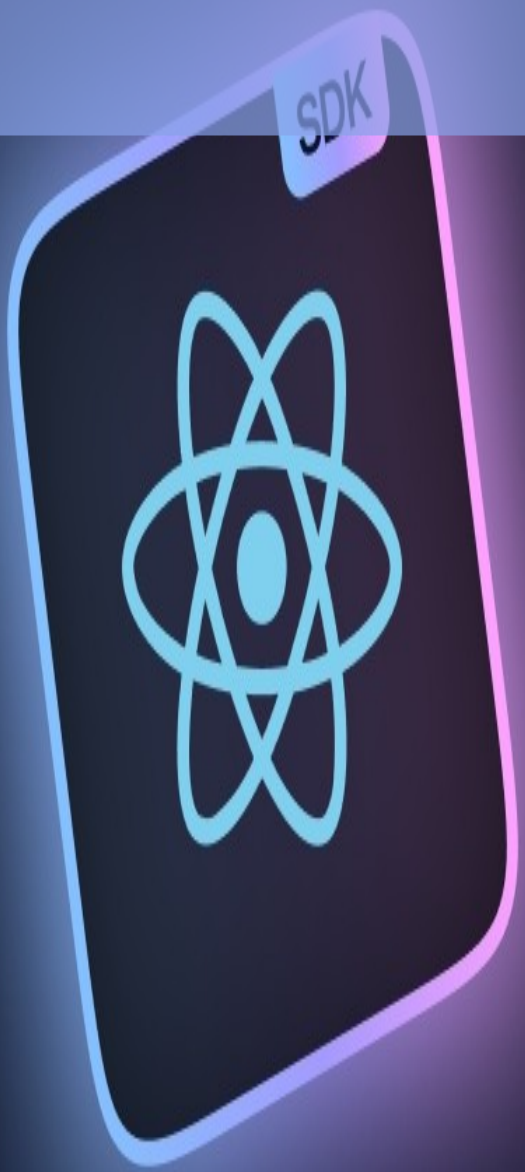
TRABAJO FINAL DE CICLO



IES Teis -DAW Distancia - 2024-2025

Alumno: Iago Ares Rodríguez

Sumario

DESCRIPCIÓN

Presentación del producto

El objetivo de este proyecto es el desarrollo de una pagina web destinada a Star Citizen para creación de misiones de comunidad para clanes y jugadores en solitario, funcionando como un tablón de anuncios con las misiones que han sido publicadas.

Tendrá filtrado por tipo de misión, roles y cronología, historial de misiones completadas, así como sección de pagos pendientes, recibidos y por hacer en caso de resultar ser el pagador.

De esta manera se cubriría la necesidad de los jugadores de tener que emplear tiempo de juego cada día para conseguir compañeros con los que jugar, ya que el tablón permitirá poner anuncios de larga duración, por ejemplo de un día para otro, para que al día siguiente ya haya usuarios dispuestos a la sesión de juego.

Diferenciación del producto

A diferencia de cualquier programa chat de voz, esta propuesta supone un enfoque diferente, permitiendo a jugadores individuales encontrar compañeros de juego, a grupos de jugadores pequeños expandirse en número y a las grandes organizaciones de jugadores una herramienta para organizar sus eventos y los jugadores que tomarán lugar en ellos de la manera apropiada.

JUSTIFICACIÓN

Estudio del macroentorno

En comunidades muy extensas de videojuegos como puede ser el caso de Star Citizen, juego para el que voy a hacer la página web, a los jugadores les resulta extremadamente difícil compatibilizar tiempos de juego entre miembros de un grupo de amigos o clanes. Es por ello que páginas de este tipo son necesarias, pues permiten a un jugador que necesite a más miembros en su grupo para una determinada actividad, publicar su necesidad e intención así como los pagos esperados y que en poco tiempo gente se le una voluntariamente.

Para entender porqué es necesaria esta web, hay que comprender la escala del juego y los tiempos involucrados en desplazarse. Es un juego espacial comprendiendo a veces tiempos de hasta 20 minutos de vuelo entre localizaciones (sólo un viaje, requiriendo ida y vuelta en la mayoría de las ocasiones), debido a estos largos tiempos de vuelo, cualquier misión puede resultar en una duración total de 1 hora y media, haciendo que una vez iniciada la misión sea contraproducente para el líder dar la vuelta y cancelarla, por pérdida de tiempo para los integrantes y la posibilidad de que uno de los integrantes ya no pueda quedarse hasta el final de la misión, abandonando y necesitando reemplazarlo para continuar con lo planeado.

La plataforma permitirá a la gente coordinarse libremente entre ellos de manera colaborativa, para que todos puedan disfrutar del juego dentro de la ventana de tiempo de la que cada uno dispone y sin la consecuencia de no poder disfrutar del contenido de comunidad que este ofrece porque sus amigos o clan están inactivos en ese instante.

En esencia solventaré el principal problema de los juegos MMORPG que es la incompatibilidad de horarios entre la gente que compone un clan o grupo, no es una solución que se pueda implementar a todos los juegos de este estilo, pero la economía del juego premia el trabajo en equipo incluso entre desconocidos, resultando en un ambiente que ampararía bastante el uso de mi sitio web.

Estudio del microentorno

En la actualidad no existen soluciones como las que se proponen en el proyecto, y las pocas que hay siguen la misma filosofía que pretendo implementar, que son “open-source” y sin ánimo de lucro. El cambio al modelo monetario sucedería después de haber reunido una masa crítica de usuarios, que el proyecto se encuentre en un estado de desarrollo que lo avale y justifique debido a los gastos en que incurriría debido al mantenimiento y el gasto de luz que supondría mantener la infraestructura.

Segmentacion del mercado

El mercado al que accedo es uno de grandes gastos, los jugadores de este juego se encuentran en un percentil muy alto de ingresos en relación a la población promedio, incurriendo en gastos superiores a 300€ en el juego al año, sopesando este dato, se decidió que cuando transicione a un modelo de pago, no se pedirá más allá de 2,50€ a usuarios individuales, menos si lo hacen a través de una organización de jugadores.

TECNOLOGÍAS

Front-end (gestión de eventos)

Se empleará Javascript como lenguaje para la gestión de eventos del proyecto, debido a que es el lenguaje estudiado a lo largo del cliclo como por mantener toda la lógica del sitio web en un mismo lenguaje de programación, en este caso como al haber elegido emplear Node.js en el backend, no habría transición entre lenguajes al cambiar de la gestión del frontend al backend.

