UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Aplicación Web para la gestión de competiciones de deporte electrónico usando el Framework Ruby on Rails

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Sergio Rozalén Barberán

Tutor: José Vicente Busquets Mataix

Curso 2019-2020

Resum

L'objectiu d'aquest projecte és mostrar el procés de desenvolupament d'una aplicació Web que unifique les competicions d'esport electrònic amb la comunitat, permetent que jugadors no professionals puguen viure l'experiència de competir en tornejos d'esport electrònic.

La plataforma permetrà que els jugadors puguen participar en tornejos o fins i tot crear-los ells mateixos així com formar els seus propis equips o unir-se a equips existents.Els jugadors tindran la possibilitat de participar de diferents formes de manera individual, amb amics o trobar nous jugadors gràcies a les funcionalitats de xarxa social que també inclourà la plataforma.

L'aplicació web es desenvoluparà amb el framework RubyOnRails basat en el llenguatge de programació Ruby en el costat del servidor i amb el framework Vuejs basat en el llenguatge de programació Javascript en el costat del client. L'arquitectura de l'aplicació serà muntada sota un entorn Docker el que automatitzara el procés de desplegament i permitira que el programari siga escalable.

Paraules clau: xarxa social, esport electrònics, competicions, RubyOnRails, VueJs, Docker

Resumen

El objetivo de este proyecto es mostrar el proceso de desarrollo de una aplicación Web que unifique las competiciones de deporte electrónico con la comunidad, permitiendo que jugadores no profesionales puedan vivir la experiencia de competir en torneos de deporte electrónico.

La plataforma permite que los jugadores puedan participar en torneos o incluso crearlos ellos mismos así como formar sus propios equipos o unirse a equipos existentes. Los jugadores tendrán la posibilidad de participar de distintas formas de manera individual, con amigos o encontrar nuevos jugadores gracias a las funcionalidades de red social que también incluirá la plataforma.

La aplicación web sera desarrollada con el framework RubyOnRails basado en el lenguaje de programación Ruby en el lado del servidor y con el framework Vuejs basado en el lenguaje de programación Javascript en el lado del cliente. La arquitectura de la aplicación será montada bajo un entorno Docker lo que automatizara el proceso de despliegue y permitirá que el software sea escalable.

Palabras clave: red social, deporte electrónico, competiciones, RubyOnRails, VueJs, Docker

Abstract

The objective of this project is to show the process of developing a web application that unifies the e-Sport competitions with the community, allowing non-professional players to live the experience of competing in e-Sport tournaments.

The platform will allow players to participate in tournaments or even create them themselves, as well as form their own teams or join existing teams. players will have the possibility to participate in different ways individually, with friends

or find new players thanks to the social network functionalities that the platform will also include.

The web application will be developed with the RubyOnRails framework based on the Ruby programming language on the server side and the Vuejs framework based on the Javascript programming language on the client side. The application architecture will be mounted under a Docker environment which will automate the deployment process and allow the software to be scalable.

Key words: social media, eSports, competitions, RubyOnRails, VueJs, Docker

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

1.1 Motivación

Actualmente usamos todo tipo de dispositivos electrónicos distintos por lo que un requisito fundamental es que las aplicaciones puedan funcionar en todos los dispositivos, las tecnologías de desarrollo de aplicaciones web nos permiten que nuestras aplicaciones funcionen y se adapten al dispositivo desde el que se esta accediendo a la plataforma.

La motivación por la que me decante por este proyecto fue por el uso de tecnologías web como Ruby on Rails o VueJS. Ya que considero que aprender a usar estas tecnologías puede abrirme un gran abanico de posibilidades en el mercado laboral.

Por otro lado la temática de la aplicación esta relacionada con la industria de los videojuegos un sector en pleno auge que año tras año se esta consolidando como uno de los sectores mas fuertes en el ámbito del ocio junto a la musica y al cine.

Los antiguos juegos enfocados en manejar un personaje y narrar una historia cada vez tienen menos mercado [4], los jugadores ya no se conforman con el entretenimiento y lo que buscan es competir en un ranking con el resto para superarse día a día.

El problema a solucionar con esta aplicación consiste en que jugadores de un nivel semejante puedan interactuar en la plataforma para formar un equipo y competir juntos.

1.2 Objetivos

El objetivo principal de este proyecto es desarrollar una aplicación web que unifique las competiciones de deporte electrónico con las funcionalidades de una red social. El proceso de desarrollo de la plataforma consistirá en el aprendizaje de las tecnologías para en un futuro poder aplicarlas en el mundo laboral.

Los objetivos planteados son los siguientes:

- Aprender a hacer uso de Git como control de versiones.

- Diseñar una aplicación web desde cero analizando las necesidades del software.
- Diseñar las interfaces de la aplicación.
- Aprender a automatizar el despliegue de aplicaciones con la herramienta docker.
- Aprender un framework de desarrollo web como lo es RubyOnRails y desarrollar un API REST.
- Añadir seguridad a la API mediante el uso de tokens.
- Aprender un framework de desarrollo frontend basado en javascript en este caso VueJs y desarrollar el lado frontend de una aplicación web.

1.3 Metodología

El proceso de desarrollo en cascada o secuencial ha sido el elegido para desarrollar el proyecto. Este proceso consiste en ordenar las etapas de desarrollo donde una etapa es iniciada una vez que la anterior ha sido finalizada.

Las etapas por las que ha pasado el proyecto son las siguientes:

- **Análisis** Se analizara los requisitos tanto funcionales como no funcionales y las restricciones que la aplicación debe cumplir.
- **Diseño.** En esta etapa se diseñara la base de datos, la estructura de la aplicación y los prototipos de la interfaz.
- **Implementación.** La etapa de codificación se implementara el código tanto de la API REST como del frontend y además se dockerizará el entorno.
- **Implantación.** En esta etapa se desplegara la plataforma bajo el sistema operativo Linux Mint.
- **Pruebas.** En esta etapa se realizaran pruebas a la aplicación para comprobar su correcto funcionamiento.

El diagrama de la metodología en cascada se puede ver con mas detalle en la Figura 1.1

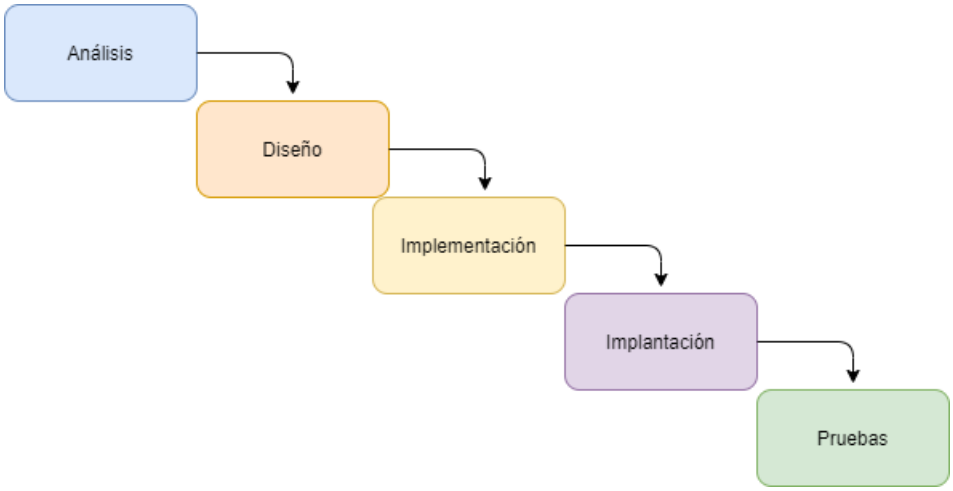

Figura 1.1: Metodología cascada

1.4 Estructura de la memória

Esta sección detalla la estructura que se ha seguido a lo largo del documento. La memoria se ha dividido en los siguientes capítulos:

Capítulo I: Introducción

En el capítulo I se ha dado una visión global del proyecto planteando la motivación, objetivos y la metodología a seguir en el proceso del proyecto.

Capítulo II: Herramientas de desarrollo y tecnologías

El capítulo II presenta las distintas herramientas de desarrollo, tecnologías y lenguajes de programación utilizados en el proceso de desarrollo del proyecto.

Capítulo III: Análisis del problema

El capítulo III contiene el Análisis del problema en el cual se ha analizado la aplicación a desarrollar, el análisis esta compuesto por una descripción general del producto y por la especificación de requisitos y restricciones necesarias en la aplicación.

Capítulo IV: Diseño de la solución

El capítulo IV detalla el diseño propuesto para la solución conteniendo la arquitectura del sistema, los prototipos de las interfaces gráficas y el esquema relacional de la base de datos.

Capítulo V: Desarrollo de la solución propuesta

El capítulo V muestra el proceso de desarrollo del proyecto el cual cuenta con tres grandes partes diferenciadas, la dockerización del entorno , la implementación de la API y la implementación del frontend.

Capítulo VI: Implantación

El capítulo VI describe el proceso a seguir para desplegar la aplicación.

Capítulo VII: Pruebas

El capítulo VII se detallaran las pruebas realizadas tras la implantación de la aplicación.

Capítulo VIII: Conclusiones

El capítulo VIII contiene las conclusiones del proyecto desarrollado en las cuales se detallan posibles trabajos futuros para ampliar la aplicación y la relación del trabajo con los estudios cursados en el grado.

CAPÍTULO 2
Herramientas de desarrollo y tecnologías

2.1 Lenguajes de Programación

2.1.1. Ruby

Lanzado en 1995 por el programador japonés Yukihiro Matsumoto el lenguaje de programación Ruby [19] es un lenguaje interpretado, reflexivo y orientado a objetos. Su sintaxis esta inspirada en los lenguajes de programación Phyton [18] y Perl [17]. [27]

Una de las peculiaridades de ruby es que todos los tipos de datos son tratados como objetos, incluidos los tipos de datos primitivos como lo pueden ser un entero, una cadena o un decimal. También soporta herencia con enlace dinámico singleton y mixins pero no soporta herencia múltiple.

2.1.2. Javascript

Javascript [15] fue lanzado en 1995 con el nombre de Mocha fue creado por Brendan Eich de Netscape, es un lenguaje interpretado, orientado a objetos, imperativo, débilmente tipado y dinámico.

Actualmente javascript es uno de los lenguaje de programación mas populares del mundo [13], aunque inicialmente fuese popular en el lado del cliente han ido surgiendo soluciones del lado servidor como NodeJs [16].

2.2 Frameworks

2.2.1. Ruby On Rails

Ruby on Rails [5] fue lanzado el 13 de Diciembre de 2005 es un framework basado en el lenguaje de programación Ruby.

Este framework forma parte de la familia de frameworks Modelo Vista Controlador (MVC), entender el funcionamiento de MVC ayudara en gran medida a entender el funcionamiento de Ruby On Rails. [32]

La filosofía de Ruby on Rails se basa en dos principios:

- **DRY ("Dont't Repeat Yourself").** Duplicar código es una mala practica en lugar de escribirlo una vez y utilizarlo en varias partes del código.
- **Convención sobre configuración.** Lo que permite simplificar el código si seguimos las convenciones de rails en lugar de nuestras propias reglas.

Concretamente en este proyecto se ha usado Ruby On Rails como API REST, una funcionalidad disponible en rails desde su versión 5.

2.2.2. VueJs

En 2014 Evan You decidió crear su propio framework de frontend con la idea de minimizar el código de javascript y ahorrar tiempo al programador. [28]

VueJs [6] es un framework de javascript para desarrollar interfaces de usuario y aplicaciones basadas en una sola página, su popularidad ha ido creciendo hasta alcanzar el top de frameworks de javascript [25].

Esta basado en el uso de componentes los cuales permiten crear elementos HTML personalizados para ser reutilizados en toda la aplicación.

A continuación podemos ver un ejemplo de como declarar un nuevo componente en VueJs:

```
Vue.component('componente', {
  // opciones del componente
})
```

Lo que permitirá usar el componente en el resto de la aplicación de la siguiente manera:

```
<div id="container">
  <componente></componente>
</div>
```

2.3 Otras tecnologías empleadas

2.3.1. PostgreSQL

Postgre SQL [22] es uno de los sistemas de gestión mas populares [2] hace uso del lenguaje SQL por lo que es relacional y orientado a objetos. Entre varias de sus ventajas destacan:

- Sigue el estándar SQL.
- Gratuito y libre.

- Potente y robusto
- Gran escalabilidad.
- Es ampliable gracias a la cantidad de extensiones distribuidas que existen.

2.3.2. Dbeaver

Como software cliente de SQL para poder gestionar la base de datos se ha optado por Dbeaver [23] un software lanzado en 2010 creado por Serge Rider.

2.3.3. RubyMine

RubyMine [12] es el IDE seleccionando para llevar a cabo la implementación de este proyecto, lanzado en 2015 este IDE esta enfocado al desarrollo con Ruby, Ruby on Rails y desarrollo web.

Jetbrains [14], la compañía a la cual pertenece este IDE, ofrece un gran abanico de software para desarrolladores contando con un IDE para prácticamente cualquier lenguaje de programación.

El acceso a este IDE ha sido gracias a una licencia de uso educativo por ser estudiante universitario ya que el software de Jetbrains es de pago.

2.3.4. Docker

Docker [10] es un software que permite construir contenedores para automatizar el despliegue de aplicaciones, nos proporciona una capa de abstracción que nos permite ejecutar los contenedores en distintos sistemas operativos facilitando tanto el despliegue como el montaje del entorno de desarrollo. [30]

Al contrario que las maquinas virtuales las cuales consumen una gran cantidad de recursos del sistema los contenedores de Docker se ejecutan directamente en el Kernel lo que permite optimizar mas los recursos del sistema.

Otra de las ventajas es que solo necesitaremos instalar la dependencia de Docker en el lugar que queramos desplegar la aplicación para poder lanzarla, este proyecto por ejemplo se ha desarrollado en varios equipos distintos donde cada uno de ellos contaba con distintos sistemas operativos.

2.3.5. Control de versiones: Git y Git Hub

Las herramientas de control de versiones ayudan al equipo de desarrollo a llevar un seguimiento de todos los cambios que el código va experimentando a lo largo de la etapa de implementación.

La implementación del proyecto se ha desarrollado desde tres equipos distintos contar con el código en un repositorio online ayuda a minimizar errores en la aplicación así como a poder desarrollar distintas partes del proyecto sin tener que preocuparse por fusionar el código manualmente.

En caso de algún error o conflicto las herramientas de control de versiones nos permiten regresar a un estado anterior del proyecto.

En este proyecto se ha optado por el uso de git [8] como herramienta de control de versiones, git fue lanzado en 2005 por su creador Linus Torvalds. El repositorio se ha alojado en la plataforma Git Hub [21] lanzada en 2008 la cual aloja mas de 100 millones de repositorios contando con 31 millones de programadores registrados y hasta 1.1 billones de contribuciones entre proyectos. [24]

2.3.6. Gestor de paquetes: NPM

El gestor de paquetes NPM (Node Package Manager) [7] fue desarrollado por Isaac Schlueter, esta desarrollado en el lenguaje javascript siendo el gestor de paquetes por defecto de Node Js.

