

TRABAJO DE FIN DE GRADO

DESARROLLO DE APLICACIONES WEB



LUCKYLOOP

Luckyloop es ese rincón donde el juego se siente como en casa. Aquí no solo apostamos, nos divertimos, aprendemos y sobre todo nos fiamos el uno del otro, ¿te unes?

Alumnos: Darío Collar, Alejandro Cernada y Omar Daza

Tutor: Gustavo Millán

Curso académico: 2024-2025

Centro: IES Cañaveral

Ciclo: Desarrollo de Aplicaciones Web

Fecha de presentación: junio 2025

RESUMEN PROYECTO

Luckyloop es más que un casino, es el desarrollo completo de una plataforma de un casino online, implementando desde los juegos básicos hasta los sistemas de seguridad y gestión de usuarios. El objetivo es crear una experiencia de juego segura, atractiva y totalmente funcional que demuestre el dominio de las tecnologías web.

ÍNDICE

1. INTRODUCCIÓN	3
• 1.1. Descripción y contexto del proyecto	3
• 1.2. Motivación del proyecto	3

2. OBJETIVOS DEL PROYECTO 3

3. CONTEXTO ACTUAL 4

4. ANÁLISIS DE REQUISITOS 5

- 4.1. Estado del arte 5
- 4.2. Conceptos clave 5

5. DISEÑO DE LA APLICACIÓN 6

- 5.1. Diagrama de casos de uso 6
- 5.2. Requisitos funcionales principales 6
- 5.3. Requisitos no funcionales 6
- 5.4. Descripción de los usuarios y sus necesidades 6

6. DESARROLLO DE LA APLICACIÓN 7

- 6.1. Mockups y prototipos 7
- 6.2. Arquitectura del sistema 7
- 6.3. Diagramas de clases y entidad-relación 7
- 6.4. Diseño de la base de datos 7

7. PLANIFICACIÓN DEL PROYECTO 12

- 7.1. Tecnologías y herramientas utilizadas 12
- 7.2. Descripción de las funcionalidades implementadas 12

8. PLANIFICACION DEL PROYECTO 13

- 8.1. Acciones 13
- 8.2. Temporalización y secuenciación 13

9. PRUEBAS Y VALIDACIONES 14

10. RELACION DEL PROYECTO CON LOS MODULOS 15

11. CONCLUSIONES 16

12. PROYECTOS FUTUROS 17

13. BIBLIOGRAFIA 18

14. ANEXOS 19

1. INTRODUCCION:

En el mundo del entretenimiento los casinos online han experimentado una transformación y crecimiento a nivel online. Y este proyecto nace de la fascinación por crear experiencias que combinen ocio, entretenimiento, seguridad y la tecnología más moderna.



1.1 DESCRIPCION Y CONTEXTO DEL PROYECTO:

Consiste en el desarrollo de una plataforma de casino online, abarcando desde la implementación de juegos clásicos de toda la vida como ruleta, blackjack hasta creación de juegos actuales de los casinos convencionales hoy en día, ya que el gaming online ha crecido.

Además, creación de un sistema de gestión de usuarios, transacciones seguras, y administración de la plataforma.

Aportamos una plataforma buscando la experiencia digital y ocio disponible 24/7.

1.2 MOTIVACION DEL PROYECTO:

La motivación principal en este proyecto fue la necesidad de enfrentarse a un desafío técnico de alta complejidad que unía tanto un complejo sistema backend con gestión de usuarios, seguridad, fiabilidad de nuestro sistema, etc. Por otra parte, en el frontend una gran variedad y compleja visualización de cada juego, interfaces atractivas e interactivas, etc.

Además el proyecto nos permitió profundizar y aprender más de los casinos online actuales, ya que tuvimos que investigar cómo hacer muchos de los juegos que queríamos implementar en nuestro proyecto, así como el backend para estos, o para transacciones

1.3 BENEFICIOS ESPERADOS:

Podríamos decir varios beneficios que querríamos tener, pero a nivel educativo el dominio de tecnologías modernas y su implementación, comprensión de los procesos de desarrollo de un proyecto, experiencia en manejo de seguridad en proyectos. Por otra parte, a nivel profesional ojalá un posicionamiento gaming en la industria que hoy en día tiene una gran demanda, demostrar competencias de tener una capacidad de trabajar con tecnologías modernas.

Y a nivel proyecto que se puede tener una gran confianza en cuanto seguridad de gestión de usuarios, password, y transacciones.

2. OBJETIVO/S GENERALES DEL PROYECTO:

Nuestro objetivo fue desarrollar una plataforma de un casino online que ofrezca una experiencia de juego seguro, y sea atractiva al usuario. Implementando tecnologías modernas e intentando mejores prácticas de desarrollo para crear un sistema confiable y escalable. Englobando la creación digital que no solo sea entretenimiento de calidad si no que demuestre un dominio de las tecnologías trabajadas. Por otra parte, somos estudiantes que nos gusta los desafíos complicados.

3. OBJETIVOS ESPECIFICOS:

Para alcanzar esos objetivos propuestos hemos desarrollado la siguiente estructura.

3.1 ANALISIS Y PLANIFICACION:

Analizar y definir los requisitos funcionales y no funcionales de nuestro sistema, identificando las necesidades de usuarios y estableciendo un desarrollo sostenible de nuestra plataforma.

Investigar y evaluar tecnologías más adecuadas para el desarrollo de este.

3.2 DISEÑO DEL SISTEMA:

Diseñar la arquitectura de entidad-relación y modelos de datos de nuestro proyecto para que este se vea de forma clara y eficiente todas las relaciones que nuestro proyecto contiene.

Desarrollar el diseño de la base de datos que sea optima, y represente de forma clara la integridad de datos y facilidad para realizar consultas entre las diferentes entidades de nuestro proyecto.

3.3 DESARROLLO DE LA INTERFAZ DE USUARIO:

Diseñar y desarrollar la interfaz de usuario que sea atractiva para estos y fácil de utilizar, que proporcione una experiencia fluida.

Implementar un diseño responsive que garantice la experiencia de dispositivos tanto de escritorio, como móviles.

Crear interfaces para cada tipo de juego que contiene nuestra plataforma manteniendo la coherencia, pero también adaptando a las necesidades de entretenimiento que queremos aportar.

3.4 DESARROLLO DEL BACKEND Y LOGICA DEL PROYECTO:

Implementar la lógica del casino desarrollando la parte de atrás, la parte “oculta” de nuestro proyecto.

Desarrollar un sistema de gestión de usuarios robusto que incluya registro y login, como autorización y privacidad.

Crear u sistema de gestión financiera dentro de nuestra plataforma con la que trabajamos con tecnologías nuevas, que suelen usar en proyectos grandes. En el cual contiene retiros, depósitos, saldos de usuario etc.

3.5 SEGURIDAD Y CALIDAD:

Implementar medidas de seguridad avanzadas con las cuales también hemos trabajado con otra tecnología moderna que suelen trabajar en pymes y empresas grandes. Previniendo SQL injection, XSS entre otros.

3.6 TESTING Y VALIDACION:

Realizar pruebas de nuestro sistema para garantizar que todas las funcionalidades operen correctamente.

Validar que la experiencia del usuario sea fácil y entretenida.

4. CONTEXTO ACTUAL:

4.1 ESTADO DEL ARTE:

El mercado de casinos online ha experimentado un gran crecimiento y ha dado lugar a varias plataformas en el sector.

Plataformas líderes como:

- Bet365, que se convirtió en una de las plataformas más completas del mercado, que incluye casino y deportes.
- 888Casino que ha llegado a implementar IA en su experiencia de usuario.
- PokerStars que revolucionó el póker teniendo funcionalidades a tiempo real.

Actualmente las plataformas existentes garantizan transparencia en los juegos, IA para determinados juegos etc.

