Universitat Oberta de Catalunya

Análisis y desarrollo de una plataforma de creación y explotación de aplicaciones interactivas multijugador

Autor: Manuel Lillo Vera
Tutor: Jordi Ustrell Garrigos
Profesor: Ferran Adell Español

Grado de Multimedia
Ingeniería web

06/06/2022

Créditos/Copyright

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Análisis y desarrollo de una plataforma de creación y explotación de aplicaciones interactivas multijugador*
Nombre del autor:	*Manuel Lillo Vera*
Nombre del colaborador/a docente :	*Jordi Ustrell Garrigos*
Nombre del PRA:	*Ferran Adell Español*
Fecha de entrega (mm/aaaa):	*06/2022*
Titulación o programa:	*Grado multimedia*
Área del Trabajo Final:	*Ingeniería web*
Idioma del trabajo:	*Español*
Palabras clave	*navegador, videojuego, multijugador, editor*

Resumen del Trabajo:

Los videojuegos para navegador representan la hibridación perfecta entre comunicación multimedia e ingeniería web, combinando arte y tecnología en un producto atractivo y altamente rentable. Sin embargo, la cantidad de disciplinas involucradas hacen que su desarrollo sea muy complicado para un único individuo. Partiendo de esta idea, el proyecto explora la posibilidad de crear una plataforma que permita a cualquiera con una idea publicar y explotar un videojuego con arquitectura web, sin necesidad de contar con conocimientos técnicos. Desde el punto de vista tecnológico, se basará en una pila MEVN (MongoDB, Express, Vue.js, Node.js). Desde el punto de vista conceptual, se fundamentará en dos pilares. Por un lado, un sistema de reglas configurable que permitirá modelar el desarrollo de la actividad de los jugadores. Por otro, la capacidad de personalizar los elementos multimedia (texto, imagen y sonido) asociados a todos y cada uno de los elementos involucrados en el juego, de manera que el autor pueda adaptar el aspecto gráfico a la temática elegida.

Abstract:

Browser-based games depict the perfect hybridization between multimedia communication and web engineering, blending art and technology in an engaging, highly profitable product. However, the large number of disciplines involved makes their development too complex for a single individual. Starting from this idea, the project probes the possibility of creating a platform that allows for anyone with an idea, to publish and operate a videogame with web architecture. From a technological point of view, it will be based on a MEVN stack (MongoDB, Express, Vue.js, Node.js). From the conceptual point of view, it will be based on two pillars. On the one hand,a parameterizable rule-based system that will allow to model player's activity development. On the other, the ability to customize all multimedia elements (text, image and sound) associated with each and all one of the elements involved on the game.

Notaciones y Convenciones

# 1. Titular, nivel 1	Arial 20pt, negrita
## 1.1 Titular, nivel 2	Arial 13pt, negrita
### 1.1.1 Titular, nivel 3	Arial 13pt, negrita
1.1.1.1 Titular, nivel 4	Arial 10pt, negrita, cursiva
Párrafo	Arial 10pt
`Referencia a código fuente`	Courier new 10pt

Índice

Figuras y tablas

Índice de figuras

Índice de tablas

1 Introducción

1.1 Introducción

Durante el periodo en que se tuvo que decidir la temática de este trabajo, se valoró la idea de desarrollar un videojuego multijugador completo que se pudiera ejecutar en el navegador del usuario. Este tipo de aplicaciones son un buen ejemplo de gran parte de los conocimientos adquiridos a lo largo del grado, por lo que a priori parecería una elección muy acertada. Sin embargo, el esfuerzo necesario para generar el contenido multimedia podría hacer inviable un proyecto de este tipo.

Así pues, se llevó a cabo una búsqueda de modelos de juego simplificados con desarrollos minimalistas, donde cabe destacar títulos como Ogame[1], Bitefight[2], Travian[3] o Grepolis[4]. Todos ellos presentan un alto nivel de organización frente a un bajo nivel de complejidad gráfica. Al analizar estos títulos se observa que todos siguen el mismo patrón: hacer acopio de recursos que se gastan en llevar a cabo determinadas acciones que cambian el estado del juego.

Esta observación lleva a plantear la hipótesis de si sería posible diseñar una herramienta de autoría que permitiera crear cualquier juego de este tipo, simplemente alterando las reglas y la interfaz gráfica. El presente trabajo analizará esta idea e intentará materializarla en una solución completamente operativa.

1.2 Descripción/Definición

El objetivo del proyecto es la creación de una plataforma con arquitectura web con un doble propósito. Por un lado, permitir a los diseñadores de juegos explotar su capacidad creativa mediante una herramienta que les facilite la creación de mundos temáticos persistentes sin necesidad de tener conocimientos técnicos. Por otro, poner estos mundos a disposición de una comunidad de usuarios que podrán disfrutar de una experiencia de juego compartida en tiempo real.

La plataforma consta de tres elementos. En primer lugar, el *frontoffice*, una web donde los usuarios podrán seleccionar un juego para iniciar o continuar una partida. En segundo lugar, un *backoffice* dirigido a los administradores del sitio y a los diseñadores de juegos, que ofrecerá las herramientas de gestión y configuración necesarias. Por último, una aplicación en el *backend* que se encarga de man-

1 https://lobby.ogame.gameforge.com/en_GB/ (recuperado el 02/04/2022)
2 https://es.bitefight.gameforge.com/game (recuperado el 02/04/2022)
3 https://onlinegame.travian.net/ (recuperado el 02/04/2022)
4 https://om.grepolis.com/grepo/ (recuperado el 02/04/2022)

tener funcionando las instancias de los mundos persistentes y que comunica con la web y el *backoffice* a través de una API Rest.

1.2.1 Web

La web es el punto de entrada de todos los jugadores. La página principal ofrece una lista de todos los títulos publicados, donde se da la opción al usuario de seleccionar cualquiera de los títulos disponibles. Al hacer esto se presenta la vista detallada del juego, donde se facilita información relevante como la imagen de portada, valoración de los usuarios, número de jugadores, etc. Si el usuario ya está identificado verá un control para continuar la partida o entrar al título seleccionado. De lo contrario, se le ofrece la posibilidad de identificarse o crear una nueva cuenta.

Cuando un jugador entra en un mundo seleccionado la interfaz cambia al cargador del juego. El cargador es una vista que ocupa toda la pantalla, tematizada según el juego seleccionado. Cuando el indicador de progreso llega al 100% se presenta al jugador la interfaz de juego donde llevará a cabo su actividad.

1.2.1.1 Elementos y mecánica de juego

El juego, despojado de cualquier rasgo de personalidad que el autor haya podido darle, se basa en dos conceptos fundamentales: la ejecución de actividades y la producción y consumo de recursos.

Toda acción que un jugador pueda llevar a cabo se considera una actividad, y toda actividad puede llevar asociado un consumo de recursos. Las actividades tienen un inicio, una duración y una finalización. Al iniciar una tarea se pone al final de la cola de ejecución del jugador. Las propiedades de esta cola determinan el número de tareas concurrentes que se pueden llevar a cabo. En el momento que una tarea llega a la cabeza de la lista, esta se activa y comienza una cuenta atrás cuya duración depende del tipo de actividad y del estado concreto del jugador. Cuando la cuenta atrás llega a su fin la tarea se elimina de la cola y se producen los efectos de finalización definidos por el autor.

El espacio en el que se desarrolla la acción es un amplio mundo bidimensional dividido en celdas. Cuando un usuario entra por primera vez se le asigna en propiedad un conjunto de celdas a una distancia prudencial de cualquier otro jugador. Este conjunto de celdas podrá ampliarse con el tiempo, llevando a cabo actividades de exploración y anexión sobre los elementos adyacentes no reclamados. La finalidad de las celdas es servir de receptáculo para la construcción de estructuras. Estas estructuras, generadas a partir de actividades de construcción, permiten recolectar recursos y modificar el estado general del juego o las propiedades del jugador.

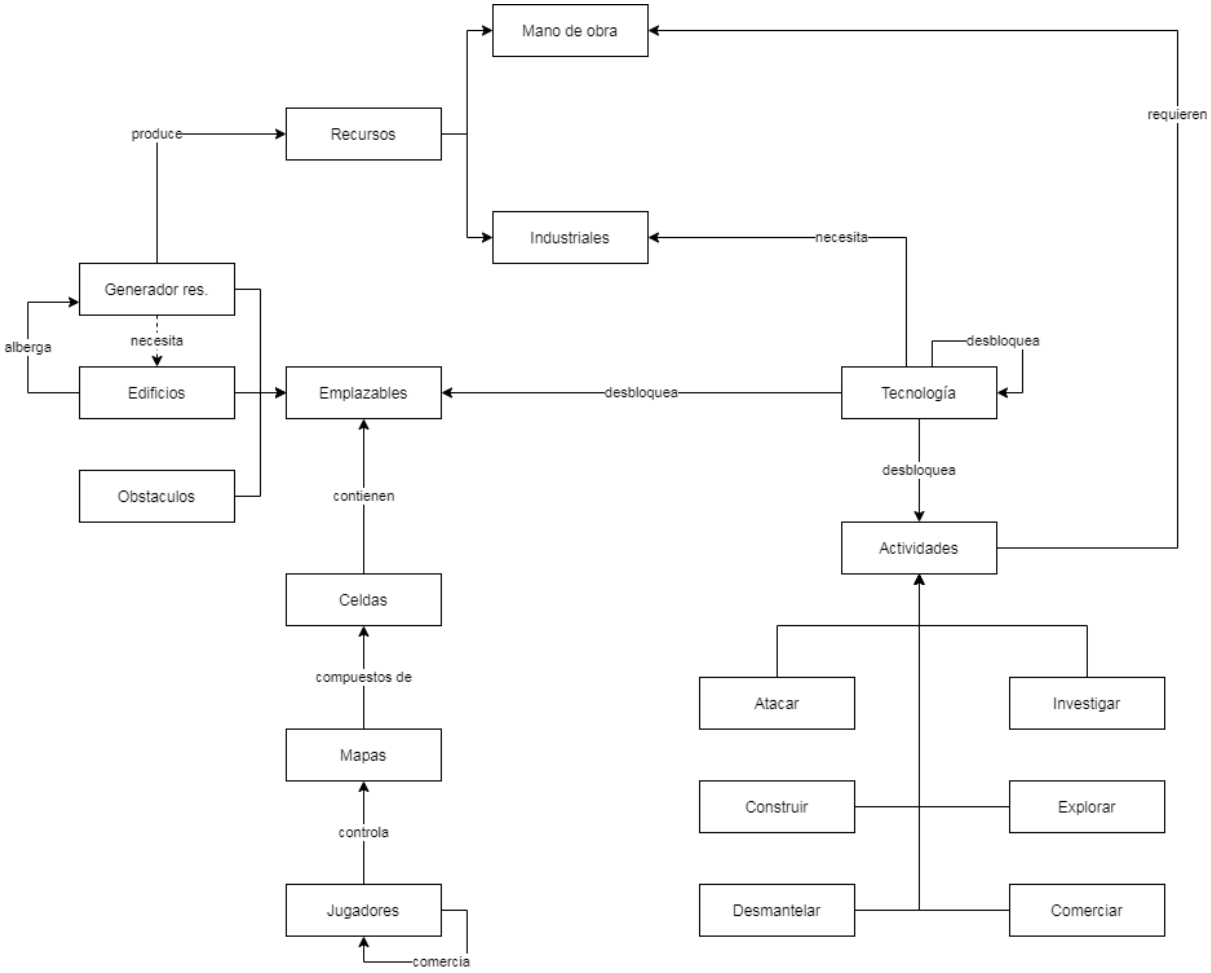


Figura 1: Mapa conceptual del juego base

Cada mapa puede albergar un número limitado de jugadores. Este requisito se impone para garantizar la calidad del servicio y la escalabilidad de la plataforma. Para asegurar que haya espacio para nuevos jugadores se ha dispuesto un sistema de instancias o "particiones". Cada instancia es una copia del juego aislada de todas las demás, donde los jugadores pueden interactuar entre sí pero no con los del resto de instancias. El jugador que entra por primera vez a un título es asignado automáticamente a la instancia con menor cantidad de jugadores con espacio disponible. Una vez establecido el vínculo entre instancia y jugador este no podrá romperse, de manera que cada vez que entre en el sistema siempre vuelva a la misma partición.

Jugadores

El jugador es un usuario del sistema, su objetivo es evolucionar su parcela del mapa y coexistir en régimen de competitividad o cooperación. El juego incorpora elementos para interactuar con el resto de jugadores, articulados a través de actividades de comunicación (mensajes), espionaje (obtención de información reservada), comercio (intercambio de recursos) y ataque.

Mapas

El mundo virtual es un mapa compartido entre todos los jugadores. A cada jugador se le asigna un área donde solo él podrá explotar los recursos asociados a las celdas y donde deberá desarrollar su actividad. El juego muestra dos tipos de mapas. En primer lugar, el mapa de área del jugador, donde puede ver las celdas de su propiedad y las colindantes, así como las estructuras que haya sobre estas . En segundo lugar un mapa del entorno que muestra el mundo compartido, donde puede obtener información sobre los jugadores del entorno y llevar a cabo actividades con ellos o en su contra.

Recursos

Los recursos son requisitos que el juego impone al jugador para llevar a cabo determinadas tareas. Estos se obtienen principalmente de dos maneras. Por un lado, construyendo un edificio adecuado sobre un generador de recursos, de manera que se pueda acceder al suministro. El ejemplo clásico sería el de construir una mina en una veta de oro. Por otro lado, construir edificios que generan recursos. Se disponen dos categorías de recursos diferentes: los recursos industriales y los humanos.

Mientras que los primeros son necesarios como coste para la construcción o desbloqueo de tecnología, los segundos son necesarios para llevar a cabo actividades. Por ejemplo, se puede definir que un ataque requiera 10 personas para llevarse a cabo. Cada generador produce recursos a un ritmo determinado de unidades por unidad de tiempo definido por el autor.

Ubicables

Los objetos ubicables pueden ser de tres tipos: edificios, generadores de recursos u obstáculos. En las celdas donde no haya obstáculos, el jugador podrá crear nuevas construcciones. Los obstáculos son ubicables sin ninguna utilidad, solo sirven para impedir que un jugador ponga un edificio en ese emplazamiento. Se dispone que podrán eliminarse si el jugador consigue la habilidad necesaria para hacerlo. Los generadores de recursos proveen al jugador de una cantidad de recursos por unidad de tiempo. Se establece que para que un generador de recursos provea suministro antes debe emplazarse sobre el un edificio adecuado para este fin. Los edificios son construcciones que permiten al jugador obtener nuevos tipos de recursos o variar el estado del juego. Un edificio construido sobre una celda se podrá desmantelar, recuperando un porcentaje de la inversión.

Actividades

Las actividades son las acciones que el jugador puede llevar a cabo. Inicialmente se definen seis tipos de actividades: exploración, ataque, construcción, comercio, investigación y desmantelamiento.

Los ataques se llevan a cabo contra otro jugador, requieren de mano de obra para efectuarlos y opcionalmente una cantidad fija de recursos, además tardan una cantidad de tiempo predefinida en ter-

minar. Cuando un ataque tiene éxito, se obtienen recursos procedentes del jugador atacado. Si el ataque no es exitoso, se pierden los recursos empleados. Una parte de estos recursos pasarán a formar parte del jugador atacado.

La exploración se efectúa ante eventos aleatorios que el jugador puede aceptar si así lo decide. Al igual que los ataques, tienen un coste material y una recompensa en caso de éxito. La investigación, como actividad, tiene un coste que depende de la tecnología que se desea investigar. La construcción implica emplazar un edificio en una celda libre o disponible.

El desmantelamiento supone eliminar un edificio existente para recuperar parte de sus recursos y dejar libre el espacio. El comercio es un proceso por el que un jugador pacta un intercambio de recursos con otro. El coste de esta actividad son los suministros ofertados, la recompensa los suministros pactados y el coste en tiempo dependerá de la distancia entre los jugadores y el volumen de recursos involucrado.

Todas las actividades tienen, además del posible coste de recursos, un coste en tiempo. Cuando se ejecuta una actividad, esta se pone en la cola de actividades. La cola de actividades tiene un número variable de espacios de ejecución, pero inicialmente es solo 1. Esto significa que si una actividad en la cola no se iniciará hasta que la anterior haya finalizado. Se plantea la posibilidad de que la investigación tecnológica permita aumentar el número de espacios disponibles.

Tecnología

Las tecnologías se definen como una serie de precondiciones que deben satisfacerse antes de poder acceder a determinadas estructuras, recursos o actividades. También pueden cambiar el estado del jugador o del juego, dando una ventaja táctica sobre los competidores. Estas se encuentran organizadas como un árbol, de manera que una tecnología puede ser un prerrequisito para otra. Por ejemplo, no se podría construir una planta de tratamiento de residuos sin investigar previamente ingeniería medioambiental.

1.2.2 Backoffice

El *backoffice* es un sitio web separado del portal de juegos, y tiene un doble propósito. Por un lado, facilitar las herramientas administrativas necesarias para gestionar la comunidad de usuarios, instancias y juegos. Por otro, dotar a los creadores de juegos de los medios necesarios para llevar a cabo su trabajo.

1.2.2.1 Gestión administrativa

El *backoffice* está a cargo de gestionar el ciclo de vida de las diferentes entidades que conforman el sistema. Si bien cada una tiene sus particularidades, todas comparten un subconjunto de operaciones denominadas CRUD[5] que se describen a continuación:

Altas - Se presenta al usuario un formulario modal con las propiedades de la entidad gestionada. Al terminar la operación su valor se envía al servidor a través de la API y se guarda en la base de datos.

Modificaciones - Igual que un alta, pero presenta el estado de la entidad en los campos del formulario Al terminar la operación su valor se actualiza en la base de datos.

Consultas - Las consultas se llevan a cabo introduciendo el criterio de búsqueda en los controles que la lista de entidades facilita. Al introducir un criterio, la lista devuelve los resultados que cumplen la condición. Cada ítem de la lista de entidades presenta controles para editar, eliminar o ejecutar acciones alternativas, en función del tipo de entidad que se consulta.

Bajas - Las entidades se dan de baja directamente desde la lista de resultados, interactuando con el control dispuesto a tal efecto.

Aunque el sistema tiene decenas de entidades hay tres con un papel fundamental a las que el backoffice dedica una sección en exclusiva para su gestión y tratamiento:

Usuarios y jugadores

Un usuario es una persona con credenciales de acceso a la plataforma, ya sea para participar en un juego o para llevar a cabo tareas de administración. La vista principal de esta sección presenta un listado filtrado y paginado de usuarios en la que se presentan diferentes acciones para cada elemento. En primer lugar, la opción “ver usuario” navega hacia el visor de usuarios, donde se muestran sus datos personales y la lista de juegos que ha creado. En segundo lugar, la opción de editar lleva al formulario donde establecer los permisos y datos de contacto. En tercer lugar, existe la opción de banear a un usuario jugador, impidiéndole la entrada a los juegos durante un periodo de tiempo determinado. Para terminar, se facilita un acceso al formulario de creación de usuarios, idéntico al de edición pero para generar nuevas entidades.