NPM como gestor de paquetes nos permite administrar las dependencias necesarias en el proyecto descargando e instalando desde sus repositorio las librerías requeridas por el proyecto.

Los paquetes instalados desde NPM se almacenaran en la carpeta del proyecto llamada node_modules aunque también podemos instalar los paquetes de forma global.

2.3.7. Vue CLI

El framework VueJs cuenta con una linea de interfaz de comandos propia llamada Vue Cli, esta herramienta nos facilita el desarrollo de los proyectos Vue.

Vue Cli es una librería que podemos instalar desde el gestor de paquetes NPM.

```
1 $   npm install -g @vue/cli
```

Haciendo uso de la herramienta Vue CLI podemos crear el proyecto de Vue, instalar paquetes de Vue que nos ayudaran a aumentar las funcionalidades del propio framework o incluso lanzar el servidor con un sencillo comando.

2.3.8. JSON

JSON (JavaScript Object Notation) usa la sintaxis de los objetos de Javascript pero es aceptado por la gran mayoría de lenguajes de programación, el formato de texto JSON ha sido usado para el intercambio de datos entre la API en Ruby on Rails y el frontend en VueJs.

CAPÍTULO 3
Análisis del problema

En la última década el sector de los videojuegos ha aumentado considerablemente, año tras año el numero tanto de jugadores como de competiciones como de espectadores aumenta. El auge de los videojuegos multijugador ha echo evolucionar al sector a una escena mas competitiva donde los jugadores prefieren competir contra otros.

La plataforma web a desarrollar en este proyecto quiere acercar la sensación de las competiciones a jugadores no profesionales. Permitiendo que los propios jugadores gestionen su torneo y compitan junto a sus amigos o entablen contacto con nuevos jugadores.

A continuación en este capitulo se realizara la especificación de requisitos de este proyecto siguiendo el estándar IEEE830 [26].

3.1 Introducción

3.1.1. Propósito

En las secciones que componen este capítulo serán definidos los requisitos funcionales, requisitos no funcionales y requisitos del sistema de la plataforma web que permitirá unificar una red social con la gestión de torneos de deporte electrónico.

3.1.2. Ámbito del sistema

La plataforma web desarrollada en este proyecto bajo el nombre de Gaamix consiste en una aplicación web que permita unificar las funcionalidades de una red social con la gestión de torneos bajo una misma aplicación.

Gaamix permitirá que sus usuarios puedan participar en torneos o gestionarlos ellos mismos además de poder comunicarse con el resto de la comunidad a través de mensajes.

Los usuarios podrán participar en torneos de manera individual o en equipo, en caso de no pertenecer a ningún equipo los usuarios tendrán la posibilidad de crearse el suyo propio.

Como objetivo Gaamix pretende dar la posibilidad a los jugadores no profesionales de organizar o participar en torneos de una manera sencilla formando equipo con sus amigos o encontrando a nuevos jugadores en la comunidad con los que participar.

3.1.3. Definiciones, acrónimos y abreviaturas

- IEEE: Instituto de Ingeniería y Electrónica
- Gaamix: El nombre que recibe la plataforma a desarrollar
- Application Programming Interface (API): Interfaz de programación de aplicaciones.
- RubyOnRails: Framework basado en el lenguaje de programación ruby el cual sera usado en el backend de la aplicación.
- VueJs: Framework basado en el lenguaje de programación javascript el cual sera usado en el frontend de la aplicación

3.1.4. Visión general del documento

Es necesario analizar los requisitos antes de desarrollar cualquier producto software.

Este capítulo constará de tres partes esta misma introducción, la sección 3.2 constará de la descripción de la funcionalidad. Y por último en la sección 3.3 se especificarán los requisitos y atributos del sistema.

3.2 Descripción General

3.2.1. Perspectiva del Producto

Gaamix es una plataforma web que permite a sus usuarios participar o gestionar torneos de videojuegos. Los usuarios podrán participar en estos torneos individualmente o en equipo. El usuario podrá contactar con un equipo para unirse o podrá crear su propio equipo. Además contara con funcionalidades de red social para que los usuarios puedan interactuar entre ellos.

3.2.2. Funciones del Producto

Las siguientes funcionalidades serán implementadas en la plataforma web:

- Gestión de usuarios: registro de usuarios, edición del usuario y borrado de usuario.
- Inicio de sesión: un usuario podrá iniciar sesión con sus datos

- Publicación de mensajes: los usuarios podrán crear publicaciones.
- Visualización de publicaciones: las publicaciones se podrán leer desde un tablón.
- Gestión de torneos: crear un torneo, editar su información y reporte de partidos.
- Unirse a torneo: los usuarios podrán registrarse en los torneos de forma individual.
- Gestión de equipos: creación de equipos, edición de equipo y borrado de equipos.
- Invitaciones de equipos: los equipos podrán enviar invitaciones a los jugadores para que los jugadores formen parte del equipo.
- Invitaciones a usuarios: los usuarios podrán aceptar o rechazar las invitaciones que los equipos les envíen.
- Equipos en torneos: el administrador del equipo podrá registrar al equipo en los torneos que tengan como modalidad equipos.
- Reporte de partidos: los usuarios podrán reportar el resultado del partido jugado.

3.2.3. Características de los Usuarios

El publico objetivo de Gaamix son los jugadores de videojuegos no profesionales los cuales podrán participar en torneos de manera individual, unirse en algún equipo existente o crear su propio equipo para participar en torneos. Por otro lado además de poder participar en torneos los usuarios también podrán de disfrutar de una red social orientada a la comunidad de videojuegos dándoles la posibilidad de conocer a nuevos jugadores.

3.2.4. Restricciones

Durante el desarrollo del proyecto se deben de tener en cuenta ciertas restricciones:

- Para mayor seguridad algunas peticiones a la API deberán estar autenticadas enviando un token en la cabecera de la petición. Este token sera devuelto por la API al inicio de sesión del usuario.
- Sera necesario que el diseño de la plataforma se adapte a distintas resoluciones permitiendo su correcto funcionamiento en cualquier dispositivo.
- Los lenguajes de programación utilizados serán RubyOnRails en el lado servidor y VueJs en el lado cliente.

3.2.5. Suposiciones y Dependencias

Será necesario que el dispositivo con el cual se acceda a Gaamix cuente con conexión a internet.

3.3 Requisitos específicos

3.3.1. Funciones

Las tablas que se muestran a continuación especifican uno a uno los requisitos funcionales que requiere la aplicación, son un total de veintiún requisitos funcionales.

Identificador	**RF-01**
Nombre	Registro de usuario
Descripción	Un usuario debe de registrarse para poder acceder a la mayoría de funcionalidades del sistema.
Precondición	-
Postcondición	-

Tabla 3.1: Requisito funcional RF-01

Identificador	**RF-02**
Nombre	Edición de usuario
Descripción	Un usuario deberá tener la posibilidad de actualizar la información de su perfil.
Precondición	El usuario debe de iniciar sesión.
Postcondición	-

Tabla 3.2: Requisito funcional RF-02

Identificador	**RF-03**
Nombre	Borrado de usuario
Descripción	Un usuario tendrá la posibilidad de borrar su perfil y todos sus datos.
Precondición	El usuario debe de iniciar sesión.
Postcondición	-

Tabla 3.3: Requisito funcional RF-03

Identificador	**RF-04**
Nombre	Inicio de sesión
Descripción	Un usuario podrá acceder a la plataforma con sus datos.
Precondición	El usuario debe de estar registrado.
Postcondición	-

Tabla 3.4: Requisito funcional RF-04

Identificador	**RF-05**
Nombre	Publicar mensaje
Descripción	Un usuario registrado podrá publicar un mensaje en el tablón de la plataforma.
Precondición	El usuario debe de iniciar sesión.
Postcondición	-

Tabla 3.5: Requisito funcional RF-05

Identificador	**RF-06**
Nombre	Visualizar publicaciones
Descripción	Un usuario podrá visualizar el tablón con las publicaciones del resto de usuarios.
Precondición	-
Postcondición	-

Tabla 3.6: Requisito funcional RF-06

Identificador	**RF-07**
Nombre	Crear Torneo
Descripción	Un usuario podrá crear un torneo.
Precondición	El usuario debe de iniciar sesión.
Postcondición	-

Tabla 3.7: Requisito funcional RF-07

Identificador	**RF-08**
Nombre	Editar Torneo
Descripción	Un usuario podrá actualizar los datos de un torneo.
Precondición	El usuario debe de iniciar sesión.
Postcondición	-

Tabla 3.8: Requisito funcional RF-08

Identificador	**RF-09**
Nombre	Reportar partidos como Administrador
Descripción	Un usuario que administre un torneo podrá reportar los partidos.
Precondición	El usuario debe de iniciar sesión y ser el administrador del torneo.
Postcondición	-

Tabla 3.9: Requisito funcional RF-09

Identificador	**RF-10**
Nombre	Participar en un torneo
Descripción	Un usuario podrá participar en un torneo.
Precondición	El usuario debe de iniciar sesión.
Postcondición	-

Tabla 3.10: Requisito funcional RF-10

Identificador	**RF-11**
Nombre	Crear un equipo
Descripción	Un usuario podrá crear su propio equipo.
Precondición	El usuario debe de iniciar sesión y no pertenecer a ningún otro equipo.
Postcondición	-

Tabla 3.11: Requisito funcional RF-11

Identificador	**RF-12**
Nombre	Editar información de equipo
Descripción	Un usuario que administre un equipo podrá modificar su información.
Precondición	El usuario debe de iniciar sesión y ser el administrador del equipo.
Postcondición	-

Tabla 3.12: Requisito funcional RF-12

Identificador	**RF-13**
Nombre	Eliminar equipo
Descripción	Un usuario que administre un equipo podrá eliminarlo.
Precondición	El usuario debe de iniciar sesión y ser el administrador del equipo.
Postcondición	-

Tabla 3.13: Requisito funcional RF-13

Identificador	**RF-14**
Nombre	Invitaciones de equipos
Descripción	Un usuario que administre un equipo podrá enviar invitaciones a otros usuarios en nombre del equipo.
Precondición	El usuario debe de iniciar sesión y ser el administrador del equipo.
Postcondición	-

Tabla 3.14: Requisito funcional RF-14

Identificador	**RF-15**
Nombre	Invitaciones a usuarios
Descripción	Los usuarios podrán aceptar o rechazar las invitaciones que los equipos les envíen.
Precondición	El usuario debe de iniciar sesión.
Postcondición	-

Tabla 3.15: Requisito funcional RF-15

Identificador	**RF-16**
Nombre	Participación de equipos en torneos
Descripción	Los usuarios que administren un equipo podrán registrar a su equipo en un torneo con modalidad de equipos.
Precondición	El usuario debe de iniciar sesión y administrar un equipo.
Postcondición	-

Tabla 3.16: Requisito funcional RF-16

Identificador	**RF-17**
Nombre	Reporte de partidos de equipo
Descripción	Los usuarios que administren un equipo y ese equipo participe en un torneo podrán reportar los resultados de los partidos jugados por el equipo.
Precondición	El usuario debe de iniciar sesión y que el equipo administrado participe en el torneo.
Postcondición	-

Tabla 3.17: Requisito funcional RF-17

Identificador	**RF-18**
Nombre	Visualizar listado de usuarios
Descripción	Un usuario podrá visualizar el listado de usuarios.
Precondición	-
Postcondición	-

Tabla 3.18: Requisito funcional RF18

Identificador	**RF-19**
Nombre	Buscar en el listado de usuarios
Descripción	Un usuario podrá buscar en el listado de usuarios.
Precondición	-
Postcondición	-

Tabla 3.19: Requisito funcional RF-19

Identificador	**RF-20**
Nombre	Visualizar el listado de equipos
Descripción	Un usuario podrá visualizar el listado de equipos.
Precondición	-
Postcondición	-

Tabla 3.20: Requisito funcional RF-20

Identificador	**RF-21**
Nombre	Buscar en el listado de equipos
Descripción	Un usuario podrá buscar en el listado de equipos.
Precondición	-
Postcondición	-

Tabla 3.21: Requisito funcional RF-21

3.3.2. Requisitos de Rendimiento

La plataforma web esta enfocada a que la usen un gran numero de usuarios concurrentemente por lo que se espera que el servidor sea capaz de procesar las peticiones en menos de 2s.

3.3.3. Restricciones de Diseño

El acceso a la plataforma web será desde distintos dispositivos como lo pueden ser ordenador de mesa, portátiles, móviles o tabletas por lo que el diseño deberá adaptarse a las distintas resoluciones de pantalla por lo que se aplicara el uso de diseño responsive.

3.3.4. Atributos del Sistema

El software debe de cumplir ciertos requisitos no funcionales los cuales se detallan en las tablas que se muestran a continuación.

Identificador	**RNF-01**
Nombre	Usabilidad
Descripción	La plataforma deberá de tener un tiempo de aprendizaje menor a 15 minutos.

Tabla 3.22: Requisito no funcional RNF-01

Identificador	**RNF-02**
Nombre	Usabilidad
Descripción	La plataforma se deberá de visualizar correctamente tanto en ordenadores personales, dispositivos móviles y tabletas.

Tabla 3.23: Requisito no funcional RNF-02

Identificador	**RNF-03**
Nombre	Eficiencia
Descripción	La plataforma deberá de responder las peticiones al servidor en menos de 2s.

Tabla 3.24: Requisito no funcional RNF-03

Identificador	**RNF-04**
Nombre	Disponibilidad
Descripción	Se espera que cuando un usuario acceda a la plataforma este disponible el 99.99 % de los accesos.

Tabla 3.25: Requisito no funcional RNF-04

Identificador	**RNF-05**
Nombre	Mantenibilidad
Descripción	Debido a necesidades perfectivas, evolutivas y correctivas el código fuente del software debe de ser mantenible.

Tabla 3.26: Requisito no funcional RNF-05

CAPÍTULO 4
Diseño de la solución

4.1 Introducción

En este capitulo se procederá a detallar la arquitectura del sistema y el trabajo de diseño realizado antes de comenzar a desarrollar el proyecto. En el apartado del diseño se detallaran los prototipos de las interfaces de la aplicación y el diseño de la base de datos.

4.2 Arquitectura del Sistema

La plataforma desarrollada en este proyecto es una aplicación web por lo que la arquitectura del sistema por lo que se ha optado ha sido Cliente/Servidor.

Por un lado tenemos al cliente el cual podrá hacer uso de distintos dispositivos como lo puede ser un ordenador de mesa, un portátil, una tableta o un teléfono móvil para poder acceder a la plataforma.

