Jaxpi: Ciencia de Datos y Analíticas de Aprendizaje aplicadas a videojuegos educativos en web.

Jaxpi: Data Science and Learning Analytics applied to web-based educational video games.

Trabajo Fin de Grado
Curso 2023-2024

Autores
MARCOS COLOMBÁS GARCÍA
MIRIAM ELIZABETH CABANA RAMÍREZ
SERGIO JOSÉ GÓMEZ CORTÉS

DIRECTORES
ANTONIO CALVO MORATA
MANUEL FREIRE MORÁN

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Jaxpi: Ciencia de datos y Analíticas de Aprendizaje aplicadas a videojuegos educativos en web.

Jaxpi: Data Science and Learning Analytics applied to web-based educational video games.

Trabajo de Fin de Grado en Ingeniería Informática

Autores

MARCOS COLOMBÁS GARCÍA

MIRIAM ELIZABETH CABANA RAMÍREZ

SERGIO JOSÉ GÓMEZ CORTÉS

DIRECTORES

ANTONIO CALVO MORATA

MANUEL FREIRE MORÁN

Convocatoria: Septiembre 2024

Grado en Ingeniería Informática

Facultad de Informática

Universidad Complutense de Madrid

13 de Septiembre de 2024

Resumen

En la actualidad, el uso de juegos digitales como herramientas educativas se está volviendo cada vez más popular, por lo tanto surge la necesidad de desarrollar herramientas para que los profesores puedan saber si sus alumnos aprenden o mejoran usándolos, y para que los desarrolladores de juegos educativos sepan si sus juegos son efectivos en esta tarea. JaXpi nace como una de las respuestas a estas necesidades, utilizando el estándar xAPI como vehículo para construir las actividades de los alumnos en estos juegos educativos en web desarrollados con JavaScript, permitiendo la supervisión del alumnado y su análisis.

JaXpi es un proyecto que presenta un ecosistema de herramientas creadas para el envío, recogida y análisis de datos provenientes de juegos educativos en web. El objetivo es facilitar la integración de analíticas en cualquier juego y proporcionar al cuerpo docente la información sobre el aprendizaje y progreso de los estudiantes al usar juegos educativos.

El ecosistema JaXpi consiste en una librería con el estándar xAPI para la representación de los datos de interacción, que envía estos a un servidor LRS donde se recogen y guardan las trazas recibidas por dicha librería, y finalmente una plataforma de análisis de datos que permite mostrar información relevante mediante gráficos.

Palabras clave

JaXpi, juegos educativos, JavaScript, xAPI, análisis de datos, datos de interacción, MongoDB, JSON, aplicación web

Abstract

Currently, the use of digital games as educational tools is becoming increasingly popular, this creates the need to develop tools for teachers that show if their students are learning and improving upon their use, and for serious game developers to know if their games are effective for this purpose. JaXpi emerges as one of the responses to these needs, using the xAPI standard as a vehicle to track students' activities in these web-based serious games developed with JavaScript, enabling both student supervision and analysis.

JaXpi is a project that presents an ecosystem of tools created for the sending and collection of data from educational web games and its subsequent analysis. The goal is to provide an application that is easy to integrate into any game and simple for educators to use, adaptable to their specific needs.

The JaXpi ecosystem consists of a library using the xAPI standard for representing interaction data, which sends this data to a LRS server where the received traces are collected and stored, and finally, a visual data analysis platform that displays relevant information through charts.

Keywords

JaXpi, serious games, JavaScript, xAPI, data analysis, interaction data, MongoDB, JSON, web application

Índice de contenidos

Índice de figuras

Índice de tablas

1. Introducción

1.1 Motivación

En el mundo digital actual en el que vivimos, los videojuegos presentan características que los hacen una herramienta idónea para la enseñanza (Shreve, 2005) (Shaffer, 2005), como un aumento de la motivación de los estudiantes a aprender o una mejora de la participación en el proceso de aprendizaje (Sánchez-Mena, 2017). Esto los hace valiosos en el ámbito educativo, especialmente de los más jóvenes, ya que estos se encuentran expuestos a toda la tecnología digital desde una edad muy temprana (Takeuchi, 2011).

A medida que los videojuegos se popularizan en el ámbito educativo, estos generan retos tanto para los desarrolladores, teniendo que implementar componentes pedagógicos, como para los educadores, debido a sus nociones peyorativas y bajas expectativas en cuanto a su relevancia y utilidad (Kenny, 2011). Por lo tanto, surge la necesidad de desarrollar herramientas que permitan analizar cómo los estudiantes interactúan con los juegos (Squire, 2005), proporcionando información crítica sobre su progreso y habilidades aprendidas a los docentes que les enseñan, y ayudando a los desarrolladores de estos videojuegos con su efectividad pedagógica durante el proceso de enseñanza.

Para abordar esta necesidad, durante este proyecto hemos desarrollado JaXpi. JaXpi es un ecosistema de herramientas diseñadas para enviar, recabar y analizar datos de usuarios en videojuegos, facilitando el desarrollo de estos juegos y proporcionando a profesores una forma más efectiva de evaluar el progreso de sus alumnos.

Elegimos desarrollar JaXpi ya que nos permitía la oportunidad de contribuir en la mejora de la educación a través de la tecnología además de aprovechar los conocimientos adquiridos en la universidad, como JavaScript, Web Workers, y xAPI.

En definitiva, JaXpi representa un esfuerzo por contribuir al avance del aprendizaje digital, proporcionando una plataforma que une la tecnología con la educación de manera versátil y eficaz, aplicándolos en un proyecto real.

1.2 Objetivos

El objetivo general de este proyecto es crear un conjunto de herramientas que facilite la captura y análisis de datos provenientes de videojuegos desarrollados en JavaScript. Este sistema debe ser accesible tanto para los desarrolladores de videojuegos y sus jugadores, como para los educadores y sus estudiantes.

Para lograr este objetivo general, se plantearon los siguientes objetivos específicos:

- **Estudio del estado del arte**: Investigar plataformas y estándares relacionados con el análisis y la representación de datos de interacción de juegos. Además de las tecnologías adecuadas para el desarrollo del proyecto.
- **Desarrollo de una librería de recogida y envío de datos de interacción para videojuegos JavaScript**: Crear una librería que permita registrar las interacciones de los jugadores de manera estandarizada utilizando xAPI.
- **Desarrollo de una plataforma de análisis de datos**: Implementar un servidor que reciba los datos enviados por la librería y los almacene en un Learning Record Store (LRS) así como una plataforma de análisis y visualización a través de gráficos y estadísticas, adaptadas a los distintos tipos de usuarios (alumnos, profesores, desarrolladores).

- **Gestión de usuarios y roles**: Incorporar funcionalidades que permitan a la plataforma gestionar diferentes tipos de usuarios, como profesores y alumnos, facilitando la organización de las clases y el seguimiento del progreso individual y grupal de los estudiantes.
- **Integración y validación**: Integrar la librería y la plataforma con videojuegos ya desarrollados para validar su funcionalidad, asegurando que se cumplan los objetivos propuestos y que la solución sea útil y eficiente en un entorno real.

1.3 Plan de trabajo

Para el desarrollo del proyecto, seguimos un enfoque de trabajo incremental. Cada vez que añadimos nuevas funcionalidades, repetimos el ciclo de investigación, implementación e integración. Este enfoque nos permitió abordar el desarrollo de manera gradual, asegurando que cada nueva funcionalidad fuera probada antes de integrarse en el sistema completo. Para organizar nuestras tareas y facilitar la comunicación, establecimos un conjunto de prácticas que nos permitieron coordinar los esfuerzos del equipo y seguir el progreso del desarrollo:

- **Planificación inicial:** El primer paso fue definir las tecnologías y lenguajes más adecuados para cada parte del proyecto. JavaScript y xAPI eran fundamentales, con MongoDB, Node, y Express elegidos para el servidor LRS, y Angular/Vue junto a D3/C3 sugeridos para la visualización de datos. Esta planificación inicial también incluyó especificar juegos de distintos géneros para integrarlos con la librería.
- **Organización del desarrollo por iteraciones:** Planificamos el desarrollo en ciclos iterativos. Cada ciclo comienza con la investigación, donde evaluamos cómo aplicar el estándar xAPI y otras tecnologías, y qué ajustes eran necesarios. Luego, seguíamos con la implementación de las funcionalidades, asegurando

que todo estaba probado antes de ser integrado en el sistema completo. Para ello, nos asignamos tareas claras. En cada ciclo se priorizaron las funcionalidades clave, y la planificación fue adaptándose conforme avanzaba el proyecto y surgían nuevos requisitos.

- **Control de versiones y organización del código**: Creamos una organización en GitHub para alojar el proyecto, dividida en tres repositorios separados (uno para la librería, otro para el servidor LRS y el tercero para la aplicación web). Cada miembro del equipo tenía responsabilidades específicas sobre uno de los repositorios. También dimos acceso a los directores para que pudieran supervisar el proyecto durante su desarrollo. Aquí está el enlace a la organización: https://github.com/UCM-FDI-JaXpi/
- **Reuniones y comunicación**: El equipo mantenía reuniones semanales por videollamada para revisar el trabajo realizado y planificar las siguientes tareas. Además, nos reunimos con los directores del proyecto aproximadamente cada dos semanas para discutir nuevas funcionalidades, recibir retroalimentación y ajustar el plan de desarrollo. A medida que el proyecto avanzaba y las aplicaciones requerían más pruebas de integración, la frecuencia de estas reuniones aumentó.
- **Registro de trabajo**: Se mantuvo un registro detallado de todas las reuniones con los directores, en el que se documentaron las tareas asignadas y las resoluciones de las dudas que iban surgiendo. Tenemos un resumen de todos estos documentos en el *Apéndice A: Registro de Trabajo*.

1. Introduction

1.1 Motivation

In the digital world we live in today, video games possess characteristics that make them an ideal tool for teaching (Shreve, 2005) (Shaffer, 2005), such as increasing students' motivation to learn or enhancing their participation in the learning process (Sánchez-Mena, 2017).

As video games become increasingly popular in education, they create challenges for both developers, who need to implement pedagogical components, and educators, due to their negative perceptions and low expectations regarding their relevance and usefulness (Kenny, 2011). Therefore, there exist a need to develop tools that can analyze how students interact with them (Squire, 2005), providing critical information on their progress and their skills learned to the teachers who guide them, while also helping game developers improve the pedagogical effectiveness of their games during the teaching process.

To address this need, we have developed JaXpi as our project. JaXpi is an ecosystem of tools designed to send, collect and analyze user data in video games, facilitating the development of these games and providing teachers with a more effective way to evaluate their students' performance.

We chose to develop JaXpi because it allowed us the opportunity to contribute to the improvement of education through technology, while also applying the knowledge we gained at our university, such as JavaScript, Web Workers and xAPI.

In short, JaXpi represents an effort to contribute to the advancement of digital learning, providing a platform that unites technology with education in a versatile and effective way, applying them in a real project.

1.2 Goals

The overall goal of this project is to create a set of tools that facilitates the capture and analysis of data from video games developed in JavaScript. This system should be accessible to both video game developers and their players, as well as to teachers and their students.

To achieve this goal, the following partial goals were followed:

- **Study of the state of the art:** Research of platforms and standards related to the analysis and representation of interaction data from games, in addition to identifying the appropriate technologies for the development of the project.
- **Development of a data collection and transmission library for JavaScript games:** Create a library that allows the standardized recording of player interactions using xAPI.
- **Development of a data analysis platform:** Implement a server that receives the data sent by the library and stores it in a Learning Record Store (LRS), along with a data analysis and visualization platform using charts and statistics, adapted to different types of users (students, teachers, developers).
- **User and role management:** Incorporate functionalities that enable the platform to manage different types of users, such as teachers and students, facilitating class organization and tracking of both individual and group performance.
- **Integration and validation:** Integrate the library and platform with already developed video games to validate its functionality, ensuring that the proposed goals are met and that the solution is useful and efficient in a real-world context.

1.3 Work plan

For the development of the project, we followed an incremental approach. Each time we added new features, we repeated the cycle of research, implementation and integration. This approach allowed us to address development gradually, ensuring that each new feature was tested before being integrated into the complete system. To organize our task and facilitate communication, we established a set of practices that allowed us to coordinate the team's efforts and track the progress of the development:

- **Initial planning:** The first step was to define the most suitable technologies and program languages for each part of the project. JavaScript and xAPI as a pillar, and MongoDB, Node and Express chosen for the LRS server, meanwhile Angular/Vue and D3/C3 were suggested for data visualization. This initial planning also included specifying games of different genres to integrate with the library.
- **Iterative development organization:** We planned the development in iterative cycles. Each cycle began with research, where we evaluated how to apply the xAPI standard and other technologies and what adjustments were necessary. Then, we proceeded with implementing the features, ensuring everything was tested before integrating it into the complete system. To do this, we assigned ourselves clear tasks. In each cycle, key features were prioritized, and the planning adapted as the project progressed and new requirements emerged.
- **Version control and code organization:** We created an organization on GitHub to host the project, divided into three separate repositories (one for the library, another for the LRS server, and a third for the web application). Each team member had specific responsibilities for one of the repositories. We also gave access to the project supervisors so they could oversee the development. Here is the link to the organization: https://github.com/UCM-FDI-JaXpi/

- **Meetings and communication:** The team held weekly video call meetings to review the work done and plan the next task. Additionally, we met with the project supervisors biweekly to discuss new features, receive feedback and adjust the development plan. As the project progressed and the applications required more integration testing, the frequency of these meetings increased.
- **Work log:** A detailed record of all the meetings with the supervisors was maintained, documenting the task assigned and the resolutions of questions that arose. We have a summary of all these documents in the *Appendix A: Registro de Trabajo*.

2. Estado de la cuestión

En la actualidad, los videojuegos se han convertido en una de las principales formas de entretenimiento a nivel global, superando desde hace años en ingresos a las industrias del cine, la televisión o la música[1], siendo especialmente relevantes entre los más jóvenes[2], debido a una mayor exposición desde una edad más temprana a estas tecnologías[3].

Dada esta popularidad, es interesante poder desarrollar videojuegos como herramientas educativas, que con la debida supervisión de un docente, pueden convertirse en una parte crítica de la enseñanza en el mundo moderno[4].

Para este fin se están empezando a desarrollar herramientas analíticas capaces de extraer las actividades realizadas por los alumnos en estos videojuegos, facilitando a los profesores su labor de supervisión en las sesiones de juego y permitiendo a los desarrolladores mejorar la efectividad de sus juegos.

Una de las tecnologías utilizadas con este fin es el estándar xAPI como formato de representación de la actividad de los alumnos en forma de trazas, que se vuelcan sobre un LRS que las recoge para su posterior análisis[5].

Actualmente, existen diferentes herramientas que nos permiten recolectar, analizar y visualizar datos de videojuegos facilitando su uso para obtener hallazgos sobre el comportamiento y rendimiento de los jugadores. En nuestro caso, queremos

---

[1] https://www.elconfidencial.com/tecnologia/2006-03-11/videojuegos-el-gran-negocio-del-siglo-xxi_253538/

[2] https://www.scielo.cl/scielo.php?pid=s0370-41062008000700012&script=sci_arttext

[3] https://www.infobae.com/2014/08/10/1586645-a-los-6-anos-los-ninos-tienen-mejor-manejo-la-tecnologia-que-los-adultos-45/

[4] https://revistas.uptc.edu.co/index.php/ingenieria_sogamoso/article/view/7184

[5] http://www.uajournals.com/ojs/index.php/campusvirtuales/article/view/298

estudiar qué herramientas hay para los videojuegos educativos desarrollados con JavaScript en particular, y vimos que unas herramientas se ajustan mejor que otras, ya que por ejemplo *Unity Analytics*[6], está diseñado principalmente para videojuegos desarrollados con el motor *Unity* y sería mejor opción, ver otras.

La más conocida para la generación de informes estadísticos es *Google Analytics*[7]. Ampliamente utilizado tanto para analizar eventos de páginas web como de aplicaciones, incluyendo así los videojuegos educativos. Enfocado solo en videojuegos tendríamos *GameAnalytics*[8] que ofrece informes detallados del comportamiento del jugador. Y pudimos ver *T-Mon* como ejemplo de recogida y posterior análisis de trazas. (Alonso Fernández et al., 2021)

Estas solo son algunas opciones de las muchas que están surgiendo, fiel reflejo de la relevancia que están tomando los videojuegos o tecnologías en la educación.

2.1 ¿Qué es xAPI?

xAPI (antiguamente Tin Can API) es un estándar para las aplicaciones software de eLearning que permite almacenar las interacciones de una persona ante distintas actividades de aprendizaje en línea. (*Tin Can Api*, 2024). No solo es aplicado en juegos serios, si no en cualquier tipo de actividad educativa.

Su objetivo es capturar estos datos en un formato consistente, para que sistemas muy diferentes puedan comunicarse entre sí de forma segura, intercambiando flujos de actividades usando el vocabulario sencillo de xAPI.

xAPI captura datos en forma de declaraciones o trazas y cada una de estas se compone de tres elementos: actor, verbo y objeto. Esta información se almacena en registros de aprendizaje llamados LRS.

[6] https://docs.unity.com/ugs/manual/analytics/manual/overview
[7] https://developers.google.com/analytics?hl=es
[8] https://gameanalytics.com/

El estándar xAPI no solo es aplicado en juegos serios, si no en cualquier tipo de actividad educativa. Se suele usar en conjunto con LMS *(Learning Management System)* aplicaciones software utilizadas para planificar, implementar y evaluar un proceso de aprendizaje específico[9], como por ejemplo Moodle.

2.2 Trazas xAPI

Las trazas o declaraciones de xAPI se suelen presentar en formato JSON y contienen campos obligatorios y opcionales. Según el estándar xAPI, los cuatro campos obligatorios son:

- **actor**: Indica quien participa en la actividad de aprendizaje. Puede ser un usuario individual o un grupo. En nuestro proyecto, este actor será el jugador de los juegos que usen la librería JaXpi. Está compuesto por un nombre y un mail que determinan un IFI (identificador funcional inverso).
- **verb**: Describe la acción realizada por el actor, identificado mediante un descriptor con forma URI, como por ejemplo `https://github.com/UCM-FDI-JaXpi/lib/accepted`
- **object**: Representa el objeto de la acción, es decir, el recurso o actividad sobre la cual se realiza la acción, el cual también se identifica a traves de un URI, por ejemplo `https://github.com/UCM-FDI-JaXpi/objects/item`
- **timestamp**: Indica el momento en el que ocurrió la experiencia de aprendizaje. Esto puede ser especialmente práctico para calcular tiempos que tarda el actor en realizar una acción concreta, comparando el timestamp de diferentes trazas. Este se implementa en formato ISO 8061, por ejemplo `2024-08-30T19:20:46.602+00:00`

[9] https://www.techtarget.com/searchcio/definition/learning-management-system

Además de estos campos obligatorios, existen varios campos opcionales que también pueden ser muy útiles. Los más relevantes y que emplearemos en nuestro proyecto son:

- **result**: Proporciona información adicional sobre el resultado de la acción, como puntuación, o detalles específicos.
- **context**: Contiene detalles contextuales adicionales, como la ubicación o las condiciones en las que se llevó a cabo la experiencia de aprendizaje. Estos detalles vienen a su vez en campos contenidos dentro de context:
  - **instructor**: Contiene datos sobre el profesor o supervisor de la actividad.
  - **contextActivities**: Proporciona información sobre la actividad educativa, como el nombre del grupo o de la institución.
  - **extensions**: Puede contener cualquier otra información relacionada con el contexto de la actividad. La clave de cada una de estas extensiones debe estar en formato URI. En nuestro proyecto se empleó para proporcionar datos sobre las sesiones de juego.

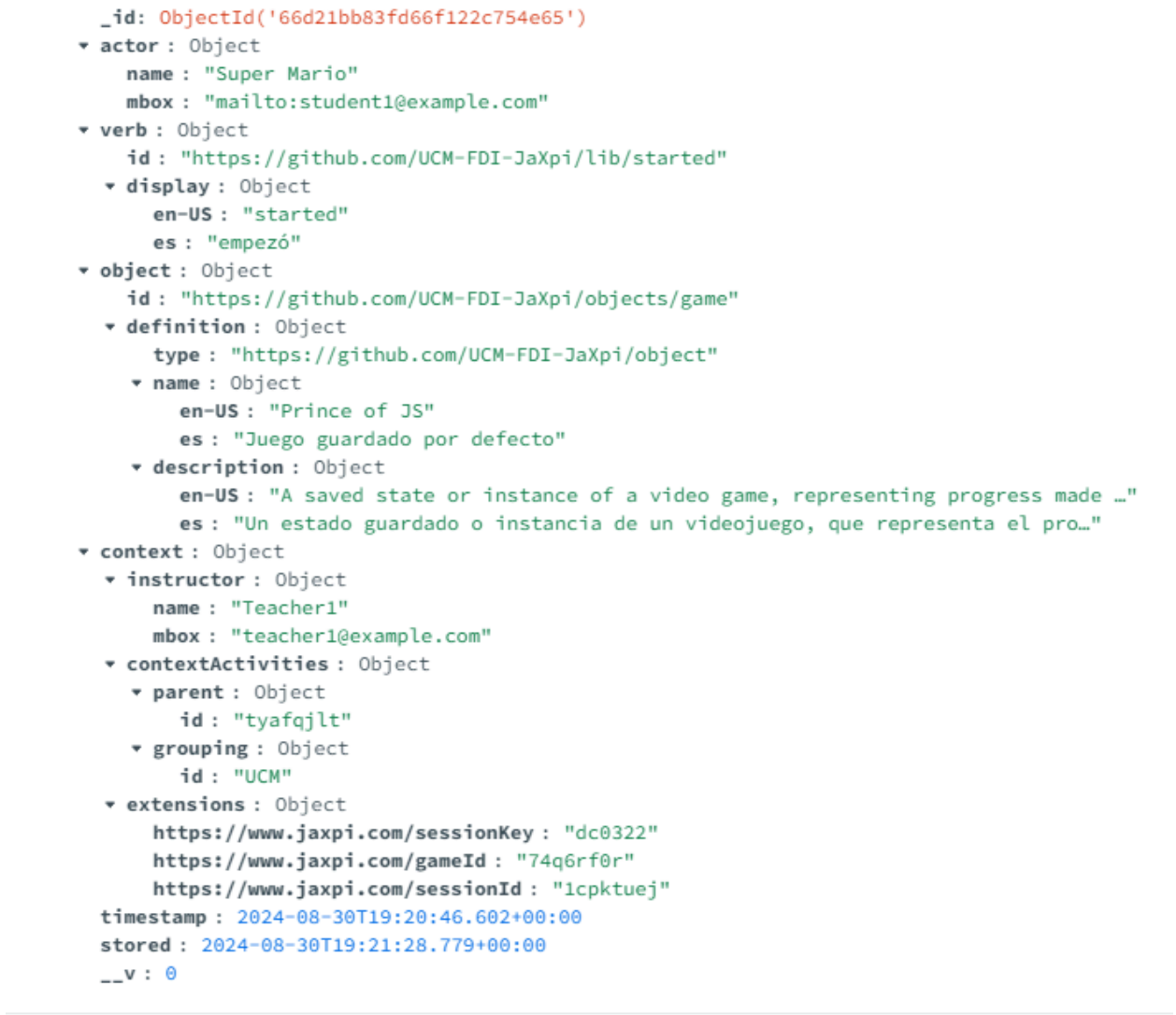

Figura 1. Traza desglosada en la base de datos.

2.3 ¿Qué es un LRS?

Un *LRS (Learning Record Store)* es un servidor que actúa como contenedor de datos que sirva para almacenar cualquier registro generado en actividades de aprendizaje, con la capacidad de compartir estos datos con otras plataformas[10].

[10] https://xapi.com/learning-record-store/?utm_source=google&utm_medium=natural_search

Puede estar integrado en un LMS[11]. En nuestro caso estos registros serán las trazas generadas por los juegos educativos que usarán nuestra librería JaXpi, y el LRS será el *endpoint* al que se envíen estas trazas, siendo este un servidor que las almacene en una base de datos y las envíe a un cliente web que realice analíticas.

Los dos LRS más usados son Watershed LRS y Veracity LRS. Watershed se destaca por sus avanzadas capacidades analíticas y visualización de datos, mientras que Veracity LRS es conocido por su cumplimiento estricto con la especificación xAPI y su facilidad para integrarse con otros sistemas de gestión del aprendizaje[12].

2.4 Tecnologías y herramientas utilizadas

- **JavaScript (ES6+)**

  El proyecto está desarrollado utilizando JavaScript moderno tanto para la librería como el servidor y la aplicación web de análisis. Este lenguaje de programación, junto con el estándar xApi, son las herramientas fundamentales del proyecto, pues la intención es utilizar ambas para integrarlas con los videojuegos web.

- **TypeScript**

  La librería además utiliza TypeScript, un superconjunto de JavaScript, que añade un tipado estático al lenguaje, ayudando a crear un código más robusto y a detectar errores de manera más rápida durante el desarrollo del proyecto.

- **NodeJS**

  Node.js proporciona un entorno de ejecución para el código javascript fuera de los navegadores, haciendo más accesible tanto la librería como el servidor a cualquier desarrollador.

[11] https://xapi.com/ecosystem/?utm_source=google&utm_medium=natural_search
[12] https://visme.co/blog/learning-record-store/

❖ **Express**

Express es el *framework backend* más popular para NodeJS, y ofrece un sistema de enrutamiento y características simplificadas que permiten desarrollar aplicaciones backend escalables[13]. Decidimos usarlo por su popularidad y sencillez de aprendizaje gracias al gran volumen de documentación y tutoriales con los que se puede aprender su uso.

❖ **MongoDB**

Decidimos usar MongoDB, una base de datos NoSQL de documentos que ofrece una gran escalabilidad y flexibilidad[14], y es una de las opciones más empleadas para almacenar datos en aplicaciones NodeJS + Express, por lo que nos facilita su integración y uso.

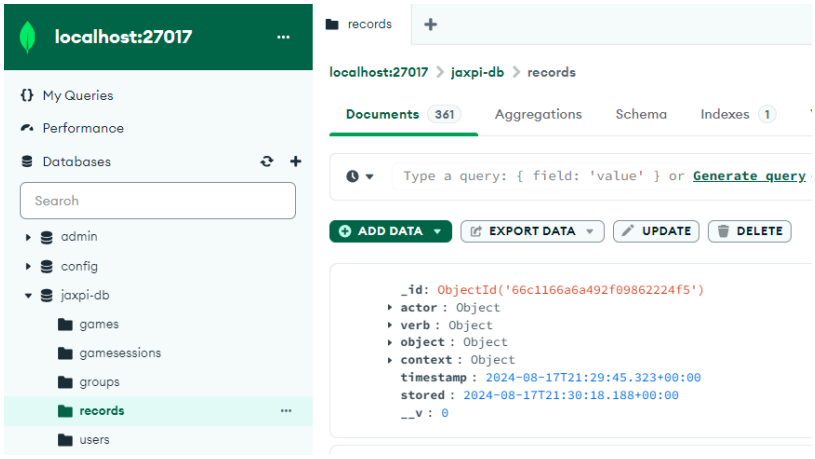

Figura 2. Vista de nuestra base de datos usando MongoDB Compass.

