Universidade Vigo

DESARROLLO DE DASHBOARDS PARA E-LEARNING

Martín Blanco Landa

Trabajo de fin de grado
Escuela de Ingeniería de Telecomunicación
Grado en Ingeniería de Tecnologías de Telecomunicación

Tutor
Manuel Caeiro Rodríguez

Curso 2020/2021

Contenido

1 Introducción

Los Lenguajes de Modelado Educativo (EML) [1], en especial el estándar IMS Learning Design (IMS-LD) [2], surgieron a principios de los años 2000 en el área de la tecnología educativa como una propuesta de notación y modelado computacional destinados a documentar escenarios de enseñanza/aprendizaje, como pueden ser unidades didácticas, planes de lecciones, cursos completos, etc. La característica principal de estos lenguajes es que siguen una aproximación centrada en actividades, de forma similar a como se hace con los lenguajes de *workflow* o sistemas basados en procesos. Para la descripción de los escenarios de enseñanza/aprendizaje, los EMLs se centran en la caracterización de las actividades que tienen que llevarse a cabo, indicando para cada una de ellas los roles participantes como profesores y alumnos, recursos necesarios, servicios, etc. Se establece también la secuencia para llevar a cabo actividades, transferencias de documentos e información entre actividades, restricciones sobre el acceso, o condiciones de completitud. El estado de la tecnología actual en forma de ordenadores portátiles, smartphones y acceso a internet en las aulas, permite que dichas descripciones o modelos se puedan llevar a cabo en base a su procesamiento por aplicaciones específicas, como podrían ser sistemas de gestión de aprendizaje o LMSs, bien en contexto educación presencial o del *e-learning*.

Durante varios años estos lenguajes captaron un cierto interés de la comunidad investigadora y de innovación educativa en base al uso de tecnologías. Se han publicado artículos con propuestas de EMLs e implementaciones de sistemas diversa índole, como herramientas de autoría y de soporte a la ejecución, así como estudios sobre su usabilidad y utilidad. Ahora bien, a pesar de todo ello la realidad a día de hoy es que este tipo de sistemas no ha sido adoptado por la comunidad educativa y el interés por los mismos ha disminuido en los últimos años.

Una de las cuestiones sobre las que se han desarrollado más propuestas y que puede tener un papel más relevante para facilitar la adopción es el de la representación gráfica de los modelos [3]. Por una parte, la representación gráfica es clave para facilitar la creación de los modelos por parte de los diseñadores y creadores de escenarios de enseñanza/aprendizaje. Por otra parte, las representaciones gráficas también juegan un papel destacado durante el desarrollo de las actividades por parte de profesores y alumnos, que deben poder interpretarlos y utilizarlos de forma sencilla. También pueden ser una ayuda durante la búsqueda y selección de los escenarios, permitiendo la comparación entre ellos.

2 Objetivos

El objetivo de este trabajo es desarrollar una aplicación informática que permita probar un nuevo concepto de visualización gráfica para EMLs. Esta aplicación está dirigida a dar soporte a profesores y alumnos durante el desarrollo de experiencias de enseñanza/aprendizaje. En concreto, se hará uso de una propuesta de visualización ya existente que hasta el momento solo ha sido considerada durante la autoría de escenarios de enseñanza/aprendizaje. Por tanto, la propuesta original será implementada y extendida de manera sustancial considerando su uso durante la realización de actividades de enseñanza/aprendizaje, con nuevas funcionalidades y servicios. Se pretende ofrecer una asistencia amplia al profesor y al alumno incluso durante el transcurso de cada clase, y no sólo a priori y a posteriori, que es donde mayormente se suele poner el foco.

La propuesta existente en la que se basará este trabajo es el formato de planificaciones de enseñanza propuesto por *LePlanner* [4]. Se trata de una innovadora plataforma de uso público y gratuito, creada y gestionada por la *Universidad de Tallin*, que busca asistir al profesorado con una plataforma en la que pueden planificar las actividades a desarrollar durante una clase. Se presenta como una representación gráfica secuencial, en la que el eje horizontal corresponde al tiempo transcurrido de la sesión y el vertical la carga cognitiva de los distintos bloques de actividad dentro de la sesión. En la *Figura 1* se puede observar un ejemplo real de una planificación de clase de *LePlanner*. Se incluye en el Anexo A un análisis completo de su estructura y comportamiento.

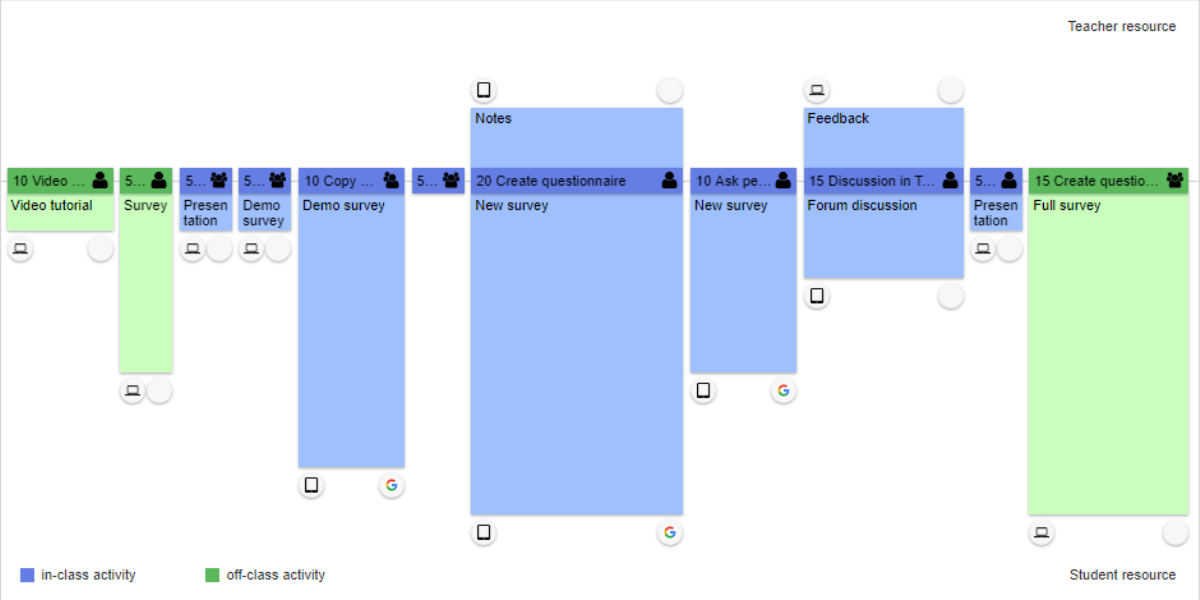

Figura 1: Ejemplo real de una planificación en LePlanner

3 Requisitos

El primer paso, necesario para perseguir los objetivos propuestos y abordar el desarrollo de la app de una manera estructurada, es definir los requisitos funcionales y no funcionales. Para esta finalidad, se ha optado por crear un informe de captura de requisitos, conformando las distintas interacciones entre los usuarios y el sistema que la aplicación final ha de tener.

Las características del sistema a desarrollar son:

- Formado por sesiones de clase individuales, cada una correspondiente a un escenario de enseñanza/aprendizaje único, usando como modelo de entrada el de *LePlanner*.
- A cada sesión le corresponde un PIN de acceso. Cualquier usuario con ese PIN puede acceder a la sesión.
- Accesible por usuarios con dos roles: estudiantes y profesores.
- Cada clase está compuesta por una serie de bloques de actividad y un reloj central que recorre los distintos bloques, indicando a todos los usuarios, en todo momento, la actividad en la que se encuentra, así como brindando acceso a todos los recursos y parámetros de dicha actividad, hasta el fin de la clase.
- Debe ofrecer un comportamiento interactivo, en el que los usuarios pueden interactuar con el sistema, e incluso entre ellos.
- Todos los eventos que ocurren en una sesión quedan registrados en una línea temporal y lista cronológica, accesible tanto durante la clase como a posteriori, quedando así un informe del transcurso de la clase.

Para los usuarios con rol de profesor, se buscan las siguientes funcionalidades:

1. Tienen la capacidad de crear una nueva sesión de clase y de acceder a una existente.
2. Tienen la capacidad de comenzar, pausar, reanudar y parar una clase en cualquier momento.
3. Tienen la capacidad de interactuar con uno o más alumnos (por ejemplo, pidiendo su intervención).
4. Tienen capacidad de lectura de todos los parámetros de la clase y parámetros individuales de cada bloque de actividad.
5. Tienen capacidad de escritura de ciertos parámetros de los bloques de actividad (por ejemplo, anotaciones públicas).

Para los alumnos, se buscan las siguientes:

1. Tienen la capacidad de acceder a una clase existente.
2. Tienen capacidad de lectura de todos los parámetros de la clase y parámetros individuales de cada bloque de actividad pudiendo observar la evolución dinámica de los mismos.
3. Tienen la capacidad de interactuar con el profesor (por ejemplo, levantando la mano para llamar su atención).

Además, la aplicación debe cumplir una serie de requisitos no funcionales fundamentales:

- **Compatibilidad:** La app debe funcionar en cualquier navegador con acceso internet, independientemente del dispositivo: PC, tableta, teléfono inteligente. Podrá ser, además, instalada como aplicación en aquellos dispositivos que lo admitan, como tabletas o smartphones iOS o Android.
- **Usabilidad:** La aplicación debe poseer un diseño *responsive* a fin de garantizar la adecuada visualización en dispositivos con tamaños de pantalla distintos. Así mismo, las interfaces gráficas deben ser intuitivas, claras y bien formadas, con el fin de que el tiempo de aprendizaje y la experiencia de uso resulte muy sencilla incluso para usuarios no experimentados. Todos los mensajes de error serán informativos y orientados al usuario final.
- **Eficiencia y disponibilidad:** El sistema debe ser capaz de operar con un alto número de usuarios manteniendo tiempos de respuesta bajos (deseablemente imperceptibles por el usuario). Debe, además, ser capaz de albergar varias sesiones de clase simultáneas. Así mismo, se busca que la aplicación sea un recurso accesible en internet de manera global y pública y que tenga un alto tiempo de disponibilidad, para no mermar la experiencia de usuario.

En el Anexo B se adjunta la captura de requisitos completa que sirvió para establecer las bases necesarias para abordar el desarrollo de la aplicación.

4 Diseño

4.1 Tecnología implicada en la aplicación

El siguiente paso a la hora de empezar a construir una aplicación es seleccionar las tecnologías y el software que se va a utilizar para su desarrollo.

La primera cuestión a afrontar consistió en buscar el tipo de aplicación que se adecuase al objetivo buscado. Entre las distintas opciones están la clásica web, la posibilidad de desplegar una aplicación móvil, o incluso utilizar mecanismos más modernos como *webs responsive* o *progressive web apps*.

Tras un proceso de estudio y selección, se ha decidido finalmente programar una *progressive web app* (PWA), pues reúne una serie de requisitos que encajan a la perfección con lo que se busca en este trabajo: flexibilidad, velocidad de carga, ligereza, accesibilidad en cualquier dispositivo con acceso internet sin necesidad de instalación, soporte para notificaciones *push*, etc. En el Anexo C se adjunta un pequeño estudio en el que se realiza una comparativa de las distintas tecnologías [5] y, teniendo en cuenta las necesidades particulares de este trabajo, se busca la mejor opción, justificando su elección.

La plataforma elegida para el desarrollo de la aplicación es *Ionic* [6]. Aunque su foco principal son las apps híbridas, tiene compatibilidad absoluta con el despliegue de *Aplicaciones Web Progresivas*. Destaca por tener un desarrollo muy activo, una documentación muy buena y una grandísima comunidad. Entre las ventajas de usar *Ionic* están la posibilidad de desplegar con un solo comando la aplicación en *Android*/*iOS*/*Web*, la utilización de componentes propios que sirven de puente con componentes nativos de *Android*, *iOS* y *Web*, de manera que un único código se traduce, en la renderización del cliente, en componentes de su sistema nativo, y un muy buen soporte al programador.

El *framework* sobre el que se va a usar *Ionic* es *Angular* [7], un famoso marco para aplicaciones web *frontend* modernas (donde todo el código es ejecutado y renderizado en el lado del cliente), desarrollado en *TypeScript* (un superconjunto de JavaScript que añade capacidad de tipado estático y objetos basados en clases). Este *framework* ofrece una base para el desarrollo de aplicaciones robustas y de alta escalabilidad y optimización, promoviendo un estilo de codificación homogéneo y de gran modularidad. Con Angular, la navegación entre secciones y páginas de la aplicación, así como la carga de datos, se realiza dinámicamente y asíncronamente haciendo llamadas al servidor, sin necesidad de refrescar la página en ningún momento. Esto implica que, si por ejemplo hay un cambio en la base de datos, este es recibido y actualizado automáticamente en el lado del cliente sin necesidad de esperar o refrescar la página que se está viendo. Esta cualidad resulta de gran interés para este proyecto donde, en cada sesión de clase, habrá mucho dinamismo y actividad.

Así pues, el *stack* con el que se va a trabajar es HTML5, CSS3 y *TypeScript* (Equivalente a *JavaScript*. De hecho, *NodeJS*, el entorno de ejecución para *JavaScript* usado en el lado del servidor, traduce código *TypeScript* a *JavaScript* en tiempo de ejecución).

4.2 Arquitectura, aplicaciones utilizadas y flujo de funcionamiento

Una vez definida la tecnología, ya es posible adentrase en la estructura y arquitectura de la app, desgranando las distintas partes que la componen.

