MÓDULO DE PROYECTO

ALONSO DE AVELLANEDA

Alonso de Avellaneda

Técnico Superior en Desarrollo de Apps Web

“GameTracker”

Autor: “Carlos Martín Salvatierra”

Profesor tutor: “María Luz Elola”

Alcalá de Henares, Junio de “2025”

1. Introducción y Justificación

1.1. Descripción del Proyecto: GameTracker

El presente proyecto, denominado **GameTracker**, consiste en el desarrollo de una plataforma web integral diseñada específicamente para entusiastas de los videojuegos. Se concibe como un espacio centralizado donde los usuarios pueden gestionar de manera eficiente su biblioteca de juegos, realizar un seguimiento detallado de su progreso individual, interactuar con una comunidad de jugadores con intereses similares y descubrir nuevos títulos acordes a sus preferencias.

GameTracker busca ofrecer una solución robusta y atractiva que combine funcionalidades de catalogación, seguimiento, socialización y descubrimiento, enriqueciendo la experiencia global del videojugador.

1.2. Finalidad del Proyecto

La finalidad principal de GameTracker es **mejorar y centralizar la experiencia de gestión y disfrute de videojuegos para los usuarios**. En un panorama donde los jugadores a menudo interactúan con múltiples plataformas, bibliotecas dispersas y comunidades fragmentadas, GameTracker aspira a servir como un nexo unificador.

El proyecto busca proporcionar una herramienta útil que no solo organice la actividad lúdica del usuario, sino que también fomente la conexión social y el descubrimiento continuo, añadiendo una capa de gamificación interna a través de un sistema de logros propio.

1.3. Objetivos del Proyecto

Una vez implementado y puesto en marcha, GameTracker permitirá a los usuarios:

- *Gestionar su biblioteca de videojuegos:* Registrar títulos de diversas plataformas, marcar su estado (pendiente, jugando, completado, etc.), añadir valoraciones personales y organizar juegos en listas personalizadas.
- *Realizar un seguimiento del progreso:* Visualizar estadísticas personales, juegos completados, tiempo invertido (si se implementa) y logros obtenidos dentro de la plataforma.
- *Descubrir nuevos títulos:* Explorar un extenso catálogo de juegos mediante búsquedas, filtros avanzados, un asistente de descubrimiento basado en preferencias y una función de "ruleta" para selecciones aleatorias.
- *Interactuar con otros jugadores:* Establecer conexiones de amistad, participar en chats privados, crear y unirse a grupos temáticos con sus propios foros de discusión o chats.
- *Obtener recompensas y reconocimiento:* Desbloquear logros internos de GameTracker por la actividad y participación en la plataforma, fomentando la exploración de sus funcionalidades.
- *Disfrutar de contenido lúdico adicional:* Acceder a una "Zona Arcade" con minijuegos relacionados con el mundo de los videojuegos.
- *Personalizar su experiencia:* Configurar un perfil de usuario detallado, incluyendo avatar, banner, biografía y preferencias de juego.
- *Garantizar la seguridad de su cuenta:* Mediante un sistema de autenticación robusto que incluye verificación de correo electrónico y autenticación de dos factores (2FA).

1.4. Motivación

La elección de desarrollar GameTracker surge de una combinación de pasión personal por el mundo de los videojuegos y la identificación de una oportunidad para crear una herramienta que responda a necesidades comunes entre los jugadores. Actualmente, los aficionados a los videojuegos suelen recurrir a múltiples aplicaciones y plataformas para gestionar sus colecciones, interactuar con amigos o descubrir nuevos títulos. Esta fragmentación puede resultar ineficiente y, en ocasiones, desconectada.

Se considera que un proyecto de estas características no solo cumple con los requisitos académicos del ciclo, sino que también tiene el potencial de convertirse en una herramienta valiosa y apreciada por la comunidad de jugadores.

2. Estudio de la Viabilidad del Proyecto

Este apartado analiza la viabilidad del proyecto GameTracker desde las perspectivas económica, legal y temporal, con el objetivo de justificar su desarrollo y ejecución.

2.1. Viabilidad Económica

2.1.1. Estimación de Costos (*)

Para el desarrollo y despliegue inicial del proyecto GameTracker, se han considerado los siguientes recursos y sus costos aproximados:

1. Hosting y Despliegue:
    - Backend (Node.js API): Se prevé el uso de plataformas PaaS (Platform as a Service) como Render o Heroku, que ofrecen niveles gratuitos suficientes para el desarrollo, pruebas y un lanzamiento inicial con tráfico moderado. Un costo estimado para escalar a un plan básico podría oscilar entre 5€ - 20€ mensuales, dependiendo del proveedor y los recursos necesarios (CPU, RAM, ancho de banda).
    - Frontend (Astro - Estático o SSR): Plataformas como Vercel, Netlify o Render (para sitios estáticos) ofrecen generosos niveles gratuitos. Para SSR con Astro (usando el adaptador de Node.js), los costos serían similares al backend.
    - Base de Datos (MongoDB): Se utilizará MongoDB Atlas, que proporciona un clúster gratuito (M0 Sandbox) adecuado para desarrollo y aplicaciones de pequeña escala. Para necesidades de producción mayores, los planes de pago comienzan alrededor de 9€ mensuales (plan M2).
2. Dominio:
    - Un nombre de dominio personalizado (ej. `www.gametracker.com`) tendría un costo anual aproximado de 10€ - 20€, dependiendo del registrador y la extensión (.com, .es, .app, etc.).
3. Herramientas de Desarrollo y Software:
    - Entorno de Desarrollo Integrado (IDE): Visual Studio Code, de uso gratuito.
    - Tecnologías Core (Node.js, Astro, React, Express): Todas son de código abierto y gratuitas.
    - Gestor de Base de Datos: MongoDB Compass, de uso gratuito.
    - API Externa (RAWG.io): La API de RAWG (https://rawg.io/apidocs) ofrece un nivel gratuito con limitaciones en el número de peticiones. Para un uso intensivo que exceda estos límites, sería necesario considerar sus planes de suscripción, aunque para la escala de este proyecto, el nivel gratuito es suficiente.

- Servicio de Correo Electrónico (Nodemailer): Para el envío de correos de verificación, notificaciones, etc. Durante el desarrollo se puede usar Ethereal (gratuito para pruebas) o una cuenta de Gmail (con limitaciones). Para producción, servicios como SendGrid o Mailgun ofrecen niveles gratuitos con un volumen considerable de envíos, y planes de pago a partir de **15€ - 30€ mensuales** si se supera dicho volumen.