[13] https://kinsta.com/es/base-de-conocimiento/que-es-express/
[14] https://www.mongodb.com/es/company/what-is-mongodb

- **Vue.js**

  Vue.js, conocido comúnmente como Vue, es un framework progresivo de JavaScript utilizado para el desarrollo de interfaces de usuario interactivas y aplicaciones de una sola página (SPA) o Single Page Application.

  Es conocido por su simplicidad, flexibilidad y eficiencia, con una gran comunidad activa y abundantes recursos. Gracias a estas y otras muchas características, vimos idóneo elegir a Vue.js como framework para construir nuestra aplicación web. En la sección 6.1 entramos más en detalle.

- **Sockets.io**

  Biblioteca JavaScript que nos permite realizar una comunicación en tiempo real entre servidores y aplicaciones web de una forma más sencilla. Ideal para chats en tiempo real, videojuegos y todo lo que se necesite en tiempo real. Nos ofrece una API más sencilla en comparación con el uso directo de WebSockets y por ello elegimos probarla en nuestra aplicación ya que era justo lo que necesitábamos para recibir las trazas a tiempo real de los jugadores. En las secciones de 5 y 6, vemos el uso que ha tenido.

- **C3.js**

  Biblioteca gráfica muy utilizada a la hora de elegir cómo dibujar gráficos de datos (Figura 3). Su facilidad de uso y la conservación de toda la robustez de D3 sin entrar tanto en la configuración técnica detallada, nos deja centrarnos en la funcionalidad y el diseño. En la sección 6.3.3 damos más detalles y en la sección 7.1 vemos imágenes de los gráficos utilizados para la demo *Prince of JS*.

Figura 3. Algunos tipos de gráficos de C3.js

- **Web Workers**

  Los Web Workers se utilizan para el envío de trazas en segundo plano, evitando congelar la interfaz de los juegos educativos en los que se integra la librería.

- **Webpack**

  Se utiliza en las demostraciones con juegos que requieren de la importación de la biblioteca para exportar todos los ficheros del proyecto en un formato que el navegador entiende. No se usa directamente en el código del proyecto, pero es una parte fundamental de su correcto funcionamiento.

- **Visual Studio Code**

  Este editor de código ha sido la herramienta principal con la que hemos desarrollado todo el proyecto.

- **GitHub**

  Se utiliza GitHub para el control de versiones, lo que facilita el trabajo conjunto del equipo y un seguimiento rápido y efectivo de los cambios. Librería,

servidor y análisis tienen sus respectivos repositorios y gracias a ello, hemos podido aportar, corregir y tener un seguimiento de todo el proyecto.

- **Google Drive**

Drive se utilizó como nuestro servicio de almacenamiento gracias a su sencillez y a que todos los miembros del equipo sabían utilizarlo. Con él organizamos nuestros avances en registros que han sido almacenados a lo largo del desarrollo del proyecto.

- **Miro**

Esta aplicación la descubrimos al buscar maneras de poder generar maquetas de la aplicación y de compartir una pizarra en la que poder debatir de forma remota y simultánea entre todos. Nos ha resultado muy útil ya que al hacer reuniones entre los integrantes del equipo, ver la lluvia de ideas y poder pintar y hacer anotaciones al respecto, han surgido nuevas propuestas y correcciones de una manera más cómoda y rápida para todos.

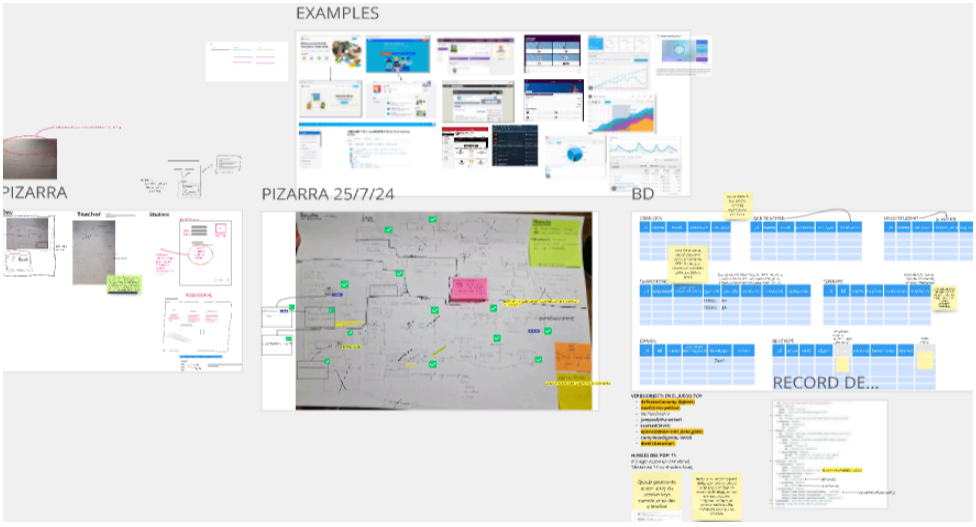

Figura 4. Tablero de Miro.

3. Arquitectura de JaXpi

Antes de entrar en detalle sobre las tres partes principales del sistema (librería, servidor y web de análisis), vamos a hacer un pequeño resumen de cómo interactúan estas aplicaciones en su conjunto, usando diagramas que representan los flujos más importantes. Estos flujos muestran cómo se manejan los datos y la autenticación dentro del sistema. Veamos primero los tipos de usuario que pueden utilizar JaXpi.

3.1 Tipos de usuario

JaXpi está diseñado para soportar diferentes tipos de usuarios, cada uno con funciones y permisos específicos, ajustados a las necesidades de los distintos roles en el entorno educativo y de desarrollo de juegos.

3.1.1 Estudiante

Los estudiantes tienen acceso exclusivamente a sus propias estadísticas de juego, lo que les permite monitorear su progreso y rendimiento en las distintas sesiones de juego en las que participan. Son registrados por su profesor, y acceden a la aplicación mediante cualquiera de las claves de sesión dadas por su profesor.

3.1.2 Profesor

Los profesores pueden acceder a las estadísticas de juego de todos sus alumnos, lo que les facilita el seguimiento del rendimiento académico y la identificación de áreas de mejora en cada uno de ellos. Además, los profesores tienen la capacidad de generar credenciales a sus estudiantes, gestionar grupos de alumnos y crear sesiones de juego. Se registran y acceden a la aplicación con usuario y contraseña.

3.1.3 Desarrollador

Los desarrolladores tienen acceso a estadísticas detalladas de todos los juegos que han creado, lo que les proporciona información valiosa para mejorar el diseño y la funcionalidad de sus juegos. Asimismo, los desarrolladores pueden generar tokens para sus juegos, lo que es esencial para autenticar las solicitudes y asegurar que solo usuarios autorizados puedan interactuar con los juegos y sus estadísticas. Este control les permite mantener la integridad y seguridad de los datos generados por sus juegos. Al igual que los profesores, se registran y acceden a la aplicación con usuario y contraseña.

3.2 Funcionalidades principales

3.2.1 Autorización del envío de datos a través de GameTokens

El envío de los datos necesita de algún tipo de identificación que impida que cualquiera pueda enviar datos a la plataforma sin autorización. Para ello, se emplean los *GameTokens.* El desarrollador puede acceder a la página web del proyecto para generar un token de juego, que actúa como un identificador único para cada juego registrado. Este token es esencial, ya que asegura que las trazas enviadas desde el juego sean identificadas y vinculadas correctamente a ese proyecto en la plataforma.

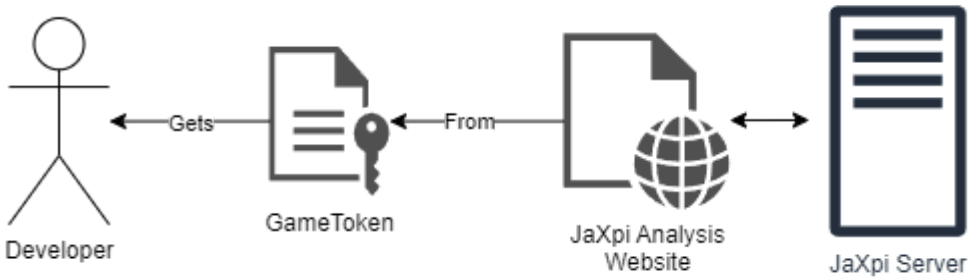

Figura 5. Desarrollador obtiene el token de juego.

A continuación, el desarrollador integra la librería en el código fuente del juego, incluyendo el token de juego generado. De esta manera, cada vez que el juego envíe trazas, el servidor podrá reconocer que provienen de ese juego en particular, autorizando su almacenamiento y análisis en la plataforma.

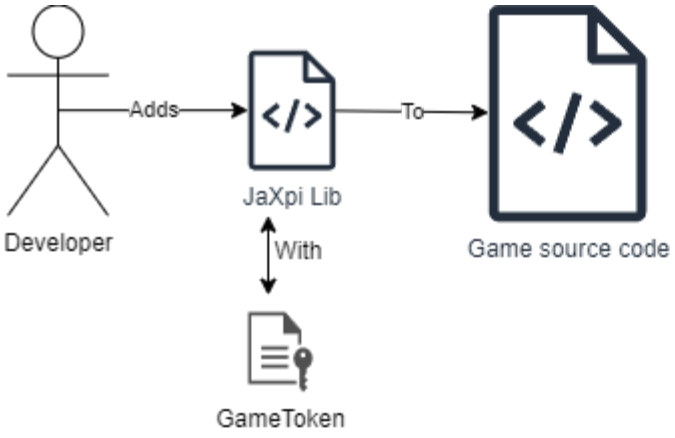

Figura 6. Desarrollador incorpora la librería al código fuente del juego.

3.2.2 Generación de credenciales y envío de trazas

En este segundo proceso, el profesor, a través de la página web, genera las credenciales para sus alumnos así como sus claves de sesión o *sessionkey*. Estas claves son únicas para cada sesión y permiten vincular los datos de las trazas a un alumno y una sesión de juego específica. Esto facilita el seguimiento individualizado del progreso de los estudiantes.

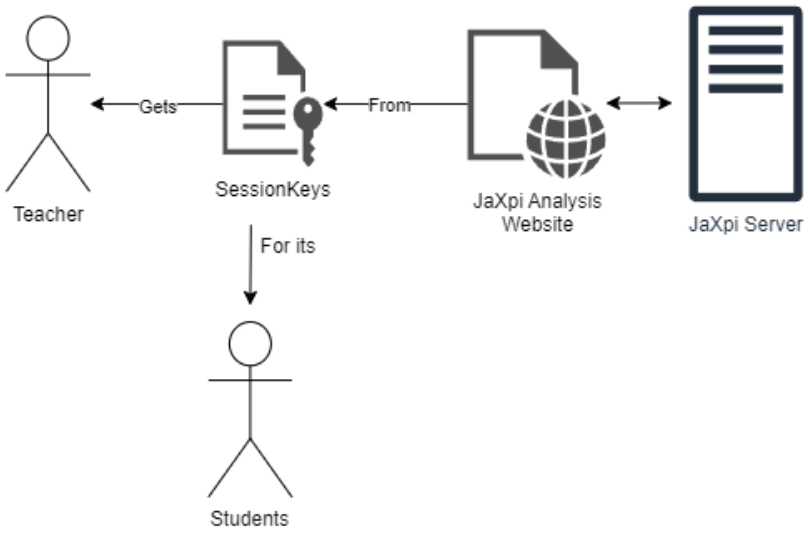

Figura 7. Profesor obtiene claves de sesión para sus alumnos.

Posteriormente, los alumnos utilizan esas claves de sesión para ingresar al juego. Cuando el alumno comienza a jugar, el juego envía las trazas xAPI al LRS del servidor, incluyendo tanto el *GameToken* del juego como la clave de sesión del alumno. Es importante destacar que los desarrolladores deben asegurarse de que el juego pase correctamente esta clave de sesión a la librería para que las trazas se asocien correctamente al alumno correspondiente.

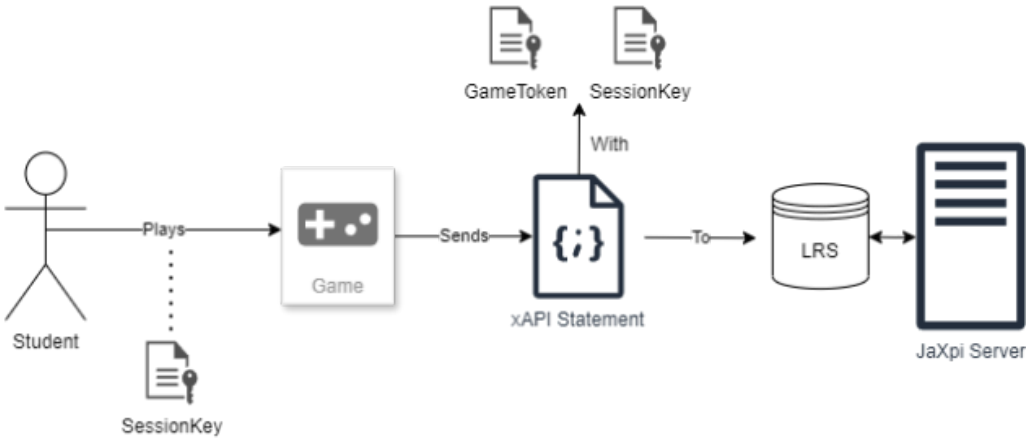

Figura 8. Envío de trazas al jugar

3.2.3 Visualización de datos

Finalmente, una vez que los datos han sido enviados y almacenados en la plataforma, los tres tipos de usuarios pueden acceder a las analíticas a través de la aplicación web, cada grupo con un nivel de acceso distinto. Los alumnos solo pueden ver sus propios datos de juego, mientras que los profesores pueden visualizar el rendimiento de todos sus alumnos. Por último, los desarrolladores pueden ver los datos de todos los jugadores de sus juegos, aunque sin acceder a información personal como nombres o correos electrónicos, preservando la privacidad de los usuarios.

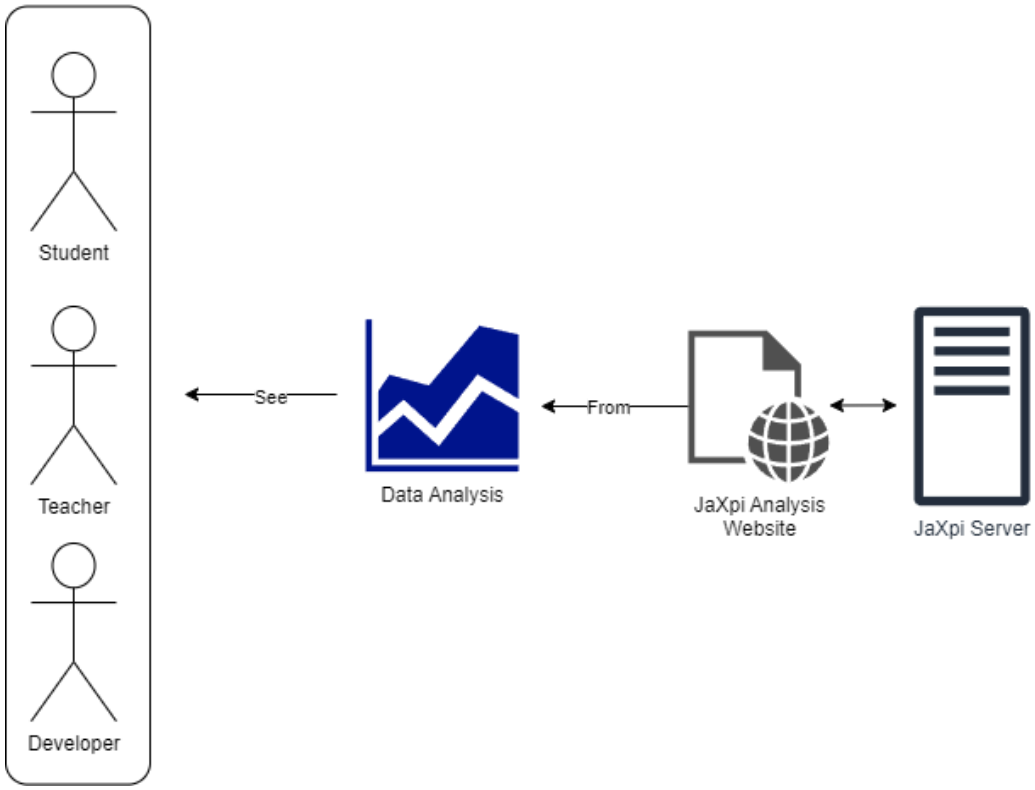

Figura 9. Visualización de datos por parte de los usuarios

4. Librería

La librería se utiliza en los juegos educativos en web desarrollados en JavaScript para generar y enviar las trazas con las acciones que los usuarios toman durante el transcurso del juego (ver Figura 1). Los desarrolladores deben encargarse de incluirla en sus códigos durante el desarrollo del juego, y esta les brinda información de la efectividad de sus juegos para la enseñanza. Además los docentes pueden supervisar la actividad de sus alumnos configurando el servidor al que las trazas van a ser enviadas.

Las trazas se componen de varios elementos que reflejan la actividad realizada, para ello se utiliza el estándar xAPI (ver sección 2.1). Los más importantes, actor, verbo y objeto reflejan quien realiza la acción, qué acción realiza y sobre qué objeto la está realizando. Para generar estas trazas la librería utiliza varias funciones anidadas verbo-objeto generadas automáticamente a partir de dos carpetas que contienen los verbos y objetos en formato JSON (ver sección 4.1 y 4.3.2). Adicionalmente las trazas se almacenan en la memoria del navegador local, permitiendo su reenvío en caso de error (ver sección 4.3.4).

La librería tiene diferentes modos de envío configurables por el desarrollador, con el uso de un temporizador, de un límite de trazas encoladas (ver sección 4.1.2), o con el uso de una función, flexibilizando las opciones del desarrollador.

4.1 Creación y validación de trazas

4.1.1 Generación de trazas

Originalmente las trazas las planteamos como una sola unidad, de forma que se pudieran almacenar en un array de JSON preparadas para ser enviadas a través de una función de búsqueda. Sin embargo, este enfoque no nos permite utilizar varios

tipos de objetos para cada verbo por lo que tuvo que ser descartado. Tras ello se optó por un nuevo enfoque basado en crear dos listas separadas, una para los verbos y otra para los objetos, que serían utilizados dentro de nuestra librería. Aunque esta solución mejoraba nuestra flexibilidad para construir trazas, ahora se presentaba el problema inverso, no podemos limitar el número de pares verbo-objeto que el usuario podría generar, dado que no todos los objetos son compatibles con todos los verbos.

Finalmente, nos decantamos por usar un sistema de funciones anidadas 'verbo.objeto' cuya estructura nos permitía la libertad de usar varios pares de verbos y objetos pero con las limitaciones necesarias. Además este modelo presentaba la ventaja de poder utilizar el predictor del editor de código, lo que facilita su uso para los desarrolladores de videojuegos que integran nuestra librería.

4.1.1.1 Gestión de Campos Extra

Solucionar el problema de los pares verbo-objeto no es suficiente para completar una traza en su totalidad, ya que estas podían contener información en campos extra. Para abordar este problema se implementa la posibilidad de agregar estos campos como parámetros en la función anidada. Como no podemos confiar en que el desarrollador tenga todos los conocimientos sobre el estándar xAPI, creamos varias funciones de validación para las trazas generadas, de esta forma si a una traza un desarrollador le agrega algo erróneo la traza es ignorada en el envío y se le devuelve un error al desarrollador que no interrumpe la ejecución del resto del código.

4.1.1.2 Extensiones en los Objetos

Haciendo pruebas de generación de trazas en un juego nos dimos cuenta que para algunas trazas los desarrolladores podrían querer enviar informacion adicional no especifica al estándar xAPI, para ello creamos el campo extensiones en el objeto, en el que se podrían guardar cualquier cantidad y tipo de parámetros extra con una clave en formato URI y el valor deseado.

4.1.1.3 Código autogenerado y Documentación

Con la inclusión del código autogenerado se incluyó en los JSON de los verbos dos nuevos campos: objetos y documentación de las extensiones.
El campo objetos incluye una lista de los nombres de objetos que son compatibles con el verbo.

El campo de documentación incluye una frase descriptiva en caso de que el verba tenga ligado algún parámetro extra, esta frase es utilizada para crear documentación en el código generativo con la herramienta JSDoc.

4.1.2 Encolado y protocolo de envío

Una vez generadas las trazas estas se van encolando en una cola asíncrona para ser procesadas de manera independiente al resto de la ejecución. Esta estrategia asegura que la generación de trazas no interfiera con el rendimiento del juego. Cuando la cola llega a una longitud crítica o tras un intervalo de tiempo, configurables por el desarrollador, se utilizan Web Workers para el envío de estas trazas con el protocolo Axios al servidor LRS escogido.

Los Web Workers dedicados proveen un medio sencillo para que el contenido web ejecute scripts en hilos en segundo plano, de esta forma impedimos que el rendimiento del juego sufra, evitando colapsar el código en ejecución con el envío de las trazas al servidor.

Se decidió usar Axios sobre Fetch porque teníamos algún conocimiento sobre esta librería y sabíamos que es más potente que el método Fetch, además de ser más legible a la hora de mantener el código.

4.1.2.1 Utilizacion de Web Workers

JavaScript es un entorno de un solo subproceso, es decir, no se pueden ejecutar varias secuencias de comandos al mismo tiempo. La mejor solución a este problema

es la utilización de Web Workers[15]. Los Web Workers son una API para generar secuencias de comandos en segundo plano en tu aplicación web, esto nos permite ejecutar el código de envío de trazas sin necesidad de bloquear la ejecución del juego, por lo tanto no afectando negativamente a su rendimiento.

En el uso tradicional de los Web Workers, su código debe estar creado en un script separado del principal, que luego se importa al principal para su utilización. Esta limitación hace imposible su uso en proyectos de ámbito global o en la creación de un sólo archivo descargable con la funcionalidad de JaXpi. Para superar esta limitación se puede usar la clase Blob[16], que permite encapsular el código del hilo en un objeto binario de gran tamaño. Este objeto Blob se puede convertir en una URL temporal que el navegador puede tratar como si fuera un archivo externo, cumpliendo así con las restricciones de los Web Workers.

4.1.3 Enriquecimiento de trazas

Si se usa toda nuestra implementación cuando las trazas llegan al servidor estas se enriquecen con los datos de la institución a la que pertenecen, incluyendo el instructor y el alumno si el jugador se ha identificado al jugar. Finalmente, también indicamos a qué juego pertenecen estas trazas. En caso de mandar las trazas a otro servidor el desarrollador se tendrá que hacer cargo de aportar los datos que sean requeridos a través del campo context, o de cualquier otro dato extra que se quiera analizar.

[15] https://developer.mozilla.org/es/docs/Web/API/Web_Workers_API
[16] https://developer.mozilla.org/es/docs/Web/API/Blob

4.2 Generación automática de código

4.2.1 Motivación para hacer código generativo

La incorporación de código generativo para crear las funciones de los pares verbo-objeto ha sido clave para mejorar la escalabilidad de la librería. Esto facilitó en gran medida la adaptación del código a cualquier cambio en los requisitos de la generación de trazas. Además esta estrategia permite que la librería sea fácilmente extensible por los desarrolladores.

4.2.2 Script de generación de código

El código generativo está alojado en su propio archivo, el cual al ejecutarse genera dos nuevos, index.ts y jaxpi.ts.

El archivo index.ts es el archivo principal de la librería, el cual contiene todas las funciones verbo-objeto que generan las trazas, así como el control de errores y la gestión de envío (ver secciones 4.1 y 4.3). Para generar estas funciones se recoge cada archivo JSON de la carpeta verbs y se crean funciones anidadas objeto gracias al campo 'objects', una lista de nombres de los objetos que acepta dicho verbo. Con esta lista y los JSON en la carpeta objects se generan todas las funciones anidadas.

Gracias a esta funcionalidad el código es fácilmente extendible simplemente añadiendo nuevos JSON con los verbos y objetos que el desarrollador quiera usar en su videojuego.

El archivo jaxpi.ts es el archivo descargable que contiene toda la librería, en caso de que el desarrollador prefiera mantener el ámbito global de su videojuego en web. Su código es completamente compatible con el navegador sin necesidad de utilizar una herramienta de empaquetación de módulos como Webpack[17]. Al compilar

[17] https://webpack.js.org/

este archivo se genera un archivo jaxpi.js en caso de que el desarrollador prefiera evitar TypeScript.

4.3 Diferentes funciones de la librería

4.3.1 Cómo usar la librería

Tras importar la librería JaXpi con '*npm install jaxpi*' o descargar el archivo javascript, necesitas crear un objeto de la clase Jaxpi y colocar en su constructor el actor que vas a usar en las trazas, la URL del LRS al que se enviarán y el token de autenticación para nuestro servidor o en caso de usar otro LRS las cabeceras que este requiera para su conexión (ver figura 10). Además puedes configurar el intervalo de tiempo en segundos en el que se enviaran las trazas y el tamaño máximo de la cola antes de enviarlas. Las trazas también se pueden enviar manualmente a través de la función '`jaxpi.flush()`'.

Hay que tener en cuenta que la librería de JaXpi funciona con Node.js, por lo tanto si ha sido importada deberás utilizar una herramienta como Webpack 5.x.x o similar para compilar tu proyecto y que funcione en entorno de navegador.

En caso de descargar el archivo.js es innecesario, está ya preparado para su uso en navegadores con proyectos en ámbito global.

```
js13k-2019-back > src > app > JS main.js > ...
1  import Game from './Game';
2  import Jaxpi from 'jaxpi'
3  import { SERVER_URL, ACTOR_NAME, ACTOR_MAIL, GAME_TOKEN_POP } from "./Config";
4
5  export const jaxpi = new Jaxpi({name: ACTOR_NAME, mail: ACTOR_MAIL}, SERVER_URL, GAME_TOKEN_POP);
6
7  Game.init();
8  Game.start();
```

Figura 10. Ejemplo de importación de la librería.

4.3.2 Funciones de verbos y objetos

Para generar una traza solo debes encontrar en tu código donde quieres que la traza se genere y utilizar una función verbo-objeto anidada, por ejemplo '`jaxpi.started().level("Level 1")`'. Es obligatorio darle nombre al objeto, pues es necesario para crear analíticas más adelante, además se le pueden añadir más parámetros a estas funciones, como una descripción que sobreescribirá la que viene por defecto en el objeto, una lista de parámetros extra con cualquier información que se quiera enviar al LRS en formato 'URI : valor', y los campos resultado, contexto y autoridad propios del estándar xAPI. Estos tres últimos esperan cada uno un JSON con un formato válido para un LRS.

4.3.3 Función verbo no estándar y añadir nuevos verbos u objetos

En caso de no existir una combinación de verbo-objeto satisfactoria para el juego en cuestión, el desarrollador tiene dos opciones, utilizar la función '*customVerb*', que le permite emparejar cualquier JSON de verbo y objeto agregados por el propio desarrollador. Se puede crear un verbo desde cero o utilizar uno de los existentes, por ejemplo '`jaxpi.verbs.accepted`', ya que estos se encuentran en una variable pública (verbs), lo mismo ocurre con los objetos, por ejemplo, '`jaxpi.objects.item`'. Esta función además acepta todos los parámetros de cualquier función anidada, salvo el nombre y la descripción, que se sobreentiende serán modificados en los JSON de entrada de ser requerido.