Back-end

Emplearé Node.js o PHP. Node.js me interesa emplearlo debido a que tiene fáciles implementaciones con Javascript, y en caso de resultar muy complejo el aprendizaje, revertiré a PHP, por haber sido el lenguaje empleado durante el ciclo para el manejo del back-end

Se empleará Node.js montado en React, importando la librería express() para el desarrollo de una API RESTful, completamente nativa en Javascript, se pretende conseguir un desarrollo completo del sitio web en este lenguaje, necesitando solo SQL para las consultas y el trato con la base de datos.

Base de datos

MySQLWorkbench fue el entorno que empleé durante el ciclo, y con el que más familiaridad tengo, en aras de la productividad y la posibilidad de identificar errores más fácilmente, implementaré su uso en el proyecto.

Front-end (estilos)

Emplearé Tailwindcss y Primeflexcss, se trata de dos librerías de componentes CSS que he usado anteriormente en Angular, resultando en quizás la mejor opción para un correcto desarrollo de una interfaz agradable al usuario.

Desarrollo en React

Parecer ser el framework más adoptado (ligeramente más que Angular, y más que Vue), empleándolo en el proyecto, pretendo acostumbrarme a su uso de cara al mundo laboral.

MODELO DE CAPAS

Diagrama de clases

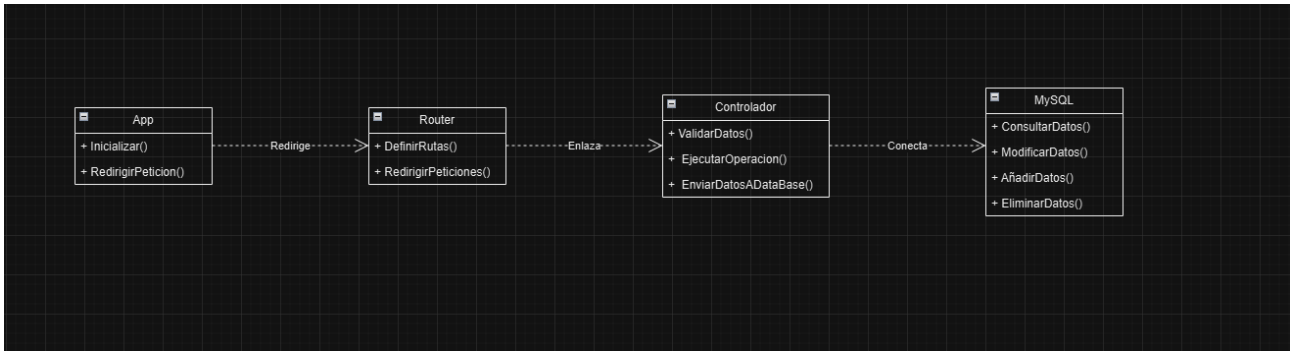


Ilustración 1: Diagrama de clases

Partiendo de la capa de aplicación se importan diversas utilidades, como la librería express() para habilitar el uso del enrutador que provee de servicio a todo el proyecto, Morgan para monitorizar las llamadas a la API desde la consola, Cookie-Parser para la creación de cookies de sesión y habilitar el uso de JSON así como el trato de HTTPs en express().

Una vez la app esta haciendo uso de todos los módulos importados, se configuran los puertos, y la dirección que usar para enrutar, en el caso de este proyecto sería “/api/usuarios, usuarios”, usuarios en esa declaración refiere a una constante con ese nombre que vincula con el archivo rutas.js, el router de este proyecto:

```
// Middlewares
app.use(morgan('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(cookieParser("Ares_Rodriguez_Iago_PFC_2024")); // Configurar cookies firmadas

// Configuration
app.set("port", config.app.port);

// Rutas
app.use('/api/usuarios', usuarios);
app.use(errorHandler);

module.exports = app;
```

Ilustración 2: Configuración de la capa de aplicación

El router distribuye todas las interacciones del proyecto con la base de datos a distintos end-points que después se usarían para conectar con elementos de la interfaz además de enlazar con el controlador cada end-point y mostrar los mensajes adjuntos de respuesta del servidor, aquí adjunto una imagen temprana del desarrollo del router:

```
const controlador = require('./controlador.js');
const { ro } = require('@faker-js/faker');


router.get('/', todos);
router.get('/id/:id', uno);
router.get('/session/login',login);
router.post('/session/logout',logout);
router.post('/', agregar);
router.put('/', update);
router.delete('/', eliminar);
router.get('/initialize/:initialize', initialize);
```

Ilustración 3: Declaración de distintos endpoints de la API

El controlador gestiona todas las interacciones del proyecto con la base de datos llamando a las distintas funciones que corresponderían a la acción que se quiere desempeñar y haciendo las comprobaciones de que los datos adjuntados cumplen con los requisitos para la ejecución de la siguiente función así como de que no hay un error de tipo o campo en blanco:

```
const db = require('../../db/mysql.js');

const tabla = "usuarios";



function todos (){
    return db.todos(tabla);
}

function uno(id){
    return db.uno(tabla, id);
}
```

Ilustración 4: Redirección de la capa Controlador a la capa de Base de Datos

La clase de base de datos fue llamada mysql y se encarga de todas las consultas y controles de error relacionados con la base de datos, carencia de datos en la llegada de la consulta etc, aquí adjunto la lógica de las funciones para que se entienda:

```
function update(tabla, user, data) {

    console.log(user.nombre);
    return new Promise((resolve, reject) => {

        // Validar que los identificadores contengan los campos necesarios
        if (!user.nombre || !user.password) {
            return reject('Se requieren los campos "nombre" y "password" para identificar el regist
        }

        // Verificar que se hayan proporcionado campos a modificar
        if (Object.keys(data).length === 0) {
            return reject('No se han proporcionado datos para actualizar');
        }

        // Ejecutar la búsqueda
        conexion.query(`SELECT * FROM ${tabla} WHERE nombre = '${user.nombre}' AND password = '${us
            if (err) {
                return reject(err);
            }

            // Si no se encuentran resultados, rechazamos la promesa
            if (resultados.length === 0) {
                return reject('No se encontró un registro con los identificadores proporcionados');
            }

            // Si el registro existe, proceder a actualizarlo
            const queryActualizar = `UPDATE ${tabla} SET ? WHERE nombre = ? AND password = ?`;
            console.log(sanitizeRequestBody(data).length);
            conexion.query(queryActualizar, [sanitizeRequestBody(data), user.nombre, user.password]
                if (err) {
                    return reject(err);
                }
                resolve(resultado); // Resolución exitosa con el resultado de la actualización
            });
        });
    });
}
```

Ilustración 5: Trato de los datos directamente con la Base de Datos

Capa de presentación (Front-end)

- Implementada con React.