4.2 CONCEPTOS CLAVE:

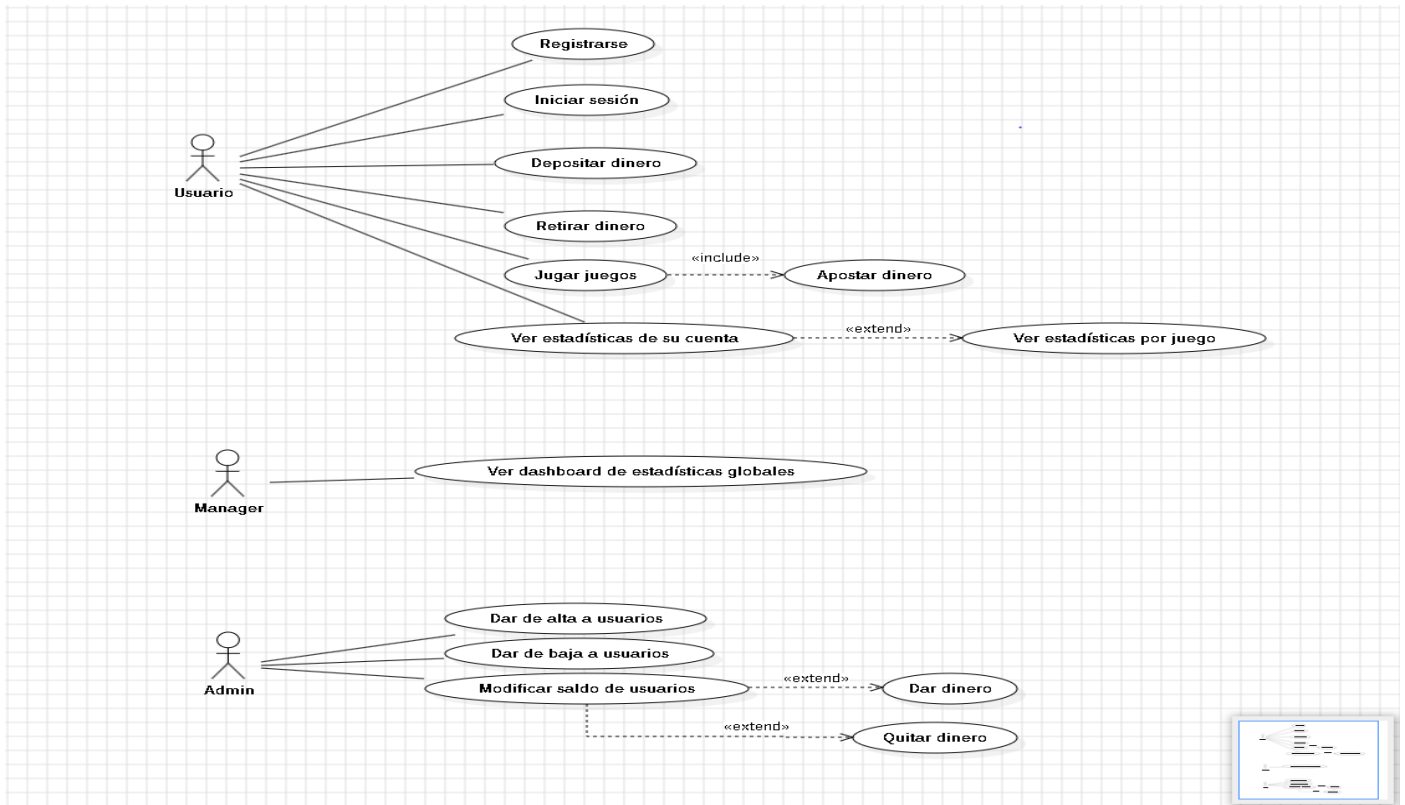
Hemos llegado a conclusión que los conceptos que tendría nuestro proyecto serían:

- Juegos y mecánicas de casino: juegos clásicos, incluyendo ruleta, blackjack, etc..., que requieren implementaciones para las probabilidades en cuanto juego.
- Sistema de apuestas y gestión financiera del usuario: manejando depósitos, retiros y el balance de fondos usuario y juegos.
- Futuras implementaciones de promociones por juegos, registros, días de recarga... etc.
- Experiencia de usuario e interfaz: diseño responsive en nuestra plataforma para que se pueda ver correctamente como en dispositivos de escritorio como móviles.
- Seguridad de nuestra plataforma: con sistema de autenticación y autorización con verificación de identidad de usuario y contraseña propia. Además de usar un sistema de autenticación con JWT.
- Juego responsable

5. ANALISIS DE REQUISITOS:

5.1 DIAGRAMA DE CASOS DE USO:

Nuestro diagrama de casos de uso representa una visión más general de todas las interacciones posibles entre los tipos de usuario que tenemos en nuestro proyecto.



5.2 REQUISITOS FUNCIONALES PRINCIPALES: FUNCIONALIDADES QUE DEBE TENER LA APLICACION:

- Usuario:
 - Registrarse o loguearse en nuestra plataforma.
 - Depositar dinero o retirarlo a través de unas de nuestras herramientas (stripe)
 - Jugar a diferentes juegos dentro de nuestra plataforma
 - Apostar dinero, con posibilidades de ganarlo.
 - Visualizar estadísticas generales de su cuenta a los juegos que juego
 - Ver estadísticas y ranking mundial
- Manager:
 - Ver el dashboard de estadísticas globales en la plataforma (balances, juegos...)
- Administrador:
 - Dar de alta, baja usuarios.
 - Modificar el saldo de usuarios, así como quitar o dar dinero a estos

5.3 REQUISITOS NO FUNCIONALES: RENDIMIENTO, SEGURIDAD, USABILIDAD, ETC:

Además de las funcionalidades anteriores, el sistema debe incluir requisitos no funcionales:

- Seguridad:
 - Los usuarios autenticados pueden acceder al sistema, poder jugar, apostar, y ver ranking entre otras opciones
 - Los datos personales y financieros están hasheados y protegidos
- Rendimiento:
 - El tiempo de respuesta de nuestras operaciones está hecho para que tarde lo mínimo posible
- Disponibilidad:
 - Estamos disponibles todo el día 24/7, con nuestra plataforma online
- Usabilidad:
 - La interfaz es intuitiva y fácil de usar para todos nuestros usuarios.
 - Además, optamos por la experiencia fluida y comprensible de nuestra plataforma

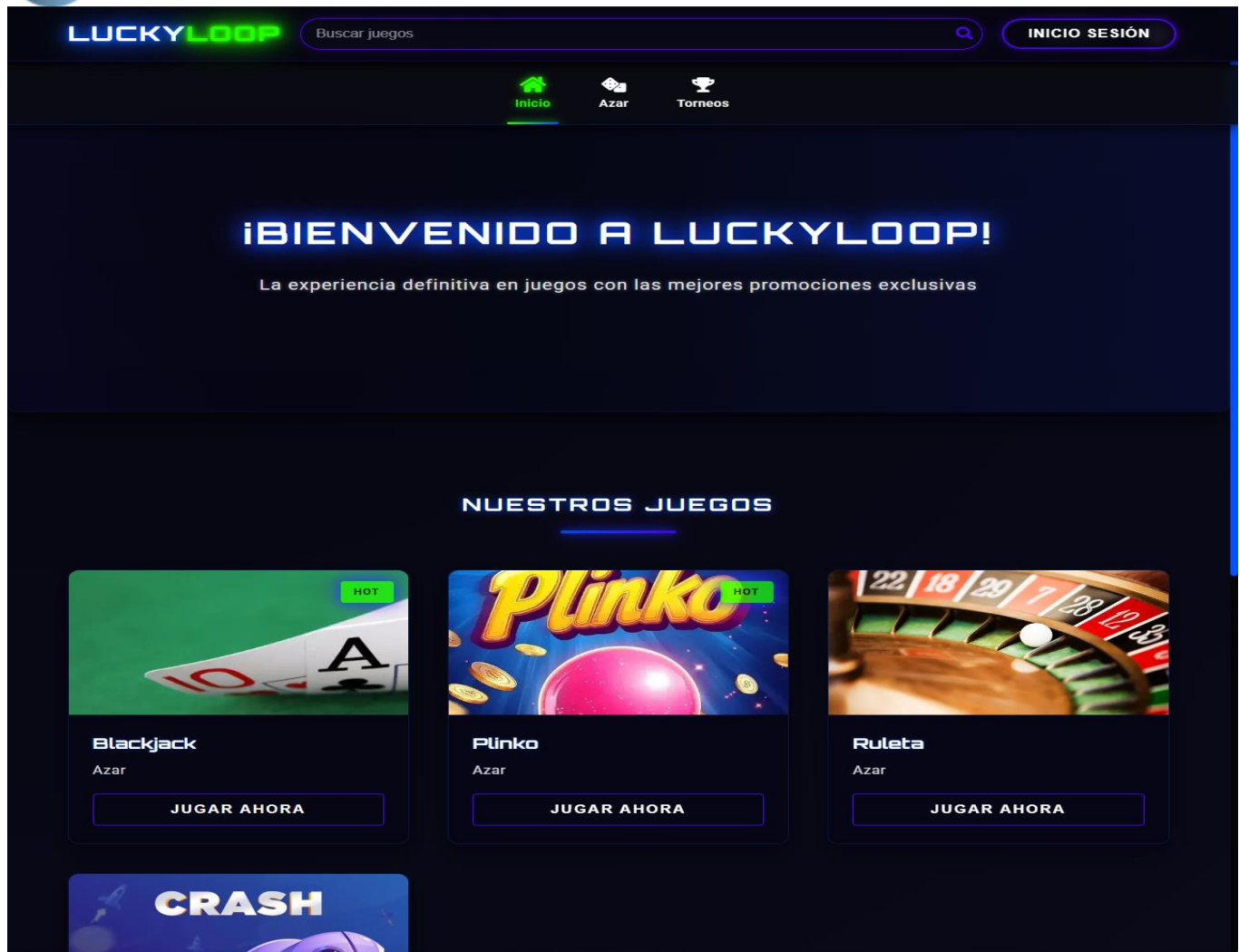
5.4 DESCRIPCION DE LOS USUARIOS Y SUS NECESIDADES:

- Usuario: El cual se registra y logea en nuestra plataforma, apuesta, juega y se divierte en nuestra plataforma. Puede depositar y retirar dinero, al igual que puede ver si está de los primeros en nuestro ranking mundial
- Manager: Puede ver las estadísticas globales de nuestra plataforma, así como los juegos, el balance de los usuarios, pero no interactúa con los usuarios de forma directa.
- Administrador: Es el responsable de la gestión de los usuarios, con esto puede crear, borrar y editar usuarios, desactivar manualmente muchas de las opciones en gestión de usuarios. Además, tiene un rol más técnico y de mantenimiento dentro de nuestra plataforma.

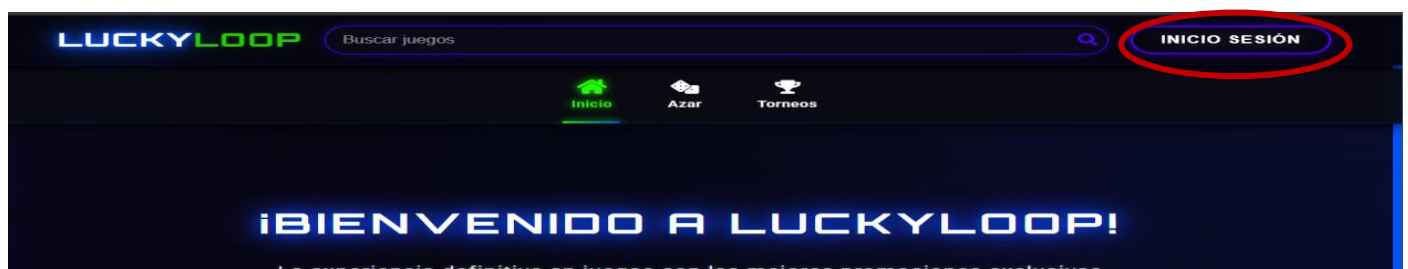
6. DISEÑO DE LA APLICACION:

6.1 MOCKUPS O WIREFRAMES O PROTOTIPOS DE LA INTERFAZ GRAFICA DE USUARIO:

- El menú principal de nuestra página al encontrarnos




- Al querer acceder a tu cuenta en nuestra plataforma a través del inicio de sesión.



- Podrás iniciar sesión en nuestra plataforma siempre que tengas cuenta de usuario. En caso de no tener, tendrás que registrarte, siendo fácil e intuitivo.



Iniciar Sesión

ENTRAR

¿No tienes cuenta todavía? [Registrarse](#)
[Olvidé mi contraseña](#)

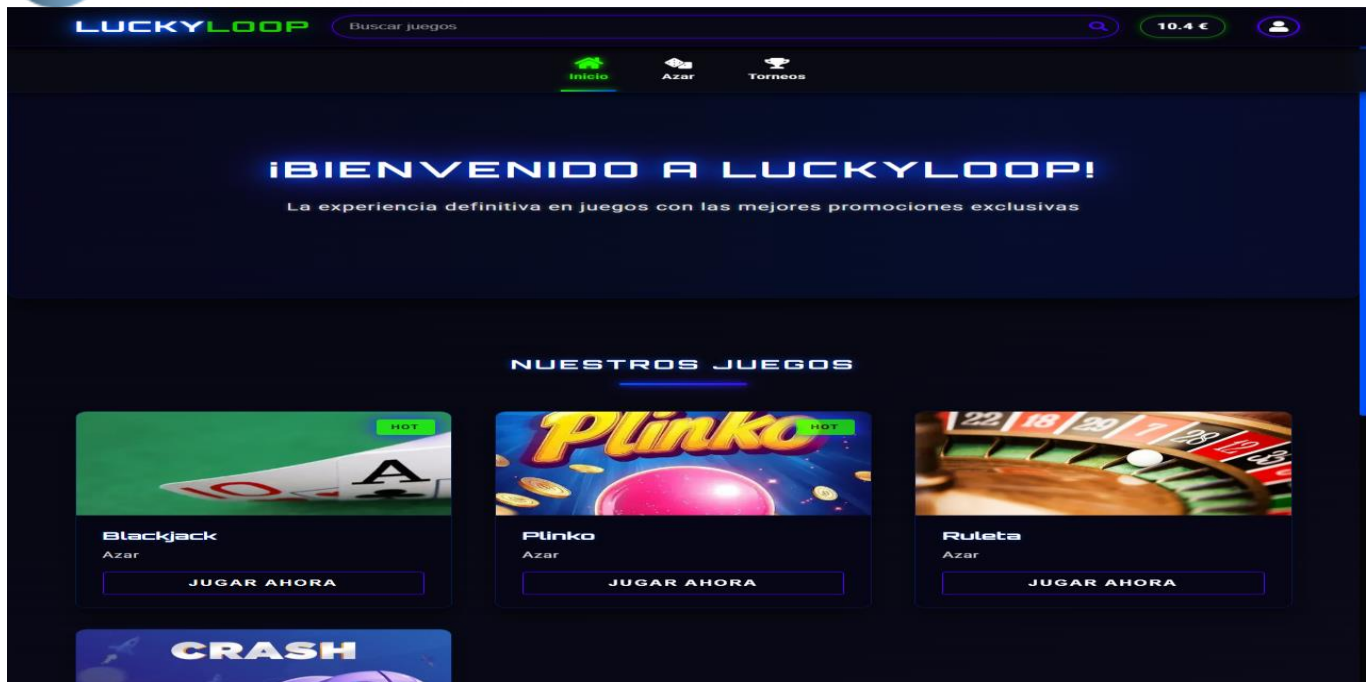
Registro

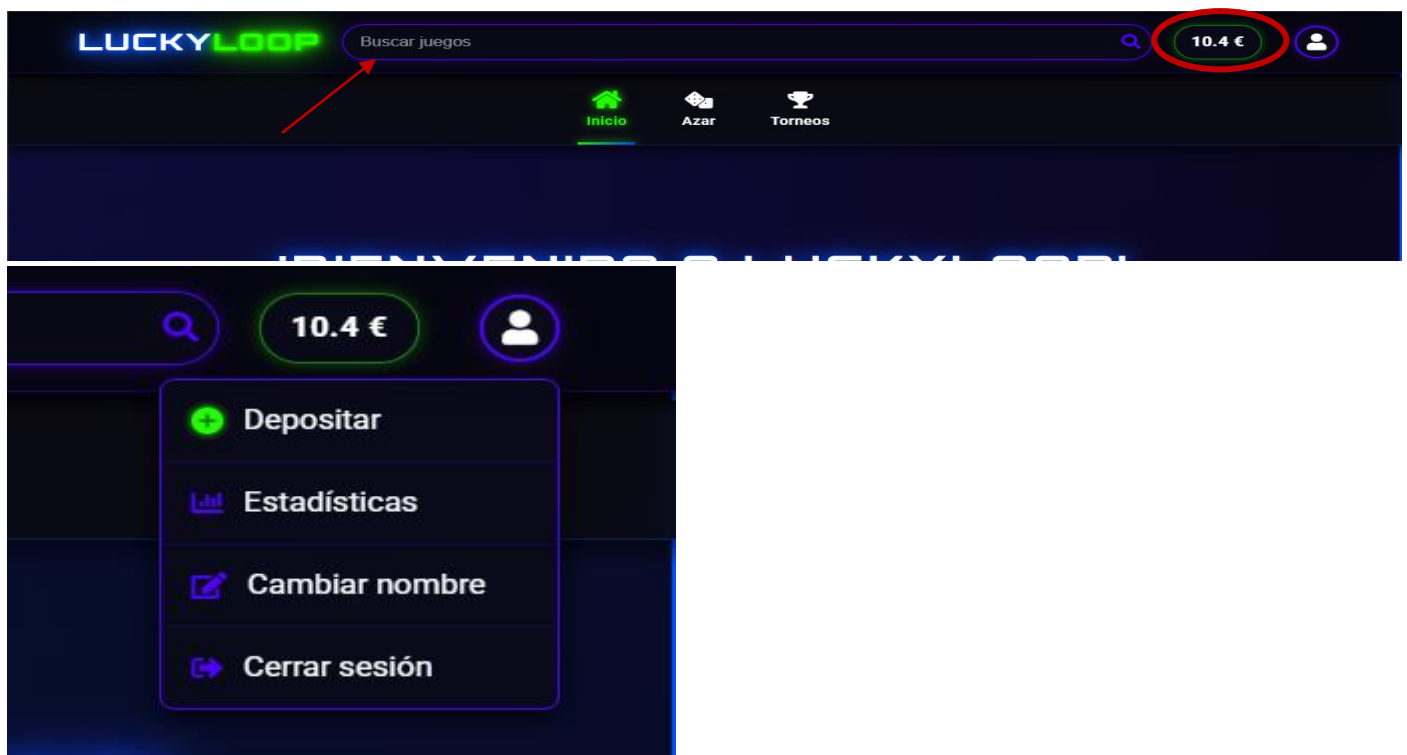
REGISTRARSE

¿Ya tienes cuenta? [Iniciar sesión](#)