El cliente enviara peticiones al servidor el cual gestionara los datos y consultara o escribirá toda la información necesaria en la base de datos. Una vez que el servidor ha manejado la petición le devolverá al cliente una respuesta con la información requerida.

En la siguiente Figura 4.1 se muestra el diagrama de la arquitectura del sistema.

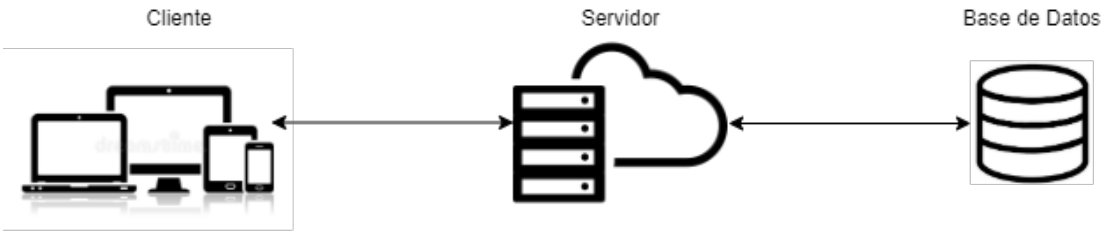

Figura 4.1: Arquitectura del sistema

4.3 Diseño Detallado

4.3.1. Diseño de la interfaz gráfica

Antes de realizar cualquier desarrollo de un producto software es necesario realizar un trabajo previo diseñando las interfaces que lo compondrán obteniendo una primera versión la cual nos ayudara a tener una visión mas clara del producto final.

En caso de incoherencias en el primer planteamiento tener una base clara sobre la que desarrollar minimizara los riesgos de los posibles cambios que pudiesen surgir.

Haciendo uso de la herramienta draw.io [3] se han realizado prototipos de interfaces de cada una de las vistas involucradas los cuales se detallaran a continuación.

Barra de navegación

En la parte superior de todas las vistas se encontrara el menú de navegación el cual nos permitirá tanto navegar por las distintas secciones de la plataforma como también nos mostrara los botones de cerrar sesión, registro e inicio de sesión.

Figura 4.2: Prototipo de la barra de navegación

Vista principal

La vista principal sera lo primero que los usuarios se encontrarán al entrar en la plataforma web, esta vista debe de mostrar el feed de posts de todo el resto de usuarios así como permitir escribir un nuevo post si el usuario esta logueado en la aplicación. Esta vista permitirá que la comunidad de jugadores interactué entre sí.

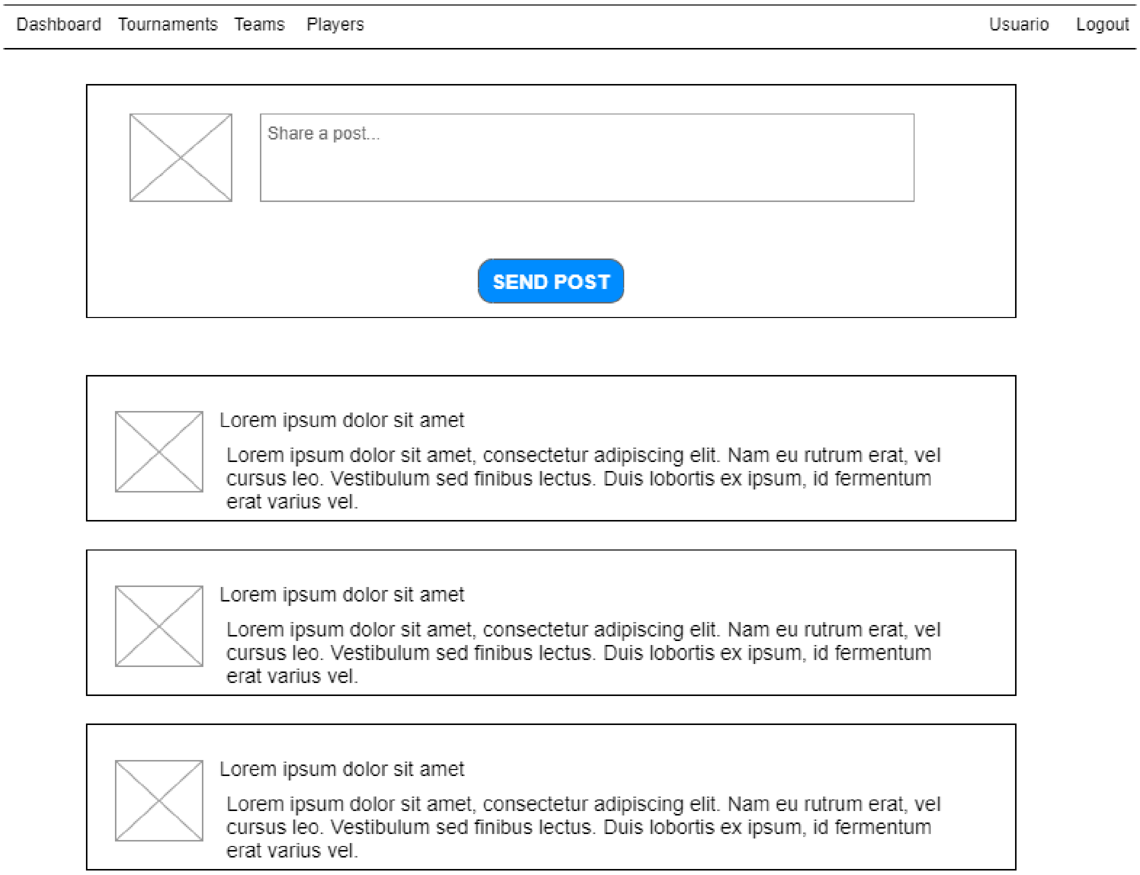

Figura 4.3: Prototipo de la vista principal

Vista Registro de usuario

Los usuarios deben de poder registrarse por lo que es necesario un formulario de registro, este formulario sera un dialogo que se mostrara al pulsar sobre el botón de registro de la barra de navegación.

En el formulario de registro el usuario introduciría sus datos y subirá a la plataforma su foto de perfil.

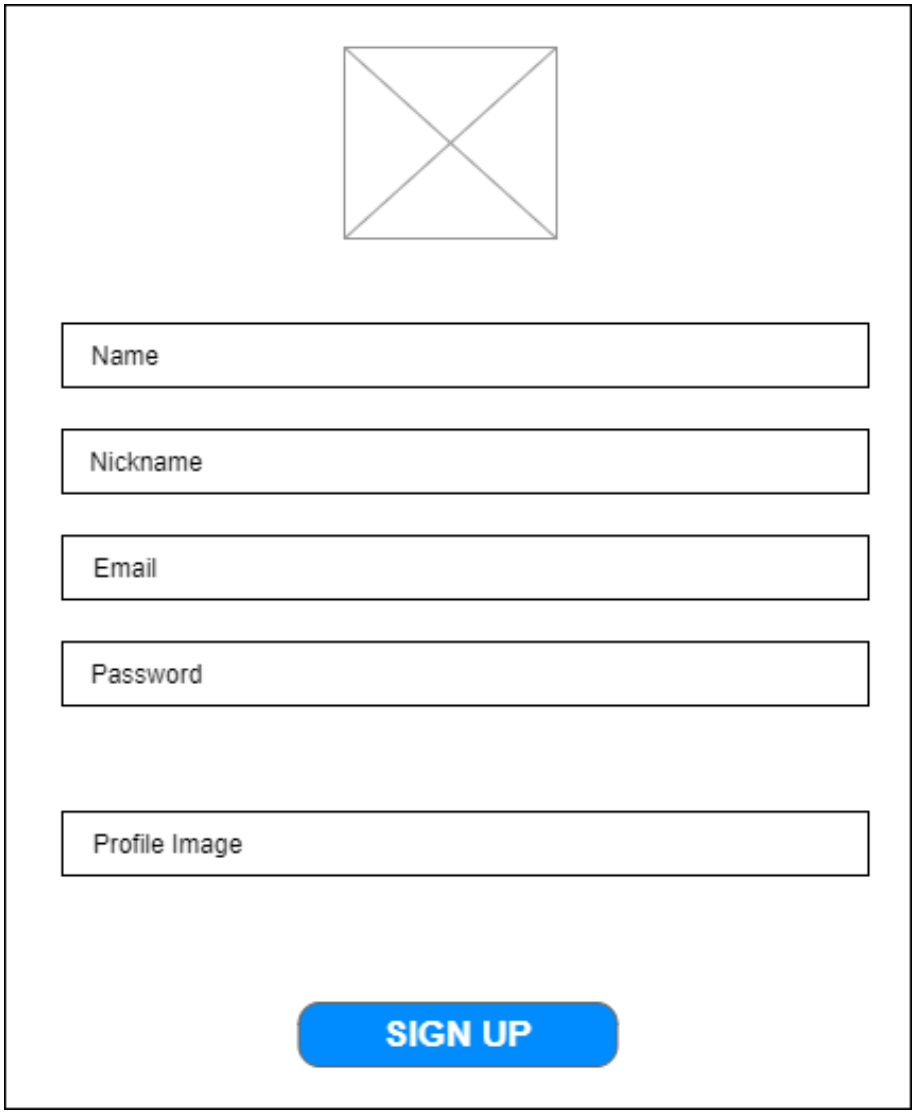

Figura 4.4: Prototipo de la vista de registro de usuario

Vista de Inicio de sesión de usuario

Los usuarios registrados necesitan un formulario de inicio de sesión, este formulario sera un dialogo que se mostrara al pulsar en el botón de login de la barra de navegación.

En el formulario de inicio de sesión el usuario introducirá su email y su contraseña para poder acceder a la plataforma.

Figura 4.5: Prototipo de la vista de inicio de sesión de usuario

Vista del perfil de Usuario

Los usuarios que estén registrados contaran con una pagina del perfil la cual mostrará su foto de perfil junto a su información, el equipo al que pertenezcan si es que pertenecen alguno y los posts que hayan enviado bajo su nombre.

En el caso de recibir invitaciones para formar parte de un equipo el usuario podrá aceptarlas o rechazarlas desde su perfil ya que le aparecerán las invitaciones que haya recibido. Estas invitaciones solo estarán accesibles al propio usuario si ha iniciado sesión en la plataforma.

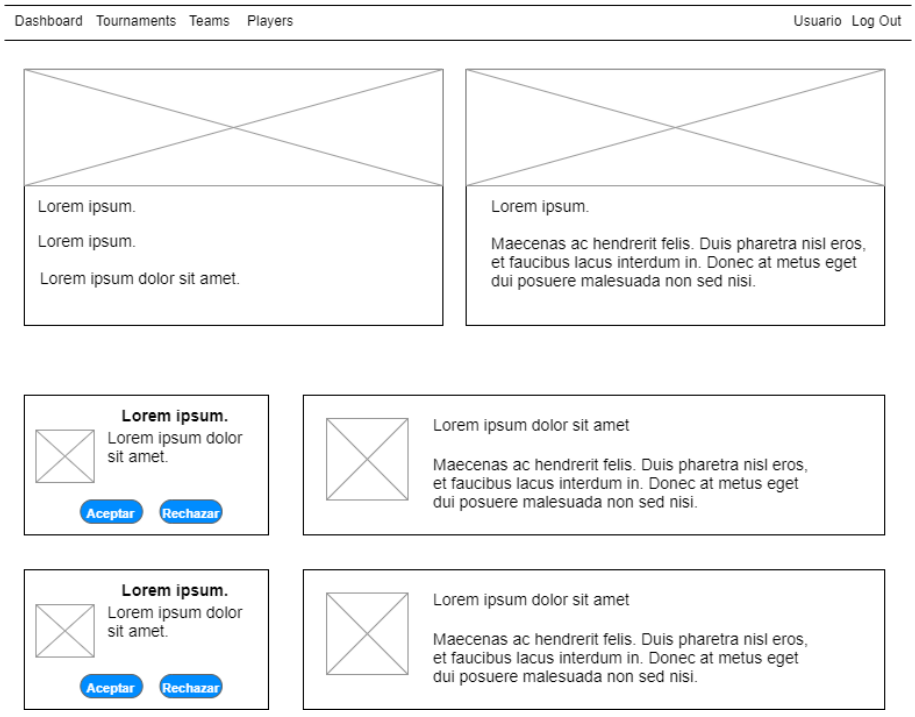

Figura 4.6: Prototipo de la vista del perfil de usuario

Vista de configuración

Toda aplicación web que gestione usuarios debe de contar con una pagina de configuración en concreto desde esta vista se necesita que el usuario pueda modificar sus datos y su foto de perfil.

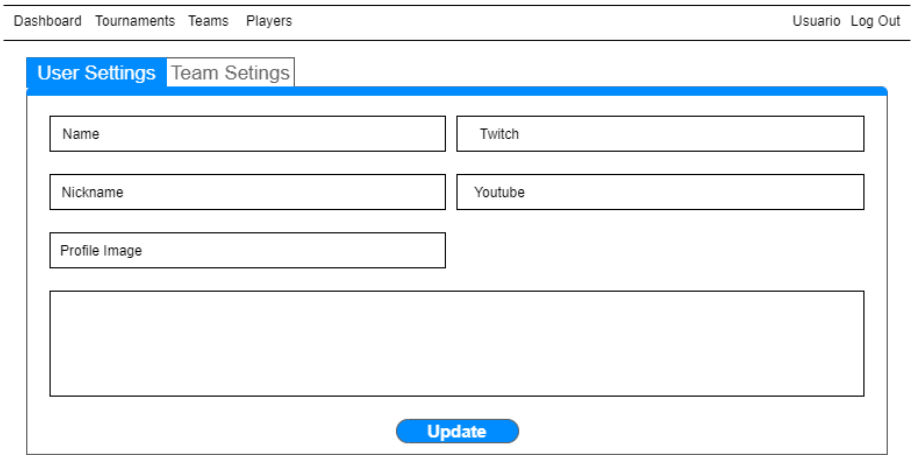

Figura 4.7: Prototipo de la vista de configuración de usuario

Por otro lado si el usuario es el dueño de un equipo desde esta vista sera capaz de modificar la información del equipo, en caso de no ser el dueño ni de pertenecer a ningún equipo desde esta vista el usuario podrá crear el suyo propio.

