - Febrero 2024

- Fase 3: Marzo 2024 - Mayo 2024

En la primera fase se destaca como objetivo principal crear una primera versión sencilla de la aplicación con las funcionalidades de juego básicas. Se propone iterar el desarrollo varias veces cumpliendo pequeños objetivos para lograr esta versión preliminar de forma satisfactoria.

En la segunda fase se propone desarrollar una versión avanzada de la aplicación con elementos adicionales. De forma similar, se propone trabajar mediante pequeñas iteraciones donde se pretende implementar reglas y funcionalidades adicionales al juego además de testing.

Habiendo completado estos puntos, a excepción del testing, se plantea una tercera fase enfocada en el despliegue de la aplicación y en la revisión del código presente (además de la documentación necesaria para el trabajo de fin de grado). Además, se plantea nuevamente implementar finalmente el testing deseado.

Todas estas fases de desarrollo han terminado con reuniones enfocadas en discutir el avance del proyecto y los objetivos completados de la fase en cuestión. De forma adicional, se ha tenido en cuenta este desarrollo por fases incluso para las subidas de código y otros elementos al repositorio del proyecto. En la Figura 3.1 se pueden observar diferentes aportaciones por parte del alumno al repositorio con su número de versión asociado y una pequeña descripción.

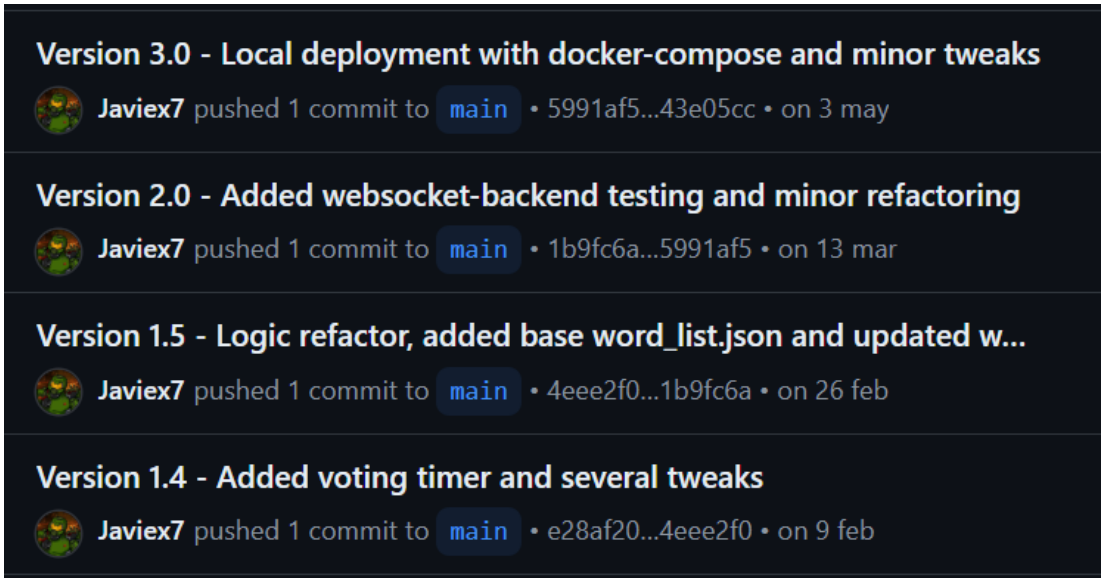

Figura 3.1: Aportaciones al repositorio de GitHub

Esta metodología ha demostrado ser efectiva para este tipo de proyecto, ya que permite un avance progresivo y detallado, evitando la acumulación de muchas tareas. Además, al planificar con visión a futuro, la aplicación ha crecido de manera sólida y constante, sin necesidad de realizar cambios radicales.

4

Descripción informática

En esta sección se expondrá la mayor cantidad de información sobre el desarrollo del proyecto, desde la definición de requisitos y análisis inicial, hasta el proceso final de testing y despliegue.

4.1. Requisitos

Definidos los objetivos del proyecto y tras realizar una primera reunión con el tutor del TFG, se pueden definir de forma más concisa tanto los requisitos funcionales como los no funcionales que debe cumplir la aplicación.

4.1.1. Requisitos funcionales

- RF1 Página de inicio - Debe existir una página principal que permita establecer el nombre dentro del juego al usuario, que explique las reglas de juego y que establezca una interfaz sencilla para poder crear o unirse a una sala.
- RF2 Creación de salas - Un usuario debe ser capaz de crear una sala y también poder unirse a salas ya creadas mediante un código único.
- RF3 Sala de juego y chat - Dentro de una sala, cada jugador debe poder visualizar a todos los jugadores con los que va a jugar así como interactuar con ellos de forma sencilla mediante un chat.

- RF4 Empezar partida - Dado un número mínimo-máximo de jugadores uno de los jugadores debería tener la capacidad para empezar una nueva partida.

- RF5 Conexiones - Tiene que existir una lógica que asuma conexiones o desconexiones accidentales y deliberadas por parte de alguno o varios jugadores antes de empezar la partida en la sala, durante el juego o en la visualización de resultados.

- RF6 Inicio de partida - Al empezar la partida, el sistema debería proporcionar un orden aleatorio de turnos secuencial además de la asignación de un rol junto a la palabra clave a los jugadores Inocentes.

- RF7 Turnos - El orden secuencial a la hora de enviar palabras debería ser visible y destacar cuando es el turno de un jugador en concreto.

- RF8 Enviar palabras - De forma secuencial, todos los jugadores deberían poder enviar una palabra semejante a la pista asumiendo el juego de roles Outsider/Inocente sin interferencias por parte de otros jugadores.

- RF9 Votaciones - Al finalizar el envío de palabras se asume una fase de votaciones en la que, de forma asíncrona, cada jugador vota a otro con el fin de acusarlo de Outsider y eliminarlo del juego.

- RF10 Resultados - Habiendo realizado la votación todos los usuarios, el sistema calcularía el resultado de esta votación para definir el desenlace final de la ronda y se les mostraría a cada uno de los jugadores de forma simultánea este resultado.

- RF11 Gameloop - El sistema debe identificar cuando todos los jugadores han enviado sus palabras, mostrar la interfaz de votación y finalmente gestionar y revelar los resultados a todos los jugadores de forma consistente.

- RF12 Múltiples rondas - Si el número de jugadores y el resultado de la ronda lo permite, el sistema debe asumir que tras la finalización de una ronda pueden jugarse otras rondas de forma adicional si se desea. Por ejemplo, si hay cuatro jugadores y los tres jugadores Inocentes no han identificado y/o votado al Outsider de forma correcta, se podría jugar otra ronda con tres jugadores si se ha votado de forma errónea a un jugador Inocente o con los mismos cuatro si ha habido un empate en la votación.

- RF13 Varios Outsider - Si el número de jugadores es bastante alto, el sistema asignará el rol de Outsider a otro jugador de forma adicional.

- RF14 Última oportunidad - En el caso en el que el jugador con más votos sea el último Outsider en la partida, este tendrá un intento de adivinar la palabra clave que conocen los jugadores Inocentes.

4.1.2. Requisitos no funcionales

- RNF1 Usabilidad y accesibilidad básicas - La aplicación web debe atender a valores de usabilidad universales, como el uso del zoom o el de diferentes resoluciones así como accesibilidad básica como el uso de colores con contraste por indicar un ejemplo práctico.
- RNF2 Calidad de las conexiones - Se debe tener en cuenta la calidad y estabilidad de las conexiones mediante websocket para evitar interrumpir, en la medida de lo posible, a los jugadores.
- RNF3 Multiplataforma - La aplicación debe ser accesible desde diferentes tipos de dispositivos como teléfonos móviles u ordenadores portátiles y de sobremesa.
- RNF4 Información de estado - Se debe mantener informado al usuario del estado general de la partida así como de las posibles conexiones y desconexiones por parte de otros jugadores.
- RNF5 Testing - El sistema tendrá una opción de desarrollo para poder ejecutar diversos tests que ponen a prueba el correcto transcurso de una partida así como las conexiones y desconexiones en el entorno websocket del backend.
- RNF6 Despliegue y administración - La aplicación estará lista para el despliegue y será accesible a la hora de ofrecer información de administración del servidor backend.

4.2. Arquitectura general

En este apartado, se pretende definir de forma más concisa la arquitectura software de la aplicación teniendo en cuenta las tecnologías usadas y su interoperatividad además del por qué se han tomado estas decisiones de diseño.

Después de haber analizado los diferentes requisitos y considerando las herramientas y conocimientos de partida, se definen los servicios mínimos que debería ofrecer el sistema en conjunto:

1. Backend - Django. Es fundamental el uso de un servidor backend capaz de gestionar la lógica de peticiones básica y que tenga capacidad para hacer uso de tecnología websocket. Para esta tarea, se propone una implementación sencilla a través de Django. Se destaca que hay muy poca información que se quiera tratar de forma persistente, por ello, no será necesario un servicio de base de datos adicional y se recurrirá a la configuración por defecto de Django con un archivo SQLite.

2. Servidor websocket - Redis (Django Channels). Al hacer uso de Django y de su particular implementación de websockets, se considera indispensable el uso de un servicio adicional, un servidor Redis. Este servidor es necesario para poder hacer uso del sistema de "capas", que permiten que varias instancias de controladores websocket (denominados "consumers") puedan comunicarse entre sí y con otras partes de Django [18].

3. Frontend - Vue3. Finalmente, solo quedaría gestionar la vista de la aplicación a través de Vue que, para el desarrollador, facilita mucho el trabajo de frontend, integrando una gran cantidad de herramientas y componentes. Dentro de este servicio se tendrá que trabajar la comunicación con el backend así como la visualización y lógica web, por ejemplo, la lógica de navegación entre páginas a través de vue-router.

Estos serían los elementos básicos del sistema, pero hay que tener en cuenta su despliegue final. Para entenderlo todo de forma más organizada en la Figura 4.1 se muestra un esquema de la arquitectura a nivel general. En el Apéndice A.1 se muestra la misma imagen de mayor tamaño.

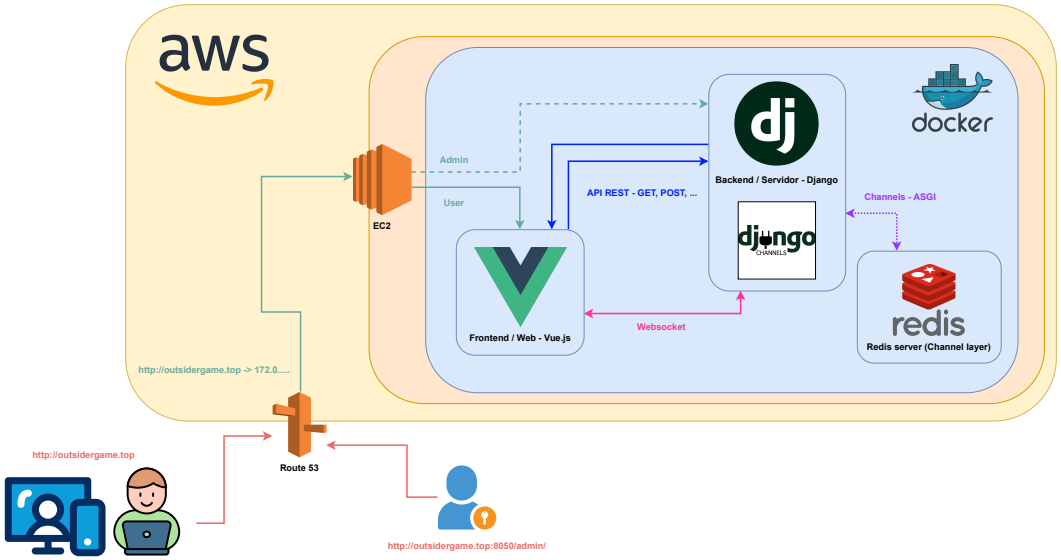

Figura 4.1: Arquitectura final del sistema

De este esquema inicial cabría destacar un poco más en detalle el funcionamiento de las conexiones websocket, ya que, Django Channels está construido sobre la especificación ASGI (Asynchronous Server Gateway Interface), diseñada para proporcionar una interfaz estándar entre servidores web Python, frameworks y aplicaciones que soportan asincronía [19]. Dicho de otra forma, permite hacer uso tanto de conexiones y peticiones HTTP como del protocolo websocket sin problemas, algo totalmente necesario para el desarrollo del proyecto.