Si se quiere también se pueden añadir nuevos verbos u objetos en sus respectivas carpetas, usando como base cualquier otro ya existente, y utilizando el comando '*npm run generate*' para compilar y crear los archivos necesarios con las funciones anidadas actualizadas.

4.3.4 Memoria local, manejo de señales y token de sesión

La librería mantiene una gestión sobre las trazas con localStorage, lo que le permite guardar las trazas en el navegador hasta que se complete su envío en caso de error (ver sección 4.3.1), para evitar colapsar la memoria del navegador estas trazas se borran de localStorage en caso de tener demasiados intentos sin éxito de envío o si son muy antiguas. Cada vez que se intentan enviar nuevas trazas también se encolan todas las que queden en localStorage para su envío.

Además la librería cuenta con una gestión de señales de forma que si el usuario decide cerrar el navegador o algo similar ocurre, las trazas encoladas son enviadas antes de cerrar.

Por último, en caso de querer utilizar nuestro servidor con las identificaciones por clave de sesión, existen dos funciones claves dentro de la librería, 'validateKey' que te permite saber si una clave dada es aceptada por el servidor, y 'setKey', que teniendo una clave valida, la utiliza para identificar al alumno en todas las trazas que está generando sin necesidad de que el desarrollador se preocupe de hacerlo.

4.3.5 Inclusión de documentación en las funciones

Se incluye documentación autogenerada para todas las funciones verbo y cada una de sus funciones objeto anidadas de forma automática gracias al código autogenerado utilizando la sintaxis JSDoc, que permite agregar documentación de la API al código fuente de los archivos JavaScript, además en caso de que estas funciones presentan parámetros extra designados por el desarrollador al crear nuevos verbos, se puede agregar una descripción automática de estos parámetros usando el campo extensions-doc en el JSON de dicho verbo.

Para el resto de funciones públicas de la librería que no se autogeneran también se han agregado manualmente esta documentación, para la facilitación de su uso cuando los desarrolladores la quieran integrar a sus videojuegos web.

verbs > {} started.json > ...

```json
{
    "id": "https://github.com/UCM-FDI-JaXpi/lib/started",
    "display": {
        "en-US": "started",
        "es": "empezó"
    },
    "objects": ["level","game"],
    "description": "The player starts a level or a new game"
}
```

objects > {} game.json > ...

```json
{
    "id": "https://github.com/UCM-FDI-JaXpi/objects/game",
    "definition": {
        "type": "https://github.com/UCM-FDI-JaXpi/object",
        "name": {
            "en-US": "Default saved game",
            "es": "Juego guardado por defecto"
        },
        "description": {
            "en-US": "A saved state or instance of a video game, representing progress made by the player",
            "es": "Un estado guardado o instancia de un videojuego, que representa el progreso realizado por el jugador"
        }
    }
}
```

Figuras 11 y 12. Ejemplos de verbo y objeto en formato JSON.

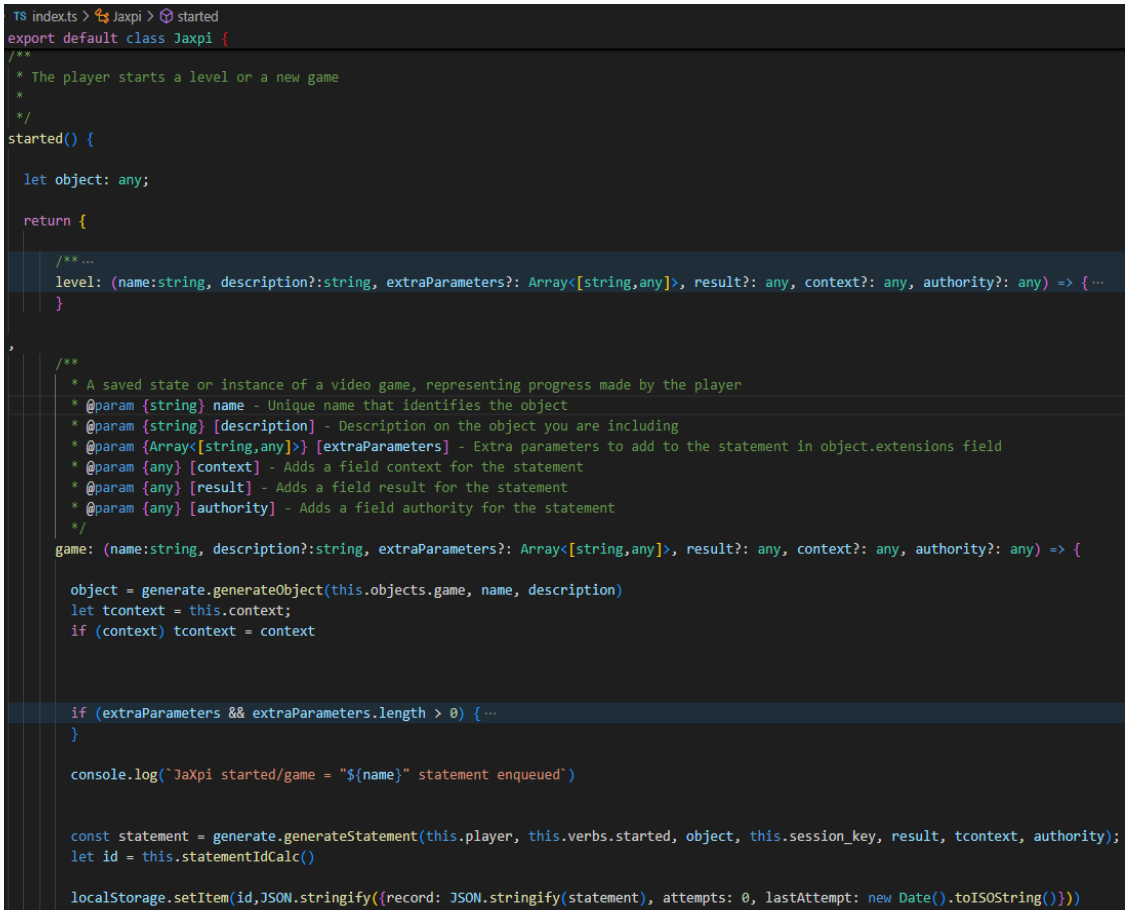

Figura 13. Ejemplo de código generado a partir del JSON.

5. Servidor

En esta sección se describe la construcción del servidor y su LRS, el cual es de gran importancia ya que actúa como punto intermedio entre la librería, encargada de enviar datos en forma de trazas xAPI, y la web de análisis que recibirá esos datos para poder realizar analíticas y visualizar datos.

5.1 Creando un servidor desde cero

La creación del servidor implicó una selección de tecnologías para desarrollar un sistema que fuera sencillo y eficiente. Para este propósito, se decidió usar Node.js, una plataforma de desarrollo backend que permite ejecutar JavaScript. Esta herramienta fue elegida principalmente por ser la herramienta de desarrollo JavaScript más usada, por lo que se disponía de multitud de documentación y tutoriales con los que aprender del uso de la herramienta.

Además, el otro gran atractivo de Node.js era contar con *npm* (Node Package Manager), un gestor de dependencias que nos facilita la instalación de librerías y herramientas adicionales necesarias para el funcionamiento del servidor, tales como Express, Mongoose y Passport, entre otros.

Mongoose fue la librería elegida para la interacción con la base de datos MongoDB. Esta proporciona una capa de abstracción sobre MongoDB, permitiendo trabajar con modelos que estructuran y validan los datos de manera más sencilla.

El aprendizaje necesario para implementar estas tecnologías incluyó familiarizarse con Node.js y el manejo de rutas y middleware en *Express*, y el manejo de autenticación y sesiones con *Passport*.

5.1.1 Configuración de un backend usando Node.js

El Backend fue montado utilizando *Express*, un marco de aplicación web para Node.js que facilita la creación de servidores de forma rápida y sencilla. Express permite la creación de rutas y el manejo de solicitudes HTTP de una manera muy intuitiva y organizada. Además, es capaz de utilizar middleware, lo que permite la adición de funcionalidades adicionales como la autenticación y la protección de rutas, entre otras.

Gran parte de la configuración del servidor se realizó en el archivo *index.js*, e incluye lo siguiente:

- **Configuración de Express:**
  - Inicialización de la aplicación Express.
  - Configuración de middleware para manejo de sesiones, autenticación y métodos HTTP.
- **Configuración de Sockets:**
  - Inicialización de socket.io para la comunicación en tiempo real.
  - Manejo de eventos de conexión y desconexión de sockets.
- **Configuración de Passport:**
  - Inicialización de Passport para la autenticación de usuarios.
  - Definición de funciones para obtener usuarios por email e ID.
- **Configuración de la base de datos:**
  - Conexión con la base de datos
  - Gestión de errores en la conexión
- **Configuración de Rutas**
- **Funciones de Autenticación y Autorización**

Las rutas del servidor fueron organizadas usando *routers*. Cada router se encarga de gestionar un conjunto específico de rutas, lo que facilita la organización y mantenimiento del código. Las principales rutas implementadas son:

- **/records**: Encargada de recibir las trazas enviadas por la librería y almacenarlas en la base de datos. También es la encargada de enviar esos datos a la web de análisis.
- **/teacher**: Encargada de facilitar a los profesores la creación de sus grupos de alumnos, así como la creación de sesiones de juego para esos grupos.
- **/dev**: Encargada de permitir a los desarrolladores registrar sus juegos en la base de datos, facilitándoles el *game-token* necesario para autenticar posteriormente el envío de trazas desde esos juegos.
- **/register, /login, /logout**: Encargadas de manejar la autenticación de usuarios mediante Passport.

5.1.2 Middleware de autenticación.

Para la autenticación y gestión de sesiones, se utilizó Passport junto con express-session. Passport es una librería de autenticación para Node.js que facilita la implementación de estrategias de autenticación como *login local* u *OAuth*, entre otras.

5.1.3 Mongodb como base de datos

MongoDB fue seleccionada como la base de datos para este proyecto debido a su flexibilidad y escalabilidad. MongoDB es una base de datos *NoSQL* que almacena datos en documentos tipo *JSON*, lo que facilita su manejo y permite almacenar datos no estructurados o semi-estructurados de manera eficiente.

Para interactuar con MongoDB, se utilizó *Mongoose*, una biblioteca que proporciona una capa de abstracción sobre MongoDB, permitiendo definir modelos

para los datos. Los modelos en Mongoose permiten definir la estructura de los documentos, validar los datos y trabajar con métodos.

Se crearon varios modelos para representar diferentes tipos de datos. Estos son:

- **user**: contiene los datos de los distintos tipos de usuario
- **game**: contiene los datos asociados a un juego y su game-token
- **record**: contiene el esquema de una traza xAPI, con sus campos obligatorios y opcionales
- **group**: representa a un grupo de alumnos con un profesor asociado
- **gamesession**: contiene los datos de una sesión de juego asociada a un grupo

Para visualizar el contenido almacenado en la base de datos durante el desarrollo, se empleó la herramienta como MongoDB Compass, que proporciona una interfaz gráfica para consultar y manipular los datos.

5.1.4 Configuración de CORS

CORS (Cross-Origin Resource Sharing) permite a los servidores controlar el acceso a sus recursos desde diferentes orígenes, lo cual es esencial para aplicaciones web que interactúan con múltiples dominios.

En nuestra aplicación, se establece una configuración de CORS para las rutas /records y /api/session, permitiendo solicitudes desde los dominios de la web de análisis y del entorno de juego (especificados por los puertos correspondientes). Esta configuración incluye los métodos HTTP permitidos (GET, POST, PUT, DELETE, OPTIONS) y permite el intercambio de credenciales (cookies y encabezados de autenticación).

Por defecto, la configuración de cors permite tres puertos: el puerto 3000 para peticiones internas, el puerto 8080 para la web y el puerto 8081 para los juegos. Estos

puertos son definidos como variables de entorno, y se pueden cambiar. Existe más información en el README del repositorio.

Para todas las demás rutas, dedicadas exclusivamente a la interacción con la web, se aplica una configuración de CORS similar para asegurar que el acceso sea consistente y controlado a lo largo de toda la aplicación. Además, la configuración de CORS también se aplica a las conexiones de Socket.io, asegurando que las comunicaciones en tiempo real entre el servidor y los clientes sean seguras y permitan el intercambio de credenciales.

Esta configuración se implementa mediante el middleware ***cors***, que facilita la configuración.

5.1.5 Dependencias de desarrollo: dotenv y nodemon

Durante el desarrollo del servidor, se utilizaron varias dependencias para facilitar y optimizar el proceso de desarrollo:

- **dotenv**: Permite cargar variables de entorno desde un archivo .env en el proceso de Node.js. Esto es útil para mantener configuraciones sensibles, como claves de API, secretos de sesión o sales empleadas en el cifrado de algunos datos.
- **nodemon**: Una herramienta que reinicia automáticamente el servidor de Node.js cada vez que se detecta un cambio en el código fuente. Esto aceleró el ciclo de desarrollo al eliminar la necesidad de reiniciar manualmente el servidor cada vez que se realizaba una modificación.

5.2 Página de Desarrollador

En la aplicación, los desarrolladores tienen acceso a una página dedicada donde pueden gestionar los juegos que han creado, con la ruta */dev*. Esta página les permite realizar diversas operaciones relacionadas con sus juegos, tales como la creación, visualización, actualización y eliminación de juegos. A continuación, se describe cómo se implementan estas funcionalidades en el servidor.

5.2.1 Visualización de Juegos

El desarrollador puede ver todos los juegos que ha creado mediante una solicitud *GET* a la subruta */games*. El servidor responde con una lista de todos los juegos asociados al nombre del desarrollador que ha iniciado sesión. Esta funcionalidad permite al desarrollador acceder rápidamente a la información de sus juegos y gestionarlos de manera eficiente.

5.2.2 Creación de Nuevos Juegos

Para añadir un nuevo juego, el desarrollador realiza una solicitud *POST* a la subruta */games*. En el cuerpo de la solicitud, se incluye el nombre del juego y, opcionalmente, una descripción. El servidor genera un ID único y un token aleatorio para el nuevo juego, y asocia este juego con el desarrollador que ha iniciado sesión. Luego, guarda el juego en la base de datos y devuelve los detalles del juego recién creado como respuesta. Este proceso asegura que cada juego tenga un identificador y un token únicos, necesarios para la autenticación y la identificación del juego en el sistema, así como el envío de trazas.

5.2.3 Generación de IDs y Tokens Aleatorios

Para garantizar que cada juego tenga un identificador único y un token seguro, el servidor utiliza funciones especiales que generan un ID basado en una combinación

del tiempo actual y caracteres aleatorios. Estas funciones aseguran la unicidad y la seguridad de los juegos dentro del sistema.

5.2.4 Actualización de Juegos

Los desarrolladores pueden actualizar los detalles de un juego específico enviando una solicitud *PATCH* a la subruta */games/:id*, donde :id es el ID del juego que se desea modificar. El desarrollador puede actualizar el nombre y la descripción del juego. El servidor valida y aplica los cambios en la base de datos, asegurando que la información del juego esté siempre actualizada.

5.2.5 Eliminación de Juegos

Si un desarrollador decide eliminar uno de sus juegos, puede hacerlo mediante una solicitud *DELETE* a la subruta */games/:id*. El servidor elimina el juego especificado de la base de datos y confirma la eliminación enviando un mensaje de éxito al cliente. Esta funcionalidad es crucial para que los desarrolladores mantengan su lista de juegos organizada y libre de elementos obsoletos.

5.2.6 Recuperación de Juegos Específicos

El middleware *getGame* se utiliza en las rutas que requieren el acceso a un juego específico. Este middleware busca un juego en la base de datos utilizando su ID. Si se encuentra el juego, se adjunta a la respuesta para su uso en la siguiente función de middleware. En caso contrario, se envía un mensaje de error al cliente. Esta capa adicional de verificación asegura que solo se trabajen con juegos válidos, reduciendo errores y mejorando la robustez de la aplicación.

5.3 Página de Profesor

La página de profesor permite a los profesores gestionar grupos de estudiantes y sesiones de juego. A través de esta página, los profesores pueden realizar las siguientes acciones

5.3.1 Gestionar grupos de estudiantes

5.3.1.1 Crear un grupo de estudiantes

Los profesores pueden crear grupos de estudiantes para generar usuarios para sus alumnos. Pueden crear esos usuarios desde cero usando la solicitud *POST /create-group-from-scratch*, y proporcionando el nombre del grupo y el número de estudiantes. El sistema genera automáticamente los nombres de usuario para los estudiantes y guarda el nuevo grupo en la base de datos. Por otro lado, también es posible crear un grupo proporcionando un nombre y una lista de estudiantes mediante la solicitud *POST /create-group-with-students*. El sistema genera los nombres de usuario para los estudiantes listados, crea el grupo y lo guarda en la base de datos. La respuesta incluye la información del grupo creado, o un mensaje de error si algo falla.

5.3.1.2 Obtener grupos

Los profesores pueden recuperar todos los grupos que tienen a su cargo mediante la solicitud *GET /get-groups* o bien obtener información de un grupo específico con una solicitud *GET /get-group/:id*, donde *:id* es el identificador del grupo. Si se encuentra algún grupo, se devuelve una lista de estos en formato JSON. Por el contrario, si el grupo no se encuentra o ocurre un error durante la consulta, se envía un mensaje de error.

5.3.2 Gestionar sesiones de juego

5.3.2.1 Crear una sesión de juego

Los profesores pueden crear una nueva sesión de juego usando la solicitud *POST /create-game-session*. En esta solicitud, deben proporcionar el identificador del grupo, el identificador del juego y el nombre de la sesión de juego (opcional). El sistema genera claves de sesión para cada estudiante del grupo y registra una nueva sesión de juego en la base de datos. Estas claves son las que luego introducirán los estudiantes en los juegos para poder enviar trazas asociadas a esa sesión. Además, los estudiantes pueden usar esas claves para acceder a la página y ver sus analíticas. La información de la nueva sesión se devuelve en la respuesta, o un mensaje de error si ocurre algún problema.

5.3.2.2 Obtener sesiones de juego

Los profesores pueden consultar todas las sesiones de juego asociadas a todos sus grupos mediante una solicitud *GET /get-game-sessions*, o todas las sesiones de juego de un grupo en específico mediante la solicitud *GET /get-game-sessions/:groupId*, donde *:groupId* es el identificador del grupo. Esta acción recupera los grupos del profesor y luego busca sesiones de juego relacionadas con esos grupos. En caso de éxito, se devuelve una lista de sesiones de juego. Si no se encuentran grupos o se produce un error, se devuelve un mensaje de error.

5.4 Gestión de trazas

5.4.1 Almacenamiento de trazas en la base de datos

Las trazas que llegan al servidor son enviadas por la librería que se integra en los juegos educativos. Estas se envían al servidor mediante una petición *POST* a la ruta */records*.

5.4.1.1 Autorización del envío

Antes de almacenar la traza, el servidor debe validar que el envío de esa traza es autorizado. Para ello, se emplea el *game-token* que los desarrolladores pueden crear para sus juegos y luego deben incluir en la cabecera x-authentication de la petición HTTP.

El middleware *verifyToken* es el encargado de verificar que cada solicitud enviada al servidor incluya este token en la cabecera x-authentication. Primero, extrae el token de la cabecera y comprueba si está presente. Si falta, responde con un error 403 (Forbidden). Luego, busca en la base de datos un juego que coincida con ese token. Si no se encuentra un juego asociado, responde con un error 401 (Unauthorized). Si el token es válido, permite que la solicitud continúe hacia la siguiente función. Si ocurre un error durante el proceso, responde con un error 500 (Internal Server Error).

5.4.1.2 Verificación de la clave de sesión

Como ya hemos visto antes, las trazas pueden incluir claves de sesión de los estudiantes para poder identificar correctamente la actividad a la hora de realizar analíticas dentro del contexto Jaxpi. Si la traza no incluye dicha clave, se almacena directamente en la base de datos. Pero por el contrario, si la traza contiene un *session-key* en su contexto, el servidor realiza una serie de operaciones para enriquecer la traza con información adicional:

- **Búsqueda de la Sesión:**

  El *session-key* se utiliza para buscar una sesión de juego en la base de datos (*GameSession*). Si la sesión no se encuentra, se devuelve un error 404.

- **Validación del Token y el Juego:**

  Se busca el juego asociado a la sesión (Game) y se verifica que el token proporcionado coincida con el token almacenado para ese juego. Si no coinciden, se devuelve un error 403.

- **Obtención del Grupo y el Instructor:**

  Se obtiene el grupo asociado a la sesión (*Group*) y el profesor que supervisa el grupo (*User*). Si alguno de estos datos no se encuentra, se devuelve un error 404.

- **Enriquecimiento del Contexto:**

  El servidor modifica el contexto de la traza añadiendo información sobre el instructor, el juego, la sesión y el grupo. Estos datos adicionales se almacenan en la traza para proporcionar un contexto más completo y útil en análisis posteriores:

  - Se añade la información del *instructor* (nombre y correo electrónico) al contexto de la traza.
  - Se agregan los identificadores del juego y de la sesión.
  - Se añade el identificador del grupo en el que el estudiante está participando.

5.4.1.3 Cifrado de la clave de sesion

Si la traza incluía *session-key*, es necesario proteger la privacidad de los usuarios. Para ello, el *session-key* se *hashea* antes de ser almacenado en la base de datos. El hashing se realiza utilizando el algoritmo SHA-256, junto con un salt y una frase secreta. Solo se almacena el hash del session-key, lo que asegura que no se pueda rastrear el session-key original.

5.4.1.4 Almacenamiento en la base de datos

Después de realizar todas las verificaciones y enriquecimientos, se crea un nuevo registro (*Record*) con la traza modificada. Este registro se guarda en la base de datos MongoDB. Durante el almacenamiento, MongoDB añade los campos _id y __v, y convierte los campos timestamp y stored a tipo Date.

5.4.1.5 Emitir la traza por el socket

Después de almacenar la traza, se emite un evento en tiempo real que notifica a los clientes conectados a través del websocket (más detalles más adelante) que una nueva traza ha sido registrada. Para esta emisión, se utiliza una versión de la traza sin el session-key hasheado.

5.4.1.6 Respuesta al cliente

Finalmente, el servidor responde con un código de estado 201 y la traza registrada, confirmando que se ha almacenado correctamente.

5.4.2 Devolución de Trazas al Cliente

La funcionalidad de devolución de trazas al cliente es una parte crítica del servidor, ya que permite que los distintos tipos de usuarios (estudiantes, profesores y desarrolladores) accedan a las trazas almacenadas, adaptadas según su rol y nivel de acceso. La web de análisis puede acceder a las trazas a través de una petición **GET** en la ruta */records*, la cual está protegida por el middleware *checkAuthenticated* que asegura que solo los usuarios autenticados puedan acceder a los datos.

Una parte importante del proceso es la función *unhashSession*, que toma una clave de sesión hasheada y la compara con las claves de sesión originales en la base de datos, devolviendo la original si hay coincidencia. Esto es crucial para obtener el usuario al que pertenece la traza.

El proceso de devolución de trazas varía dependiendo del tipo de usuario que realiza la solicitud.

5.4.2.1 Devolución de trazas a usuarios estudiantes

- **Identificación del grupo:** Se busca el grupo al que pertenece el estudiante basándose en su nombre de usuario. Si se encuentra el grupo, se recuperan las trazas asociadas a ese grupo y al profesor que lo gestiona.
- **Agrupación y filtrado de trazas:** Las trazas se agrupan por el ID del grupo y la clave de sesión. Luego, se filtran para que el estudiante solo pueda ver sus propias trazas.
- **Desencriptado de claves de sesión:** Se utiliza la función *unhashSession* para convertir las claves de sesión hasheadas a su formato original.
- **Asignación de nombres**: A cada actor en las trazas se le asigna su nombre real utilizando la información de la base de datos, permitiendo al estudiante ver sus datos personales.
- **Emisión de datos:** Los datos procesados se devuelven al cliente en formato JSON.

5.4.2.2 Devolución de trazas a usuarios profesores

- **Recuperación de trazas:** Se obtienen todas las trazas relacionadas con el profesor, comparando el nombre de usuario que realiza la petición con el campo *instructor* de las trazas.
- **Agrupación por contexto:** Las trazas se agrupan por ID de grupo y clave de sesión, similar al caso de los estudiantes.
- **Descifrado y asignación:** Se descifran las claves de sesión y se asignan los nombres reales de los estudiantes involucrados en las trazas.

- **Asignación de nombres de grupo y sesión:** Se añaden los nombres del grupo y la sesión correspondientes a cada traza.
- **Emisión de datos:** Los datos agrupados y procesados se devuelven en formato JSON.

5.4.2.3 Devolución de trazas a usuarios desarrolladores

- **Filtrado por juegos:** Se recuperan las trazas relacionadas con los juegos desarrollados por el usuario, excluyendo información sensible como el actor y el instructor.
- ***"Pseudonimización"***: Es fundamental para nuestra aplicación proteger la privacidad de los usuarios, especialmente para los estudiantes ya que estos pueden ser menores de edad. Para proteger la privacidad de los estudiantes, se generan pseudónimos en lugar de mostrar nombres reales. Esto se realiza mediante una función que convierte el nombre en un pseudónimo usando un hash MD5.
- **Asignación de nombres de sesión**: A diferencia de los estudiantes y profesor, las claves de sesión no se descifran, si no que se le asigna el pseudónimo creado para el alumno.
- **Emisión de datos:** Finalmente, los datos se devuelven en formato JSON.

5.4.3 Actualización de trazas mediante WebSockets

Como ya hemos visto antes, al enviar las trazas se realiza la emisión de un evento socket. Esto se debe a que la aplicación web carga una vez todas las trazas existentes para el usuario, pero para obtener actualizaciones habría que estar constantemente recargando la página. Para solucionar esto, al enviar una traza mediante una solicitud POST se emite un evento de nueva traza mediante la librería *socket.io*, una librería que permite la comunicación de baja latencia, bidireccional y

basada en eventos entre un cliente y un servidor[18]. En el evento se envía el contenido de la traza al cliente conectado al socket. De esta manera, las trazas nuevas llegan al cliente sin necesidad de estar recargando la página

5.5 Testing

5.5.1 Uso de un archivo route.rest para testear peticiones HTTP sencillas

En el proyecto, se utiliza un archivo *route.rest* ubicado en el directorio/test para realizar pruebas rápidas y sencillas de las API. Este archivo permite ejecutar peticiones HTTP directamente desde Visual Studio Code, sin necesidad de herramientas externas como Postman. Contiene ejemplos de peticiones GET, POST, PUT y DELETE que se pueden ejecutar para verificar el correcto funcionamiento de los endpoints.

5.5.2 Uso de scripts elaborados con axios para generar una base de datos

Para poblar la base de datos con datos de prueba, se han creado varios *scripts* utilizando la biblioteca axios. Estos scripts se encuentran en el directorio /test y permiten automatizar la creación de usuarios, en la base de datos. Los scripts realizan peticiones HTTP POST a los endpoints correspondientes para crear los datos necesarios. Esto facilita la creación de datos de prueba consistentes, lo que es crucial para realizar pruebas automatizadas y garantizar la calidad del software. Además, estos scripts permiten verificar que los endpoints funcionan correctamente y que los datos se almacenan de manera adecuada en la base de datos.