- Se encarga de la interacción con el usuario, gestionando la interfaz gráfica y las peticiones al backend.
- Comunicación: Usa peticiones HTTPS (RESTful) para interactuar con la capa de negocio.

Capa de negocio (Backend)

- Desarrollada en Node.js
- Contiene:
  - Router: Define las rutas y distribuye las peticiones hacia los controladores correspondientes.
  - Gestionan la lógica de negocio específica, interpretando las peticiones, validando datos y coordinando operaciones con la base de datos.

Capa de datos (Base de datos)

- Utiliza MySQL para almacenar y manipular datos.
- Se accede desde el backend mediante el uso de la librería mysql en node.js para realizar consultas y actualizaciones.

Empleando cookies se controlan los datos del usuario requiriendo las distintas acciones que desembocan en una consulta a la base de datos. Por ejemplo:

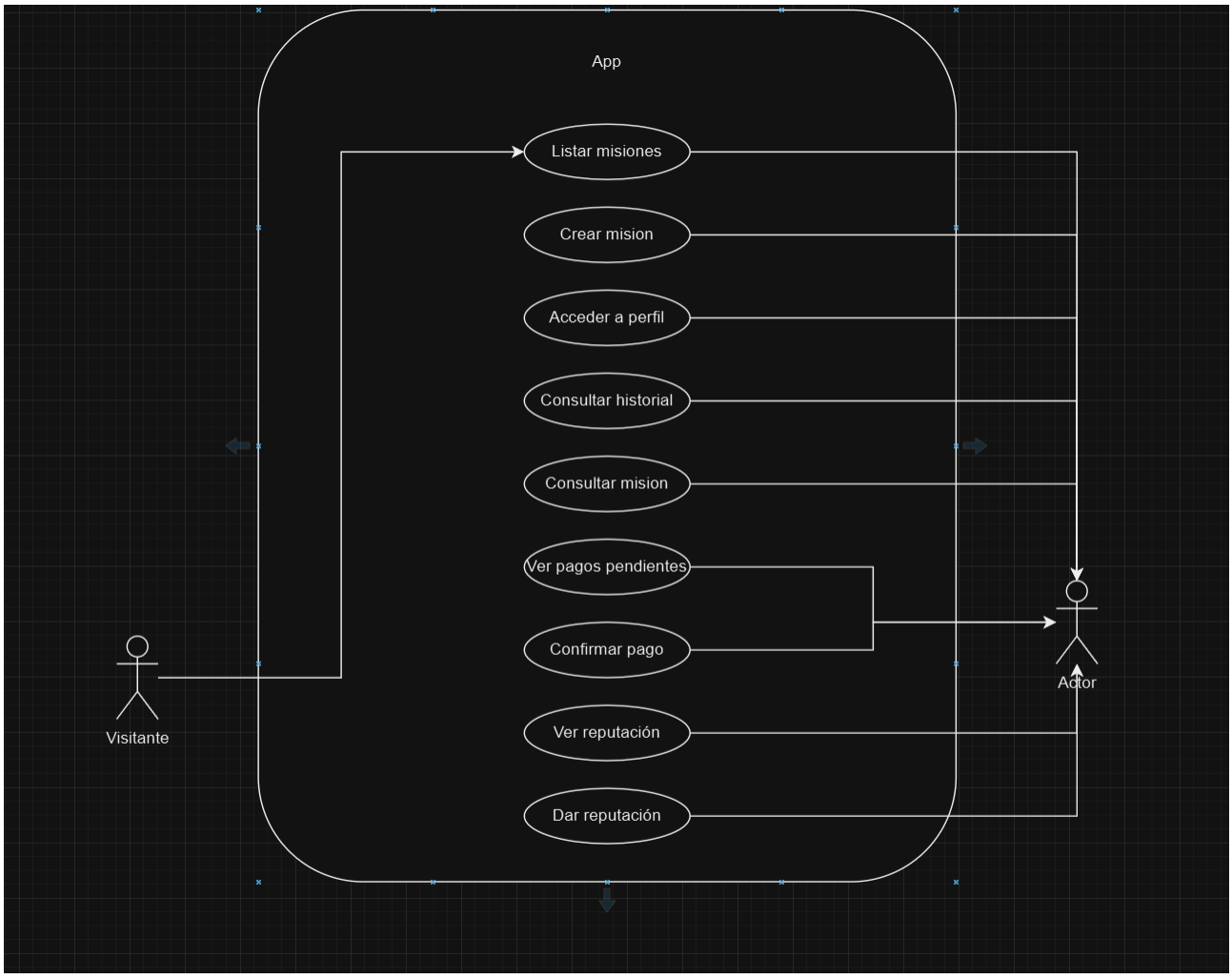
- Un usuario que entre a la web sin iniciar una sesión o registrarse, solo podrá visualizar las misiones en el tablón de anuncios. Dichas misiones solo aparecen ahí mientras no tengan todos los integrantes necesarios, una vez se llenan, desaparecen. He de aclarar que el tablón de anuncios, solo incluirá información genérica de la misión a no ser que el creador autorice que sea pública, para proteger datos importantes, como la hora de inicio, lugar de encuentro, destino…
- Un usuario con la sesión iniciada, tendrá la misma vista que un visitante a excepción de que tendrá un botón para crear misiones y una pestaña de Perfil, la cual incluirá todos los datos importantes de un usuario (nombre, nombre en el juego, y sus dos roles, que serán editables) y el acceso al historial de misiones, los pagos pendientes por recibir o realizar y su reputación[1].

1 La web empleará un sistema de reputación para evitar que los usuarios que hayan decidido dedicarse a jugar fuera del amparo de la ley, puedan apuntarse a todas las misiones que aparezcan para recopilar datos y ubicaciones de las

Se ha de aclarar que en ningún momento se espera poder ofrecer servicios a usuarios que no se hayan registrado, es decir, un visitante solo podrá ver la actividad de misiones ofertadas y para conocer el resto de posibilidades, habrá de iniciar sesión. Sin embargo y reconociendo que una página principal tan espartana en cuanto a posibilidades de interacción puede no ser un reclamo para jugadores que no conozcan la página, cabe la posibilidad de revisar en el futuro esta funcionalidad para crear una experiencia más amigable de cara a posibles usuarios que no hayan conocido la página gracias al boca a boca o a amistades que se la recomendaran.