4. Licencias de Software Adicional:
   - La mayoría de las librerías y frameworks utilizados son de código abierto (MIT, Apache, etc.), por lo que no incurren en costos de licencia directos (ver sección 2.2.2).
5. Recursos Humanos:
   - Al tratarse de un proyecto académico individual, no se contabiliza un costo salarial directo. Sin embargo, se estima un esfuerzo de desarrollo considerable (ver sección 2.3).

- *Costo Total Estimado Inicial (Anual):*
  - Considerando un despliegue básico en niveles gratuitos o de bajo costo, el gasto principal inicial sería el dominio: **aproximadamente 10€ - 20€ anuales**. Si se requiere escalar servicios o superar los límites de las APIs gratuitas, los costos mensuales podrían incrementarse a **25€ - 70€** o más, dependiendo del uso.

2.1.2. Retorno de la Inversión (ROI)

Dado el carácter académico y de aprendizaje del proyecto GameTracker, el retorno de la inversión (ROI) principal no se mide en términos económicos directos a corto plazo, sino en:

- *Adquisición y consolidación de conocimientos técnicos:* Aplicación práctica de tecnologías full-stack, diseño de bases de datos, desarrollo de APIs, gestión de proyectos, etc.
- *Desarrollo de un portafolio profesional:* El proyecto sirve como una pieza demostrable de las capacidades del desarrollador.
- *Potencial de aprendizaje y crecimiento personal.*

No obstante, si GameTracker evolucionara hacia un producto con una base de usuarios significativa, se podrían explorar las siguientes vías de monetización para generar un ROI económico:

- *Modelo Freemium:* Ofrecer funcionalidades básicas gratuitas y características premium (estadísticas avanzadas, mayor personalización, sin publicidad) bajo suscripción.
- *Publicidad no intrusiva:* Integrar espacios publicitarios discretos relevantes para la audiencia gamer.
- *Marketing de Afiliación:* Enlaces de afiliados a tiendas de videojuegos (ej. Steam, GOG) por los juegos listados o descubiertos en la plataforma.
- *Contenido Patrocinado o Destacado:* Para desarrolladores o editores de juegos que deseen promocionar sus títulos.

Plazo para Recuperar la Inversión:
En un escenario hipotético de monetización, recuperar la inversión inicial (principalmente tiempo de desarrollo y costos mínimos de infraestructura) dependería críticamente de la capacidad para atraer y retener una base de usuarios activa. Esto podría llevar desde varios meses hasta años, y requeriría inversión adicional en marketing y desarrollo continuo. Para el alcance actual del proyecto, el ROI se considera alcanzado con la finalización exitosa del mismo y la consecución de los objetivos de aprendizaje.

2.2. Viabilidad Legal

2.2.1. Cumplimiento Normativo

El proyecto GameTracker, al manejar datos de usuarios y operar en línea, debe considerar las siguientes normativas:

- Protección de Datos Personales (RGPD y LOPDGDD):
  - La plataforma recopila datos personales como nombre, nickname, dirección de correo electrónico (para registro, verificación, notificaciones, recuperación de contraseña), y potencialmente dirección IP (almacenada en los logs de sesión, como se ve en el modelo `Sesion.js`).
- Es **imprescindible** implementar:
  - Una Política de Privacidad clara que informe a los usuarios sobre qué datos se recopilan, cómo se usan, con quién se comparten (si aplica), sus derechos (acceso, rectificación, supresión, etc.) y cómo ejercerlos.
  - Mecanismos para obtener el consentimiento explícito para el tratamiento de datos donde sea necesario.
  - Medidas de seguridad para proteger los datos, como el hasheo de contraseñas (implementado con `bcrypt` según `INSTALL.md`) y la protección contra accesos no autorizados.
  - Considerar la gestión de la edad mínima para el registro si aplica.
- Política de Cookies:
  - Si la aplicación utiliza cookies (para sesiones, preferencias, análisis, etc.), se debe informar a los usuarios mediante un banner de cookies y una Política de Cookies detallada. Se debe permitir al usuario aceptar o rechazar cookies no esenciales.
- Comercio Electrónico:
  - Actualmente, GameTracker no implementa funcionalidades de comercio electrónico directo. Si en el futuro se introdujeran suscripciones de pago o ventas, se deberían cumplir las normativas específicas de comercio electrónico y contratación a distancia.
- Propiedad Intelectual de Terceros:
  - Se debe asegurar el respeto a los derechos de autor de los contenidos de la API de RAWG (descripciones, imágenes de juegos) según sus términos de servicio.
  - Para el contenido generado por el usuario (avatares, banners, mensajes), se deben establecer términos de uso que definan la propiedad y responsabilidad sobre dicho contenido.

2.2.2. Licencias de Software (*)

Las herramientas, frameworks y librerías principales utilizadas en el desarrollo de GameTracker se distribuyen bajo las siguientes licencias de código abierto, que permiten su uso (incluso comercial) sin costo de licencia, generalmente requiriendo la conservación de los avisos de copyright y licencia:

- Frontend (Cliente - `client/package.json`):
  - Astro: MIT License
  - React: MIT License
  - TailwindCSS: MIT License
  - Socket.IO Client: MIT License
  - Axios: MIT License
  - date-fns: MIT License
  - react-hot-toast: MIT License
  - framer-motion: MIT License
  - Otras dependencias listadas en package.json (se recomienda verificar cada una, pero las más comunes suelen ser MIT, Apache 2.0, BSD).
- Backend (Servidor - `server/package.json`):
  - Node.js: MIT License
  - Express: MIT License
  - Mongoose: MIT License
  - bcrypt: MIT License
  - jsonwebtoken: MIT License
  - cors: MIT License
  - cookie-parser: MIT License
  - dotenv: MIT License
  - socket.io (Server): MIT License
  - nodemailer: MIT License
  - speakeasy: MIT License
  - qrcode: MIT License
  - multer: MIT License
  - axios: MIT License
- Base de Datos:
  - MongoDB Community Server: Server Side Public License (SSPL). Esta licencia tiene implicaciones si se ofrece MongoDB como un servicio gestionado, pero para el uso como base de datos de una aplicación como GameTracker, no presenta restricciones significativas.
- Otros:
  - Visual Studio Code: Licencia MIT.
  - Iconos (ej. Font Awesome): Si se utiliza Font Awesome Free, su licencia permite uso gratuito con atribución.

El uso de estas herramientas bajo sus respectivas licencias es compatible con el desarrollo del proyecto.