La comunicación a bajo nivel se hace un poco más compleja, ya que, lo que facilita Django Channels son las herramientas de alto nivel para trabajar directamente con los websockets. En la Figura 4.2 se puede observar con mayor detalle como funcionan las conexiones a bajo nivel en el entorno ASGI de Channels: desde el cliente web (Browser), se realizan peticiones tanto HTTP como websocket donde las flechas indican que, mientras los mensajes HTTP tienen un envío y respuesta, las conexiones websocket crean un canal bidireccional, con capacidad para enviar y recibir mensajes de forma simultánea. Cuando estos mensajes llegan al servidor gestionado a través de Django Channels, toda la comunicación se trabaja mediante ASGI y se dirige el tráfico de la forma correspondiente; mientras que las peticiones HTTP acaban siendo tratadas en las vistas con las que trabaja Django, los mensajes websocket son despachados por los consumers, la implementación de alto nivel que ofrece Channels para realizar la comunicación websocket [20] [21].

De esta forma se puede entender un poco más sobre como funcionan las conexiones a bajo nivel respecto al anterior esquema general, el cual solo hace referencia a las conexiones a mayores entre los servicios principales de la aplicación.

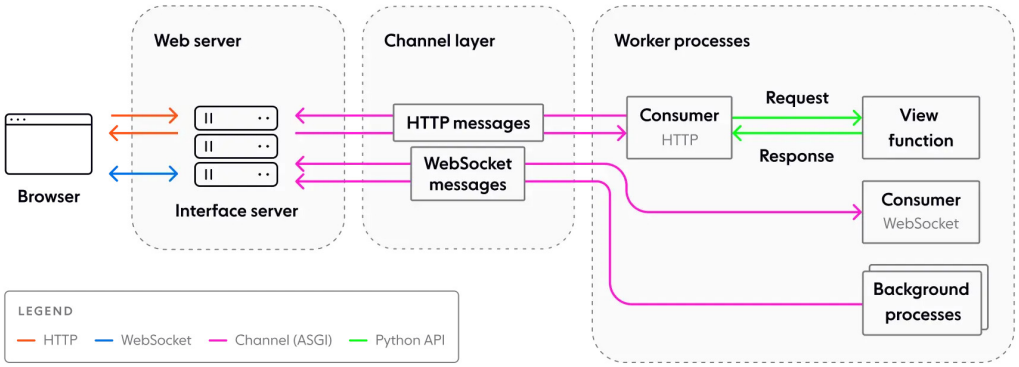

Figura 4.2: Arquitectura de Django Channels

Haciendo referencia solo a los aspectos de alto nivel y sin tener en cuenta los elementos asociados al despliegue (que se trataran con más detalle en las siguientes secciones), el resto del esquema es más sencillo: el usuario utilizará una URL, outsidergame.top para acceder a la aplicación web y hacer uso de esta. La aplicación web (Vue) hará las necesarias comunicaciones (tanto HTTP como websocket) para acceder al contenido y la lógica del juego contenidas en el servidor de Django/Channels. De forma adicional se dispone una URL de administrador, outsidergame.top:8050/admin/ protegida con credenciales para que el supuesto administrador de la web pueda gestionar algún dato de forma directa (principalmente la lista de palabras disponible).

4.3. Diseño visual

En una primera fase de conceptualization, se ha hecho uso de Figma [7] para calcular los tamaños y estructura general que deberían tener algunas de las páginas principales de la web. Se ha tenido en cuenta la disposición de los elementos cuando se está usando un dispositivo móvil o un ordenador portátil (se entiende que el diseño funcione igual de bien en tablets y otros dispositivos similares), ya que, estos tipos de dispositivo tienen diferentes proporciones y hay pantallas que comprometen mucho la visualización en general.

En la Figura 4.3 se pueden observar a grandes rasgos los prototipos para la página principal y para la página de espera de juego. Se destaca la existencia de algunos comentarios además de colores y estilos provisionales.

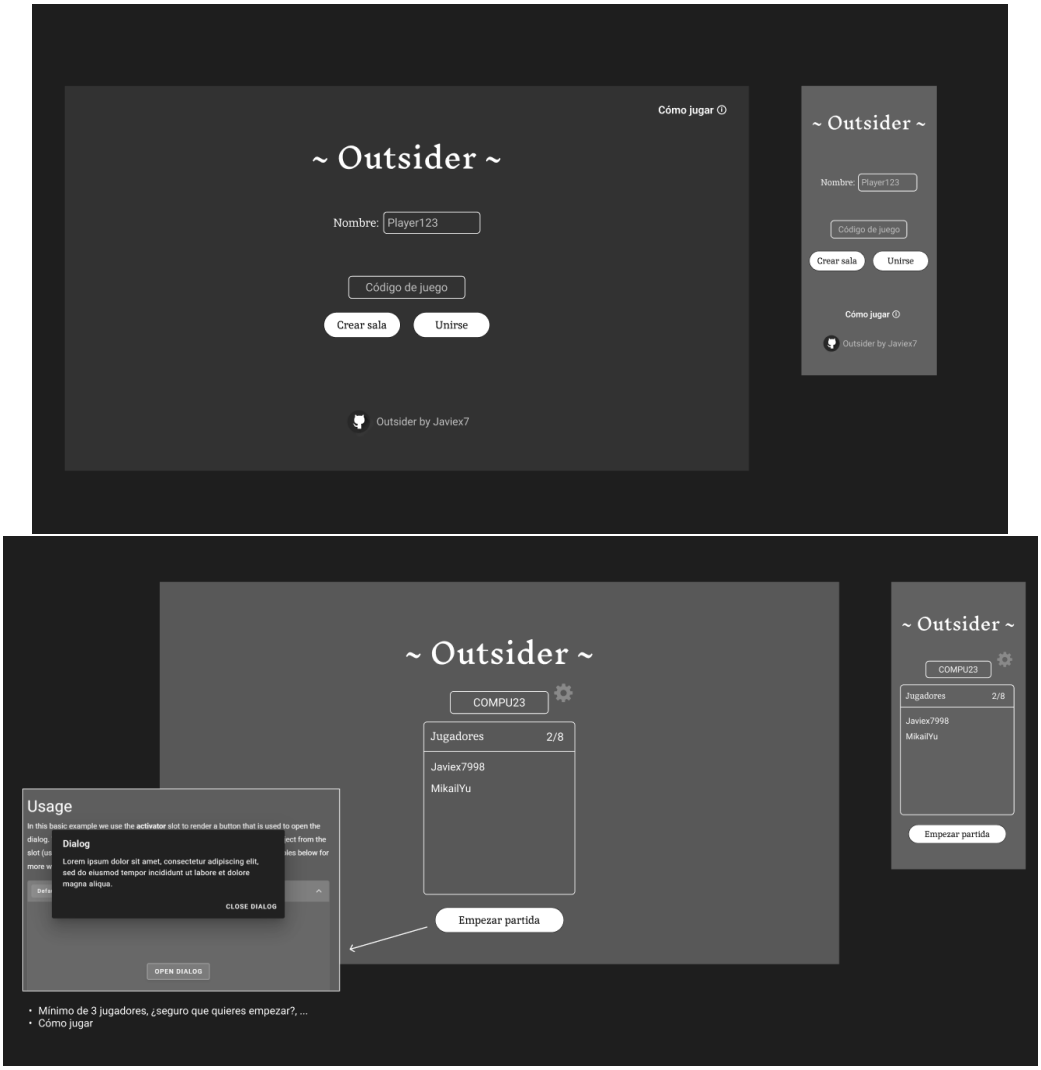

Figura 4.3: Prototipos iniciales

Estos prototipos permiten concretar mejor el estilo y el diseño general de la página web. Además, se llevan a cabo varias pruebas de colores, tipografías y otros elementos estéticos antes de implementar estos estilos en la aplicación. En la Figura 4.5 se puede observar el primer diseño para la pantalla de juego.

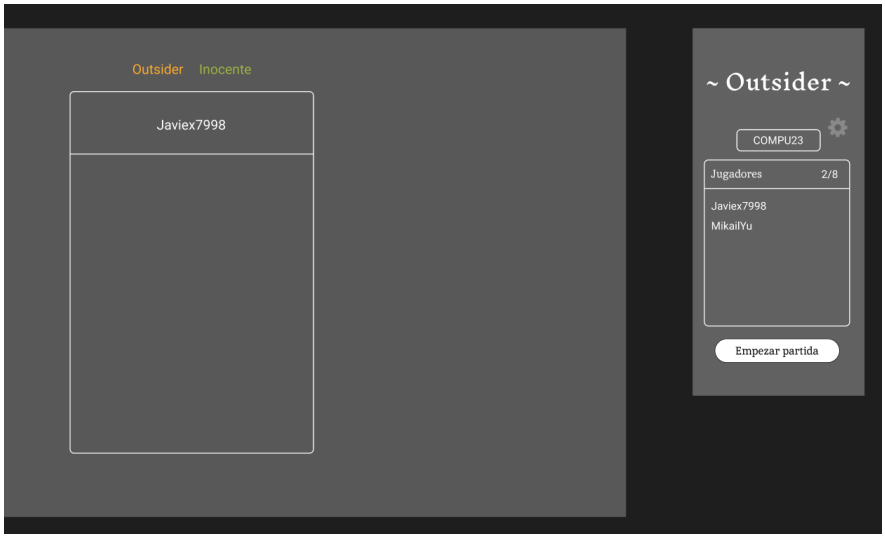

Figura 4.4: Prototipo de la página de juego

Cabría destacar que solo se ha tenido en cuenta la creación de unos pocos diseños debido a la facilidad de trabajar directamente sobre los componentes de Vuetify y la necesidad de cambios constantes en la interfaz. De esta forma se ha evitado un sobre esfuerzo derivado de la iteración entre diseño e implementación.

4.4. Implementación

En esta sección se detallarán los aspectos más importantes para la implementación del sistema propuesto. Se abordarán tanto las funcionalidades y usos generales como los pequeños avances que se consideran significativos durante el proceso de desarrollo.

4.4.1. Implementación básica Backend/Frontend

La primera parte del desarrollo se ha basado en la implementación básica tanto del frontend a través de Vue como del backend con el uso de Django. Para esta tarea, lo más importante es la correcta instalación de dependencias así como de la preparación del entorno de desarrollo.

Debido a la experiencia previa en este apartado, no ha sido necesariamente complicada esta instalación inicial. Para Django, la instalación es muy sencilla,

solo se necesita tener Python en el sistema y ejecutar un conjunto mínimo de mandatos, por otra parte, la instalación de Vue puede resultar algo más complicada, y en este caso se ha optado por hacer uso directamente de Vuetify para tener todas las dependencias derivadas de Node.js integradas en el proyecto. De esta forma, se logra una estructura como la que se puede observar en la Figura 4.5. En el directorio "outsider" se encuentran los archivos de configuración base de Django, mientras que en la carpeta "outsider-front" se encuentra todo el código fuente relacionado con el uso de Vue/Vuetify.