Instancias

Las instancias -o particiones- son copias del mundo persistente definido por el título al que pertenecen, y tienen como propósito limitar el número de jugadores que pueden interactuar. Esta limitación se impone por motivos técnicos y se describirá con posterioridad. Cada instancia tiene un número

5 Acrónimo de Create, Read, Update, Delete, más información en https://developer.mozilla.org/es/docs/Glossary/CRUD (recuperado el 05/03/2022)

máximo de espacios donde pueden entrar los jugadores. Cuando un jugador entra por primera vez a un juego queda vinculado automáticamente con la instancia que tenga un hueco libre. Si no hubiera ninguna, recibiría un mensaje de error. La interfaz administrativa permite crear, modificar y buscar instancias. El formulario de creación de instancias permite indicar el número de huecos y el título al que pertenece. El formulario de modificación solo permite modificar el número de huecos, y nunca por debajo del número de jugadores vinculados. Cada ítem de la lista de instancias presenta un control para localizar los jugadores participantes, y lleva a la vista de consulta de jugadores con el filtro de instancia precargado.

Títulos

La sección de títulos permite crear, editar, modificar y consultar los títulos de la plataforma. El formulario de alta y modificación contiene el nombre del título, la descripción, la valoración media, la imagen de miniatura y la de portada. La lista de ítems tiene controles para lanzar el editor de juegos y para consultar las instancias asociadas.

1.2.2.2 El editor de configuración

El editor de configuración es una herramienta específica para los diseñadores de juegos, y obedece a dos propósitos principales: personalizar la interfaz del juego y definir las reglas que regirán el mundo persistente. La interfaz del editor organiza la información en ocho secciones con sus respectivas subsecciones. Cada subsección incorpora un control para editar un aspecto específico del juego, o en el caso de presentar una lista de elementos, muestra un modal con un tercer nivel de información.

Personalización de la interfaz

El editor permite configurar el aspecto visual de cada elemento del juego dentro de sus respectivas secciones. Se establecen como configurables los siguientes elementos: celdas, tecnologías, actividades, estructuras y recursos. Las características editables incluyen recursos textuales (etiquetas y descripciones) y recursos gráficos (iconos, miniaturas e imágenes de portada). Por otro lado, se da al autor la posibilidad de cambiar el aspecto general de los controles de interacción del juego (UI[6]). Esto se traduce en un formulario donde podrá seleccionar las familias tipográficas empleadas, las gamas cromáticas y las imágenes de fondo.

Configuración de las reglas

El editor permite la edición de las reglas de juego de dos formas diferentes. En primer lugar, dando al autor el control sobre la forma en que se administra la economía del juego, permitiéndole definir qué recursos habrá disponibles y qué será necesario para recolectarlos. Para esto se facilitan los CRUDs de celdas, recursos y estructuras. Cada elemento se relaciona con los otros dos para definir la base de la generación de recursos, y se da al autor el control necesario para ajustarlo.

6 Acrónimo de *User Interface*, o interfaz de usuario.

En segundo lugar, con la parametrización de las actividades. La sección de actividades no permite crear nuevos tipos de actividad, ya que son elementos que necesariamente deben estar cableados en el código de la aplicación. Sin embargo, permite al autor determinar qué acciones estarán disponibles y cuales no, así como establecer la base de cálculo para la duración, el coste y el resultado de cada actividad.

Para terminar, algunos elementos configurables presentan una subsección de propiedades generales.. Estas consisten en un conjunto de claves y valores que mejoran las estadísticas del usuario a la hora de llevar a cabo determinadas actividades, o que varían la experiencia de juego.

1.2.3 Servidor

El servidor es el nexo de unión entre la web y el *backoffice*. Su misión es la de recibir las órdenes de los jugadores y traducirlas en cambios de estado dentro de la aplicación, con su consiguiente respuesta. El servidor consta de cuatro partes bien diferenciadas.

En primer lugar se expone una API Rest que será consumida tanto por la web como por el *backoffice*. Este mecanismo permite separar la representación de los datos de su gestión y tratamiento. En consecuencia, se simplifica el diseño de la plataforma tanto en cliente (*frontend*) como en servidor (*backend*).

En segundo lugar, introduce un sistema de comunicaciones asíncronas para facilitar actualizaciones al juego en tiempo real. Esta funcionalidad obedece a la necesidad de informar al primero de eventos relevantes que ocurren al margen de la actividad del jugador, tales como mensajes de otros usuarios, actividades que involucran al jugador o resultados de finalización de actividades propias. Aunque se podría pensar que este objetivo se puede conseguir mediante *polling*, cabe señalar que solicitar actualizaciones de estado periódicas supone una sobrecarga en la API que escala con el número de usuarios de cada instancia de juego, lo que puede derivar en una degradación del servicio y de la experiencia de juego.

En tercer lugar, una capa de aplicación a cargo de la gestión de todos los elementos de juego. Por un lado, tiene como misión gestionar las instancias de los diferentes títulos y los usuarios conectados. Esto incluye procesar las actividades de las colas de ejecución de los usuarios, calcular los balances de recursos teniendo en cuenta los ingresos y los gastos, así como generar las notificaciones informativas que deben enviarse de vuelta a la web como respuesta a un fin de actividad o suceso acaecido. Por otro, hace de intermediario entre la API y el sistema de almacenamiento de datos, facilitando la consulta de información y actualizando los datos cuando así se le requiera.

Por ultimo, la base de datos. Este elemento está oculto al resto del sistema y solo es accesible desde la capa de aplicación. Su finalidad es servir de almacén de datos a largo plazo para toda la información del sistema, lo que incluye (pero no se limita a) los títulos disponibles, las configuraciones de juego, los usuarios de la plataforma y la información de jugadores. Además, tiene la misión de servir de apoyo a las instancias de juego en caso de desastre. Los cambios de estado que producen las acciones de los jugadores sobre las instancias se mantienen siempre en memoria por motivos de eficiencia. Esto ofrece la ventaja de ser tremendamente rápido. Sin embargo, tiene el inconveniente de que si el software que ejecuta el código del servidor termina de forma abrupta, toda la información de juego se pierde irremediablemente. En consecuencia, la capa de aplicación guarda periódicamente el estado de cada instancia, de manera que en caso de fallo del sistema haya un punto anterior reciente al que volver. Este mecanismo de copia diferida ofrece un buen compromiso entre eficiencia y seguridad.

1.3 Objetivos generales

A continuación se enumeran los objetivos del proyecto, agrupados por tipo y ordenados por relevancia.

1.3.1 Objetivos principales

Objetivos de la aplicación

- Crear videojuegos temáticos para navegador a partir de un sistema base.
- Mantener mundos persistentes basados en los juegos creados.
- Ofrecer una experiencia multijugador en tiempo real a los usuarios.

Objetivos para el usuario:

- Crear y diseñar juegos de navegador de manera sencilla.
- Administrar y gestionar los usuarios, jugadores e instancias de la plataforma.
- Participar en la experiencia de juego.

Objetivos personales:

- Crear un proyecto completo desde su planteamiento hasta su puesta en marcha.
- Plantear un sistema complejo a partir de componentes simples bien organizados.
- Emplear las prácticas de desarrollo de necesarias para generar un software mantenible y de calidad.

1.3.2 Objetivos secundarios

- Unir diferentes disciplinas transversales en el grado multimedia.

1.4 Metodología y proceso de trabajo

1.4.1 Evaluación de la metodología de trabajo

Para elegir la metodología de trabajo adecuada se deben tener en cuenta las características del producto, el tiempo disponible y el contexto en el que se ejecuta.

En primer lugar, la plataforma está compuesta de tres partes bien diferenciadas: la web, el *backoffice* y el servidor. Aunque todas son necesarias, cada una aporta una cantidad de valor diferente al usuario. Por ejemplo, la web puede reproducir los juegos incluso si no hubiera un *backoffice*, simplemente leyendo una configuración prediseñada. En segundo lugar, la lista de funcionalidades que hay que desarrollar es muy amplia, pero el tiempo destinado a planificar, analizar, desarrollar y probar es limitado. Por último, la naturaleza del proyecto hace que deba ser evaluado en cuatro sesiones a lo largo de tres meses.

Una metodología de desarrollo en cascada puede no ser muy adecuada en este contexto, ya que sería necesario tener el análisis y diseño de los tres componentes de software antes de empezar a materializar el producto. Por otro lado, nada garantiza que el análisis inicial sea correcto, lo que obligaría a examinar de nuevo los requisitos y rehacer el producto. Además, una metodología incremental permite obtener secciones funcionales de la plataforma sin necesidad de que las demás estén completas. Por ejemplo, sería posible tener toda la gestión de usuarios desarrollando las partes necesarias en *backoffice*, servidor y *frontend*. Sin embargo, este método obliga a consumir tiempo en desarrollar completamente aspectos que pueden aportar poco o ningún valor. Finalmente, la metodología iterativa permite la entrega de la funcionalidad justa y necesaria para hacer una prueba y verificar el resultado.

Así pues, tras evaluar los pros y los contras de cada una, la metodología iterativa se evidencia como la más adecuada para el desarrollo de este proyecto.

1.4.2 Proceso de trabajo

Para la elaboración de las características de la plataforma se ha tomado prestado de las metodologías ágiles el concepto de historia de usuario. Cada una de estas historias es un breve texto que describe la necesidad de un usuario que utilice la plataforma, acompañado de una serie de puntos que el

sistema debe cumplir para satisfacer la historia. Estos son los denominados criterios de aceptación, y son los que en última instancia identifican si el producto cumple con las expectativas.

A través de la evaluación de diferentes portales de juegos (GameForge[7], Mundijuegos[8]) se han identificado un total de 74 historias de usuario que se consideran necesarias para el producto. Cabe señalar que las historias describen el producto desde la perspectiva del usuario, y por tanto no entran a valorar la infraestructura de la plataforma. Cuando se considera cerrada la lista de historias de usuario, se procede a analizar cada una para romperlas en tareas transversales más pequeñas, que puedan llevarse a cabo en un día o menos.

Una vez elaborada la lista se procede a ordenar las tareas en función del valor que aportan al usuario. Así pues, se establece que las tareas más prioritarias son aquellas que involucran a los jugadores, seguidas de las necesarias para los creadores, y después el resto. A continuación se definen las diferentes iteraciones por las que pasará la plataforma. Dado que hay cuatro sesiones de evaluación separadas casi un mes entre sí, se definen dos iteraciones o *sprints* entre cada sesión, de forma que si fuera necesario efectuar correcciones sobre el producto, estas se puedan llevar a cabo antes de la sesión de evaluación.

1.5 Planificación

Fecha	Hito	Iteración	Funcionalidad esperada
06/03/2022	PEC2	Sprint 1	Carga de juego, mapa de zona
		Sprint 2	Investigación tecnológica
03/04/2022	PEC3	Sprint 3	Procesos en *background* y notificaciones
		Sprint 4	Actividades con jugadores
08/05/2022	PEC4	Sprint 5	Editor de configuración de juego
		Sprint 6	Producto completo
06/06/2022	PEC5	-	

Tabla 1: Hitos del proyecto

7 https://gameforge.com/es-ES/ (recuperado el 02/04/2022)
8 https://www.mundijuegos.com/ (recuperado el 02/04/2022)

1.5.1 Diagrama de Gantt

Nombre de tarea	Comienzo	Fin	Duración
PEC1	dom 20/02/22	dom 20/02/22	0 días
PEC2	dom 06/03/22	dom 06/03/22	0 días
PEC3	dom 03/04/22	dom 03/04/22	0 días
PEC4	dom 08/05/22	dom 08/05/22	0 días
PEC5	lun 06/06/22	lun 06/06/22	0 días
Sprint 1	**lun 07/03/22**	**dom 20/03/22**	**12,36 días**
Sprint 2	**lun 21/03/22**	**dom 03/04/22**	**12,36 días**
Sprint 3	**lun 04/04/22**	**dom 17/04/22**	**12,36 días**
Sprint 4	**lun 18/04/22**	**dom 08/05/22**	**18,07 días**
Sprint 5	**lun 09/05/22**	**dom 22/05/22**	**12,36 días**
Sprint 6	**lun 23/05/22**	**lun 06/06/22**	**12,36 días**

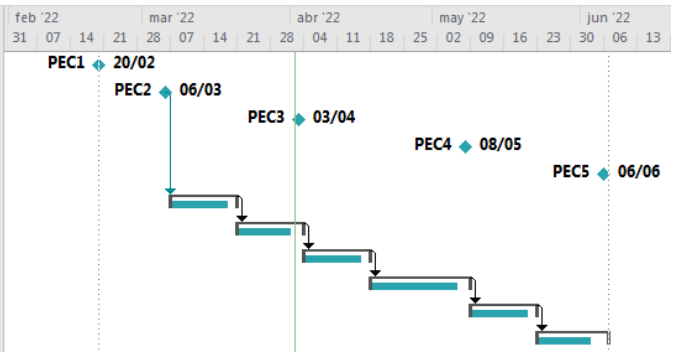


Figura 2: Cronograma completo del proyecto

Nombre de tarea	Duración
Sprint 1	**12,36 días**
9. Lobby de juegos	4 horas
11. Ficha de juego	4 horas
14. Componente del cliente de juego	24 horas
15. Componente de panel de recursos	6 horas
24. Componente de notificaciones emergentes (toasts)	4 horas
16. Componente de panel informativo	8 horas
17. Componente de área de jugador	4 horas
18. Mapa de zona del jugador	14 horas

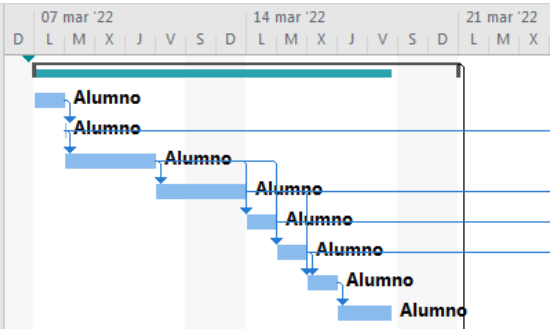


Figura 3: Cronograma del primer sprint

Nombre de tarea	Duración
Sprint 2	**12,36 días**
49. CRUD juegos	4 horas
70. Cargar configuración	4 horas
10. Lista de juegos disponibles	4 horas
71. Guardar configuración	16 horas
50. CRUD usuarios	4 horas
1. Proceso de registro	3 horas
3. Gestión de la sesión del usuario	4 horas
51. CRUD instancias	4 horas
19. Componente de vista de recursos	4 horas
20. Componente de recurso generado	4 horas
21. Componente de vista de tecnologías	4 horas
22. Mapa de tecnologías investigadas y disponibles	16 horas

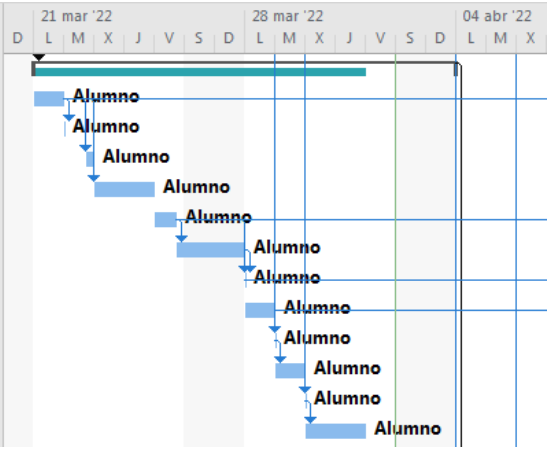


Figura 4: Cronograma del segundo sprint

Nombre de tarea	Duración
Sprint 3	**12,36 días**
23. Componente de vista de notificaciones	4 horas
25. Componente lista de notificaciones	4 horas
26. Componente detalles de notificación	4 horas
30. Componente de vista de actividades	24 horas
31. Componente de lista de actividades	8 horas
32. Lista de actividades propias	2 horas
33. Lista de actividades con vinculación	2 horas
38. Gestión de producción de recursos	8 horas
42. Gestión de notificaciones asíncronas	8 horas
43. Notificar eventos de fin de actividad	2 horas

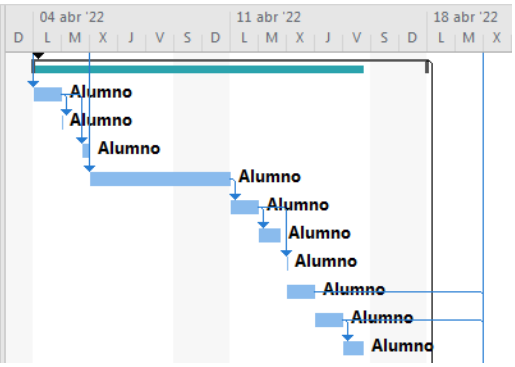


Figura 5: Cronograma del tercer sprint

Nombre de tarea	Duración
Sprint 4	**18,07 días**
39. Gestión de colas en background	8 horas
40. Añadir actividad a la cola del jugador	2 horas
41. Cancelar actividad de la cola del jugador	2 horas
44. Notificar eventos de recolección de recursos	2 horas
45. Notificar evento de sincronización de actividad	2 horas
34. Componente de ventanas modales	6 horas
35. Componente de ventana de ataque	10 horas
36. Componente de ventana de mensaje	8 horas
37. Ventana modal de comercio	8 horas
27. Componente de vista de entorno	4 horas
29. Componente lista de jugadores	8 horas
28. Componente mapa del mundo	16 horas
46. Generar imagen de entorno basada en ubicación del mapa	16 horas
53. Gestión de sesiones de administrador	4 horas
47. Listar jugadores por zona	8 horas
52. Login de admistración	2 horas
68. Subida de imagenes	4 horas

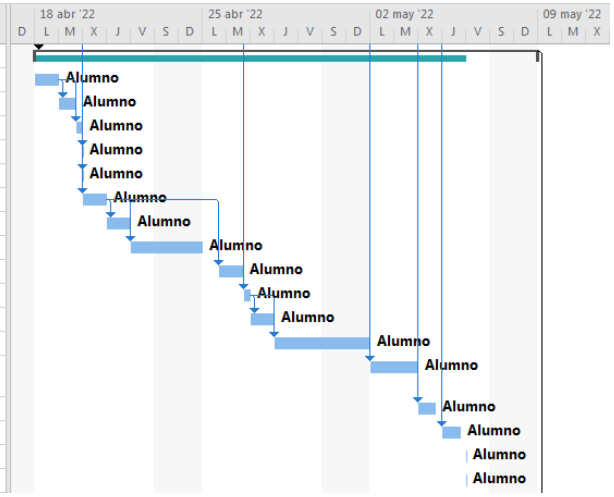


Figura 6: Cronograma del cuarto sprint

Nombre de tarea	Duración
Sprint 5	**12,36 días**
48. Componente maestro/detalle parametrizable	16 horas
61. Formulario de edición de instancias	2 horas
73. Formulario de propiedades de jugador	2 horas
54. Formulario de edición de usuarios	2 horas
56. Ventana modal de permisos	2 horas
63. Ventana modal de prohibición de entrada	2 horas
58. Cargador del editor de juegos	4 horas
12. Modificar rating de juego	2 horas
2. Formulario de login	2 horas
66. Eliminación completa de un jugador	4 horas
4. Formulario recuperación contraseña	2 horas
59. Editor de configuración de juego	**4,86 días**
67. Componente de selección/carga de imagen con previsualización	8 horas
55. Sección de usuarios	4 horas
60. Sección de instancias	4 horas
62. Sección de jugadores	4 horas
80. Formulario playlists	4 horas
57. Sección de títulos	4 horas
69. Componente general de entidades de juego (nombre,desc,img, etc)	6 horas