2.2.3. Propiedad Intelectual (*)

- Originalidad del Proyecto:
  - La originalidad de GameTracker radica en su integración única de gestión de juegos, un sistema de logros interno, una "Zona Arcade" y funcionalidades sociales avanzadas (chat/grupos) en una sola plataforma. Si bien se inspira en conceptos conocidos, la arquitectura técnica específica y la implementación del código fuente que amalgaman estas características de forma cohesiva constituyen la aportación distintiva del proyecto.
- Uso de Recursos de Terceros:
  - Código Fuente: Se utilizan librerías y frameworks de código abierto, como se detalla en la sección anterior (2.2.2). Su uso se realiza respetando los términos de sus licencias (principalmente MIT), que generalmente permiten la incorporación en proyectos derivados sin restricciones significativas más allá de la conservación de los avisos de licencia.
  - **Contenido de Juegos (API de RAWG):** La información de juegos (nombres, descripciones, imágenes, géneros, plataformas, fechas de lanzamiento, ratings) se obtiene de la API de RAWG.io. Su uso está sujeto a los Términos de Servicio de RAWG, que deben ser consultados y respetados, incluyendo posibles requisitos de atribución o limitaciones de uso.
  - Imágenes y Multimedia:
    - El favicon (`favicon.svg`) y la imagen de login (`Imagen_Login.png`) se asumen como creaciones originales del autor o recursos con licencia adecuada para su uso.
    - Los iconos utilizados para la interfaz (ej. logros, interfaz general) provienen de librerías como Font Awesome (versión gratuita), cuyo uso está permitido bajo su licencia (generalmente requiere atribución si no se adquiere una licencia Pro).
    - El contenido subido por los usuarios (avatares, banners de perfil) es responsabilidad de los mismos. La plataforma debería incluir en sus Términos de Servicio cláusulas que indiquen que el usuario posee los derechos necesarios sobre el contenido que sube y otorga a GameTracker una licencia para mostrarlo.
    - Textos: Aparte de los textos de la interfaz, que son originales, las descripciones de juegos provienen de la API de RAWG.
- Licencia del Proyecto GameTracker:
  - Se tiene previsto aplicar la **Licencia MIT** al código fuente del proyecto GameTracker.
  - Esta licencia permisiva permitirá los siguientes usos:
    - Uso comercial y no comercial.
    - Modificación y distribución del software.
    - Uso privado.
  - La única obligación principal es **incluir el aviso de copyright original y el texto de la licencia MIT** en todas las copias o porciones sustanciales del software. No impone restricciones de "copyleft" (obligación de que las obras derivadas también sean MIT).
  - Para la documentación (como esta memoria, README.md, INSTALL.md, `DOCUMENTATION.md`), se podría considerar una licencia Creative Commons, por ejemplo, CC BY 4.0 (Atribución).

2.3. Viabilidad de Tiempo o Cronograma (*)

La gestión del tiempo ha sido un factor crítico para el desarrollo de GameTracker, dada la ambición del proyecto y el plazo disponible. El proyecto se inició el *10 de marzo de 2025* y se completó en su fase principal de desarrollo el *27 de mayo de 2025*, abarcando un periodo intensivo de aproximadamente *11 semanas*.

- Planificación Inicial y Etapas (Ajustadas a un Cronograma Intensivo):
  Debido al corto plazo, el trabajo se abordó de manera muy enfocada, solapando algunas fases y priorizando el desarrollo de funcionalidades clave:

1. Análisis, Diseño y Configuración Inicial (aprox. 1.5 semanas):
   - Periodo: Semana del 10 de marzo - Mediados de la semana del 17 de marzo de 2025.
   - Tareas: Definición rápida del alcance del MVP, selección y configuración del stack tecnológico (Astro, React, Node.js, Express, MongoDB), diseño ágil de la base de datos (modelos como [`Usuario.js`](server/src/models/Usuario.js), [`UserGameList.js`](server/src/models/UserGameList.js), [`Achievement.js`](server/src/models/Achievement.js)) y estructura básica de la API.
2. Desarrollo del Backend - Funcionalidades Core (aprox. 3.5 semanas):
   - Periodo: Finales de la semana del 17 de marzo - Mediados de la semana del 7 de abril de 2025.
   - Tareas:Implementación del sistema de autenticación ([`authController.js`](server/src/controllers/authController.js) - registro, login, JWT), desarrollo de controladores y rutas para usuarios, y la gestión básica de listas de juegos ([`userGameListController.js`](server/src/controllers/userGameListController.js)). Integración con MongoDB.
3. Desarrollo del Frontend - Interfaz Básica y Conexión API (aprox. 3 semanas):
   - Periodo: Finales de la semana del 7 de abril - Finales de la semana del 28 de abril de 2025.
   - Tareas: Configuración del proyecto Astro ([`client/astro.config.mjs`](client/astro.config.mjs)), desarrollo de componentes React esenciales para autenticación y listas de juegos, creación de páginas principales (login, home, mis listas), conexión inicial con la API del backend. Estilización básica con TailwindCSS.
4. Desarrollo de Funcionalidades Avanzadas y Sociales (Backend y Frontend en paralelo) (aprox. 2 semanas):
   - Periodo: Semana del 28 de abril - Mediados de la semana del 12 de mayo de 2025.
   - Tareas: Implementación de sistema de amigos ([`friendsController.js`](server/src/controllers/friendsController.js)), grupos ([`grupoController.js`](server/src/controllers/grupoController.js)), chat con Socket.IO, sistema de logros ([`achievementService.js`](server/src/services/achievementService.js)), y Zona Arcade (ej. [`TimelineSortGame.jsx`](client/src/components/minigames/TimelineSortGame.jsx)). Desarrollo de los componentes de UI correspondientes.

5. Integración, Pruebas Intensivas y Refinamiento (aprox. 1 semana):
    - Periodo: Finales de la semana del 12 de mayo - Mediados de la semana del 19 de mayo de 2025.
    - Tareas: Pruebas exhaustivas de todas las funcionalidades, depuración intensiva, mejoras de usabilidad y rendimiento.
6. Documentación y Preparación Final (aprox. 1 semana):
    - Periodo: Finales de la semana del 19 de mayo - 27 de mayo de 2025.
    - Tareas: Redacción de la memoria del proyecto ([`README.md`](README.md), [`INSTALL.md`](INSTALL.md), [`DOCUMENTATION.md`](DOCUMENTATION.md)), preparación para la entrega.

- Metodología Aplicada:
  Dado el cronograma extremadamente ajustado, se adoptó un enfoque de desarrollo rápido y altamente iterativo, similar a un *sprint* continuo:
    - Priorización Extrema (MVP): Se identificaron las funcionalidades absolutamente esenciales para una primera versión funcional, dejando características secundarias o menos críticas para posibles mejoras futuras.
    - Desarrollo Incremental Rápido: Se construyeron pequeñas partes funcionales y se integraron continuamente.
    - Solapamiento de Tareas: En la medida de lo posible, tareas de backend y frontend para una misma funcionalidad se abordaron de forma casi paralela o con transiciones muy rápidas.
    - Comunicación y Auto-gestión Constante: Se requirió una disciplina rigurosa para mantener el enfoque y el progreso diario.