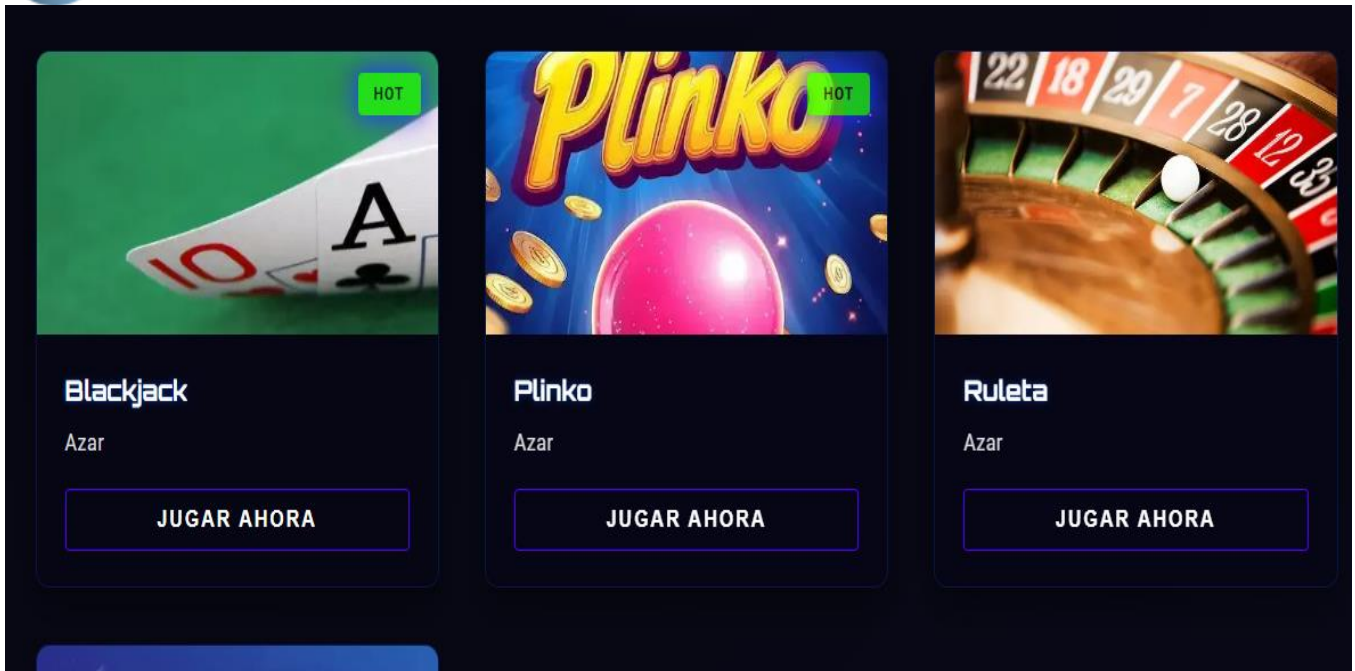
- Cuando inicies sesión nuestra plataforma tiene un menú principal hecho para ti con los juegos.



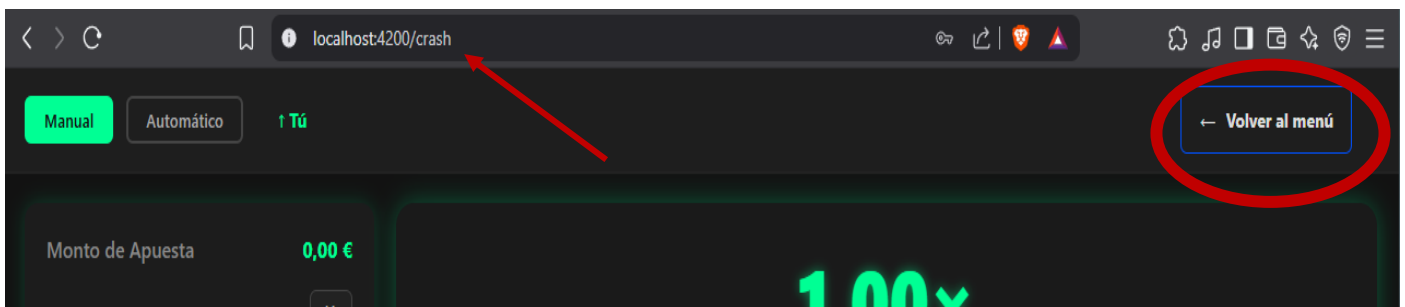
- La barra de búsqueda podrás filtrar los juegos que quieras buscar, y tendrás tu balance como usuario (dinero depositado para jugar). Y en tu usuario tendrás varias opciones interesantes.



- Si el usuario quiere jugar a un juego determinado solo tiene que pulsar el botón de jugar y será redirigido al juego en específico



- Dentro del juego en específico podrás jugar siempre que tengas dinero depositado ya que este estará basado en tu balance personal de dinero. Y siempre podrás salir del juego, volver atrás al menú principal



- Cualquier usuario registrado que pueda jugar a nuestros juegos siempre podrá ver un ranking mundial, ¿A quién no le gusta ganar? Pueden acceder pinchando en ranking



Volver al
menú



RANKING DE BENEFICIOS

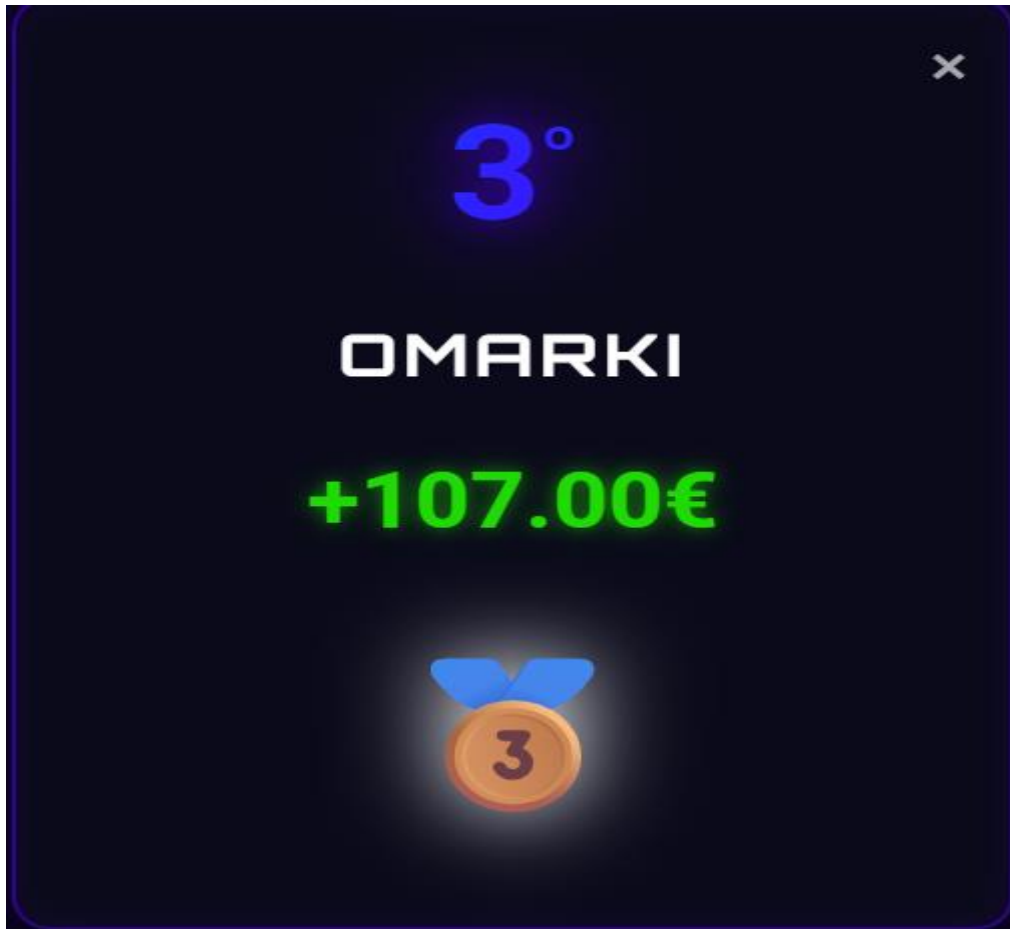


OTROS PARTICIPANTES

4 A

+19.20€

- El ranking estará compuesto por todos los usuarios registrados, y marcará el balance total que tienen en sus cuentas. Apostar y jugar para ser los mejores en nuestro ranking.