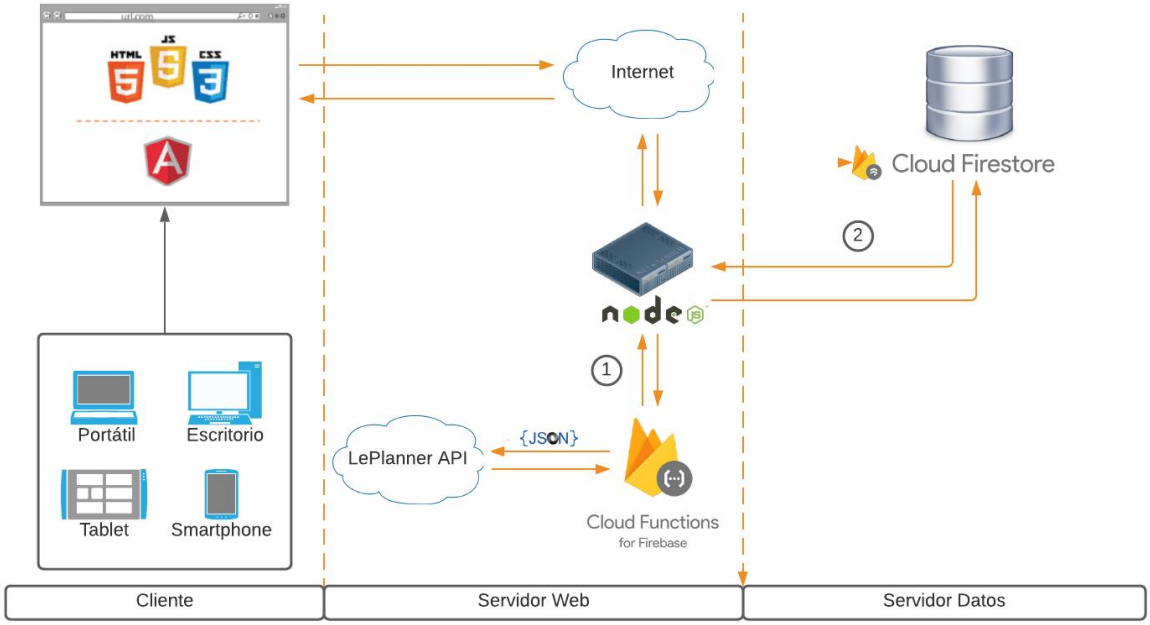

Figura 2: Arquitectura de la aplicación

En el diagrama de la *Figura 2* se detalla el flujo completo de la aplicación. Antes de entrar a detallar las distintas partes y la secuencia de funcionamiento, conviene primero presentar *Firebase*, cuyos servicios *Cloud Functions* y *Cloud Firestore* se utilizan en este proyecto.

Firebase (de *Google*) no es más que un conjunto de herramientas situadas en la nube, integradas con *Google Cloud Platform*. Es una suite que incluye servicios interesantes como la ejecución de funciones a nivel de servidor y que se utiliza en este proyecto (*Cloud Functions* [8]), una base de datos NoSQL orientada a documentos que también se emplea para este trabajo (Cloud *Firestore* [9]) y otros servicios adicionales como la gestión de autenticación y credenciales en aplicaciones (identificación y registro), Almacenamiento de archivos, Hosting y Machine Learning, entre otros.

Se eligieron los servicios de *Firebase* para este proyecto debido a su buen acoplo y compatibilidad con *Angular*. Las bibliotecas existentes de *Firebase* para *Angular* tienen una gran popularidad y ambos suelen emplearse de manera conjunta. Por otro lado, *Firebase* incluye un servicio gratuito para proyectos pequeños que apenas requieren almacenamiento y ancho de banda, que se amolda muy bien a este trabajo. Por último, se consideró conveniente *Firebase* debido a que el modelado de datos usados en esta aplicación, y la relación entre ellos, no tienen una alta complejidad, por lo que *Cloud Firestore* con su modelo de datos orientado a documentos (muy similar a JSON) se adapta muy bien a las necesidades de la aplicación.

Volviendo a la arquitectura, en el nivel más alto, se puede establecer una separación en tres partes: Cliente, Servidor Web y Servidor de datos.

- **Cliente.** Es el usuario final, en este caso profesor o alumno, que accede a la aplicación desde su navegador o aplicación *Android*/*iOS* con un dispositivo. Como estamos ante una aplicación de tipo PWA, se puede acceder desde prácticamente cualquier dispositivo con acceso a internet, sea con la URL accediendo desde navegador o a partir de la instalación de la app. Al ser *Angular* una tecnología *client side*, es el cliente el que construye una vista a partir del código *JavaScript* y los datos que recibe del servidor, mediante la manipulación del *Document Object Model (DOM)* llevada a cabo por el propio navegador. Es decir, la renderización se produce desde la máquina del cliente, y no en el servidor como en otras tecnologías clásicas (PHP, Java, Python).
- **Servidor Web.** Está compuesto por el servidor principal que lanza la aplicación. *Ionic* utiliza *NodeJS* [10], un entorno de ejecución de *JavaScript* orientado a eventos asíncronos pensado para la creación de aplicaciones escalables. Desde aquí se lanza la aplicación y gracias a *NodeJS* se puede acceder a ella (y a sus datos) desde el dispositivo del cliente.

  Además, el servidor principal se apoya de *Cloud Functions*, servicio de *Firebase*. En este caso, nos apoyamos en las *Cloud Functions* para ejecutar bloques de código a nivel de servidor en caso de que sea necesario y no sea posible, o no convenga, que se ejecute desde el lado del cliente.
- **Servidor de datos.** Se utiliza en este trabajo *Cloud Firestore*, un servidor de base de datos en la nube que ofrece *Firebase*. Es *NoSQL* (bases de datos no relacionales y que no utilizan SQL como lenguaje de consultas) y orientado a documentos, muy comparable al conocido JSON. Entre las bondades de *Firestore*, destaca su flexibilidad (pues admite estructuras de datos flexibles y jerárquicas, almacenadas en documentos, que a su vez se organizan en colecciones). Otra virtud que resulta clave destacar por su extrema importancia en este proyecto es su capacidad de ofrecer actualizaciones en tiempo real. Lo hace usando la sincronización de datos para actualizar los datos de todos los dispositivos conectados. Esto permitirá en la implementación del proyecto que elementos como relojes, recursos en cada bloque de actividad de la clase, anotaciones añadidas en directo, entre otros, sean inmediatamente visibles por todos los usuarios. Todos los cambios son actualizados para todos los dispositivos en tiempo real, sin necesidad de que ningún usuario actualice la página.

Ya introducidas las distintas partes, se puede profundizar en el flujo de funcionamiento de la aplicación desde el momento en el que un usuario accede, que sigue el esquema de la *Figura 2*.

Una vez el cliente entra en la aplicación realiza una petición al servidor donde *NodeJS* está en ejecución. *NodeJS* responde devolviendo al navegador una serie de dependencias *JavaScript*. Ya recibidas, procede a ejecutar diversas funciones que alteran el DOM, y finalmente así construye la vista del cliente.

Para la obtención de datos, se distinguen claramente dos vías distintas según la petición del cliente, etiquetadas en la *Figura 2* como **1)** y **2)**:

1) Si un profesor (cliente) solicita la creación de una sesión de clase, y teniendo en cuenta que la información de la planificación de las clases está alojada en https://beta.leplanner.net/ (web externa), se procede con la solicitud de dicha información mediante una petición HTTP GET. Este proceso

el servidor central (*NodeJS*) lo delega al componente *Cloud Functions*, servicio en la nube de *Firebase*, que rápidamente solicita y recibe un objeto JSON con la información. Esta la procesa y adapta para poblar inmediatamente la base de datos propia de la aplicación. Así, se logra importar la planificación del escenario de clase de Leplanner en nuestro servidor, y dejarla completamente lista para su manipulación y trabajo con ella, evitando posteriores dependencias y comunicaciones con servicios externos.

Se ilustra, en el diagrama siguiente (*Figura 3*), el flujo completo desde que el profesor inicia la app hasta que compone su vista con la sesión de clase y toda su información, lista para trabajar.

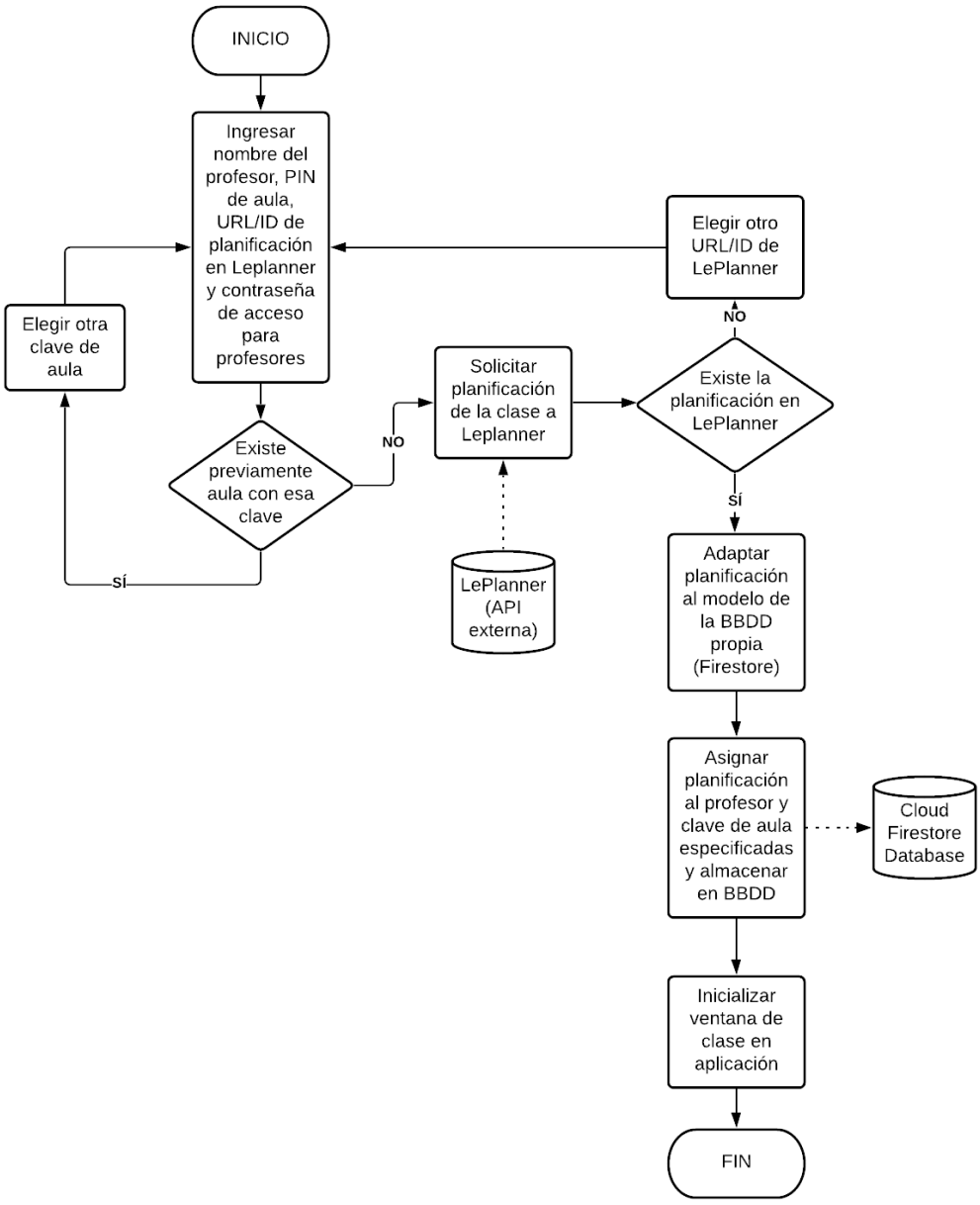

Figura 3: Diagrama de flujo: Profesor crea una sesión de aula

2) Cada vez que se quiere acceder a una sesión de clase ya existente, sea como profesor o como alumno, *NodeJS* se comunica directamente con la base de datos *Cloud Firestore*, que está alojada en la nube (*Firebase*), para recuperar toda la información de la sesión de clase.

Se ilustra, en el diagrama siguiente (*Figura 4*), el flujo completo desde que un usuario inicia sesión hasta que carga una sesión de clase:

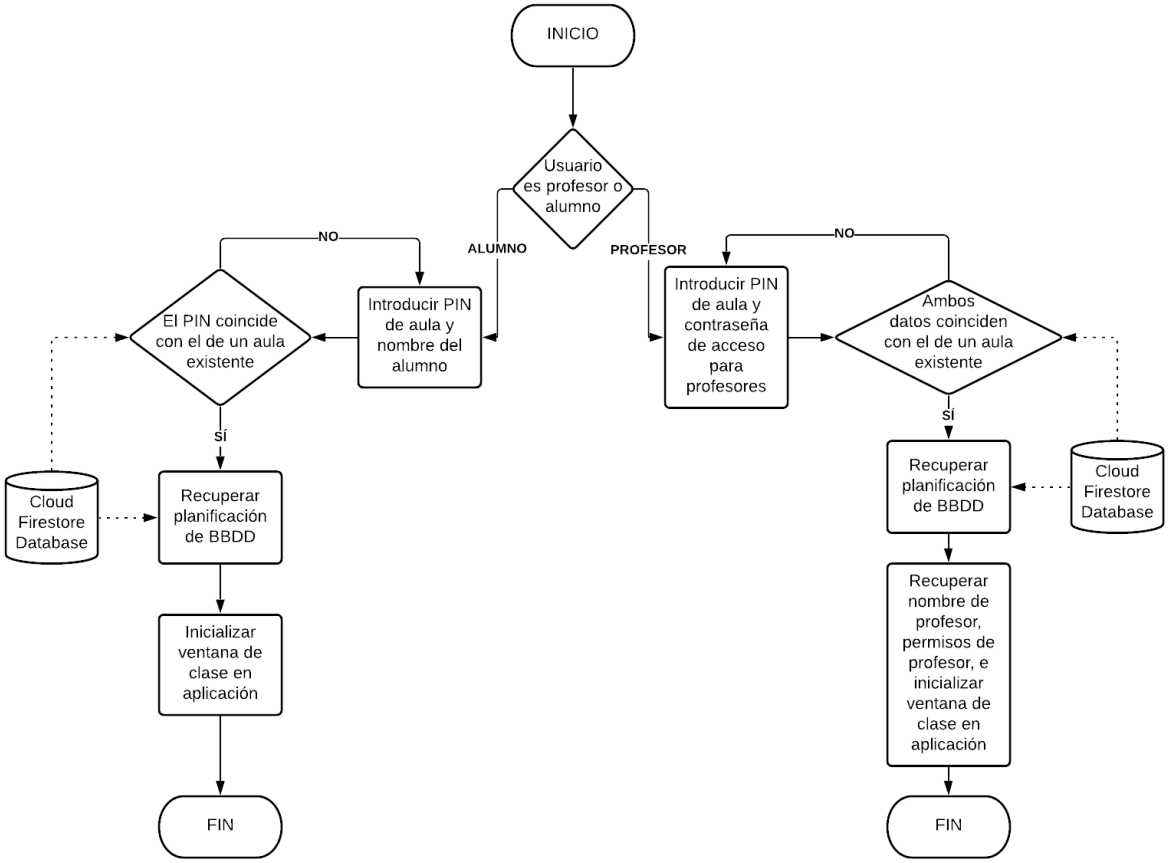

Figura 4: Diagrama de flujo: Profesor/alumno entra a una sesión de aula existente

4.3 Modelos de datos

El modelo de datos con *Cloud Firestore* resulta en un esquema de gran sencillez, sin que eso implique ninguna limitación [11]. De hecho, la velocidad de lectura y escritura de datos con este sistema es muy rápida y ligera, lo que le otorga gran escalabilidad.