- Evaluación de Desviaciones Respecto a la Planificación Inicial:
  Con un cronograma tan comprimido, cualquier desviación podría tener un impacto significativo.
    - Fases o Tareas que Resultaron Más Desafiantes (y potencialmente consumieron más tiempo proporcional del asignado):
        - Sistema de Logros ([`server/src/services/achievementService.js`](server/src/services/achievementService.js)): Incluso en una implementación simplificada, la lógica de verificación de criterios ([`Achievement.js`](server/src/models/Achievement.js)) puede ser compleja.
        - Integración del Chat en Tiempo Real con Socket.IO: Configurar y depurar la comunicación bidireccional para chats privados y de grupo ([`MyGroupsMenu.jsx`](client/src/components/groups/MyGroupsMenu.jsx), [`GroupChatWindow.jsx`](client/src/components/groups/GroupChatWindow.jsx)) suele requerir tiempo.
        - Autenticación de Dos Factores (2FA): Implementar 2FA ([`authController.js`](server/src/controllers/authController.js) en `verify2FA`) de forma segura es una tarea que no se puede apresurar.

- Funcionalidades No Completadas o Descartadas (o Simplificadas):
  - Es altamente probable que con solo 11 semanas, algunas de las funcionalidades más complejas o que requieren mucho pulido (como la "Zona Arcade" con múltiples minijuegos bien desarrollados, o un sistema de grupos con todas las características de gestión) tuvieran que ser **simplificadas significativamente** o que algunas de sus sub-características fueran **descartadas** para cumplir con el plazo.
  - Por ejemplo, el logro de "Racha de conexión diaria" (mencionado en el modelo [`Achievement.js`](server/src/models/Achievement.js)) probablemente no se implementó.

El desarrollo de GameTracker en un periodo de 11 semanas ha representado un desafío considerable, exigiendo una planificación meticulosa, una ejecución eficiente y una priorización constante para entregar un producto funcional con las características esenciales.

3. Análisis y Diseño del Proyecto

3.1. Descripción de la Arquitectura Web: MPA, MVC, SPA…

GameTracker se ha desarrollado siguiendo una arquitectura de Aplicación de Página Múltiple (MPA) para el frontend, aprovechando las capacidades de [Astro](https://astro.build) para generar sitios web optimizados y rápidos mediante el envío de HTML renderizado en el servidor (SSR) o generado estáticamente (SSG) para la mayoría de las vistas. Dentro de estas páginas, se utilizan **componentes interactivos construidos con React** para las secciones que requieren una alta dinamicidad y gestión de estado en el cliente, como los chats, los minijuegos de la Zona Arcade, o los modales de edición. Esta aproximación híbrida permite combinar la eficiencia de carga de las MPA con la rica experiencia de usuario de las SPA en partes específicas.

El backend sigue una arquitectura basada en el patrón **Modelo-Vista-Controlador (MVC), aunque adaptado a una API RESTful.

- Modelos ([/PROYECTO-TFG-ASTRO-NODE.JS/server/src/models/)): Definen la estructura de los datos (ej. [`Usuario.js`]Usuario.js), [`UserGameList.js`](PROYECTO-TFG-ASTRO-NODE.JS/server/src/models/UserGameList.js)) y la lógica para interactuar con la base de datos MongoDB a través de Mongoose.
- Controladores ([`server/src/controllers/`](PROYECTO-TFG-ASTRO-NODE.JS/PROYECTO-TFG-ASTRO-NODE.JS/server/src/controllers/)): Manejan la lógica de negocio para cada solicitud HTTP recibida, interactuando con los modelos y los servicios (ej. [`authController.js`]authController.js), [`grupoController.js`]grupoController.js)).
- Rutas ([`server/src/routes/`](PROYECTO-TFG-ASTRO-NODE.JS/server/src/routes/)): Definen los endpoints de la API (ej. [`authRoutes.js`]authRoutes.js), [`userGameListRoutes.js`]userGameListRoutes.js)) y los asocian a los controladores correspondientes. La "Vista" en este contexto es la respuesta JSON enviada al cliente.
- Servicios ([`server/src/services/`](PROYECTO-TFG-ASTRO-NODE.JS/server/src/services/)):** Contienen lógica de negocio reutilizable o compleja que puede ser invocada por múltiples controladores, como el [`achievementService.js`]achievementService.js).

La comunicación entre el frontend y el backend se realiza principalmente a través de peticiones HTTP (API RESTful) para la obtención y manipulación de datos. Adicionalmente, se utiliza WebSockets (Socket.IO) para funcionalidades en tiempo real como el chat privado y de grupo, y potencialmente para notificaciones instantáneas.

3.2. Tecnologías y Herramientas Utilizadas

La selección de tecnologías se ha realizado buscando un equilibrio entre modernidad, eficiencia, y la capacidad de aplicar los conocimientos adquiridos durante el ciclo formativo.

- Frontend:
  - Framework Principal: [Astro](https://astro.build) ([`client/astro.config.mjs`](PROYECTO-TFG-ASTRO-NODE.JS/client/astro.config.mjs)) para la construcción del sitio, optimización de la carga y renderizado de páginas.
  - Librería UI para Componentes Interactivos: [React](https://reactjs.org/) para el desarrollo de islas de interactividad y componentes dinámicos.
  - Estilización: [TailwindCSS](https://tailwindcss.com) ([`client/tailwind.config.js`](PROYECTO-TFG-ASTRO-NODE.JS/client/tailwind.config.js)) para un diseño de interfaz de usuario rápido, personalizable y responsivo.
  - Lenguaje: TypeScript ([`client/tsconfig.json`](PROYECTO-TFG-ASTRO-NODE.JS/client/tsconfig.json)) para tipado estático y mejora de la calidad del código.
  - Gestor de Paquetes: npm (manejado a través de [`client/package.json`](PROYECTO-TFG-ASTRO-NODE.JS/client/package.json)).
  - Peticiones HTTP: `fetch` API nativa y `axios` para la comunicación con el backend.
  - Almacenamiento en Cliente: IndexedDB (a través de [`client/src/services/storage/IndexedDbService.ts`](client/src/services/storage/IndexedDbService.ts)) para persistencia de datos de sesión y caché ligera.
- Backend:
  - Entorno de Ejecución: [Node.js](https://nodejs.org).
  - Framework Principal: [Express.js](https://expressjs.com/) para la creación de la API RESTful ([`server/server.js`]server.js)).
  - ODM (Object Data Modeling): [Mongoose](https://mongoosejs.com/) para la interacción con la base de datos MongoDB.
  - Lenguaje: JavaScript (ES6+).
  - Gestor de Paquetes: npm (manejado a través de [`server/package.json`](PROYECTO-TFG-ASTRO-NODE.JS/server/package.json)).
  - Variables de Entorno: `dotenv` para la gestión de configuraciones sensibles ([`server/.env`](PROYECTO-TFG-ASTRO-NODE.JS/server/.env)).
  - Base de Datos:
    - Sistema Gestor: [MongoDB](https://www.mongodb.com), una base de datos NoSQL orientada a documentos.
    - Alojamiento: MongoDB Atlas (para desarrollo y potencial despliegue) o instancia local.
  - Integración y Pruebas:
    - Pruebas Manuales: Realizadas exhaustivamente durante el desarrollo para asegurar la funcionalidad de cada módulo.
    - Utilización de las herramientas de desarrollo de navegadores para depuración y análisis de rendimiento