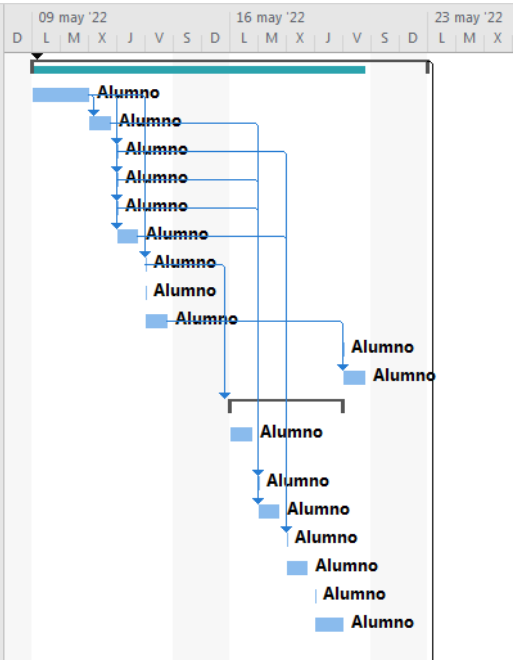
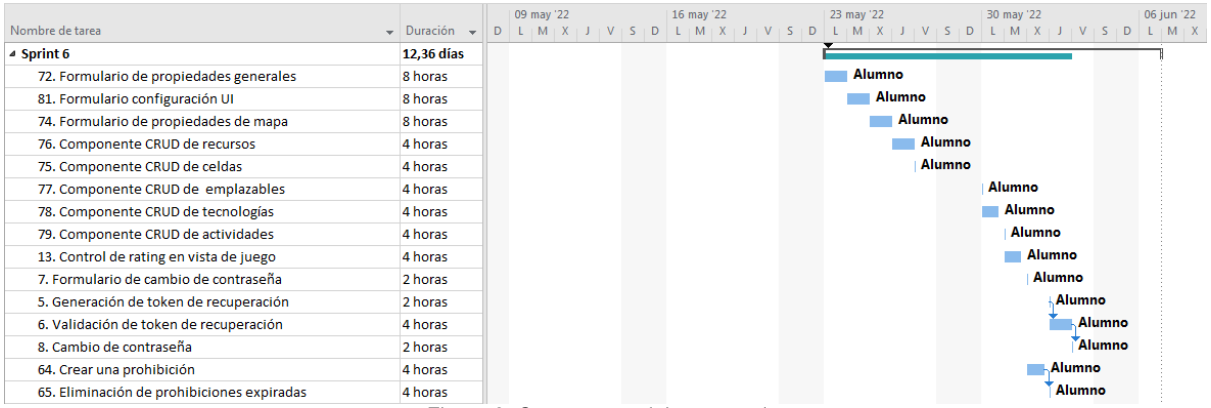


Figura 7: Cronograma del quinto sprint



Nombre de tarea	Duración
Sprint 6	**12,36 días**
72. Formulario de propiedades generales	8 horas
81. Formulario configuración UI	8 horas
74. Formulario de propiedades de mapa	8 horas
76. Componente CRUD de recursos	4 horas
75. Componente CRUD de celdas	4 horas
77. Componente CRUD de emplazables	4 horas
78. Componente CRUD de tecnologías	4 horas
79. Componente CRUD de actividades	4 horas
13. Control de rating en vista de juego	4 horas
7. Formulario de cambio de contraseña	2 horas
5. Generación de token de recuperación	2 horas
6. Validación de token de recuperación	4 horas
8. Cambio de contraseña	2 horas
64. Crear una prohibición	4 horas
65. Eliminación de prohibiciones expiradas	4 horas