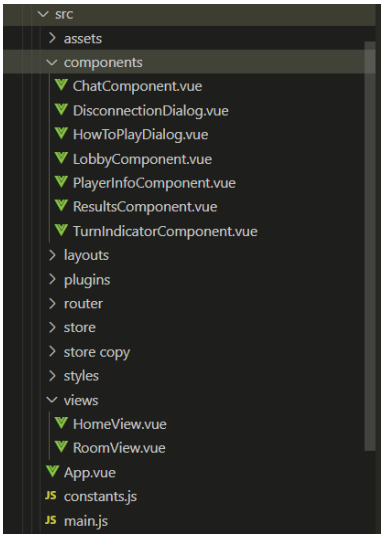

Figura 4.5: Estructura del proyecto

Se destaca que Django trabaja mediante el uso de "aplicaciones". Haciendo referencia directa a la documentación de Django: el término "aplicación" o "app" describe un paquete de Python que proporciona un conjunto de funciones en concreto. Estas aplicaciones suelen incluir una combinación de modelos, vistas, plantillas, archivos estáticos, URLs y otras configuraciones relevantes a la aplicación en concreto [1]. Debido a que la lógica estará enfocada principalmente a la gestión de las conexiones websocket, todos los modelos de datos y en general toda la lógica del backend de Django estará contenida dentro del directorio "logic".

Como se había destacado antes, dentro de la carpeta "outsider-front", se encontrará de forma bastante desglosada todos los componentes que conforman el frontend de Vue así como todas las configuraciones y ficheros auxiliares pertinentes. En la sección 4.4.4 se tratará en detalle la estructura del framework de Vue/Vuetify y la construcción de la página web.

De manera adicional, cabe destacar que, debido a las facilidades de mandatos que ofrecen los sistemas unix así como que el despliegue final será en una maquina Linux, se está trabajando a través de WSL, es decir, ambas aplicaciones (frontend y backend), así como sus dependencias (Python y Node.js principalmente) se ejecutan en Ubuntu a través de la interfaz de WSL, ya que, el ordenador en

el cual se está desarrollando el proyecto solo tiene instalado Windows 11 y por decisión propia se prefiere su uso de forma cotidiana. Es importante tener esto en cuenta en primera instancia para evitar problemas en el futuro despliegue o mientras se está trabajando en el proyecto, debido a que, el uso de mandatos y la disponibilidad de ciertos instaladores y librerías cambia drásticamente entre Windows y Linux.

Teniendo en cuenta esta instalación y configuración básica tanto del backend como del frontend, ya se puede trabajar directamente en el desarrollo de la aplicación per se.

4.4.2. Aprendizaje y uso de tecnologías websockets

Después de haber configurado el entorno de desarrollo base, cabría añadir los elementos necesarios para poder trabajar con tecnologías asíncronas.

Para el frontend solo será necesario hacer uso de la implementación del protocolo websocket [20], mientras que en el backend, como ya se había comentado, será necesario hacer uso de otros elementos, en este caso, Django Channels. Por ello, lo primero será realizar la instalación de Channels en el server de Django.

Además de la instalación sencilla mediante "pip install", es necesaria la configuración de varios "settings" del proyecto, así como implementar el uso del servidor ASGI Daphne. En la documentación [22] se indica paso a paso el proceso de instalación en detalle si se desea profundizar en el tema.

Con Django Channels instalado en el sistema, lo primero que se ha realizado son los propios tutoriales de uso, los cuales detallan la creación de un chat asíncrono, desde la configuración básica, hasta el uso de la funcionalidad de layers de forma asíncrona.

Estos tutoriales, que también se encuentran en la propia documentación de Channels [23], son de bastante calidad y se podrían comparar con la realización de una práctica guiada en una asignatura de la Escuela Técnica Superior de Ingeniería Informática. Se enseñan las capacidades básicas de los websocket y se va expandiendo a nivel funcional un ejemplo de chat bastante completo. Se han encontrado tan interesantes y útiles estos ejemplos, que el chat implementado en la aplicación final, se ha mantenido con la configuración definida en estos tutoriales.

En términos generales, el funcionamiento de Channels, vendría a ser la siguiente: En primer lugar, después de tener el proyecto configurado y una aplicación de Django lista para trabajar con websockets (para este proyecto será la aplicación de "logic"), será necesario la creación de un consumidor websocket.

Cuando Django acepta una solicitud HTTP, consulta la configuración de url

raíz para buscar una función de "vista/view" y luego llama a la función de vista para manejar la solicitud. De manera similar, cuando Channels acepta una conexión WebSocket, consulta la configuración de url para buscar un consumidor y luego llama a varias funciones dentro del propio consumidor para manejar eventos de la conexión.

Siendo más específicos, los consumidores se estructuran en torno a una serie de métodos/funciones con nombre correspondientes al tipo de mensaje que recibe del propio websocket. En otras palabras, se tendrá un consumidor con un método "connect", un método "disconnect" y un método "receive" que gestionarán o consumirán, valga la redundancia, los mensajes de tipo websocket de nueva conexión "new WebSocket", los de desconexión "WebSocket.close()" y los de envío de mensajes estándar "WebSocket.send()" respectivamente.

De esta forma, se entiende que los mensajes generados en el frontend web a través de la conexión websocket se "producen" con el objetivo de que el backend "consuma" estos mensajes mediante los propios "consumidores" que se han instanciado a la hora de crear la comunicación websocket. Por último, el uso de Django Channels se basa en el uso de estos objetos consumidores de la forma que se vea más adecuada. Por ejemplo, cabe destacar que los consumidores pueden heredar de diferentes clases con diferentes características, por ejemplo pueden existir consumidores síncronos, asíncronos, realizar decodificación JSON de forma automática, etc.

Teniendo clara como funciona la lógica de consumidores, faltaría destacar la implementación propia que se ha diseñado para este proyecto: Debido a que la lógica gira en torno al control del flujo del juego, solo se hace uso de un controlador para gestionar la única conexión websocket que debería haber entre un jugador y una sala de juego. Por esto último también se denomina al consumidor como "RoomConsumer". El Código 4.1 ejemplifica los métodos principales e inicialización del consumer en cuestión. Como se puede observar se ha hecho uso de un AsyncJsonWebsocketConsumer, con decodificación JSON y el uso de comunicación asíncrona para poder gestionar varias peticiones de forma paralela sin necesidad de esperar a que se completen las peticiones en orden.

De esta forma el servidor es capaz de trabajar con información que le llega desde el frontend, sin embargo, es necesario la capacidad opuesta, el envío de información al frontend desde el propio servidor, es más, debido a la lógica del juego, se debería poder enviar información a varios clientes web. Por ejemplo, a la hora de empezar una partida, a nivel lógico en el servidor, se quiere indicar a todos los jugadores que están en la sala que ha empezado la partida así como su rol y posible palabra clave.

Para empezar, se debe hacer uso del sistema de layers ya que, es indispensable para diferenciar diferentes salas de juego y permitir la comunicación entre diferentes instancias de consumidores. En este caso, queremos que jugadores de

Código 4.1: RoomConsumer

```
import json
from channels.generic.websocket import
    AsyncJsonWebsocketConsumer

from .utils.consumer_classes import State, WebsocketUser
from .utils import sync_rest_calls, consumer_methods

class RoomConsumer(AsyncJsonWebsocketConsumer):
def __init__(self, *args, **kwargs):

async def connect(self): ...
async def disconnect(self, close_code): ...
async def receive_json(self, content): ...
```

una misma sala se comuniquen entre sí de forma constante. Por ello, además de gestionar el servidor Redis mediante un contenedor Docker, también se tiene que procesar toda la lógica de broadcast o envío de mensajes desde el backend al frontend. Esto se realizará mediante el propio consumidor, el cual tiene la capacidad de suscribirse a un "grupo" haciendo uso de la aplicación de layers. Esta última operación se debe realizar cuando se acepta una conexión websocket en el propio consumidor.

En el Código 4.3 se puede observar el uso del método "channel_layer.group_add()" para añadir al consumidor al grupo (siempre estará relacionado el grupo a nivel lógico con la sala de juego en la que esté el propio jugador) cuando se realiza una nueva conexión websocket (método connect del consumidor).

Código 4.2: Añadir consumer a un grupo

```
async def connect(self):
    ...
    # Join room group
    await self.channel_layer.group_add(self.room_group_name
        , self.channel_name)
    await self.accept()
```

Por otro lado, dentro del método receive, puede ser que un usuario en concreto decida enviar el mensaje para empezar la partida y su consumidor, además de realizar la lógica de juego, debe hacer uso del método "channel_layer.group_send()" para que los diferentes consumidores que forman parte del mismo grupo reciban la información de que un jugador está intentando empezar un partida.

Este mensaje se tramitará en base al "tipo" de mensaje que defina el propio

método de broadcast, es decir debe existir un método denominado como "startGame" el cual, además de poder realizar lógica adicional para cada jugador (por ejemplo, solo enviar la palabra clave de juego a jugadores Inocentes) enviaría finalmente el mensaje de empezar partida desde el consumidor al propio websocket (el propio usuario en el cliente web) mediante el método "send_json()".

Se puede apreciar de forma resumida, tanto el código de broadcast como el envío del mensaje final al cliente en el Código 4.3. Para otras acciones existirán otros métodos correspondientes y en este código solo se resalta el uso de "startGame" para ejemplificar el uso de grupos de consumers.

Para finalizar este apartado y resumiendo brevemente, el objeto consumidor, además de gestionar los mensajes que le llegan desde el cliente web y procesar la lógica necesaria, es capaz de enviar nueva información de vuelta a usuarios que compartan un mismo grupo, que en el contexto de este proyecto serán las salas de juego. De esta forma se logra una comunicación bidireccional usuario-servidor con la capacidad de poder relacionar de forma consistente un grupo de conexiones en concreto, que serán en este caso las diferentes salas de juego.

Código 4.3: Broadcast a todos los consumidores y envío final al cliente

```
1
2       # RECEIVE: Websocket -> Consumer -> Consumer group
3       async def receive_json(self, content):
4           ...
5           if action == "startGame":
6           self = start_game
7           message = {...}
8           ...
9           # GROUP: Consumer -> Consumer group
10          await self.channel_layer.group_send(
11          self.room_group_name,
12          {"type": action, "message": message, "username":
                username},
13          )
14
15      # startGame: Consumer group -> Consumer -> WebSocket
16      async def startGame(self, event):
17          ...
18          # SEND: Consumer -> WebSocket
19          await self.send_json(content={...})
```

4.4.3. Estructura de código en el backend

Habiendo explicado el funcionamiento general de Django Channels, se procede a definir en detalle los métodos y funcionalidades de la implementación de la lógica del juego, incluyendo elementos adicionales, como el uso de peticiones REST y teniendo en cuenta la legibilidad y estructura del código.

Todo el código relevante que se va a tratar, pertenece a la aplicación de "logic" dentro de la carpeta fuente del proyecto. Dentro de esta carpeta destacamos los ficheros "consumers.py" y "viewsets.py".

consumers.py

Teniendo en cuenta todo lo que se ha explicado en el apartado previo, dentro de "consumers.py" se encuentra la lógica asociada al consumidor websocket de la aplicación, RoomConsumer. Se destaca que cada consumidor tiene una inicialización de datos que va a ir trabajando a lo largo de la sesión. Estos datos deberían estar sincronizados entre las diferentes instancias de consumer que pertenezcan al mismo grupo/sala, por ejemplo, la lista de jugadores Outsider o un flag booleano que indica el fin del juego. Además de estas variables el consumidor redefine con mucha lógica los métodos websocket básicos: connect, disconnect y receive (el método receive se denota "receive_json" por heredar del tipo de consumidor AsyncJsonWebsocketConsumer).