Estos *scripts*, funcionan principalmente para generar usuarios de todos los tipos (estudiantes, profesores y desarrolladores) con los que probar la aplicación. Inicialmente también lo usábamos para enviar trazas de prueba directamente a la

[18] https://socket.io/docs/v4/

base de datos que emulasen la actividad de un estudiante en un juego. Pero a medida que avanzaba el proyecto llegamos a la conclusión de que no era escalable para probar la aplicación con casos reales, ya que estos pueden tener miles de trazas.

5.5.3 Testeando desde la propia librería

Como acabamos de ver, para generar datos con los que realizar el análisis lo más idóneo era emplear la librería para emitir trazas al servidor directamente. Además, esto nos permitió comprobar que las funcionalidades de la librería de envío de trazas eran correctas.

6. Análisis y visualización de datos

En esta parte del proyecto, podremos ver la unión de las 3 secciones del proyecto trabajando mano a mano. Realizamos estadísticas y gráficos de los datos obtenidos de las interacciones de los alumnos con videojuegos web en nuestra aplicación, para que el usuario pueda explorar y comprender la información necesaria de una forma clara e intuitiva, tanto si es desarrollador, profesor o alumno. Puede ser de gran utilidad para instituciones educativas ya que en la actualidad, en muchos centros educativos están introduciendo nuevas herramientas. Nosotros, nos hemos querido focalizar en el uso de los videojuegos educativos como un gran aliado a la hora de mejorar el proceso de aprendizaje de los alumnos a través de datos que, durante mucho tiempo, han estado desaprovechados. Con esta visión de querer contribuir en la educación, podemos dar un gran valor a todos estos datos.

Veamos qué hemos empleado para llevar a cabo el análisis de datos y su posterior visualización en la aplicación web.

6.1 Vue como framework

Para el procesamiento de los datos obtenidos y la creación de nuestra aplicación web, elegimos Vue.js frente a otras opciones similares como Angular, por ejemplo. Decidimos investigar sobre Vue.js, ya que ninguno de los integrantes del equipo lo había utilizado anteriormente y vimos la ocasión perfecta para aprender de ella y desarrollar la interfaz de usuario.

Vue se adapta perfectamente al proyecto que queremos construir ya que nos permite crear componentes reutilizables, gestionar su sistema de reactividad de una forma más eficiente y todo ello con una sintaxis muy intuitiva y limpia.

6.1.1 Vue 3 y Composition API

Elegimos utilizar Vue 3 en lugar de Vue 2 por sus numerosas mejoras y nuevas características como las mejoras en el rendimiento, ya que su nuevo motor de renderizado está basado en proxies del virtual DOM, lo que le ha añadido rapidez y eficiencia. Y la *Composition API*, el cual es el mayor cambio que nos encontramos en esta versión frente a Vue 2 (Navarro Morales, 2021) con *Options API*.

Composition API es una de las formas principales de escribir componentes en Vue y es el futuro de Vue. Por ello, aunque comenzamos nuestra aplicación con *Options API*, inmediatamente pasamos a utilizar *Composition API* ya que nos permitía organizar el código de una forma más modular y reutilizable, que era algo que buscamos tener en nuestro proyecto a la hora de ayudarnos a escribir código.

También utilizamos *script setup*, una nueva característica que simplifica la sintaxis para utilizar la *Composition API*. Puedes escribir tu código directamente sin la necesidad de una función setup() explícita como se hacía tradicionalmente. Ahora, dentro de la etiqueta *<script setup>* declaramos directamente nuestras variables reactivas, funciones y código, haciendo un código menos repetitivo.

6.1.2 Vue Router

Vue Router es una biblioteca que agrega enrutamiento a las aplicaciones de Vue, es decir, podemos cambiar dinámicamente la vista que se muestra al usuario sin tener que recargar toda la página. Es el enrutador oficial para Vue.js por lo que es la opción más recomendada a la hora de construir una *SPA* con Vue.

Simplifica el enrutamiento y proporciona las herramientas necesarias para manejar una navegación más directa y fluida. Se encuentra en nuestra carpeta */src/router/index.js*, donde hemos definido las diferentes rutas de nuestras vistas, a las que accedemos a través de enlaces, botones o manejadores de eventos.

Gracias a *Vue Router*, podemos utilizar la navegación opcional con los guardianes de navegación o *Guard global,* en el que controlamos el acceso a ciertas rutas de la aplicación basándonos en la autenticación del usuario. En nuestro caso, si un usuario ya ha iniciado sesión y vuelve a la ventana de Inicio de sesión, se desconecta y se redirige a este usuario inmediatamente a la página de inicio. De este modo, nos aseguramos de que el usuario tenga un flujo de navegación seguro y coherente.

6.1.3 Gestores de estado

Un gestor de estados nos permite tener la información de manera centralizada, es decir, en toda nuestra aplicación podemos acceder a la misma información y al mismo tiempo, y si sucede algún cambio en algún lugar, se verá reflejado en toda la aplicación. Se les conoce comúnmente como store, almacén o tienda. Aquí, haremos referencia al término tienda ya que es más común utilizar este término en el contexto del uso de Vue y *Pinia*.

Vue ofrece diferentes gestores de estados. Los más conocidos y utilizados son *Vuex* y *Pinia*.

6.1.3.1 Pinia

Nos decantamos por *Pinia*[19] como gestor de estados para nuestra aplicación ya que es el gestor de estados oficial de Vue. Es considerado el sucesor de Vue, ofreciendo una interfaz más sencilla e intuitiva. Surgió precisamente al buscar rediseñar el gestor de estados con el uso de la *Composition API* junto a Vue 3, aunque sigue siendo compatible con Vue 2. *Pinia* nos permite construir aplicaciones más escalables y fáciles de mantener.

Estas son algunas ventajas a destacar que nos ofrece el uso de *Pinia*:

[19] https://vueschool.io/lessons/introduction-to-pinia

- **Separación de la lógica:** podemos modularizar la lógica de una manera muy efectiva. Por ejemplo, la lógica de autenticación de usuarios la tenemos en una tienda, en otra, tenemos todo lo relacionado con las clases, etc. Esto nos ayuda a una mejor organización del código.
- **Modularización:** la idea es tener más de una tienda, no solo una. Cómo hemos citado en el párrafo anterior, tenemos diferentes tiendas, cada una con su finalidad para así saber a qué tienda debo acudir. Podríamos tener una sola tienda para toda la aplicación pero nos resultaría más difícil de entender.
- **Menos llamadas al servidor**: al almacenar la información más relevante para toda la aplicación mediante *Pinia*, podemos llegar a hacer menos de la mitad de las llamadas al servidor que si se hacen sin el gestor de estados. Esto es especialmente útil en nuestro tipo de aplicación de una sola página.

6.2 Estructura del proyecto

A continuación, hablaremos sobre los aspectos más relevantes de la estructura de la aplicación web para saber cómo está organizada.

6.2.1 Uso del Single-File Components (SFC)

Utilizamos los componentes de un solo archivo, que son una característica definitoria de Vue, lo que nos permite tener en un solo archivo: plantilla (*HTML*), *script* (*JavaScript*) y estilo (*CSS*). Es decir, tener la estructura, lógica y estilos juntos en el mismo archivo (tipo *.vue). En la Figura 14, podemos ver un ejemplo de esta distribución en nuestros archivos *.vue, en este caso, *SelectRoleView.vue*.

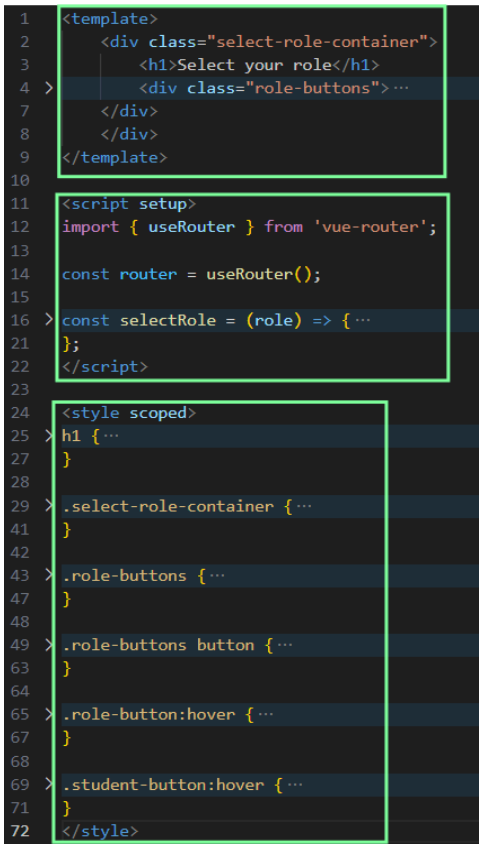

Figura 14. Cuerpo del archivo SelectRoleView.vue.

6.2.2 Organización de los directorios

El proyecto se organiza en varias carpetas, cada una de ellas con una finalidad. La figura 15, es una vista general de nuestra carpeta en *Visual Studio Code*:

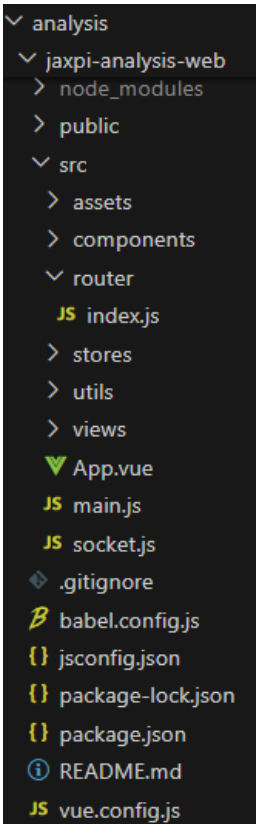

Figura 15. Estructura general de la carpeta del proyecto.

- En la carpeta **/node_modules** se encuentran todas las dependencias generadas mediante *NPM* (el gestor de paquetes). Se genera automáticamente cuando se instalan paquetes.
- Nuestra carpeta principal y la más importante es **/src,** ya que contiene todo nuestro código fuente y diferentes carpetas que tenemos que nombrar:
  - **/assets**: contiene las imágenes que utilizamos en la aplicación.
  - **/components**: tiene todos los componentes reutilizables que hemos creado. Tiene diferentes componentes y subcarpetas, con la intención de tener los componentes mejor organizados. Y son los siguientes:

- Las carpetas **/dev**, **/teacher** y **/student:** componentes específicos para cada tipo de usuario. Como sus respectivos formularios, listas de clases, juegos o estudiantes, etc.
- Las carpetas **/modals** y **/navigation:** tienen los componentes que dibujan diálogos emergentes en la interfaz y los relacionados con la navegación. En la aplicación tenemos una barra de navegación y una barra lateral.
- Los componentes **DataTable.vue**, **PaginationComponent.vue, BarChart.vue**, **LineChart.vue**, **PieChart.vue**, etc: todos estos componentes se ubican directamente en */components,* ya que son de uso general para los diferentes tipos de usuarios que tiene la aplicación.

➢ **/router**: contiene el archivo *index.js* con la configuración de *Vue Router*. Aquí, definimos las rutas de las vistas y los guardias de navegación. En la Figura 16, podemos ver algunas vistas iniciales del archivo.

Figura 16. Contenido del archivo /src/router/index.js.

➢ **/stores:** gestiona el estado global usando *Pinia,* nuestro gestor de estado elegido. *authStore.js*, *groupsStore.js, gamesStore.js* y *gameSessionsStore.js,*

son algunas de nuestras tiendas que gestionan respectivamente el estado relacionado con la autenticación, las clases, los juegos, las sesiones de juego, etc. En la figura 17 vemos el ejemplo de la tienda de sesiones de juego, con sus principales secciones: estado o "state", acciones o "actions" y los "getters".

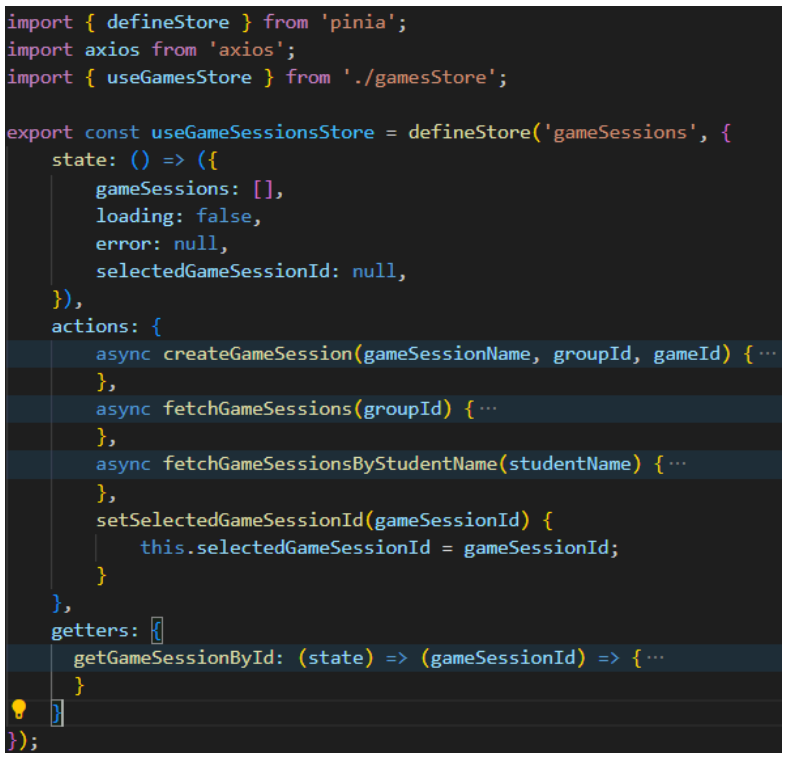

Figura 17. Código de la tienda *gameSessionsStore.js*.

En el estado de la figura 17, tenemos el objeto que contiene la información de la tienda de sesiones de juego. Tenemos *gameSessions*, la lista de sesiones de juego, *loading*, campo que nos indica si una operación asíncrona está en ejecución, *error*, que guarda cualquier mensaje de error ocurrido y *selectedGameSessionId*, donde

almacenamos el identificador de la sesión de juego seleccionada actualmente.

Por otro lado, en acciones, definimos los métodos que pueden modificar el estado. En nuestro caso tenemos:

- **createGameSession()**: crea una nueva sesión de juego realizando una solicitud POST al servidor con el nombre de la sesión, el identificador de la clase y del juego proporcionados. Esta acción la realiza un profesor cuando ya tiene una clase creada. Veremos más detalles de estas condiciones en la sección 6.4.
- **fetchGameSessions():** nos permite recuperar desde el servidor las sesiones de juego dado el identificador de una clase en concreto. También la realiza el profesor desde cada una de sus clases.
- **fetchGameSessionsByStudentName():** recupera las sesiones de juego pero en este caso cuando el usuario es un estudiante. Con su nombre único.
- **setSelectedGameSessionId():** permite establecer el identificador de la sesión de juego seleccionada para tener un mejor control de este campo en la interfaz.

Y en cuanto a los *getters,* son funciones que nos permiten acceder a los datos que tenemos almacenados en el estado de una manera específica o transformada, manteniendo el estado original sin modificaciones. Tenemos la función *getGameSessionById(),* que devuelve una sesión en concreto dado su identificador.

➢ **/utils:** tenemos el archivo *utilities.js*, donde se encuentran las funciones utilitarias, usadas en diferentes partes de la aplicación y es por ello que

las hemos alojado aquí. De esta manera, podemos procesar y manipular los datos que queremos analizar y luego representarlos en gráficos.

- **calculateLevelCompletionTimes():** procesa los datos de todos los estudiantes de una clase, ordena las trazas de cada uno, y calcula los tiempos de finalización de cada nivel del juego. Devuelve una estructura de datos, la cual nos va a ayudar luego a generar analíticas para mostrarla en diferentes gráficos.
- **calculateForStatements():** esta función es llamada por la función anterior *calculateLevelCompletionTimes()* y en diferentes partes de la aplicación ya que es la que realmente extrae toda la información posible de los datos que deseamos analizar.
- **sortStatements():** recibe una lista de trazas y las ordena por su fecha de creación. Esto es especialmente importante para garantizar el orden correcto del análisis de las trazas de los jugadores.
- **calculateScorePerLevel():** si en las trazas de un jugador tenemos un campo de puntuación, calculamos la mejor y peor puntuación por nivel de ese jugador. Sería similar si queremos obtener los datos para puntuación/sesión de juego o puntuación/día, por ejemplo. No hemos podido utilizarla, pero son estadísticas que pondremos en trabajo futuro.

➢ **/views:** otra carpeta fundamental, ya que contiene todas las vistas de la aplicación. Una vista es un componente de nivel superior, ya que organiza en ella a otros componentes y representa una página completa. Cada vista tiene asociada una ruta en el archivo */src/router/index.js*, como vimos anteriormente.

Para entender la diferencia entre una vista y un componente, una vista es una página completa y puede tener muchos componentes en ella, mientras que un componente es un bloque reutilizable, más pequeño y más especializado.

Tenemos una distribución similar a la carpeta */components:*

- Las carpetas **/dev**, **/teacher** y **/student**: agrupamos las vistas específicas por cada tipo de usuario. Como las vistas de creación de juegos, clases y sesiones de juego, de los detalles de un juego, una clase o un estudiante en concreto, o las vistas de las listas de juegos y clases, entre otras muchas.
- Las vistas **HomeView.vue**, **LoginView.vue**, **SelectRoleView.vue**, **RegisterView.vue**, **AboutUsView.vue** y **NotFoundView.vue**: son las vistas generales accesibles por todos los tipos de usuarios.

➢ **App.vue:** es el componente raíz de la aplicación. El contenedor de los componentes y vistas donde configuramos la estructura principal de la interfaz de usuario.

Aquí, en la plantilla definimos una barra de navegación en la cabecera (*NavBar.vue*), un menú lateral (*SidebarMenu.vue*) que mostramos a los usuarios del tipo desarrollador y profesor en ciertas rutas, y un contenedor con contenido dinámico dependiendo de las rutas en las que se encuentre el usuario para mostrar las vistas con <router-view> sin necesidad de recargar toda la página.

En su *script*, manejamos cuando mostrar el menú lateral, importamos el *socket* desde *sockets.js* para manejar la comunicación con el servidor en tiempo real, observamos cambios en la autenticación del usuario y cuando esto ocurre, enviamos al servidor los datos del

usuario a través del *socket*. Por último, gestionamos la desconexión del *socket* cuando nuestro componente se desmonta para asegurarnos de liberar recursos.

En los estilos definimos clases globales, para usar en cualquier lugar de la aplicación, por ejemplo, tarjetas del mismo estilo, botones, tamaños y diseños de contenedores, etc. Aunque cada componente tenga además su propio estilo, nos añade comodidad y menos redundancia a la hora de escribir las mismas líneas de código.

- **main.js:** archivo principal de configuración que ejecuta el código necesario para crear y arrancar nuestra aplicación de Vue. Importamos las funciones necesarias de *Vue* y *Pinia,* el componente *App.vue* y el enrutador. Creamos la instancia de la aplicación con *createApp(App)*, y le añadimos *Pinia*, el enrutador y montamos la aplicación para arrancarla con *app.use(pinia).use(router).mount('#app')*.
- **sockets.js:** código de la configuración inicial del *socket* con el servidor. Creamos una instancia de *Socket.io* estableciendo la conexión en el *localhost* en el puerto 3000 para enviar y recibir datos en tiempo real. Por último, exportamos esta instancia para poder utilizarla en cualquier parte de la aplicación, como en *App.vue*.

```
import { io } from 'socket.io-client';

const socket = io("http://localhost:3000");

export default socket;
```

Figura 18. Código del archivo *socket.js*.

- En el fichero **package.json** es donde gestionamos la información de las dependencias y herramientas utilizadas en la aplicación. En la figura 19 se ven las versiones de las principales dependencias que usamos (Axios, Pinia, C3, Socket.io, etc).

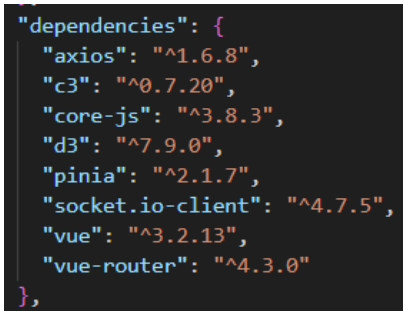

Figura 19. Principales dependencias de nuestro proyecto.

6.3 Enfoque inicial de los datos y herramientas para los gráficos

En nuestro proyecto, los datos que nos interesa analizar son los generados por los juegos *JavaScript*, es decir, trazas *JSON* para evaluar el comportamiento del jugador. Estuvimos investigando sobre cómo realizar analíticas de estos datos para poder mostrarlos de la manera más clara e intuitiva y que nos resulten útiles. En vez de mostrar datos en crudo, que aunque son de gran valor para análisis detallados, a la mayoría les podría resultar abrumador y de difícil interpretación si no están familiarizados con los conceptos estadísticos.

6.3.1 Métricas relevantes

Tenemos que elegir las métricas adecuadas que nos permitan analizar correctamente el uso del juego que realizan los jugadores. Al realizar una lluvia de

ideas a la hora de evaluar qué métricas nos pueden resultar más relevantes, primero, evaluamos los diferentes géneros de juego que existen, para luego, ver sus métricas.

Esta tabla muestra los géneros de juego que hemos evaluado junto a sus métricas más relevantes:

Género	Métricas
Plataformas	Número de objetos recogidos, tiempo por nivel, número de vidas usadas,
Aventura gráfica	Uso de objetos, evolución de un personaje, interacciones con NPC, misiones completadas, elecciones del jugador…
Puzzles	Tiempo en resolver, número de intentos, niveles completados, puntuaciones…
Carrera	Vehículos escogidos, porcentaje en carreras ganadas y carreras perdidas, selección de pistas, tiempo en completar una carrera…
Todos	Niveles completados, tiempo en completar un nivel, tiempo jugando, número de interacciones con objetos, vidas perdidas, clicks…

Tabla 1. Géneros de juegos y sus métricas.

Como podemos observar, cada tipo de juego tiene métricas muy específicas, pero también métricas comunes entre ellos.

Para nuestro proyecto escogeremos las métricas comunes más relevantes, ya que nuestra finalidad es poder analizar el mayor número de géneros de juegos posibles. Estas métricas nos ayudarán a responder a las preguntas respecto a la participación de los jugadores en los distintos niveles, ver su progreso a lo largo del

juego, sus tiempos de compleción por niveles, así como las interacciones que realizan con los diferentes elementos que se le presentan para obtener una visión general del comportamiento de los jugadores.

En la sección 7.1, podremos ver cómo se aplican en la práctica estas métricas en el análisis de este juego.

6.3.2 Tipo de visualizaciones

Para representar las métricas descritas en la sección anterior, hemos elegido algunos formatos que pueden facilitar su comprensión y su representación. Los gráficos son una gran opción para la visualización de los datos. Con ellos podemos identificar patrones, tendencias y dificultades más rápidamente. Y, junto a las tablas y cifras significativas, nos ayudarán a tener una visión más completa del análisis.

Estos son algunos de los gráficos que más se utilizan para métricas de juegos:

- **Gráficos de barras**: representan datos en forma de barras verticales u horizontales. Pueden ser gráficos de barras individuales, donde cada barra representa un sólo conjunto de datos, o de barras agrupadas, para comparar diferentes conjuntos de datos por categoría. Se utilizan mucho para mostrar el tiempo/nivel, tanto el medio, mejor o peor, o para mostrar el número de interacciones por elemento, entre otros muchos. Es muy usado ya que resulta ser un gráfico familiar para la mayoría de usuarios al utilizar en muchos medios.
- **Gráficos de líneas:** se utilizan líneas para conectar y puntos y representar tendencias a lo largo del tiempo. Son ideales para ver el progreso de los jugadores a lo largo de los días de juego.
- **Gráficos de sectores:** también conocido como gráficos de tarta. Son muy útiles a la hora de representar porcentajes y ver de una manera rápida

esa distribución. Nosotros la utilizamos para mostrar el porcentaje de jugadores que han completado el último nivel y cuántos no, por ejemplo.

- **Gráficos de dispersión:** se muestran una gran cantidad de puntos para reflejar la relación entre el eje x y el eje y. Cuando se ven muchos puntos juntos, indica que hay una mayor concentración de valores similares en esas zonas. Muy útiles para identificar patrones entre las variables, por ejemplo, el porcentaje de interacción con un objeto en un periodo de tiempo.
- **Mapas de calor:** utiliza zonas de colores para diferenciar la concentraciones de los valores. Nos ayudan a ver en qué zonas el jugador interactúa más con ciertos elementos, por ejemplo. Así podremos detectar qué elementos no están siendo usados o cuáles se usan más y en qué proporción.

Los gráficos que hemos utilizado en la aplicación, los explicaremos con más detalle en la sección 7.1.

6.3.3 Selección de bibliotecas gráficas

Hemos considerado diferentes bibliotecas para crear estos gráficos. *Chart.js, HighCharts, FusionCharts, D3.js* y *C3.js,* entre otras. Todas ellas muy interesantes ya que tienen un amplio abanico de posibilidades para dibujar.

Inicialmente, consideramos utilizar *D3.js*, ya que vimos que era muy potente y flexible. Sin embargo, aunque era muy versátil, tenía una curva de aprendizaje muy pronunciada y finalmente, decidimos utilizar *C3.js*, ya que vimos que sigue usando D3 por debajo.

C3.js nos ofrece diferentes tipos de gráficos como los gráficos de barras *(Bar Chart y Stacked Bar Chart),* gráficos de líneas *(Line Chart),* gráficos de sectores (*Pie Chart),* gráfico de anillos (*Donut Chart), Gauge Chart,* etc. Cada una de ellas

altamente personalizables y disponen de diferentes opciones que puedes añadir, cómo poder aplicarles diferentes zooms, transformaciones, animaciones, cuadrículas, regiones, elegir colores e incluso añadir más ejes.

6.4 Aplicación web

A continuación, hablaremos sobre la aplicación web creada, detallando el diseño y las diferentes funcionalidades que se han desarrollado para mostrar las analíticas de los datos de los videojuegos web en base a los tipos de usuario que usan la aplicación.

6.4.1 Navegación por la aplicación

Cuando se accede por primera vez, un usuario que no ha iniciado sesión se encuentra con la vista principal llamada "*Home*" donde damos la bienvenida al usuario contándole a grandes rasgos que nuestra aplicación consiste en visualizar analíticas de juegos con los que previamente se han interactuado (Figura 20).

En la parte superior de nuestra pantalla, vemos el menú de navegación con las diferentes opciones iniciales que tienen los usuarios no autenticados. Estos botones nos dirigen a distintas pantallas como la de inicio, de iniciar sesión, de registro para nuevos usuarios y a quiénes somos. Una vez autenticado el usuario, este menú de navegación cambiará dependiendo del tipo de usuario que sea.

Figura 20. Pantalla de Inicio de la aplicación web.

Para comenzar, si no estamos registrados en la aplicación, debemos ir a "*Register*", donde se nos mostrará un formulario (Figura 21) que debemos rellenar con los campos obligatorios (marcados con un asterisco *) y si deseamos, también con los opcionales. Tenemos que señalar que si no se introducen los campos obligatorios en este y en todos los demás formularios, se mostrará un mensaje recordándote completar estos campos para poder continuar.

Este formulario es para registrar a un profesor o a un desarrollador de juegos, ya que el estudiante no tiene que hacer este paso, puesto que lo hace su profesor, pero de otra forma como veremos más adelante.