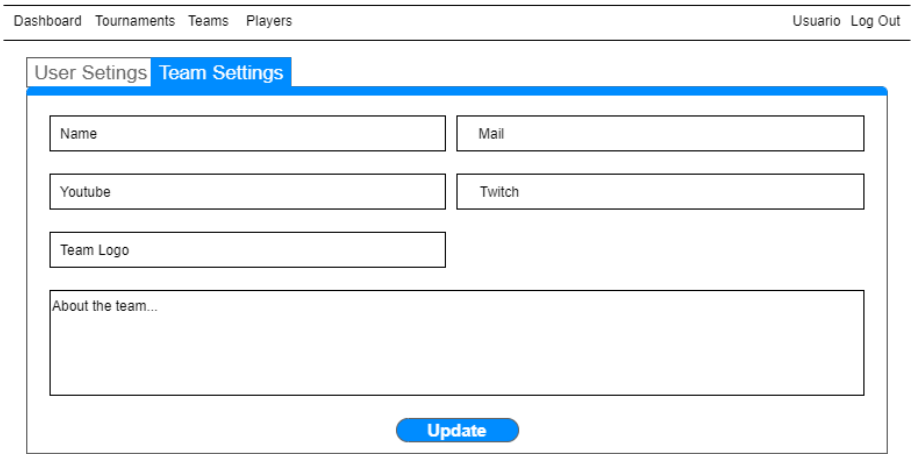

Figura 4.8: Prototipo de la vista de configuración del equipo

Vista del listado de torneos

En esta vista se listaran todos los torneos y se mostrara su información mas básica, además del listado de los torneos sera necesario contar con un buscador y con un botón que nos lleve hasta la vista del formulario de crear torneo.

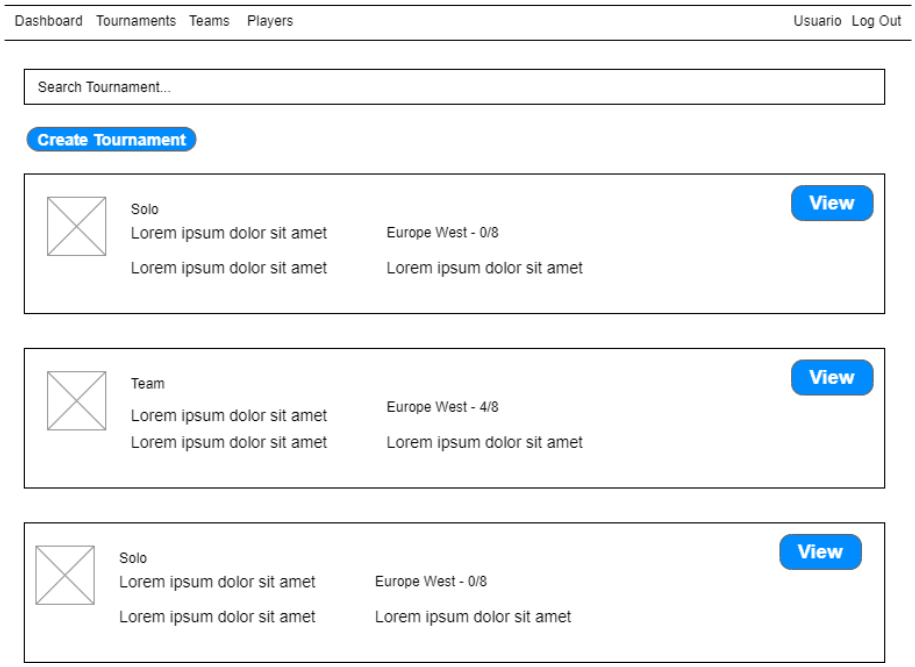

Figura 4.9: Prototipo de la vista del listado de torneos

Vista de la creación del torneo

En el formulario de creación de torneo se introducirá el nombre del torneo, la descripción, el juego del torneo, el tipo del torneo, la región y la fecha del torneo.

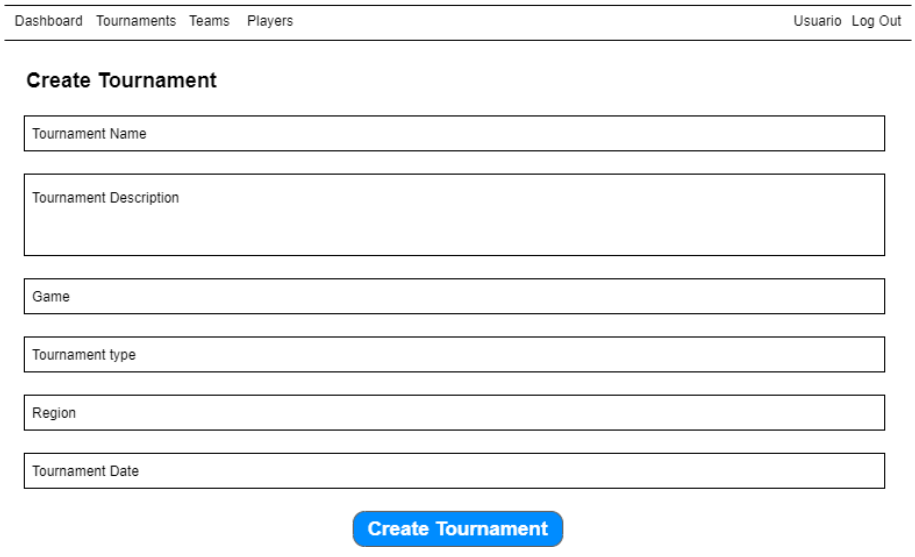

Figura 4.10: Prototipo de la creación del torneo

Vista de los torneos

Cada torneo debe de contar con su propia página la cual mostrara la tarjeta con la información básica, la descripción de torneo, un listado de los participantes y los partidos que se deben de jugar y el resultado.

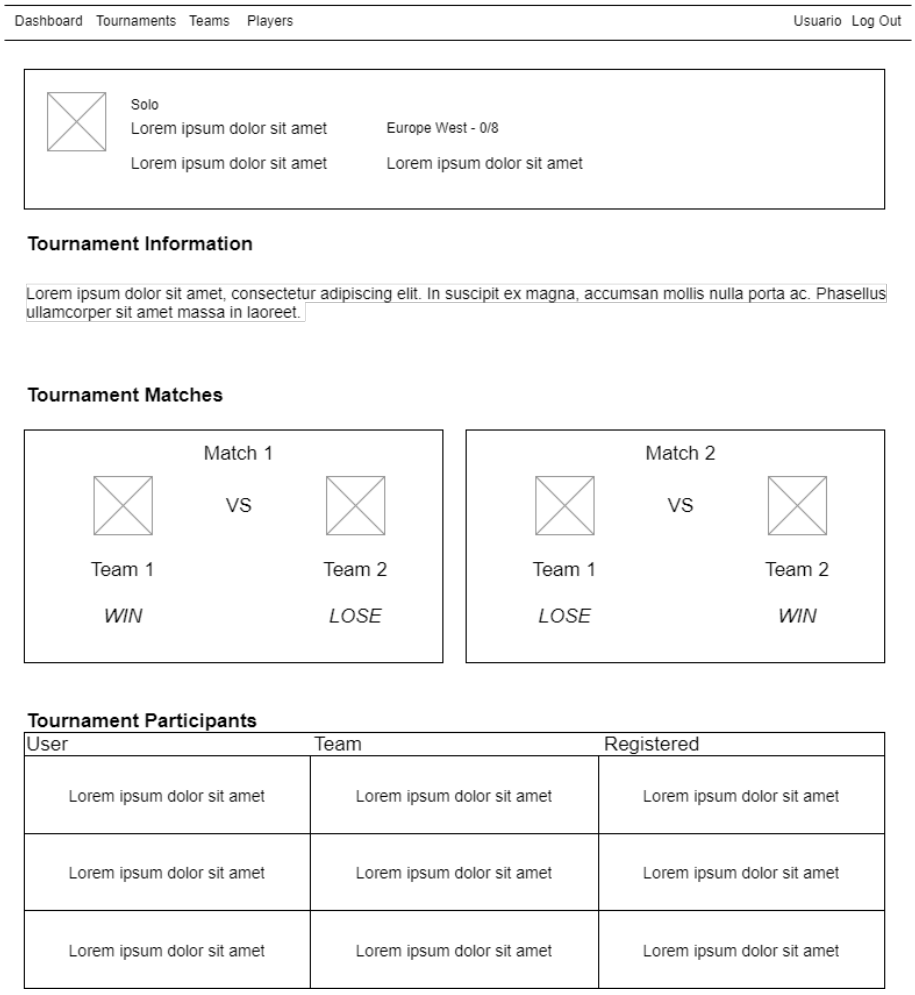

Figura 4.11: Prototipo de la vista de los torneos

Vista del listado de equipos

En esta vista se listaran todos los equipos, cada equipo sera una tarjeta la cual contendrá la imagen del equipo, el nombre y la descripción. Además del listado de los torneos contara también con un buscador.

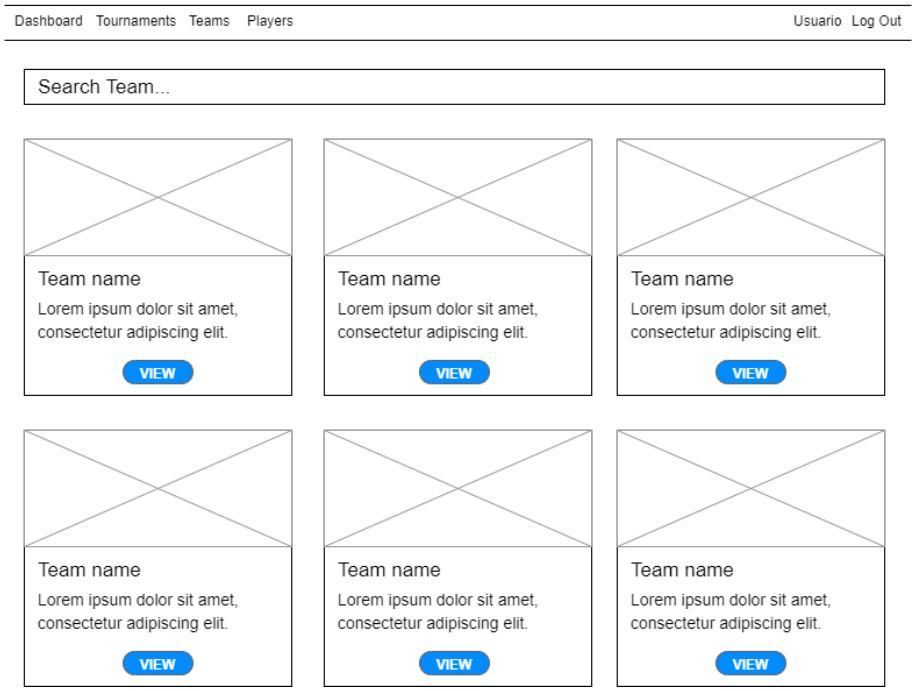

Figura 4.12: Prototipo de la vista del listado de equipos

Vista de los equipos

Cada equipo contara con un perfil el cual mostrara su información, los jugadores que forman parte del equipo y los torneos que ha ganado.

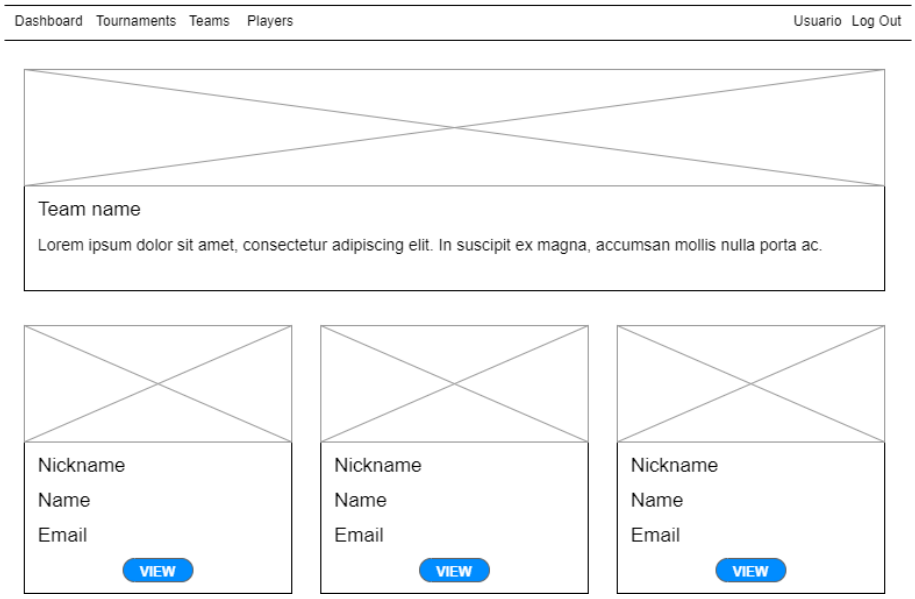

Figura 4.13: Prototipo de la vista de los equipos

Vista del listado de usuarios

En esta vista se listarán todos los usuarios, se mostrará el nombre, el nickname y el correo. Además del listado de los usuarios contará también con un buscador.

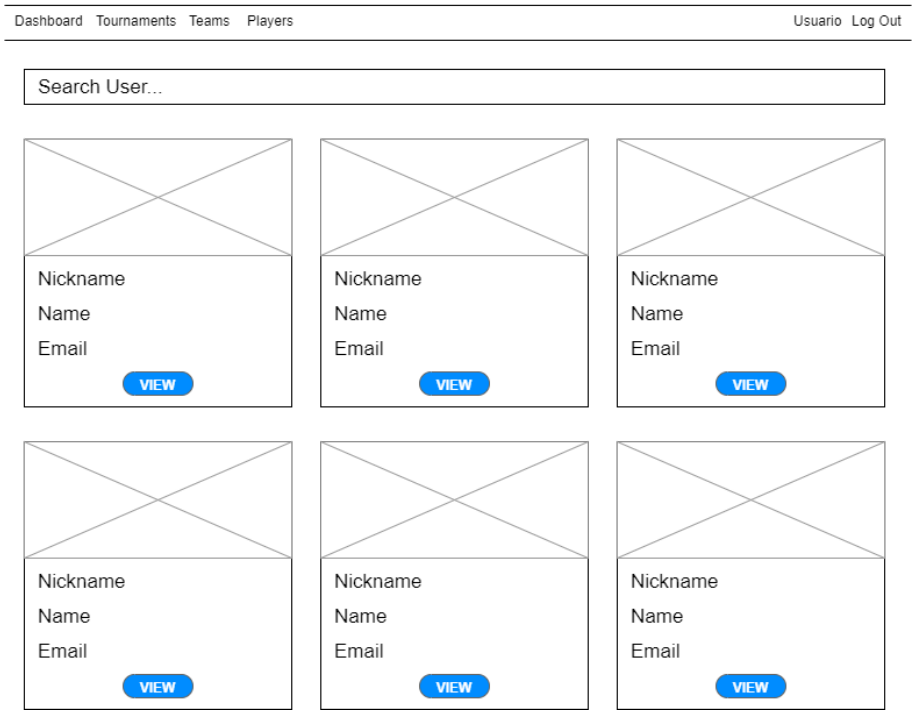

Figura 4.14: Prototipo de la vista de los equipos

4.3.2. Diseño de la Base de Datos

Analizar el esquema relacional de la base de datos es una etapa muy importante dentro del proceso del desarrollo del software ya que junto al prototipado de las interfaces nos permite asentar las bases del producto a desarrollar.