El método receive tendrá la mayor carga de lógica y es por eso que dentro de la carpeta utils (dentro de la aplicación de logic) se encontrarán otros métodos auxiliares para separar mejor el código, ya que, el método receive funciona como un gran "switch" en base al mensaje que reciba el consumidor desde websocket. Si un jugador envía el mensaje de siguiente turno (solo lo puede hacer el jugador cuyo turno este activo) el consumidor, además de gestionar la lógica pertinente para indicar que sería el turno del siguiente jugador (puede que se acabe la ronda por ejemplo), debe indicar al resto de consumidores el resultado que haya sido calculado.

Para evitar acciones erróneas o no existentes en la lógica de la aplicación, existe una selección de acciones posibles, que serían: "default", "connection", "startGame", "nextTurn", "votingOutsider", "lastChance", "nextRound" y "endGame" y estas hacen referencia a su traducción en español ("acción por defecto", "conexión", "empezar juego", "siguiente turno", "votar outsider", "última oportunidad", "siguiente ronda" y "final de juego"). Habiendo realizado la lógica de cada acción, se le comunica al resto de consumidores la información pertinente y es por ello que se definen también todos los métodos que reciben la información del grupo/sala con el mismo nombre de estas acciones. Por ejemplo, si la acción es "votingOutsider", ese consumidor enviará un mensaje a todos los consumidores

de su grupo de tipo “votingOutsider”, y cada uno de los consumidores ejecutará el método con el mismo nombre que el tipo de mensaje enviado por el consumidor original, en este caso, será el método “async def votingOutsider(self, event)”.

De forma extra, también se ha definido un enumerado que identifica el estado de un jugador y una clase “WebsocketUser” que hace referencia a la información del usuario que hace uso de la conexión websocket con el servidor a la hora de entrar en una sala de juego (dentro de “utils/consumer_classes.py”). De esta forma se facilita la lógica de la aplicación y se estructuran de forma más ordenada los datos.

viewsets.py

Se está destacando muchas veces el hecho de “entrar en una sala” por parte del usuario, pero esto no es una acción trivial, ya que, de alguna forma el cliente web debería poder conocer de forma sencilla el estado de una supuesta sala si es que un usuario quiere crearla o en todo caso entrar a una. Es por ello, que se destaca la implementación de una lógica basada en modelo de datos, es decir, el estado de las salas se define y actualiza en base de datos para poder gestionar de forma sencilla las futuras conexiones websocket.

Esta lógica se realiza mediante Django estándar: Primero se definen modelos de datos, en este caso “RoomModel”, para guardar la información de estado de una sala/partida y de forma adicional “WordsListModel” para definir una lista de palabras para el juego de forma persistente y fácilmente accesible. Definidos los modelos, simplemente se hace uso de ellos mediante las vistas implementadas en “viewsets.py”. Si un usuario quiere crear una sala, si no existe una sala con el mismo nombre, puede crearla sin problemas y ahora se puede gestionar sin problemas la lógica websocket asociada realizando una conexión inicial. De forma similar, sin un usuario quiere entrar en una sala, se comprueba si existe para poder realizar realizar la conexión pertinente.

Es importante destacar que se realizan peticiones a la base de datos desde el entorno asíncrono del consumidor websocket (desde “consumers.py” y “utils/-consumer_methods.py”), es por ello que dentro de la carpeta de utils también se define un fichero “sync_rest_calls” que contiene los métodos de acceso a base de datos con un manejo específico (a través de la definición de un contexto síncrono y seguro) para que todo funcione sin problemas. Al final, es una necesidad separar la lógica síncrona de la base de datos con la asíncrona de las comunicación con websockets.

Código adicional

Se destaca la configuración básica de urls y enrutamiento, tanto para websocket (a través de "routing.py") como para las vistas clásicas (en "urls.py") así como de configuración adicional a la hora de iniciar o reiniciar el servidor backend (en "apps.py"). Esta configuración inicial se encarga de limpiar salas si se han eliminado de forma incorrecta debido a un bug del servidor backend o problemas de conexión mayores y, por otra parte, actualizar la lista de palabras que utiliza el juego cuando no se ha encontrado una lista de palabras inicial válidas o se ha borrado de la base de datos en algún momento.

Administrador de Django

Si se accede a outsidergame.top:8050/admin/ es posible acceder a la vista de administrador de Django con las siguientes credenciales:

- Usuario: outsider_admin
- Contraseña: 2024outsider

En esta vista se podrá observar el estado de las partidas (útil para el desarrollo) y acceder a la lista de palabras clave que utiliza el juego. Como se puede comprobar en la Figura 4.6, se puede modificar de forma directa las palabras disponibles (denotadas con "a") y sus tentativas (indicadas con la letra "b") que se le proporcionan al Outsider si es el primero en empezar la ronda. Debido a que no hay más modelos de datos presentes, el resto de modificaciones al comportamiento de la aplicación dependen directamente de modificar el código y los archivos de configuración.

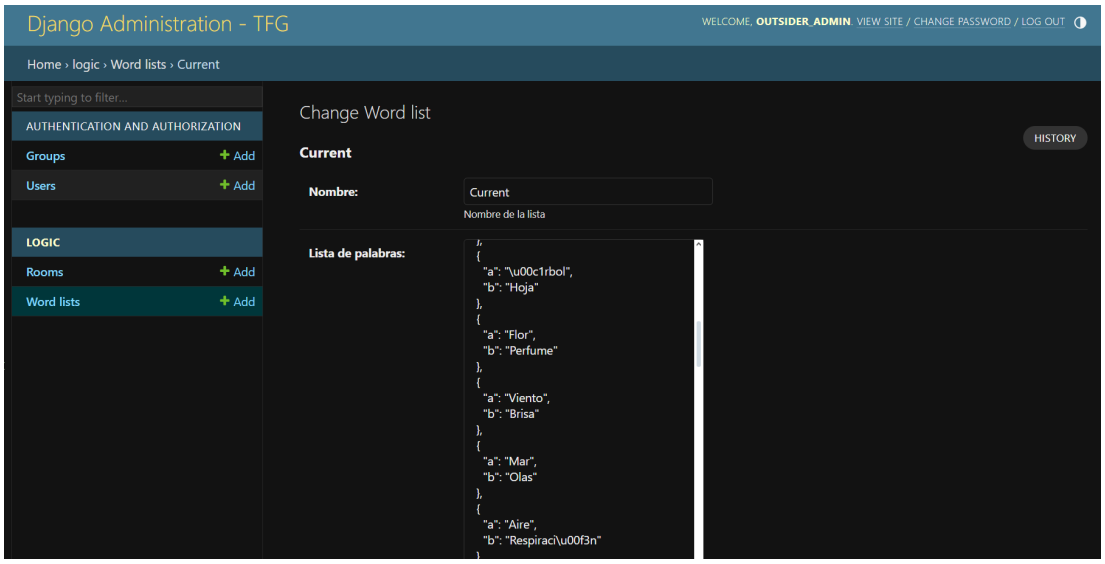

Figura 4.6: Vista de administrador de Django

4.4.4. Desarrollo del frontend

Una vez explicado el desarrollo relacionado con el servidor/backend, se puede comenzar a abordar el desarrollo del frontend utilizando Vue y las tecnologías web relacionadas. Para comenzar, todo el código a tratar se encuentra dentro de la carpeta "outsider-front", dentro de la carpeta principal del proyecto. Dentro de este directorio están los archivos específicos del proyecto de Vue (principalmente en la carpeta src), junto con las librerías de Node.js y otras configuraciones adicionales.

Vue solo hace una gran distinción entre elementos: vistas y componentes, que se distinguen por su uso específico. La diferencia es simplemente que las vistas son componentes con capacidad de enrutamiento: Es decir, mientras que los componentes pueden actuar como bloques reutilizables de comportamiento, estilo y estructura en diversas partes de la página web, las vistas tienen una ruta específica (url) dentro de la página web. En el desarrollo del proyecto, como se pude observar en la siguiente Figura 4.7, se han planteado dos vistas que definen la página de inicio y la página que gestiona la sala de juego. Por otra parte existen diversos componentes reutilizables que se integran dentro de cada una de la vistas.

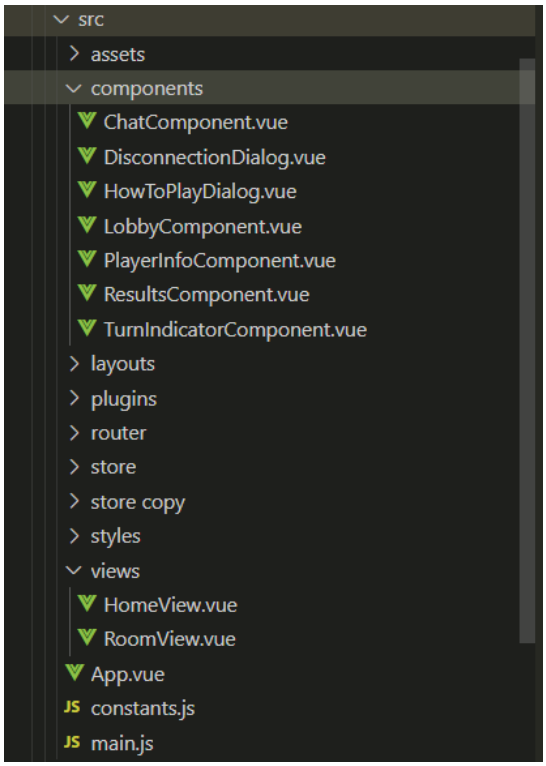

Figura 4.7: Estructura de ficheros en el frontend

Antes de continuar con la descripción de las vistas, cabría destacar nuevamente el uso de Vuetify, ya que, se ha utilizado en gran medida para facilitar el

trabajo de diseño y maquetación web. Esto se ha llevado a cabo principalmente mediante el uso de componentes de Vuetify. Estos componentes de diferentes tipos, facilitan la construcción de interfaz facilitando elementos como botones, carruseles de elementos o menús, con la facilidad de cambiar su diseño de forma sencilla sin hacer uso extensivo de CSS o en todo caso de código adicional para definir animaciones y comportamientos. De esta forma, se consigue una interfaz visualmente atractiva y de sencilla creación.

Dicho esto, en la Figura 4.8 se puede observar la vista principal, "HomeView.vue" en un dispositivo de escritorio. Es una vista muy sencilla y el código asociado no es nada complejo, ya que solo tiene en cuenta la actuación de los botones para crear una sala y unirse a una sala mediante una petición sencilla al servidor con el uso de Axios [24]. Se tiene en cuenta que el nombre de usuario no sea ni demasiado corto, ni demasiado largo así como que el código de sala esté presente a la hora de intentar acceder a una sala de juego (o intentar crearla). También se proporcionan mensajes de "error" si ha habido alguna incidencia (normalmente el intentar unirse a una sala que no existe o al intentar crear una sala ya existente). Para finalizar, la página también muestra con mucho detalle un diálogo con todas las instrucciones del juego si se pulsa el botón de "Cómo jugar".

Figura 4.8: Página principal

La otra vista, "RoomView.vue", bastante más compleja, maneja toda la interfaz gráfica así como la lógica de peticiones websocket asociadas al juego per se, desde la visualización y funcionamiento del chat, hasta la correcta presentación de los resultados de una ronda.

Para empezar, destacamos el "LobbyComponent.vue", encargado de mostrar y gestionar, dentro de la vista de la sala todos los elementos relacionados con la sala de espera, entre los que se incluye un chat entre los miembros actuales de

la sala, diferentes indicadores del estado de la sala (código de sala para que se puedan unir otros usuarios, número de jugadores y lista con indicadores sobre capitanía de la sala) y un botón que sirve para iniciar la partida por parte del capitán de la sala si hay más de dos usuarios presentes.