6.2 ARQUITECTURA DEL SISTEMA:

Nuestro proyecto sigue la arquitectura cliente-servidor que contiene el MVC, el cual el front anteriormente comentado contendría html,css, js y ts hecho con framework de angular.

Nuestro back estaría formado por una estructura de Symfony, un framework de php. Dentro de este, tenemos:

- entidades que representan las tablas de la base de datos y definen la estructura de nuestro back y maneja la aplicación.
- Controladores que se encargan de recibir las peticiones del cliente y procesarlas para devolverlas al front
- Servicios que serían las clases auxiliares que encapsulan la lógica del proyecto y mantiene todo limpio y reutilizable.

Authorize

Estadísticas

POST	/api/getEstadisticas	Obtiene las estadísticas del usuario actual	
POST	/api/getEstadisticas/juego/{id}	Obtiene las estadísticas de un juego específico	

Juego

GET	/api/getJuegos	Obtiene la lista de juegos disponibles	
GET	/api/getCategorias	Obtiene la lista de categorías disponibles	

Partida

POST	/api/finPartida	Inserta los datos de la partida	
------	-----------------	---------------------------------	--

Ranking

POST	/api/getRanking		
------	-----------------	--	--

Autenticación Autenticación

POST	/api/login	Obtener token JWT	
------	------------	-------------------	--

Usuario Usuario

POST	/api/usuario/emailToken	Genera un token a un usuario si su email existe y le envía un email	
GET	/api/usuario/comprobarToken/{token}	Comprueba si el token del usuario es válido	
POST	/api/usuario/cambiarPassword	Cambia la contraseña del usuario	
POST	/api/usuario/registrarse	Registra un nuevo usuario	
GET	/api/usuario/getSaldo	Obtener el saldo actual de un usuario por ID	
POST	/api/usuario/updateSaldo	Resta o suma saldo a un usuario	
POST	/api/usuario/cambiarNombre	Cambiar el nombre del usuario	

La imagen anterior contiene todo nuestro api/doc que contiene como manejamos las APIs en nuestro proyecto, las cuales están en back, y que las recoge después el front. Además, nos muestra cómo funciona cada endpoint, ejemplo una de las APIs y muestra lo que devuelven.

Juego

GET /api/getJuegos Obtiene la lista de juegos disponibles

Devuelve todos los datos de los juegos disponibles en el sistema

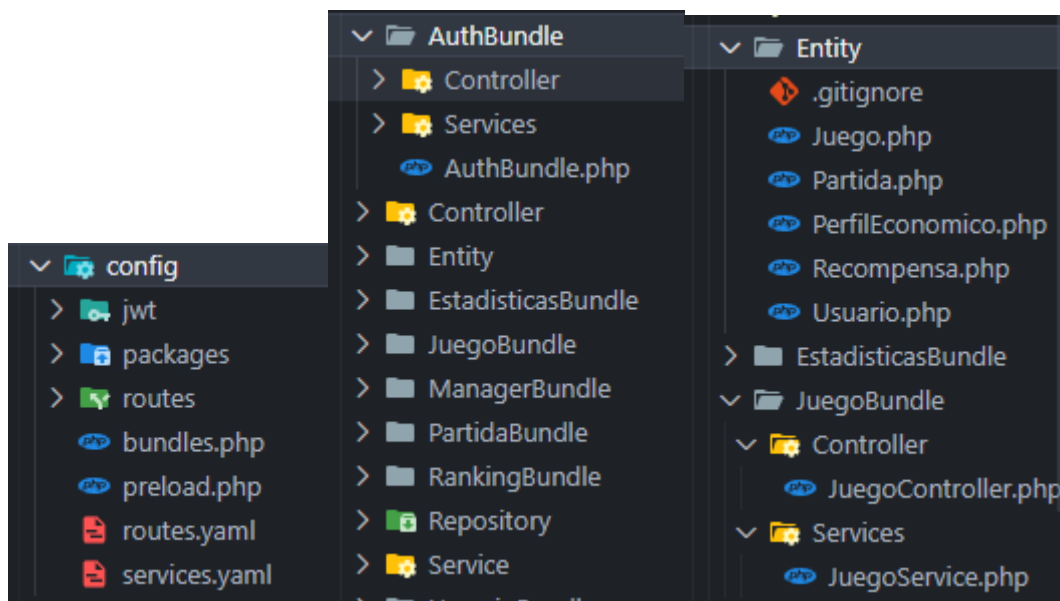
Parameters
No parameters

Try it out

Responses

Code	Description	Links
200	Lista de juegos devuelta correctamente Media type application/json Controls Accept header. Example Value Schema <pre>{ "status": "success", "data": [{ "id": 1, "name": "BlackJack", "image": "assets/images/blackjack.webp", "category": "Juego de Azar", "ishot": true, "url": "/blackjack" }] }</pre>	No links
404	No se encontraron juegos Media type application/json Example Value Schema <pre>{ "message": "No se encontraron juegos" }</pre>	No links

La estructura de nuestro backend estaría formada por configuración en el que tenemos la configuración del JWT, por ejemplo, las rutas y los paquetes instalados. Y en nuestra estructura las entidades están divididas en controller y servicios como se explicó anteriormente.



Juego

GET `/api/getJuegos` Obtiene la lista de juegos disponibles

Devuelve todos los datos de los juegos disponibles en el sistema

Parameters [Try it out](#)

No parameters

Responses

Code	Description	Links
200	Lista de juegos devuelta correctamente	No links
	<p>Media type: <input type="text" value="application/json"/></p> <p>Controls Accept header.</p> <p>Example Value Schema</p> <pre>{ "status": "success", "data": [{ "id": 1, "name": "BlackJack", "image": "assets/images/blackjack.webp", "category": "Juego de Azar", "isHot": true, "url": "/blackjack" }] }</pre>	
404	No se encontraron juegos	No links
	<p>Media type: <input type="text" value="application/json"/></p> <p>Example Value Schema</p> <pre>{ "message": "No se encontraron juegos" }</pre>	

Diagrama ER:

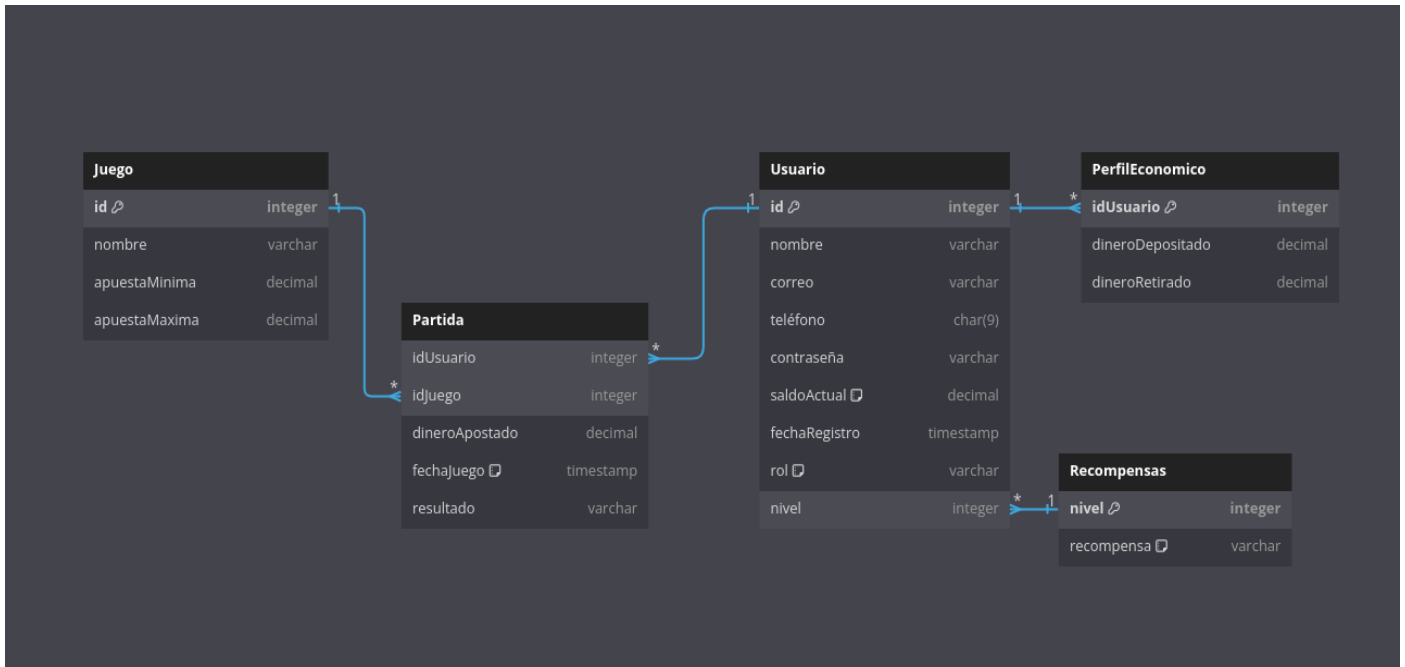
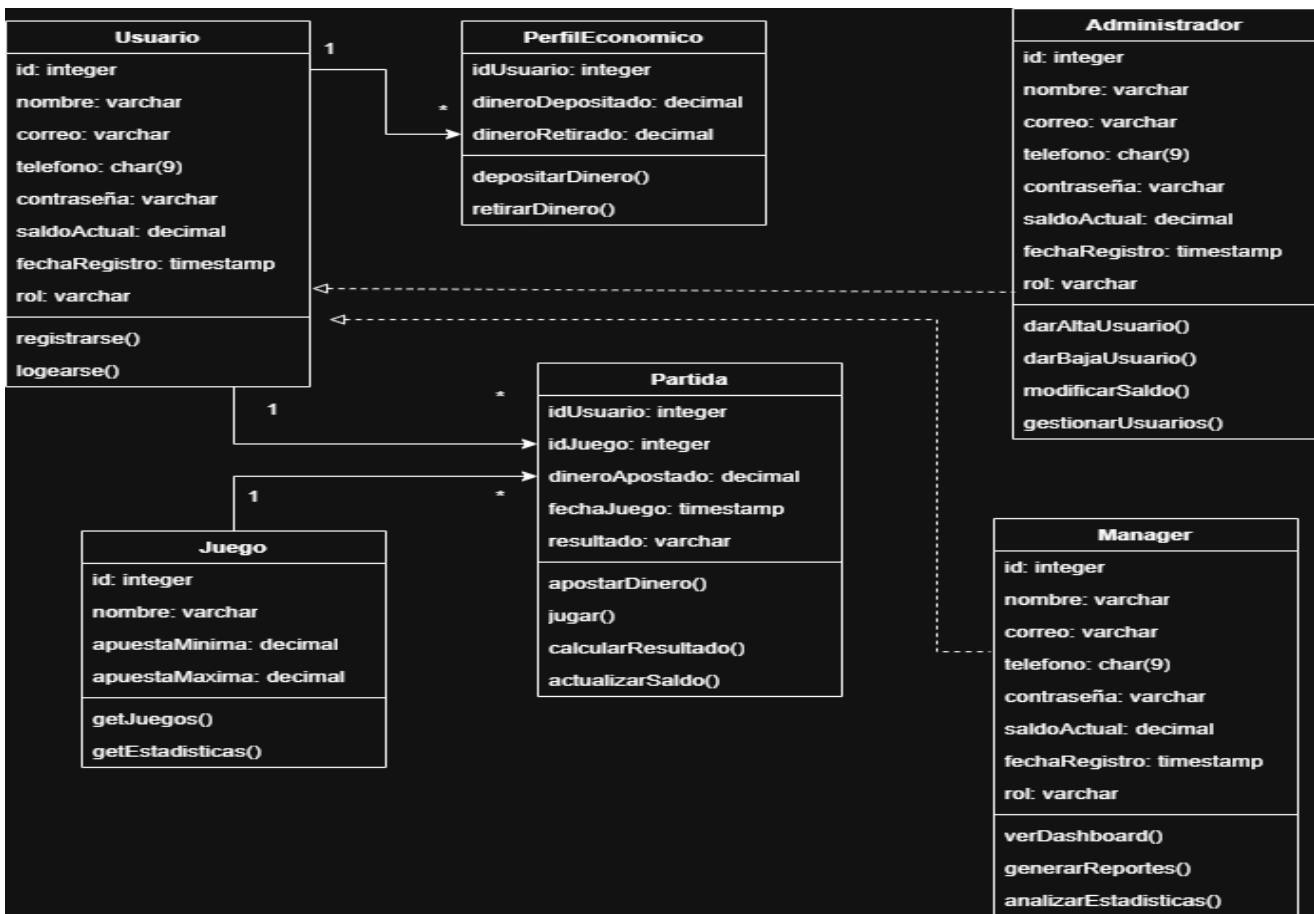


Diagrama de clases:



En ellos podemos observar:

- usuario puede registrarse o logearse en nuestro sistema, y desde aquí puede iniciar una partida y poder apostar, jugar y tendrá su saldo actualizado al momento de jugar, así como su balance.
- Que el menú principal el cual cargará los juegos tiene getJuegos() y las categorías de estos. Y que en estos juegos tienen partidas que son jugadas por estos usuarios.
- El usuario siempre podrá ver su perfil económico así con sus estadísticas de juegos, balance y luego tener el ranking a su disposición.
- Mientras que el mánager y administrador tienen sus funciones, así como el control del dashboard en el mánager, y gestión de usuarios sobre todo en el administrador, que sería el perfil más técnico.

6.4 DISEÑO DE LA BASE DE DATOS: ESQUEMA Y TABLAS:

- Juego:

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
nombre	varchar(50)	NO	UNI	NULL	
apuesta_minima	int	YES		1	
apuesta_maxima	int	NO		NULL	
categoria	varchar(50)	YES		NULL	

- Usuario:

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
nivel_id	int	YES	MUL	NULL	
roles	json	NO		NULL	
nombre	varchar(30)	NO		NULL	
email	varchar(50)	NO	UNI	NULL	
telefono	varchar(9)	NO	UNI	NULL	
password	varchar(80)	YES		NULL	
saldo_actual	double	YES		0	
fecha_registro	datetime	YES		NULL	
token_password	varchar(50)	YES		NULL	

- Partida:

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
usuario_id	int	NO	MUL	NULL	
juego_id	int	NO	MUL	NULL	
dinero_apostado	double	NO		NULL	
fecha_juego	datetime	YES		NULL	
resultado	varchar(10)	NO		NULL	
beneficio	double	NO		NULL	

- Perfil económico:

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
usuario_id	int	NO	UNI	NULL	
dinero_depositado	double	YES		0	
dinero_retirado	double	YES		0	

- Recompensa:

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
nivel	int	NO		NULL	
recompensa	int	YES		0	

7. DESARROLLO DE LA APLICACION:

7.1 TECNOLOGIAS Y HERRAMIENTAS UTILIZADAS (LENGUAJES PROGRAMACION, FRAMEWORKS, BASES DE DATOS... ETC)

- Lenguajes de programación:
 - Html, css y javascript para la parte frontend
 - PHP para la parte backend y lógica de nuestro proyecto
 - MySQL para almacenar toda la información de nuestra plataforma
- Frameworks:
 - Angular acompañando la parte frontend donde llamabamos a las APIs del back con html, css, javascript y typescript.
 - Symfony para hacer entidades, controllers y servicios de nuestra lógica del proyecto, el cual trabaja con php
- Herramientas:
 - Github fue nuestro repositorio central para todo el código conjunto del proyecto, para trabajar de forma colaborativo entre nosotros.
 - JWT (JSON Web Tokens), lo implementamos para la autenticación y seguridad de nuestro proyecto, estos nos permitio proteger rutas del back y garantizar datos de usuarios.
 - Stripe para gestionar los depósitos y retiradas de dinero de los usuarios en nuestra plataforma. Nos permitió procesar pagos de forma segura y fiable y nos ayudó siendo una experiencia sencilla para el usuario.
 - VsCode: ha sido el entorno de trabajo principal de nuestro desarrollo del proyecto, por sus extensiones y buen rendimiento.
 - Docker: al principio nos ayudó a trabajar con symfony y mysql dentro de este, pero más tarde decidimos que para la velocidad de trabajo y mayor rendimiento en este lo mejor era trabajar de forma local con symfony y mysql, lo cual nos ayudó a rendir más.