La aplicación web Gaamix requiere que el esquema relacional contenga 8 tablas las cuales se detallaran a continuación.

- Users: La tabla usuarios sera el eje central de la aplicación guardando la información de los usuarios, su contraseña y los tokens necesarios para poder hacer consultas a la API REST. También contara con una clave ajena a la tabla equipos ya que los jugadores pueden pertenecer a un equipo.
- Posts: Esta sera la tabla en la que los mensajes que posteen los usuarios se almacenan por lo que es necesario que contenga una clave ajena a la tabla usuarios.
- Teams: La tabla equipos almacenara toda su información como puede ser el nombre la descripción o su imagen del equipo.
- Invitations: Los equipos tendrán la posibilidad de enviar invitaciones a jugadores que no formen parte de ningún equipo, en esta tabla se almacenaran dichas invitaciones. Sera necesario que cuente con dos claves ajenas la primera hacia la tabla equipos y la segunda hacia la tabla usuarios.
- Tags: Esta tabla sera la encargada de almacenar la variedad de juegos sobre los que se puede crear un torneo, el que exista esta tabla en lugar de un enum en base de datos permitirá que un futuro se puedan añadir nuevos

juegos o incluso la posibilidad de sacar estadísticas del numero de torneos por juego.

- Tournaments: La tabla de los torneos sera también fundamental en el software ya que almacenara todos los datos de los torneos como pueden ser el nombre del torneo, la fecha, región o el tipo de torneo. Serán necesarias dos claves ajenas una hacia la tabla Tags la cual indicara el juego del torneo y otra hacia la tabla Users indicando el propietario del torneo.
- Participants: Debido a que un usuario puede participar en muchos torneos y un torneo tiene muchos usuarios es necesario contar con una tabla que indique que usuario participa en que torneo, esta tabla contara con 3 claves ajenas. La primera hacia la tabla Tournaments indicando a que torneo pertenece ese participante, la segunda si el participante es un equipo hacia la tabla equipos y la última si el participante es un usuario hacia la tabla usuarios.
- Matches: La tabla Matches guardara la información de los partidos que se jugaran en los torneos. Se almacenaran tanto los participantes como el resultado del partido. Esta tabla contara con tres claves ajenas la primera hacia el torneo que pertenece el partido, la segunda hacia participantes indicando el ganador y la tercera hacia participantes indicando el perdedor.

En la siguiente Figura 4.15 se detalla el diagrama Entidad/Relación del esquema relacional de la base de datos.

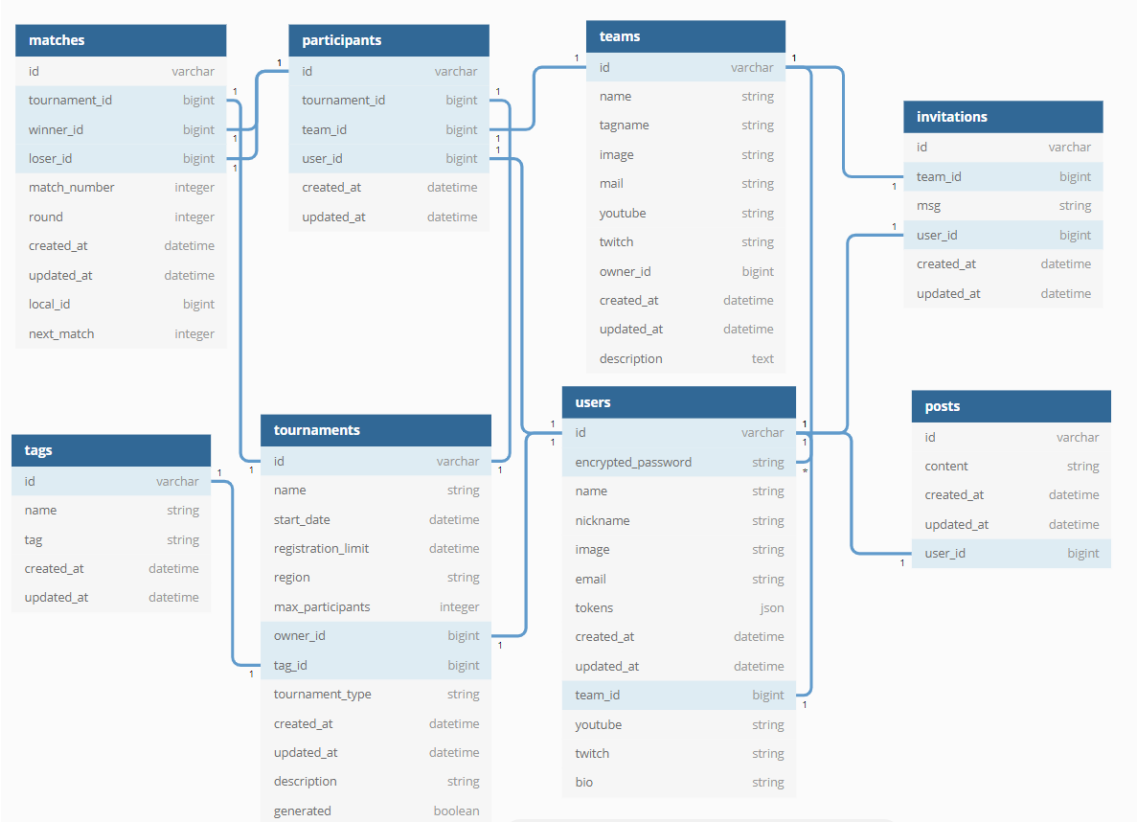

Figura 4.15: Diagrama Entidad/Relación de la base de datos

CAPÍTULO 5
Desarrollo de la solución propuesta

5.1 Introducción

En este capitulo se detallara el desarrollo de la solución, en concreto la fase de implementación del producto.

Esta fase se ha dividido en tres partes claramente diferenciadas la dockerización del entorno, implementación de la API REST en ruby on rails y la implementación del lado cliente con VueJs.

5.2 Implementación

5.2.1. Dockerización del entorno

En el desarrollo de cualquier producto software es necesario contar con un entorno de programación sobre el que comenzar a desarrollar el producto. Sera necesario instalar ciertas dependencias dependiendo de las tecnologías que se usen en el proyecto.

Esto puede causar problemas si el desarrollo del software se realiza desde varios equipos distintos o si existen varios desarrolladores involucrados en un mismo proyecto. Ya que si existen demasiadas dependencias puede que sea muy costoso dejar listo un equipo para comenzar a trabajar por conflictos con otros entornos de desarrollo de otros proyectos.

Por otro lado tener que instalar todas las dependencias del proyecto a mano también puede provocar que tanto la fase de despliegue a producción del producto o futuras migraciones a otros servidores aumenten los costos y los riesgos.

Debido a lo comentado anteriormente se ha optado por la virtualización del entorno de desarrollo permitiendo aislar las dependencias de cada tecnología dentro de su propio contenedor, disminuyendo casi al mínimo el tiempo de instalación del entorno o del despliegue a producción.

Para lograr la virtualización se ha usado Docker un software que nos permite crear contenedores los cuales alojaran las dependencias necesarias, una de las ventajas de docker es que optimiza al máximo los recursos del sistema ya que

aunque sean contenedores virtualizados se ejecutan sobre el mismo kernel del equipo que lo ejecuta sin tener que soportar un sistema operativo completo por cada contenedor.

En este proyecto se han desplegado tres contenedores distintos cada uno de ellos soporta un área principal de la aplicación.

- **Base de Datos**: Este contenedor alojara la base de datos PostgreSQL.

- **Lado cliente**: El contenedor del lado cliente es el que aloja el proyecto de frontend en VueJS.

- **Lado servidor**: Este contenedor aloja la API REST en Ruby on Rails y las dependencias necesarias. [29]

La estructura del proyecto se ha dividido en un directorio para cada contenedor en la siguiente Figura 5.5 se muestra la estructura del proyecto.

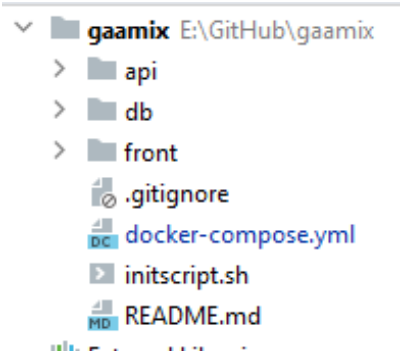

Figura 5.1: Estructura de directorios del proyecto

En el archivo docker-compose.yml Figura 5.2 se configura docker, se crean los contenedores que va a albergar el proyecto y se añaden los volúmenes que compartirán el equipo anfitrión y los contenedores.

```yaml
version: "3.3"
services:
  db:
    image: postgres
    restart: on-failure
    volumes:
      - ./db:/var/lib/postgresql
    environment:
      POSTGRES_USER: gaamix
      POSTGRES_PASSWORD: 123456
    ports:
        - "6543:5432"
  web:
    build: ./api
    container_name: ruby_container
    ports:
      - "3000:3000"
    volumes:
      - ./api/:/home/app
  front:
    build: ./front
    ports:
      - "8091:8080"
    volumes:
      - ./front:/front
      - /front/node_modules
```

Figura 5.2: docker-compose.yml

Cada contenedor cuenta con un archivo llamado Dockerfile en el cual se declara que tipo de contenedor es y los comandos que se ejecutaran cada vez que se inicien. En este archivo los comandos que se ejecutan son la instalación de las dependencias que se necesiten y el comando que inicia el servicio del contenedor.

A continuación se pueden ver los archivos Dockerfile del contenedor de Api Figura 5.3 y del contenedor del front Figura 5.4.

```
FROM ruby
WORKDIR /home/app
ENV PORT 3000
EXPOSE $PORT
RUN curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg |  apt-key add -
RUN echo "deb https://dl.yarnpkg.com/debian/ stable main" | tee /etc/apt/sources.list.d/yarn.list
RUN gem install rails bundler
RUN gem install rails
RUN apt-get update -qq && apt-get install -y nodejs
RUN apt-get install yarn
RUN echo "alias ll='ls -alF'" >> ~/.bashrc
RUN echo "alias la='ls -A'" >> ~/.bashrc
RUN echo "alias l='ls -CF'" >> ~/.bashrc
RUN echo "alias tailf='tail -f'" >> ~/.bashrc
CMD ["./initscript.sh"]
```

Figura 5.3: Dockerfile del contenedor API

```
FROM node:12.2.0-alpine
WORKDIR /front
ENV PATH /front/node_modules/.bin:$PATH
COPY package.json /front/package.json
RUN npm install
RUN npm install @vue/cli -g
CMD ["npm", "run", "serve"]
```

Figura 5.4: Dockerfile del contenedor Front

La dockerización permite que solo sea necesario instalar docker en el equipo para poder comenzar a desarrollar sobre la aplicación, tan solo se tendrán que levantar los contenedores con el comando:

```
$ docker-compose up
```

En la siguiente Figura 5.5 se muestra la arquitectura dockerizada.

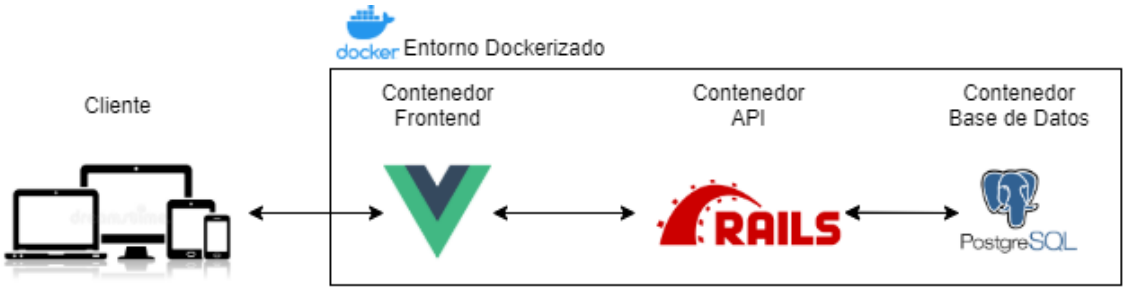

Figura 5.5: Arquitectura del sistema Dockerizada

5.2.2. Implementación de la Api Rest

En el lado del backend se ha optado por el desarrollo de una API (Application Programming Interface) en el framework Ruby on Rails el cual esta basado en el lenguaje de programación Ruby.

Las apis permiten proporcionar los datos de nuestra sistema siguiendo un conjunto de restricciones y protocolos, en este proyecto se ha desarrollado una API de tipo REST (Representational State Transfer).

Las API Rest hacen uso del protocolo HTTP para la comunicación entre las distintas partes intercambiando mensajes en formato XML o JSON. [31]

Algunas de las características de una API Rest son:

- Cuentan con una dirección URI única por cada recurso que sirva.
- Al no existir estado las consultas al servidor son únicas e independientes entre sí.
- Al generar una arquitectura débilmente acoplada el cliente no tiene el por que lanzar la petición directamente al servidor permitiendo contar con intermediarios o balanceadores de carga que aumenten la escalabilidad del software.
- Permiten que el cliente y servidor estén débilmente acoplados ya que no sera necesario que compartan el mismo tipo de estructura.
- Hace uso de los métodos HTTP para lanzar las consultas en la tabla 5.1 a continuación se pueden ver los distintos métodos disponibles.

La primera tarea realizada en la fase de implementación del backend ha sido crear la estructura de base de datos para ello se ha usado el sistema de migraciones de RubyOnRails.

Método HTTP	Acción que realiza
GET	Operación de lectura del recurso
PUT	Operación de actualización del recurso
DELETE	Operación de borrado del recurso
POST	Operación de creación del recurso y operaciones que no encajen en leer, actualizar y borrar.

Tabla 5.1: Métodos HTTP

El sistema de migraciones nos permite gestionar la creación y modificación de la estructura de la base de datos en caso de querer modificar alguna migración realizada se podría realizar rollback a un estado anterior.

Una vez que la estructura de la base de datos esta preparada se crean los controladores que serán los encargados de manejar las peticiones a la API. Los controladores serán creados bajo un directorio llamado V1 lo que nos dará la posibilidad de en un futuro añadir cambios a la API generando una nueva versión V2, de esta manera podrá desarrollarse los nuevos cambios sin afectar el uso de la primera versión.

En la siguiente Figura 5.6 se detalla uno de los controladores.

```ruby
class V1::UsersController < ApplicationController
  before_action :set_user, only: %i[show update destroy]

  def index
    search = params[:search]
    if search == ''
      @users = User.all
    else
      @users = User.where("name ilike ?", "%#{search}%").or(User.where("nickname ilike ?", "%#{search}%"))
      @users = @users.order(name: :asc)
    end
  end

  def show
    user_id = params[:id]
    @invitations = Invitation.where(user_id: user_id)
  end

  def update
    if @user.update(user_params)
      render json: @user
    else
      render json: @user.errors, status: :unprocessable_entity
    end
  end

  private

  def set_user
    @user = User.find(params[:id])
  end

  def user_params
    params.permit(:name, :nickname, :password, :password_confirmation, :twitch, :youtube, :bio, :team_id, :image, :search)
  end
end
```

Figura 5.6: Controlador de la clase User

También sera necesario que en los modelos de las entidades se añadan las relaciones con el resto de entidades.