En la Figura 4.9 se puede ver un lobby/sala de dos jugadores interactuando con el chat así como otra sala con seis jugadores listos para jugar. De esta sección solo cabría resaltar que se indica cuando se jugará con un outsider adicional (con seis o más jugadores) y también que, al lado de cada nombre de usuario se indica quién es el capitán mediante un logo de corona y resaltado en azul. Además, también se muestra al propio usuario que está haciendo uso de la aplicación mediante un icono de cuenta/usuario al lado del nombre introducido antes de ingresar en la sala.

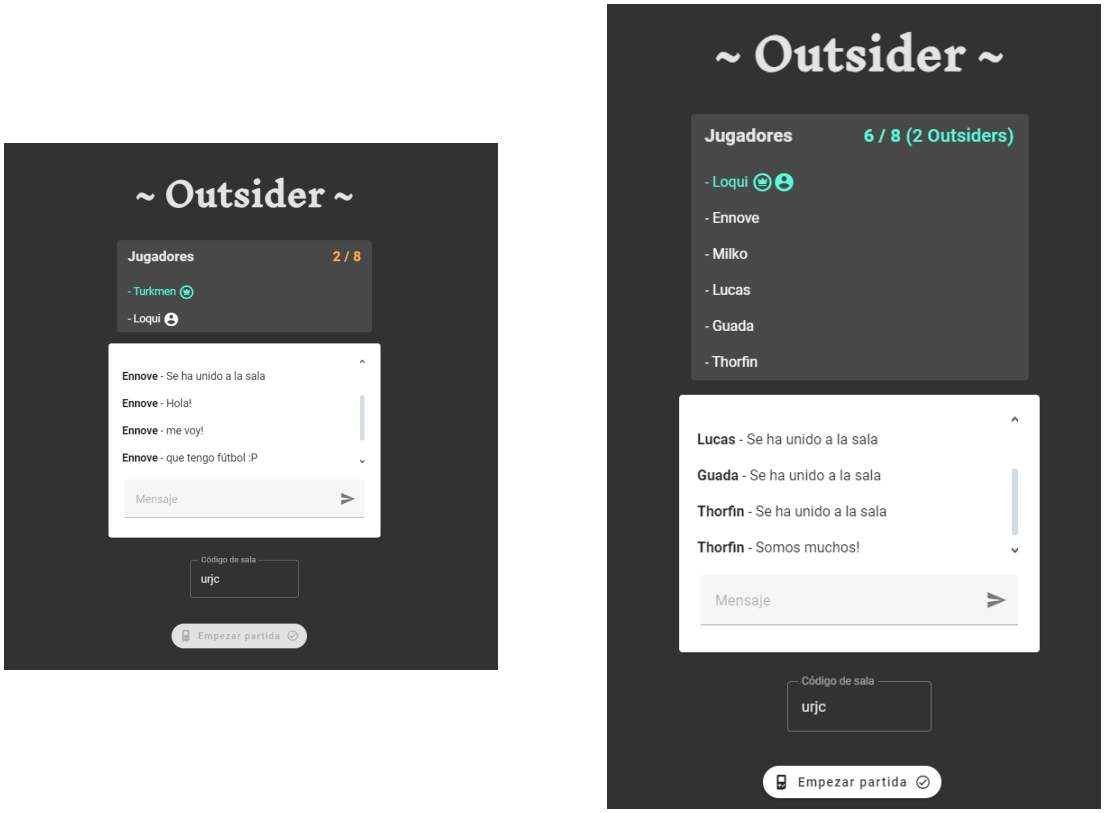

Figura 4.9: Sala de espera con diferentes números de usuarios

Después de empezar una partida y gestionar la lógica de las conexiones websocket, la aplicación debería mostrar una sección de juego donde se indique de forma clara el orden de turnos de los jugadores además de brindar controles para poder escribir palabras. Como se puede ver en la Figura 4.10, los jugadores disponen de diversa información como la palabra clave, su nombre y rol junto a la lista de jugadores en la partida y, en dispositivos lo suficientemente grandes, un chat a mano derecha para poder interactuar en plena partida. También existe

un texto que recuerda cuando es el turno del usuario así como el funcionamiento básico de juego.

Debido a que se trata de la parte con más lógica de la aplicación se gestiona directamente desde la vista "RoomView.vue" mientras que se hacen uso de otros componentes, como "PlayerInfoComponent.vue" o "TurnIndicatorComponent.vue", para ordenar el código.

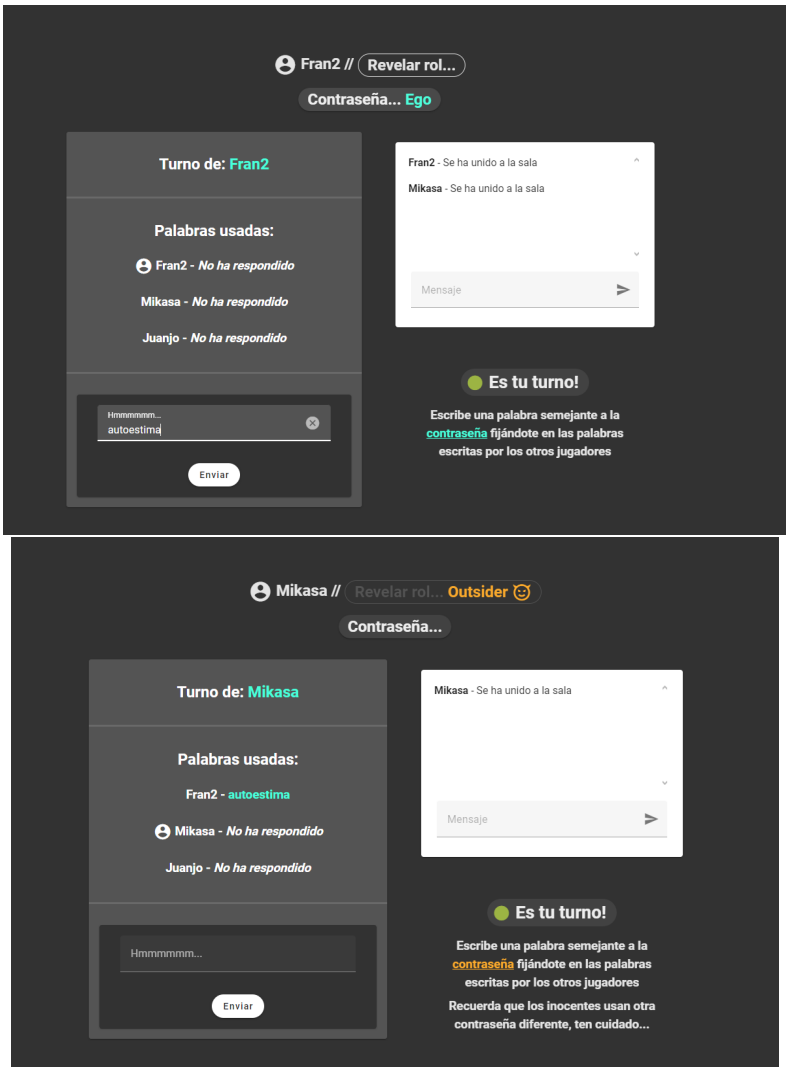

Figura 4.10: Pantalla de juego en la fase principal de juego

También se destaca la gestión de elementos para poder mostrar de forma clara y sencilla una interfaz de votación. En esta parte del juego los jugadores podrán seleccionar a cualquier jugador (incluso a ellos mismos) para votarlo con intención de eliminarle del juego. Debido a que en esta sección del juego se realiza una votación asíncrona, se incluye un temporizador para votar en blanco si el jugador no ha escogido a un jugador en el tiempo indicado como se puede apreciar en la Figura 4.11.

Figura 4.11: Proceso de votación entre jugadores

En último lugar se destaca que la lógica y elementos relacionados con los resultados entre rondas se describen en el componente "ResultsComponent.vue". En la Figura 4.12 se muestra un caso específico de fin de ronda, en el cual los jugadores han votado en su mayoría a un jugador Inocente, el cual es eliminado. En estos casos, se le facilita al capitán de la sala poder jugar otra ronda adicional o darle la victoria directamente al jugador Outsider; queda en manos del jugador/capitán en cuestión.

Figura 4.12: Resultados de ronda

4.4.5. Problemas afrontados durante el desarrollo

En esta sección se destacan los principales problemas que se han ido encontrado a lo largo del desarrollo, especialmente los relacionados con la lógica en el backend.

Problemas de sincronización

En primer lugar, es muy importante tener en cuenta problemas de sincronización como salas ya creadas, o códigos no válidos. A la hora de gestionar a varios jugadores en una misma sala eran recurrentes los problemas relacionados con la información que tiene un consumidor y otro, ya que, la mayor parte de la lógica, por ejemplo, calcular el resultado de una votación, es una tarea que solo debería realizar un consumidor en concreto que estuviese a la espera del resto de votaciones. Por ello, se trabajó mucho en la sincronización de información entre todos los consumidores, así como en la creación de una lógica de listeners robusta en el frontend; siempre a la espera de nuevos mensajes websocket provenientes del servidor.

Por otra parte, se ideó un mecanismo clave en la gestión asíncrona de la lógica del juego: Hay muchos puntos en los cuales cualquier jugador podría empezar/acabar con la partida o como se acaba de comentar, hay partes de código que solo tiene sentido que gestione un usuario/consumidor en exclusiva, para no repetir cálculos y respetar una sincronía dentro de la comunicación asíncrona. Este tipo de problemas pueden resultar familiares a los desarrolladores que hayan trabajado con este tipo de tecnologías, por ejemplo, una solución se basa en el uso de "barreras" a la hora de forzar la espera de un proceso en un punto hasta que el resto de procesos hayan llegado al mismo punto [25].

Finalmente, se llegó a una solución sencilla que también ayuda al diseño y experiencia de usuario: Un rol adicional de capitán o jefe de la sala. La clave radica en la sencillez de la idea, donde, el primer jugador que entra en una sala (el creador) es el capitán de la sala, y eso le permite iniciar la partida y tener la capacidad de terminarla entre rondas. De esta forma, gran parte de lógica la asume el consumidor del websocket asociado al usuario capitán. El único problema es la posible desconexión de este capitán, por ello, una gran parte del desarrollo ha sido la correcta sincronización de conexiones dentro de la sala. Para evitar posibles problemas en la lógica del juego, si un jugador (que no ha sido eliminado y no está como espectador en el juego) se desconecta en mitad de partida, se cancela la partida y los jugadores vuelven a la página principal.

Esto es debido a que re-calcular la lógica en mitad de partida se hace demasiado compleja, especialmente mostrar los cambios de forma visual y sencilla a los usuarios. Por otra parte, las partidas son rápidas y se persigue un trato de la

información sencillo y poco persistente. Solo existe información de los usuarios mientras están en una sala de juego, si se cierra la página web no hay ninguna traza de información que se haya almacenado obviando el historial de navegación y los datos asociados a la caché del propio navegador. En la Figura 4.13 se puede apreciar el diálogo de desconexión estándar que se muestra cuando un jugador abandona una partida en curso.