DIAGRAMA DE CASOS DE USO

Empezando por el caso de uso de un visitante, no tiene muchas opciones, tan solo ver un listado de las misiones en busca de integrantes.



misiones para beneficio propio.

En el caso de un usuario registrado, empezando por las posibilidades de la vista **“Tablón de anuncios”**, solo podrán buscar o crear misiones en esta vista. Comprobación de datos y su modificación por parte del usuario en la pestaña **“Profile”.** Acceso a la pestaña **“Active missions”**, para la consulta de datos y la modificación por parte del usuario de las misiones que haya creado, y para consultar la información privada de la misión en caso de ser un simple usuario, en la pestaña **“Profile”**. Acceso a la pestaña **“History”**, para la consulta de las misiones realizadas, ejecución o confirmación de pago dependiendo de si eres el creador de la misión o un participante, en la pestaña **“Profile”**. Por último, acceso a la pestala **“Reputation”**, para la consulta de la puntuación relativa a tu reputación, obtenida como reseña del creador al final de la misión, en la pestaña **“Profile”**.

Flujo de consulta

Como ya se ha detallado con anterioridad, no se podrá controlar la actitud de los jugadores dentro del juego, así que para proteger la misión planteada y su correcto desarrollo, se darán datos genéricos de misión, así como los roles, naves que se van a usar, cantidad de jugadores necesarios y pagos a realizar. Un jugador que haya gastado tiempo en encontrar una herramienta de ayuda como la que planteo para el juego, sabrá leer entre lineas de la descripción genérica, y saber sin conocer datos exactos si es el tipo de misión a la que interesa apuntarse.

Una vez el usuario que quiere apuntarse ha enviado la solicitud, del lado del usuario que la ha creado, aparecerá una notificación en el área de “Active missions” en “Profile”, una vez ingrese en ese área, tendrá que seleccionar la misión activa, que tendrá un “dashboard” específico para ella, que consistirá en un listado por cada rol con los miembros ya apuntados y los huecos en blanco, y revisar uno a uno los jugadores que se quieren apuntar, recurriendo al sistema de reputación para saber si un jugador es de confianza o no. El creador usará los roles que el usuario tiene puestos como públicos en su perfil, para asignarle una posición en la misión.

Está propuesta la idea de un filtrado de misiones por roles públicos, para que a un “minero” y “transportista” no le salgan misiones de combate por ejemplo, pero para una primera versión se presumirá que el usuario solo busca misiones de sus roles públicos en el momento que hizo click en unirse a la misión.

Flujo de creación

Para la creación de una misión, el usuario deberá especificar distintos campos claves, siendo estos:

- Roles necesarios
- Jugadores por rol necesarios
- Pago
- Hora de inicio
- Requerimiento mínimo de reputación.
- Descripción de la misión.
- Ubicación de arranque.
- Duración esperada de la misión.
- Naves necesarias

Como ya hemos aclarado antes, de todos los campos que acabo de detallar, solo la descripción genérica, los roles necesarios así como la cantidad de jugadores por rol, las naves, requerimiento de reputación y el pago se verán en el tablón público.

La hora, la ubicación de arranque y la descripción en detalle solo entrarán a la vista de “Active missions” una vez el creador haya aceptado al jugador solicitante. En un futuro se pretende añadir la posibilidad de designar jugadores específicos a naves especificas dentro de la misión, así como el rol que tienen que cumplir, por ejemplo:

- En una nave médica, designar a un medico y decirle cual es su asiento en la nave.
- En una nave militar asignar una torreta especifica a un artillero.
- A los ingenieros, asignarles áreas de la nave que reparar en caso de averías.

Una vez creada la misión, el jugador que la ha creado y el que se haya apuntado solo podrá acceder a ella desde la vista “Profile” y clickando en “Active Missions”,al hacer click en esa pestaña del submenú, se hará una consulta SQL para construir la vista con las misiones correspondientes, y serán dos vistas diferentes dependiendo de si eres el dueño o un usuario apuntado.

DIAGRAMA DE LA BASE DE DATOS

Para permitir el flujo de uso planteado en los puntos anteriores, se tuvo que diseñar una base de datos con múltiples relaciones 1,N, pues un usuario puede tener múltiples misiones creadas, y cada misión puede tener múltiples roles, naves y solicitudes de usuarios para unirse.

La vista de un propietario de misión y de un usuario apuntado en la seccion “Profile” y en la subsección “Active missions”, será diferente y se controlará con el campo (creator_id) de la tabla de misión que contrastará con el id que la cookie de sesión le enviará adjunto a la petición HTTPS.

Para que la misión pase a la sección “History”, el creador debe dar por finalizada la misión desde su vista personalizada de la misión, y entonces la misión pasará a esa sección. A continuación se añade un diagrama relacional de la base de datos.