A continuación en la Figura 5.7 se detalla el modelo de la clase User en la cual se declara el uso de la gema Devise, para la gestión del registro e inicio de sesión de los usuarios, y dos relaciones donde se indica que un usuario tiene muchos posts y que puede pertenecer a un equipo.

```ruby
# frozen_string_literal: true

class User < ActiveRecord::Base
  has_many :posts
  belongs_to :team, optional: true
  extend Devise::Models
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :trackable, :validatable
  include DeviseTokenAuth::Concerns::User
end
```

Figura 5.7: Modelo de la clase User

Las API Rest se basan en peticiones en formato URI para ello debemos declarar en el archivo routes.rb que controlador y que método será el encargado de manejar la petición.

En el archivo routes.rb el cual podemos ver en la siguiente Figura 5.8 además de las URIS también se declarara el espacio de trabajo de los controladores que ira ligado a la versión de la API. En un futuro en caso de que la API necesite grandes cambios se debería de crear un nuevo espacio de trabajo V2 para que no afecte las peticiones ya desarrolladas.

```ruby
Rails.application.routes.draw do
  # For details on the DSL available within this file, see https://guides.rubyonrails.org/routing.html
  namespace :v1, defaults: { format: 'json' } do
    resources :posts
    resources :users
    resources :teams
    resources :tournaments
    resources :invitations
    resources :participants
    resources :matches
    get 'teams/:id/users', action: :team_users, controller: 'teams'
    get 'tournaments/:id/exists/:user_id', action: :exists_user, controller: 'tournaments'
    get 'tournaments/:id/existsteam/:team_id', action: :exists_team, controller: 'tournaments'
    post 'tournaments/:id/generate', action: :generate_tournament, controller: 'tournaments'
    delete 'invitations/user/:id', action: :delete_invitations, controller: 'invitations'
    mount_devise_token_auth_for 'User', at: 'auth'
  end
end
```

Figura 5.8: Archivo routes.rb

Una vez terminada la implementación de la API Rest se obtiene una batería de URIS con las que se podrá gestionar la base de datos enviando peticiones desde el lado frontend.

Cada uno de los modelos de la aplicación cuenta como mínimo con las 5 peticiones básicas. Las cuales serán la actualización, inserción y borrado de un registro y la lectura de todos los registros o la lectura de un registro en concreto.

Los usuarios cuentan con dos tipos de URI la primera será del tipo

"/v1/users/" la cual servirá para devolver los usuarios o actualizarlos y la segunda seguirá el patrón "/v1/auth" que sera utilizada en el proceso de registro, inicio y cierre de sesión.

En la siguiente tabla 5.2 a modo de ejemplo se detallaran algunas de las URIS involucradas en la gestión de los usuarios.

Método HTTP	**URI**	**Acción que realiza**
GET	/v1/users	Devuelve el listado completo de usuarios
GET	/v1/users/:id	Devuelve el usuario con el identificador de la URI
PUT	/v1/users/:id	Actualiza el usuario del identificador de la URI con los parámetros recibidos.
DELETE	/v1/users/:id	Elimina el usuario con el identificador de la URI
POST	/v1/auth/sign_up	Registra un usuario con los parámetros recibidos
POST	/v1/auth/sign_in	Inicia sesión con el email y password recibidos en los parámetros
POST	/v1/auth/sign_out	Cierra sesión del usuario con email y password recibidos en los parámetros

Tabla 5.2: Ejemplos de URIS de la API Rest

A continuación en la Figura 5.9 se muestra la estructura del directorio del lado servidor del proyecto.

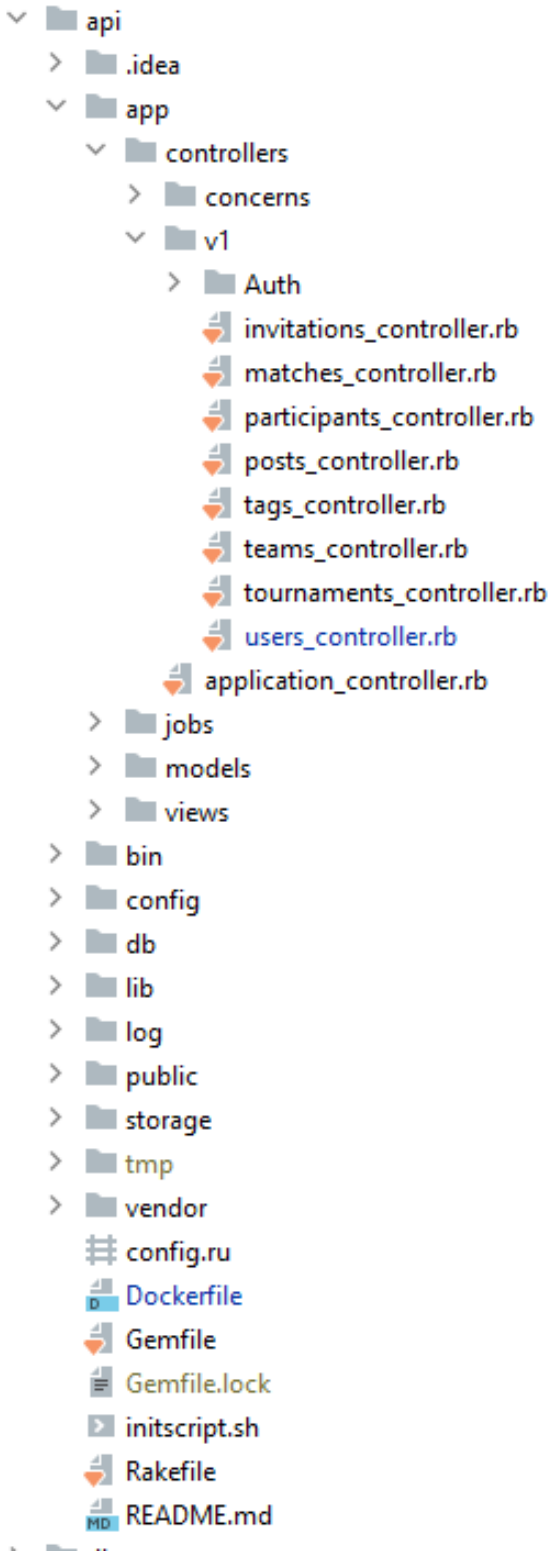

Figura 5.9: Estructura del directorio del lado servidor

5.2.3. Implementación del frontend

En los últimos años han surgido nuevos frameworks dentro del desarrollo web enfocado al lado cliente algunos de estos frameworks pueden ser Angular,

React o VueJs. Estos tres frameworks están en la cima de los frameworks de frontend mas usados en 2019 [20].

Los tres frameworks tienen en común que están basados en el lenguaje de programación javascript y que se basan en el uso de componentes. [1]

El desarrollo basado en componentes consiste en dividir las interfaces web en varios elementos, cada uno de estos elementos contendrá el HTML, CSS y JavaScript necesario en un mismo fichero separado del resto. Gracias al encapsulamiento de cada uno de los componentes permite que el código sea altamente reutilizable.

En la implementación del frontend de este proyecto se ha usado el Framework Vue js junto a algunas librerías que se detallaran a lo largo de este capítulo.

Vue Axios

Ha sido necesario poder enviar peticiones a la API Rest, para añadir esta funcionalidad al proyecto se ha echo uso del plugin Vue Axios el cual permite lanzar solicitudes HTTP.

Para ello se debe de configurar la URL de la API Rest a consultar y los headers personalizados que la Api requiera, en la siguiente Figura 5.10 se detalla el archivo de configuración de Vue Axios.

```
1  import axios from 'axios';
2  const API_URL = "http://localhost:3000/v1/"
3
4  export default axios.create({
5      baseURL: API_URL,
6      headers: {
7          'Content-Type': 'application/json',
8          'access-token': localStorage.getItem( key: 'token'),
9          'client': localStorage.getItem( key: 'client'),
10         'uid': localStorage.getItem( key: 'uid')
11     }
12 })
```

Figura 5.10: Configuración del plugin Vue Axios

En la siguiente Figura 5.11 se puede apreciar un ejemplo de consulta a la API Rest haciendo uso del plugin Vue Axios.

```
this.$http.post(
    'teams/',
    {
        name:   this.name ,
        tagname:this.tagname,
        image:  this.image,
        mail:   this.mail,
        youtube:this.youtube,
        twitch: this.twitch,
        description: this.description,
        owner_id: this.user_id
    }
).then(resp => {
    this.editUser(resp.data.id);
})
.catch(err => {
    console.log(err)
})
```

Figura 5.11: Ejemplo de petición a la Api con Vue Axios

Vue Router

Las aplicaciones de tipo SPA (Single Page Application) necesitan hacer uso de un sistema de rutas el cual indique las rutas que se usaran en la aplicación, en cada una de estas rutas se configurara un componente vista.

Cuando los usuarios naveguen e interactúen con la aplicación variaran de ruta y de vistas pero el navegador no recargara la página en ningún momento. Cada una de las rutas tiene un componente de tipo vista asociado el cual se cargara cuando esa ruta sea solicitada por el navegador.

En la siguiente Figura 5.13 se muestran algunas de las rutas creadas en la aplicación desarrollada en este proyecto.

```
import Vue from 'vue'
import VueRouter from 'vue-router'
import Home from '../views/Home.vue'
import Profile from '../views/Profile.vue'
import ProfileSettings from '../views/ProfileSettings.vue'
import Players from "../views/Players";

Vue.use(VueRouter)
  const routes = [
  {
    path: '/',
    name: 'Home',
    component: Home,
    meta: {
      title: 'Home'
    }
  },
  {
    path: '/players',
    name: 'Players',
    component: Players,
    meta: {
      title: 'Players'
    }
  },
  {
    path: '/profile/:id',
    name: 'Profile',
    component: Profile,
    meta: {
      title: 'Profile'
    }
  },
  {
    path: '/profile/:id/settings',
    name: 'ProfileSettings',
    component: ProfileSettings,
    meta: {
      title: 'ProfileSettings'
    }
  },
]

const router = new VueRouter({
  mode: 'history',
  base: process.env.BASE_URL,
  routes
})
export default router
```

Figura 5.12: Declaración de rutas de la aplicación

Vuetify

Vuetify es un framework para Vue Js que se usa como herramienta para maquetar interfaces siguiendo el patrón de diseño Material Design se consiguen interfaces con un diseño atractivo y muy visual. La herramienta cuenta con una gran cantidad de componentes básicos para ser usados en los componentes personalizados que han sido desarrollados para la aplicación.

Una de las restricciones de la aplicación era que el diseño de la plataforma se adapte a distintas resoluciones de pantalla permitiendo su uso en distintos dispositivos, para ello se ha usado el sistema de rejilla de Vuetify.

El sistema de rejilla permite que el desarrollador adapte un mismo diseño en distintos dispositivos. Haciendo uso de la directiva “<v-row>"para crear una fila

y la directiva “<v-col>"para crear una columna se va dando forma a la estructura de la vista. En la directiva “<v-col>"se especifica el numero de columnas que ocupara según la resolución detectada por el navegador.

Como ejemplo del sistema de rejilla se usara el componente de la vista del listado de usuarios en la cual se especifica que en caso de ser una resolución de escritorio larga cada tarjeta de usuario ocupara 4 columnas, en caso de ser una resolución de escritorio media o una tableta cada tarjeta de usuario ocupara 6 columnas y en el caso de ser un dispositivo móvil cada tarjeta de usuario ocupara las 12 columnas totales.

```
<template>
    <v-container class="wraper">
        <v-row class="d-flex justify-center">
            <v-col cols="12" sm="6">
                <searchBar
                    @updateSearch="getUsers"
                >
                </searchBar>
            </v-col>
        </v-row>
        <v-row
            class="d-flex justify-center"
            v-if="users.length > 0"
        >
            <v-col
                cols="12" sm="6" md="4" lg="3"
                v-for="user in users"
                :key="user.id"
            >
                <profile-card
                    :user="user"
                    showView="true"
                ></profile-card>
            </v-col>
        </v-row>
        <v-row v-else>
            <v-col cols="12">
                <div
                    class="text-h4"
                >
                    No players found...
                </div>
            </v-col>
        </v-row>
    </v-container>
</template>
```

Figura 5.13: Maquetado del componente listado de usuarios

En la siguiente Figura 5.14 se muestra la vista listado de usuarios mostrando cuatro tarjetas de usuario por cada fila debido a que la resolución del navegador es la de un escritorio largo.

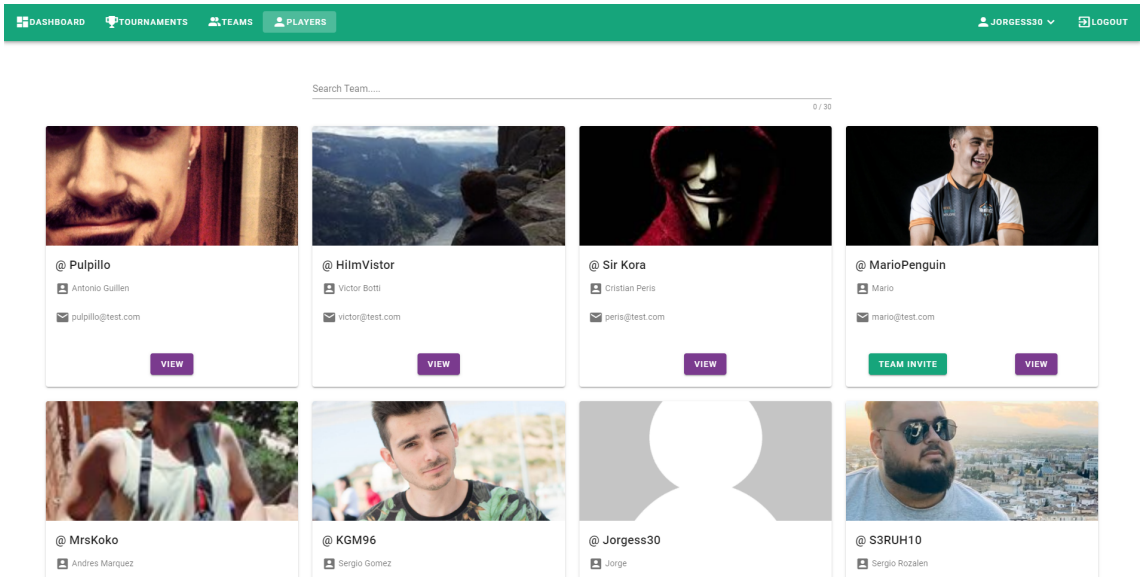

Figura 5.14: Vista listado de usuarios en escritorio largo

En la siguiente Figura 5.15 se muestra la vista listado de usuarios desde una tableta la cual muestra dos tarjetas de usuario por cada fila.