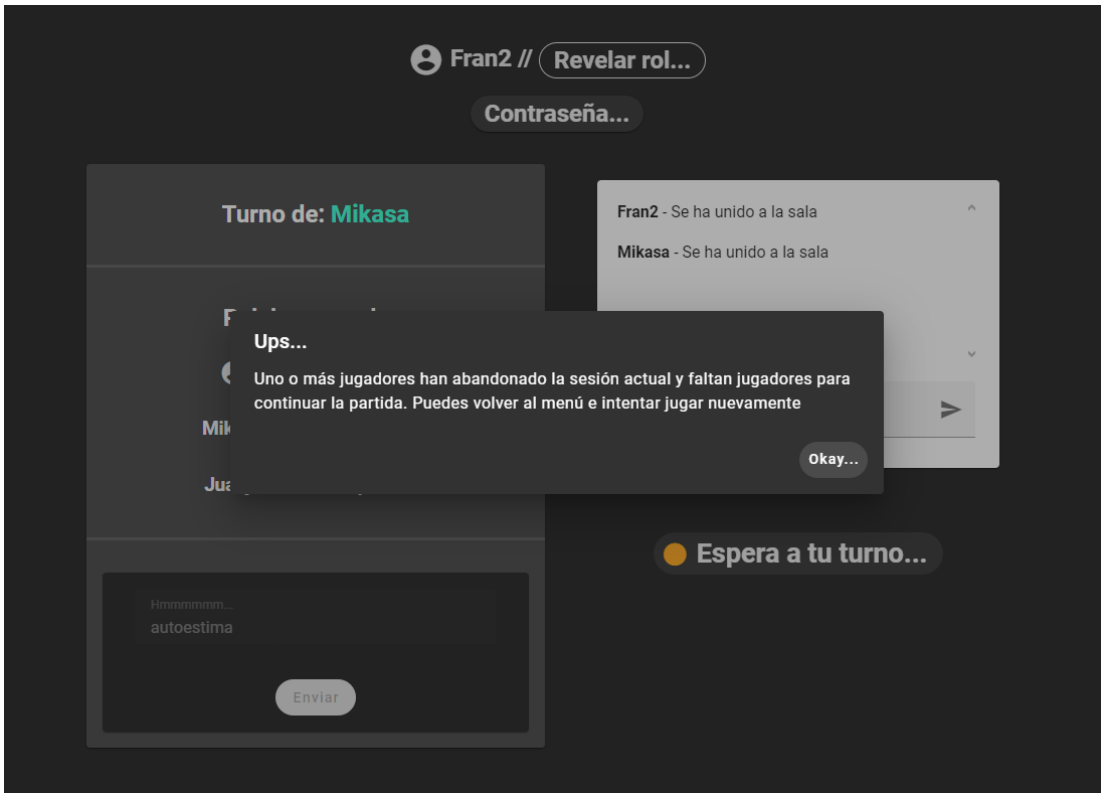

Figura 4.13: Diálogo de desconexión en la pantalla de juego

Resultados de ronda

La gestión de los resultados entre diferentes rondas se considera bastante compleja; el juego funciona de forma sencilla, pero con todas las reglas aplicadas, existen una gran cantidad de resultados a la hora de tratar la información de votación de los jugadores, desde empates, hasta la imposibilidad de seguir jugando de forma coherente (mismo número de Outsiders que de jugadores Inocentes). Por ende, es importante tener en cuenta lo complicado que se puede volver un simple diálogo de texto. Además de tener que gestionar la lógica y comunicar esta información entre jugadores, es necesario considerar todos los resultados posibles. En este aspecto el testing automático puede facilitar las cosas y por ello puede resultar bastante útil aplicarlo desde un primer momento.

Lista de palabras

El juego debería proporcionar una lista de palabras para que los usuarios jueguen con las diversas palabras clave. De forma adicional, se ha querido poder modificar esa lista de palabras de forma sencilla mediante el administrador de Django y un campo JSON. Dicho esto, se ha ido modificando una regla que indica que se le debe proporcionar al jugador Outsider una "palabra clave falsa" si es el primer jugador en una ronda; esto es porque si no tiene ninguna clase de pista adicional, se hace complicado evitar ser descubierto como Outsider. Por ende, la lista de palabras se hace algo más compleja, ya que se propone que el Outsider debería tener otra palabra que se parezca a nivel semántico a la palabra clave, pero sin ser la propia palabra clave. Para generar esta lista de tuplas se ha hecho uso de ChatGPT [26] para poder tener al menos unos datos iniciales con los que testear y mostrar la aplicación evitando palabras repetidas entre diferentes partidas. Igualmente, esto sería un aspecto a mejorar e incluso se podrían modificar las reglas para poder tratar con este aspecto en especial, porque, incluso haciendo uso de IA, se ha hecho muy poco consistente esta generación de tuplas de palabras semánticamente similares.

4.5. Pruebas automáticas

Ya sea para detectar errores a la hora de añadir una nueva funcionalidad, garantizar la calidad general del código o directamente descubrir problemas desconocidos hasta el momento, el proceso de testing es una práctica fundamental en el desarrollo de software.

Para este proyecto se ha querido experimentar con algún framework de testing para poner a prueba las conexiones websocket. Por ejemplo, como se había indicado en la sección anterior de problemas, los resultados entre rondas (4.4.5) pueden llegar a ser muy variados e impredecibles debido a la cantidad de combinaciones posibles dependientes del uso de conexiones websocket estables y controladas. Por otro lado, hacer todas estas pruebas de forma manual puede resultar bastante abrumador y siempre existe el propio error humano característico en estas pruebas; confundirse al votar a un jugador o agotar el tiempo de manera involuntaria pueden comprometer este tipo de pruebas manuales especialmente si se quieren testear gran cantidad de casos.

A este respecto, se ha hecho uso de PyTest [3], un framework que permite la creación de tests de forma sencilla en Python. Se ha hecho uso de PyTest sobre otros de los muchos framework de testing debido al uso directo de Python y a las facilidades para testear tecnologías asíncronas, lo cual cubre la parte de código más sensible y compleja de la aplicación.

Dentro de la aplicación "logic" en la carpeta fuente del proyecto, se encuentra el fichero "test_websocket_backend.py". Dentro de este fichero se encuentran preparados para ejecutarse diferentes tests que ponen a prueba las conexiones websocket y la lógica de juego. El funcionamiento de un test de PyTest es bastante sencillo: Como se puede observar en la Figura 4.13, lo primero es destacar el nombre del test, este debería ser representativo y claro; en este caso se trata del test "test_room_connection_wrong_room", lo que indica que se comprobará la conexión a una sala errónea, es decir, se va a suponer un usuario que quiera conectarse a una sala que existe pero va a hacer uso de un código de sala erróneo, por ende, el sistema debería rechazar esa conexión errónea por parte del usuario.

Lo primero que se suele hacer en un test es crear información simulada, normalmente denominada como "mock data". Estos datos simulados sirven para darle un contexto al test en cuestión. En este caso, previamente al código de la función, con ayuda de los "helpers/fixtures" de Pytest se creará una sala en la base de datos simulada la cual está vacía. Dependiendo del entorno de trabajo, los datos de simulación y el cómo se crean variarán. Además, debido a la asincronía presente en la mayor parte del código, se harán uso de herramientas específicas, ya que, al igual que en el framework estándar, mezclar peticiones síncronas a la base de datos con mensajes websocket suele requerir algún paso extra. En este caso y en los tests que lo requieran se hará uso de etiquetas como sync_to_async.

Teniendo los datos de mock preparados, el test ejecuta la lógica de peticiones pertinente, en este caso, primero se hace una simple comprobación para verificar que se ha creado la sala de pruebas de forma correcta. Luego, como con la mayoría de tests, se intentará realizar una conexión websocket haciendo uso de las capacidades de testing integradas en Channels (a través del uso de "WebsocketCommunicator").

Finalmente se realizan comprobaciones mediante funciones "assert". En este caso simplemente se comprueba que no se ha realizado o en todo caso se ha rechazado la conexión websocket debido a que se hace uso de un código erróneo de sala; se utiliza "wrong_room" en vez de "test_room". Dependiendo del test se querrán comprobar diferentes cosas, por ejemplo, comprobar que el número de Outsiders sea el adecuado al iniciar una partida o que las conexiones y desconexiones generen los resultados esperados dependiendo del estado de la partida.

Cabría destacar que la mayoría de lógica que se testea es en su mayoría las conexiones websocket, por ello, se agiliza la creación de salas a traves de los "helpers/fixtures" de PyTest, que están diseñados con esa misma intención, preparar el entorno de forma previa a la ejecución de un test.

En el Anexo B.1 se pueden encontrar las instrucciones en detalle para preparar el entorno y ejecutar los tests.

4.6. Pruebas manuales

El aseguramiento de la calidad (frecuentemente denominado con el anglicismo "Quality Assurance", "QA") comprende un conjunto de actividades planificadas y sistemáticas implementadas dentro de un sistema con el objetivo de garantizar que se cumplan los requisitos de calidad de un producto o servicio.

Además de todo el trabajo relacionado con las pruebas automáticas, es destacable el esfuerzo de probar todos los aspectos de la aplicación de forma directa. En grandes proyectos es normal encontrar un equipo dedicado exclusivamente a QA, haciendo uso de la aplicación en detalle además de reportando de forma conveniente todos los problemas encontrados a los desarrolladores. Se han tenido en cuenta la realización de partidas con diferente número de jugadores, los diferentes resultados de ronda y la correcta visualización de elementos en pantalla; algo más difícil de probar de forma automática. A modo de ejemplo, detectar un correcto uso de colores o corroborar que existe una lógica de navegación intuitiva no son pruebas que se puedan ejecutar fácilmente (hay opciones, pero son costosas o simplemente requieren esfuerzo adicional de desarrollo) a través de un test automático.

Debido a que este proyecto ha sido realizado por una sola persona, el testing manual ha sido trabajado por el propio desarrollador de la aplicación. Por ello, aunque se prueben diferentes casos de prueba con diferentes dispositivos y configuraciones es destacable que se pueden obviar elementos y problemas debido al uso continuado e informado de la aplicación. El desarrollador sabe jugar al juego y puede que en comparación a otra persona externa al proyecto realice acciones más rápidas sin fijarse en elementos con claros errores en la interfaz de la aplicación. Por estas razones, se ha querido tener en cuenta la realización de una prueba con usuarios externos para tener en cuenta sus opiniones.

Esta prueba se realizó con la mayor parte de la lógica de la aplicación lista y sirvió para ajustar algunos elementos y corregir pequeños errores. En la Figura 4.14 se muestra una foto del día que se realizó esta prueba. Es importante destacar que se hizo con un número significativo de jugadores (en torno a diez jugadores) en un mismo espacio físico, debido a que la aplicación no estaba desplegada en la nube y era solo accesible de forma local. Aunque no se tuvieron en cuenta problemas derivados del juego a larga distancia o en otras redes, se pudieron observar problemas derivados de su uso en este tipo de escenarios. Al final, la aplicación posee un importante componente social y es relevante ver como las personas actúan y hacen uso de la aplicación en un mismo espacio y tiempo: Las expresiones faciales, la conversación en vivo o incluso la capacidad de hacer trampas son viables en este contexto.

Los errores a solventar y las mejoras propuestas en esta prueba fueron:

- Solucionar un error a la hora de enviar la palabra de juego escrita por el jugador. Es posible bloquear una ronda debido a la posibilidad de enviar dos veces la palabra por un jugador.
- Permitir la ocultación de roles.
- Cambiar una regla para que en el caso de existir dos Outsider, se elimine al primero directamente y solo se le de la capacidad de adivinar la palabra clave al segundo Outsider.
- Solo proporcionar la palabra clave (una diferente a la de los jugadores Inocentes) al Outsider si es el primer jugador de la ronda. En otro caso nunca tendrá nada al empezar.
- Ajustar el número máximo de jugadores por partida.

Las diferentes propuestas fueron realizadas en la aplicación final y el error fue solventado. Además de esto, sirvió para observar su uso por parte de otros usuarios y tener en cuenta el uso de diferentes dispositivos a la hora de utilizar la aplicación, ya que cada uno hizo uso de su propio teléfono móvil.

Figura 4.14: Prueba con usuarios

4.7. Despliegue

Para finalizar este capítulo sobre la descripción informática de la aplicación, quedaría por detallar el proceso de despliegue de la aplicación.