Figura 8: Cronograma del sexto sprint

1.5.2 Diagrama de dependencias entre tareas

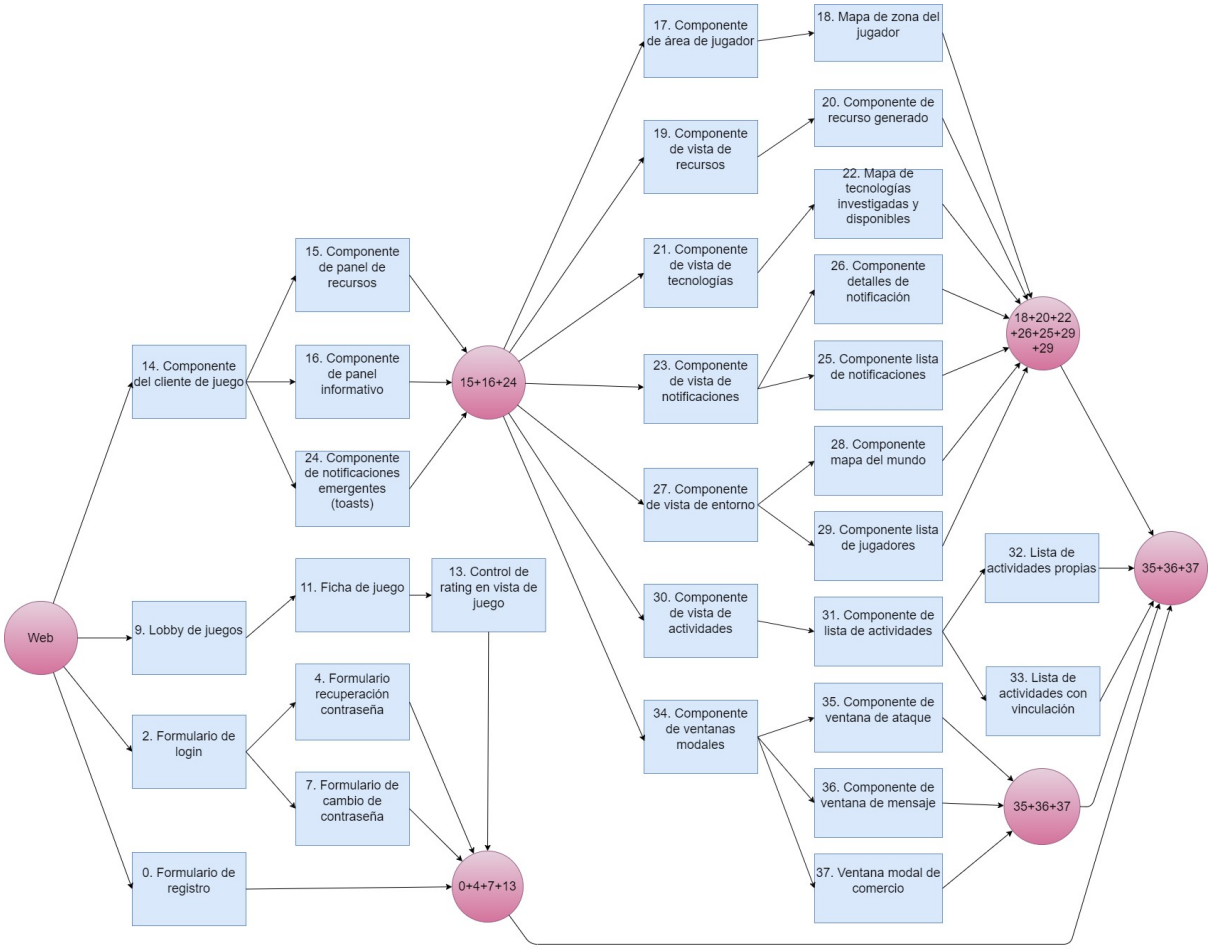


Figura 9: Diagrama de dependencia de tareas en el *frontoffice*

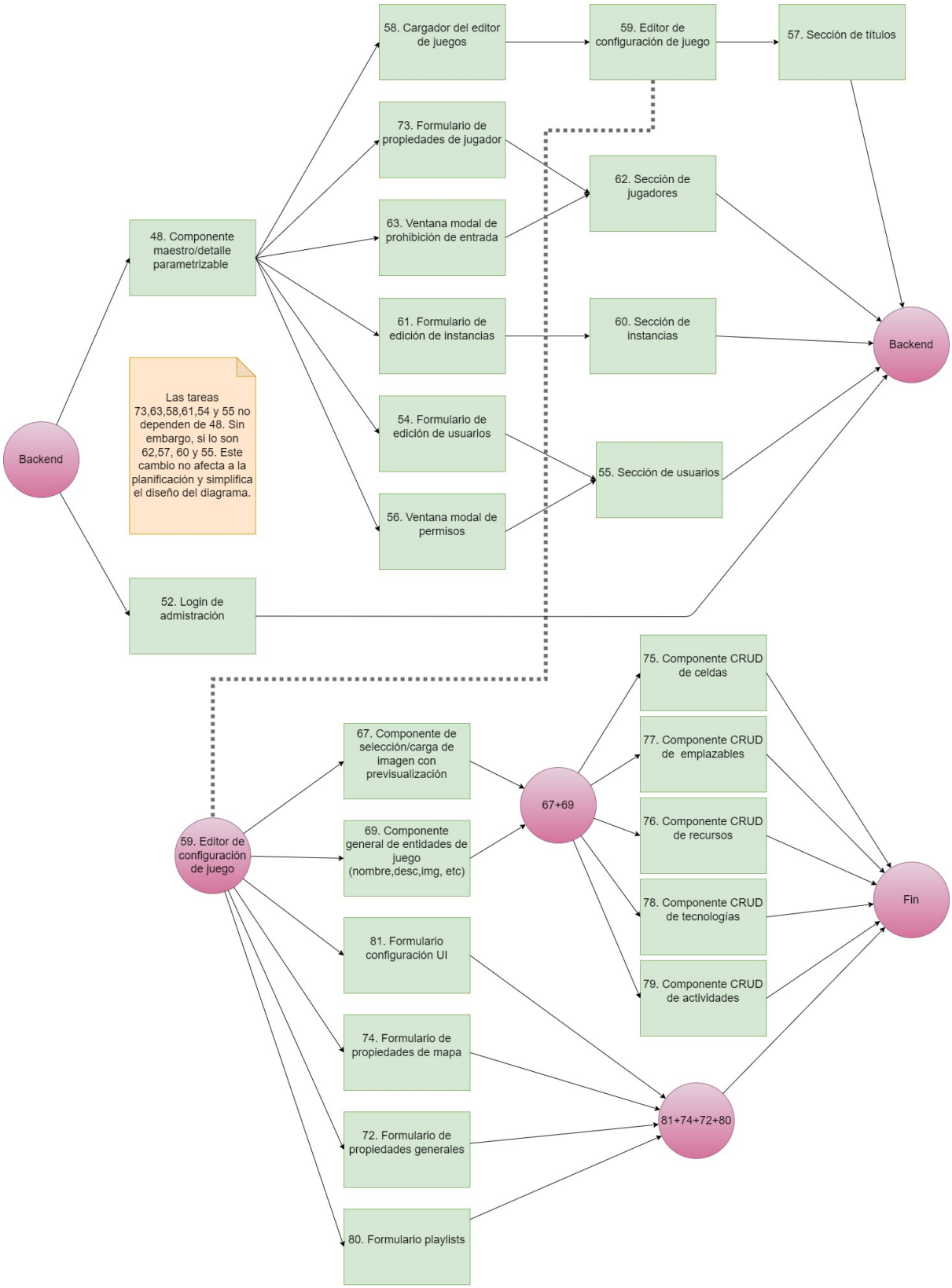


Figura 10: Diagrama de dependencia de tareas en el *backoffice*

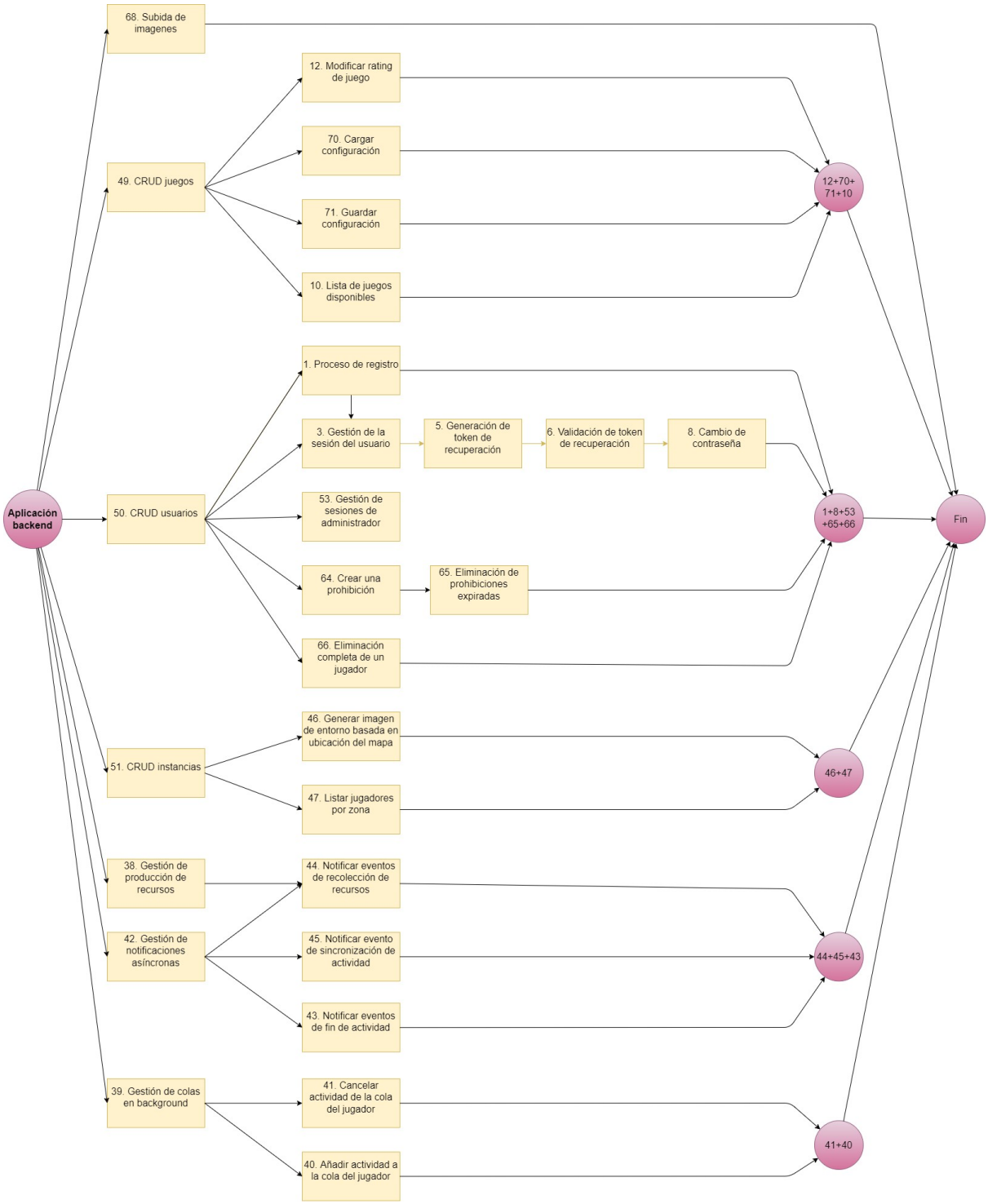


Figura 11: Diagrama de dependencia de tareas en la aplicación de *backend*

1.5.3 Estructura de división del trabajo

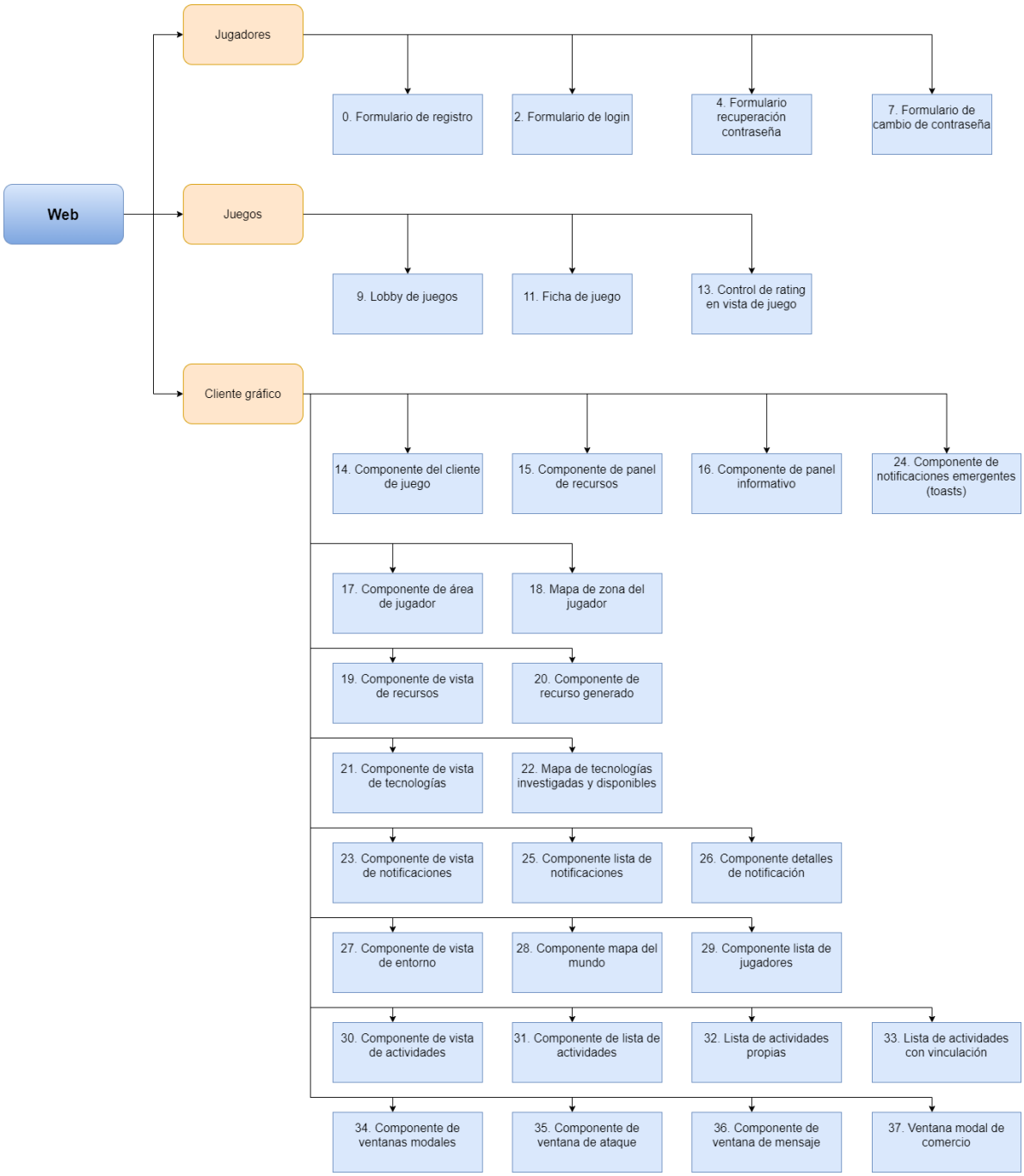


Figura 12: Estructura de división del trabajo en el *frontoffice*

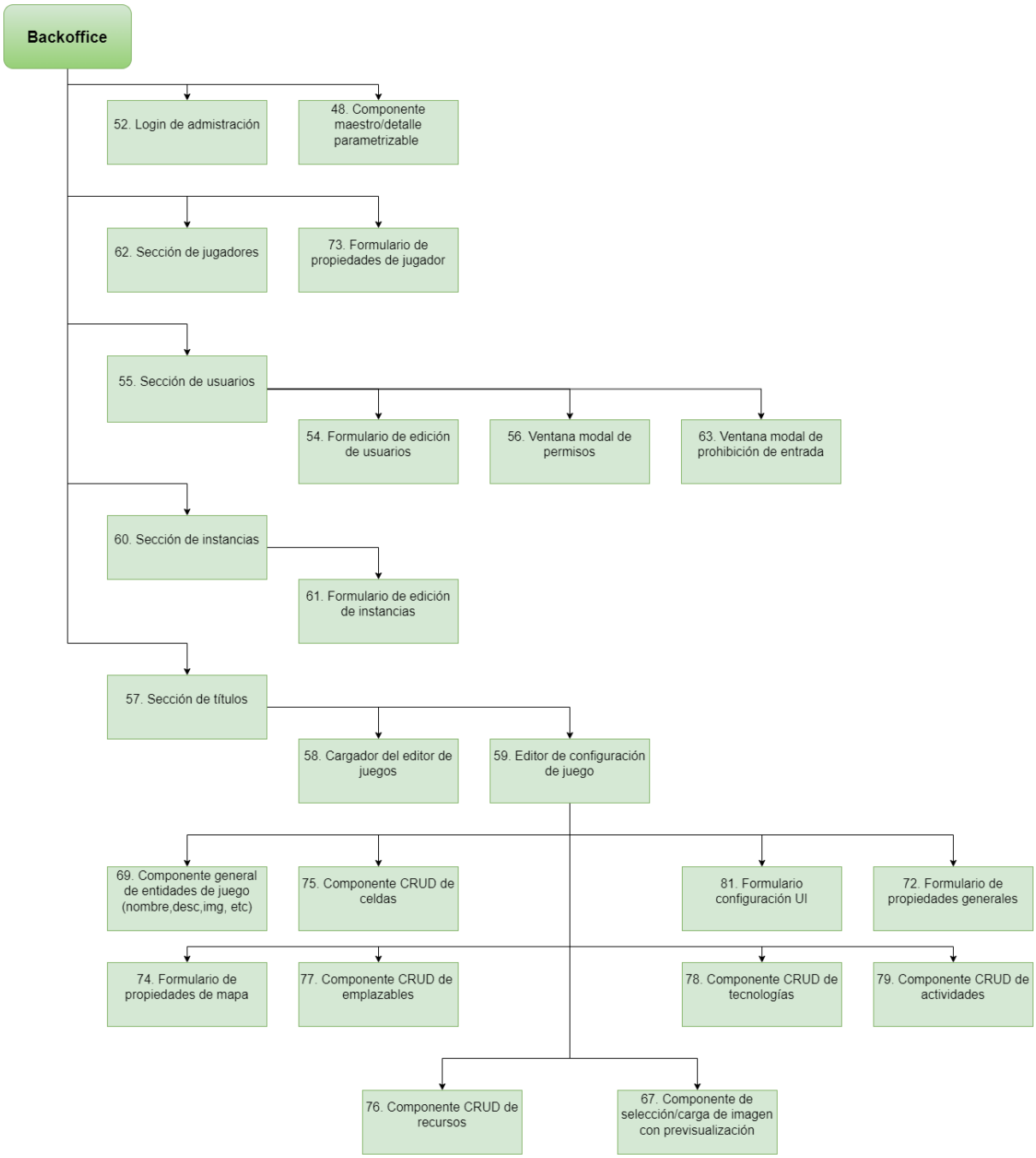


Figura 13: Estructura de división del trabajo en el *backoffice*

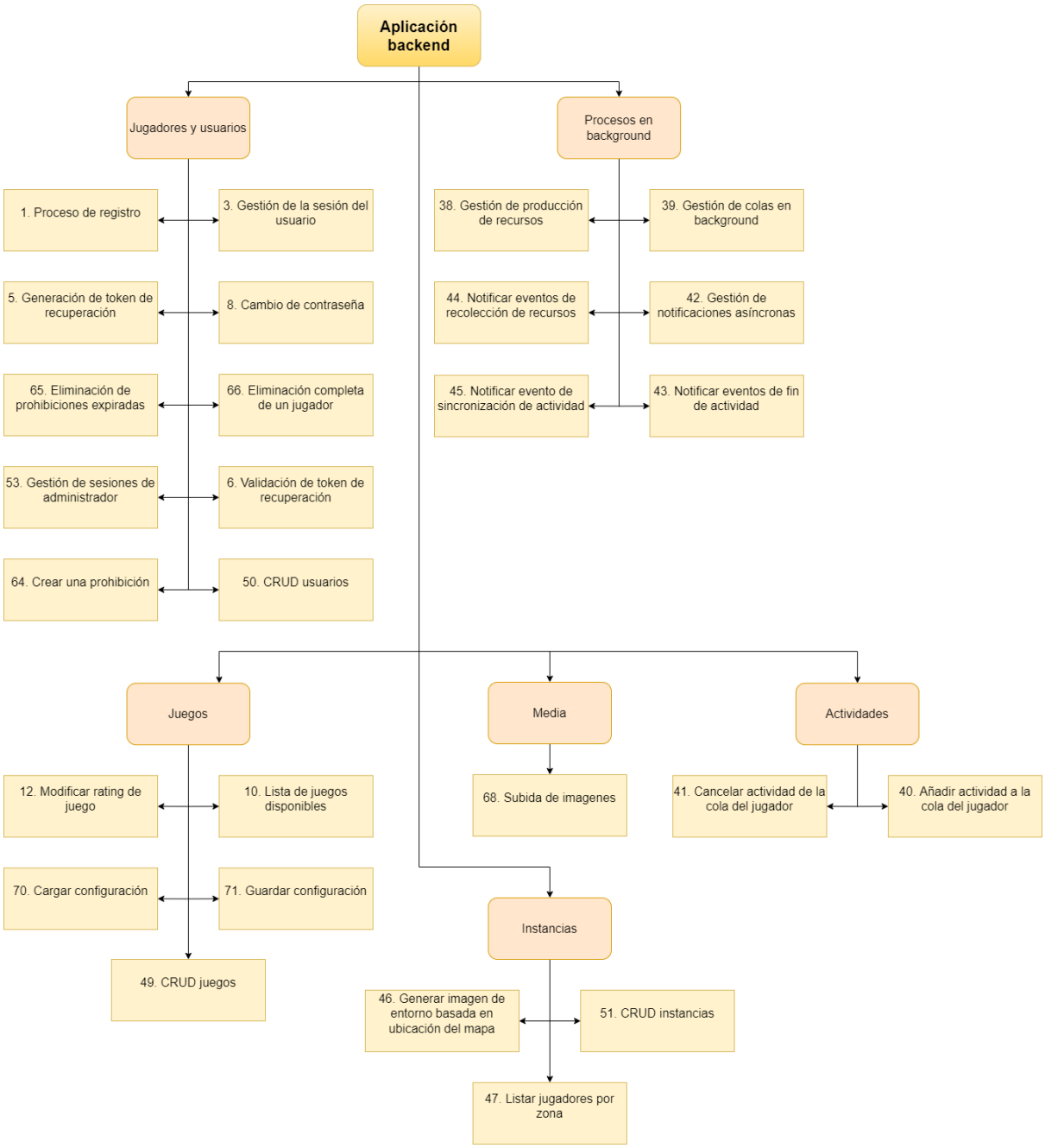


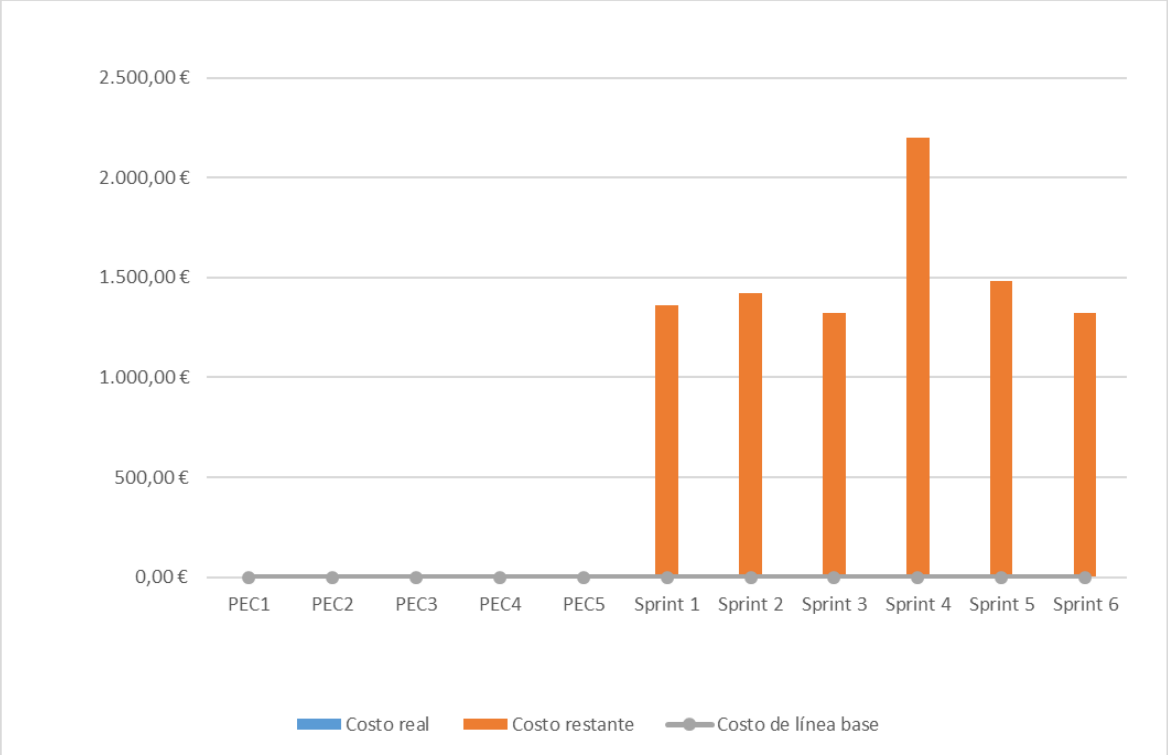
Figura 14: Estructura de división del trabajo en la aplicación del *backend*.

1.6 Presupuesto

Se han estimado 81 tareas a llevar a cabo a lo largo de 3 meses con un único programador, que se encargará de desarrollar tanto el *frontend* como el *backend*. Se establece un horario de trabajo de 8 horas diarias con una tarifa única de 20 €/hora. A continuación se presenta el presupuesto obtenido desglosado por iteración:

ESTADO DE COSTO

Estado de costo para las tareas de nivel superior.



DETALLES DE COSTOS

Detalles de costos para todas las tareas de nivel superior.

Nombre	Costo fijo	Costo real	Costo restante	Costo	Costo de línea base	Variación de costo
PEC1	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €
PEC2	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €
PEC3	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €
PEC4	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €
PEC5	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €
Sprint 1	0,00 €	0,00 €	1.360,00 €	1.360,00 €	0,00 €	1.360,00 €
Sprint 2	0,00 €	0,00 €	1.420,00 €	1.420,00 €	0,00 €	1.420,00 €
Sprint 3	0,00 €	0,00 €	1.320,00 €	1.320,00 €	0,00 €	1.320,00 €
Sprint 4	0,00 €	0,00 €	2.200,00 €	2.200,00 €	0,00 €	2.200,00 €
Sprint 5	0,00 €	0,00 €	1.480,00 €	1.480,00 €	0,00 €	1.480,00 €
Sprint 6	0,00 €	0,00 €	1.320,00 €	1.320,00 €	0,00 €	1.320,00 €

Figura 15: Coste del proyecto por hito

En la tabla siguiente, se observa el presupuesto desglosado por área:

Área	Web	Backoffice	Backend	TOTAL
Tiempo	221 h	88 h	146 h	455 h
Coste	4420 €	1760 €	2920 €	9100 €

Tabla 2: Coste del proyecto por área de trabajo.

Para la puesta en marcha del servicio se opta por una solución híbrida entre el *hosting*[9] tradicional y los servicios en la nube, de manera que se pueda aprovechar al máximo las características de cada plataforma. El *frontend* (*backoffice* y *frontoffice)* estará alojado en un *hosting* tradicional asociado a un dominio, ya que sólo se requiere un buen ancho de banda y espacio disponible en disco. El *backend* que alberga la API necesita una infraestructura algo más compleja, por lo que se opta por un VPS[10] en la nube con las características requeridas (acceso por consola, apertura de puertos, etc.). Finalmente, la base de datos estará externalizada en los servicios en la nube de MongoDB Atlas, agilizando la administración y dejando libre al servidor virtual para dedicarse íntegramente al *backend*.

Tomando como referencia los proveedores Digital Ocean[11] y Hostalia[12], se establecen los siguientes costes asumiendo un total de 5 juegos con 2 instancias cada uno, durante un periodo de doce meses:

Concepto	Proveedor	Cantidad	Precio	Total
Droplet (unidad de ejecución)	DigitalOcean	10	5 € / mes	600 €
MongoDB cloud service	MongoDB	1	15 € / mes	180 €
Dominio	Hostalia	1	6,99 € / año	6,99 €
Alojamiento	Hostalia	1	7,99 € / mes	95,88 €
Total				882,87 €

Tabla 3: Costes de explotación del proyecto

En resumen, el coste total del proyecto asciende a 9982,87 € incluyendo un año de servicio en la nube, alojamiento y dominio.

9 Servicio de alojamiento de páginas web, puede permitir tanto contenido estático como dinámico.
10 Servidor privado virtual, una solución de computación basada en virtualización. Más información en https://es.wikipedia.org/wiki/Servidor_virtual_privado (Recuperado el 30/05/2022)
11 https://www.digitalocean.com/ (recuperado el 02/04/2022)
12 https://www.hostalia.com/ (Recuperado el 30/05/2022)

1.7 Estructura del resto del documento

A continuación se describe la estructura del resto de secciones de este documento.

- **Análisis de mercado** – Información relativa al mercado al que está orientado el producto.
- **Propuesta** – Propuesta del proyecto, con énfasis en las características diferenciadoras.
- **Diseño** – Características del proyecto desde el punto de vista de la arquitectura de la información, la interfaz y el estilo visual.
- **Implementación** – Información relevante para la puesta en marcha
- **Demostración** – Prototipos de alta y baja calidad e información sobre el procedimiento de explotación de la plataforma
- **Conclusiones y líneas de futuro** – Retrospectiva y nuevas oportunidades.

2 Análisis de mercado

2.1 Público objetivo

La plataforma, dada su naturaleza, tiene tres tipos de usuarios finales como objetivo: los jugadores, los creadores de contenido y los administradores de la plataforma. En todos los casos, son usuarios competentes en el uso de los ordenadores y los dispositivos móviles.

El *target* de los jugadores se concentra en el rango de edad entre los 18 y los 34 años (Statista, 2021), sin importar el género. Su motivación para usar la plataforma es principalmente el entretenimiento y la socialización con otros jugadores. Por su parte, los creadores de contenido buscan la posibilidad de materializar sus ideas dentro de la plataforma, dando rienda suelta a su creatividad y facilitándoles la labor de crear comunidades de usuarios alrededor de sus invenciones. Por ultimo, los administradores de la plataforma son aquellos que manipulan el *backoffice* para llevar a cabo las operaciones administrativas que aseguren el buen funcionamiento del sistema.

2.2 Casos de estudio

A continuación se analizarán dos de los productos en los que se ha inspirado este proyecto para determinar sus características básicas. Para ello, se establecen dos categorías de criterios de comparación: criterios técnicos y criterios conceptuales o de *gameplay*:

- **Técnicos**
  - Tecnología del producto
  - *Responsiveness*[13]
  - Actualización de estado del sistema
  - Interacción
- **Gameplay**
  - Temática
  - Recursos
  - Mapa local
  - Mapa general
  - Estadísticas
  - Colas
  - Grafismo
  - Mensajería
  - Construcciones
  - Actividades

13 La cualidad de un producto web para adaptarse a los diferentes tamaños y esquemas de pantalla.

2.2.1 OGame

Figura 16: Vista general de la interfaz de OGame

Ogame es un MMORTS[14] para navegador, basado en PHP,Javascript y CSS. El juego está basado en el género espacial, y se especializa en la colonización espacial y las batallas entre grandes flotas de naves. El mapa local está compuesto por los diferentes planetas que el jugador puede haber conquistado, aunque no se dispone de un mapa físico que los relacione espacialmente. De igual forma, el mapa global se limita a un listado de los planetas cercanos y sus propietarios, que permite al jugador conocer su entorno.

La interfaz no es adaptable, en dispositivos móviles se visualiza a una escala reducida para ocupar el 100% de la pantalla, mientras que en escritorio surgen barras de desplazamiento cuando la resolución baja de los 1360 píxeles.

La actualización de estado se lleva a cabo mediante variaciones de la URL que fuerzan una recarga completa de toda la página, aunque es un proceso muy rápido. La interacción es únicamente a través de hipervínculos ubicados en los menús y en las diferentes secciones del juego. Los elementos interactivos se muestran resaltados al pasar por encima (*hover*). La información contextual (*tooltips*) es rica en detalles y tiene asignados estilos visuales integrados en la temática del sitio web.

14 Estrategia en tiempo real con miles de jugadores, más información en https://es.wikipedia.org/wiki/Videojuego_de_estrategia_en_tiempo_real_multijugador_masivo_en_l%C3%ADnea (recuperado el 03/04/2022)

En relación a los recursos, solo define cuatro tipos básicos. Existe una vista de estadísticas de producción donde se muestran los recursos producidos y consumidos por cada elemento del juego. El juego contempla el concepto de flotas de ataque, elementos que pueden enviarse en una actividad de combate. Al ser necesarios para llevar a cabo una actividad, tomarán la consideración de recursos.