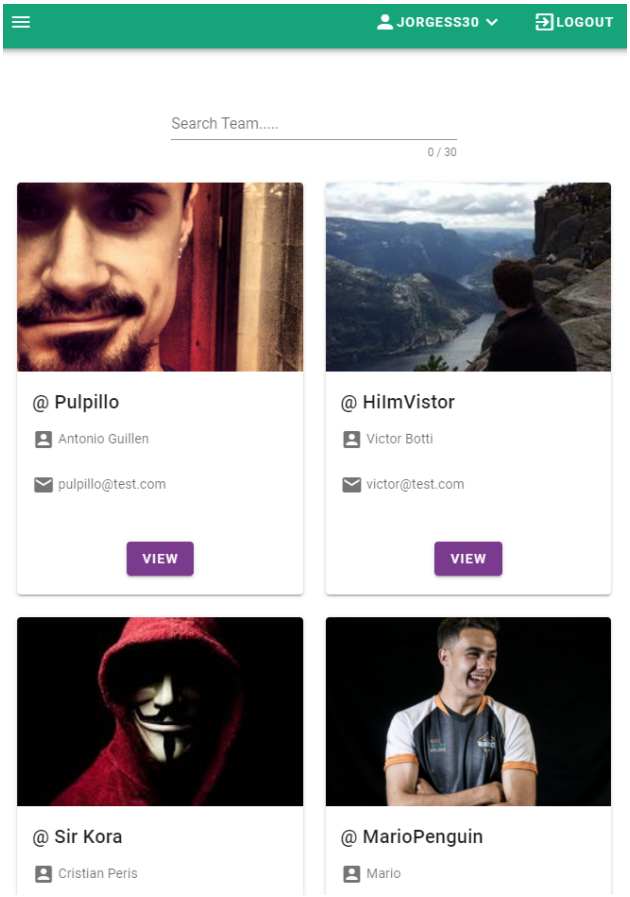

Figura 5.15: Vista listado de usuarios en tableta

En la Figura 5.16 se muestra la vista listado de usuarios desde un dispositivo móvil la cual muestra una tarjeta de usuario por cada fila.

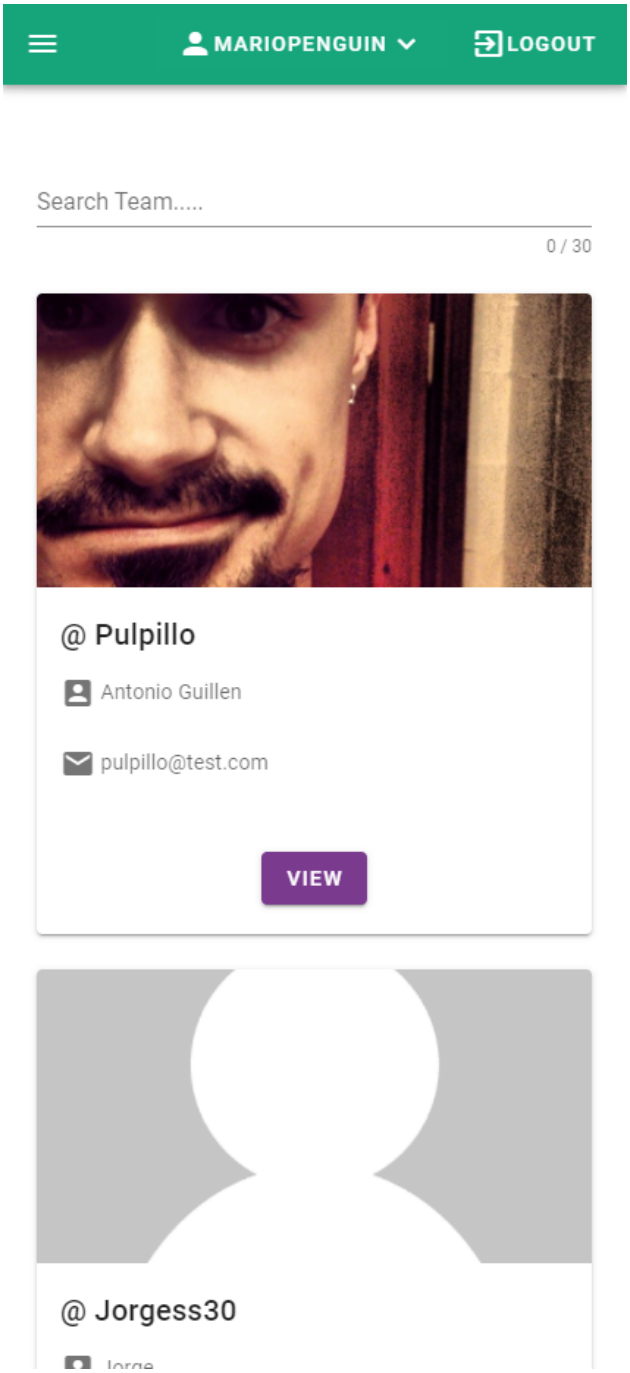

Figura 5.16: Vista listado de usuarios en dispositivo móvil

En total se han implementado diecisiete componentes que han sido usados en nueve vistas distintas. En la siguiente Figura 5.17 se muestra el directorio del lado cliente mostrando los componentes y vistas implementados.

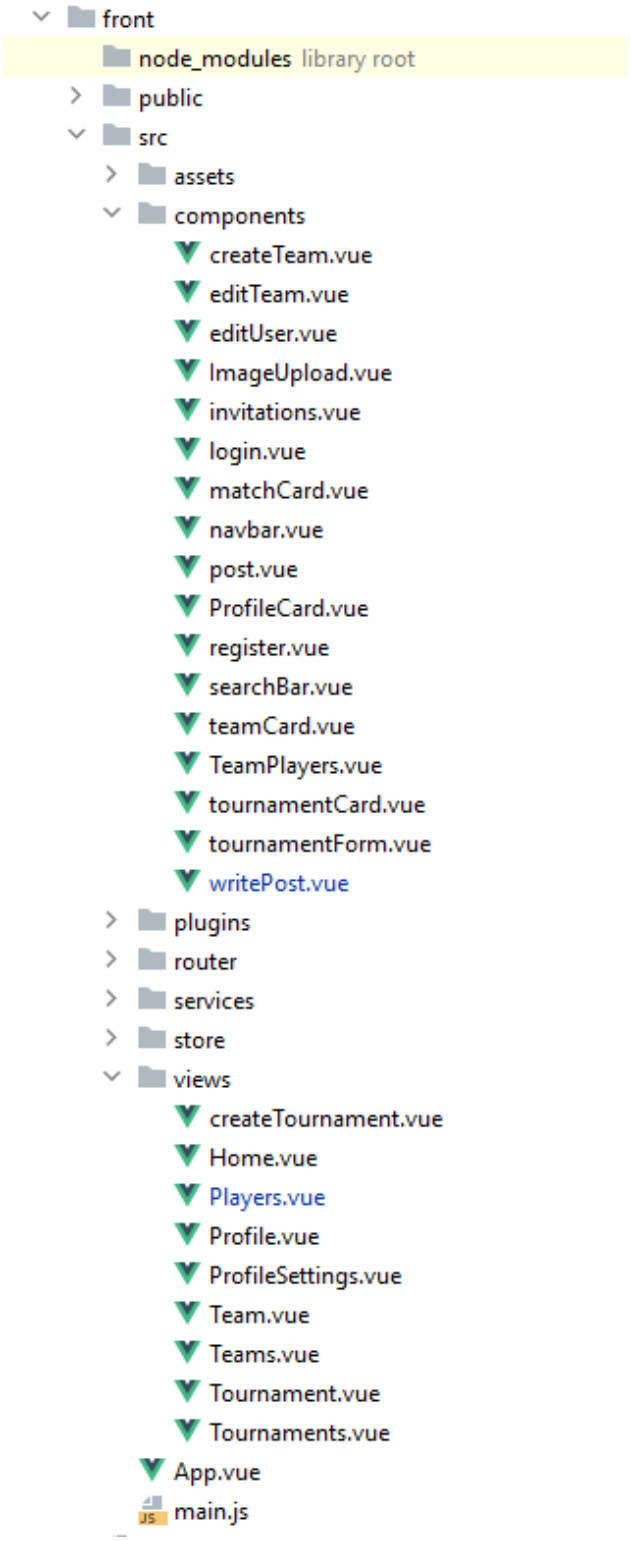

Figura 5.17: Estructura del directorio del lado cliente

CAPÍTULO 6
Implantación

En este capitulo se explica el proceso necesario para implantar la aplicación bajo un sistema con el sistema operativo Linux Mint.

El primer paso necesario para desplegar el proyecto es tener en la maquina en la cual se desplegara el código. En este proyecto se ha echo uso de Git como control de versiones y de la plataforma Git Hub para alojar el repositorio del proyecto.

Para instalar Git en el sistema ejecutaremos el siguiente comando:

```
$ sudo apt-get install git
```

Debido a que el repositorio que se ha usado para alojar el proyecto es Privado es necesario añadir la llave ssh del sistema en la cuenta de Git Hub.

Generaremos la llave ssh del sistema con el siguiente comando:

```
$ ssh-keygen -t rsa
```

Nos generara la llave publica en el archivo '/home/User/.ssh/id_rsa.pub' la cual añadiremos en la cuenta de Git hub. Una vez añadida la llave ssh en la configuración de la cuenta en Git Hub podemos descargar el repositorio del proyecto con el siguiente comando:

```
$ git clone git@github.com:rozalen/gaamix.git
```

Teniendo el repositorio del proyecto en el sistema lo siguiente que se necesita es tener Docker instalado para ello seguiremos los pasos detallados a continuación:

Instalaremos los paquetes necesarios con el siguiente comando:

```
$ sudo apt-get -y install apt-transport-https ca-certificates curl
  software-properties-common
```

Para poder descargar Docker deberemos de añadir el repositorio necesario a linux mint y actualizar para ello ejecutaremos los siguientes comandos:

```
$ sudo apt-add-repository 'deb https://apt.dockerproject.org/repo
  ubuntu-xenial main'
$ sudo apt-get update
```

Teniendo la lista de repositorios actualizada con el repositorio de Docker deben de instalarse varios paquetes para descargarlos se debe de ejecutar el siguiente comando:

```
$ sudo apt-get install -y docker docker.io docker-compose
```

Una vez instalado Docker comprobaremos que el proceso esta levantado con el comando:

```
$ service docker status
```

En la siguiente Figura 6.1 veremos el resultado del comando anterior mostrando que el servicio de Docker esta activo.

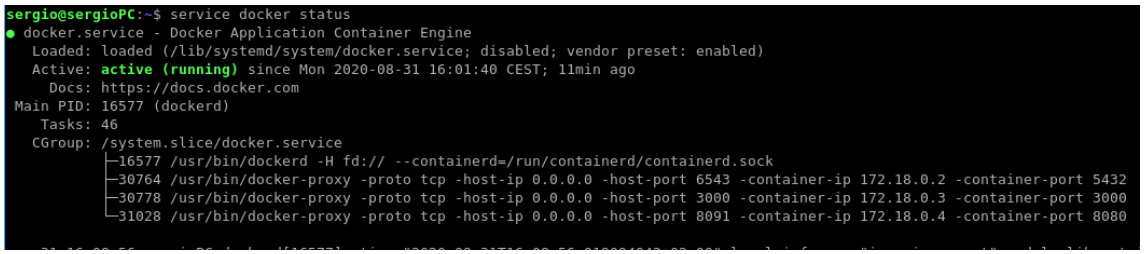

Figura 6.1: Estado del servicio Docker

El siguiente paso será arrancar los contenedores para ello desde el directorio del proyecto se ejecutara el siguiente comando para arrancarlos:

```
$ sudo docker-compose up
```

Teniendo los contenedores arrancados nos conectaremos al contenedor de la API para crear la base de datos desde el framework RubyOnRails para ello es necesario listar los contenedores en ejecución localizar el contenedor de la API y conectarnos para ejecutar algunos comandos de rails.

En la siguiente Figura 6.2 se muestra el comando necesario para listar los contenedores en ejecución.

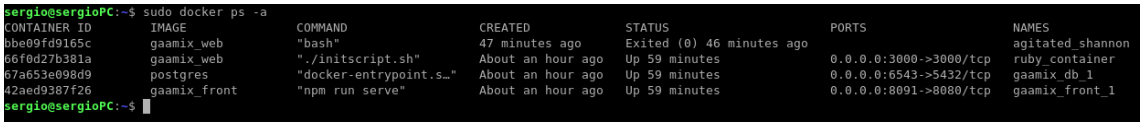

Figura 6.2: Listado de contenedores en ejecución

Localizamos el Id del contenedor de la API y nos conectamos a el mediante el comando:

```
$ sudo docker exec -it ID del Contenedor bash
```

Dentro del contenedor deberemos de ejecutar los siguientes comandos para inicializar la base de datos, ejecutar las migraciones y se ha añadido también la ejecución de una semilla de datos para poblar la base de datos con ejemplos desde un inicio.

```
$ rails db:create
$ rails db:migrate
$ rails db:seed
```

En la Figura 6.3 se muestra el dashboard de la aplicación tras desplegarla.

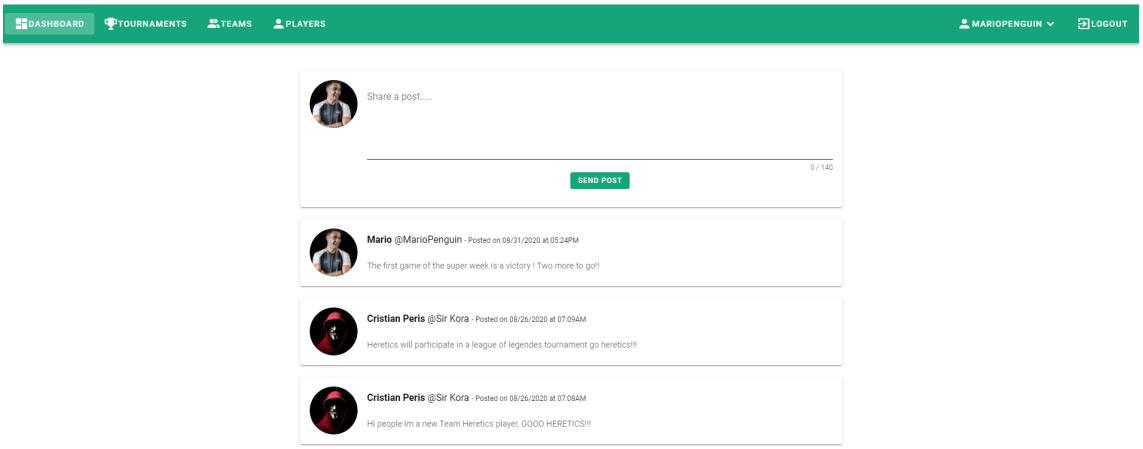

Figura 6.3: Dashboard de Gaamix tras desplegar la aplicación

CAPÍTULO 7
Pruebas

La fase de pruebas es una fase fundamental en el desarrollo de software ya que nos ayudan a detectar errores para ser solucionados antes de que el producto sea lanzado a producción.