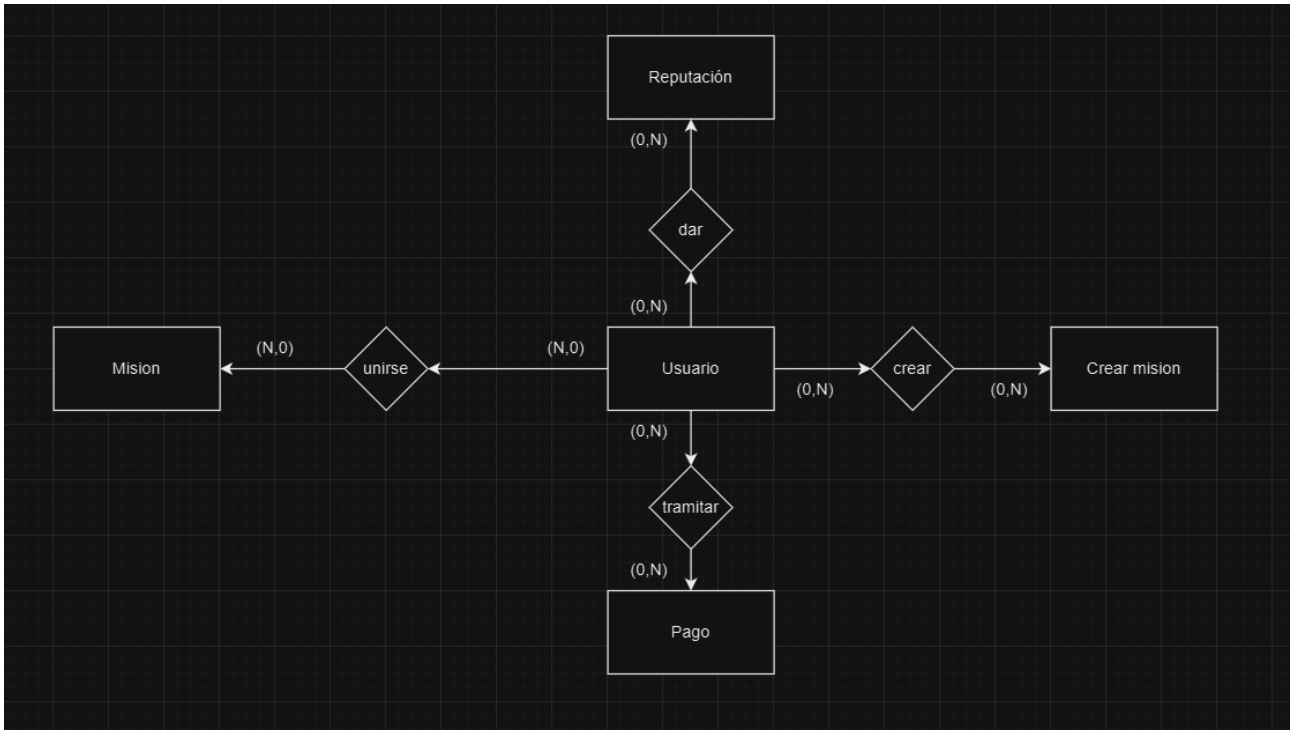


Ilustración 6: Modelo entidad relación

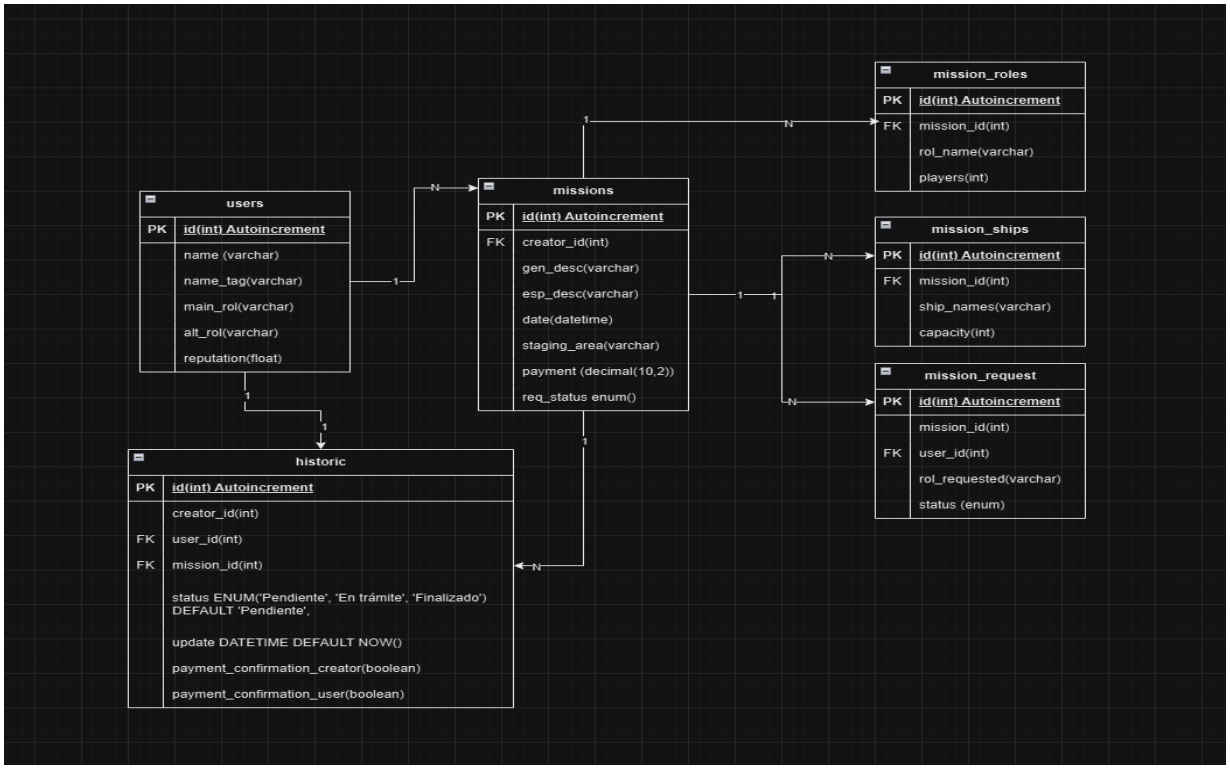


Ilustración 7: Diagrama de la Base de Datos

Como se puede apreciar, el uso de la web depende de 6 tablas, una de las cuales se podría decir que es el pilar del funcionamiento de la web, se trata de“Users” , de ella cuelgan todas las demás tablas.

Para prevenir cualquier error de interpretación del funcionamiento de la base de datos, procederá una explicación de la lógica de la base de datos y las posibles consultas que se realizarán sobre ella.

Tabla Users

Se trata de la tabla primaria de la que todo lo demás cuelga, ya que sin un “user_id” no hay identificación posible, las consultas a realizar sobre esta tabla para la creación de vistas se limitarían a la comprobación de datos para en “Profile”, “Customize”, poder modificar la información.

Tabla Missions

Es el elemento más complicado de la base de datos, ya que de ella se generan 3 tablas adicionales y su finalización de ciclo de vida, causa la creación de una nueva entidad de la tabla “Historic”. De cada entidad que se cree de esta tabla colgarán 3 tablas con definiciones que el usuario debe dar al momento de la creación, se rechaza la posibilidad de que ya estén indexados en la Base de datos los posibles roles y naves. Al habilitar al usuario el detallar el nombre de la nave y

su capacidad, así como los roles y su cantidad, se da libertad a los usuarios de crear nuevos nombres para naves específicas, así como para los roles, dando lugar a una experiencia un poco más personalizada.

Además, una tabla hija como es la tabla de solicitudes pendientes solo se tramitará mientras la misión tenga en la entidad de la base de datos, el “req_status” en open, tratándose de un enum() y habiendo definido ese estado como default. Una vez una “request” ha sido aceptada, se empleará esa tabla para identificar a los usuarios que si participan en la misión, para construir la vista detallada de misión a todos ellos.