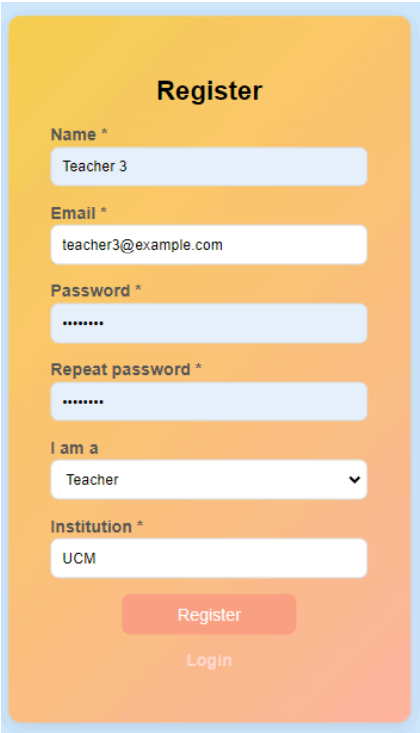

Figura 21. Formulario de registro.

Como nombramos en la sección 3.1, tenemos tres roles de usuario diferentes. Desarrollador, profesor y estudiante.

Para iniciar sesión vamos a "*Login*" del menú de navegación, y a continuación, lo primero que vemos es la selección del tipo de usuario (Figura 22). Desarrollador y profesor tienen que hacer click en "*Login*" y estudiante en "*Login as student*".

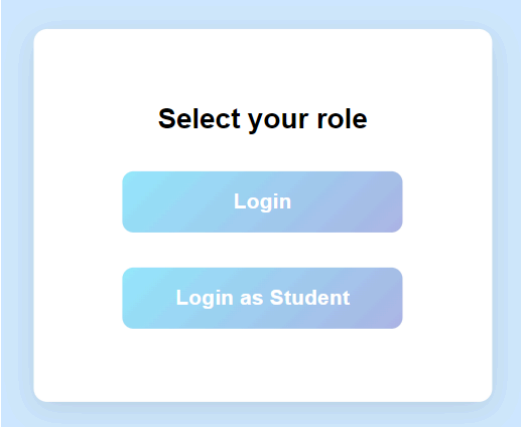

Figura 22. Menú para seleccionar el tipo de usuario para iniciar sesión.

Los desarrolladores y profesores tendrán que introducir correo electrónico y contraseña, y los estudiantes solo su clave de sesión (Figura 23). La clave del estudiante ha sido creada por su profesor, quitándoles así ese paso extra que en diferentes ocasiones, son muchos los estudiantes los que no llegan a crearse estas cuentas. Más adelante, veremos cómo un profesor crea esta clave para sus alumnos.

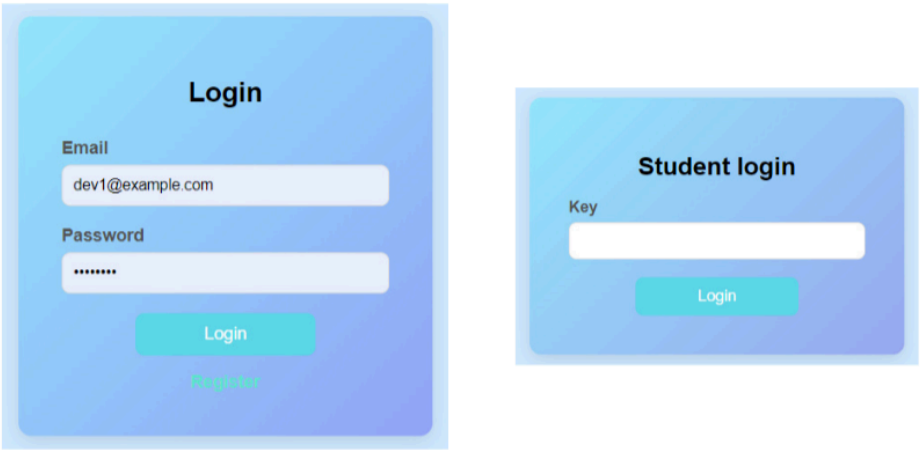

Figura 23. Menú de inicio sesión (para desarrollador y profesor, y para estudiante).

Con la sesión iniciada, vamos a ver las pantallas principales de cada tipo de usuario y sus funcionalidades.

Los desarrolladores pueden añadir sus juegos, dejarlos registrados y ver analíticas sobre ellos. Esto significa que hay gente jugando y podrás ver las estadísticas más relevantes de sus jugadas. Todo ello sin saber a qué persona en concreto corresponde ya que hay una privacidad de esos datos y esto es importante. Hemos hablado de cómo realizamos esta *pseudonimización* en la sección 5.4.2.3.

El desarrollador cuando inicia sesión, ve en su pantalla principal (Figura 24) sus datos personales y el número de juegos que tiene registrados. Cuando entra por primera vez, no tiene ningún juego registrado. Y mostramos un mensaje con las instrucciones para conocer cómo funciona la aplicación y qué debe hacer para registrar sus juegos y poder ver sus analíticas.

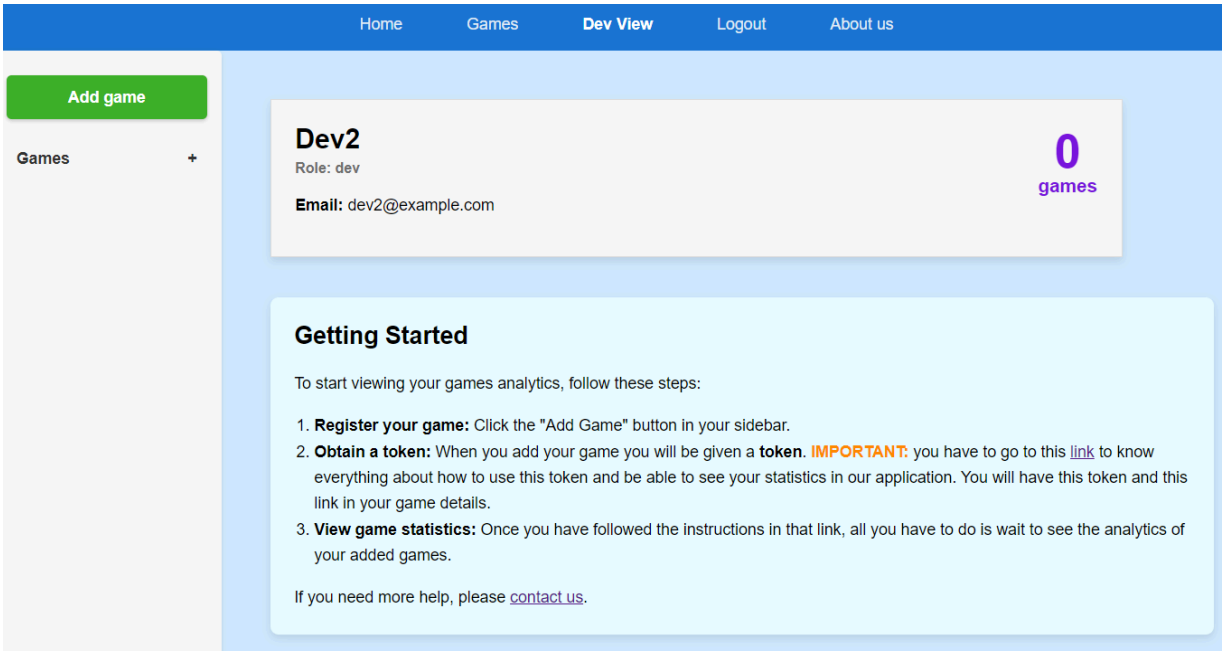

Figura 24. Pantalla principal del desarrollador sin juegos.

Primero deberá hacer click en el botón "*Add game*" del menú lateral que tiene a su izquierda. Deberá introducir el nombre de su juego así como una descripción, siendo esta última opcional (Figura 25-izquierda). Una vez haga click en "*Add*", si todo ha ido bien, se le mostrará un mensaje de confirmación con los datos introducidos junto con un nuevo dato importante, el *token* del juego (Figura 25-derecha). Tiene un link que lleva al usuario al *README*[20] del repositorio de Librería, donde aparecen detalladamente los pasos que debe realizar para poner en marcha el proceso de analíticas de su juego en nuestra aplicación. Este link lo tendrá disponible en los detalles de su juego junto al *token* para poder utilizarlo cuando lo necesite.

Figura 25. Formulario para Añadir juego y su mensaje de éxito(desarrollador).

Una vez añadido el juego, la aplicación nos redirige automáticamente a una pantalla donde se muestran los detalles del juego añadido (Figura 26). Podemos ver el

[20] https://github.com/UCM-FDI-JaXpi/lib/blob/main/README.md
Icono realizado por Octopocto de www.flaticon.es (Marca de verificación verde)

nombre del juego, *token* y la opción de copiarlo al portapapeles, su link de instrucciones, y la descripción que se le puso, o en su defecto, no se muestra. Y el botón "*Delete game*" situado en esta pantalla para poder borrar el juego que desees. Lo hemos colocado aquí para evitar borrar un juego por equivocación al saber exactamente en qué juego te encuentras.

Figura 26. Pantalla de los detalles de un juego sin datos (desarrollador).

Al hacer click en el botón de borrar, aparece un diálogo emergente (Figura 27) preguntándonos si estamos seguros de esta acción, avisándonos de que se perderán los datos definitivamente.

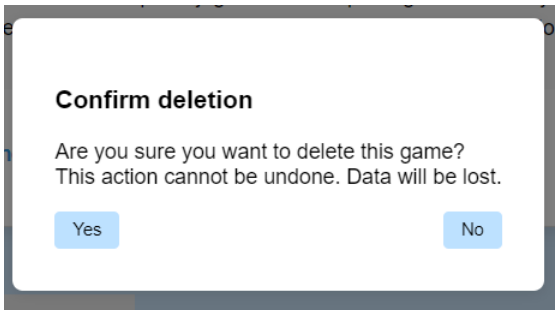

Figura 27. Diálogo emergente de confirmación de borrado del juego.

Junto a los detalles del juego vemos un menú lateral con una sección que ahora ya tiene los juegos que el desarrollador va agregando a la aplicación.

En la esquina superior derecha, tenemos un contenedor donde aparecerán los elementos/objetos más usados de dicho juego. También se ven datos del número de jugadores y más opciones que veremos mejor cuando el juego tenga datos.

En la parte inferior, nos encontramos con el contenedor principal de gráficos, separados por unas pestañas que hemos agregado para tener una mejor organización. Como de momento no hay jugadores que hayan jugado con el juego, se muestra un mensaje informativo de que no hay datos disponibles.

Ahora veamos cuando el desarrollador tiene juegos en su perfil. Su pantalla principal (Figura 28) mostrará los datos personales, el número de juegos registrados en la aplicación y la lista de juegos del desarrollador. Tendremos a mano, en la lista de juegos, el *token* junto a su botón de copiar al portapapeles y el botón de ver detalles del juego para mayor comodidad del usuario. También se puede ir a los detalles de un juego en concreto desde el menú lateral y desde el menú de navegación superior, en "*Games*" seleccionando un juego.

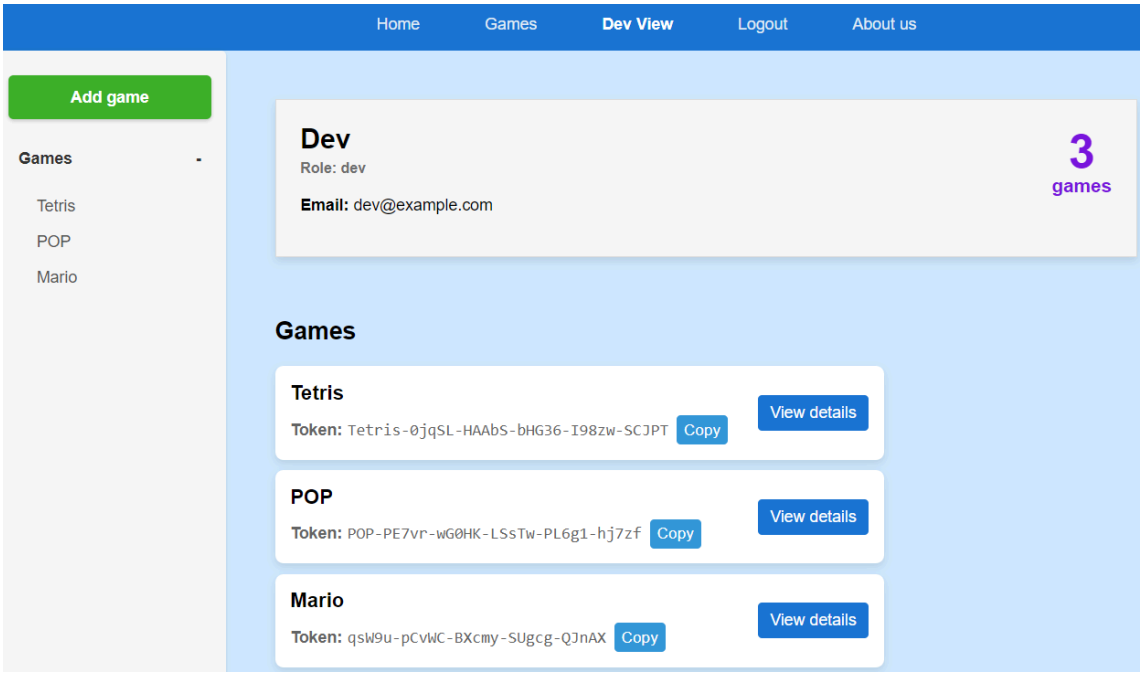

Figura 28. Pantalla principal del desarrollador con juegos añadidos.

Cuando hacemos click en el botón "*View details*" vamos a la pantalla con los detalles del juego que vimos anteriormente pero esta vez, este juego tiene datos y por lo tanto ha sido jugado (Figura 29).

Podemos observar los primeros datos y analíticas que ve un desarrollador cuando su juego ya ha sido jugado. Como vemos, en nuestro ranking de objetos más populares tenemos los tres objetos más utilizados del juego junto al número de veces utilizados. También podemos ver el número de usuarios totales que han jugado registrados, ya que solo calculamos analíticas para los usuarios que han jugado con una clave de juego que un profesor les ha dado. Además, vemos el número de usuarios que han completado el juego llegando al último nivel, así como la fecha y hora de la última interacción con su juego recibida.

El contenedor de gráficos con sus respectivas pestañas lo ampliaremos en la sección 7.1, donde mostraremos y explicaremos los gráficos obtenidos con la demo del juego *Prince of JS*.

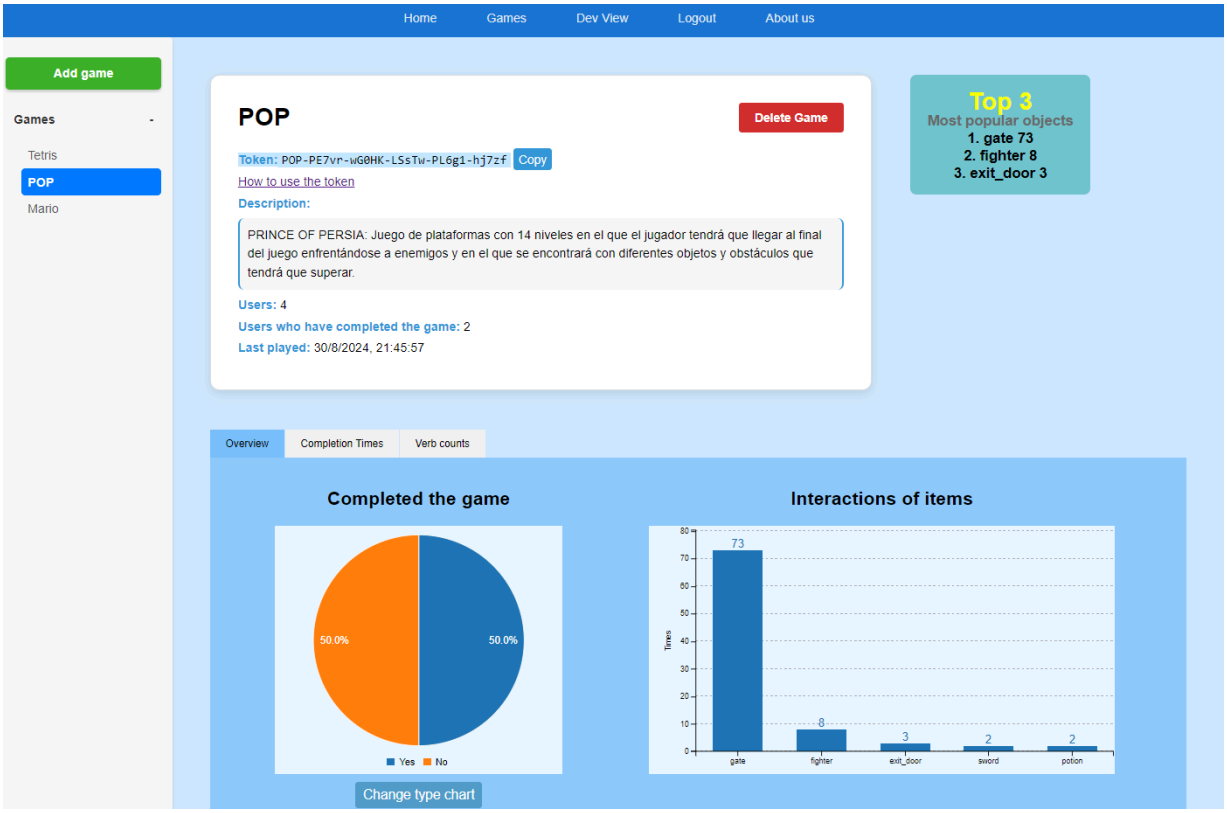

Figura 29. Pantalla de los detalles de un juego con datos (desarrollador).

Continuamos con el tipo de usuario profesor. El uso de nuestra aplicación como profesor, es la más completa y la que más gráficos y estadísticas tiene ya que a este tipo de usuario le interesan los datos de sus alumnos para poder ver si participan en los sesiones de juego o no, viendo las últimas interacciones que hacen con el juego, en cuanto tiempo pueden completarlos, qué niveles les cuesta más… Todas estas estadísticas son de máximo valor para poder ver el comportamiento, destreza y dificultades de sus alumnos con los juegos propuestos.

Al iniciar sesión como profesor por primera vez (Figura 30), se muestran los datos personales del profesor junto a unas instrucciones para mostrarle como empezar a ver analiticas de juego de sus estudiantes. Lo primero que tendrá que hacer es crear una clase. Para ello, tendrá que hacer click en el botón "*Create class*" de su menú lateral.

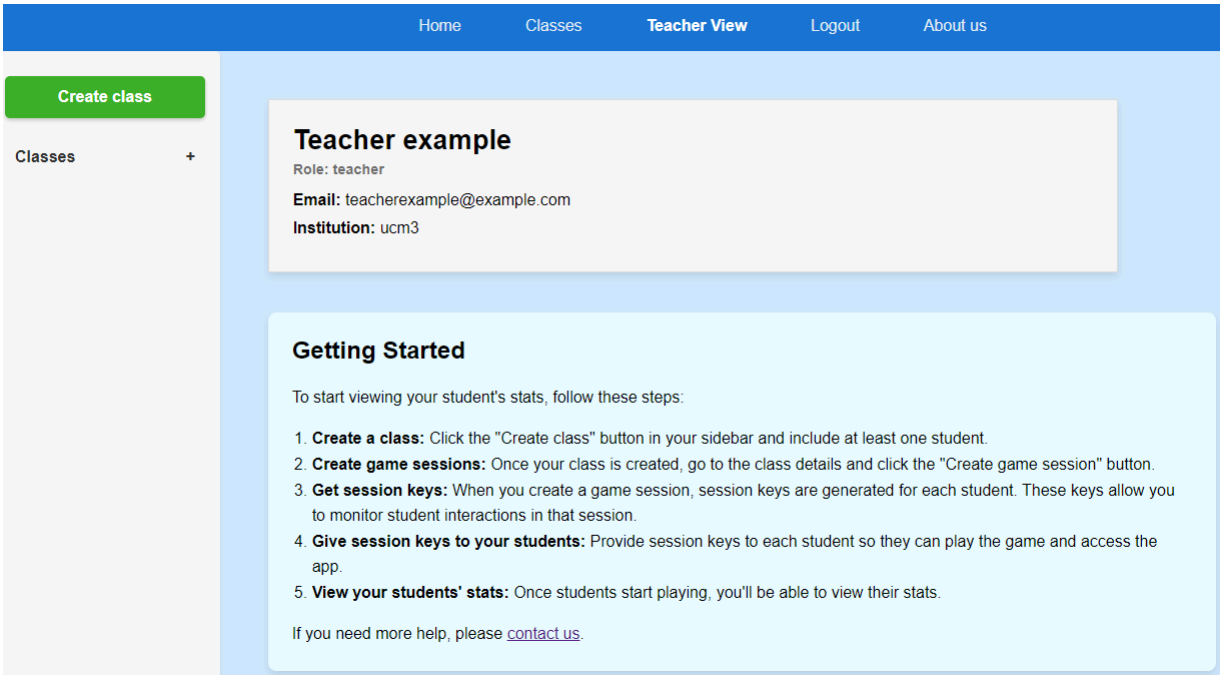

Figura 30. Pantalla principal del profesor sin clases.

Aparecerá un formulario (Figura 31) para la creación de una clase donde tendrás dos opciones para ello, una manual y otra aleatoria. Para la forma manual, deberás introducir los nombres de los estudiantes separados por un salto de línea. Como mínimo uno y como máximo hemos establecido 50. En cuanto a la forma aleatoria de creación, debemos elegir un número de nombres generados como mínimo 1 y como máximo 50, también. Una vez creada la clase con éxito, según la forma elegida, se muestran sus respectivos mensajes de éxito donde se verán los datos más relevantes de la clase junto a la lista de estudiantes con sus nombres.

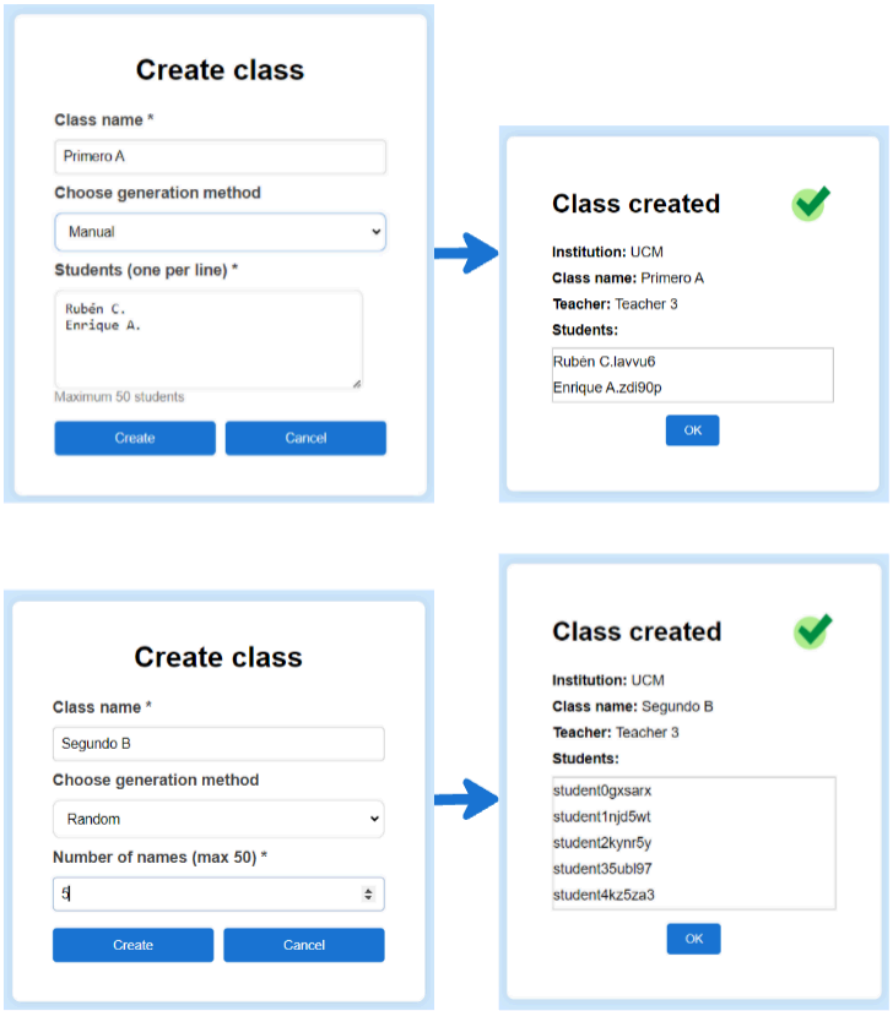

Figura 31. Formularios de creación de clase con sus mensajes de éxito (profesor).

Una vez se tenga la clase creada, se le redirige automáticamente a la pantalla de los detalles de la clase (Figura 32) donde se le podrá crear sesiones de juego, sólo cuando exista una clase. Si el profesor no tuviera clases, no podría tener esta funcionalidad.

También se pueden ir a los detalles de una clase desde el menú lateral y desde el menú de navegación superior, en "*Classes*" seleccionando una clase.

Aquí, se muestran el nombre de la clase y el número total de estudiantes de la clase seleccionada, así como el número de estudiantes activos y no activos, para que el profesor tenga una vista general nada más entrar del número de alumnos que han participado o no, en las sesiones de juego. Debajo, se encuentra el contenedor con las pestañas de sesiones de juego y lista de estudiantes de la clase seleccionada. Si esa clase aún no tiene sesiones de juego, se le mostrará un recordatorio de que necesita crear sesiones de juego para poder ver analíticas de sus estudiantes.

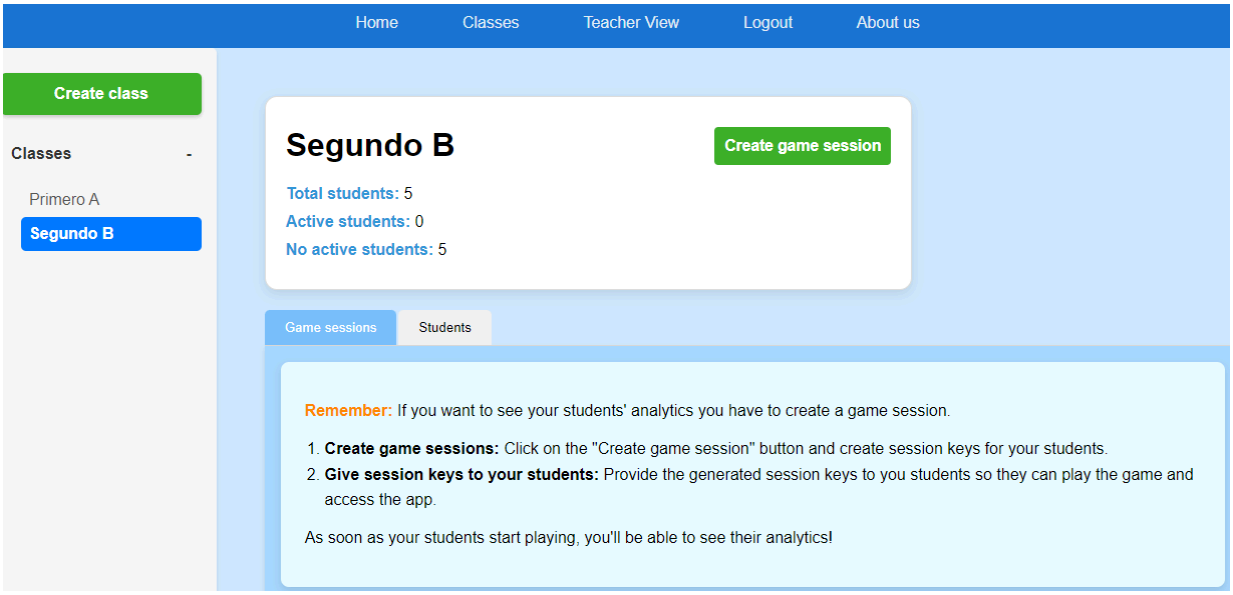

Figura 32. Pantalla de los detalles de una clase sin sesiones de juego (profesor).