7.2 DESCRIPCION DE LAS PRINCIPALES FUNCIONALIDADES IMPLEMENTADAS:

- Registro de usuarios:

Backend:

```
public function registrar(Request $request) {
    $data = json_decode($request->getContent(), true);
    $nombreUsuario = $data['nombreUsuario'];
    $correoElectronico = $data['correoElectronico'];
    $contrasena = password_hash($data['contrasena'], PASSWORD_DEFAULT);
    $telefono = $data['telefono'];

    if ($data['contrasena'] !== $data['repetirContrasena']) {
        return new JsonResponse([
            'message' => 'Las contraseñas no coinciden'
        ], 400);
    }
}
```

Frontend:

```
export class AuthService {
    private apiUrl = 'http://localhost:8000/api';
```

```
registrar(nombreUsuario: string, contrasena: string, repetirContrasena: string, telefono: string, correoElectronico: string) {
    return this.http.post(`${this.apiUrl}/usuario/registrar`, { nombreUsuario, contrasena, repetirContrasena, telefono, correoElectronico })
}
```

- Login de usuarios:

Backend:

```
api_login:
  path: /api/login
  methods: ['POST']

app_logout:
  path: /logout
  methods: ['GET']
```

- Ver estadísticas del usuario:

Backend

```
public function getEstadisticas()
{
    $usuario = $this->security->getUser();

    $juegos=$this->em->getRepository(Juego::class)->findAll();
    $arrayJuegos=[];
    foreach ($juegos as $juego) {
        $arrayJuegos[]= [
            'id'=>$juego->getId(),
            'nombre'=>$juego->getNombre()
        ];
    }

    $partidas = $this->em->getRepository(Partida::class)->findBy(['usuario' => $usuario]);
    $numeroPartidas=count($partidas);
    $derrotas=0;
    $victorias=0;
    $empates=0;

    foreach ($partidas as $partida) {
        if($partida->isDerrota()){
            $derrotas++;
        }elseif ($partida->isVictoria()) {
            $victorias++;
        }else{
            $empates++;
        }
    }

    $saldoActual=$usuario->getSaldoActual();

    $perfilEconomico=$this->em->getRepository(PerfilEconomico::class)->findOneBy(['usuario'=>$usuario]);
    $retirado=$perfilEconomico->getDineroRetirado();
    $depositado=$perfilEconomico->getDineroDepositado();

    $beneficio=($retirado+$saldoActual)-$depositado;

    return new JsonResponse([
        'total_partidas' => $numeroPartidas,
        'partidas_ganadas'=>$victorias,
        'partidas_perdidas'=>$derrotas,
        'partidas_empatadas'=>$empates,
        'beneficio_total'=>$beneficio,
        'retirado'=>$retirado,
        'depositado'=>$depositado,
        'juegos'=>$arrayJuegos
    ]);
}
```

```
public function getEstadisticasPorJuego(int $juegoId)
{
    $usuario = $this->security->getUser();

    if (!$usuario) {
        throw new AuthenticationException('Usuario no autenticado');
    }

    $partidas = $this->em->getRepository(Partida::class)->findBy([
        'usuario' => $usuario,
        'juego' => $juegoId
    ]);

    $numeroPartidas = count($partidas);
    $partidasGanadas = 0;
    $partidasEmpatadas = 0;
    $partidasPerdidas = 0;
    $beneficioTotal = 0;

    foreach ($partidas as $partida) {
        if ($partida->isVictoria()) {
            $partidasGanadas++;
            $beneficioTotal += $partida->getBeneficio();
        } elseif ($partida->isDerrota()) {
            $partidasPerdidas++;
            $beneficioTotal -= $partida->getDineroApostado();
        } else {
            $partidasEmpatadas++;
        }
    }

    return new JsonResponse([
        'juego_id' => $juegoId,
        'total_partidas' => $numeroPartidas,
        'partidas_ganadas' => $partidasGanadas,
        'partidas_perdidas' => $partidasPerdidas,
        'partidas_empatadas' => $partidasEmpatadas,
        'beneficio_total' => $beneficioTotal
    ]);
}
```

Frontend

```
export class EstadisticasService {
    private apiUrl = 'http://localhost:8000/api';

    constructor(private http: HttpClient) { }

    private getAuthHeaders(): HttpHeaders {
        const token = localStorage.getItem('auth_token');
        return new HttpHeaders({
            'Content-Type': 'application/json',
            'Authorization': `Bearer ${token}`
        });
    }

    getEstadisticas(): Observable<EstadisticasResponse> {
        const headers = this.getAuthHeaders();
        return this.http.post<EstadisticasResponse>(`${this.apiUrl}/getEstadisticas`, {}, { headers });
    }

    getEstadisticasJuego(juegoId: number): Observable<EstadisticasJuegoResponse> {
        const headers = this.getAuthHeaders();
        return this.http.post<EstadisticasJuegoResponse>(`${this.apiUrl}/getEstadisticas/juego/${juegoId}`, {}, { headers });
    }
}
```

- Cambio de nombre del usuario:

```
public function cambiarNombre(Request $request): JsonResponse
{
    //buscar usuario
    $usuario = $this->security->getUser();

    $data = json_decode($request->getContent(), true);
    $nuevoNombre = $data['nombre'] ?? null;

    if (!$nuevoNombre || trim($nuevoNombre) === '') {
        return new JsonResponse([
            'error' => 'El campo de nombre no puede estar vacío. Por favor, ingrese un nombre válido.'
        ], 400);
    }

    if ($usuario->getNombre() == $nuevoNombre) {
        return new JsonResponse([
            'error' => 'El nombre proporcionado es idéntico al actual. Por favor, elija un nombre diferente.'
        ], 400);
    }

    //cambiar nombre
    $usuario->setNombre($nuevoNombre);
    $this->entityManager->persist($usuario);
    $this->entityManager->flush();

    return new JsonResponse([
        'mensaje' => '¡Nombre actualizado con éxito! Su nuevo nombre de usuario ha sido guardado.',
        'nuevoNombre' => $nuevoNombre,
        'codigo' => 200,
        'fechaActualizacion' => (new \DateTime())->format('Y-m-d H:i:s')
    ]);
}
```

Frontend:

```
export class CambiarNombreService {
    private apiUrl = 'http://localhost:8000/api';

    constructor(private http: HttpClient) { }

    private getAuthHeaders(): HttpHeaders {
        const token = localStorage.getItem('auth_token');
        return new HttpHeaders({
            'Content-Type': 'application/json',
            'Authorization': `Bearer ${token}`
        });
    }

    cambiarNombre(nuevoNombre: string): Observable<any> {
        const headers = this.getAuthHeaders();
        return this.http.post(
            `${this.apiUrl}/usuario/cambiarNombre`,
            { nombre: nuevoNombre },
            { headers }
        );
    }
}
```

- Recuperar password:

Backend:

```
public function emailToken(Request $request) {
    $userRepo = $this->entityManager->getRepository(Usuario::class);
    $data = json_decode($request->getContent(), true);
    $email = $data['email'];
    $usuario = $userRepo->findOneBy(['email' => $email]);

    if (!$usuario) {
        return new JsonResponse(['error' => 'Email no encontrado'], 400);
    }

    $token = bin2hex(random_bytes(32));
    $usuario->setTokenPassword($token);
    $this->entityManager->persist($usuario);
    $this->entityManager->flush();

    $mensaje = '
    <!DOCTYPE html>
    <html>
    <head>
        <title>Recuperar Contraseña</title>
    </head>
    <body>
        <p>Hola,</p>
        <p>Para recuperar tu contraseña, haz click en el siguiente enlace:</p>
        <p><a href="http://localhost:4200/cambiarPassword?token=' . $token . '">Recuperar Contraseña</a></p>
        <p>Si no solicitaste un cambio de contraseña, puedes ignorar este mensaje.</p>
    </body>
    </html>';

    $this->mailer->enviarEmail($email, 'Recuperar Contraseña', $mensaje, 'Recuperar Contraseña');

    return new JsonResponse([
        'message' => 'Email enviado correctamente'
    ]);
}

public function cambiarPassword(Request $request){
    $data = json_decode($request->getContent(), true);
    $user= $this->entityManager->getRepository(Usuario::class)->findOneBy(['email' => $data['email']]);

    if (!$user) {
        return new JsonResponse([
            'message' => 'Email no encontrado'
        ], 400);
    }

    $password = $data['password'];
    $confirmPassword = $data['confirmPassword'];

    if($password !== $confirmPassword) {
        return new JsonResponse([
            'message' => 'Las contraseñas no coinciden'
        ], 400);
    }

    if(password_verify($password, $user->getPassword())) {
        return new JsonResponse([
            'message' => 'La nueva contraseña no puede ser igual a la anterior'
        ], 400);
    }

    $user->setPassword(password_hash($password, PASSWORD_DEFAULT));
    $user->setTokenPassword('');
    $this->entityManager->persist($user);
    $this->entityManager->flush();

    return new JsonResponse([
        'message' => 'Contraseña actualizada correctamente'
    ], 200);
}
```