Tabla Historic

Recoge los valores del pago esperado de la misión activa al finalizarla el creador y crea una entidad para cada uno de los integrantes de la tabla “Mission_request”, para que todos ellos tengan una entrada en el historial en relación a esa misión, y que en el historial del creador aparezca un desplegable con los nombres de los participantes, y pueda seleccionar uno a uno mientras realiza los pagos dentro del juego.

Para evitar problemas de finalización de pagos, por el momento se asume que el creador de la misión tiene la potestad última de finalizar el trámite, asumiendo que seguirá un principio honorable y no se quedará con los pagos prometidos. Así que cuando el creador, dice en primera instancia que el pago está en tramite, hasta que el usuario no lo corrobore en su historial, seguirá en pendiente, pero pasado un tiempo arbitrario, se enviará automáticamente una consulta a la base de datos para que modifique el campo boolean de confirmación del usuario y que el creador pueda cerrar la misión.

DECISIONES DE DISEÑO

Para el diseño de la web se ha propuesto una solución sencilla y con pocos o ningún elemento dinámico, tan solo se compone de descripciones, texto y botones llamativos para resaltar acciones o texto importantes para el usuario, como podrían ser, la hora de inicio de la misión, el emplazamiento desde el que saldrá, nombre del líder o del capitán de tu nave.

Además se busca un diseño compartimental de dos partes, con menú lateral de navegación así como una sección principal con el contenido que se mostrará en función de los permisos que la cookie de sesión te otorgue. En la imagen de referencia que se adjunta, no se visualiza toda la visión de la navegación que se espera disponga la página web. Por ejemplo:

- La barra lateral de navegación solo observa elementos de una acción, es decir, seleccionas “Overview” y no tiene subelementos que en el caso del sitio web que se plantea, si hay.
- Tiene una estética muy “broker” de la que se pretende escapar, pero la parte superior de la imagen contiene un texto que reza “transaction details”, ese nivel de detalle de hacer click y que cause un “dropdown” con el detalle completo de la misión es lo que se pretende alcanzar a nivel de diseño para la web.

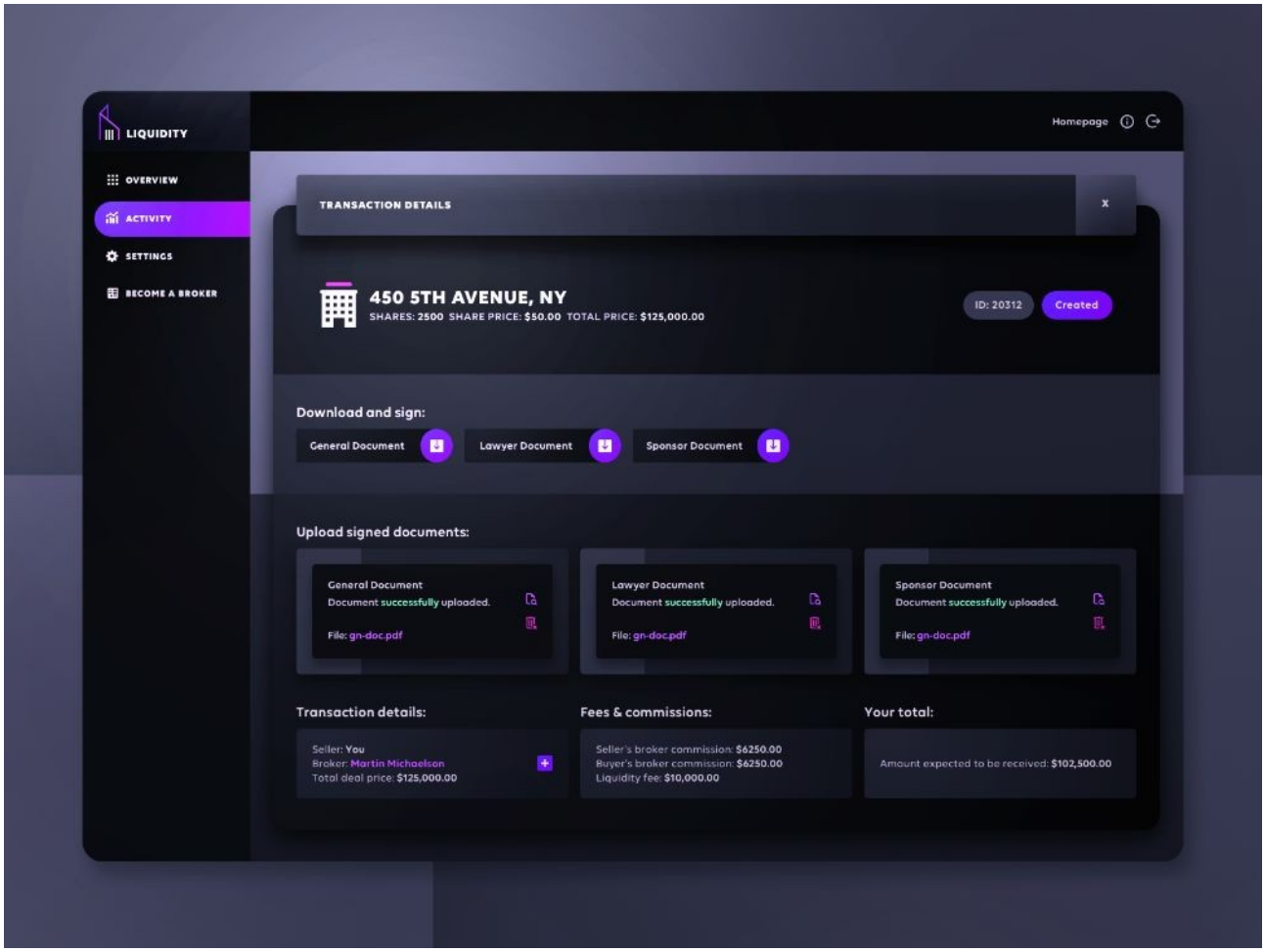


Ilustración 8: Inspiración para el aspecto final de la aplicación

Filosofía de diseño de la web

En cuanto al diseño del general de la aplicación en cuanto a software, se ha buscado mantener al mínimo la generación de distintas clases ajenas a los casos de uso de la web, es decir, si se revisase el código del backend así como del frontend, se vería una clara correlación entre los elementos con los que el usuario puede interactuar, y sus elementos de control y generación de respuestas en el código.