Una vez se haya creado una clase, el siguiente paso será crear una sesión de juego. Para ello tendrá que hacer click en el botón "*Create game session*". En el formulario (Figura 33-izquierda), se tendrá que completar el nombre de la sesión de juego y seleccionar un juego de todos los disponibles (campos obligatorios *), ya que el campo *Class* viene rellenado por defecto con la clase desde la que se hizo click más el número de estudiantes que tiene, como información complementaria. En el

mensaje de confirmación (Figura 33-derecha), se muestran los datos de la sesión de juego: nombre, juego elegido, clase y lista de estudiante-clave. Como vemos, también tenemos el botón "*Export CSV*". Esto permite al profesor descargarse la lista de estudiante-key como un archivo CSV, un archivo de texto para representar los datos en formato tabla. Las columnas se separan por comas, y las líneas se separan por saltos de línea.

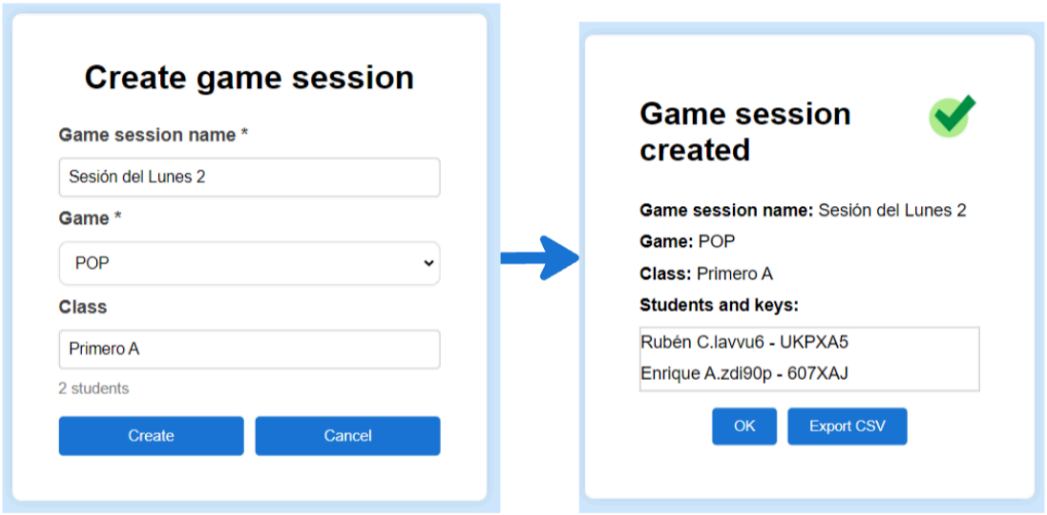

Figura 33. Formulario para crear una sesión de juego de una clase (profesor).

El profesor proporcionará a cada alumno sus claves de acceso generadas para entrar en la aplicación y que puedan ver sus propias analíticas. Hay que recalcar, que un estudiante tiene una clave por cada sesión de juego que el profesor agregue, pero puede acceder a sus analíticas con cualquier clave de sesión que tenga, ya que podrá ver todas sus sesiones de juego cuando inicie sesión. Lo podremos leer líneas más abajo. Como son importantes para el acceso, el profesor, por cada sesión de juego que añada, tendrá la lista de sus estudiantes con sus claves por sesión por si tiene que dársela a algún alumno más de una vez.

Una vez creada la sesión (Figura 34), en la pestaña de "*Game sessions*", aparecerá el listado de todas las sesiones que vaya creando con su nombre, juego y fecha de creación. Haciendo click en "*View statistics*", se navegará a una vista con sus analíticas. En este caso, no hay datos porque aún no han jugado en la sesión. También en la pestaña "*Students*", podemos ver la lista de estudiantes de esa clase para ver la última interacción que realizaron de cualquier sesión. Haciendo click en el icono del ojo, podremos ver más detalles de un estudiante como todas sus claves de sesión.

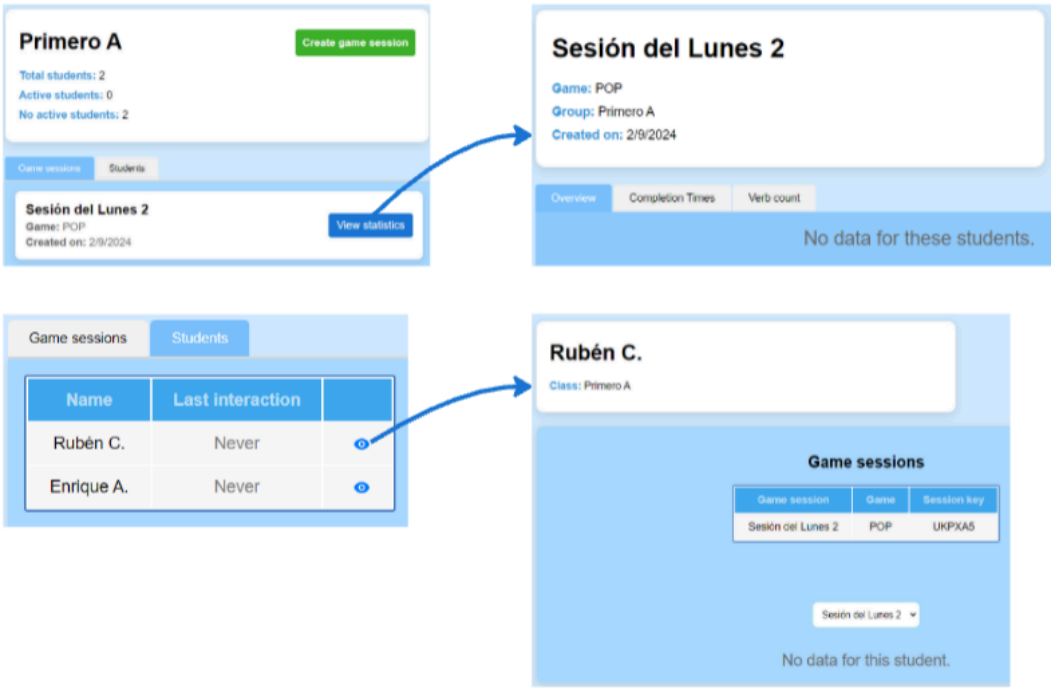

Figura 34. Diferentes pantallas cuando existe una sesión de juego sin datos (profesor).

En la Figura 35, vemos un listado de sesiones dada una clase, y al ir a "*View statistics*" de una sesión de juego con datos, vemos analíticas mediante una tabla, un diagrama de líneas con filtros, etc. Entraremos más en detalle en la sección 7.1. También vemos que en "*Students*" tenemos el formato de *Hace X días…* en la última

interacción realizada por los alumnos. Y por último, en la vista de detalles de un alumno en concreto vemos más analíticas personalizadas de un estudiante.

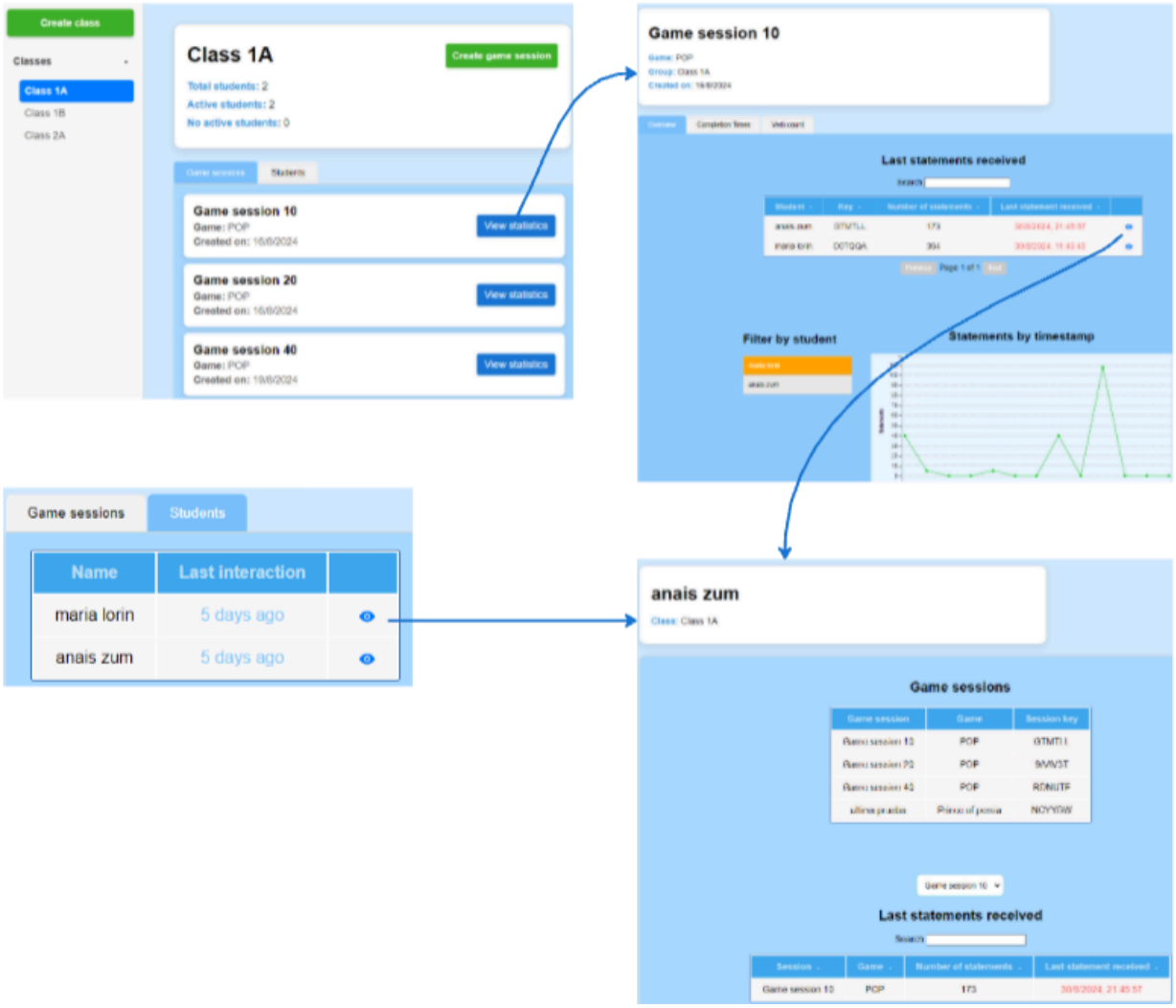

Figura 35. Diferentes pantallas cuando existe una sesión de juego con datos (profesor).

El último tipo de usuario es estudiante. Si hemos accedido como estudiante significa que, tenemos un profesor que nos ha registrado en una clase y ha creado, como mínimo, una sesión de juego, el cual nos ha generado una clave para poder acceder a la aplicación y así poder ver los gráficos de nuestras jugadas. Un estudiante

podrá acceder con esta clave y con otras que le entregue su profesor para ver todas los gráficos de todas las sesiones de juego de su clase con ese profesor. Gracias a una de esas claves, podrá ver todas sus estadísticas. Una forma de acceder que hemos elegido para darle la mayor facilidad a la hora de acceder a sus datos de juego.

Al iniciar sesión podremos ver en su pantalla principal (Figura 36), nombre y rol del estudiante, y contenedor principal con las pestañas "*Overview*" y "*Game sessions*".

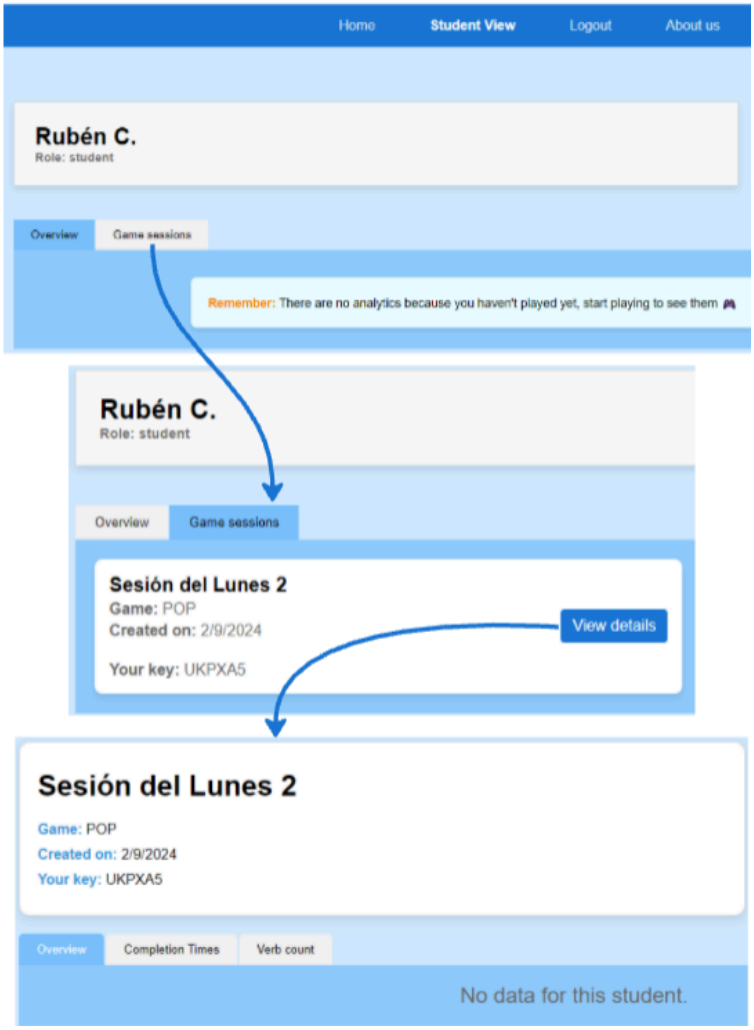

Figura 36. Pantallas de un estudiante que aún no ha jugado.

En la pestaña "*Overview*" (Figura 37) mostraremos una tabla con las últimas trazas que el jugador envió. Y en la pestaña "*Game sessions*" (Figura 38) siempre habrá una sesión como mínimo puesto que el estudiante accedió con la clave de una sesión. Esta lista de sesiones tiene información relevante como el nombre de la sesión, el juego, la fecha de creación y lo más importante, la clave de la sesión que le fue entregada por el profesor. Gracias a esto, tendrá todas sus claves existentes juntas y sabiendo a qué sesión corresponde, por si se pierde alguna.

Vemos que un estudiante no tiene menú lateral, ya que queremos que la aplicación sea lo más sencilla posible para que le resulte fácil de utilizar.

Una vez el estudiante juegue, se mostrarán sus analíticas como en estas figuras, que detallaremos en la sección 7.1:

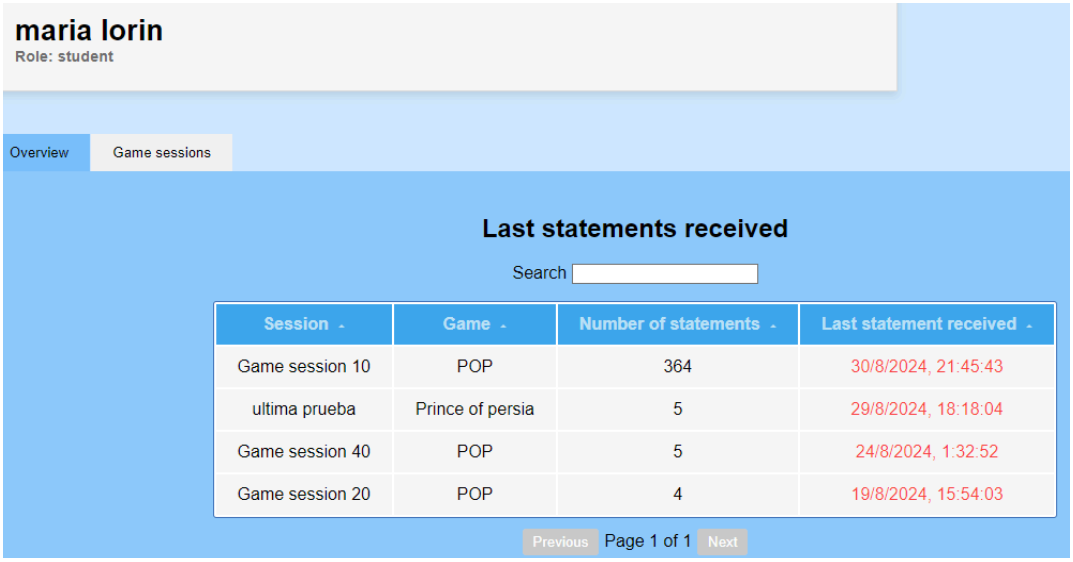

Figura 37. Pantalla de un estudiante con datos I.

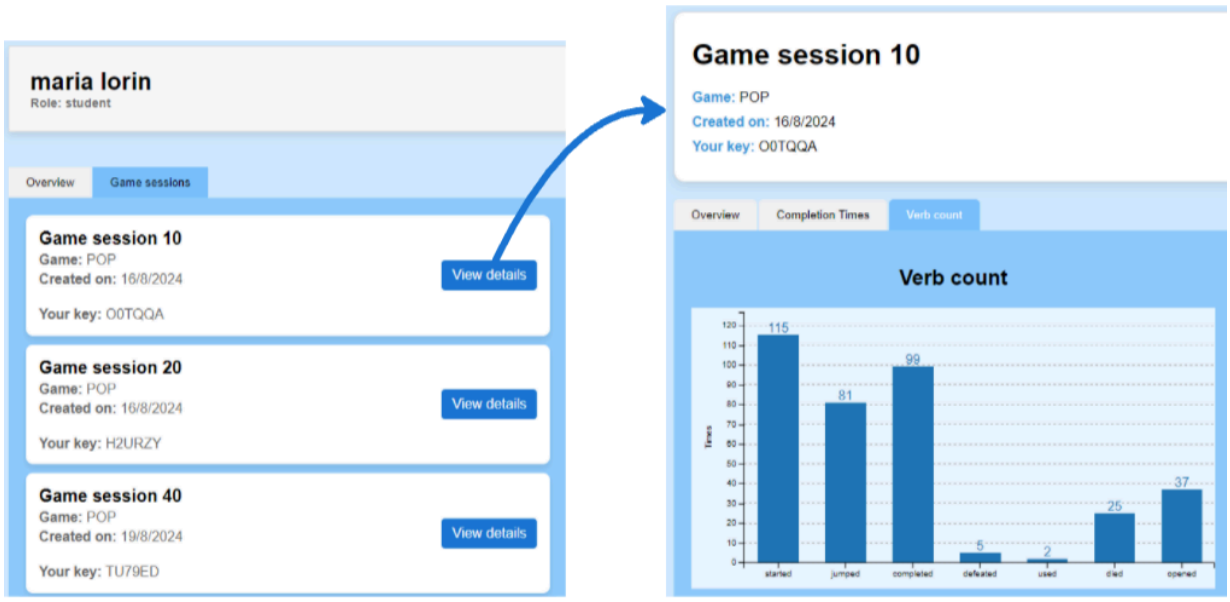

Figura 38. Pantalla de un estudiante con datos II.

Y por último, para salir de la aplicación, los tres tipos de usuario tienen que cerrar la sesión de la misma manera. Tienen que ir al menú de navegación en "*Logout*" donde aparecerá una ventana emergente para que confirmen o cancelen esta acción (Figura 39).

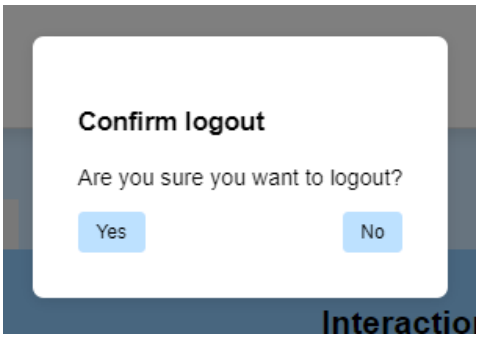

Figura 39. Diálogo emergente de confirmación del cierre de sesión (cualquier tipo de usuario).

7. Demostraciones con juegos

Estas demostraciones se consiguieron utilizando proyectos copyleft colgados en GitHub, en el que se hubiese desarrollado un videojuego de navegador con JavaScript o TypeScript. Con ellos queremos demostrar la flexibilidad y la facilidad de adaptación de la librería a cualquier proyecto al que quiera ser incluida.

7.1 Demo y analíticas de Prince of JS (juego de plataformas)

El proyecto Prince of JS[21] es una recreación del videojuego clásico Prince of Persia desarrollado en JavaScript para su ejecución en navegador.

Este proyecto utiliza un patrón modular, que se empaqueta utilizando Webpack[22] para su utilización en navegadores. Esto permite la instalación de la librería por medio de npm y su utilización con una simple importación en su archivo principal, en el que crea un objeto JaXpi que será llamado donde se quieran encolar trazas. Gracias a esto el número de líneas necesarias para agregar la librería se mantiene muy bajo, haciendo muy fácil su implementación.

Para este caso se ha modificado el archivo index.html en el que se ha incluido un menú simple con en el que se identifique el usuario a través de una clave de sesión, o juegue sin identificarse. Gracias a esto, se pueden generar analíticas específicas para un alumno, que tanto él como su profesor pueden ver, mientras que el desarrollador puede comprobar las estadísticas de todos los jugadores sin discernir personas concretas.

Un ejemplo de las funciones agregadas en el código del juego es '`jaxpi.equipped().item("sword")`' utilizada en el archivo Kid.js cuando el jugador interactúa con una espada con la función '`gotSword() {...}`' propia del juego. Esta

[21] https://github.com/ultrabolido/PrinceJS
[22] https://webpack.js.org/

función encola una traza con los datos del actor, el verbo '*equipped*' y el objeto '*item*' con nombre '*sword*'.

Tras cumplir con el requisito de envío, estas trazas se mandan al servidor, donde, de estar un alumno identificado, se le agregan datos del contexto de la clase, profesor e institución a la que pertenece, mejorando su posterior análisis.

Ahora, mostraremos cómo se procesan y se muestran estas trazas a través de diferentes analíticas en la interfaz de usuario. En la aplicación web, se han desarrollado varios gráficos para representar las métricas que consideramos más relevantes y las que se podían obtener con los datos que nos proporcionaban las trazas del juego (verbos y objetos). Con esto, obtenemos una visión general del proceso de los jugadores en las sesiones de juego, no solo en los resultados finales, ya que las trazas llegan en tiempo real por medio de los sockets entre servidor y aplicación web mientras un jugador está jugando. Por lo tanto, vemos cómo los diferentes gráficos van modificándose en tiempo real, recalculando y mostrando la información actualizada, así como las tablas y datos relevantes que mostramos por pantalla.

Como bien hemos dicho, un desarrollador observa las estadísticas de sus juegos con datos anónimos ya que solo nos fijamos en las acciones de los jugadores, tiempos y objetos que usan del juego. El profesor puede ver las analíticas de todos los estudiantes de sus clases que hayan jugado al juego con sus respectivas claves, y el alumno mira solo sus propias estadísticas ya que no nos interesa que se compare con los demás, sino que se pueda centrar en mejorar su propio rendimiento. Veamos los diferentes formatos que tenemos para este juego:

- **Participación de los jugadores:** podemos medir la participación a través del número de jugadores registrados activos de nuestro juego así como a través del número de jugadores que han podido completar el juego, es decir, que han llegado al último nivel y lo han completado. En *Prince of*

JS, son 14 niveles, lo que significa que de 4 jugadores que han jugado, dos de ellos han completado el último nivel del juego. Así podemos observar el compromiso de los jugadores respecto al juego. Lo hemos representado tanto en texto como en gráficos. Primero, mostramos el gráfico de pastel con los porcentajes, este tiene un botón para cambiar el gráfico en uno de barras apiladas con el número exacto de jugadores que han completado el juego.

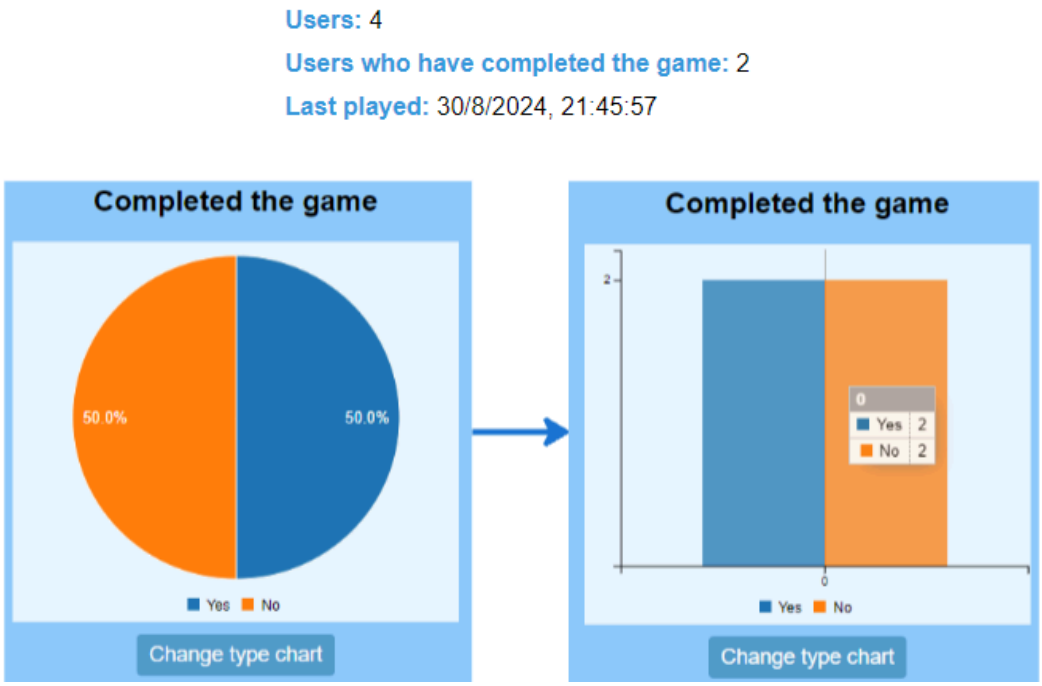

Figura 40. Analíticas sobre la participación de los jugadores (PieChart y StackedBarChart).

Todo lo de la figura 40 lo tiene un desarrollador, pero el profesor también tiene en sus sesiones de juego el gráfico de pastel, así ambos pueden ver el porcentaje de jugadores que han completado el juego.