Frontend:

```
export class AuthService {
    private apiUrl = 'http://localhost:8000/api';
```

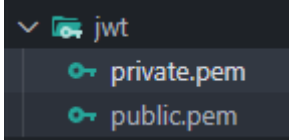


```
comprobarEmail(email: string) {  
    return this.http.post(`${this.apiUrl}/usuario/emailToken`, { email })  
}
```

```
cambiarPassword(token: string, password: string, email: string, confirmPassword: string) {  
    return this.http.post(`${this.apiUrl}/usuario/cambiarPassword`, { token, password, email, confirmPassword })  
}
```

- JWT para el control y seguridad:

Backend:



```
api:  
  pattern: ^/api  
  stateless: true  
  provider: app_user_provider  
  json_login:  
    check_path: api_login  
    username_path: email  
    password_path: password  
    success_handler: lexik_jwt_authentication.handler.authentication_success  
    failure_handler: lexik_jwt_authentication.handler.authentication_failure  
  jwt: ~
```

```
# .env  
STRIPE_PUBLIC_KEY="pk_test_51RA7GDATTvNshzGt9tfwJ2zwlAFwufXmZvtHRk1vMtSiLepXFmjdMLKQpepgptoq7JsqvpCCa0uAub5vfj8babr00nawAFq91"  
STRIPE_SECRET_KEY="sk_test_51RA7GDATTvNshzGtOMfC7fBiefAEkmfU8qT7AW3ZmGcDPauRooLQZ18lf2YWixugNjiNkCRHYELGgaXB0anTwNwb00AADVwWz0"  
###> lexik/jwt-authentication-bundle ###  
JWT_SECRET_KEY=%kernel.project_dir%/config/jwt/private.pem  
JWT_PUBLIC_KEY=%kernel.project_dir%/config/jwt/public.pem  
JWT_PASSPHRASE=7f6b1eaddb49fabbeed9a08842871e16f7b9a1d96271083b279100e0088ab6dd  
###< lexik/jwt-authentication-bundle ###
```

Frontend:

```
comprobarToken(token: string) {  
    return this.http.get(`${this.apiUrl}/usuario/comprobarToken/${token}`)  
}
```

- Depositar dinero con Stripe:

Backend:


```
#[Route('/create-payment-intent', name: 'create_payment_intent', methods: ['POST'])]
public function createPaymentIntent(Request $request): JsonResponse
{
    $stripe = new StripeClient($_ENV['STRIPE_SECRET_KEY']);

    $data = json_decode($request->getContent(), true);
    $amount = $data['amount'];

    try {
        $paymentIntent = $stripe->paymentIntents->create([
            'amount' => $amount,
            'currency' => 'eur',
            'payment_method_types' => ['card'],
        ]);

        return $this->json([
            'clientSecret' => $paymentIntent->client_secret,
        ]);
    } catch (\Exception $e) {
        return $this->json([
            'error' => $e->getMessage(),
        ], 400);
    }
}
```

Frontend:

```
try {
    // 1. Crear Payment Intent
    const paymentIntentResponse = await fetch('http://localhost:8000/create-payment-intent', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'Authorization': 'Bearer ' + localStorage.getItem('auth_token')
        },
        body: JSON.stringify({ amount: this.amount * 100 })
    });

    if (!paymentIntentResponse.ok) {
        throw new Error('Error al crear la intención de pago');
    }

    const { clientSecret } = await paymentIntentResponse.json();

    // 2. Confirmar pago con Stripe
    const { paymentIntent, error } = await this.stripe.confirmCardPayment(clientSecret, {
        payment_method: { card: this.cardElement }
    });

    if (error) throw error;

    // 3. Actualizar saldo local
    this.saldoService.setSaldo(this.amount, true, false).subscribe({
        next: (response) => {
            alert(`✅ Depósito exitoso, Nuevo saldo: €${response.nuevoSaldo}`);
            this.loading = false;
        },
        error: (err) => {
            throw new Error(err.error?.message || 'Error actualizando saldo');
        }
    });
} catch (error: any) {
    alert(`❌ Error al depositar`);
    this.loading = false;
}
```

- Apostar para juegos:

Frontend:

```
// Métodos para controles de apuesta
setMode(mode: string): void {
  this.mode = mode;
}

halfBet(): void {
  this.betAmount = Math.floor(this.betAmount / 2);
  this.calculatePotentialWin();
}

doubleBet(): void {
  if (this.betAmount * 2 ≤ this.playerBalance) {
    this.betAmount = this.betAmount * 2;
  } else {
    this.betAmount = this.playerBalance;
  }
  this.calculatePotentialWin();
}

updateCashoutAt(value: number): void {
  if (value ≥ 1.01) {
    this.cashoutAt = Number(value.toFixed(2));
  }
}

calculatePotentialWin(): void {
  this.potentialWin = this.betAmount * this.multiplier;
}
```

* setMode cambia el modo del juego, half reduce y double doblar apuesta, y calcular ganancias

- Saldo del usuario y actualizarlo:

Backend:

```
public function updateSaldo(Request $request): JsonResponse {
    $data = json_decode($request->getContent(), true);

    if (!isset($data['dinero'])) {
        return new JsonResponse(['message' => 'El parámetro "dinero" es requerido.'], 400);
    }

    $dinero = $data['dinero'];
    $usuario = $this->security->getUser();

    if ($data['deposito'] != true && $data['retirada'] != true) {
        //RESTAR SALDO
        if ($dinero < 0) {
            if ($usuario->getSaldoActual() < abs($dinero)) {
                return new JsonResponse(['message' => 'No tienes tanto saldo.'], 400);
            }

            $usuario->setSaldoActual($usuario->getSaldoActual() - abs($dinero));
        } else {
            //SUMAR SALDO
            $usuario->setSaldoActual($usuario->getSaldoActual() + $dinero);
        }
    } elseif ($data['deposito'] = true && $data['retirada'] != true) {
        $usuario->setSaldoActual($usuario->getSaldoActual() + $dinero);

        //Insertar en dineroDepositado
        $perfilEconomico = $this->entityManager->getRepository(PerfilEconomico::class)->findOneBy(['usuario' => $usuario]);
        $perfilEconomico->setDineroDepositado($perfilEconomico->getDineroDepositado() + $dinero);
    } else {
        $usuario->setSaldoActual($usuario->getSaldoActual() - $dinero);

        //Insertar en dineroRetirado
        $perfilEconomico = $this->entityManager->getRepository(PerfilEconomico::class)->findOneBy(['usuario' => $usuario]);
        $perfilEconomico->setDineroRetirado($perfilEconomico->getDineroRetirado() + $dinero);
    }

    $this->entityManager->persist($usuario);
    $this->entityManager->flush();

    return new JsonResponse([
        'nuevoSaldo' => $usuario->getSaldoActual()
    ]);
}

public function getSaldo(){
    $usuario = $this->security->getUser();

    if (!$usuario) {
        return new JsonResponse([
            'error' => 'Usuario no encontrado',
        ], 400);
    }

    return new JsonResponse([
        'saldo' => $usuario->getSaldoActual()
    ]);
}
```

```
export class SaldoService {
  private apiUrl = 'http://localhost:8000/api'

  constructor(
    private http: HttpClient
  ) {}

  getSaldo(): Observable<any> {
    const token = localStorage.getItem('auth_token');
    const headers = new HttpHeaders({
      'Authorization': `Bearer ${token}`
    });

    return this.http.get(`${this.apiUrl}/usuario/getSaldo`, { headers });
  }

  setSaldo(dinero: number, deposito: boolean=false, retirada: boolean=false ): Observable<any> {
    const token = localStorage.getItem('auth_token');
    const headers = new HttpHeaders({
      'Authorization': `Bearer ${token}`,
      'Content-Type': 'application/json'
    });

    const body = {
      dinero: dinero,
      deposito: deposito,
      retirada: retirada
    };

    return this.http.post(`${this.apiUrl}/usuario/updateSaldo`, body, { headers });
  }
}
```