Como se describió en el apartado de arquitectura, *Firestore* es una base de datos NoSQL alojada en la nube (*Google*) a la que accede directamente la aplicación para realizar escrituras y consultas. Su jerarquía es sencilla: una base de datos está compuesta *documentos*, que contienen campos a los que se asignan valores (tipo *clave-valor*). A su vez, los documentos se almacenan en *colecciones*, que son contenedores de documentos, útiles para organizar datos y compilar consultas de manera sencilla y eficiente.

Cada documento es flexible, en el sentido de que admite varios tipos de datos distintos, desde tipos simples hasta objetos anidados complejos. Las consultas también resultan flexibles y eficientes, pues brinda la posibilidad de crear consultas específicas a nivel de documento, sin la necesidad de recuperar una colección completa.

En la *Figura 5* se puede apreciar la estructura de la base de datos de la aplicación. Está compuesta por únicamente dos colecciones:

1. SCENARIOS: Es una colección que contiene distintos documentos, cada uno de ellos correspondiendo a una planificación de clase (escenario de *LePlanner*). Los documentos no son más que una **réplica** de las planificaciones de clase obtenidas de la web *LePlanner* mediante su API pública.

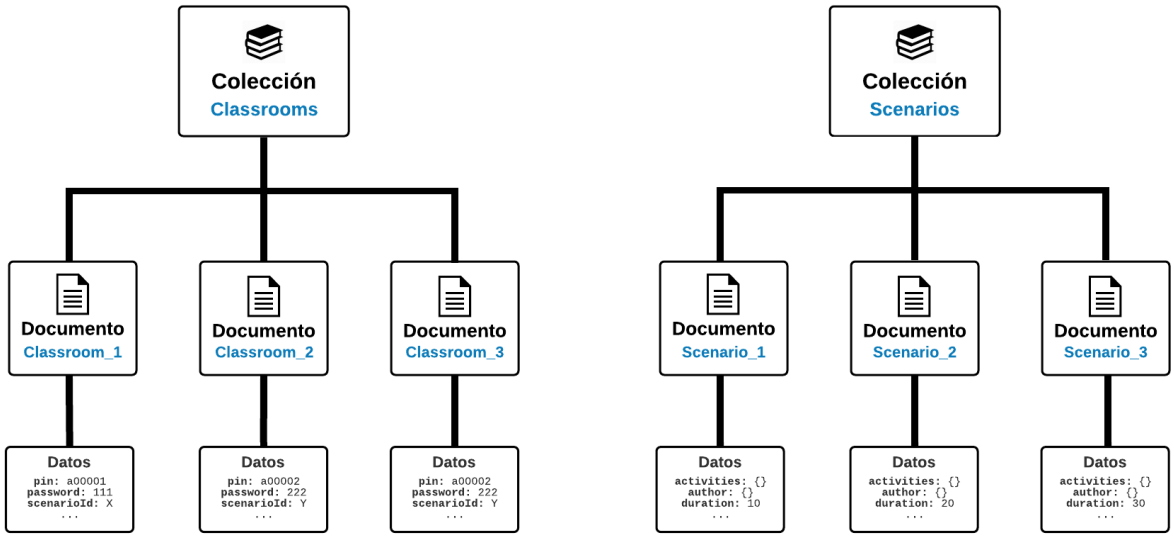

Figura 5: Estructura de la base de datos. Colecciones, documentos y datos.

Para trabajar de manera más cómoda y evitando el abuso de peticiones externas, cada vez que un profesor crea un aula de clase se importan los datos de la planificación de dicha clase de beta.leplanner.net a la base de datos propia de la aplicación. Apenas requiere transformación, ya que los objetos JSON enviados por *LePlanner* con los datos de la planificación utilizan una estructura de clave-valor al igual que *Firestore*. La única diferencia apreciable es que aquellos datos que no resultan de interés, son filtrados y eliminados, conservando solo aquellos campos necesarios para el desarrollo de la app. Se incluye en el Anexo A un breve análisis de los escenarios de *LePlanner* que puede ser de utilidad para comprender el material presentado en esta sección.

Un *escenario* no es más que la planificación de una clase. Cada escenario está compuesto por unas variables básicas (Identificador, nombre, descripción, duración, resultados de aprendizaje (*outcomes*), fecha de creación y, por último, por una serie de bloques de actividad.

Las ***actividades*** contienen detalles de cada bloque de actividad individual dentro de la clase. Estos incluyen nombre, duración, organización de los participantes (individual, por parejas, en grupo), si se realiza en clase o en casa (*in_class*), y, por último, una serie de materiales (*recursos*).

Los ***materiales*** son lo que en la aplicación se conocerá como *recursos*. Cada actividad está compuesta de 0 a 2 recursos. Puede tener, como máximo, uno destinado al profesor y otro destinado al alumno. Cada recurso no es más que una indicación/anotación de lo que el profesor/alumno deberá hacer en cada actividad. Cada recurso puede, además, tener vinculado una URL. En el caso de un alumno, por ejemplo, el recurso podría tener vinculado un PDF conteniendo apuntes. Los recursos, además, llevan asociados otras características importantes:

- ***Esfuerzo cognitivo (involvement level).*** Exclusiva para los recursos del alumno. Indica una medida subjetiva del esfuerzo cognitivo que le requerirá la actividad a un alumno, en una escala del 0 (menos exigente) al 6 (más exigente).
- ***Medios (conveyors).*** Son los medios, alojados en la red, que pueden servir como herramienta para llevar a cabo una actividad. Por ejemplo, si la actividad requiere crear una encuesta, aquí se vincularía una herramienta como *Google Forms*. O si la actividad consiste en un *quiz*, podría añadirse *Kahoot!* como medio.
- ***Pantallas (displays).*** Indica el hardware (o material) con el que es posible seguir y realizar una actividad. Papel, móvil, Tablet, PC, …

En la *Figura 6* se presenta una instantánea completa del modelo de la colección *scenarios*. En dicha colección puede haber desde cero a infinitos documentos *scenario*, uno por cada escenario importado

por un profesor. Cada escenario tiene como nombre su propio identificador, facilitando su ubicación y acceso.

Realmente, un documento está formado por un único gran bloque de campos clave-valor, muchos de ellos anidados (utilizando *arrays* y mapas). En esta figura (y la siguiente), para facilitar la comprensión del lector, se utiliza un esquema clásico, separando los elementos anidados en bloques distintos para representar mejor la naturaleza del documento.

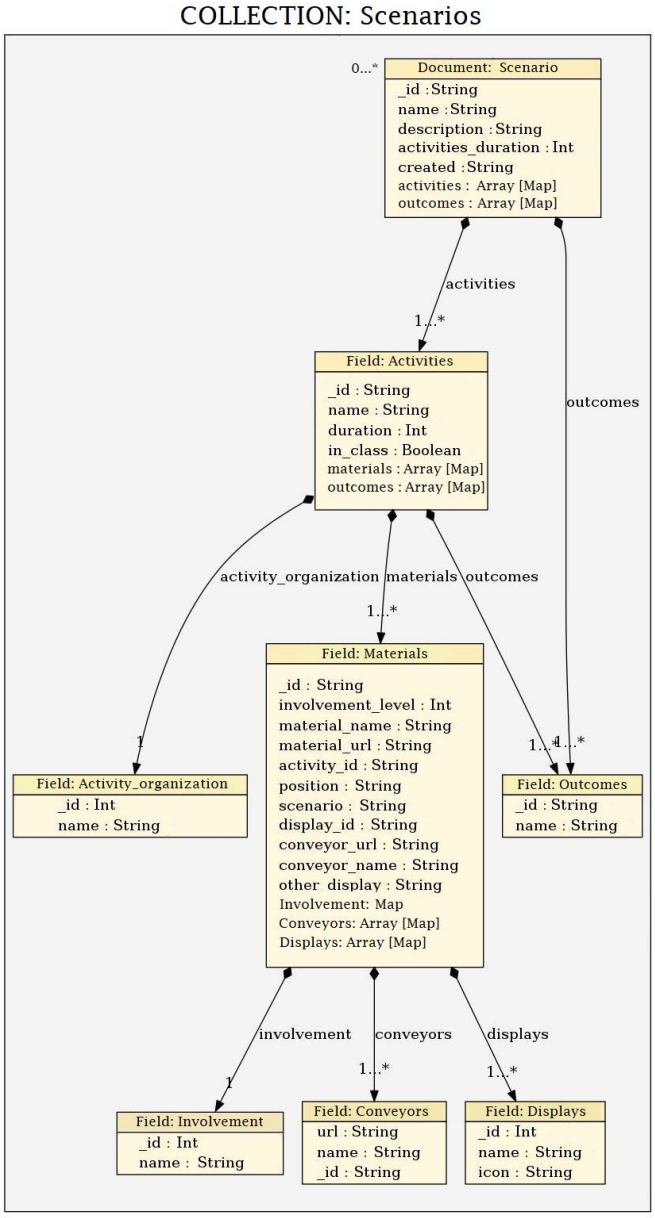

Figura 6: Esquema de datos de un escenario importado de LePlanner.

2. CLASSROOMS. Representan las aulas virtuales creadas por los profesores en esta aplicación. La colección está compuesta por distintos documentos, cada uno de ellos representando un aula virtual distinta. **Cada documento tiene como nombre su PIN** (funciona como identificador ya que sólo puede existir un documento con mismo PIN), facilitando durante la programación su ubicación y acceso.

El valor más importante, que da vida a una sesión de clase, es su escenario asociado de *LePlanner* (varias pueden, incluso, tener asociada el mismo). Sin embargo, como estamos ante una base de datos no relacional, este vínculo no es apreciable a nivel estructural, pues no existen las figuras de claves primarias ni foráneas. Resulta suficiente con almacenar la ID de cada escenario en el campo *scenario*

para que, de forma programática, se pueda poblar una clase con la información de la planificación (escenario) a la que apunta, accediendo a partir de su ID, a su correspondiente documento en la colección *scenarios* y extrayendo de ahí toda su información. A continuación (*Figura 7*), se presenta el esquema completo de la colección *classrooms*:

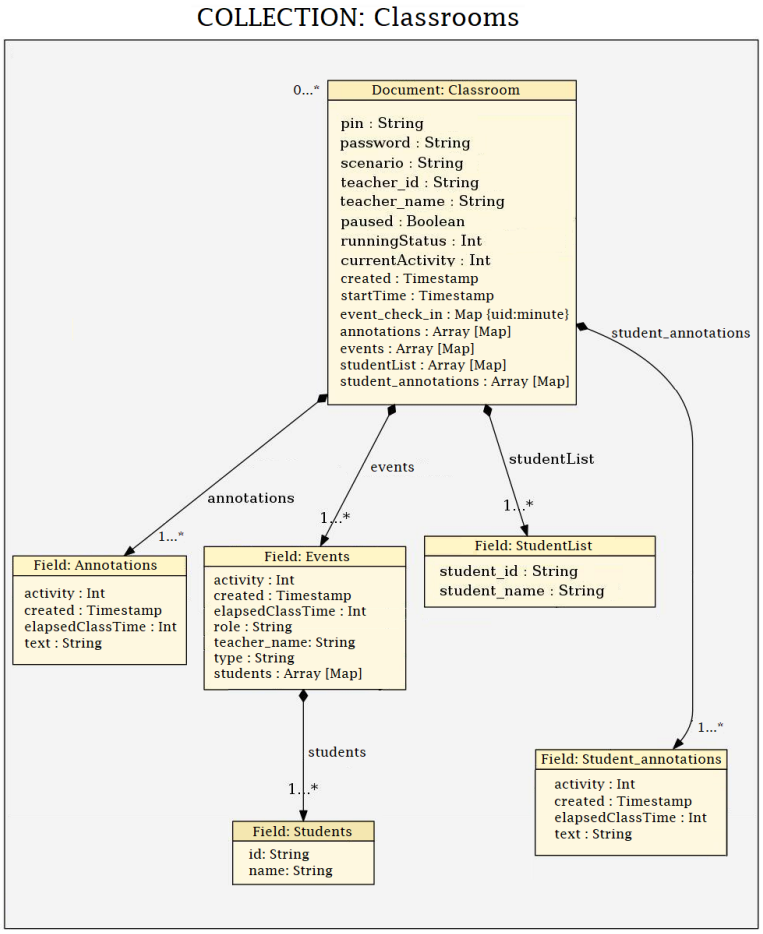

Figura 7: Esquema de datos de una sesión de clase de la aplicación.

El PIN es uno de los valores más importantes de una clase. Es único. En el momento de creación de una clase se asegura que previamente no exista otra clase con el mismo PIN. Además, cada PIN debe tener obligatoriamente 6 caracteres formados obligatorio por letras y dígitos, complejidad suficiente para que en una población pequeña de usuarios resulte altamente improbable que se produzca la repetición de un patrón. **El fin del PIN es compartirlo con los alumnos, pues serán lo que usen para acceder a la sesión.**

Al PIN le acompaña la *contraseña de profesor*, que tiene un fin privado. Sirve para que el profesor pueda acceder en sesiones posteriores manteniendo sus privilegios de profesor. Basta con indicar el PIN de la clase y su correspondiente contraseña, en caso de coincidir, el usuario entrará a la sesión de clase con privilegios de profesor.