- Seguridad:
  - Autenticación: Basada en JSON Web Tokens (JWT) ([`jsonwebtoken`](https://www.npmjs.com/package/jsonwebtoken)).
  - Hashing de Contraseñas: `bcrypt` ([`bcrypt`](https://www.npmjs.com/package/bcrypt)) para el almacenamiento seguro de contraseñas.
  - Autenticación de Dos Factores (2FA): `speakeasy` y `qrcode` para la generación y verificación de códigos TOTP.
  - Middleware de Autenticación: [`server/src/middlewares/authMiddleware.js`]authMiddleware.js) para proteger rutas.
  - Middleware de Autorización de Grupos: [`server/src/middlewares/grupoAuthMiddleware.js`] grupoAuthMiddleware.js) para control de acceso específico en grupos.
  - CORS: Configurado en el backend ([`server/pipeline/pipeline.js`]pipeline.js)) para permitir peticiones desde el dominio del cliente.
  - HTTPS: Previsto para el entorno de producción para asegurar la comunicación.
- Otras Herramientas:
  - Comunicación en Tiempo Real: [Socket.IO](https://socket.io/) para chats y notificaciones.
  - Envío de Correos Electrónicos: [Nodemailer](https://nodemailer.com/) para verificación de email, recuperación de contraseña, etc.
  - Subida de Archivos: [Multer](https://www.npmjs.com/package/multer) para gestionar la subida de imágenes de perfil, banners y archivos de chat.
  - Peticiones HTTP (Backend): `axios` para comunicarse con la API de RAWG.
  - Control de Versiones: Git y GitHub.
  - Entorno de Desarrollo Integrado (IDE): Visual Studio Code.

3.3. Análisis de Usuarios (Perfiles de Usuario)

GameTracker está diseñado principalmente para un tipo de usuario, aunque con diferentes niveles de interacción y privilegios dentro de ciertas funcionalidades:

- Usuario Registrado (Jugador):
  - Descripción: Es el perfil principal. Son entusiastas de los videojuegos que desean organizar su colección, seguir su progreso, descubrir nuevos títulos, interactuar con otros jugadores y participar en la comunidad.
  - Necesidades:
    - Registrar y gestionar su biblioteca personal de juegos.
    - Marcar estados de juego (jugando, completado, pendiente, etc.).
    - Personalizar su perfil público (avatar, banner, biografía, juego favorito).
    - Descubrir nuevos juegos mediante búsquedas, filtros y recomendaciones.
    - Conectar con otros usuarios (sistema de amigos).
    - Comunicarse mediante chat privado.
    - Crear y/o unirse a grupos temáticos, participando en sus chats.

        - Desbloquear logros y ver su progreso.
        - Acceder a minijuegos en la Zona Arcade.
        - Gestionar la seguridad de su cuenta (contraseña, 2FA).
    - Privilegios:
        - Acceso a todas las funcionalidades básicas de la plataforma tras el login.
        - Crear y gestionar su propio contenido (listas, perfil).
        - Dentro de los grupos, pueden tener diferentes roles:
            - Miembro: Puede participar en el chat del grupo y ver contenido.
            - Admin (del grupo): Además de los permisos de miembro, puede gestionar otros miembros (cambiar rol a miembro, expulsar) y editar detalles del grupo. Asignado por el líder o por otro admin.
            - Líder (del grupo): Creador del grupo. Tiene todos los permisos de admin y, adicionalmente, puede eliminar el grupo y asignar roles de admin.

No existe un perfil de "Administrador Global de la Plataforma" accesible desde la interfaz de usuario en la versión actual, aunque la estructura permitiría su desarrollo futuro para tareas de moderación o gestión del sistema.

3.4. Definición de Requisitos Funcionales y No Funcionales

- Requisitos Funcionales:

La aplicación GameTracker debe ser capaz de realizar las siguientes funciones principales:

1. Gestión de Cuentas de Usuario:
    - Registro de nuevos usuarios con verificación por correo electrónico ([`server/src/controllers/authController.js`]authController.js) - `register`, `verify`).
    - Inicio de sesión seguro ([`server/src/controllers/authController.js`]authController.js) - `login`).
    - Recuperación de contraseña.
    - Configuración y uso de Autenticación de Dos Factores (2FA) ([`server/src/controllers/authController.js`]authController.js) - `setup2FA`, `verify2FA`, `disable2FA`).
    - Gestión de sesiones activas ([`server/src/controllers/authController.js`]authController.js) - `getActiveSessions`, `logoutSession`, `logoutAllSessions`).
    - Cambio de contraseña.
    - Eliminación de cuenta.
2. Gestión de Perfil de Usuario:
    - Visualización y edición de información personal: nombre, biografía, juego favorito, cumpleaños, plataformas ([`server/src/models/Usuario.js`]Usuario.js)).
    - Subida y cambio de imagen de avatar y banner de perfil ([`server/src/routes/authRoutes.js`]authRoutes.js) - ruta `/profile`).