Manteniendo este esquema referencial entre clases, eventos y elementos gráficos, los propios nombres de los elementos declaran qué elemento controlan y a quién responden, haciendo muy sencilla la navegación entre clases en el desarrollo o control del código.

Se están reutilizando las mínimas funciones posibles, se están desarrollando funciones especificas para cada consulta, controlador, y end-point, necesitando que el enrutador reconozca varias archivos javascript para que la api resuelva, por ejemplo:

/api/usuarios/listado/
/api/invitados/listado/

RECURSOS MATERIALES Y PRESUPUESTO

A largo plazo se pretende implementar un servidor propio, habiendo comprado ya la infraestructura que se detalla a continuación:

Componente	Nombre	Precio
Placa Base	B450M Soyo	80€
Procesador	AMD Ryzen 5 5600g	49€
Ram	G.Skill Tridentz 3600Mhz	32€
Caja	Tacens Mars G1	29€
SSD	Crucial MX 512gb	60€
HDD	Seagate Barracuda 1TB	19€

Para un despliegue inicial se buscaba mantener un presupuesto no demasiado elevado, recurriendo así al mercado de segunda mano, todos y cada uno de los componentes listados han sido comprados en plataformas de compra-venta online, o son reciclados de sistemas retirados.

La razón para el empleo de estos componentes es la búsqueda de una infraestructura testeada en el tiempo como es la plataforma AM4 de AMD, así como de ciertas capacidades básicas como puede ser la capacidad de montar un Ubunru Server con interfaz gráfica, para una mejor monitorización del sistema, en vez de hacerlo integramente por SSH. Al final las capacidades gráficas del procesador no se requirieron ya que con el empleo de Docker implementando Portainer, una interfaz web para la administración de Docker, se pudo administrar en remoto el servidor sin necesidad de SSH más allá de configurar el servicio de Docker para que arranque en cuanto la máquina inicie.

IMPLEMENTACIÓN EN LOCAL

Objetivo

Desplegar de manera autónoma la infraestructura para crear un servidor web de altas capacidades propio, sin necesidad del pago de un hosting, con capacidad de hacer DDNS (Dynamic DNS, para prevención del cambio de IP publica que puede ocurrir con un proveedor de internet convencional) y un proxy a través de Oracle Tunnels para evitar conexiones directas al router privado.

¿Cómo funciona Docker?

La tecnología Docker utiliza el kernel de linux y sus funciones, como los grupos de control y los espacios de nombre, para dividir los procesos y ejecutarlos de manera independiente. El propósito de los contenedores es ejecutar varios procesos y aplicaciones por separado para que se pueda aprovechar mejor la infraestructura y, al mismo tiempo, conservar la seguridad que se obtendría con los sistemas individuales.

Las herramientas de los contenedores, como Docker, proporcionan un modelo de implementación basado en imágenes. Esto permite compartir fácilmente una aplicación o un conjunto de servicios, con todas las dependencias en varios entornos. Docker también automatiza la

implementación de las aplicaciones (o los conjuntos de procesos que las constituyen) en el entorno de contenedores.

¿Qué es Portainer?

Portainer es una interfaz de usuario para administrar contenedores Docker, muy simple, tanto a la hora de la facilidad con la instalación como a la hora administrar los contenedores.

¿Que es un homelab?

Un homelab es un entorno local en el que puedes experimentar con diferentes tecnologías e implementaciones de códigos con los que estás trabajando. Enlazando con el proyecto, se pretende implementar un servidor local a través de docker que saldría al exterior a través de Nginx, un servidor HTTPS como hosteo y empleando Cloudflare para hacer de servidor Proxy inverso, de servidor DNS dinámico, así como para la compra del dominio a emplear (raiares97.es

IMPLEMENTACIÓN EN LA NUBE

Cloudflare

Para la entrega del proyecto se buscará una manera más inmediata de que sea accesible, empleando la implementación directa con GitHub de la que dispone Cloudflare para importar

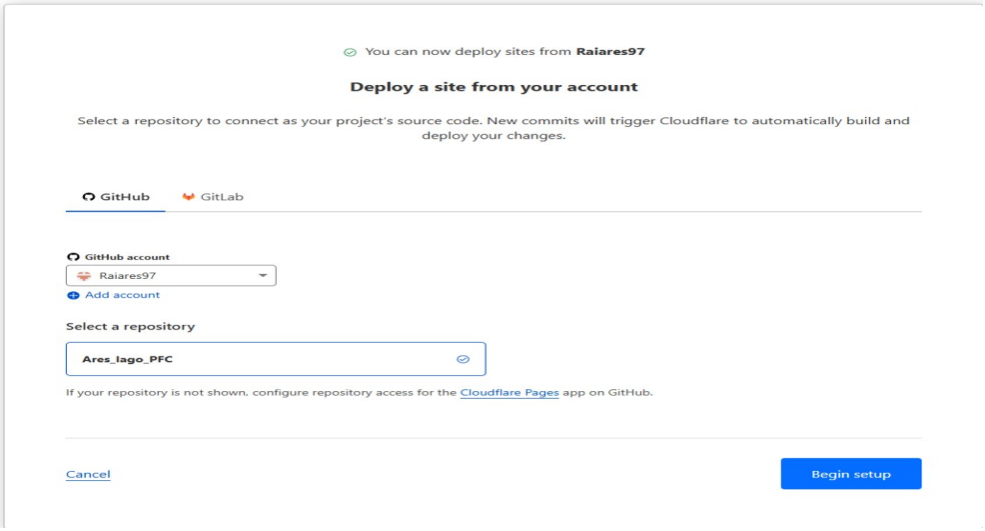
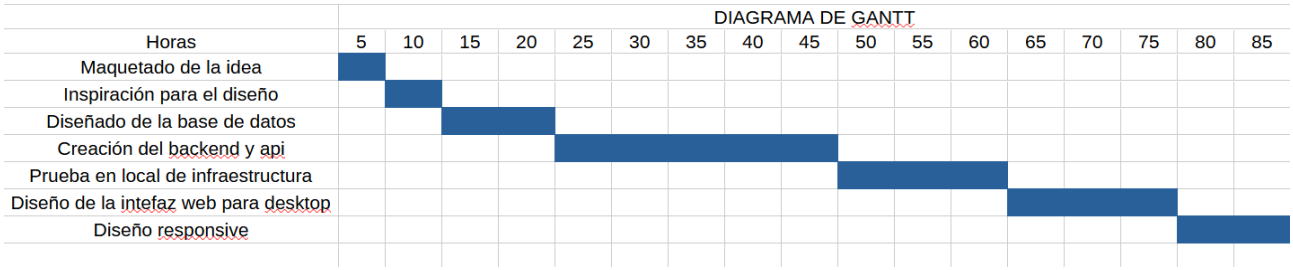


Ilustración 9: Cloudflare integration for GitHub

directamente el proyecto a su servicio de Hosting y DNS, como se realizó la compra de un dominio a través de ellos, tengo acceso a sus servicios de Hosting gratuitos.