El árbol tecnológico es limitado, y se encuentra dividido en cuatro categorías. Sin embargo, cada tecnología se puede investigar múltiples veces, permitiendo ventajas acumulativas. Con respecto a las colas, se define una para cada tipo de actividad (construcción, investigación, etc), aunque están limitadas a un único elemento. En relación al apartado gráfico, muestra ilustraciones de alta calidad, pero completamente estáticas. Las animaciones son anecdóticas, y se limitan a aspectos puramente estéticos. La interfaz está ricamente ornamentada.

El numero de edificios es muy limitado, y solo permite construir un edificio a la vez. Sobre los edificios pueden llevarse a cabo actividades de mejora o desmantelamiento. Por otro lado, el apartado de mensajería es bastante completo, y categoriza los mensajes en dos niveles: tipo y subtipo. Permite marcar mensajes como favoritos y muestra una papelera donde enviar los no deseados. Diferencia la mensajería del juego de los mensajes entre jugadores.

Finalmente, se identifican las siguientes actividades: espionaje, para obtener información de los activos de los enemigos; colonización, para formar nuevos planetas; construcción, para crear o mejorar instalaciones del jugador; investigación, para subir un nivel tecnológico; astillero, para la creación de recursos navales; combate, para el robo de activos a otros jugadores y/o reducir su capacidad ofensiva; comercio, para intercambio de recursos con otros jugadores.

2.2.2 Travian

Figura 17: Vista general de la interfaz de Travian

Travian es un MMORTS para navegador, basado en PHP,Javascript y CSS. El juego está ambientado en las civilizaciones de la antigua roma: celtas, romanos, germanos, etc. Permite al jugador elegir una facción de la que heredará una serie de características base. El juego incorpora el concepto de héroe, un tipo de unidad especial cuyos atributos se usarán en las misiones.

La interfaz de juego no es adaptable, y por debajo de los 1296px aparecen las barras de desplazamiento. En dispositivos móviles el producto intenta centrarse en el 100% de la pantalla sin reducir contenido, lo que produce que se visualiza a una escala muy reducida.

La navegación se basa en variaciones de la URL, que recargan por completo la página realizando nuevas solicitudes al servidor. La interacción se lleva a cabo a través de hipervínculos, botones y elementos gráficos dispuestos en el mapa del jugador. El juego muestra elementos de interfaz avanzados como ventanas modales, desplegables y barras de progreso.

Existen tres tipos de mapas, a diferentes escalas. En primer lugar, el mapa del centro de la aldea, donde el jugador puede añadir y mejorar los edificios administrativos. En segundo lugar, el mapa de los alrededores de la aldea, donde se construyen y mejoran las construcciones de generación de recursos. En tercer lugar, un mapa global que muestra una cuadrícula del mundo, donde se observan las aldeas de todos los jugadores y las propiedades de cada celda.

El juego provee de cuatro tipos de recursos necesarios para la construcción de los edificios y la ejecución de actividades. Además, dispone de diferentes tipos de tropas en función de la facción elegida. Estas tropas son necesarias para llevar a cabo actividades de ataque, asalto y atracos. Las tropas pueden ser destruidas en el transcurso de una actividad o creadas dentro de un edificio, por lo que se consideran recursos también.

No existe el concepto de árbol tecnológico. En su lugar, el juego introduce un tipo de edificio que permite llevar a cabo actividades que mejoran los atributos de las tropas. En relación al apartado gráfico, el juego exhibe ilustraciones de calidad media y baja, sin pretensiones de realismo. Se observan algunas animaciones y transiciones CSS. Con respecto a las colas, cabe señalar solo hay una de tamaño indeterminado, donde todas las actividades se ponen en orden de llegada y esperan su turno de ejecución.

El sistema de mensajería se divide en dos partes. Por un lado los mensajes entre jugadores, donde recibir y enviar mensajes a otros usuarios. Por otro, los informes del juego, un listado con todos los eventos y notificaciones recibidos organizados por categorías.

El juego propone 28 edificaciones diferentes, agrupadas en tres categorías. Cada edificación tiene como objetivo modificar los atributos del jugador o del héroe de este. Además, cada edificio posee

mejoras adicionales que aumentan sus características. Por otro lado, existe un sistema de hitos en el el progreso del jugador hace que estos vayan avanzando. Al llegar a los criterios definidos en cada hito, se otorga una recompensa establecida.

Para terminar, se identifican las siguientes actividades: espionaje, para obtener información de los activos de los enemigos; construcción, para crear o mejorar instalaciones del jugador; investigación, para mejorar las tropas; combate, para robar recursos a otros jugadores; comercio, para intercambiar bienes; fundación, para crear nuevas aldeas; simulación, para estudiar los resultados de un combate previo a su ejecución; exploración, para descubrir información oculta en una celda cercana.

2.2.3 Comparativa

	Ogame	**Travian**
Tecnología	PHP,Javascript, CSS	PHP, Javascript, CSS
Responsive	No	No
Actualización estado	Recarga, activa	Recarga, activa
Elementos interacción	Enlaces	Enlaces, Botones, otros
Temática	Espacial	Medieval
Mapa local	No	Si
Mapa general	No	Si
Estadísticas producción	Si	Si
Investigación	Lineal, acumulable	Lineal, acumulable
Grafismo	Rico en detalles, estático	Pobre, estático
Mensajería	Notificaciones, mensajes	Notificaciones, mensajes
Edificios	Pocos, mejorables	Muchos, mejorables
Cola de actividades	Sin cola, un espacio por tipo	Cola única, sin limite
Actividades	Espionaje Investigación Construcción Colonización Comercio	Espionaje Investigación Construcción Fundación Comercio Simulación Exploración

Tabla 4: Comparativa entre los casos de estudio propuestos

2.3 Análisis DAFO

Debilidades

- Todos los juegos están limitados a un tipo concreto
- Ausencia de contenido gráfico avanzado
- Interacción limitada con el usuario
- Falta de componente social

Amenazas

- *Target* muy acotado
- Baja respuesta por parte de los autores de juegos
- Experiencia de usuario percibida como demasiado similar entre títulos

Fortalezas

- Diseño *responsive*, contenido totalmente adaptado para móviles
- Aplicación de página única, no requiere recargas para mostrar el nuevo contenido
- Actualizaciones en background, el servidor notifica los cambios
- Reglas configurables
- Contenido multimedia configurable
- Tiempos de diseño y publicación extremadamente cortos

Oportunidades

- Creación de *branded content* para otras empresas o marcas (Pla, 2021)
- Usar a los propios usuarios como creadores de contenido

3 Propuesta

Este proyecto pretende definir una plataforma web que, haciendo uso de las tecnologías de vanguardia, sea capaz de aunar las generalidades de los casos de estudio anteriormente enumerados, al tiempo que solventa sus principales carencias. En primer lugar, permitirá la creación y publicación de juegos basados en un paradigma MMORTS. En segundo lugar, facilitará el registro de jugadores y su vinculación con los títulos publicados. En tercer lugar, permitirá a estos disfrutar de la experiencia de juego personalizada según los criterios de su autor. Por último, proveerá de un *backoffice* web para la gestión administrativa de la plataforma.

3.1 Definición de objetivos/especificaciones del producto

A continuación se describen los requisitos de primer nivel:

Web

- Crear una cuenta de jugador
- Identificar a un jugador en la plataforma
- Presentar el listado de juegos
- Presentar la ficha de un juego
- Retomar una partida en un juego

Juego

- Ver el mapa interactivo del jugador
- Ver el mapa interactivo del mundo
- Ver las tecnologías investigadas y disponibles
- Ver los recursos generados y producidos
- Ver los edificios construidos y edificables
- Ver los mensajes y notificaciones recibidos
- Enviar mensajes a otros jugadores
- Llevar a cabo actividades comerciales con otros jugadores
- Llevar a cabo actividades hostiles contra otros jugadores
- Llevar a cabo actividades de exploración en celdas
- Llevar a cabo actividades de espionaje en celdas de otros jugadores
- Llevar a cabo actividades de anexión en celdas desocupadas
- Llevar a cabo actividades de investigación
- Llevar a cabo actividades de construcción

Backoffice

- Alta, baja y modificación de usuarios
- Moderación de jugadores
- Alta, baja, modificación y configuración de juegos
- Alta, baja y modificación de instancias de juego

4 Diseño

4.1 Arquitectura general

La plataforma propuesta se basa en una arquitectura cliente/servidor. En la parte de cliente, cuenta con dos aplicaciones web diferenciadas a cargo de la capa de presentación e interacción. La primera, el *frontoffice*, orientado a los usuarios jugadores. El segundo, el *backoffice*, orientado a los usuarios administradores y creadores de contenido. En la parte de servidor, hay una aplicación encargada de gestionar el *input* de los usuarios y mutar el estado del sistema en consecuencia, así como informar de los cambios a los agentes implicados en cada momento. En este lado, además, se dispone de acceso a un servicio de datos en la nube como motor de persistencia y almacenamiento a largo plazo.

Figura 18: Vista general de la organización de los componentes de software

La plataforma inicia un único servicio que ejecuta la aplicación principal. Esta, durante su inicialización, crea una instancia de un servidor web que se encargará de facilitar el acceso a la API Rest, así como al servicio de *websockets* para comunicaciones asíncronas.

4.1.1 Aplicación

La aplicación es un software escrito en Typescript y transpilado[15] a Javascript, que finalmente es ejecutado por una instancia de node.js. Su misión principal es la de procesar periódicamente las instancias de juego para ejecutar las actividades en cola de los usuarios, así como mover los flujos de recursos de los jugadores. Por otro lado, tiene como misión notificar los eventos *offline* a los jugadores, esto es, facilitarles información sobre aquellos sucesos que son de su interés pero que no tienen forma de conocer.

15 La transpilación o transcompilación es un proceso de traducción que convierte un código fuente de un lenguaje a otro. Más información en Wikipedia https://en.wikipedia.org/wiki/Source-to-source_compiler (recuperado el 02/04/2022)

4.1.1.1 Funcionamiento

El proceso comienza al arrancar la instancia de node.js con la ruta al archivo Javascript que sirve como punto de entrada a la aplicación. En primer lugar lee la configuración, establece la conexión con la base de datos y lee todo el estado previo. En segundo lugar, crea una instancia de la API de juego, encargada de toda la lógica de negocio de la plataforma. En tercer lugar, levanta una instancia del servidor web (express.js) y le suministra una referencia a la API anterior; express.js usará esta referencia para crear una API Rest que expondrá a la capa de cliente, y que servirá como enlace de información.

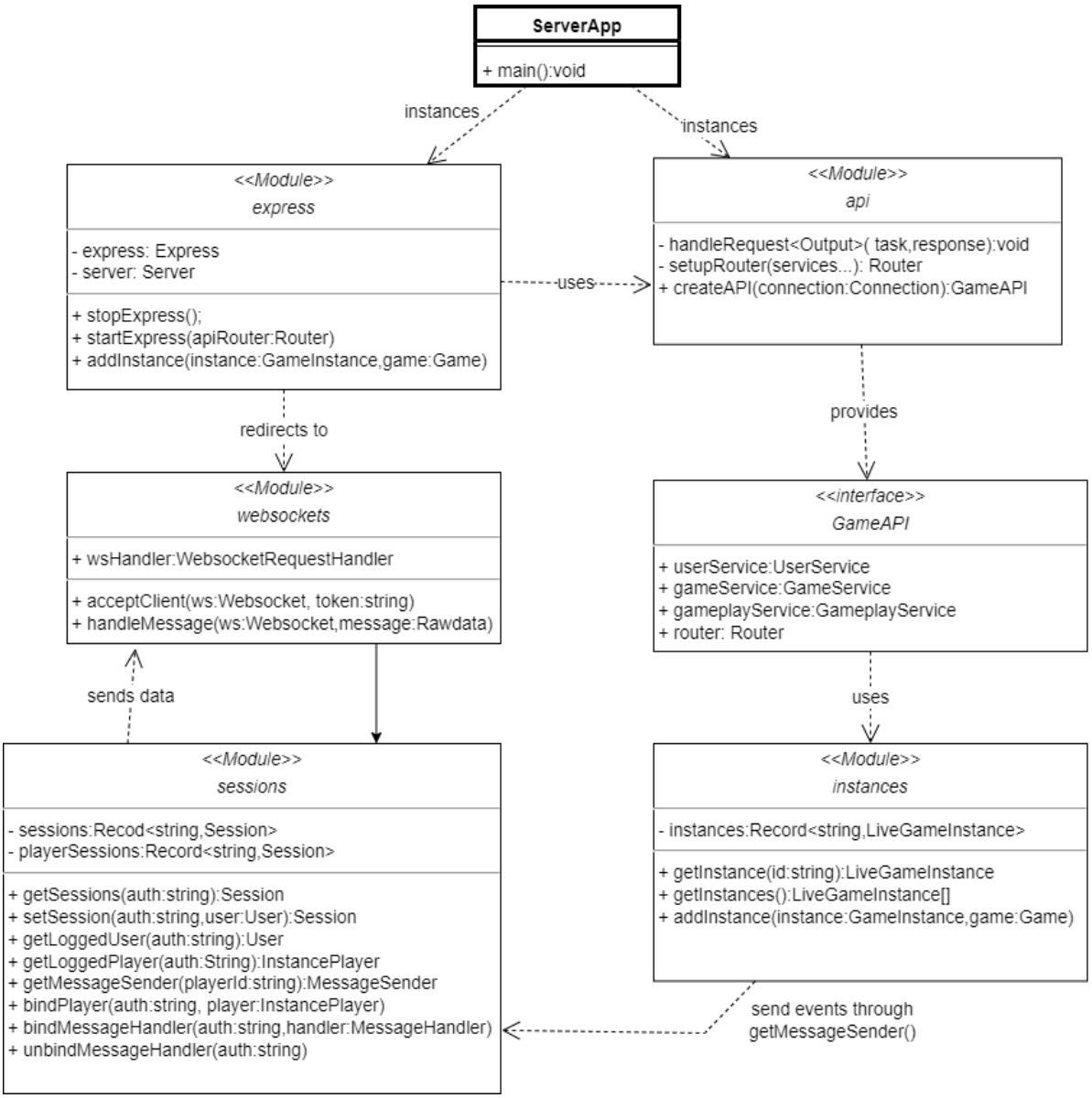


Figura 19: Diagrama UML simplificado de la aplicación ejecutada en el backend

El servidor web se ha configurado con el *middleware*[16] express-ws[17], que inicializa de forma transparente el servidor de *websockets* y redirige todas las solicitudes a través de express, de manera que las conexiones entrantes se puedan gestionar como si de una solicitud HTTP regular se tratara. Para terminar, se interpreta el estado obtenido de la base de datos, se generan las instancias de los diferentes juegos e inicializan los temporizadores que llevan a cabo las diferentes operaciones en *background*[18].

4.1.1.2 Estructura y componentes

Internamente, la aplicación se estructura siguiendo una arquitectura multicapa. Esto permite delimitar la responsabilidad de cada componente, aumentar la cohesión y reducir el acoplamiento.

Capa de control

Aquí se encuentran todas las clases encargadas de instrumentar el arranque de la aplicación, gestionar las solicitudes entrantes y emitir la información saliente.

Capa de servicio

Esta capa tiene como misión encapsular la lógica y las reglas de negocio (Stafford, s. f.). Aquí se encuentran las instancias de los servicios encargados de gestionar las entidades principales del sistema. Al igual que los repositorios, estas son propensas a la repetición de código debido a la multiplicidad de entidades gestionadas, y a que cada una de ellas necesita un conjunto similar de métodos (CRUD[19]). Para paliar este problema, se crea una clase base `BasicRESTService<Type>`, parametrizable por tipo, que permita gestionar las operaciones CRUD/REST[20] básicas de forma estandarizada, aprovechando que el sistema de repositorios será capaz de trabajar de forma paramétrica.

Capa de acceso a datos

Los repositorios sirven como mediadores ente la capa de lógica de la aplicación y la de persistencia de datos (Hieatt & Mee, s. f.), lo que permite el desacoplamiento entre ambas y por tanto facilita el agnosticismo de la base de datos. Aprovechando la potencia del sistema de tipos de Typescript y la flexibilidad de MongoDB, se ha creado un único repositorio parametrizable que encapsule toda la lógica común a todos ellos, reduciendo así la complejidad global del sistema, el número de clases y la cantidad de código similar duplicado.