3. Gestión de Listas de Juegos Personales:
    - Crear, editar y eliminar listas de juegos personalizadas ([`server/src/routes/userGameListRoutes.js`]userGameListRoutes.js)).
    - Añadir juegos a las listas, especificando estado (pendiente, jugando, completado, etc.), puntuación personal, y notas.
    - Visualizar y filtrar juegos dentro de las listas.
4. Descubrimiento de Juegos:
    - Búsqueda global de juegos utilizando la API de RAWG ([`server/src/controllers/rawgController.js`]rawgController.js)).
    - Visualización de detalles de juegos (descripción, géneros, plataformas, imágenes, vídeos, etc.).
    - Filtros avanzados para refinar búsquedas.
    - Asistente de descubrimiento guiado por preguntas.
    - Ruleta de juegos para selección aleatoria.
5. Funcionalidades Sociales:
    - Sistema de Amigos: Enviar, aceptar, rechazar y cancelar solicitudes de amistad; ver lista de amigos; eliminar amigos ([`server/src/routes/friendsRoutes.js`](/PROYECTO-TFG-ASTRO-NODE.JS/server/src/routes/friendsRoutes.js)).
    - Chat Privado: Comunicación en tiempo real (texto y archivos) con amigos (manejado por Socket.IO y [`server/src/controllers/chatController.js`]chatController.js) para historial).
    - Grupos: Crear grupos, unirse a grupos, salir de grupos; chat de grupo en tiempo real; gestión de miembros y roles (líder, admin, miembro) ([`server/src/routes/grupoRoutes.js`]grupoRoutes.js)).
6. Sistema de Logros Interno:
    - Desbloquear logros basados en acciones en la plataforma (ej. registrarse, añadir X juegos, hacer amigos) ([`server/src/services/achievementService.js`]achievementService.js)).
    - Visualizar logros desbloqueados y puntuación total ([`server/src/routes/achievementRoutes.js`]achievementRoutes.js)).
7. Zona Arcade:
    - Acceso y participación en diversos minijuegos (Higher/Lower, Memory Match, Guess the Game, Timeline Sort, Doom, Pokémon Crystal).
8. Notificaciones y Buzón Interno:
    - Recepción de notificaciones sobre eventos importantes (solicitudes de amistad, mensajes, etc.) en un buzón interno.
    - Marcar notificaciones como leídas.

- Requisitos No Funcionales:

1. Rendimiento:
    - Tiempos de carga de página rápidos, especialmente para las vistas principales (facilitado por Astro).
    - Respuestas de API eficientes (consultas optimizadas a la base de datos).
    - Manejo fluido de interacciones en tiempo real (chat).
2. Usabilidad:
    - Interfaz de usuario intuitiva, fácil de aprender y navegar.
    - Diseño responsivo adaptable a diferentes tamaños de pantalla (escritorio, tablet, móvil).
    - Feedback claro al usuario sobre sus acciones (mensajes de éxito, error, carga).

3. Seguridad:
    - Protección contra vulnerabilidades comunes (XSS, CSRF, Inyección SQL - mitigado por Mongoose).
    - Almacenamiento seguro de contraseñas (hashing con bcrypt).
    - Transmisión segura de datos sensibles (HTTPS en producción).
    - Protección de rutas de API mediante autenticación y autorización.
    - Validación de entradas tanto en el cliente como en el servidor.
4. Escalabilidad:
    - La arquitectura modular del backend (Node.js, Express) y el uso de MongoDB permiten escalar componentes horizontal o verticalmente.
    - El frontend con Astro puede manejar un gran número de usuarios concurrentes para contenido estático/SSR.
5. Mantenibilidad:
    - Código fuente organizado en módulos y componentes bien definidos ([`client/src/components/`](client/src/components/), [`server/src/controllers/`](PROYECTO-TFG-ASTRO-NODE.JS/server/src/controllers/), [`server/src/services/`](PROYECTO-TFG-ASTRO-NODE.JS/server/src/services/)).
    - Uso de TypeScript en el frontend para mejorar la legibilidad y reducir errores.
    - Comentarios adecuados en el código.
    - Documentación clara ([`README.md`]README.md), [`INSTALL.md`]INSTALL.md), [`DOCUMENTATION.md`]DOCUMENTATION.md)).
6. Fiabilidad:
    - La aplicación debe estar disponible y funcionar correctamente la mayor parte del tiempo.
    - Manejo adecuado de errores para evitar caídas inesperadas.
7. Compatibilidad:
    - Funcionar correctamente en las últimas versiones de los navegadores web modernos (Chrome, Firefox, Edge, Safari).
8. Accesibilidad (A11y):
    - Se buscará seguir las pautas básicas de accesibilidad web (WCAG) para que la aplicación pueda ser utilizada por el mayor número de personas posible (uso de HTML semántico, contraste adecuado, navegación por teclado donde sea aplicable).

3.5. Estructura de Navegación

El mapa del sitio de GameTracker se organiza de la siguiente manera (principales páginas accesibles tras el login, algunas rutas pueden ser dinámicas con IDs):

- `/` (Raíz): Página de inicio / Login / Registro.
- `/home`: Dashboard principal del usuario tras iniciar sesión.
- `/my-lists`:
- `/my-lists`: Vista general de las listas de juegos del usuario.
- `/my-lists/[listId]`: Vista detallada de una lista específica.
- `/discover`: Página para descubrir nuevos juegos (búsqueda, filtros, asistente, ruleta).
- `/discover/game/[gameSlugOrId]`: Detalles de un juego específico.

- `/profile/[userId]`: Perfil público de un usuario. (Si `userId` es el del usuario actual, puede redirigir o mostrar opciones de edición).
- `/me/achievements`: Página para ver los logros del usuario.
- `/arcade`: Zona Arcade con la selección de minijuegos.
- `/arcade/[minigameId]`: Pantalla de un minijuego específico.
- `/buzon`: Buzón de entrada de notificaciones y mensajes del sistema.
- `/settings`: Página de configuración de la cuenta del usuario.
- `/settings/profile`: Editar perfil.
- `/settings/security`: Seguridad de la cuenta (contraseña, 2FA).
- `/settings/sessions`: Gestionar sesiones activas.
- `/settings/danger-zone`: Eliminar cuenta.
- `/friends`: (Conceptual, la gestión de amigos se realiza a través de menús/modales, no necesariamente una página dedicada).
- `/groups`: (Conceptual, la gestión de grupos se realiza a través de menús/modales, no necesariamente una página dedicada para listar todos los grupos, aunque podría existir una página de "explorar grupos").
- `/groups/[groupId]`: (Conceptual, para ver detalles de un grupo específico si se implementa una página dedicada).

La navegación principal se realiza a través de una barra de navegación persistente (Header) y menús desplegables o modales para acciones específicas.

3.6. Organización de la Lógica de Negocio

Backend:

La lógica del backend está estructurada siguiendo un patrón similar a MVC, organizado en las siguientes carpetas dentro de [server/src/]:

- models/

Contiene los esquemas de Mongoose que definen la estructura de los datos para cada colección en MongoDB (ej. Usuario.js, Grupo.js, Achievement.js). Estos modelos también pueden contener métodos estáticos o de instancia para operaciones comunes.

- controllers/

Cada archivo de controlador (ej. authController.js, userGameListController.js, grupoController.js) agrupa las funciones que manejan las solicitudes HTTP para una entidad o recurso específico. Estos controladores interactúan con los modelos para acceder/modificar datos y con los servicios para lógica más compleja.

- routes/

Define los endpoints de la API RESTful (ej. authRoutes.js, grupoRoutes.js). Cada archivo de ruta mapea URLs y métodos HTTP a las funciones correspondientes en los controladores. También aplican middlewares como el de autenticación (authMiddleware.js) o autorización específica (grupoAuthMiddleware.js).