TIEMPOS DE EJECUCIÓN

A continuación se desglosarán las horas dedicas a cada elemento del sitio web, como el desarrollo sigue en activo solo puedo adjuntar una estimación de las horas que se le dedicaron a cada elemento.



API

La complejidad de la API requiere que se vaya montando y testeando extensamente cada una de las integraciones, probando todos los casos de usos para los que fueron diseñadas, eso ha resultado en más de 40 horas empleadas entre el aprendizaje del entorno React y Node.js

En estos últimos días ha sido obvia la mejoría en el rendimiento personal, ya que y por dar un ejemplo, la implementación del protocolo HTTPS ha sido mucho más llevadera y rápida de lo esperado.

Back-end

El desarrollo del back-end fue en simultaneo con la API así pues el enroutador, la capa de gestión de errores, la configuración del certificado SSL para la comunicación HTTPs, la capa del controlador de la base de datos, y las consultas a la base de datos están siendo todas desarrolladas en conjunto con la API, resultando en extensos tiempos de testeo para un correcto funcionamiento del proyecto. Sin embargo, pocas veces se ha necesitado revisitar un método o una función una vez esta ha sido testeada. Como ya se mencionó antes, el tiempo de desarrollo se estima que alcanzará las 100h antes de la entrega del proyecto, una estimación conservadora ha de añadirse.

Front-end

El desarrollo front-end ha llevado el menor de los tiempos de todos los elementos que componen el sitio web, debido a que se determinó que no habría tiempo para un correcto desarrollo de una interfaz web con todos los controles responsive correspondientes, resultando en la toma de la decisión de limitar el desarrollo de la interfaz a lo mínimo exigible para demostrar el flujo de uso propuesto en el proyecto.

Es decir para la entrega del proyecto se buscará que el sitio web tenga 2 paginas, un formulario de LogIn y que todos los elementos con los que se puede interactuar funcionen, entreguen la información al backend correctamente y que muestren la información de la manera correcta también.

A futuro se espera dedicar una cantidad bastante superior de horas a las estimadas para el backend y API, ya que se contemplaría la implementación de un diseño responsive a todos los dispositivos posibles.

LINEAS FUTURAS

futuro la propuesta del proyecto contempla una serie de características adicionales que ampliaran el alcance e interacción con los usuarios, detallados a continuación:

Integración con Discord

Previa explicación de la idea, cabe una explicación de qué es Discord. Se trata de una aplicación de voz que permite a los jugadores comunicación en directo entre ellos contemplando el esquema de salas, es decir, existen servidores con salas de voz que permiten a los usuarios con los permisos adecuados, unirse a esa sala.

Durante el desarrollo del proyecto se descubrió la existencia de una API de discord que permite integrar un bot con permisos de administrador que gestiona la creación de salas de audio y asignación de permisos a la sala y a los usuarios. Se pretende integrar el bot de manera que se le pueda dar la opcion al creador de la misión adjuntar un servidor de Discord propio para que el bot

cree una sala, invite a los integrantes de la misión al servidor y les dé los permisos necesarios para que puedan interactuar entre ellos por voz.

Previamente tendrá que instalar el bot de manera manual, ya que no existe manera de automatizar la instalación en un sistema operativo sin interacción y consentimiento del usuario.

Integración de Clanes u Organizaciones

Se pretende integrar la posibilidad de un tablón de anuncios personalizado para grupos grandes de jugadores, que se mostraría en una pestaña a parte con el logo de las organizaciones, así como siendo completamente privados e inaccesibles a excepción de por invitación a las mismas.

Backend y API estandarizados

Una vez alcanzado el final del desarrollo de esta propuesta del proyecto, so contempla la idea de estandarizar la API y el backend, para que funcionen de con la mínima modificación en otros juegos. Se conseguiría dedicando considerables recursos al aprendizaje de como optimizar la API, pero la idea sería en crear un dominio que conceda el acceso a distintos subdominios con acceso a utilidades como la que se desarrolla en este proyecto, para distintos juegos resguardando el acceso a cada uno de estos subdominos con un micropago mensual.

La cuantía por persona variaría en función de si es una organización la que quiere el servicio o un particular para jugar con otros jugadores, pero no superaría los 2.50€ mensuales en todo caso, no se contempla que el precio sea una barrera para que los usuarios decidan probar los servicios.

CONCLUSIONES

En conclusión, el proyecto no plantea un modelo empresarial, tan solo un proyecto personal que explora la posibilidad de la monetización en aras de un coste de mantenimiento 0 y una presentación de un portfolio personal que acredite experiencia sólida en un framework así como en distintas tecnologías y procedimientos. Con eso no significa que no se contemple el pivotar enteramente a una dedicación profesional a este proyecto, decidiendo declarar una empresa individual como autónomo o una pequeña sociedad limitada.

Pretende también suplir la necesidad de la comunidad española de una plataforma como la que planteo, ya que todas son extranjeras, ignorando en todo caso el Español como idioma al que dar soporte, añadiendo una barrera lingüística a aquellos jugadores que no entienden Inglés.

BIBLIOGRAFÍA

Pretende también suplir la necesidad de la comunidad española de una plataforma como la que planteo, ya que todas son extranjeras, ignorando en todo caso el Español como idioma al que dar soporte, añadiendo una barrera lingüística a aquellos jugadores que no entienden Inglés.

Bibliografía

1: JGraph Ltd, Draw.io, 2014, https://app.diagrams.net/

2: Solomon Hykes, , 2013, https://docs.docker.com/engine/install/ubuntu/

3: Solomon Hykes, , 2013, https://docs.portainer.io/start/install-ce/server/docker/linux

4: César Arango, , 2022, https://www.youtube.com/@cesararangoweb/videos

5: Adam Wathan, , 2017, https://tailwindcss.com/docs/installation