16 En el contexto de express.js, un middleware es una librería o software de terceros que añade características al framework de manera transparente para el programador. Más información en http://expressjs.com/en/guide/using-middleware.html (recuperado el 04/05/2022)
17 https://www.npmjs.com/package/express-ws (recuperado el 04/05/2022)
18 Procesos en segundo plano
19 Acrónimo de Create, Read, Update, Delete, más información en https://developer.mozilla.org/es/docs/Glossary/CRUD (recuperado el 07/05/2022)
20 Acrónimo de “Transferencia de estado representacional”, hace alusión a un conjunto de técnicas y metodologías que facilitan la eficiencia y escalabilidad en los sistemas distribuidos. Mas información en https://www.redhat.com/es/topics/api/what-is-a-rest-api (Recuperado el 05/04/2022)

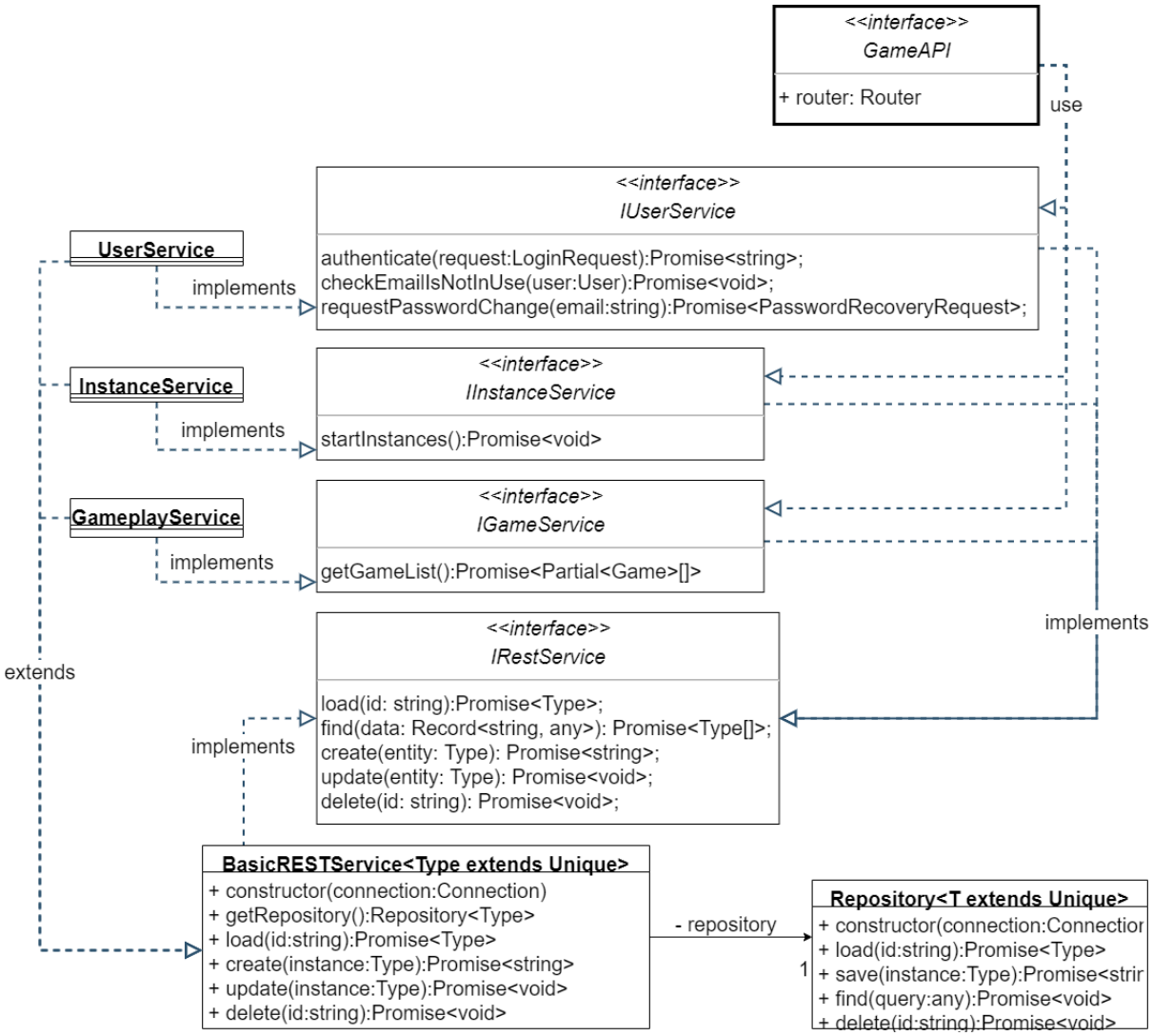


Figura 20: Diagrama UML de los servicios CRUD en backend

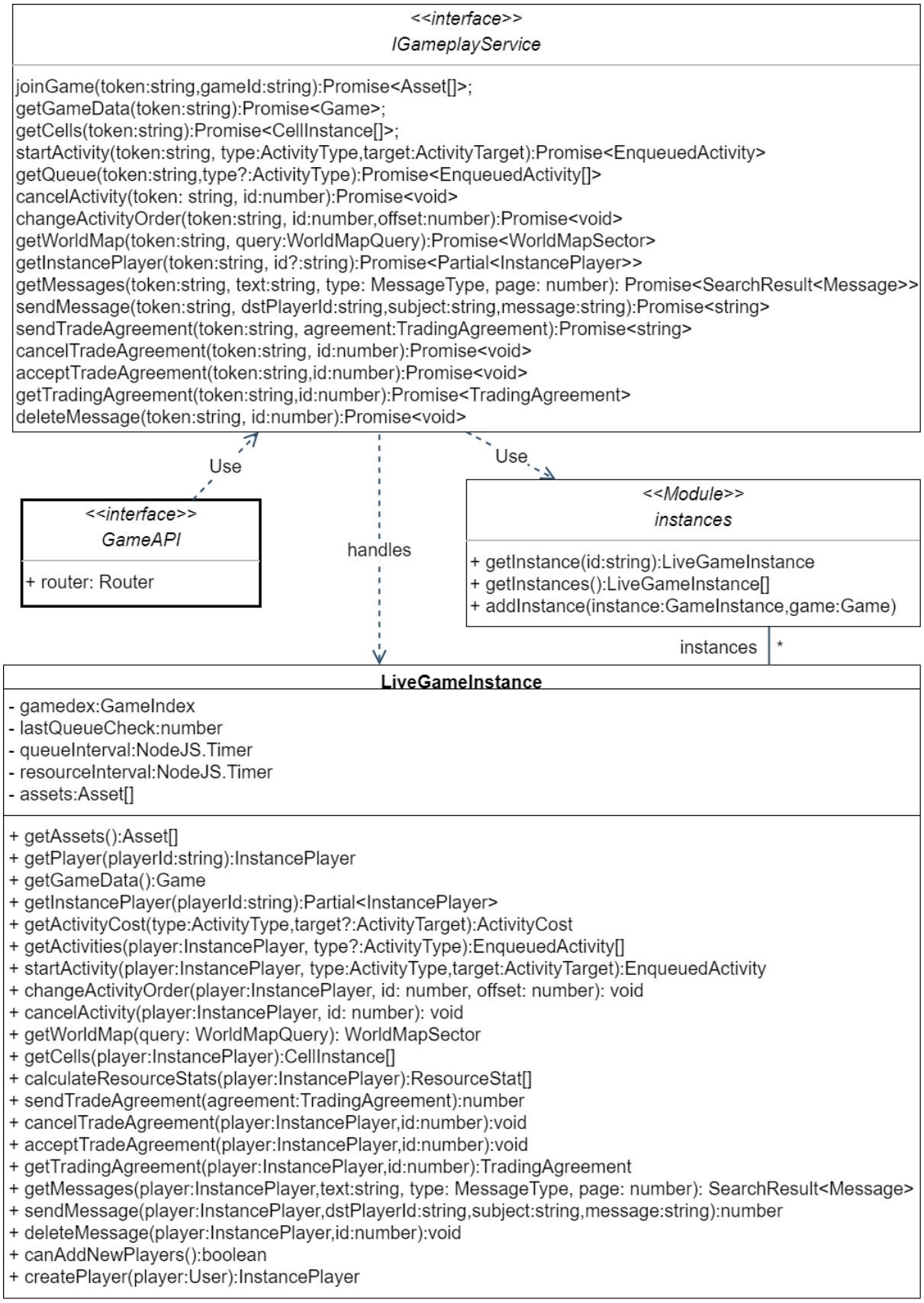


Figura 21: Diagrama UML simplificado de la gestión de instancias

4.1.2 Website

El *website* o *frontoffice,* es un portal web que sirve como punto de entrada y escaparate a los juegos publicados. Entre las responsabilidades de este componente del sistema se encuentra:

- Canalizar el flujo de usuarios, permitiendo el registro y *login.*
- Mostrar información general de los juegos, presentándolos en el home (*featured*) y en el buscador.
- Mostrar información detallada de cada juego en la vista de ficha del juego.
- Ejecutar el juego seleccionado, presentando al jugador todos los elementos de juego.
- Proveer al jugador de los mecanismos de interacción para progresar en el juego.
- Mostrar al jugador los cambios de estado en el sistema.

Tanto el *website* como el cliente de juego que incorpora es una aplicación SPA[21] formada por componentes web construida usando el *framework* Vue.js. Físicamente, se encuentra alojada en un *hosting* separada de la aplicación *backend*.. El control de la visualización y la navegación se lleva a cabo mediante un enrutador (vue-router) basado en el "History API" del navegador. El juego, en cambio, emplea mutaciones del estado global para mostrar los diferentes componentes. Para llevar a cabo esta tarea hace uso de la librería vuex[22], que queda a cargo de la gestión centralizada del estado de la aplicación (de Oliveira et al., 2021).

Cada juego tiene sus propios recursos, y como no es eficiente cargar todos al inicio, se ha diseñado un sistema de carga asíncrona de *assets*[23]. Así pues, en el momento de iniciar un juego, el cliente pregunta a la API qué recursos debe cargar, y esta devuelve una lista JSON[24] con los archivos de medios que deberán ser enlazados. A continuación, se usa un cargador asíncrono de recursos (resource-loader) para gestionar su descarga y procesado. Al terminar, este notifica al juego la finalización para que pueda iniciar su actividad.

Determinados elementos del juego requieren de interacción avanzada: la pantalla del mapa de jugador, del mapa general y del árbol tecnológico requieren llevar a cabo una serie de operaciones que no se pueden llevar a cabo de manera práctica en el navegador: *panning*, *scrolling*, *zoom*, etc. Para poder desarrollar estos componentes se hace uso del componente canvas, que permite un control avanzado sobre lo que se visualiza y su interacción con el usuario. No obstante, aunque permite un mayor control, también añade un orden de complejidad adicional al desarrollo.

21 Acrónimo de *Single Page Application*, o aplicación de página única.
22 https://vuex.vuejs.org/ (Recuperado el 06/05/2022)
23 Desde el punto de vista de la aplicación, un asset es un recurso multimedia (audio, imagen, video) cuya ubicación no se conoce con antelación, o depende del contexto actual.
24 Acrónimo de "Java Script Object Notation", es un formato de intercambio de datos. Más información en https://www.json.org/json-en.html (Recuperado el 06/05/2022)

Figura 22: Componentes principales del juego

4.1.2.1 Estructura y componentes

El *website* consta de un componente principal (`App`) con un hueco o para mostrar otro componente embebido. El controlador de rutas está a cargo de decidir qué elemento se carga en cada momento en función de qué URL haya en el navegador. Los dos componentes que pueden cargarse son el del sitio web (`WebsiteApp`) o el cliente de juego (`GameApp`). Cada uno de estos componentes posee a su vez otros componentes, que mostraran en cada momento en función del estado de la aplicación.

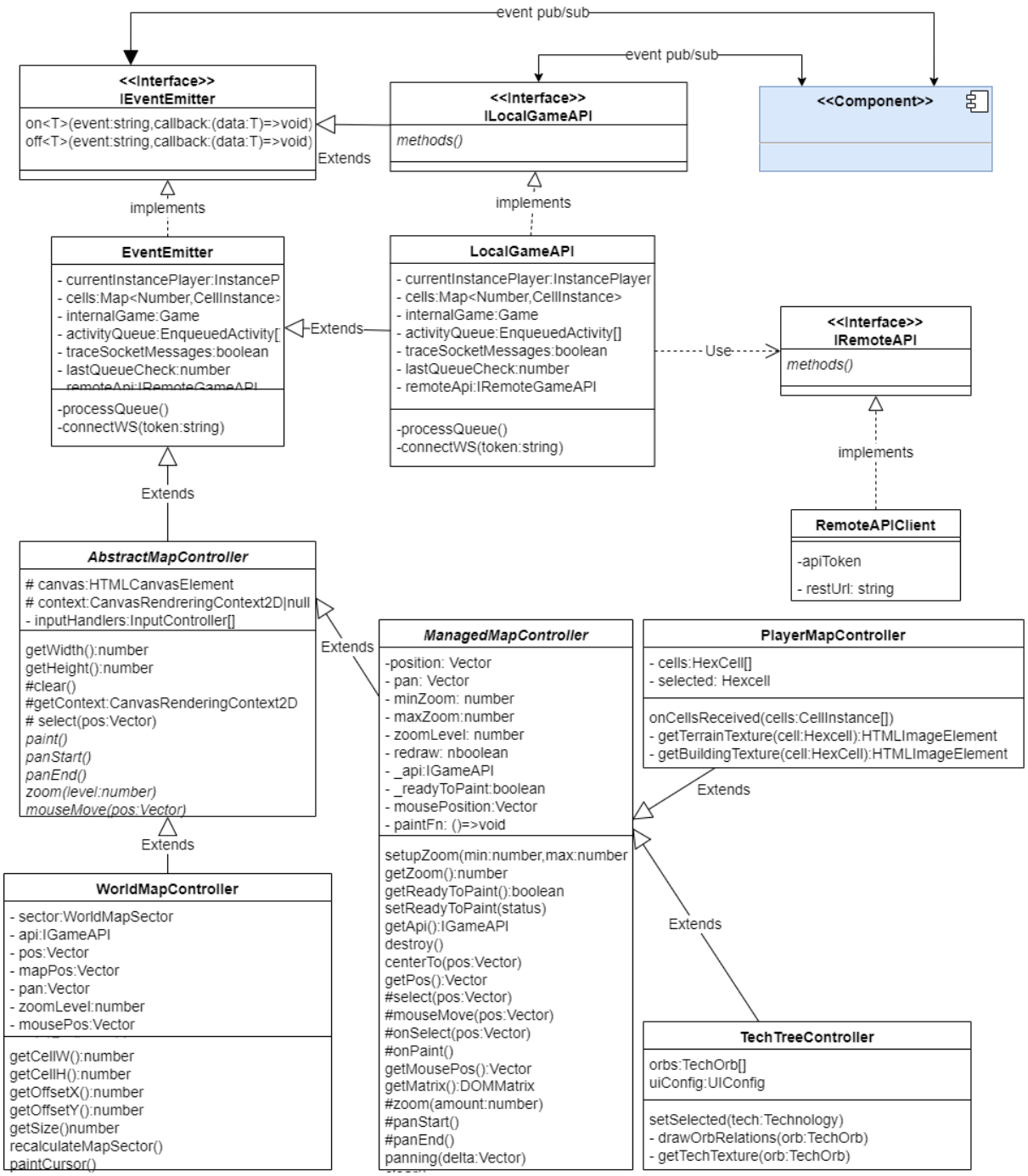


Figura 23: Diagrama UML de los controladores de interacción

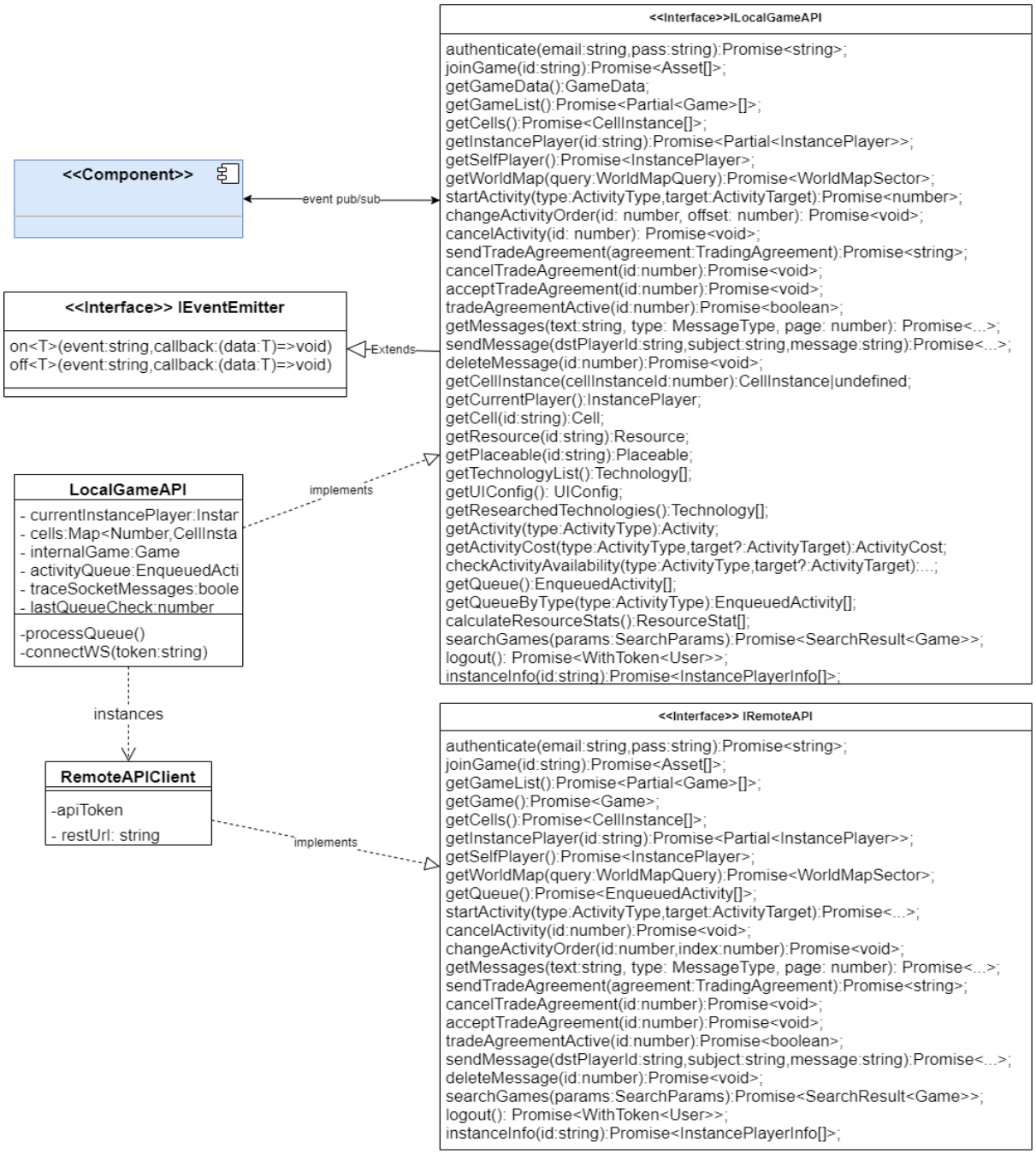


Figura 24: Diagrama UML simplificado de la API del juego

4.1.2.2 Interfaz de usuario mejorada

Una de las características de los juegos publicados es la capacidad que ofrecen a sus autores de personalizar el aspecto y la interfaz. Para controlar de forma unificada el aspecto de la interfaz que se presenta al jugador, se ha diseñado una jerarquía completa de componentes o *widgets*, que tomarán el control de la interacción con el jugador. Entre otros, se pueden encontrar:

- `UIPane` - Panel flex que actúa como contenedor de otros componentes.
- `UIModal` - Ventana en primer plano que limita la interacción el jugador al elemento que se le presenta.

- `UIFlex` - Permite distribuir los componentes siguiendo un esquema CSS flex parametrizado.
- `UIButton` - Elemento de interacción, permite texto, iconos, diferentes estados y eventos de interacción.
- `UIList` - Presenta una lista de elementos seleccionables, provee de eventos de selección.
- `UIIcon` - *Widget* para la presentación unificada de iconos.
- `UILabel` - *Widget* para la presentación unificada de etiquetas de texto.
- `UIDropdown` – *Widget* que presenta una lista desplegable que permite elegir una opción.

4.1.3 Backoffice

El *backoffice* es una aplicación web basada en componentes a cargo de la gestión administrativa de los elementos involucrados en la plataforma. Entre sus responsabilidades se encuentra:

- La gestión de los jugadores y usuarios:altas, bajas, consultas, permisos y moderación.
- La gestión de las instancias: altas, bajas, consultas, modificaciones, inicios y paradas.
- La gestión de los juegos: altas, bajas, consultas y modificaciones.

La aplicación está modelada como una SPA basada en componentes. Para su desarrollo se ha empleado las tecnologías previamente utilizadas en el proyecto: vue.js y vue-router. Como novedad, incorpora Bootstrap 5 para la capa de estilo e interacción. Físicamente está alojada como contenido estático en el mismo hosting que el *frontoffice*.