El ID y nombre del profesor se almacenan directamente en el documento de la clase, y en caso de que un usuario acceda con privilegios de profesor, automáticamente ambos datos se consultan y guardan en caché, de manera que el dispositivo será consciente en todo momento de quién es y cómo se llama, incluso si se desconecta y se vuelve a conectar.

A partir de ahí, el resto de valores son relativos a la propia sesión de clase en sí:

- ***Paused.*** Indica si la clase está en pausa o no.

- ***runningStatus.*** Indica el estado de ejecución de la clase. 0: No comenzada. 1: En progreso. 2: Finalizada.
- ***currentActivity.*** Indica el bloque de actividad actual en el que se encuentra la clase.
- ***created.*** Fecha y hora de creación del aula.
- ***startTime.*** Fecha y hora de comienzo del aula. Este dato es fundamental para sincronizar los relojes, pues se ha optado por una solución de cálculo del tiempo transcurrido de clase **local**. Es decir, es cada cliente quien compara constantemente cuanto tiempo pasó desde que comenzó la clase hasta el momento actual para actualizar su reloj.
- ***events.*** Los eventos llevan un registro de lo ocurrido en la clase, y se van almacenando de manera cronológica, incluso mostrándose en la barra de progreso con iconos. Existen varios eventos distintos, algunos lanzados por profesores (como añadir una anotación global) y otros por alumnos (como levantar la mano para realizar una intervención). Estos eventos almacenan la actividad en la que se dispararon, sello temporal, rol del usuario que lo disparó (alumno o profesor), identificador de tipo (p. ej. *raise-hand* si un alumno levanta la mano), y el estudiante o lista de estudiantes a las que va dirigida la petición, pues, por ejemplo, un profesor puede pedir la intervención de varios alumnos simultáneamente.
- ***event_check_in.*** Contiene una lista de los usuarios y el índice del último evento que presenciaron (es decir, último evento disparado estando dichos usuarios conectados). Esto permite que, por ejemplo, si un usuario se desconecta de la clase, recibe una notificación (como la petición de intervención a un alumno por parte de un profesor), y entra posteriormente, el usuario sea notificado, pues se comprobará que aún no se sincronizó con el evento en cuestión y lo hará tras la reconexión; Al analizar el evento de petición de intervención, verá que va dirigido a él y recibirá la notificación con la petición. Una vez más, se puede apreciar cómo se ha optado por una solución a nivel de cliente. Existen otras soluciones muy buenas a nivel de servidor como la habilitación de *notificaciones push*.
- ***annotations.*** Son anotaciones que los profesores pueden publicar en cada bloque de actividad, visibles para todos los usuarios. Cada anotación lleva ligado un sello temporal (a nivel global, con una fecha y hora, y a nivel relativo al comienzo de la clase, considerando el minuto transcurrido en el que se publicó la anotación). Lleva también asociada la actividad en la que se produjo y, por último, el texto conteniendo la anotación.
- ***student_annotations.*** Análogo al anterior pero exclusivo para estudiantes y sólo visible por ellos, son anotaciones privadas.
- ***studentList.*** Lista de estudiantes conectados a la clase, con nombre e identificador. El nombre sirve para poder mostrar una lista de usuarios conectados, y el identificador es necesario para realizar acciones sobre ellos de manera programática. Por ejemplo, si un profesor selecciona a un alumno y le pide intervenir, a nivel interno se hace un ping a su ID. Es ahí donde el alumno comprueba que la ID le corresponde a él, y en caso de cumplirse recibe la notificación.

Se detalla en la *Figura 8* una captura de la base de datos con la aplicación en funcionamiento, y varias clases añadidas. Se pueden observar varios de los detalles mencionados a lo largo de la sección. Existen únicamente dos colecciones: *classrooms* y *scenarios*, y dentro de la colección *scenarios* se pueden apreciar tres sesiones de clase inicializadas, cuyo nombre coincide con su PIN. Dentro del documento de una clase, se aprecia un extracto de los campos ya expuestos en este capítulo.

En el Anexo D se hace un breve análisis del funcionamiento de las actualizaciones en tiempo real, sincronizando los datos disponibles en *Cloud Firestore*, y utilizando *Angular* y una de sus librerías para implementar este proceso de manera programática.

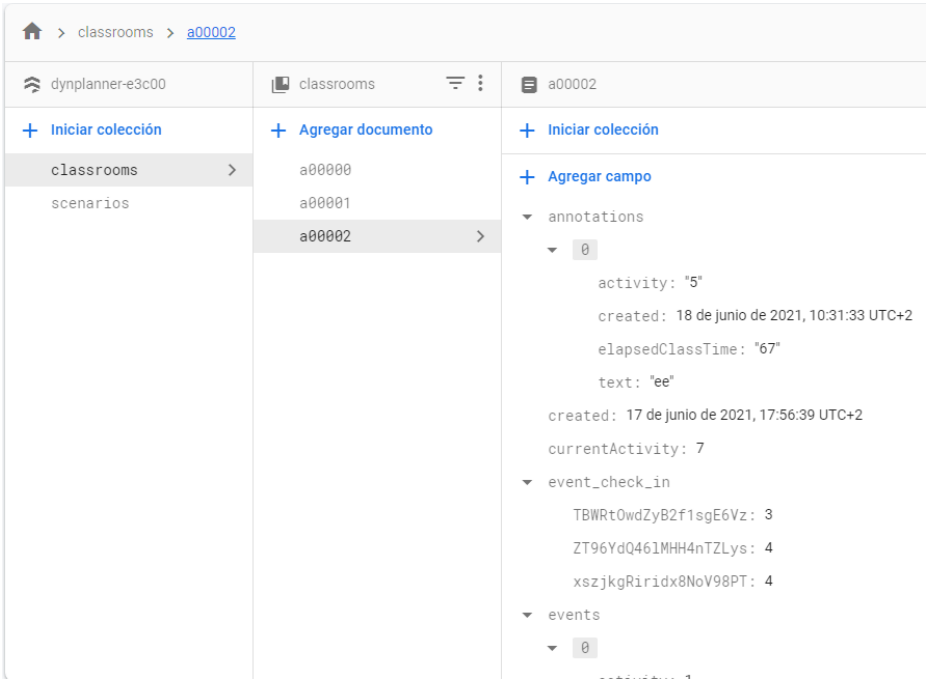

Figura 8: Captura de la base de datos Firestore en funcionamiento.

5 Resultados

Se recomienda al lector la previa lectura sobre la creación y especificación planificaciones de clase de *LePlanner* (Anexo A), pues serán las que se importarán a la aplicación, así como la especificación de los modelos de escenario y sesión de clase sección 4.3, con el fin de comprender con profundidad las distintas partes que conforman la aplicación y su funcionalidad.

Las pruebas son realizadas en la versión desplegada y disponible en Internet, ya convertida a PWA, y por tanto lista para ser vista por cualquier dispositivo independientemente de su tamaño o sistema operativo. El proceso de despliegue de PWA es automático con *Angular*, pues basta con añadir el paquete de *Angular @angular/pwa* al proyecto previamente a su despliegue.

Además, *Firebase* también tiene entre sus servicios *Firebase Hosting* [12], que tiene soporte para PWAs de Angular. Dicho soporte es tan sencillo que sólo es necesario un comando para desplegar la aplicación en *Firebase Hosting* [13], devolviendo como respuesta la URL en la que *Firebase* ha alojado la aplicación. Dicha URL es accesible por cualquier usuario con acceso a internet.

A continuación, se analizarán las distintas vistas de la aplicación en funcionamiento. Por el bien del desarrollo de la sección, se ha creado de antemano un escenario de aprendizaje en *LePlanner*, que será la planificación que se añadirá posteriormente a la aplicación para trabajar sobre ella. Este escenario coincide con el usado para explicar el funcionamiento de LePlanner en el Anexo A.

En la ventana inicial se ofrece la posibilidad a profesores de crear una sesión nueva de aula, o bien, independientemente de si uno es profesor o alumno, entrar a un aula existente (*Figura 9*)

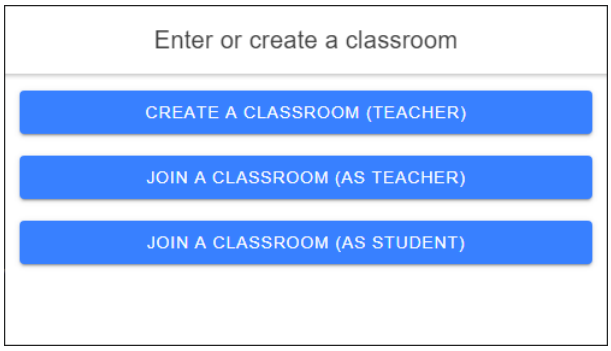

Figura 9: Menú de bienvenida de la aplicación

Si se elige la opción de crear un aula, se piden los datos fundamentales: Nombre de profesor, PIN identificador del aula, contraseña de profesor, y, lo más importante, ID del escenario de *LePlanner* del cuál extraer la planificación de clase (*Figura 10*)

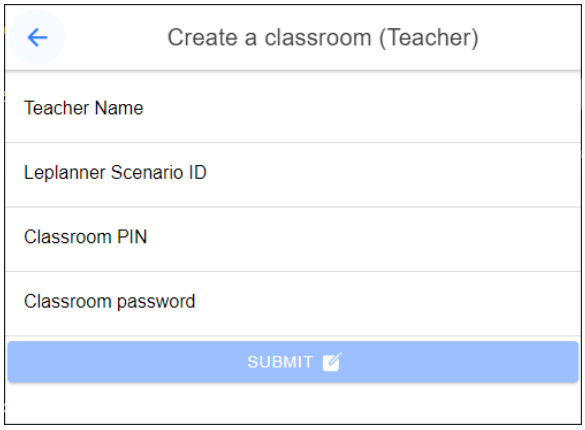

Figura 10: Ventana de creación de una nueva clase (Para profesores)

Si, por otro lado, se elige la opción de entrar a un aula existente, los datos pedidos cambian según el rol del usuario (*Figura 11*):

- Profesor: PIN identificativo del aula y contraseña de profesor
- Alumno: Nombre y PIN identificativo del aula

Cabe mencionar que están implementadas todo tipo de comprobaciones, de manera que si, por ejemplo, el PIN no se identifica con ninguna clase existente o la contraseña del profesor no es correcta, salta una alerta y no permite al usuario avanzar a la clase.

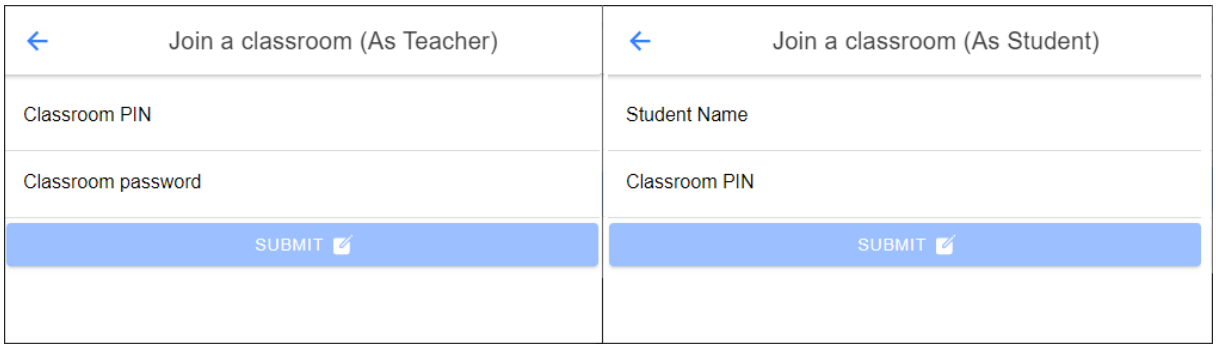

Figura 11: Inicio de sesión a una clase existente (izq. Vista de profesor, dcha. Vista de alumno)

A continuación (*Figura 12*), se presenta la pantalla de bienvenida a una clase existente (en este caso, recién creada), que contiene el nombre del usuario y del profesor, fecha de creación, PIN de acceso,

estado de la clase (no comenzada, comenzada o finalizada), así como la lista de estudiantes conectados y la lista de acciones a realizar. Los profesores tienen la capacidad de poder dar comienzo a la clase.