- **Interacciones de los jugadores:** estas interacciones pueden ser con los diferentes elementos del juego y con las diferentes acciones/verbos que realiza un jugador. Tenemos dos formas diferentes para representar el uso de los objetos (Figura 41), uno es el *Top 3* (sólo desarrollador), y otro, es un gráfico de barras en el que hemos puesto etiquetas con el número exacto de veces que se interactuó con los objetos/verbos, para que se vea el gráfico más claro a primera vista.

  El gráfico de interacciones de elementos lo tienen desarrollador y profesor, pero en dos formatos diferentes. El profesor tiene este gráfico en una sesión de juego, y por ello utilizamos un gráfico de barras apiladas (Figura 42) para ver de un primer vistazo la cantidad de veces que utilizan sus alumnos estos elementos y así, detectar qué dificultades pueden estar teniendo al conseguir algunos objetos. En cambio, el desarrollador, tiene un gráfico de barras simples pues le interesa el uso en general de los objetos, ver si su juego tiene los elementos bien repartidos o si debe hacer cambios en el juego para mejorar.

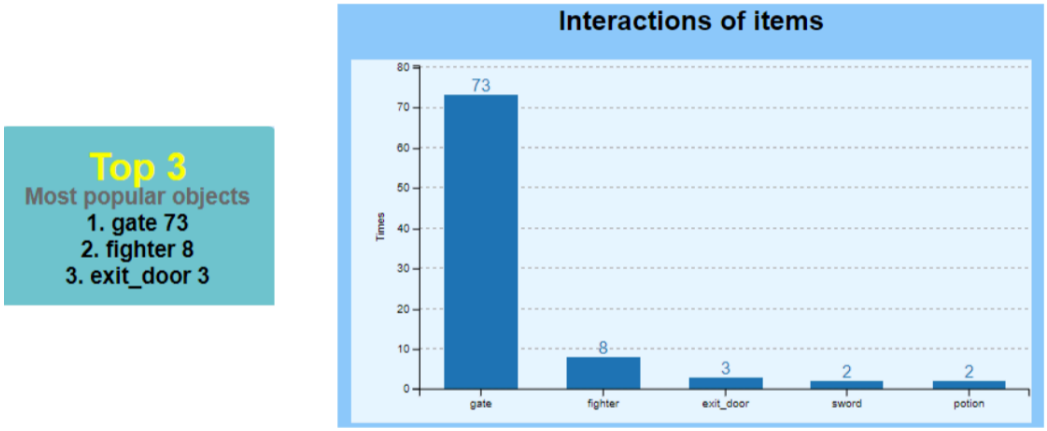

Figura 41. Analíticas específicas de desarrollador (BarChart).

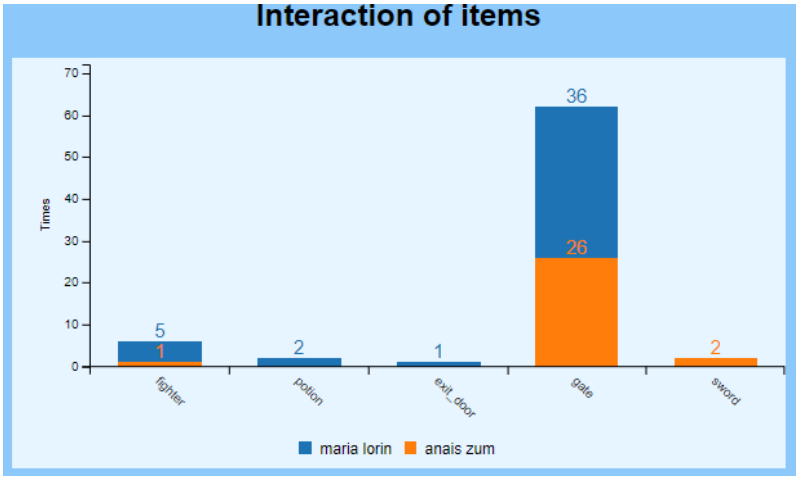

Figura 42. Gráfico de barras apiladas de un profesor (StackedBarChart).

El gráfico de contador de verbos (Figura 43) lo ven los tres tipos de usuario. Los desarrolladores pueden ver si su juego está funcionando como se espera, los profesores, identificar qué dificultades pueden estar teniendo sus estudiantes o si usan el juego de manera correcta mientras que los alumnos pueden reflexionar sobre sus jugadas para poder mejorar su rendimiento.

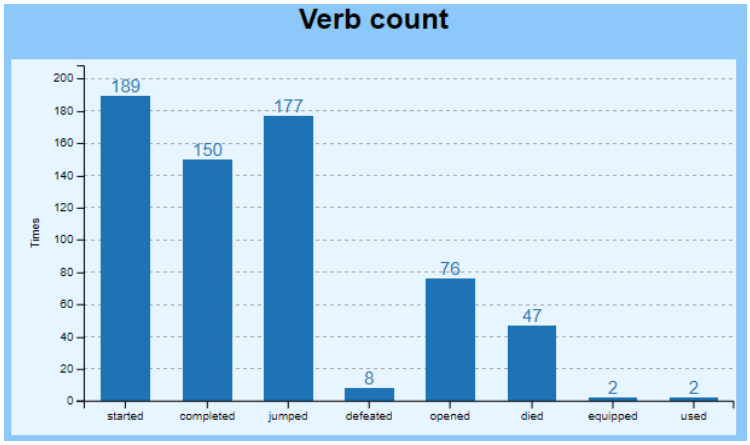

Figura 43. Analíticas sobre las interacciones de los jugadores (BarChart).

Además, tenemos una tabla que nos muestra la actividad reciente de los estudiantes que han jugado. Los que no han jugado no aparecen en esta tabla. Esta tabla (Figura 44), es la que ven los profesores dada una sesión de juego de una clase. Pero también tiene una similar de un estudiante en concreto (Figura 45), así tiene una visión general de su clase como también la de un solo alumno a lo largo de sus diferentes sesiones de juego. Un estudiante también tiene la tabla de la Figura 45.

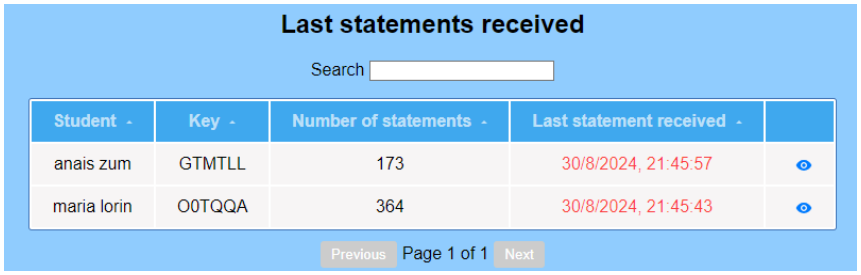

Figura 44. Tabla de registro de participación en una sesión de juego (profesor).

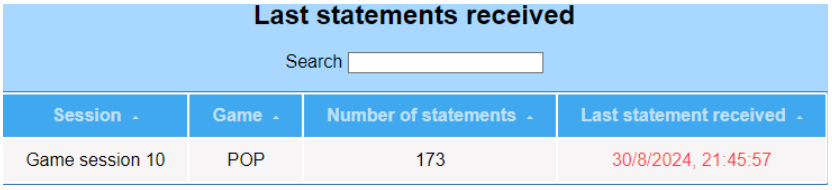

Figura 45. Tabla de registro de participación de un alumno en concreto (profesor y/o estudiante).

Y por último, tenemos este gráfico (Figura 46) que muestra el número de niveles completados por alumno dada una sesión de juego. Este gráfico lo tiene solo el profesor, para ver de un solo vistazo qué niveles son los más jugados o a cuales no suelen llegar los alumnos.

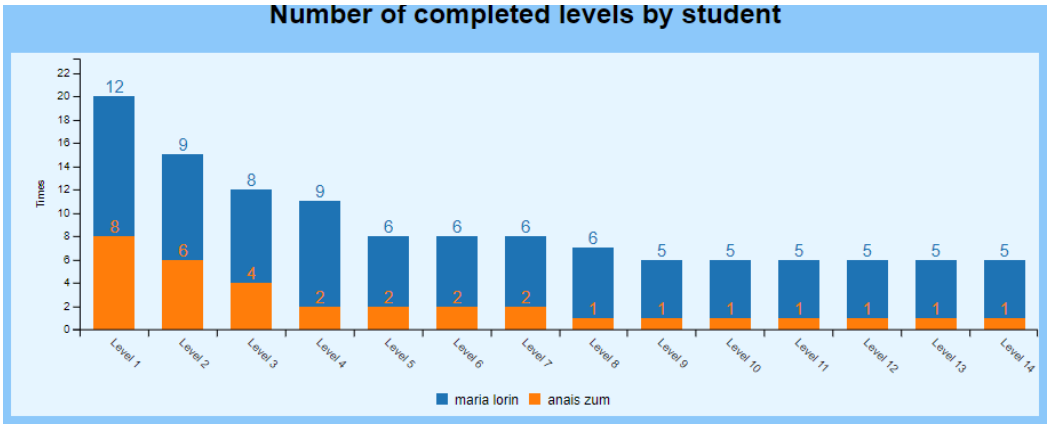

Figura 46. Gráfico de barras del número de niveles completados por estudiante (profesor).

- **Tiempos de los jugadores:** esta métrica es fundamental ya que nos ayuda a ver el nivel de dificultad de los distintos niveles para los jugadores y también el tiempo que se emplea en cada intento por nivel de los estudiantes, pero esto último solo lo tiene un profesor. Usamos gráficos de barras donde representamos en el eje horizontal los niveles con los que han interactuado los jugadores, y en el eje vertical los tiempos (milisegundos) que han tardado en completar cada nivel (Figuras 47 y 48). Y tenemos tanto el gráfico del mejor tiempo completado como el de tiempo medio para más información. Y como dijimos, también tenemos un gráfico de tiempo por intento de un nivel y estudiante seleccionado con un filtro que hemos realizado para mayor comodidad del usuario (Figura 49) ya que al seleccionar un estudiante muestra los niveles jugados de ese estudiante, y si algún jugador no ha jugado a un nivel, este no aparece para poder seleccionarlo.

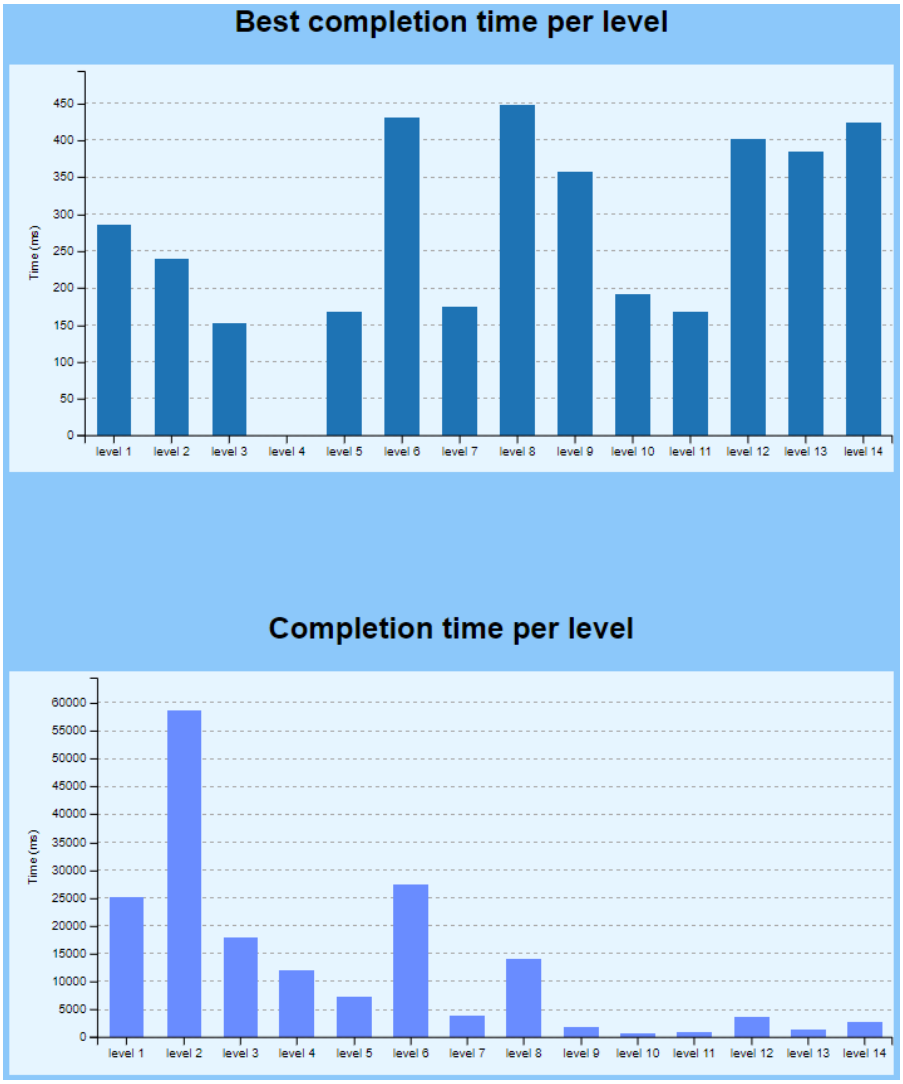

Figura 47. Analíticas sobre el tiempo de los jugadores (BarChart)(desarrollador).

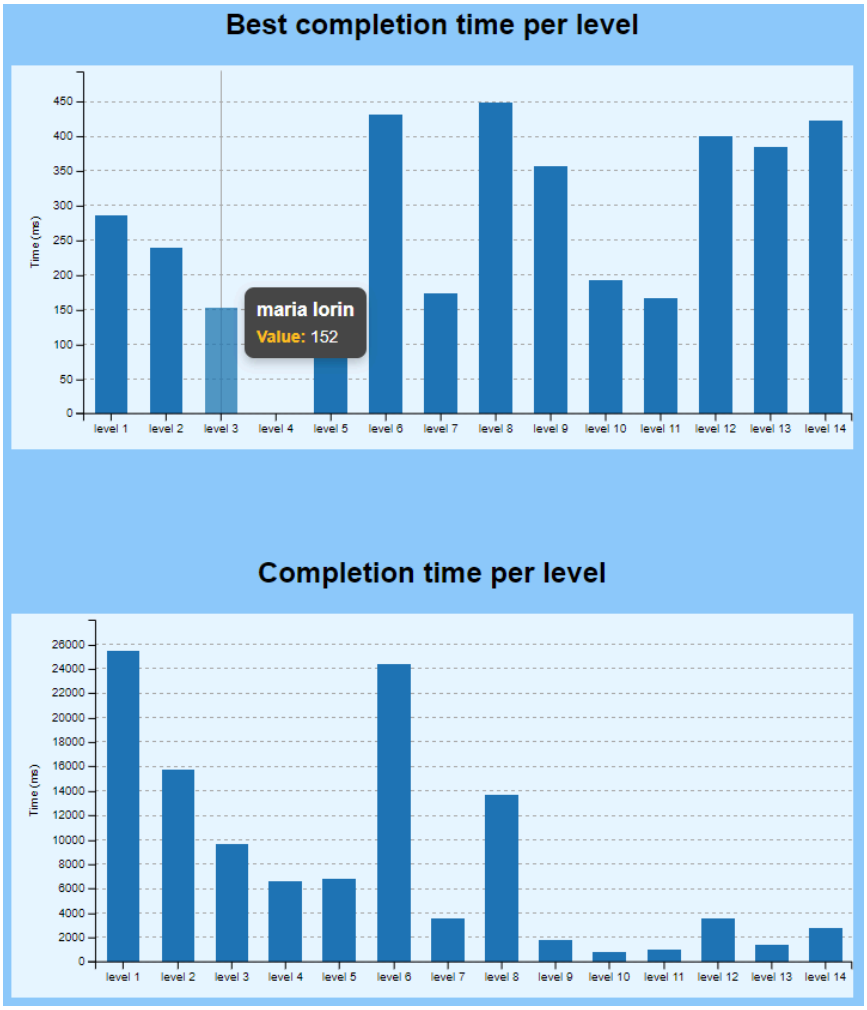

Figura 48. Analíticas sobre el tiempo de los jugadores (BarChart)(profesor).

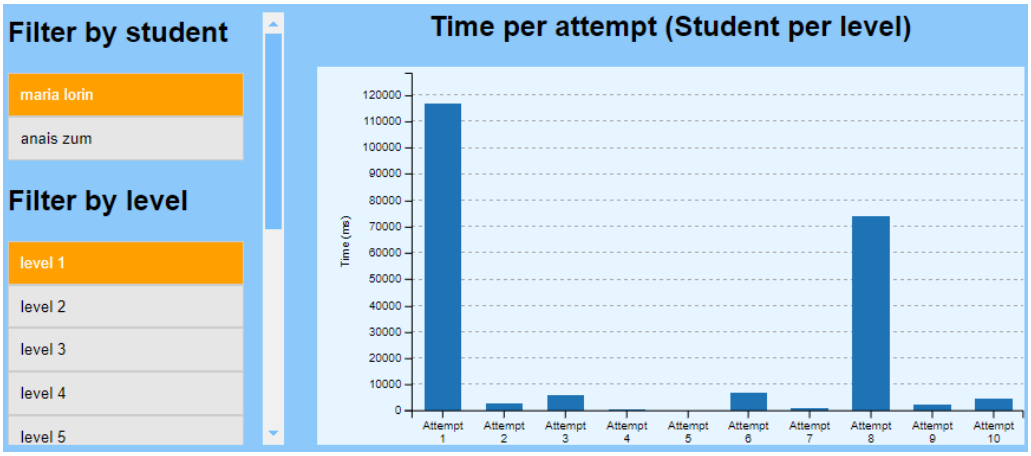

Figura 49. Analíticas sobre el tiempo de los jugadores (BarChart con filtros a su izq.)(profesor).

- **Línea de tiempo:** para un profesor ver en un gráfico de líneas un registro de las fechas en las que sus jugadores han participado, puede ser también útil para ver si sus estudiantes han participado y cuánto han interactuado con el juego a lo largo del tiempo. Por ello este gráfico lo tienen solo los profesores con un filtro a su izquierda y poder cambiar de alumno en este mismo sitio (Figura 50).

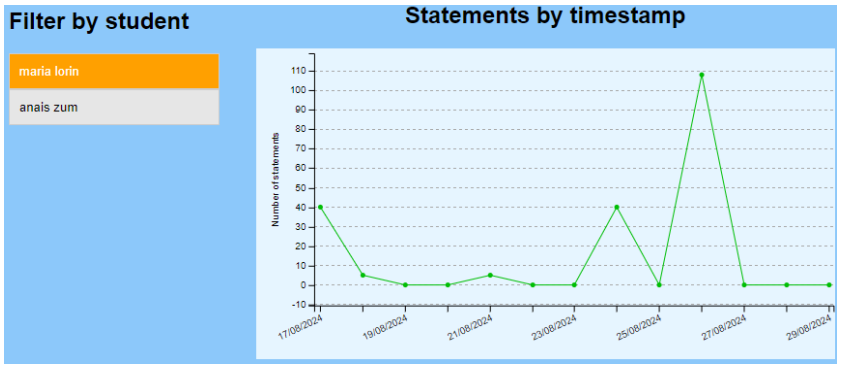

Figura 50. Línea de tiempo (LineChart con filtros a su izq. profesor).

7.2 Demo Dwarfs2019 (juego de aventura gráfica)

Dwarfs2019[23] es un videojuego de navegador desarrollado en TypeScript en ámbito global. El juego es del estilo '*clicker*[24], en el que el usuario interactúa con botones en su pantalla repetidamente.

Para la inclusión de JaXpi en este proyecto se probaron dos métodos distintos, intentar convertir el proyecto utilizando módulos que permitieran la importación de la librería que posteriormente se empaquetaron con webpack, y agregar un solo archivo con el código de la librería que no necesitase dicha empaquetación.

Con el primer método se produjeron numerosos fallos de dependencias y se tuvo que modificar en gran medida el código original del juego, si Dwarfs2019 hubiese sido un proyecto más complejo este enfoque sería completamente inviable.

El segundo fue mucho más manejable, necesitando solo incorporar el archivo con el resto y simplemente creando un objeto Jaxpi que se fue usando donde se necesitaba.

Dada la simplicidad del proyecto este apenas podía incluir trazas muy simples de interacción con los botones como '`jaxpi.clicked().dialog("Draft Another Dwarf")`', que dan lugar a analíticas muy pobres.

7.3 Demo Back Attacker (juego de puzzles)

El último de los juegos probados es Back Attacker[25], en el que tienes que acabar con todos los guardias en el nivel, actuando como puzzles en cada uno de ellos.

Este juego vuelve a utilizar un diseño modular con la particularidad de usar unas versiones desactualizadas de las dependencias requeridas para JaXpi. Sin embargo,

[23] https://github.com/mvasilkov/dwarfs2019
[24] https://es.wikipedia.org/wiki/Videojuego_incremental
[25] https://github.com/yeonjuan/js13k-2019-back

no supuso mucho problema, ya que tras actualizarlas e incorporar la librería al código con su importación, esta se pudo utilizar sin problemas, de una forma idéntica al proyecto de Prince of JS.

Con esta demo nuestra intención es demostrar la facilidad de adaptación de la librería a los juegos a los que se quiera agregar.

8. Contribuciones

8.1 Sergio José Gómez Cortés

Comenzamos este proyecto con una reunión con los directores del trabajo Antonio y Manuel, que nos explicaron de qué se trataba y qué herramientas podíamos utilizar para empezar a trabajar sobre el proyecto, además de presentar a los miembros del equipo.

Tras esto nuestro primer objetivo fue entender el estándar xAPI que íbamos a utilizar para representar la información de la actividad de los usuarios. Para ello se nos encomendó crear un pequeño manual de xAPI que luego podríamos usar para la memoria. También empezamos a pensar sobre qué trazas podría generar un juego que fueran útiles para un posterior análisis. Conseguimos crear un documento con un número respetable de verbos posible para cada uno de los tres perfiles de juegos que decidimos abordar.

Lo siguiente fue crear un prototipo de cliente el cual pudiese enviar una traza y un servidor que lo mostrara por pantalla. Para este fin creamos una carpeta con trazas prefabricadas que pensábamos utilizar para el envío.

El código de este prototipo lo subimos a GitHub, creando de paso tres repositorios, uno para la librería, otro para servidor y un final para las analíticas. Todas las reuniones que fuimos teniendo con los objetivos marcados para la siguiente reunión fueron organizadas en Drive.

Con una idea de la libreria mas elaborada, seguimos elaborando mas las trazas utilizadas para los juegos, además empezamos a utilizar un LRS[26] en el que validarlas cerciorarnos que podrían utilizarse sin problema alguno al ser enviadas a cualquier servidor y no solo al que nosotros creamos.

[26] http://adlnet.github.io/xapi-lab/#

A esta altura decidimos como grupo encargarnos de una parte del proyecto cada uno, yo me encargaría de desarrollar la librería, Marcos de desarrollar el servidor y Miriam de elaborar el diseño y las analíticas de la web, aunque nos apoyamos en nuestros compañeros a lo largo de todo el trabajo.

Dedicándome a la librería decidí migrar el proyecto a TypeScript, lo que me ayudó a desarrollar un código más robusto, mientras que mis compañeros decidieron permanecer con JavaScript como lenguaje. También empecé a ver juegos que pudiéramos utilizar como base para las pruebas de la librería. Elegimos Prince of JS tras la recomendación de nuestros directores, ya que presentaba todos los puntos que buscábamos, que estuviese desarrollado de forma modular y en JavaScript para su ejecución en navegadores, además utilizaba las versiones más recientes de Webpack y otras librerías utilizadas para su desarrollo, con lo que podríamos evitar problemas de dependencias.

Después agregué la posibilidad de elegir a qué servidor LRS querías enviar las trazas generadas por el usuario. Tras varios diseños acabé decidiendo por tener un parámetro en el constructor de la clase en el que se pudiera establecer la URL del servidor. Además agregó que el actor que creaba estas trazas también se estableciera como un parámetro de tipo '`interface Player {name: string; mail: string;}`' para que el desarrollador pudiera enlazar fácilmente con una cuenta de usuario.

Justo antes de navidad se presentó el problema de las trazas estáticas, que o bien no permitían que un mismo verbo aceptase varios objetos o el número de trazas necesarias grecia exponencialmente, además se empezó el diseño de las funciones verbo en la librería, con la que llamaríamos a una función específica cuando el desarrollador quisiera encolar una traza de actividad, en lugar de tener una función general a la que se le pasara una traza como parámetro. Para solucionar esto se planteó la idea de utilizar código generativo utilizando las trazas ya creadas en lugar de desarrollar a mano cada una de las funciones, dejando una función especial para

trazas no contempladas.

De todas formas tuvimos que hacer un parón debido a los exámenes de mis compañeros, por lo que aparcamos el TFG hasta haberlos terminado.

Con los exámenes acabados y la librería implementando el código generativo volvimos a la carga con la idea de acabar el trabajo para verano. Para lo cual fui retocando la librería y corrigiendo bugs, además de modificar la forma de conectar con el servidor a petición de mi compañero Marcos, que estaba realizando cambios profundos en este área.

Nos dimos cuenta de que al enviar generar y enviar las trazas podríamos provocar una bajada del rendimiento del juego, para solventar este problema utilice una cola dinámica en la que almacenar un número arbitrario de trazas antes de su envío, implemente Web Workers que se encargaran del envío de las trazas por hilos paralelos al principal y programe diferentes protocolos de envío configurables por el desarrollador del videojuego.

Finalmente agregue funcionalidades a la librería para gestionar errores de envío al servidor, agregamos los token de conexión con la clave de la sesión, además de agregar un menú a los juegos para poder verificar estas claves, agregue documentación autogenerada a las funciones anidadas y finalice la demo con el juego de Prince of JS para que mis compañeros pudieran utilizarla para sus pruebas.

Cuanto más se acercaba el verano, más nos dábamos cuenta que no íbamos a poder terminar el TFG antes de tener que parar para estudiar para los exámenes finales, por lo que decidimos como grupo retrasar la entrega a la convocatoria de septiembre y mejorar el trabajo.

Durante el verano pude agregar dos demos más con juegos que cumplian los perfiles propuestos al principio del proyecto, además de agregar un archivo

descargable compatible con proyectos de ámbito global, además de terminar de pulir la generación de código.

8.2 Marcos Colombás García

El proyecto comenzó con una reunión con los tutores, en la que se nos explicó a grandes rasgos la idea central del proyecto y las diferentes partes que lo componían. A partir de esa reunión, decidimos dividir las tres secciones del proyecto y que cada integrante se responsabilizara de una, aunque siempre manteniendo una colaboración constante. Opté por encargarme del servidor, dado mi interés en aprender más sobre el desarrollo backend.

Iniciamos con una pequeña investigación, tanto sobre las tecnologías a utilizar como sobre el estándar xAPI. A petición de los tutores, redacté un documento que contenía los objetivos principales del proyecto, y junto a mis compañeros, preparamos otro documento con el estado de la cuestión sobre xAPI y los Learning Record Stores (LRS). Además, colaboré en la elaboración de un anexo que proponía un conjunto de verbos y objetos predeterminados para la librería, enfocados en los tipos de juego que habíamos decidido abordar: plataformas, puzzles y aventuras gráficas.

Tras esta fase, comencé con el desarrollo del servidor. Decidí utilizar JavaScript por su simplicidad y por ser adecuado para manejar datos en formato JSON, que era el formato en el que se representan las trazas xAPI. El primer objetivo fue montar un servidor básico capaz de recibir una traza mediante una petición POST y almacenarla en el sistema local. También creé un pequeño frontal utilizando EJS para visualizar esas trazas.