El *backoffice* organiza la información en secciones que presentan las opciones que se pueden llevar a cabo sobre cada entidad. Cada una hace uso de tres componentes que dan acceso a las operaciones CRUD: `EntityEdit`, `EntityView` y `EntitySearch`. Estos son agnósticos de las entidades que manejan gracias a un enfoque inspirado en el patrón de diseño “método factoría[25]”, por el que se delega en la instancia de `SectionHandler` la responsabilidad de decidir en cada momento qué pintar, con qué datos y cómo gestionar la interacción.

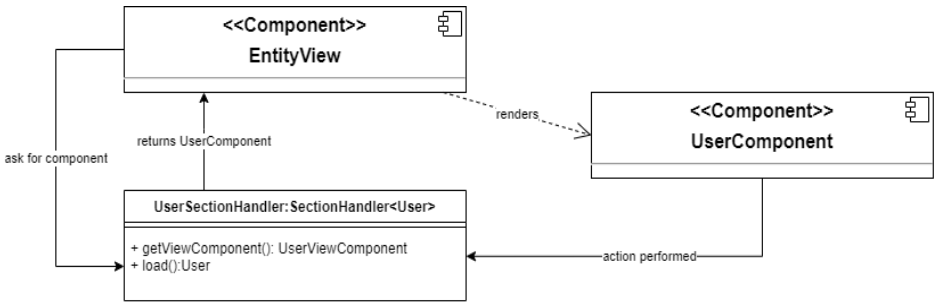


Figura 25: Intercambio de información entre un componente CRUD y un manejador de sección

25 Más información en https://refactoring.guru/es/design-patterns/factory-method (Recuperado el 31/05/2022)

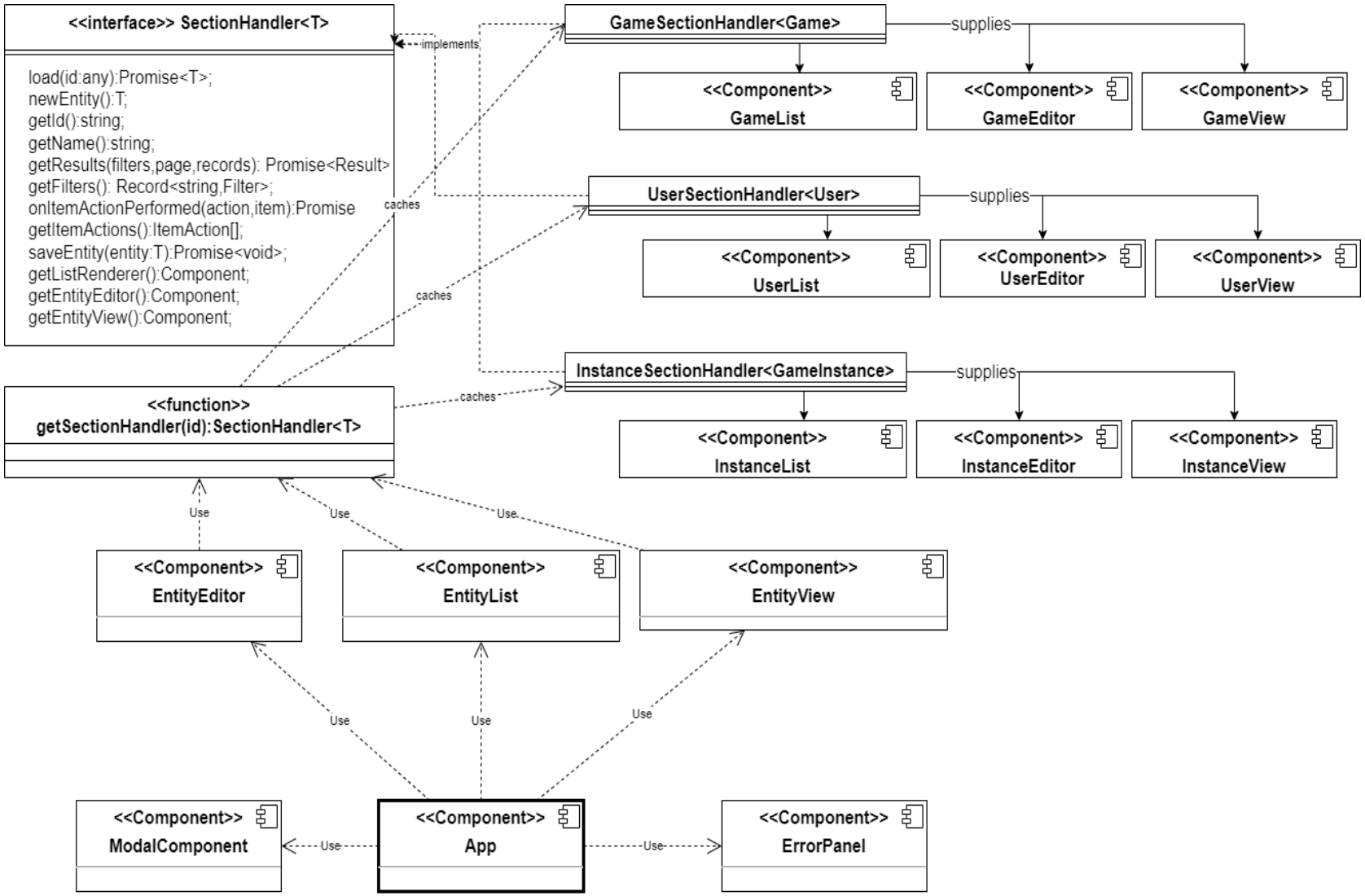


Figura 26: Diagrama UML de la arquitectura de componentes del *backoffice*

El editor de juegos merece una mención aparte, ya que la estructura de datos de la entidad subyacente es bastante más compleja que las otras dos. Presenta los diferentes aspectos del juego organizados en secciones, donde cada una da acceso a un componente de pestañas plegables que distribuye la información en subsecciones. A su vez, cada subsección carga un componente responsable de gestionar una parte del modelo de datos. En total hay 19 componentes diferentes de edición, tal y como se muestra en la siguiente figura:

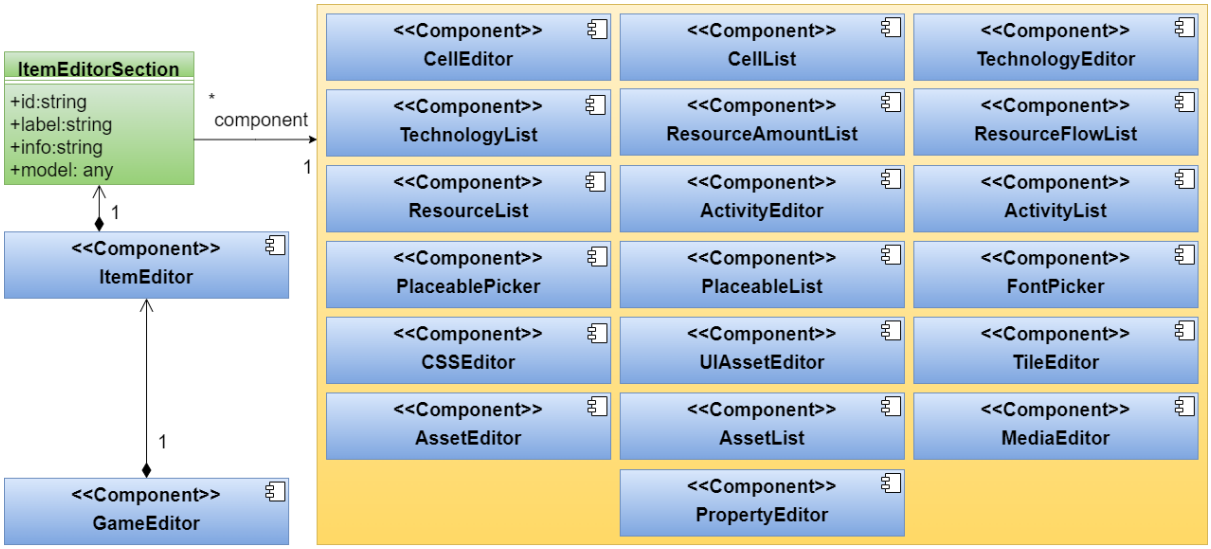


Figura 27: Relación de componentes empleados en la edición de juegos

4.1.3.1 Estructura y componentes principales

App

Componente principal, a cargo de orquestar al resto y mantener cualquier estado de aplicación. Entre sus responsabilidades se encuentra la presentación del menú de usuario, el encabezado y el componente que haya configurado para la URL activa en cada momento. Este cometido se lleva a cabo mediante la librería vue-router, que facilita un componente `<router-view />` que apunta al componente deseado en cada instante.

EntityView

Componente a cargo de representar una entidad principal para consultar sus datos. Su responsabilidad se limita a escoger la implementación de `SectionHandler` adecuada a la sección que se consulta, cargar los datos de la entidad que este facilita y mostrar el componente adecuado junto con los datos de entrada obtenidos.

EntityEdit

Componente a cargo de mostrar el formulario o vista de edición de una entidad principal. Se apoya en una implementación concreta de `SectionHandler` para determinar si debe crear una nueva enti-

dad o cargar una existente para luego inyectarla en el componente adecuado. Este último, además de la entidad recibe la propia sección, de forma que puede comunicarse con ella directamente para gestionar la interacción con el usuario.

EntitySearch

Este componente está a cargo de generar los filtros de búsqueda a partir del manejador de la sección actual, obtener la lista de resultados correspondientes al filtro vigente y facilitar estos al componente adecuado para su presentación.

SectionHandler

Este elemento es una interfaz implementada por tres clases diferentes, una por cada entidad principal: `UserSectionHandler`, `GameSectionHandler` e `InstanceSectionHandler`. La implementación vigente en cada momento se determina en función de la sección activa, y define qué entidades se representarán, qué componentes se encargarán de manipularlas y cómo debe gestionarse la interacción con el usuario.

Vistas de elemento de lista (*List)

Estos componentes son los que serán suministrados a `EntitySearch` a la hora de pintar los resultados. La elección de uno u otro componente depende de la implementación de `SectionHandler` que se use en cada momento, y por tanto del tipo de sección que se visualiza.

Formularios de edición (*Editor)

Componentes suministrados a `EntityEdit` que serán mostrados al hacer clic en enlace “Crear” de cada sección, el botón "Añadir" del componente `EntitySearch` o el menú contextual "Editar" de cada elemento.

Vistas de presentación de entidades (*View)

Vistas empleadas para representar las entidades implicadas en cada sección. Esta vista se carga cuando el usuario ejecuta la acción "ver" desde la lista de resultados, y sustituye al componente de sección variando la URL del navegador. Al igual que con el resto de componentes relacionados con `SectionHandler`, el tipo de componente que se facilita depende de la implementación.

4.1.4 Base de datos

El servidor de bases de datos es un software independiente de la aplicación principal, capaz de almacenar y recuperar la información de forma eficiente y transparente. Su misión es servir de almacén de respaldo para las instancias de juego que, aunque viven en la memoria del servidor, necesitan un soporte a largo plazo en caso de desastre. Además, opera como almacenamiento principal para el resto de entidades: juegos, instancias y usuarios.

MongoDB se rige mediante una arquitectura cliente/servidor, de manera que es necesario tener una instancia del servicio `mongod` activo y accesible a través de la red. Con el servicio activo y credenciales apropiadas para acceder, se crea una instancia del cliente de comunicaciones en el lado de la aplicación. Este cliente es el encargado de abstraer a la capa de lógica de todo el proceso de comunicaciones con el servidor de la base de datos. En el lado de la aplicación este cliente nunca será visible. En su lugar, queda encapsulado en un objeto `Connection`, que será suministrado a los repositorios a fin de convertir las solicitudes de consulta de la aplicación en consultas específicas del protocolo MongoDB.

4.1.4.1 Esquema de base de datos

Al contrario que los sistemas de gestión de bases de datos normativos, organizados en tablas y tuplas, MongoDB se estructura en colecciones, que se corresponden con conjuntos indexados de documentos. Cada documento es un objeto JSON almacenado en un formato binario denominado BSON (MongoDB Documentation Team, s. f.). Esta característica de la base de datos es especialmente interesante, ya que todo el modelo de datos de la aplicación se basa en interfaces Typescript implementadas en objetos con traducción inmediata a JSON. Esto significa que la carga y almacenamiento de datos desde y hacia la aplicación es directo, sin necesidad de efectuar transformaciones intermedias ni realizar mapeos entre tipos de datos.

Cabe señalar que MongoDB no impone ninguna restricción a la estructura que deben seguir los documentos de una misma colección. En cualquier caso, para facilitar la normalización de las operaciones en la aplicación, todos los documentos de una misma colección compartirán el mismo tipo de dato, y por tanto idéntico esquema. La flexibilidad de los esquemas por colección, unido al hecho de que la mayor parte de la información de la plataforma está pensada para residir habitualmente en la memoria principal, son los motivos por el que se ha escogido el esquema de datos que se presenta en la imagen a continuación:

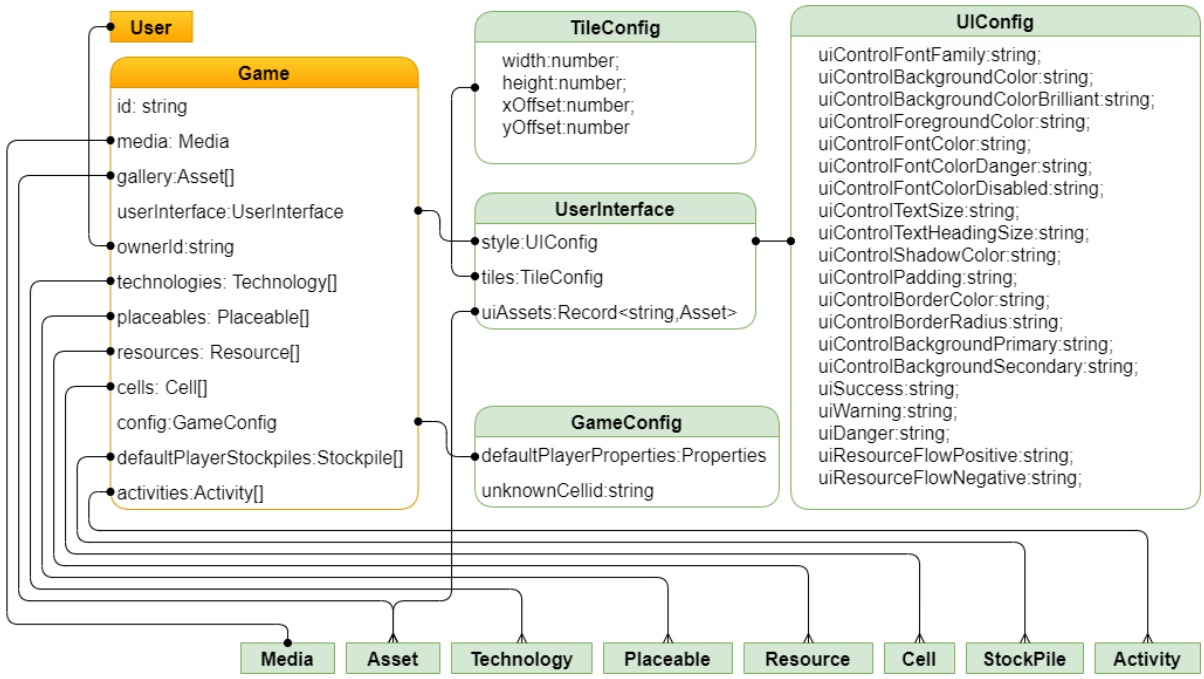


Figura 28: Esquema de datos, relaciones de primer nivel (1)

En las figuras 28 y 29 se puede observar que, aunque hay una veintena de tipos involucrados, sólo existen tres clases con colección propia. El resto se almacenan como subdocumentos embebidos dentro de cada colección principal. Este mecanismo presenta la ventaja de que evita el problema N+1 (Ignjatovic, 2021), ya que todos los elementos de la relación se obtienen junto con el principal. Sin embargo, también impide que el esquema pueda normalizarse (Wikipedia contributors, 2022), lo que produciría datos duplicados e incoherencias en el modelo de datos.

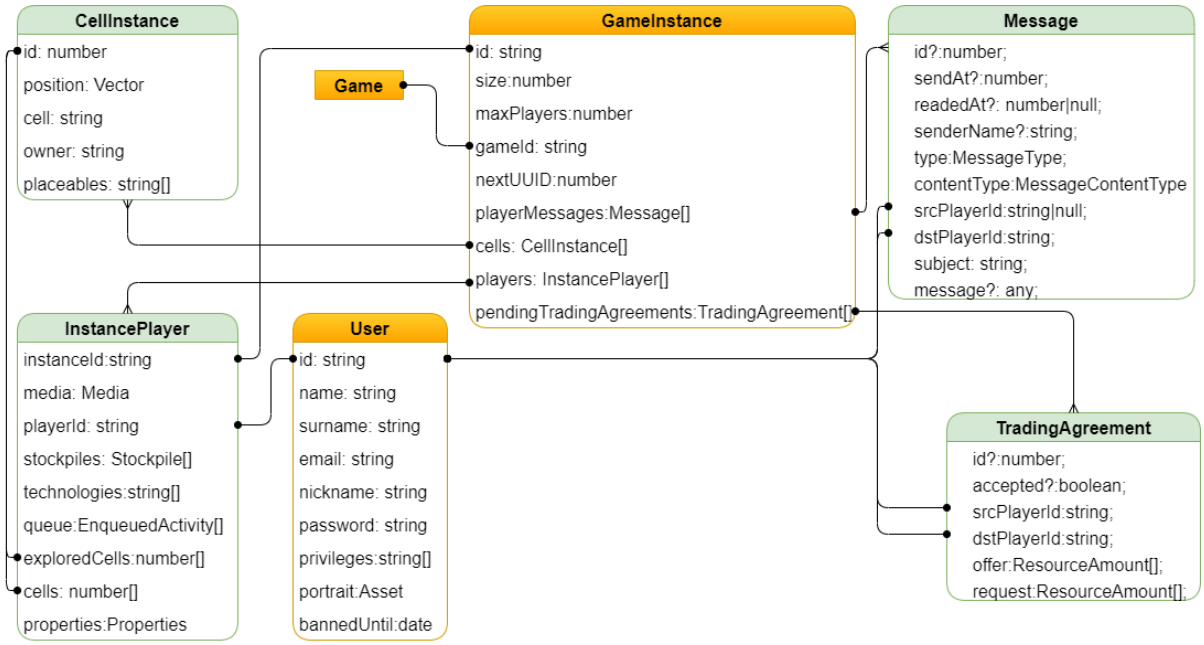


Figura 29: Esquema de datos, relaciones de primer nivel (2)

Así pues, para aprovechar las ventajas y sortear los inconvenientes se sigue un enfoque híbrido a la hora de modelar las relaciones de los documentos, en función de la naturaleza de las relaciones que intervienen:

- Las relaciones de asociación entre objetos se guardarán como referencias a los identificadores involucrados.
- Las relaciones de composición entre objetos se guardarán como colecciones embebidas en el documento principal.