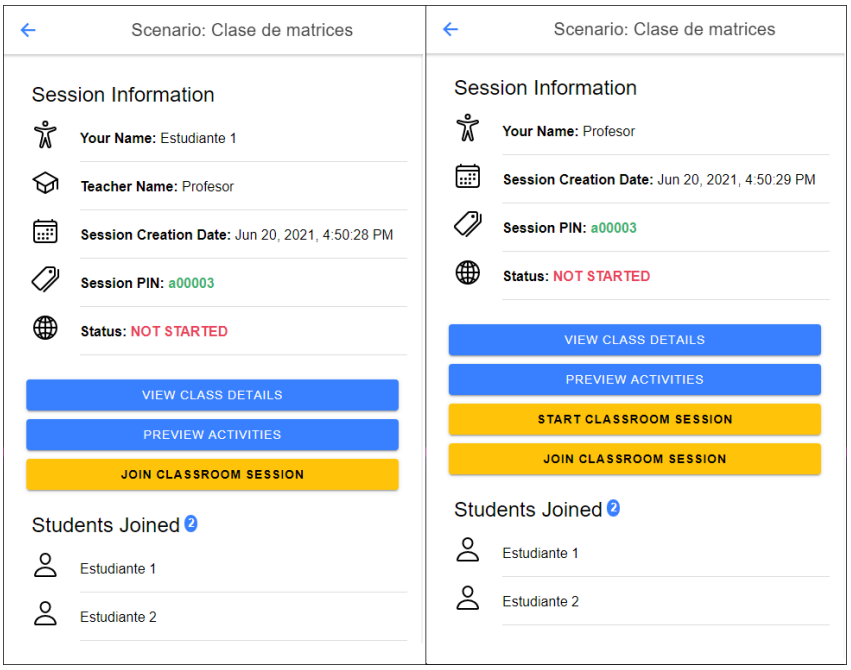

Figura 12: Previsualización de clase en la aplicación (izq. Vista de profesor, dcha. Vista de alumno)

En la sección *View class details* se muestran detalles generales sobre el escenario creado en LePlanner: Nombre, descripción y fecha de creación, autor de la planificación, asignatura y curso, licencia y la URL del escenario. Por otro lado, en la sección *Preview activities* se presenta un *slider* que recorre las actividades de la planificación, permitiendo previsualizar su contenido (*Figura 13*)

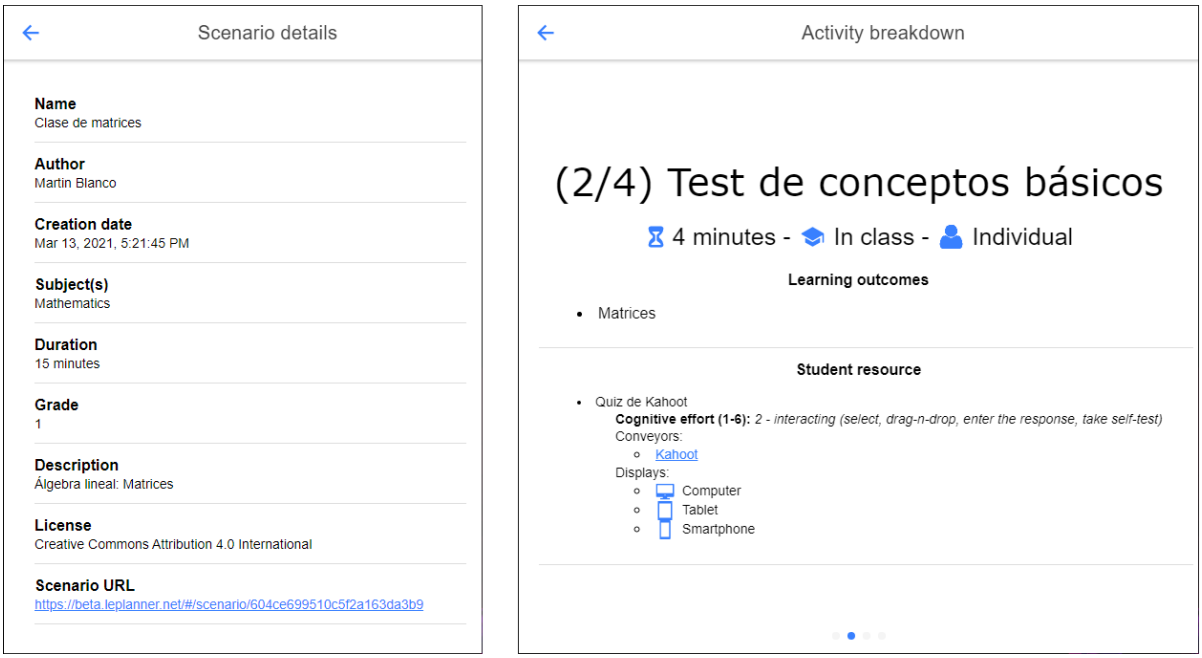

Figura 13: Secciones "Scenario details" y "Preview activities"

Finalmente, la sección *Join classroom session* permite a alumnos y profesores entrar a la sesión de clase. Esta es la vista más importante de la aplicación, en la que se concentra la mayoría de la lógica y funcionalidad. En la *Figura 14* es posible observar la vista completa (correspondiente al profesor):

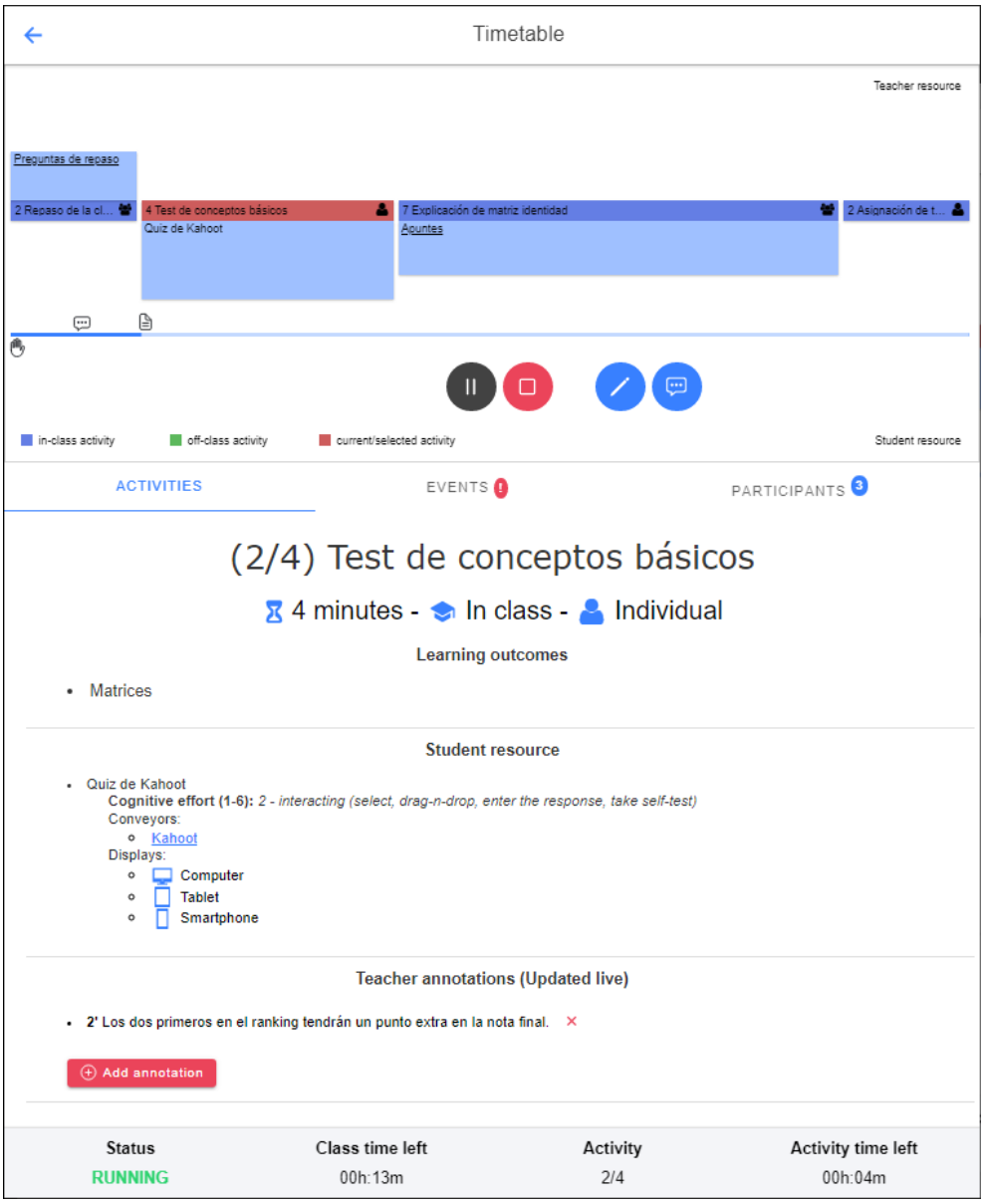

Figura 14: Vista de la sesión de aula por parte del profesor.

Lo primero que llama a la vista es que, en la parte superior, se hace una representación de la planificación de clase de una manera prácticamente idéntica a la observada en *LePlanner*, usando bloques de distintos altos y anchos representando a cada actividad de la clase.

A esta vista se le añade una línea temporal, que permite apreciar de forma gráfica en qué minuto se encuentra la clase, en qué bloque, y qué porcentaje de tiempo transcurrido/restante hay en el bloque actual. Además, la línea temporal registra eventos en el momento de su realización, con un icono distinto para cada tipo de evento. En el ejemplo de la imagen:

- Minuto 1, Actividad 1, Mano. Un alumno levantó la mano para llamar al profesor.
- Minuto 2, Actividad 1, Globo de diálogo. Un profesor solicitó la intervención de uno o varios alumnos.
- Minuto 3, Actividad 2, Hoja de papel. El profesor añadió una anotación en la actividad actual.

Inmediatamente debajo, se presenta el panel de botones. Los profesores tienen, con él, la posibilidad de pausar/reanudar y parar/comenzar la clase. En azul, se presentan los iconos de funcionalidad, como la

posibilidad de añadir anotaciones, invitar a alumnos a participar, o, si el usuario es un alumno, solicitar la participación.

En la *Figura 14* está abierta la pestaña *activities*, en ella se tiene completo acceso a todos los detalles de la actividad, y adicionalmente funciones extra, añadidas como extensión a la aplicación, como la posibilidad de que profesores hagan anotaciones públicas (visibles por todos) o que los alumnos hagan anotaciones privadas (visibles solo por ellos).

Por último, se presenta en el pie de página la información temporal sobre el transcurso de la clase, para que sea siempre visible por el usuario.

Resta presentar la pestaña *events* y *participants* (*Figura 15*). La primera, desglosa cada evento de manera individual, ordenándolo por orden cronológico y según si fue disparado por un alumno o por un profesor. La segunda, presenta la lista de participantes y permite al profesor realizar acciones sobre uno o varios estudiantes.

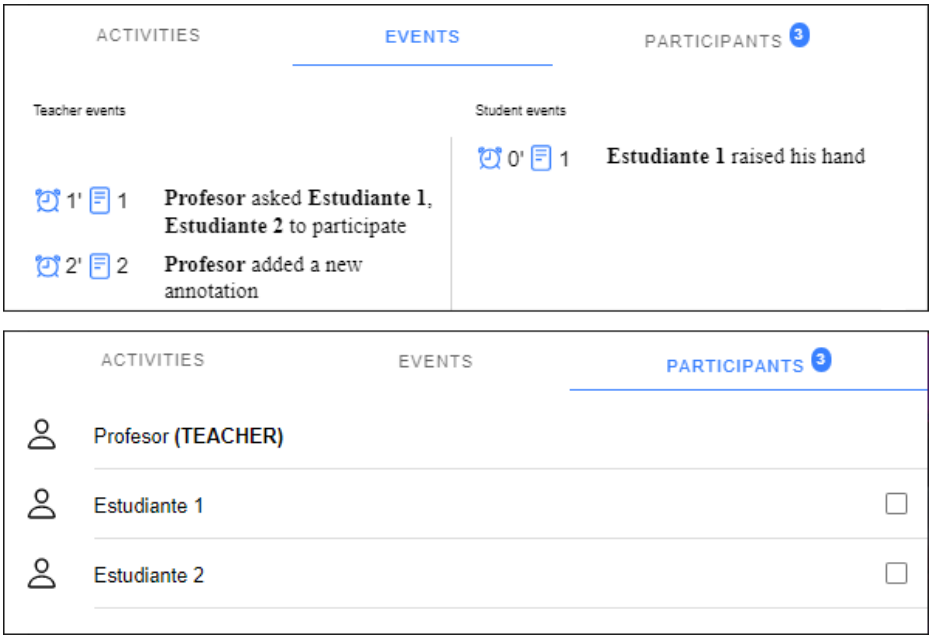

Figura 15: Pestañas 'events' y 'participants' dentro de la sesión de aula

Lo más destacable de la vista de sesión de clase, es que todos los usuarios conectados están constantemente en sincronía, independientemente de si la clase está pausada o no. Esto incluye el estado (pausada, en transcurso o finalizada), el bloque en el que se encuentra, los eventos y las acciones sobre ellos (adición o borrado), y el resto de parámetros que entran en juego durante la sesión.

6 Conclusiones y líneas futuras

Una vez finalizado el proyecto con una aplicación funcional como resultado, se puede concluir que, en general, se han cumplido los objetivos y requisitos propuestos dentro de los plazos esperables con un gran resultado.

La selección de tecnologías ha resultado ser oportuna. En las fases de pruebas tras el despliegue de la aplicación se pudo probar la aplicación en distintos dispositivos (tabletas, *iPhones y* smartphones *Android*, PC) y la experiencia de usuario es muy buena y fluida independientemente del dispositivo, gracias a las bondades de las PWA. Esta era una de las cuestiones importantes ya que existe mucha variabilidad en el tipo de hardware que los alumnos llevan consigo a clase. De esta manera, no se imponen restricciones a la hora de qué dispositivo han de portar consigo a las aulas.

Así mismo, el uso de *Angular* en conjunto con los servicios en la nube de *Firebase* fueron un acierto, ahorrando mucho trabajo al programador. Su gran sinergia, junto a la simplicidad en las bases de datos

de *Firebase* y su soporte moderno a la sincronización de eventos, permiten que un proyecto de este calibre sea totalmente implementable sin impedimentos.

En el global, se tiene como resultado una primera versión totalmente usable, lo cual supone un resultado satisfactorio teniendo en cuenta la relación entre el tiempo disponible para la realización del proyecto y la dificultad que ha supuesto.

Queda principalmente como asignatura pendiente la necesidad de hacer pruebas a mayor escala con usuarios reales para detectar y depurar posibles errores que hayan podido pasar desapercibidos en las pruebas realizadas durante la fase de desarrollo. Así mismo, resulta necesario con el fin de obtener retroalimentación de los usuarios en materia de utilidad y usabilidad de la aplicación en escenarios reales, opiniones sobre el diseño y funcionalidades implementadas, así como ideas sobre funcionalidades que puedan echar en falta.

Para probar su utilidad, se han implementado, en la primera versión de la aplicación, las funcionalidades más básicas, como la petición de intervención a alumnos, el soporte de anotaciones en cada actividad, o la posibilidad de que los propios alumnos soliciten intervenir. Sin embargo, esto no es más de una fracción de lo que podría soportar. Durante su desarrollo han surgido ideas como la implementación de un chat en el aula, soporte de mensajería privada alumno-profesor, soporte de preguntas y respuestas entre distintos participantes, etcétera, que podrían abordarse en el futuro.