- services/

Contiene módulos con lógica de negocio más compleja o reutilizable que no encaja directamente en un controlador o modelo. Un ejemplo clave es achievementService.js, que maneja la verificación y otorgamiento de logros. También se encuentra aquí la lógica para el envío de correos (emailService.js).

- middlewares/

Contiene funciones middleware de Express, como authMiddleware.js para la autenticación basada en JWT, grupoAuthMiddleware.js para la autorización en grupos, y uploadMiddleware.js para la gestión de subida de archivos con Multer.

- utils/

Funciones de utilidad genéricas que pueden ser usadas en diferentes partes del backend (ej. buzonUtils.js, nicknameUtils.js).

- config/

Archivos de configuración, como la conexión a MongoDB (MongoDB.js) y la carga de variables de entorno (ENVConfig.js).

- sockets/

Lógica relacionada con Socket.IO, como chatSocket.js, que maneja los eventos de chat en tiempo real.

- server.js

El archivo principal server.js configura la aplicación Express, inicializa middlewares, registra las rutas y arranca el servidor HTTP y Socket.IO.

Conexión con APIs de Terceros o Servicios Externos:

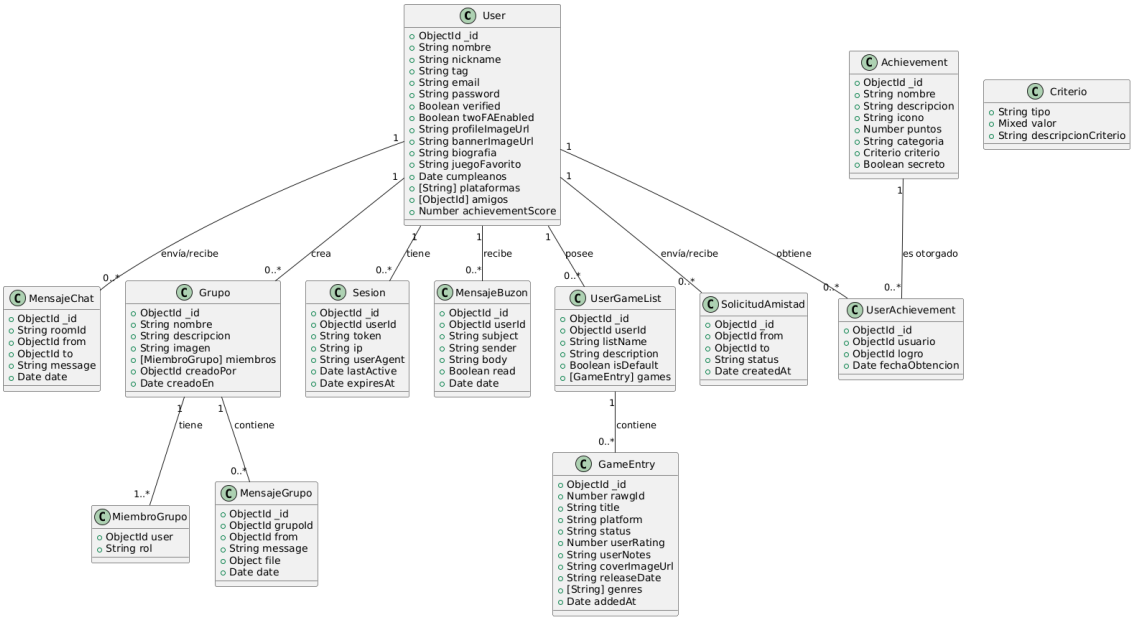
- API de RAWG (https://rawg.io/apidocs):

Uso: GameTracker utiliza extensivamente la API de RAWG como fuente principal de información sobre videojuegos. Se consulta para:
- Búsqueda de juegos.
- Obtención de detalles de juegos (descripciones, géneros, plataformas, fechas de lanzamiento, ratings, imágenes, tráilers, etc.).
- Obtención de listas de géneros, plataformas, desarrolladores, etc., para filtros y exploración.
Implementación: La interacción con la API de RAWG se centraliza en el backend, principalmente a través del controlador rawgController.js y las rutas definidas en rawgRoutes.js. Esto permite que el backend actúe como un proxy, protegiendo la clave API de RAWG (almacenada en las variables de entorno del servidor) y potencialmente cacheando respuestas para mejorar el rendimiento y reducir el número de llamadas a la API externa.

- Servicio de Correo Electrónico (Nodemailer):

Uso: Para enviar correos transaccionales a los usuarios, tales como:
- Verificación de la dirección de correo electrónico durante el registro.
- Restablecimiento de contraseña.
- Notificaciones importantes (configurable).
Implementación: Se utiliza la librería Nodemailer, configurada a través de variables de

entorno para el proveedor SMTP. La lógica de envío está encapsulada en server/src/services/emailService.js.
No se utilizan pasarelas de pago ni servicios de autenticación externos (como OAuth con Google/Facebook) en la versión actual del proyecto.



3.7. Modelo de Datos Simplificado

La base de datos de GameTracker es NoSQL, utilizando MongoDB. A continuación, se describen las principales colecciones y la estructura simplificada de sus documentos JSON, destacando los atributos clave y las relaciones entre ellas.

1. Colección: users (Modelo: [Usuario.js]Usuario.js)) Almacena la información de los usuarios registrados.
    a. Relaciones:
        i. amigos: Referencia a otros documentos en la colección users.

2. Colección: sesiones (Modelo: [Sesion.js]Sesion.js)) Registra las sesiones activas de los usuarios.
    a. Relaciones:
        i. userId: Referencia a un documento en la colección users.

3. Colección: user_game_lists (Modelo: [UserGameList.js]UserGameList.js)) Define las listas de juegos creadas por los usuarios.
    a. Relaciones:
        i. userId: Referencia a un documento en la colección users.

4. Colección: solicitudes_amistad (Modelo: [SolicitudAmistad.js]SolicitudAmistad.js)) Modela las solicitudes de amistad entre usuarios.
    a. Relaciones:
        i. from, to: Referencias a documentos en la colección users.

5. Colección: mensajes_chat (Modelo: [MensajeChat.js]MensajeChat.js)) Mensajes individuales del chat privado.
    a. Relaciones:
        i. from, to: Referencias a documentos en la colección users.

6. Colección: grupos (Modelo: [Grupo.js]Grupo.js)) Información de los grupos creados por usuarios.
    a. Relaciones:
        i. miembros.user, creadoPor: Referencias a documentos en la colección users.