En este capítulo se detallaran las pruebas realizadas en el proceso de desarrollo del proyecto.

Al terminar el desarrollo de la API REST se ha probado cada una de las peticiones desarrolladas comprobando que la petición funciona y que nos devuelve el resultado esperado.

Estas pruebas a la Api se han realizado con la herramienta Postman [11] la cual permite realizar peticiones a una API REST de manera sencilla teniendo la posibilidad de formar la petición con los parámetros y cabeceras necesarias desde un formulario.

En la siguiente Figura 7.1 se muestra una prueba realizada a la Api REST haciendo uso de la plataforma Postman.

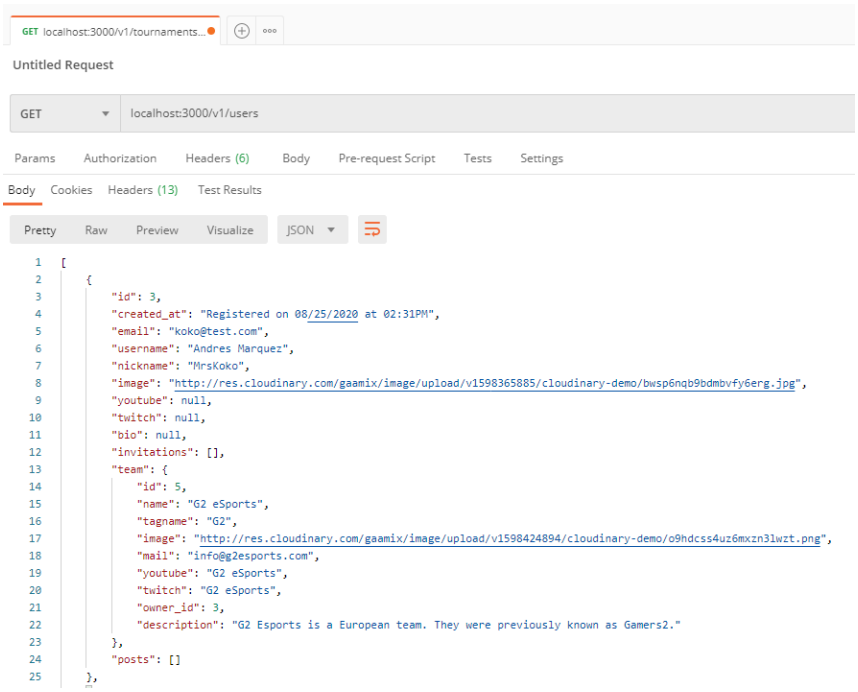

Figura 7.1: Prueba de petición a la API REST con Postman

La aplicación requería que el diseño se adaptara al dispositivo desde el cual se accedía, para ello una vez finalizada la maquetación del frontend se han realizado pruebas navegando por toda la web usando la herramienta Responsively [9] la cual nos permite testear el diseño en distintos dispositivos al mismo tiempo.

En la siguiente Figura 7.2 se muestra el uso de la herramienta Responsively.

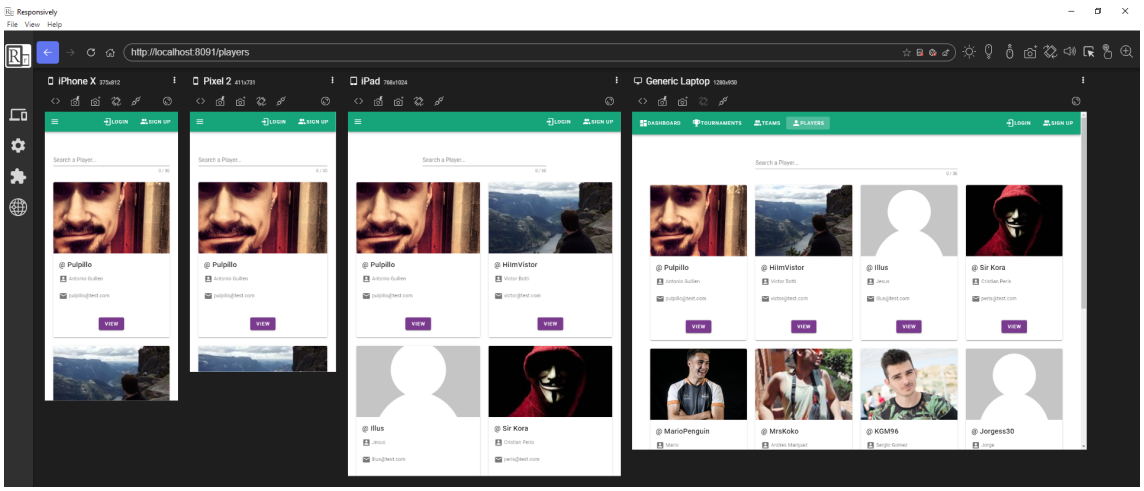

Figura 7.2: Testing del diseño haciendo uso de la herramienta Responsively

Se han testeado todas las vistas de la aplicación desde la herramienta Responsively desde la visión de cuatro dispositivos distintos.

- Iphone X para testear el diseño en dispositivos móviles donde la resolución es más alargada y hacen uso del sistema operativo móvil IOS.
- Pixel 2 para testear el diseño en dispositivos móviles que usen el sistema operativo Android.
- Ipad para testear el diseño en tabletas o en móviles con una gran resolución.
- Escritorio genérico para testear el diseño desde navegadores de ordenadores portátiles y sobremesas.

CAPÍTULO 8
Conclusiones

El desarrollo de una aplicación web que permita a los usuarios participar en torneos creados por ellos mismos y además comunicarse con el resto de la comunidad se ha cumplido satisfactoriamente.

A pesar de que la plataforma desarrollada no cuente con todas las funcionalidades que estaban planificadas a priori se han alcanzado todos los objetivos marcados.

Se ha aprendido a usar la herramienta Git como control de versiones del proyecto gestionando el repositorio con varias ramas.

Se ha diseñado una aplicación web desde cero, pasando una idea a un producto software, analizando los requisitos tanto funcionales como no funcionales, las restricciones necesarias y analizando el esquema relacional necesario para desarrollar el software.

Se ha realizado un análisis previo de las vistas necesarias de la aplicación y se han elaborado prototipos de las interfaces.

Se ha aprendido a usar la herramienta Docker para automatizar el despliegue de aplicaciones usándola para dockerizar el entorno de desarrollo y agilizar el despliegue en producción.

El hacer uso de herramientas de desarrollo como Docker han añadido a la aplicación una gran flexibilidad en el despliegue reduciendo los riesgos de conflictos entre librerías ya instaladas en el servidor. Por otro lado docker ha aportado escalabilidad en el sistema permitiendo migrar de una manera sencilla si el proyecto lo requiriese en el futuro.

El desarrollo de la Api REST en el lado backend, haciendo uso del framework Ruby On Rails me ha permitido tanto aprender a desarrollar una Api REST como aprender el lenguaje de programación Ruby y el framework Ruby On Rails. Se ha añadido seguridad a la Api haciendo uso de tokens los cuales deben de ser enviados en las cabeceras de las peticiones, sin las cabeceras necesarias la Api REST no responderá a las peticiones.

El framework Ruby On rails ha aportado al software consistencia y robustez ya que al basarse en el patrón de diseño Modelo, Vista y Controlador ocasiona que el código este muy bien estructurado. Así como el uso de ORM Active Record simplifica las consultas a base de datos.

En la actualidad el desarrollo web esta en pleno auge y los frameworks de desarrollo web basados en el lenguaje de programación Javascript son ampliamente conocidos y utilizados, aunque ya conocía javascript y lo había usado con frameworks como JQuery jamás había trabajado con un framework moderno de javascript como lo es VueJs.

Vue Js ha aportado versatilidad en el lado cliente de la aplicación ya que aunque es un framework liviano al ser progresivo permite añadir librerías si el proyecto las requiere.

Gracias al desarrollo de este proyecto he tenido la posibilidad de tanto asentar mis conocimientos adquiridos en el grado como adquirir nuevos conocimientos en el área del proceso de desarrollo de software, en el área del desarrollo web y el área de despliegue de aplicaciones.

8.1 Relación del trabajo desarrollado con los estudios cursados

Durante todo el proceso del desarrollo de la aplicación se han aplicado conocimientos adquiridos en el grado.

Como en *Proyecto de ingeniería del software* en este proyecto se ha desarrollado una aplicación partiendo de una idea hasta llegar a un producto. Esta asignatura me ayudo a estructurar el proceso de un desarrollo, seguir una metodología de trabajo y plantear fases durante el desarrollo de la aplicación.

En la fase de análisis del proyecto se han aplicado conocimientos adquiridos en las asignaturas de *Análisis, validación y depuración de software*, *Análisis y especificación de requisitos*.

Conocimientos aprendidos en las asignaturas de *Ingeniería del Software* y *Base de Datos* han sido usados para analizar y diseñar el esquema relacional de la base de datos de la aplicación.

Los prototipos de las interfaces se elaboraron basándome en lo aprendido en la asignatura *Interfaces persona computador*.

Tanto en el diseño de la arquitectura del sistema, como en la dockerización de la arquitectura del sistema fueron necesarios competencias adquiridas en la asignatura *Redes de computadores* e *Integración e Interoperabilidad*.

Las asignaturas de *Mantenimiento y evolución del software*, *Diseño de software* y *Calidad del software* me aportaron técnicas que permitieron diseñar la arquitectura software, implementar código limpio y a consecuencia conseguir un software de calidad y mantenible con la posibilidad de evolucionar en el futuro.

Las asignaturas matemáticas, aun siendo de las mas odiadas por la mayoría de alumnos, nos aportan los conocimientos matemáticos necesarios para que un Ingeniero realice su trabajo de una manera profesional, así como las asignaturas relacionadas con la Ingeniería del Software, las cuales todas me han aportado nociones que aplico día a día en el mundo laboral.

Todas y cada una de las asignaturas cursadas en el grado me han aportado grandes conocimientos que sin ellos no hubiese sido capaz de desarrollar este proyecto.

8.2 Trabajos futuros

En este capítulo se detallaran distintas funcionalidades y aspectos de la aplicación que podrían desarrollarse en un futuro.

- La aplicación se ha desarrollado en inglés, en un futuro seria interesante añadir internacionalización a la aplicación, implementando un sistema de traducciones en los textos para poder dar el servicio en distintos idiomas.
- Ampliar la creación de los torneos desarrollando nuevos algoritmos que permitan generarlos con distintos formatos.
- Desarrollar el sistema de seguidores entre usuarios permitiendo a los usuarios tener un dashboard de posts personalizado según las cuentas que sigan.
- Implementar tags en los posts enviados por los usuarios, esto permitiría que los usuarios personalicen los mensajes del dashboard filtrando por el contenido que quieran visualizar.
- Añadir a los posts funcionalidades típicas de las redes sociales, como pueden ser sistema de reacciones en los mensajes o posibilidad de compartirlos, las cuales no se han implementado por falta de tiempo.
- A pesar de que la aplicación se ha desarrollado con un diseño responsive el cual se adapta a distintos dispositivos, seria muy interesante desarrollar una aplicación móvil la cual se pueda utilizar nativamente sin hacer uso de navegadores. Solo seria necesario desarrollar la parte frontend de la aplicación móvil ya que se seguiría utilizando la Api Rest desarrollada en este proyecto.

Bibliografía

[1] Comparación con otros frameworks. https://es.vuejs.org/v2/guide/comparison.html.

[2] Db-engines ranking. https://db-engines.com/en/ranking.

[3] Draw.io herramienta web para elaborar prototipos de interfaz y diagramas. https://www.draw.io/.

[4] Ea: Single-player games are 'finished'. https://www.wired.com/2010/12/ea-single-playe.

[5] Framework ruby on rails. https://rubyonrails.org/.

[6] Framework vue js. https://vuejs.org/.

[7] Gestor de paquetes npm. https://www.npmjs.com/.

[8] Herramienta de control de versiones git. https://git-scm.com/.

[9] Herramienta desarrollo responsive. https://responsively.app/.

[10] Herramienta docker. https://www.docker.com/.

[11] Herramienta peticiones a la api. https://www.postman.com/.

[12] Ide de ruby on rails. https://www.jetbrains.com/es-es/ruby/.

[13] Javascript is the most popular programming language: Stack overflow survey. https://fossbytes.com/javascript-most-popular-programming-language-stack-overflow/.

[14] Jetbrains. https://www.jetbrains.com/es-es/.

[15] Lenguaje de programación javascript. https://developer.mozilla.org/es/docs/Web/JavaScript.

[16] Lenguaje de programación node js. https://nodejs.org/es/.

[17] Lenguaje de programación perl. https://www.perl.org/.

[18] Lenguaje de programación phyton. https://www.python.org/.

[19] Lenguaje de programación ruby. https://www.ruby-lang.org/es/.

[20] Most used javascript frameworks 2019. https://2019.stateofjs.com/front-end-frameworks/.

[21] Plataforma git hub. https://github.com/.

[22] Sistema de base de datos postgre sql. https://www.postgresql.org/.

[23] Software cliente de base de datos: Dbeaver. https://dbeaver.io/.

[24] Thank you for 100 million repositories. https://github.blog/2018-11-08-100m-repos/.

[25] Top javascript frameworks 2020. https://www.storyblok.com/tp/top-javascript-frameworks-2020.

[26] R. M. Agut. Especificación de requisitos software según el estándar de ieee 830. *Universidad Jaume I. Departamento de Inform {á} tica. Paper*, 2000.

[27] G. Fast and R. Fast. *Introducción al idioma Achuar*. Summer Institute of Linguistics, 2008.

[28] O. Filipova. *Learning Vue. js 2*. Packt Publishing Ltd, 2016.

[29] R. Isenberg. *Docker for Rails developers: build, ship, and run your applications everywhere*. Pragmatic Bookshelf, 2019.

[30] D. Jaramillo, D. V. Nguyen, and R. Smart. Leveraging microservices architecture by using docker technology. In *SoutheastCon 2016*, pages 1–5. IEEE, 2016.

[31] M. Masse. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. .ºReilly Media, Inc.", 2011.

[32] L. A. Pineda Soto et al. Metodologías ágiles de desarrollo-el caso ruby on rails. B.S. thesis, Bogotá-Uniandes, 2007.