En primer lugar, sería indicado comentar que el despliegue se ha realizado para hacer uso de la aplicación desde cualquier dispositivo con conexión a internet y un navegador web. El despliegue se puede hacer de muchas formas y haciendo uso de diferentes tecnologías. Para este proyecto se ha escogido Amazon Web Services (AWS) como plataforma de despliegue principal [13]. AWS ofrece una cantidad muy abundante de servicios, ya que, da servicios a toda clase de clientes y es una de las plataformas más usadas y conocidas en el ámbito de computación en la nube. Son tantos los servicios, que se hace complejo el simple hecho de listarlos todos.

En este caso, se hará uso de la plataforma Amazon Elastic Compute Cloud (Amazon EC2), normalmente denominada como EC2. Esta plataforma proporciona capacidad de cómputo con tamaño modificable en la nube aunque en el marco del proyecto solo se hará uso de una pequeña máquina t2.micro con Linux (específicamente con una imagen "Ubuntu Server 24.04 LTS") como sistema operativo. Además de venir incluida en la capa gratuita de AWS, no es necesario contratar una máquina con más capacidad de cómputo.

Además de la instanciación de la máquina, se propone la obtención de un dominio web para facilitar y dejar más vistoso el acceso a la aplicación web así como de hacer uso de Route 53 para gestionar el DNS requerido para direccionar este dominio con la IP de la máquina.

Con la plataforma lista para el despliegue, solo quedaría preparar la aplicación. Esto se ha hecho principalmente con el uso de Docker y Docker Compose a través del uso de contenedores. En concreto, se ha creado un fichero "docker-compose.yaml" para gestionar la "dockerización" de los servicios de la aplicación. En el siguiente Código 4.4 se destacan las secciones de este fichero en el cual se configuran el servidor Redis para conexiones websocket, el backend de Django y el frontend desarrollado en Vue. Además de lo dockerización per se, es necesario ajustar el uso de urls y gestionar la configuración de carga de los diferentes servicios, por ejemplo, se definen archivos "Dockerfile" tanto para los contenedores de Django (localizado en "outsider/Dockerfile") como para Vue (definido en "outsider-front/Dockerfile").

Esta configuración puede resultar compleja si no se tiene experiencia previa, ya que, lo que se hace es preparar la aplicación para poder ejecutarse más rápido a través de Docker. Es necesario describir todos las instalaciones que se deben realizar y configurar el entorno de ejecución de cada servicio en particular. Por ejemplo, se deben indicar los puertos de escucha del backend a la hora de ejecutar el comando "manage.py" de Django.

De esta forma, se tiene una aplicación portable y lista para poder ejecutarse en la nube. Para ello, solo habrá que descargarse su contenido en la máquina EC2 que se halla escogido y ejecutar el mandato de Docker Compose pertinente. En la Figura 4.15 se muestran los contenedores docker desplegados en la máquina a

Código 4.4: Estructura del fichero Docker Compose

```
services:
    # Redis server for websocket use
    redis: ...

    # Django backend container with WS logic (Django
        Channels)
    django: ...

    # Main container / Vue frontend
    vue_frontend: ...
```

través de AWS y a través del dominio outsidergame.top se puede acceder fácilmente a la aplicación web. En el Anexo B.2 se especifican las instrucciones para el despliegue de la aplicación.

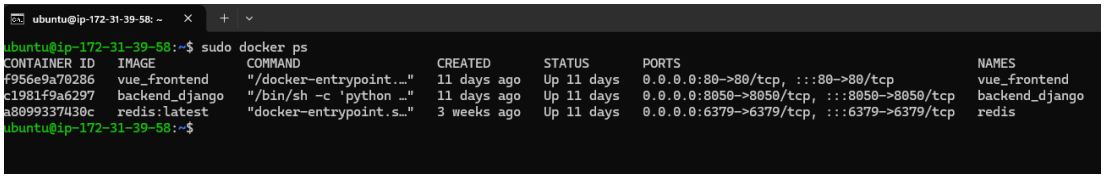

Figura 4.15: Contenedores Docker en ejecución de la aplicación

5

Conclusiones y trabajos futuros

5.1. Conclusiones

La finalidad de este proyecto es, tanto aprender y hacer uso de tecnologías asíncronas modernas, como la creación de una aplicación totalmente funcional que implemente estas tecnologías en detalle.

El objetivo principal es la creación de una aplicación que emule de la forma más completa el juego de adivinanzas de palabras y roles que se propone. Además de ir implementando diferentes reglas y funcionalidades al juego base, se proponen otros objetivos secundarios como la implementación de testing o el despliegue de la aplicación a través de un proveedor de servicios cloud. Se quiere profundizar en el uso de websockets y lo que implica su desarrollo, no simplemente quedarse en la superficie, por ello, todo el desarrollo gira entorno a este eje central.

La aplicación final incluye múltiples sistemas interconectados que permiten su correcto funcionamiento y uso por parte del usuario final a través de un navegador web. Para lograr esto, ha sido necesario trabajar con diversas tecnologías/frameworks e implementar la lógica y diseño necesarios para crear: Un servidor backend encargado de manejar la lógica websocket y un frontend responsable de facilitar una interfaz web con todas las funcionalidades pertinentes. También ha sido necesario gestionar la comunicación necesaria entre los sistemas así como administrar los elementos adicionales necesarios.

Para el trabajo relacionado con el backend se destaca el uso complejo y en detalle de tecnologías websocket a través de Django Channels. Todos los mensajes

de juego y la comunicación en tiempo real entre jugadores no es una implementación sencilla y hay que tener en cuenta muchos aspectos de la comunicación asíncrona entre varios usuarios. Además de esta implementación, se ha realizado trabajo adicional de testing, tanto para aprender el correcto funcionamiento de la pruebas automáticas en estos entornos, como para poder probar de forma más elaborada la aplicación.

Por otra parte, para el desarrollo de la interfaz web, se han implementado interfaces mediante el uso de librerías de componentes (Vue y Vuetify) que facilitan mucho el trabajo de desarrollo a la vez que ofrecen un resultado bastante atractivo. También se destaca toda la lógica de intercomunicación y el código necesario implementado directamente en el frontend.

Además de la implementación, se destaca la preparación de la aplicación mediante Docker, para mejorar su portabilidad, y el despliegue final a través de AWS, lo que permite su fácil acceso y uso por parte de cualquier usuario.

Teniendo en cuenta lo explicado en los anteriores párrafos, se puede concluir que el proyecto cumple los objetivos propuestos a lo largo del desarrollo.

5.2. Aspectos pendientes y trabajos futuros

Teniendo en cuenta la realización de objetivos, existen varios aspectos a tratar en cuanto a la posibilidad de implementar mejoras o realizar trabajos de ampliación sobre la aplicación.

5.2.1. Información de usuarios

Debido a que no era el objetivo principal del proyecto, no hay ninguna implementación relacionada con la administración persistente de información sobre los jugadores. Añadir la capacidad a los usuarios para poder registrarse y tener guardada información sobre sus partidas e información en detalle podría ser interesante, especialmente a la hora de personalizar un poco más la aplicación.

Sería posible añadir un ranking o tener la capacidad de tener "amigos" dentro del juego para poder acceder a partidas entre usuarios conocidos de forma más sencilla. Los añadidos pueden ser muy diversos, pero todos requerirían un esfuerzo adicional para gestionar la comunicación síncrona de una base de datos de algún tipo junto a la dificultad añadida de implementar un sistema de autenticación de usuarios.

Por otra parte, todos los elementos adicionales que se han comentado (como la capacidad de tener una lista de amigos) deberían trabajarse de forma individual.

5.2.2. Lista de partidas

Otro añadido que se muestra interesante, es la capacidad de tener una lista de partidas abiertas para que un jugador se pueda unir. Este añadido permitiría a los jugadores que quieran jugar con desconocidos unirse de forma directa a una partida sin hacer uso de otros medios.

Con esta lista de partidas, sería ideal añadir la capacidad de crear una sala privada o una sala pública. Mientras que a una sala pública podría acceder cualquier usuario desde esta lista, una sala privada no aparecería en este listado (en el caso de existir muchas salas de juego) o en todo caso solo sería accesible con un código de acceso.

5.2.3. Opciones de partida

Relacionado con el anterior punto, la capacidad de modificar el estado de la sala y la partida por parte de los usuarios se planteó como un posible añadido para la aplicación. Por ejemplo, definir el número de jugadores Outsider o limitar el número de rondas por partida.

Esta implementación, debido a la carga de lógica que acarrea no se llegó a implementar, pero se destaca que podría ser un gran añadido y no requeriría la implementación de sistemas nuevos. Sería necesario guardar esta configuración de partida e implementar una lógica más compleja que tenga en cuenta todas estas variaciones de juego posibles. Ayudaría bastante seguir haciendo uso del sistema de testing para comprobar de forma más rápida todos estos nuevos cambios.

5.2.4. Escalabilidad

Además del testing implementado, es importante poner a prueba las capacidades de rendimiento de los sistemas. Podría ser muy interesante plantear un uso masivo de la aplicación y como se tendría que modificar el entorno para adaptarse a estos cambios. Ya que, puede que se llegué a un punto donde los servidores se vean sobrepasados o en todo caso, si se sigue haciendo uso de una máquina EC2 y de AWS, se genere un sobrecoste importante del uso de los servicios de cloud.

En el proyecto no se han realizado pruebas de este tipo y el simple hecho de plantear este tipo de problemas es bastante atractivo si se quieren explorar estos conceptos de rendimiento y capacidad de carga en un sistema informático.

Bibliografía

[1] D. S. Foundation and individual contributors, *Página principal de Django*. [Online]. Available: https://www.djangoproject.com

[2] D. S. Foundation, *Documentación de Django Channels*. [Online]. Available: https://channels.readthedocs.io/en/latest/

[3] p.-d. t. Holger Krekel, *Página principal de pytest*. [Online]. Available: https://docs.pytest.org/en/8.2.x/

[4] V. t. Evan You, *Introducción a Vue*. [Online]. Available: https://vuejs.org/guide/introduction.html

[5] V. t. Evan You, *Diferencias entre Vue2 y Vue3*. [Online]. Available: https://vuejs.org/about/faq#what-s-the-difference-between-vue-2-and-vue-3

[6] V. team, *Introducción a Vuetify, "¿Qué es Vuetify?"*. [Online]. Available: https://vuetifyjs.com/en/introduction/why-vuetify/#what-is-vuetify3f

[7] F. Inc, *Página principal de Figma design*. [Online]. Available: https://www.figma.com/es-es/design/

[8] J. Ltd, *Herramienta web draw.io*. [Online]. Available: https://app.diagrams.net

[9] P. contributors, *Iconos mdi de Pictogrammers*. [Online]. Available: https://pictogrammers.com/library/mdi/

[10] C. Finder, *Página principal de Contrast finder*. [Online]. Available: https://app.contrast-finder.org

[11] S. H. Docker Inc, *Página principal de Docker*. [Online]. Available: https://www.docker.com

[12] S. H. Docker Inc, *Docker: Containers*. [Online]. Available: https://www.docker.com/resources/what-container/

[13] Amazon.com, *Página principal de AWS*. [Online]. Available: https://aws.amazon.com/es/

[14] G. Inc, *Pagína principal de GitHub*. [Online]. Available: https://github.com

[15] J. R. A. Tejera, *Repositorio del proyecto*. [Online]. Available: https://github.com/Javiex7/Outsider

[16] M. Inc, *Instalación de WSL*. [Online]. Available: https://learn.microsoft.com/es-es/windows/wsl/install

[17] M. Inc, *Pagína principal de Visual Studio Code*. [Online]. Available: https://code.visualstudio.com

[18] D. S. Foundation, *Uso del sistema de 'layers' en Django Channels*. [Online]. Available: https://channels.readthedocs.io/en/latest/tutorial/part_2.html#enable-a-channel-layer

[19] A. Team, *Documentación de ASGI*. [Online]. Available: https://asgi.readthedocs.io/en/latest/