A partir de ese punto, el código fue evolucionando gradualmente. Agregué la funcionalidad de almacenar las trazas en una base de datos, optando por MongoDB,

ya que era la herramienta más utilizada para servidores desarrollados con node.js y express, además de que su formato JSON facilitaba la manipulación de las trazas xAPI. Luego implementé un sistema de autenticación utilizando Passport, permitiendo el registro y autenticación de usuarios. A continuación, desarrollé rutas que permitían realizar operaciones CRUD sobre los usuarios y las trazas en la base de datos, siendo especialmente importante asegurar que los desarrolladores solo accedieran a los datos permitidos, evitando la exposición de información personal.

Paralelamente, colaboré con Sergio en la implementación de la librería, y en la integración de esta con el servidor para facilitar el envío de datos. Además, configuré aspectos técnicos del servidor como CORS y, más adelante, la configuración de WebSockets para enviar trazas en tiempo real.

Cuando Miriam comenzó con la web de análisis, trabajamos en dos frentes: por un lado, integré mi servidor con la aplicación web, adaptando el código HTML que había usado inicialmente para login y registro, y por otro lado, configuré Socket.io para permitir la actualización en tiempo real de los gráficos en la web. Además, añadí vistas html de login y registro a la web reutilizando parte de mi antiguo código ejs. También realicé pruebas del servidor y generé credenciales y datos de prueba para verificar el correcto funcionamiento de las gráficas.

Posteriormente, empecé a experimentar con diferentes formas de autenticación para el envío de trazas, usando inicialmente JSON Web Tokens, que luego fueron reemplazados por el sistema de GameTokens, generados por los desarrolladores e integrados en la librería. En este punto, tuvimos una reunión clave con los tutores para definir los flujos finales, especialmente cómo identificar trazas con un jugador específico mediante las "session keys", que permitían asociar trazas a un alumno, su clase, grupo y sesión de juego.

Implementé estos cambios ampliando la base de datos con colecciones que representaban grupos de alumnos, sesiones de juego y juegos creados por desarrolladores. También desarrollé rutas que permitían a los profesores crear grupos y sesiones de juego, y a los desarrolladores generar y gestionar sus GameTokens. Además, realicé importantes ajustes en cómo se recibían y enviaban datos entre la librería y la web de análisis, colaborando estrechamente con Miriam para cumplir con sus requisitos.

Durante todo el desarrollo me encargué de la resolución de bugs y errores que surgieron en la integración de las diferentes herramientas. Finalmente, me encargue de realizar junto a mis compañeros la memoria de este trabajo, haciéndome cargo de la parte asociada al servidor, (punto 5) y de la redacción y elaboración de diagramas del punto 3. También redacté junto con mis compañeros el resto de secciones comunes de la memoria, con especial dedicación al plan de trabajo y al estado de la cuestión asociado con el estándar xAPI.

8.3 Miriam Elizabeth Cabana Ramírez

A la hora de decidir qué TFG realizar, revisé el listado de las opciones disponibles, y recuerdo que días posteriores me comentó mi compañero Sergio que también estaba buscando un tema. Él mostró interés en el ámbito de los videojuegos, mientras que yo me decantaba más por el desarrollo de alguna aplicación web o móvil, área de la que quiero aprender más. Al considerar nuestras preferencias, vimos este tema de TFG que podría combinar ambos intereses.

Cuando nos reunimos con los tutores y Marcos, comenzamos con ello. Organizamos cada dos o tres semanas reuniones a pesar de tener horarios muy diferentes. Yo me encontraba realizando prácticas en empresa, y con una situación familiar delicada debido a un problema de salud grave de mi madre. Se lo comuniqué

a mi equipo ya que desde Noviembre me fue muy complicado estar todo lo que me gustaría con el TFG y hasta Abril más o menos, la situación fue delicada.

Para comenzar, los tutores nos habían contado la idea que tenían y que se podría dividir en tres bloques: librería, servidor y visualización de datos, las cuales decidimos repartir. Me tocó la parte de la visualización de datos y nombramos el uso de *Angular* para el desarrollo de la web, ya que lo estaba aprendiendo y pensé en ampliar mis conocimientos sobre ello. Creamos los repositorios en *Github* junto a los tutores para poder verlos en las reuniones que teníamos. Además, generamos un documento en *Google Drive* de cada reunión, para no dejarnos ningún detalle.

Comenzamos con la investigación del estándar *xAPI* y a montar un pequeño prototipo de cliente de envío de trazas, al igual que el servidor del que se encargó Marcos. Decidimos empezar con estas partes y empezar más adelante con análisis.

En los primeros meses, nos conectábamos Sergio y yo para ver acerca de la librería y el estándar *xAPI*. A la par, planteamos diferentes géneros de juegos para poder usar las trazas con verbos y objetos que enviaríamos al servidor y que luego se guardarían en un base de datos. Luego, hicimos un parón en el mes de Enero por los exámenes que teníamos. Pero ya habíamos hablado de ver cómo compilar el juego principal que hemos utilizado, *Prince of JS*.

A la vuelta, me puse a investigar y probar librerías *JavaScript* para pintar gráficos (*D3.js* y *C3.js*). Investigué sobre qué métricas eran relevantes en el análisis de juegos, qué tipo de visualizaciones se utilizan para ello y qué pasos seguir para realizar un análisis de datos. Ví varias conferencias sobre la analítica de aprendizaje y los videojuegos como herramienta educativa. Además, decidimos usar Vue ya que lo nombramos alguna vez y quise aprenderlo, era la oportunidad perfecta para

desarrollar un proyecto con un nuevo framework. Asique seguí cursos como los de Vue School[27], tutoriales, documentación, etc, mientras comenzaba con la aplicación.

Tuve que pintar en papel los primeros diseños y comentarlos con mi equipo. Mientras tanto, con la ayuda de Sergio, íbamos planteando escenarios de jugadas con las trazas *JSON* y fuimos realizando lógica para calcular algunas analíticas como puntuación por nivel, tiempos completados por nivel, etc. Que luego tuve que modificar porque cambió la lógica empleada.

Para los gráficos utilicé la librería *C3.js*. Se podía personalizar bien y tenía el tipo de gráficos que tenía en mente para los datos del juego. Comencé pintando las jugadas de prueba con verbos y objetos. Pudimos comprobar que el campo *timestamp* de la traza reflejaba el momento exacto en el que el jugador realizaba las acciones y nos guiamos en la lógica por este campo.

Vi lo útil que es el campo *timestamp* a la hora de realizar las analíticas y gracias a ello, realicé la lógica y el procesamiento de los datos siguiendo este orden. Para realizar las analíticas del juego elegido, tuve que mantener contacto continuo con Sergio, quien se ocupó de introducir los verbos y objetos que yo iba a leer en las trazas de juego. Además, jugué mucho al juego, ya que presentaba diferentes erratas que lo convirtieron en un reto y debía generar diferentes escenarios para tener datos que poder analizar y plasmar en los gráficos.

Una vez tenía diferentes jugadas, fui creando las vistas que dibujamos en papel, y para que todos pudiéramos aportar, busqué como tener una pizarra grupal a tiempo real en el que poder participar todos y encontré *Miro*. Allí pudimos comentar ideas de los bocetos y por ende, funcionalidades de la aplicación, añadí modelos de aplicaciones, diseño y colores que encontraba, y pinté en tablas nuestra base de datos. Un paso necesario para mí y que quería dejar para consulta, ya que realizaba

[27] https://vueschool.io/

peticiones *HTTP* cuando necesitaba datos o cuando recibía datos a tiempo real desde el servidor mediante sockets y era fundamental conocer los campos que tenía cada documento de cada colección.

Tuve que comprender y añadir funciones y parámetros en el servidor ya que tenía que mandar a la aplicación todo lo necesario para realizar las operaciones que iba realizando a medida que iba pintando gráficos de un determinado tipo de usuario, en una vista o desde una pestaña en concreto. Gracias a la necesidad de campos que aparecía, pude comprender mucho mejor el servidor y en cómo se comunicaba con el juego y con la aplicación. Sin duda, ha sido un aprendizaje de mucho valor para mí ya que con la vista global de cómo funciona nuestro TFG, pude ver la conexión y los diferentes flujos que existían.

La lógica para mostrar los gráficos también fue un reto para mi. Vi todo lo que se podía llegar a calcular con los campos de una traza JSON y también de la importancia de la privacidad de los datos. Además de investigar sobre los *sockets*[28] para actualizar los gráficos en tiempo real en *Vue 3*.

Por último, me encargué de hacer la memoria junto a mis compañeros, distribuyendo los diferentes puntos de la manera más equitativa posible. Para análisis, fui escribiendo lo que iba aprendiendo de *Vue* a la largo de los meses para que cuando llegara la parte de redactarlo tener cosas ya escritas sobre ello.

Sin duda ha sido todo un reto académico y personal, del que he sacado un gran aprendizaje en cuanto a conocimientos y todo lo que conlleva un trabajo en equipo. Sobre todo, la importancia de resolver los inconvenientes de la mejor manera. La comunicación es fundamental para que un proyecto salga adelante. Solo me queda dar las gracias a mi familia, en especial a mi madre, que nunca se rinde a pesar de todas las adversidades que vengan. Ella es mi mejor ejemplo.

[28] https://socket.io/how-to/use-with-vue

9. Conclusiones y trabajo futuro

En este proyecto, hemos logrado desarrollar el ecosistema JaXpi, formado por una librería en TypeScript usando el estándar xAPI, un servidor intermediario que gestiona los datos y una aplicación web que muestra analíticas de datos a través de gráficos y estadísticas. Hemos conseguido hacer funcionar estas tres herramientas en conjunto para lograr estudiar datos generados por videojuegos web mediante analíticas.

La librería se puede importar en diferentes juegos JavaScript para generar trazas, almacenarlas y después analizarlas. De esta manera, la librería es fácil de adaptar y de extender, pudiendo responder a las necesidades de los desarrolladores.

Al estar basada en xAPI, ampliamente utilizado en el ámbito del e-learning, la librería asegura que las trazas enviadas sean fáciles de recoger y de analizar por cualquier servidor LRS, permitiendo la integración de la librería con otras herramientas educativas que utilicen el estándar.

Por otro lado, el servidor y la web de análisis trabajan en conjunto para recoger esos datos y almacenarlos de una manera segura y eficiente, para luego transformar esos datos en gráficos e información que pueda ser fácilmente entendible por los usuarios. Al realizar las analíticas del juego *Prince of JS*, hemos podido ver el potencial de recoger y analizar esos datos para enriquecer el proceso de aprendizaje de los estudiantes al jugar a videojuegos educativos.

Durante el desarrollo del proyecto hemos aprendido sobre la aplicación del estándar de xAPI a videojuegos con un fin educativo, además de conocer las distintas herramientas que ya existen con el mismo fin. Además, hemos aprendido sobre el ciclo de desarrollo de software, en concreto sobre cómo desarrollar herramientas web que trabajen en conjunto para un objetivo común.

En conclusión, JaXpi beneficia a desarrolladores al detectar fallos o carencias en sus juegos, a docentes al permitir el seguimiento en tiempo real del desempeño de sus alumnos en las sesiones de juegos y a revisarlos posteriormente, y a los estudiantes al ver sus propias estadísticas y saber dónde pueden mejorar. Usada en conjunto con otras herramientas de aprendizaje, creemos que puede proporcionar un aprendizaje mucho más efectivo gracias al seguimiento y análisis tan en detalle que se puede hacer de la actividad de los estudiantes.

9.1 Trabajo futuro

A continuación, nombraremos algunos funcionalidades o mejoras que podríamos realizar en un futuro:

- Ampliar aún más el catálogo de verbos y objetos que la librería comprende, además de añadir herramientas de testing para las pruebas externas, facilitando todavía más la integración de la librería.
- Agregar en la librería código de testeo para envío de una traza/s de ejemplo a un LRS como ayuda a debugear al desarrollador.
- Añadir una funcionalidad a la librería que mande los datos a la espera de envío al LRS en caso de que se cierre la ventana donde se ejecute el juego.
- Configurar CORS para que sea más permisivo, permitiendo a los desarrolladores ejecutar su juego en puertos distintos.
- Desarrollar una página de administración desde la que un usuario administrador pueda realizar operaciones CRUD para gestionar la base de datos
- Poner el servidor en un entorno de producción, migrando a una infraestructura más robusta e implementando HTTPS para comunicaciones más seguras, además de añadiendo herramientas que permitan la monitorización para detectar errores o ataques.

- Probar Jaxpi con más videojuegos JavaScript y especialmente, con videojuegos educativos ya que este trabajo va enfocado en poder desarrollar herramientas que nos permitan realizar analiticas de aprendizaje en el área educativa.
- Crear perfiles de análisis por género de juego. Actualmente, realizamos analíticas a un tipo de juego específico (plataformas). Pero sin duda, el poder adaptar nuestros análisis a los diferentes géneros de juego (aventura gráfica, puzzles, carreras…) es una funcionalidad ideal ya que podríamos diferenciar el género del juego así como sus mecánicas, ya que cada uno maneja unas mecánicas en concreto y los datos no se interpretan de la misma forma. De este modo, podríamos tener un gran abanico de juegos con datos que analizar y de una forma correcta.
- Crear perfiles de análisis según los verbos *xAPI* que utilicen los juegos. Utilizamos principalmente los verbos *started* y *completed* para calcular los tiempos de compleción del juego o de los niveles, por ejemplo. Esto funciona con los juegos que tengan niveles como Prince of JS, pero los que no, pueden tener otros verbos más importantes que marquen eventos del juego. Cada verbo nos da un tipo de información diferente de los jugadores.
- Añadir más tipos de gráficos (como mapas de calor, gráficos de dispersión, etc) o modificar los que tenemos para buscar mejorar siempre las representaciones visuales de los datos.
- Mejorar la eficiencia de la aplicación cuando se maneje un gran volumen de datos. Para ello, recibiremos trazas en intervalos de tiempo y no continuamente. Así podremos recalcular las estadísticas de una forma más ligera.
- Descargar todas las trazas de los videojuegos cuando eres desarrollador. Dándole una gran libertad y flexibilidad en cuanto a los datos generados de sus propios videojuegos.

- Añadir más formatos en las opciones de descarga de las listas de estudiantes y sus claves de juego. Actualmente, tenemos la opción de descarga en formato CSV. Ampliar estas opciones de descarga en los formatos *Excel* y *PDF* dará a los profesores una mayor flexibilidad a la hora de manejar estas listas.

9. Conclusions and future work

In this project, we have successfully developed the JaXpi ecosystem, consisting of a TypeScript library using the xAPI standard, an intermediary server that manages the data, and a web application that displays data analytics through charts and statistics. We have managed to get these three tools to work together to study data generated by web video games using analytics.

The library can be imported into different JavaScript games to generate traces, store them, and later analyze. This makes the library easy to adapt and extend, allowing it to meet the needs of developers.

By being based on xAPI, which is widely used in the e-learning field, the library ensures that the traces sent are easy to collect and analyze by any LRS server, allowing the library to integrate with other educational tools that use the standard.

On the other hand, the server and the analytics web application work together to collect and store the data in a secure and efficient way, and then transform that data into charts and information that can be easily understood by users. While performing the analytics for the game *Prince of JS*, we have been able to see the potential of collecting and analyzing this data to enrich the learning process of students playing educational video games.

During the project development, we learned about the application of the xAPI standard to video games with an educational purpose, as well as about the different tools that already exist for the same purpose. Additionally, we have learned a lot about the software development cycle, specifically how to develop web tools that work together for a common goal.

In conclusion, JaXpi benefits developers by detecting bugs or shortcomings in their games, teachers by allowing real-time tracking of their students' performance in game sessions and reviewing them later, and students by letting them see their own statistics and understand where they can improve. When used in conjunction with other learning tools, we believe it can provide a much more effective learning experience thanks to the detailed tracking and analysis of student activity.

9.1 Future Work

Below, we will mention some functionalities or improvements that we could implement in the future:

- Further expand the catalog of verbs and objects that the library can use, as well as add testing tools for external tests, making the integration of the library even easier.
- Add testing code to send an example trace(s) to an LRS as a debugging aid for the developer.
- Add a new functionality to the library that sends the pending data to the LRS in case the window where the game is running is closed.
- Configure CORS to be more permissive, allowing developers to run their game on different ports.
- Develop an administration page from which an administrator user can perform CRUD operations to manage the database.
- Deploy the server in a production environment, migrating to a more robust infrastructure and implementing HTTPS for more secure communications, as well as adding tools that allow monitoring to detect errors or attacks.
- Test Jaxpi with more JavaScript video games and especially with serious games, as this work is focused on developing tools that allow us to perform learning analytics in the educational field.

- Create analysis profiles based on game genres. Currently, we perform analytics on a specific type of game (platformers). But without a doubt, being able to adapt our analysis to different game genres (graphic adventures, puzzles, racing, etc.) is an ideal feature, as we could differentiate the game's genre as well as its mechanics, since each one handles specific mechanics, and the data is not interpreted in the same way. In this way, we could have a wide range of games with data to analyze, and do so correctly.
- Create analysis profiles based on the xAPI verbs used by the games. We mainly use the verbs "started" and "completed" to calculate completion times for the game or levels, for example. This works with games that have levels like *Prince of JS*, but for those that don't, there may be other more important verbs that mark game events. Each verb gives us a different type of information about the players.
- Add more types of charts (such as heat maps, scatter plots, etc.) or modify the existing ones to continuously improve data visualizations.
- Improve the efficiency of the application when handling a large volume of data. To do this, we will receive traces at time intervals instead of continuously. This way, we can recalculate the statistics in a lighter way.
- Allow developers to download all game traces. This would provide great freedom and flexibility regarding the data generated by their own video games.
- Add more formats to the download options for student lists and their game keys. Currently, we offer CSV format downloads. Expanding these options to include Excel and PDF formats will give teachers more flexibility in managing these lists.

Bibliografía

Navarro Morales, J. J. (2021). *Estudio de la funcionalidad de Vue 3 en aplicaciones web*. Retrieved June 10, 2024, from https://ebuah.uah.es/dspace/bitstream/handle/10017/49951/TFM_Navarro_Morales_2021.pdf?sequence=1&isAllowed=y

Tin Can API. (2024, 16 de agosto). Wikipedia, La enciclopedia libre. Fecha de consulta: 18:58, agosto 16, 2024 desde https://es.wikipedia.org/w/index.php?title=Tin_Can_API&oldid=161904894.

Shaffer, D. W., Squire, K. R., Halverson, R., & Gee, J. P. (2005). Video games and the future of learning. *Phi delta kappan*, *87*(2), 105-111. https://journals.sagepub.com/doi/abs/10.1177/003172170508700205

Sánchez-Mena, A., & Martí-Parreño, J. (2017). Teachers acceptance of educational video games: A comprehensive literature review. *Journal of e-Learning and Knowledge Society*, *13*(2). https://www.learntechlib.org/p/188115/

Takeuchi, L. (2011). Kids closer up: Playing, learning, and growing with digital media. *International Journal of Learning and Media*, *3*(2). https://www.researchgate.net/profile/Lori-Takeuchi-2/publication/216841561_Kids_Closer_Up_Playing_Learning_and_Growing_with_Digital_Media/links/0a85e538d0a643f645000000/Kids-Closer-Up-Playing-Learning-and-Growing-with-Digital-Media.pdf?origin=journalDetail&_tp=eyJwYWdlIjoiam91cm5hbERldGFpbCJ9

Shreve, J. (2005). Let the Games Begin. Video Games, Once Confiscated in Class, Are Now a Key Teaching Tool. If They're Done Right. *George Lucas Educational Foundation*. https://eric.ed.gov/?id=ED484796

Squire, K. (2005). Changing the game: What happens when video games enter the classroom?. *Innovate: Journal of online education*, *1*(6). https://www.learntechlib.org/p/107270/

Kenny, R., & Gunter, G. (2011). Factors affecting adoption of video games in the classroom. *Journal of Interactive Learning Research*, *22*(2), 259. https://rkenny.org/6507/jillr.pdf

Alonso Fernández, C., Calvo Morata, A., Freire, M., Martínez-Ortiz, I., & Fernández Manjón, B. (2021, June 18). *Data science meets standardized game learning analytics*. IEEE. https://ieeexplore.ieee.org/abstract/document/9454134

Apéndice A. Registro de trabajo

Afortunadamente, hemos podido realizar muchas reuniones tanto remotas como presenciales con los tutores cada dos o tres semanas. Esto nos ha ayudado mucho a la hora de debatir nuestras ideas y dificultades con ellos.

Por otro lado, entre nosotros hemos realizado diferentes llamadas semanales a través de Discord y Google Meet por horarios muy distintos y de este modo, junto a Whatsapp, hemos mantenido una comunicación bastante fluida a pesar de las dificultades para coincidir. Es un punto muy positivo ya que nos hemos visto involucrados los 3 a lo largo de todo el proyecto en todo momento.

A continuación, mostraremos un registro en formato tabla con las diferentes reuniones que hemos realizado a lo largo del curso junto a los tutores.

Fecha	**Descripción**
18/10/23	Presentación del tema elegido, preguntas iniciales de conocimientos, nombramiento de tecnologías a manejar: MEAN, NPM, xAPI. Explicación de las 3 partes que podemos dividir el TFG.
27/10/23	Presentación del github con los 3 repositorios creados. Dudas sobre xAPI. Juntamos enlaces para ver diferentes LRS para poder enviar trazas y un enlace para diferentes juegos hechos en js que podríamos utilizar.
03/11/23	Debatir sobre 2 scripts que realizamos para cliente y servidor. Dudas sobre los LRS.
17/11/23	Enseñar la validación de las trazas. Dudas sobre los JSON ya que añadimos ejemplos de trazas .JSON. Elegimos 3 tipos de juego,plataformas, aventura gráfica y puzzles.En concreto uno y probamos a enviar trazas de comienzo y fin de actividad.
04/12/23	En el servidor se establece una jerarquía para guardar trazas, utilizando un id de usuario, un id de sesión y un token del juego. Para librería decidimos pasar de un modelo de importación de JSON ejemplo a una importación con URL canónica, además se planteó la idea de utilizar código generativo para la creación del llamado a las trazas.
18/12/23	Al servidor se le agrega un sistema para mantener sesiones con Express,

	además se incluyen rutas para poder obtener los JSON de prueba En librería ocurre un cambio de paradigma en lugar de usar una función de encolado al que se le pasan trazas, se generan funciones para cada verbo dejando el objeto como un parámetro aprovechando el código generativo, manteniendo un verbo custom en el que poder encolar cualquier traza. Además se incluye el código de Prince of Persia para construir una demo.
28/02/24	Servidor agrega una creación de cuentas para los distintos tipos de usuario, con distintos niveles de acceso y mejora la gestión de la información del cliente para poder recibir trazas. Por parte de la librería ocurren correcciones en el código generativo que dividen la parte dinámica de la estática y se establece una política de envío de trazas. En cuanto a las analíticas se ha ido probando el uso de D3 como posible opción para dibujar gráficos junto a Vue, elegido como framework definitivo.
13/03/24	En el servidor se agregan más rutas en función del tipo de usuario.Para el análisis se sugiere un filtrado de trazas para facilitar su visualización y estudio tomando como ejemplo Snola. La librería agrega documentación autogenerada con JSDoc, se restringe qué objetos pueden ser utilizados con qué verbos se incluyen Web Workers para evitar bloquear el juego. Se incluye un validador de trazas previo al envío y una gestión de trazas con localStorage para prevenir pérdidas de datos en caso de error en el envío, además se utilizan políticas de gestión de señales. Ahora la librería acepta en su constructor los parámetros para poder conectarse a un LRS en lugar de en una función. Analitica elige C3 como la mejor opción para visualizar datos mostrados y empieza a construir un prototipo de algunos tipos de gráficos de barras y de líneas.
10/04/24	Para servidor y análisis ocurre una fusión de la creación de usuarios y la visualización de los datos de cada uno. Además se mejora la autenticación de los usuarios. Para los análisis se sugiere el uso de WebSockets para una visualización de datos a tiempo real. Se han realizado alguna prueba de flujo de trazas JSON para realizar las primeras analíticas que hemos pensado, como el tiempo en completar un nivel. En la librería ocurre una transformación del código de los verbos a funciones verbo-objeto anidadas y se comprueba que la librería funciona con LRS convencionales
17/04/24	En análisis, se decide usar la biblioteca de js sockets.io y se descubre del problema de actualizar a cada traza, sobre todo con múltiples usuarios y por ello, se propone la investigación de usar un temporizador u otro método. La librería en un principio mantiene una versión para navegador y otra versión para aplicación pero decide no mantener la segunda, pues no está en el ámbito del proyecto.

24/04/24	En el servidor se cambian las políticas de envío para utilizar un token de usuario para la autenticación y devuelve un token de sesión al cliente que utilizará en las trazas, se agrega la política CORS para el manejo del acceso a la web. En análisis se crean nuevas vistas y nuevos gráficos, además crean nuevos test para generar trazas y mejorar sus pruebas. La librería completa la demo del juego sin percances y expande su número de verbos y objetos generados.
08/05/24	Hay un cambio en el servidor debido a la disparidad entre actor y usuario, se crean claves de sesión que los usuarios utilizaran para poder jugar, además se cambia el token de usuario por el token de juego para el envío. Ahora los alumnos son generados por el profesor en lugar de registrarse en la web. Se plantean la posibilidad de agregar más juegos como demo. Analitica crea un gran número de maquetas de vistas para la web con las funcionalidades que cada tipo de usuario diferente puede necesitar.
08/07/24	Servidor añade un autenticador de claves que los menús de juegos pueden usar para la clave de sesión además hashea la clave de sesión de las trazas recibidas antes de guardarlas en la base de datos. También se implementaron las rutas para generar GameTokens. Al recibir los datos en la aplicación web, se tenía en cuenta el campo actor, y como hubo una disparidad entre este campo y usuario, tuvimos que cambiar los campos que se tenían en cuenta antes por los nuevos, ajustando la interfaz de usuario y algunas funciones. Se establece usar Pinia como gestor de estados en la aplicación. Generamos pruebas con el juego PoP con diferentes estudiantes, diferentes clases, diferentes sesiones. En la librería ahora se envía la clave sesión en lugar del token de usuario en las trazas. Ahora los juegos necesitan una forma de conseguir una clave de sesión para jugar identificados al enviar trazas, para lo que se desarrolla un menú para ellos.
16/07/24	En servidor se agrega un enriquecedor de trazas con los datos de contexto (profesor, clase, institución, …) En el backend se implementan gran cantidad de nuevas rutas y modelos Mongoose para cumplir con las nuevas funcionalidades de crear grupos de alumnos y sesiones de juego. En la web se transforma el modelo de los alumnos de usuario y contraseña a solo contraseña (clave sesión) generada por los profesores. Debatimos sobre qué gráficos quitar y cuales dejar, qué datos relevantes faltan y modificamos funciones en servidor para que lleguen las trazas que necesita la app para pintar adecuadamente los gráficos. Para libreria se incluyen 2 nuevas demos, y se crea el archivo JavaScript descargable con toda la funcionalidad de JaXpi para proyectos en ámbito global, se añade una política de control sobre las trazas guardadas en localStorage para evitar colapsar la memoria, se corrigen bugs preparándose para versión final.