7. Colección: mensajes_grupo (Modelo: [MensajeGrupo.js]MensajeGrupo.js)) Mensajes dentro de un chat de grupo.
    a. Relaciones:
        i. grupoId: Referencia a un documento en la colección grupos.
        ii. from: Referencia a un documento en la colección users.
8. Colección: logros (Modelo: [Achievement.js]Achievement.js)) Definición de todos los logros disponibles en la plataforma.

9. Colección: user_achievements (Modelo: [UserAchievement.js]UserAchievement.js)) Vincula un usuario con un logro desbloqueado.
    a. Relaciones:
        i. usuario: Referencia a un documento en la colección users.
        ii. logro: Referencia a un documento en la colección logros.

10. Colección: mensajes_buzon (Modelo: [MensajeBuzon.js]MensajeBuzon.js)) Mensajes internos del sistema para los usuarios (notificaciones, alertas).

4. Conclusiones

Resultados obtenidos y análisis de objetivos

El desarrollo de **GameTracker** ha permitido alcanzar la mayoría de los objetivos planteados al inicio del proyecto. Se ha implementado una plataforma web funcional que permite a los usuarios:

- Gestionar su biblioteca de videojuegos.
- Descubrir nuevos títulos.
- Interactuar socialmente mediante amigos y grupos.
- Disfrutar de minijuegos en la *Zona Arcade*.

El sistema de logros internos y la integración con la API de RAWG han enriquecido la experiencia del usuario, cumpliendo con la finalidad de centralizar y mejorar la gestión lúdica personal.

Retos encontrados y soluciones implementadas

Durante el desarrollo surgieron diversos retos técnicos y de diseño, entre los que destacan:

- **Gestión de la autenticación y seguridad:**
  Implementar un sistema robusto con verificación de correo electrónico, sesiones múltiples y autenticación en dos pasos (2FA), resuelto mediante JWT, bcrypt y la integración de Speakeasy para TOTP.
- **Comunicación en tiempo real:**
  El desarrollo del sistema de chat privado y grupal requirió la integración de **Socket.IO**, con una gestión eficiente de eventos, salas y persistencia de mensajes.
- **Diseño responsivo y experiencia de usuario:**
  Se buscó una interfaz atractiva y funcional en todo tipo de dispositivos, trabajada de forma iterativa con **TailwindCSS** y pruebas constantes con usuarios.
- **Integración con APIs externas:**
  Adaptación de la lógica para consumir la API de RAWG, respetando sus limitaciones y almacenando de forma eficiente los datos más relevantes.

Aprendizajes y mejoras futuras

Este proyecto ha facilitado la consolidación de conocimientos en:

- Desarrollo *full-stack*.
- Arquitectura de aplicaciones web.
- Seguridad y autenticación.
- Integración de servicios externos.
- Aplicación de metodologías ágiles.

Mejoras planteadas a futuro:

- Implementación de pruebas automatizadas en backend y frontend.
- Mejora de accesibilidad (A11y) e internacionalización.
- Incorporación de nuevas funcionalidades sociales (foros, eventos, rankings).
- Optimización del rendimiento para entornos con alta carga de usuarios.
- Desarrollo de una aplicación móvil nativa o en formato PWA.

5. Bibliografía y fuentes de información

A continuación, se detallan las principales fuentes utilizadas durante el desarrollo del proyecto, tanto para aspectos técnicos como conceptuales:

Documentación oficial

- Astro
- React
- Node.js
- Express
- MongoDB
- Mongoose
- TailwindCSS

APIs y librerías

- RAWG Video Games Database API
- Socket.IO Documentation
- Nodemailer Documentation

Comunidades y foros técnicos

- Stack Overflow
- GitHub Discussions
- Dev.to
- Reddit: r/webdev

Recursos complementarios

- Tutoriales en YouTube y Medium sobre:
  - Integración de APIs REST.
  - Autenticación JWT.
  - Desarrollo de aplicaciones *full-stack* modernas.
- Guías de buenas prácticas de seguridad web:
  - OWASP Top Ten

6. Anexos

Manual de usuario

A continuación, se presenta una guía básica para que los usuarios puedan utilizar la plataforma **GameTracker** de forma efectiva:

1. **Registro e inicio de sesión:**
   El usuario debe registrarse con su dirección de correo electrónico. Una vez verificado, podrá iniciar sesión y activar la autenticación en dos pasos (2FA) desde la configuración.
2. **Gestión de perfil:**
   Desde el menú de usuario, se puede personalizar el avatar, el banner, la biografía y las preferencias generales.
3. **Listas de juegos:**
   Permite crear listas personalizadas, añadir videojuegos, marcar su estado (jugando, completado, pendiente, etc.) y calificarlos.
4. **Descubrimiento:**
   Herramientas como la búsqueda, filtros avanzados, el asistente personalizado y la ruleta de juegos facilitan la exploración de nuevos títulos.
5. **Amigos y grupos:**
   Se pueden enviar solicitudes de amistad, mantener conversaciones privadas o grupales, y crear o unirse a grupos de interés.
6. **Logros y arcade:**
   Visualización de logros desbloqueados y acceso a minijuegos interactivos en la sección *Zona Arcade*.
7. **Notificaciones:**
   Acceso al buzón interno para revisar mensajes y notificaciones relevantes del sistema.
8. **Seguridad:**
   Desde la configuración se puede cambiar la contraseña, gestionar sesiones activas y eliminar la cuenta de forma segura.

Guía de instalación, configuración y despliegue

Esta guía está orientada a desarrolladores o técnicos que deseen instalar y ejecutar localmente el proyecto. Ademas , dentro del proyecto hay una guía mas detallada de como instalarlo.

1. **Requisitos previos:**
   Tener instalados Node.js, npm y MongoDB.
2. **Clonar el repositorio:**

   ```
   git clone https://github.com/TFGs-2DAW/PROYECTO-TFG-ASTRO-NODE.JS
   ```

3. **Instalar dependencias:**

   ```
   cd client && npm install    # Frontend
   cd ../server && npm install # Backend
   ```

4. **Configurar variables de entorno:**
   Crear un archivo `.env` con los siguientes datos:
   - URL de conexión a MongoDB.(`MONGO_URI`)
   - Clave secreta para JWT (`JWT_SECRET`).
   - Clave de acceso a la API de RAWG (`RAWG_API_KEY`).
   - PORT(siempre el 5000)
5. **Poblar la base de datos:**
   Importar el archivo `GameTracker.logros.json` a MongoDB para incluir los logros iniciales.
6. **Ejecutar el backend:**

   ```
   cd server
   node server.js
   ```

7. **Ejecutar el frontend:**

   ```
   cd client
   npm run dev
   ```

8. **Acceder a la aplicación:**
   Abrir en el navegador la dirección:
   http://localhost:4321

Para más información técnica, consultar los archivos `INSTALL.md` y `DOCUMENTATION.md` disponibles en el repositorio del proyecto.