[20] M. Foundation, *Documentación para el uso objetos WebSocket en javascript*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebSocket

[21] Ably, *Guía de uso de Django Channels "Guide to Django Channels: What it is, pros and cons and use cases"*. [Online]. Available: https://ably.com/topic/what-is-django-channels

[22] D. S. Foundation, *Instalación de Django Channels*. [Online]. Available: https://channels.readthedocs.io/en/latest/installation.html

[23] D. S. Foundation, *Tutoriales de uso en Django Channels*. [Online]. Available: https://channels.readthedocs.io/en/latest/tutorial/index.html

[24] V. t. Evan You, *Usando Axios para consumir APIs*. [Online]. Available: https://es.vuejs.org/v2/cookbook/using-axios-to-consume-apis.html

[25] W. Foundation, *Wikipedia - Barrier (computer science)*. [Online]. Available: https://en.wikipedia.org/wiki/Barrier_(computer_science)

[26] OpenAI, *ChatGPT*. [Online]. Available: chatgpt.com

[27] R. P. Kenneth Reitz, *Instalación de Python3*. [Online]. Available: https://docs.python-guide.org/starting/install3/linux/

[28] D. Inc, *Instalación de Docker*. [Online]. Available: https://docs.docker.com/engine/install/

Apéndices

A

Esquemas y recursos adicionales

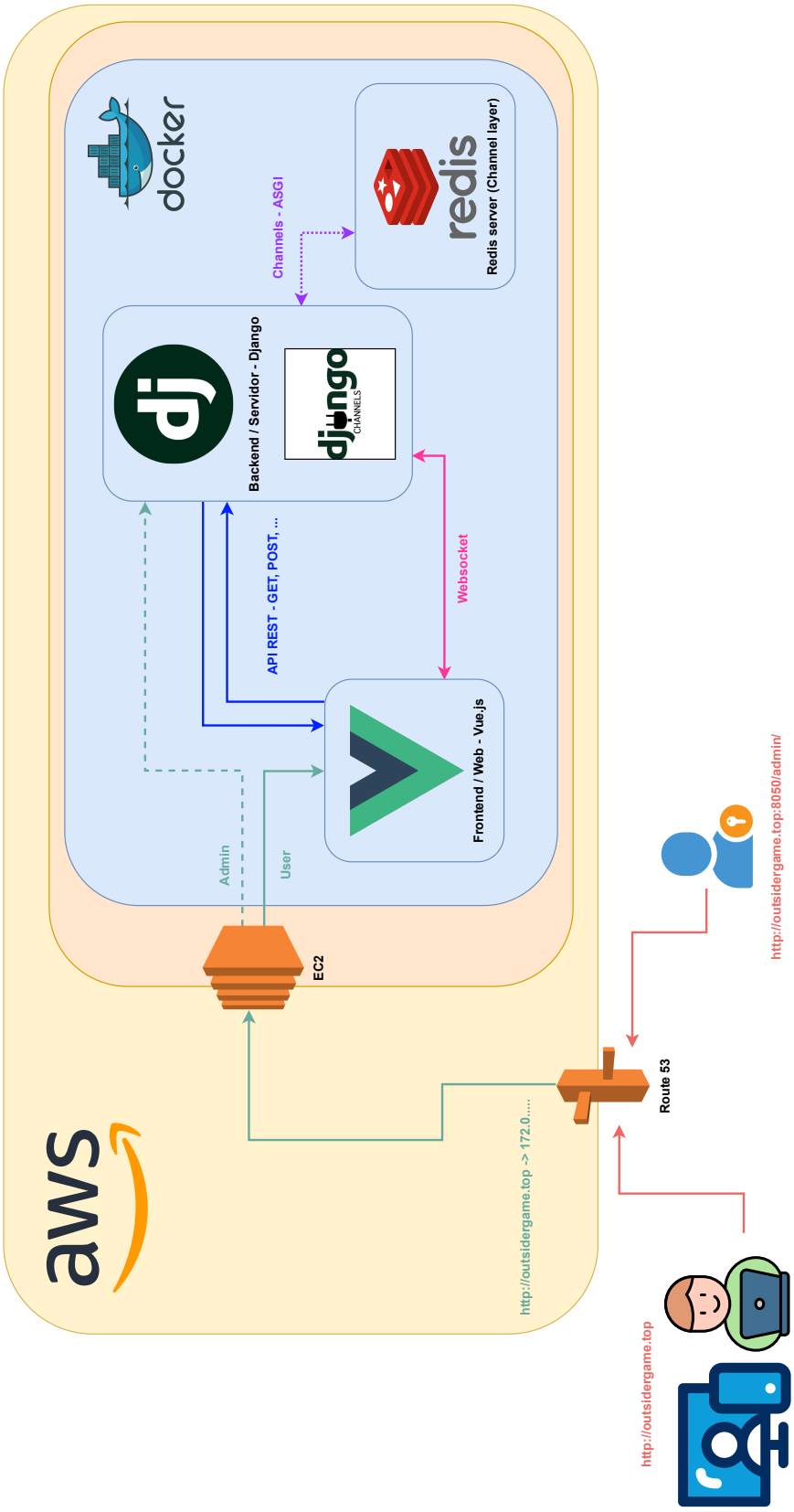

Figura A.1: Arquitectura final del sistema

B

Instrucciones de testing y despliegue de la aplicación

B.1. Tutorial de testing

En este apartado se indican los pasos a seguir para poder ejecutar los tests de la aplicación sin complicaciones. Se recomienda hacer la configuración necesaria en un sistema Unix ya que es donde se ha trabajado la aplicación y, más adelante, se indicará hacer uso de mandatos específicos de Unix.

1. Lo primero es tener el proyecto actualizado. El repositorio debería ser accesible para cualquier usuario en github.com/Javiex7/Outsider.
2. Para la ejecución de los tests en indispensable la instalación tanto de Python [27] como de Docker [28].
3. Con estos programas instalados en el sistema, es necesario acceder al repositorio para la configuración básica.
4. Dentro del proyecto, son necesarios la instalación de varios elementos en el dispositivo, específicamente Django y los paquetes pertinentes. Para evitar problemas, se recomienda descargar los elementos listados en el "requirements.txt" dentro de la carpeta principal "OutsiderProject". Esta instalación se puede realizar fácilmente mediante el mandato:

   pip install -r requirements.txt
5. A continuación, se requiere ejecutar un contenedor en Docker encargado de gestionar el servidor Redis para que se haga uso en los tests. Mediante el siguiente mandato se puede poner en ejecución el servicio:

   docker run –rm -p 6379:6379 redis:7

6. Antes de poder ejecutar los tests, se debe configurar un variable de entorno para indicar el uso de este servidor Redis. Para ello habría que acceder a "OutsiderProject/outsider/-settings.py" y modificar el flag booleano denominado "TEST" y asegurarse que su valor sea True (línea 92).
7. Finalmente, el sistema puede ejecutar los tests. Para ello solo habría que ejecutar el mandato "pytest" desde el directorio padre, "OutsiderProject". Después de la ejecución debería mostrarse por consola un resultado similar a lo que se puede visualizar en la Figura B.1

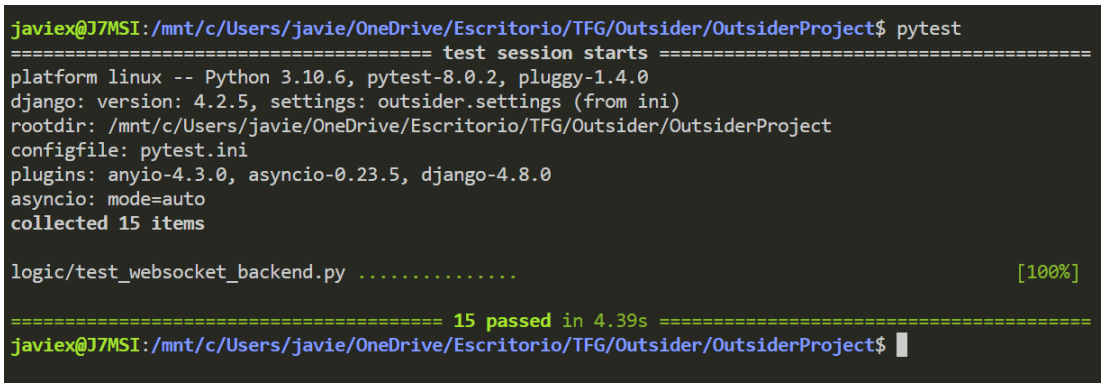

Figura B.1: Ejecución de tests

B.2. Instrucciones de despliegue

En este apartado se indican las instrucciones a la hora de realizar un despliegue de la aplicación a través de AWS. En el Código B.1 se indican todos los mandatos que se deben ejecutar y que se va a referenciar a continuación.

1. En primer lugar, es necesario tener una máquina EC2 preparada para poder conectarse a ella. En la Figura B.2 se puede observar el panel de control de EC2 de AWS. Desde este panel se pueden lanzar nuevas instancias y gestionar las que ya han sido configuradas y lanzadas anteriormente. Para este proyecto se está haciendo uso de la máquina nombrada como "outsider-machine"
2. Con la máquina lista, es necesario conectarse a ella mediante el uso del mandato ssh y las claves privadas generadas a la hora de crear la instancia de la máquina. Se destaca que se debe sustituir "machineDir" por la dirección DNS de la máquina en cuestión. Esta dirección y demás información relacionada con la conexión a una máquina EC2 se puede encontrar en la sección de "Conectarse a una instancia" dentro de la página de gestión de EC2.
3. Ya en el directorio principal, se puede crear una carpeta adicional o descargar el repositorio del código directamente. El repositorio debería ser accesible para cualquier usuario en github.com/Javiex7/Outsider y para descargarlo desde la terminal simplemente es necesario hacer uso del mandato "git clone".
4. Ahora solo quedaría acceder al directorio principal y ejecutar el mandato de Docker para ejecutar los contenedores descritos en el fichero "docker-compose.yaml".

5. Después del tiempo de descarga e instalación necesario, la aplicación estará lista para su uso mientras se mantenga activa la máquina EC2.

Código B.1: Mandatos de consola para el despliegue

```
sudo ssh -i "privatekeys.pem" ubuntu@machineDir...

git clone https://github.com/Javiex7/Outsider.git

cd Outsider/OutsiderProject
sudo docker compose -f docker-compose.yaml up --build
```

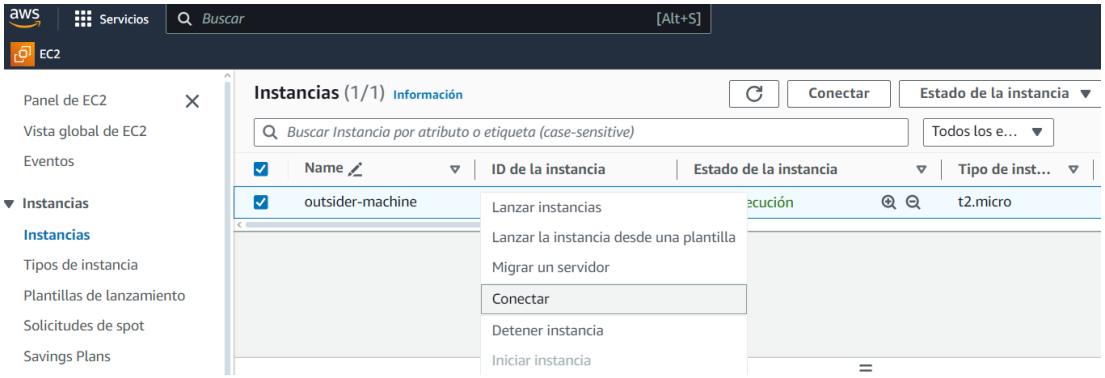

Figura B.2: Panel de control de EC2