7 Referencias

[1] Martínez-Ortiz, I., Pablo Moreno-Ger, Jose Luis Sierra, and Baltasar Fernandez-Manjon. "Educational modeling languages." In Computers and Education, pp. 27-40. Springer, Dordrecht, 2007, Disponible: https://www.researchgate.net/publication/200505307_Educational_Modeling_Languages_A_Conceptual_Introduction_and_a_High-Level_Classification

[2] Koper, Rob, and Bill Olivier. "Representing the learning design of units of learning." Journal of Educational Technology & Society 7, no. 3 (2004): 97-111, Disponible: https://www.researchgate.net/publication/26393724_Representing_the_Learning_Design_of_Units_of_Learning

[3] Caeiro-Rodriguez, Manuel. "Making Teaching and Learning Visible: How Can Learning Designs Be Represented?." Proceedings of the Seventh International Conference on Technological Ecosystems for Enhancing Multiculturality, pp. 265-274. 2019. Disponible: https://dl.acm.org/doi/abs/10.1145/3362789.3362839

[4] Leplanner, "Create and share learning scenarios, making it possible for best teaching practices to spread", Disponible: https://beta.leplanner.net/ y https://leplanner.ee/en/

[5] Anna Plachkova, "PWA vs Native vs Hybrid vs Responsive Website: Full Comparison", 6 Ago 2020, Disponible: https://www.gomage.com/blog/pwa-vs-native-vs-hybrid-vs-responsive-website/

[6] "Ionic Framework Documentation", Disponible: https://ionicframework.com/docs

[7] "Angular Features & Benefits", Disponible: https://angular.io/features

[8] "Documentación para Cloud Functions de Firebase", Disponible: https://firebase.google.com/docs/functions

[9] "Documentación para Cloud Firestore de Firebase", Disponible: https://firebase.google.com/docs/firestore

[10] "Acerca de Node.js ®", Disponible: https://nodejs.org/es/about/

[11] "Modelo de datos de Cloud Firestore", Disponible: https://firebase.google.com/docs/firestore/data-model

[12] "Documentación de Firebase Hosting", Disponible: https://firebase.google.com/docs/hosting

[13] Ionic, "Progressive Web Apps In Angular", Disponible: https://ionicframework.com/docs/angular/pwa

Anexo A. Estado del arte de lenguajes de visualización EML

La plataforma que utilizará este proyecto como modelo de entrada de las planificaciones de clase es *LePlanner*, una plataforma de código abierto que permite crear escenarios de enseñanza/aprendizaje, basada en una representación de diagrama secuencial. Opcionalmente, también existe la posibilidad de compartirlos. A continuación, se hará un análisis completo de su funcionamiento principal.

Se usará, en las figuras de la presente sección, la misma planificación de entrada que servirá de ejemplo para ilustrar el funcionamiento de la aplicación en la sección 5.

Cualquier usuario registrado puede inmediatamente crear una planificación de clase (*Figura 16*). Los parámetros de entrada son simples: Título, descripción, tags identificativos, materias implicadas, aprendizajes/objetivos conseguidos al dominar la clase, curso, duración, y una serie de actividades, que son el elemento más importante y consta, en orden de izquierda a derecha, de:

- Nombre de la actividad
- Duración
- Marca que indica si la actividad se lleva a cabo dentro del aula o fuera (por ejemplo, en casa)
- Organización del grupo que realiza la actividad: Individual, parejas, en bloque, o toda la clase.
- Aprendizajes conseguidos con el dominio de una actividad individual.

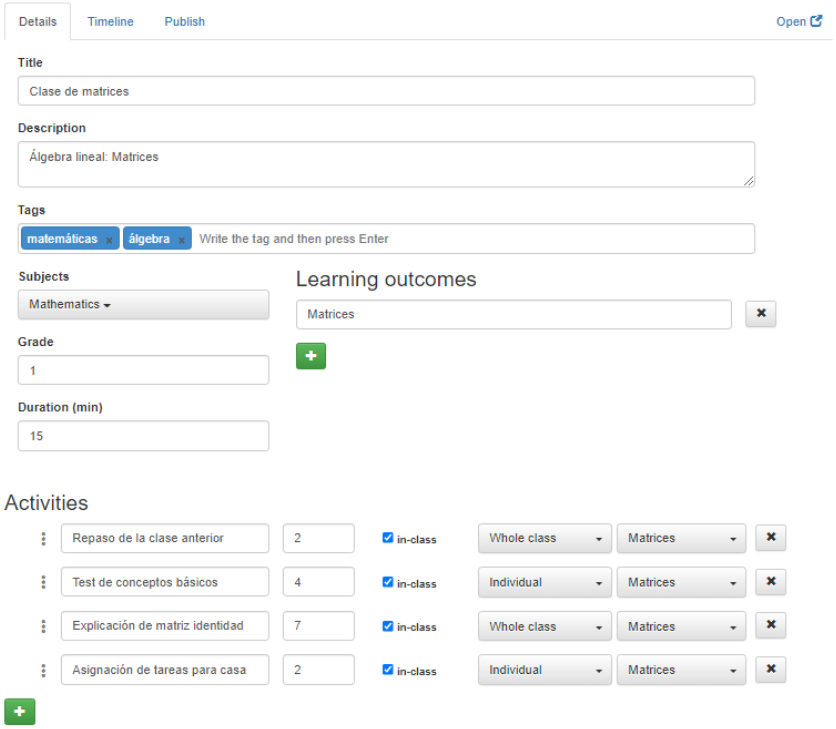

Figura 16: Ventana principal de creación de planificación de clase en LePlanner

Una vez definida la actividad, es posible añadir recursos asociados a dicha actividad (*Figura 17*). Los recursos son elementos destinados al profesor o al alumno (a elección), que representan indicaciones ligadas a dicha actividad. Pueden ser instrucciones de actuación, o notas con hipervínculos que llevan a recursos en la red que serán usados en dicha actividad, como por ejemplo apuntes, transparencias, o vídeos. Cada recurso tiene una serie de parámetros:

- Nombre (y, opcionalmente, URL) de un recurso.

- [Exclusivo para alumnos] Carga/Intensidad cognitiva de la actividad, del 0 (menos exigente) al 6 (más exigente)
- Medios: Recursos digitales o webs que servirán como medio para la realización de la actividad. Si se desea hacer un quiz, se puede indicar *Kahoot!* como medio.
- Pantallas: Dispositivos con los que la actividad es realizable: Ordenadores, móviles, tabletas…

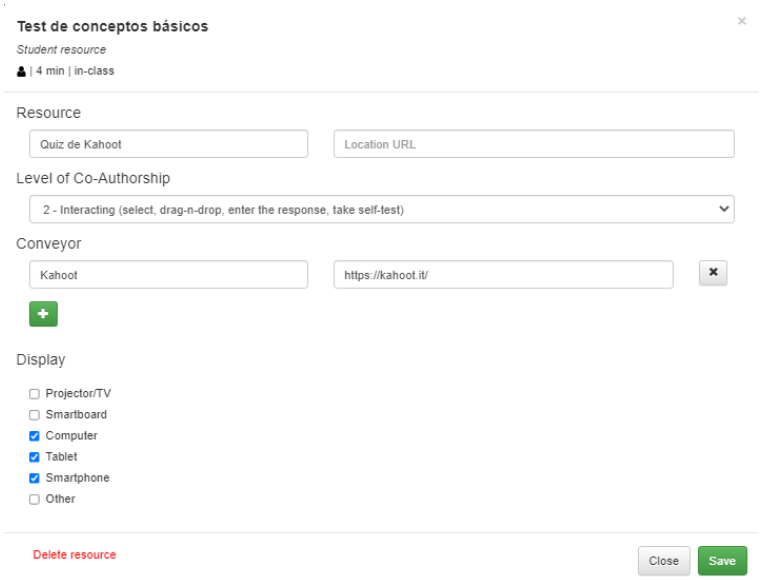

Figura 17: Ventana de adición de un recurso a una actividad en LePlanner

En la *Figura 18* se puede apreciar el resultado final, visible al público, de la planificación de clase creada.

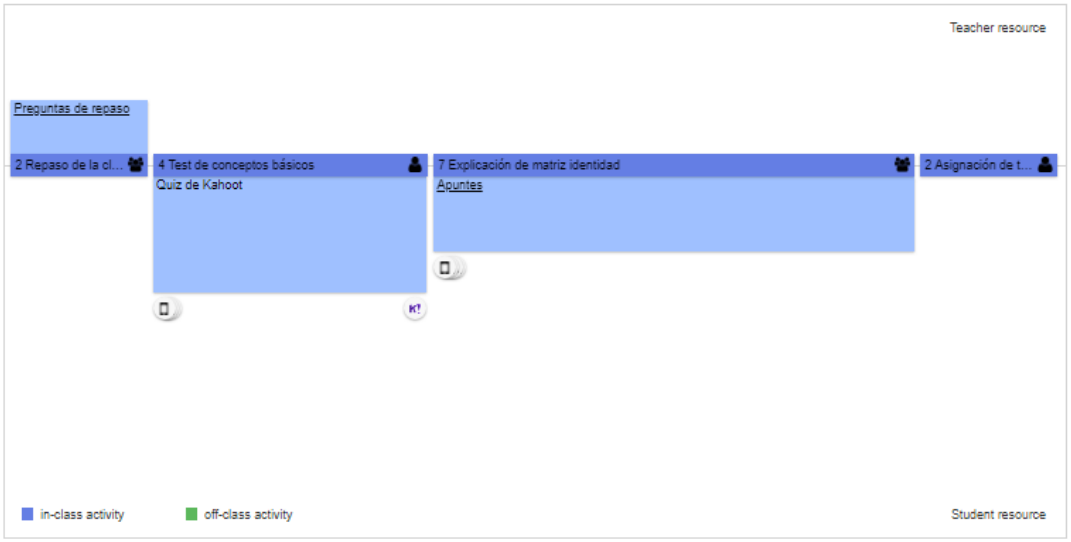

Figura 18: Publicación definitiva de una planificación en LePlanner

Como se puede observar, cada actividad está diferenciada por un bloque distinto, cada uno con características diferentes. Los principales elementos de un bloque (actividad) son:

- **Título y descripción**, y otras anotaciones adicionales.
- **Color:** Verde si la actividad se lleva a cabo fuera de clase o azul si se desarrolla en clase.
- **Duración (en minutos):** Representada a nivel gráfico con la anchura del bloque.
- **Recursos digitales:** En la parte superior, ubicados aquellos recursos destinados al profesor y posibles anotaciones o indicaciones. En la parte inferior, los destinados al alumno. Incluye

enlaces a recursos alojados en la red, como, por ejemplo, bibliográficos. Si el profesor explica un tema, enlazará aquí el recurso correspondiente a los apuntes. También se indican los dispositivos que admite la actividad (globo en la parte izquierda del recurso) e hipervínculos que dirigen a medios que son de utilidad para realizar una tarea (globo en la parte derecha del recurso) Por ejemplo, para crear una encuesta, aquí iría en enlace a *Google Forms*, pues es el medio empleado para crear encuestas.

- **Esfuerzo cognitivo:** La altura del bloque representa una estimación del esfuerzo cognitivo que requiere a un alumno completar la actividad en cuestión.

Por último, resulta de gran importancia destacar que en el momento de la publicación de una planificación de clase en *LePlanner*, la plataforma devuelve al usuario un identificador único, que puede compartir con cualquier persona (o máquina) para la visualización de la planificación. El escenario usado como ejemplo en esta sección (y que se usará como ejemplo también en la sección 5 para enseñar el funcionamiento de la app) tiene como ID *604ce699510c5f2a163da3b9*. En el momento de escribir este texto, la planificación es accesible al público de dos maneras:

1. Visualización directa en la web: https://beta.leplanner.net/#/scenario/604ce699510c5f2a163da3b9
2. Petición directa a la API, que devuelve un objeto JSON con todos los parámetros del escenario: https://beta.leplanner.net/api/scenarios/single/604ce699510c5f2a163da3b9

Se puede observar como en ambos casos se tiene una URL fija seguida del identificador único del escenario, haciendo que todo resulte muy sencillo.

En este trabajo se ha decidido usar el sistema de *Leplanner* como formato de representación de diseños de aprendizaje debido a su innovadora interfaz altamente intuitiva, gráfica y minimalista. La aplicación presente, como ya se ha comentado, tendrá la capacidad de importar estos modelos de datos directamente de *LePlanner* a la base de datos propia de la aplicación, y así trabajar directamente con ellos.

Existen muchos otros sistemas y aplicaciones que implementan distintos sistemas de representación (diagramas de flujo, de secuencia, *ad-hoc*, mapas, tablas, etc.) que buscan implementar *Learning Designs* o maneras de diseñar las experiencias de aprendizaje de los estudiantes a partir de una secuencia de tipos de actividades e interacciones.

A continuación, se describen brevemente algunas de las más relevantes:

- **Learning Design Visual Sequence (LDVS):** Representación mediante un diagrama secuencial que relaciona recursos (como libros de texto o documentos), actividades (aquello que debe hacer el estudiante con los recursos para conseguir los objetivos de aprendizaje) y herramientas de apoyo complementarias, que pueden ser útiles para el estudiante. También incluye un periodo de tiempo para el *Learning Design* y los resultados de aprendizaje previstos. En la *Figura 19* se muestra un ejemplo: Los cuadrados representan tareas o actividades de aprendizaje; los triángulos representan recursos de aprendizaje; y los círculos se usan para representar herramientas de apoyo.

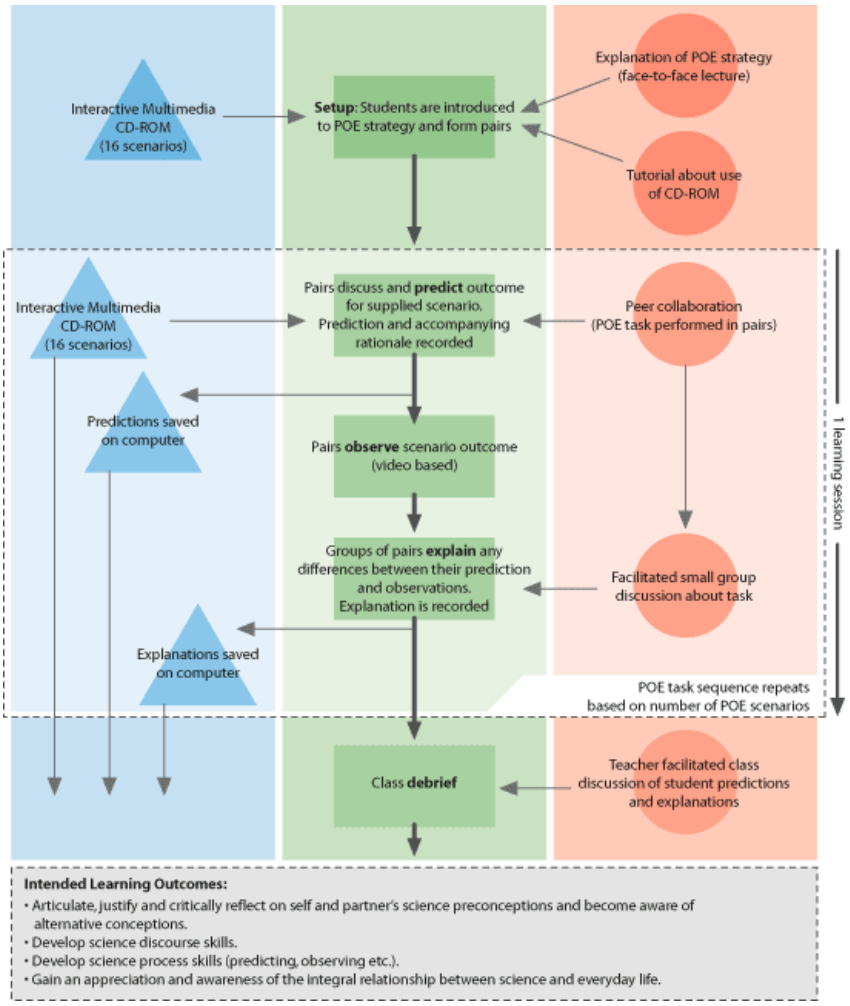

Figura 19: Ejemplo de representación gráfica utilizando LDVS.

- **Learning Activity Management System (LAMS):** Popular Sistema de *Learning Design* de Código abierto, basado en una representación de diagrama de flujo, para administrar y llevar a cabo actividades de enseñanza colaborativa online. LAMS proporciona a profesores un entorno de creación visual intuitivo para crear secuencias de actividades de aprendizaje. En la *Figura 20* se muestra un ejemplo.

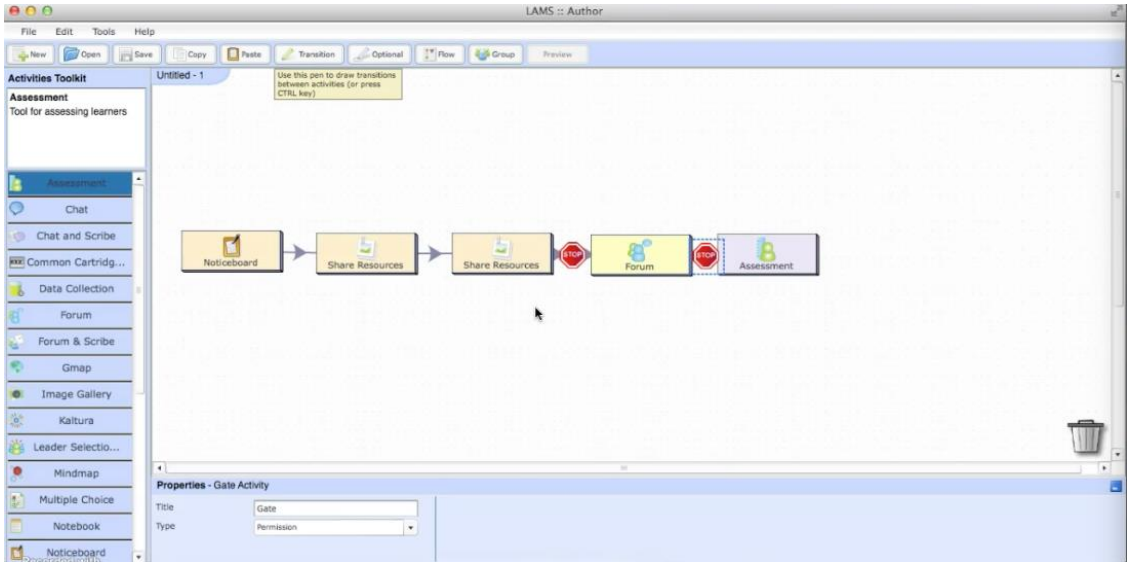

Figura 20: Captura de un Learning Design en LAMS

- **Méthode d'ingénierie d'un système d'apprentissage (MISA):** Metodología de *Learning Design*, que propone como representación gráfica un tipo de mapa mental. Relaciona los distintos recursos disponibles y las actividades con profesores y alumnos. En la *Figura 21* se muestra un ejemplo.

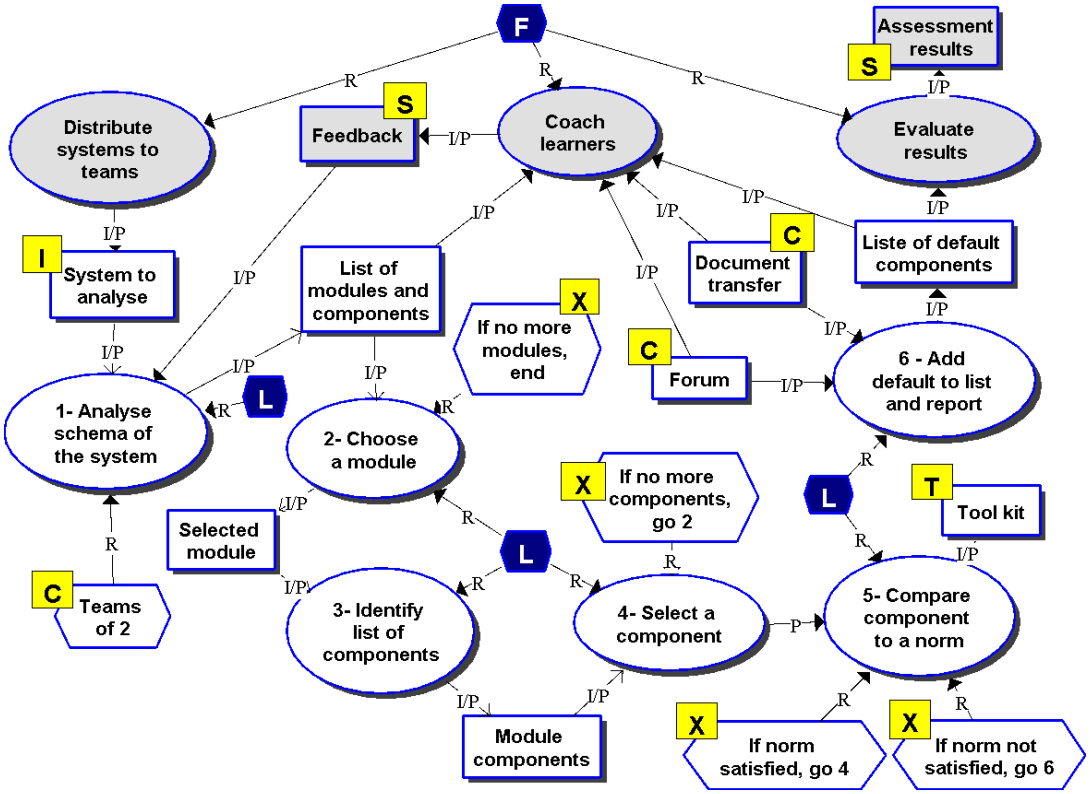

Figura 21: Escenario Educativo usando el método MISA/MOT

- **Educational Environment Modeling Language (E2ML):** Lenguaje de modelado específicamente desarrollado para el diseño de entornos educativos. Modela los objetivos, requisitos y el diseño de las actividades de enseñanza y aprendizaje. Sus principales partes son un mapeo visual (un caso específico de mapeo conceptual) y una representación visual, tipo UML, de las actividades de aprendizaje. En la *Figura 22* se muestra un ejemplo.

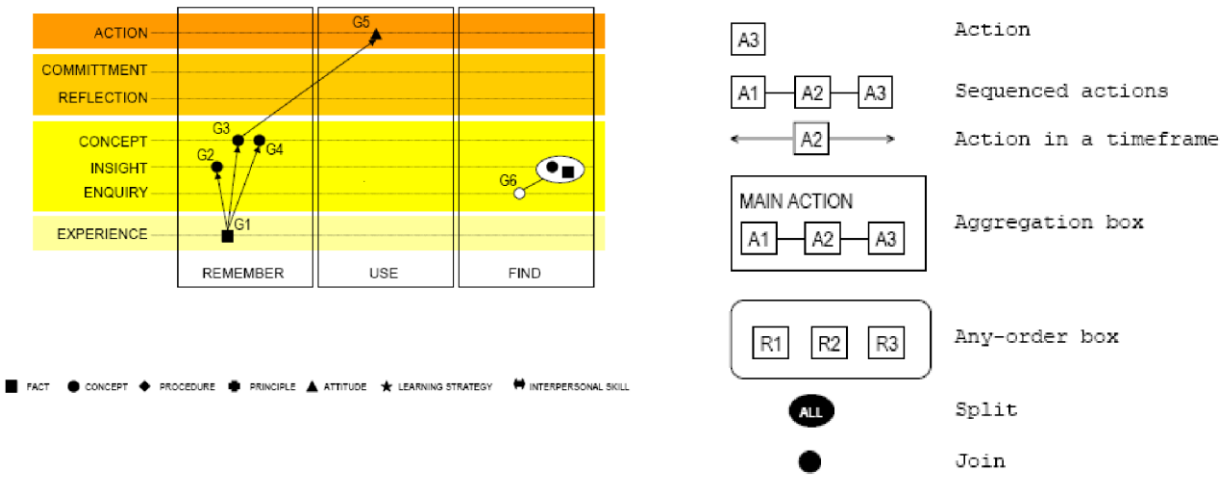

Figura 22: Representación de objetivos de aprendizaje utilizando E2ML (izquierda). Elementos E2ML utilizados en las representaciones (derecha)

Anexo B. Captura de requisitos

Se adjunta a continuación la captura de requisitos realizada en una primera instancia antes de comenzar la programación de la aplicación, consistente en un conjunto de casos de uso e historias de usuario. Nótese que por decisiones posteriores algunos detalles han podido ser levemente modificados, y muchos de ellos han sido añadidos posteriormente a la declaración de los requisitos. De todos modos, y en general, no resultó haber mucha diferencia entre los requisitos citados y la implementación final:

1. Como usuario (profesor y alumno) quiero recibir indicación en tiempo real del tiempo transcurrido de la sesión.
2. Como usuario (profesor y alumno) quiero recibir indicación en tiempo real de la actividad en la que se está actualmente.

Caso de uso	Indicación del tiempo transcurrido Y del bloque de actividad actual en tiempo real, junto a su respectiva información
Actor principal	Profesor y alumnos
Personal involucrado	Profesor y alumnos
Precondiciones	Sesión con bloques de actividad definidos, aún no empezada. Temporizador inicializado a cero. Indicador del bloque de actividad inicializado al primero
Postcondiciones	El temporizador comienza a correr de manera continua hasta el final de la sesión, pasando e indicando todos los bloques de actividad en su momento correspondiente
Escenario principal/Flujo principal	1. El reloj se inicializa con el tiempo total de la clase, sumando los tiempos individuales de cada bloque para obtener el tiempo total y el indicador de bloques se inicializa con el primer bloque de actividad de la clase. 2. El profesor pulsa el botón *Play* para comenzar la clase. 3. El reloj comienza a correr. 4. A lo largo del paso del tiempo, el indicador de bloques se sitúa en el bloque correspondiente a la cronología y muestra su información. 5. Cuando alcanza el tiempo total, indica que la clase ha llegado a su fin, y deja de correr. El indicador de bloques se queda en el último bloque de actividad de la sesión.
Extensiones / Flujos alternativos	No hay.

3. Como profesor quiero comenzar una sesión de aula en directo.

Caso de uso	Comenzar una sesión de aula en directo
Actor principal	Profesor
Personal involucrado	Profesor y alumnos
Precondiciones	Sesión con bloques de actividad definidos, aún no empezada.
Postcondiciones	La sesión comienza a correr.
Escenario principal/Flujo principal	1. El profesor pulsa en el botón *Play* y la clase comienza. 2. Una vez el tiempo total de la sesión se alcanza, la sesión se para (finaliza).
Extensiones / Flujos alternativos	Si en el paso 1. la sesión no tiene ningún bloque de actividad definido, lanza un error "Sesión no definida" y no comienza.

4. Como profesor quiero poder invitar a los alumnos a la sesión mediante un código de acceso público.

Caso de uso	Invitar a los alumnos a la sesión mediante un código de acceso público
Actor principal	Profesor
Personal involucrado	Profesor y alumnos
Precondiciones	La sesión tiene que existir y tiene que tener asociado un código identificativo
Postcondiciones	Cualquier usuario con el código puede acceder a la sesión
Escenario principal/Flujo principal	1. El profesor crea una sesión y, en el momento de su creación, define un código de acceso público 2. El alumno puede posteriormente entrar a la sala indicando su nombre y el código de acceso público
Extensiones / Flujos alternativos	No hay

5. Como profesor quiero añadir anotaciones (comentarios), indicaciones y avisos en el bloque de actividad en el que se encuentra la sesión.

Caso de uso	Añadir anotaciones (comentarios), indicaciones y avisos en el bloque actual
Actor principal	Profesor

Personal involucrado	Profesor y alumnos
Precondiciones	La sesión tiene que estar en ejecución
Postcondiciones	Las anotaciones/indicaciones/avisos se añaden en la línea temporal, en el momento exacto de la línea cronológica en la que se encuentra, y tanto el profesor como todos los alumnos con acceso a la sesión pueden acceder a la anotación añadida.
Escenario principal/Flujo principal	1. El profesor pulsa el botón de "Añadir anotación" 2. Se abre un editor de texto y añade el texto deseado. 3. Al finalizar, pulsa "Enviar" para publicar la anotación. 4. La anotación se aloja como un evento en la línea temporal de la sesión actual. 5. Los alumnos reciben una notificación con el aviso de la nueva anotación añadida. 6. Tanto el profesor como todos los alumnos tienen acceso a la anotación.
Extensiones / Flujos alternativos	En el paso 3., si el campo de texto está vacío, no se envía ninguna anotación y se lanza el error "No se ha escrito ningún texto"

6. Como profesor quiero pedir la intervención de un alumno en particular.

Caso de uso	Solicitar la intervención de un alumno
Actor principal	Profesor
Personal involucrado	Profesor y alumnos
Precondiciones	La sesión tiene que estar en ejecución y el alumno tiene que estar admitido en la sesión.
Postcondiciones	El alumno recibe la petición de intervención del profesor y esta se almacena de manera pública en la línea temporal.
Escenario principal/Flujo principal	1. El profesor accede a la lista de alumnos en la sesión y pulsa en el botón "Solicitar intervención" 2. El alumno recibe la petición de manera privada a través de una notificación. 3. La petición se almacena de manera pública como un evento en la línea temporal de la sesión.
Extensiones / Flujos alternativos	No hay.

7. Como alumno quiero añadir anotaciones privadas en las distintas actividades.

Caso de uso	Enviar anotaciones
Actor principal	Alumno
Personal involucrado	Profesor y alumnos
Precondiciones	La sesión tiene que estar en ejecución y el alumno tiene que estar dentro de la sesión.
Postcondiciones	El alumno envía una anotación durante la sesión, que se almacena en la actividad actual
Escenario principal/Flujo principal	1. El alumno pulsa en el botón "Enviar anotación" 2. Se abre un editor de texto y añade el texto deseado. 3. Al finalizar, pulsa "Enviar" para enviar la anotación a la actividad actual
Extensiones / Flujos alternativos	En el paso 3., si el campo de texto está vacío, no se envía ninguna anotación y se lanza el error "No se ha escrito ningún texto"

8. Como alumno quiero solicitar turno para hacer una intervención o pregunta de manera pública.

Caso de uso	Solicitar turno para intervenir públicamente
Actor principal	Alumno
Personal involucrado	Profesor y alumnos
Precondiciones	La sesión tiene que estar en ejecución y el alumno tiene que estar admitido en la sesión.
Postcondiciones	El alumno envía una petición de intervención al profesor
Escenario principal/Flujo principal	1. El alumno pulsa en el botón "Solicitar intervención" 2. El profesor recibe la petición (notificación) 3. La petición se almacena de manera pública como un evento en la línea temporal de la sesión y el alumno procede a intervenir.
Extensiones / Flujos alternativos	No hay.

Anexo C. Estado del arte para la realización de aplicaciones en dispositivos móviles

En esta sección se hará una reseña de las distintas tecnologías usadas en la actualidad para desplegar una aplicación, y se seleccionará aquella que ofrezca más ventajas para este caso específico, indicando a continuación los entornos y lenguajes utilizados en su implementación.

Figura 23: Distintas soluciones para la implementación de una aplicación

En la *Figura 23* se observan las principales tecnologías usadas en la actualidad para desplegar una aplicación y los dispositivos a los que se dirige cada una:

- **Aplicación nativa:** Forma tradicional, más utilizada, de crear aplicaciones. Con dicha tecnología se debe crear un código base distinto (con un lenguaje de programación también distinto) para cada plataforma final en la que se desea desplegar la aplicación, aumentando así la dificultad y el tiempo de desarrollo. Sin embargo, al ser especializada para cada plataforma, su rendimiento y la experiencia de usuario lograble con ella es muy alta.

  La mayoría de apps que existen en el mercado, y que usamos en el día a día, son nativas: *Gmail*, *WhatsApp*, *Twitter*, *Facebook*… Tienen un código base para *Android* y otro distinto para *iOS* (Sistema operativo actual de *iPhone*). A nivel de usuario, también se pueden apreciar interfaces e incluso funciones distintas según si usamos las apps mencionadas en *Android* o en *iOS*.

  Android Studio es el entorno de desarrollo oficial para la plataforma *Android*. Desde 2019, *Kotlin* es el lenguaje preferido de *Google* para el desarrollo de aplicaciones *Android*. Sin embargo, *Android Studio* también admite lenguajes como *Java* y *C++*.

  XCode es el entorno de desarrollo oficial para la plataforma *iOS*, de *Apple*. Las aplicaciones para estos dispositivos son desarrolladas con el lenguaje *Swift*, un lenguaje diseñado por *Apple* para crear apps de *iOS* y *Mac*.

- **Página web *responsive*:** Aquellas webs diseñadas para responder bien en cualquier pantalla, ya sea grande, como la de un ordenador, o pequeña, como la de un teléfono inteligente. Busca brindar una experiencia de usuario perfecta independientemente del dispositivo utilizado.

Por ejemplo, *Moodle* (La famosa plataforma de campus virtual) es una aplicación web de código abierto *responsive*. Si un alumno accede con exactamente el mismo hipervínculo a su curso en un teléfono móvil y en un ordenador, se observará un diseño distinto, adaptado a cada dispositivo.

Bootstrap es uno de los *frameworks* más populares dirigidos al diseño de webs adaptables. Contiene plantillas basadas en CSS, y, opcionalmente, *JavaScript*, con tipografía, formularios, botones y componentes de navegación personalizados, entre otros.

- **Aplicación híbrida:** Son aplicaciones que funcionan en todos los dispositivos móviles, independientemente de su plataforma. Comparten el mismo código base (generalmente HTML5, CSS y *JavaScript* o *TypeScript*) permitiendo acelerar el tiempo de desarrollo y ahorrar costes. En general, como desventaja, la experiencia de usuario no es óptima y no pueden presumir de tener un alto rendimiento.

  Por ejemplo, *Moodle App* (aplicación para móviles de *Moodle*), es una aplicación híbrida.

  Ionic es un kit de desarrollo de software (SDK) de código abierto para el desarrollo de aplicaciones móviles híbridas. Está construido a partir de *Apache Cordova* (antes conocido como *PhoneGap*), un *framework* para programar aplicaciones híbridas con CSS3, HTML5 y *JavaScript*. *Ionic* permite al usuario elegir entre varios *frameworks* para el desarrollo de la app: *Angular*, *React* o *Vue.js*.

  Flutter es la apuesta de Google como SDK para el desarrollo de apps híbridas, también de código abierto, con características similares a *Ionic*. Utiliza un lenguaje orientado a objetos propio de Google llamado *Dart*.

  React Native es un framework de código abierto creado por *Facebook*, permitiendo el desarrollo de apps nativas e híbridas mediante *React*, una librería muy popular de *JavaScript* que sirve para construir interfaces de usuario y componentes para éstos.

- **Aplicación Web Progresiva (PWA):** Combina las mejores características de las aplicaciones móviles y páginas web, pues ofrecen el mismo nivel de experiencia y rendimiento que las aplicaciones nativas, y a su vez pueden ser instaladas y ser correctamente visualizadas en cualquier dispositivo móvil. Son accesibles desde cualquier navegador, por tanto, se pueden visualizar en cualquier dispositivo con acceso a internet, ya sean móviles u ordenadores.

  Por ejemplo, la versión web de la plataforma de música en streaming más popular, *Spotify*, es una PWA, que permite a sus usuarios reproducir contenidos sin necesidad de tener la app nativa instalada, ahorrando así espacio en el dispositivo, entre otras ventajas.

  Ionic, además de ser el framework más popular en el mundo de las aplicaciones híbridas, está totalmente adaptado al desarrollo y lanzamiento de PWAs.

  Además de *Ionic*, las herramientas más utilizadas para el desarrollo de PWA son la biblioteca de *JavaScript React.js*, y los frameworks de *JavaScript Angular.js* y *Vue.js*.

En la siguiente figura (*Figura 24*) se presenta una comparativa más exhaustiva de las distintas tecnologías:

CARACTERÍSTICAS	APLICACIÓN NATIVA	APLICACIÓN HÍBRIDA	WEB RESPONSIVE	APLICACIÓN WEB PROGRESIVA
Compatibilidad multiplataforma	NO	SÍ	SÍ	SÍ
Accesible mediante URL	NO	NO	SÍ	SÍ
Indexada por motores de búsqueda	NO	NO	SÍ	SÍ
Instalable	SÍ	SÍ	NO	SÍ
Actualizaciones no percibidas por el usuario	NO	NO	SÍ	SÍ
Modo offline	NO	NO	NO	SÍ
Notificaciones Push	SÍ	SÍ	NO	SÍ
Desarrollo de bajo coste y eficiente en tiempo	NO	NO	SÍ	SÍ
Distribución	App store	App store	Web	Web
Requiere más espacio	SÍ	SÍ	NO	NO
Exigente en hardware	SÍ	SÍ	NO	NO
Facilidad en compartir contenido	NO	NO	SÍ	SÍ

Figura 24: Tabla comparativa de las diferentes tecnologías de despliegue de una aplicación

En este caso particular en el que se busca una aplicación prácticamente universal, accesible por cualquier profesor y por cualquier alumno, resulta indispensable buscar una tecnología flexible e independiente de la plataforma desde la que se accede. Además, resultan de interés características como el alto rendimiento, la ligereza y la baja exigencia a nivel de hardware, pues esto brinda más accesibilidad, especialmente en aquellas ubicaciones con prestaciones más bajas.

Una *web responsive* estaría cerca de reunir todos los requisitos, pero en este aspecto la PWA sale ganando ligeramente por ventajas como el soporte y la fácil integración de *notificaciones push*, que, en una aplicación de este tipo, es una ventaja a la que se le puede sacar mucho partido.

Es por esta serie de motivos y grandes ventajas que **se ha decidido usar la tecnología de Aplicación Web Progresiva (PWA) para el desarrollo de este proyecto.**

Anexo D. Actualizaciones en tiempo real: Angular, FireStore, AngularFire

Uno de los elementos más importantes de este trabajo, si no el que más, es la sincronización de los usuarios con los distintos eventos que ocurren a tiempo real. Este proceso se realiza de una manera extremadamente fluida, de tal manera que el cliente observa cualquier cambio sin necesidad de su interacción, como el refresco o recarga de la ventana que tiene abierta.

Esto es posible gracias a uno de los elementos diferenciadores de *Angular*, los ***Observables***. En Angular es posible usar almacenes de datos y, una vez dichos almacenes son modificados, recibir automáticamente sus cambios, sin que se tenga que programar a mano ese tránsito de información. Hay 3 componentes fundamentales que componen este sistema:

- ***Observable***: Aquello que se quiere observar. Permite al observador suscribirse a eventos que permiten realizar una acción en el momento que se realiza un cambio en lo que se está observando. Ejemplo en la implementación de esta aplicación: Observación de anotaciones en un bloque de actividad.
- ***Observador***: Actor que se dedica a observar. Se implementa a partir de una colección de funciones (*callback*) que permiten escuchar eventos o valores emitidos por un *observable*. Los *callbacks* harán posible especificar código a ejecutar frente a un dato en el flujo, por un error, o por el simple fin del flujo. Ejemplo en la implementación de esta aplicación: El alumno, pendiente de las anotaciones que el profesor publique en una actividad.
- ***Sujeto***: Emisor de eventos, que crea el flujo en el momento que el observable sufre cambios y que consumirán los observadores. En el caso de esta aplicación, es la API de *Firebase* quien, cada vez que hay un cambio, emite este flujo, notificando a los observadores.

Como se acaba de mencionar, *Firebase* facilita en gran medida esta funcionalidad, gracias a su funcionalidad de despliegue de actualizaciones en tiempo real, de manera que cada vez que ocurre un cambio en la base de datos *Firestore* (Adición/edición/borrado de un campo, documento o colección), inmediatamente dispara un evento, y los observadores suscritos a él son notificados de manera inmediata y pueden así actuar en consecuencia.

Juntando estos dos conceptos entra en acción *AngularFire*, que es la librería oficial de *Firebase* para *Angular*. Esta librería permite la interacción con todas las funciones de *Firebase*, incluida la interactividad con la base de datos, de una manera notablemente intuitiva y simple.

Con su método *valueChanges().subscribe()* se declara que el usuario comienza a observar un elemento, y, en caso de que haya un cambio, recibe los datos actualizados en tiempo real, prácticamente de manera inmediata tras el cambio.

A continuación (*Figura 25*), se presenta un ejemplo en el que el cliente se suscribe a un documento (indicando su PIN) de la colección *classrooms*, de manera que, cada vez que hay un cambio en la sesión de clase, el observador recibe el documento actualizado. Esto permite que todos los clientes puedan estar sincronizados constantemente a los distintos cambios: Reloj de la clase, bloque de actividad en la que se encuentra, anotaciones y eventos, etcétera. En este caso se decide almacenar la fecha de comienzo de la clase en una variable global cada vez que hay un cambio en el documento.

```
    this.classroomSubscription = this.firestore.collection('classrooms').doc(this
.pin).valueChanges().subscribe(data => {
      this.startTime = data['startTime'];
    });
```

Figura 25: Ejemplo de subscripción a un Observable (Cambio en un documento en Firestore)

Otro de los elementos importantes de esta librería, de obligada mención por su uso en varias partes de este proyecto, es *get()*, que implementa la posibilidad de hacer una consulta (sin suscripción) para obtener datos en una única instancia. Esto puede resultar útil, por ejemplo, para la inicialización de variables: Si se necesita un dato que se sabe que no va a ser cambiado posteriormente, es ineficiente suscribirse a él y estar pendiente de unos supuestos cambios que nunca van a ocurrir. En el ejemplo siguiente (*Figura 26*) se obtiene el valor de la fecha de comienzo de la clase y se almacena en una variable global.

```
this.firestore.firestore.doc(`classrooms/${this.pin}`)
.get()
.then((doc) => {
  this.startTime = doc.data()['startTime'].toDate();
});
```

Figura 26: Obtención de datos de un documento en Firestore