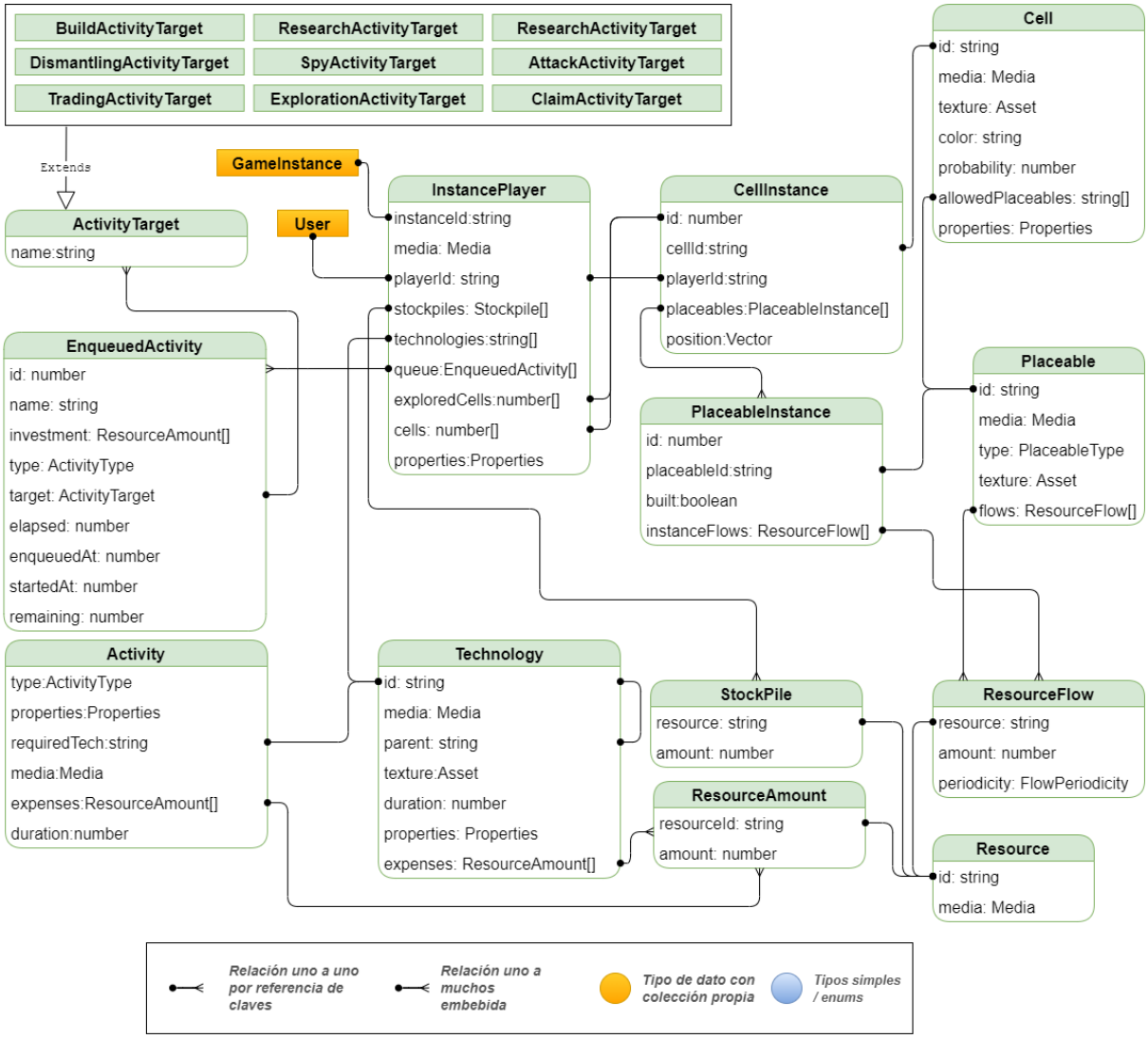


Figura 30: Esquema de datos, relaciones de segundo nivel

Con todo, aún siguiendo estas dos reglas habrá escenarios en los que sea necesario recuperar la información asociada a un identificador. Por ejemplo, para obtener el nombre del recurso asociado a un almacén (`stockpile`), será necesario cargar la instancia de juego que contiene el `stockpile.resourceId`.

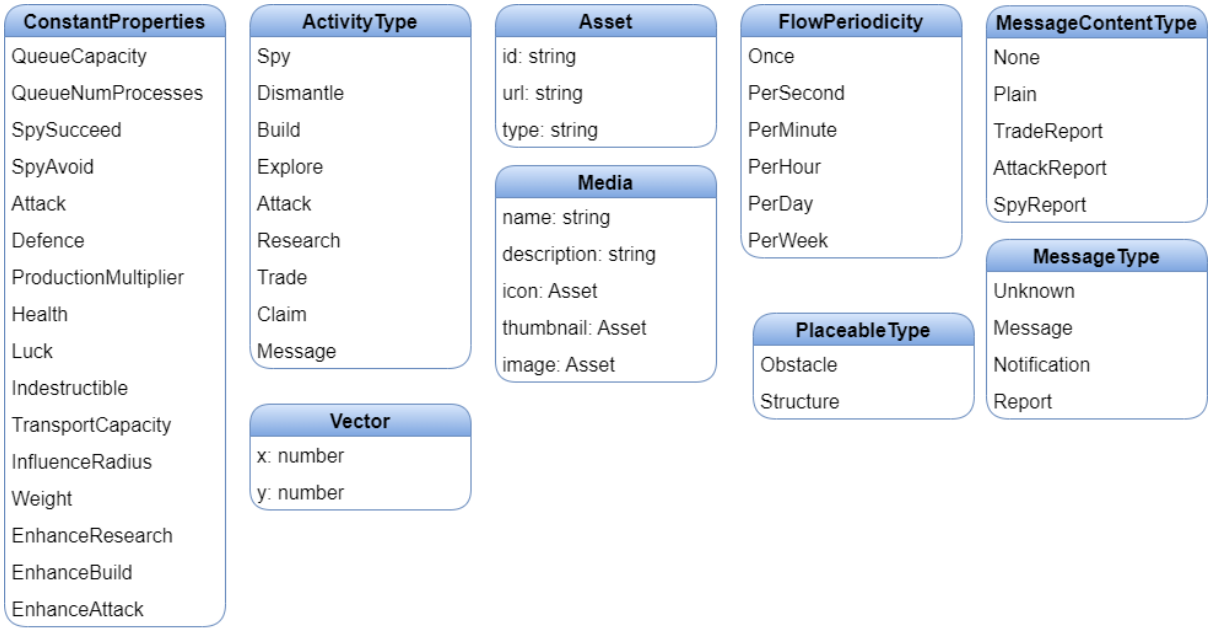


Figura 31: Esquema de datos, enumerados y estructuras simples

4.2 Arquitectura de la información y diagramas de navegación

4.2.1 Esquema de organización de la AI

Dado que la aplicación trabaja intensivamente con colecciones de entidades, tiene sentido que la arquitectura de la información emergente se organice según un esquema eminentemente temático. Además, dado que abundan las relaciones de agregación y composición, la estructura de organización predominante será la jerárquica, especialmente en el menú principal del sitio web y del *backoffice*.

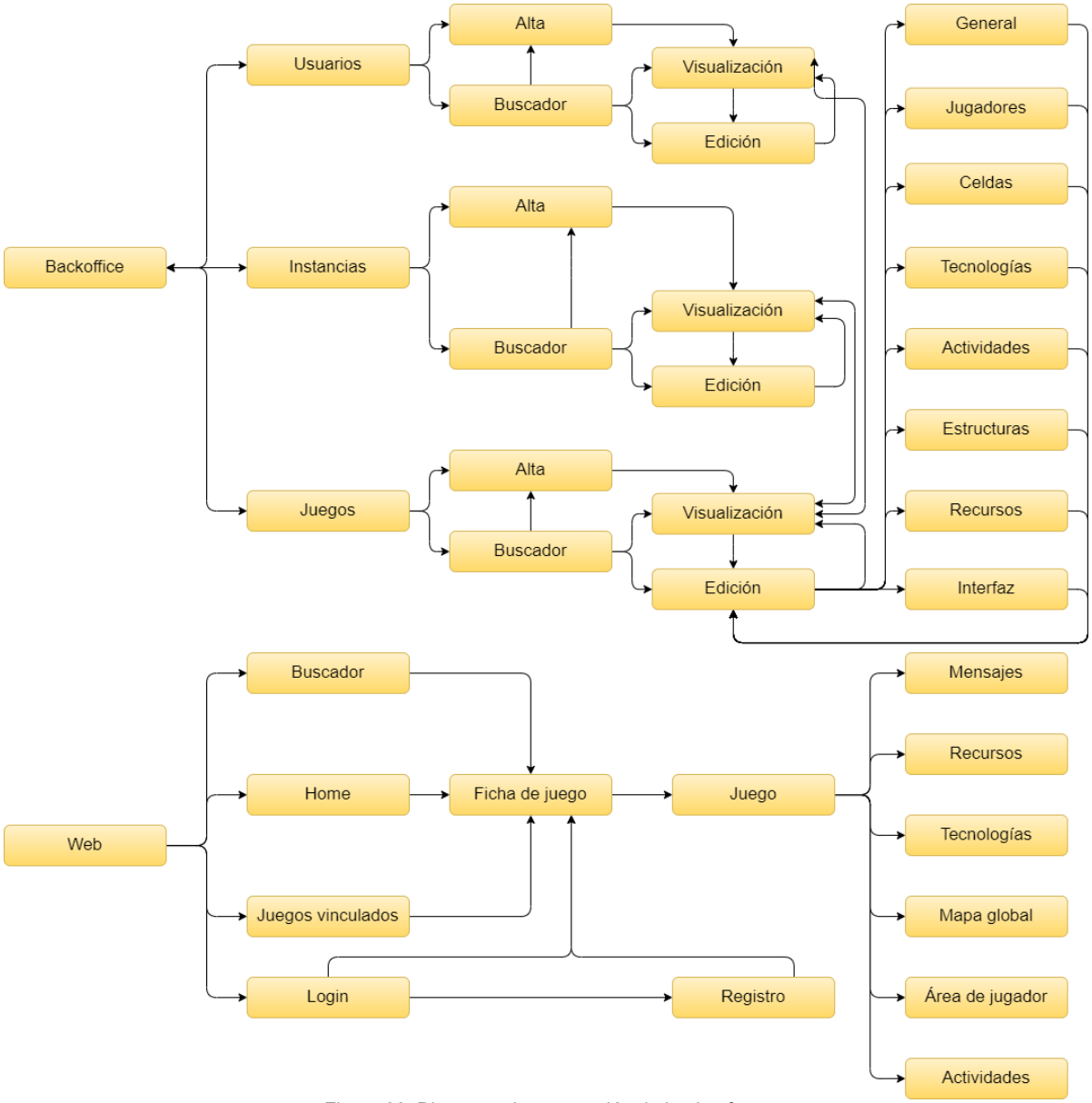


Figura 32: Diagrama de navegación de la plataforma

Por otro lado, el componente del juego aporta una nueva dimensión a la arquitectura de la información del producto. Este componente, al contrario que el resto del sistema, incorpora esquemas de organización geográficos, patentes en el mapa del mundo, el mapa del jugador y el árbol de tecnologías. Aquí, la información está organizada ya no por sus características propias, sino por la posición que ocupa respecto del resto de unidades de información.

En relación a los sistemas de navegación, tanto el *backoffice* como el *frontoffice* presentan navegación global (Rosenfeld et al., 2002) y buscadores basados en las propiedades de la entidad relacionada. El *frontoffice*, además, presenta navegación contextual, especialmente destacable en el omnipresente panel informativo del componente de juego y los mapas interactivos

4.2.2 API Rest

Para facilitar la interoperabilidad y la escalabilidad de la plataforma, el *backend* expone toda su funcionalidad a través de una API Rest accesible a través del propio servidor web de express.js. La identificación de los usuarios y jugadores se lleva a cabo mediante un token de sesión en la cabecera `Authorization`. A continuación se detallan todas las operaciones soportadas, así como los tipos de datos de entrada y salida.

4.2.2.1 Sesiones

/GET	**/ping**	
	Eco de ping. No hace nada salvo devolver la cadena PONG, indicando que la api está activa.	
	URL Params	-
	Request body	-
	Response body	200 OK – text/plain El servidor está activo

/POST	**/sessions/login**	
	Envía al backend un usuario y una clave para iniciar la sesión y obtener un token de autenticación	
	URL Params	-
	Request body	`LoginRequest` – Objeto con el usuario y la clave del jugador
	Response body	200 OK – application/json `WithToken<User>` – Combinación de datos de usuario y token de sesión 401 Unauthorized – text/plain En caso de que las credenciales no sean válidas

/DELETE	**/sessions**
	Destruye la sesión de un usuario. Generalmente se enviaría un identificador en la URL, pero sería redundante ya que debe coincidir con el api token suministrado en el encabezado.
Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	{id} – Identificador de la sesión (token)
Request body	-
Response body	200 OK – text/plain En caso de éxito, sin contenido. 401 Unauthorized – text/plain En caso de que el usuario no haya iniciado sesión 404 Not found– text/plain En caso de que no se haya encontrado la sesión

4.2.2.2 Información dentro del juego

/GET	**/instance/gamedata**
	Devuelve un objeto que contiene la definición de los diferentes elementos, reglas y parametrización UI del juego en el que el jugador ha iniciado la sesión.
Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	-
Request body	-
Response body	200 OK – application/json `Game` – Instancia de `Game` con toda la información de tecnología, recursos, tipos de celdas, configuración, interfaz, etc. 401 Unauthorized – text/plain En caso de que el usuario no haya iniciado sesión

/GET	**/instance/players/self**
	Devuelve la información del usuario vinculado con la sesión actual
Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	-
Request body	-
Response body	200 OK – application/json

`InstancePlayer` – Objeto con todas las propiedades del jugador vinculado: actividades en cola, tecnologías investigadas, celdas exploradas, celdas anexionadas, etc.
401 Unauthorized – text/plain
En caso de que el usuario no haya iniciado sesión

/GET **/instance/players/:id**

Devuelve información parcial de un usuario

Headers Authorization {token} – Token obtenido al iniciar la sesión
URL Params -
Request body -
Response body 200 OK – application/json
`Partial<InstancePlayer>` – Objeto con las características multimedia básicas de un jugador: nombre, imagen, descripción, etc.
401 Unauthorized – text/plain
En caso de que el usuario no haya iniciado sesión
404 Not found – text /plain
En caso de que no se haya encontrado el jugador requerido

/GET **/instance/cells**

Devuelve una colección con todas las celdas que entran dentro del radio de influencia del jugador vinculado con la sesión.

Headers Authorization {token} – Token obtenido al iniciar la sesión
URL Params -
Request body -
200 OK – application/json
`CellInstance[]` – Colección de objetos con la información de las celdas encontradas: propietario, estructuras, posición y tipo de terreno.
401 Unauthorized – text/plain
En caso de que el usuario no haya iniciado sesión

/GET **/instance/messages**

Obtiene los mensajes dirigidos a un jugador, paginados y filtrados por texto y tipo de mensaje.

Headers Authorization {token} – Token obtenido al iniciar la sesión

URL Params	`text:string` – Fragmento de texto para filtrar `type:number` – Tipo de mensaje para filtrar `page:number` – Número de página a partir de la cual comenzar
Request body	-
	200 OK – application/json `SearchResult<Message>` – Objeto que contiene los resultados (`Message[]`) y los metadatos de la búsqueda (n.º de resultados, página actual y n.º de páginas). 401 Unauthorized – text/plain En caso de que el usuario no haya iniciado sesión

/DELETE /instance/messages/:id

Borra un mensaje asociado con un jugador

Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	{id} – Identificador del mensaje que se pretende eliminar
Request body	-
Response body	200 OK – text/plain En caso de éxito, sin contenido. 401 Unauthorized – text/plain En caso de que el usuario no haya iniciado sesión 403 Forbidden– text/plain En caso de que el mensaje no sea propiedad del jugador 404 Not found– text/plain En caso de que el mensaje solicitado no exista

/POST /instance/messages

Envia un mensaje a otro jugador.

Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	-
Request body	application/json `Message` – Objeto con la información necesaria para hacer llegar el mensaje: destinatario, contenido y asunto.
Response body	200 OK – text/plain En caso de éxito se devuelve el identificador del mensaje recién creado. 401 Unauthorized – text/plain En caso de que el usuario no haya iniciado sesión 409 Conflict– text/plain En caso de que la información del mensaje no permita su inserción

(por ejemplo, destinatario inexistente).

/POST **/instance/queue**

Agrega una actividad a la cola de actividades.

Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	-
Request body	application/json `type:ActivityType` – identificador del tipo de actividad `target:ActivityTarget` – Objeto con los datos para iniciar la actividad.
Response body	200 OK – application/json `EnqueuedActivity` – En caso de éxito se devuelve toda la información de la tarea encolada: tiempo estimado, tiempo transcurrido, target, recursos invertidos, etc. 401 Unauthorized – text/plain En caso de que el usuario no haya iniciado sesión 409 Conflict– text/plain En caso de que la información suministrada no sea válida de cara a encolar la actividad (por ejemplo, capacidad máxima de cola superada)

/GET **/instance/queue**

Obtiene todas las actividades en cola.

Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	-
Request body	-
Response body	200 OK – application/json `EnqueuedActivity[]` – En caso de éxito se devuelve una colección con la información de todas las actividades en cola. 401 Unauthorized – text/plain En caso de que el usuario no haya iniciado sesión

/PATCH **/instance/queue/:id**

Cambia el orden de una actividad en la cola de actividades.

Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	{id} – Identificador de la actividad cuyo orden se desea cambiar
Request body	application/json

`offset:number` – Número de posiciones que se desea bajar (positivo) o subir (negativo) en el orden con respecto a su posición actual.

Response body 200 OK – application/json
En caso de éxito devuelve una respuesta vacía.
401 Unauthorized – text/plain
En caso de que el usuario no haya iniciado sesión
404 Not found– text/plain
En caso de que la actividad seleccionada no se encuentre.
409 Conflict– text/plain
En caso de que la información suministrada no sea válida de cara a modificar la actividad: si la actividad está en curso, si desplaza a una actividad en curso, si se baja la prioridad siendo la última, etc.

/DELETE /instance/queue/:id

Cancela una actividad encolada.

Headers Authorization {token} – Token obtenido al iniciar la sesión

URL Params {id} – Identificador de la actividad que se desea cancelar.

Request body -

Response body 200 OK – application/json
En caso de éxito devuelve una respuesta vacía.
401 Unauthorized – text/plain
En caso de que el usuario no haya iniciado sesión
404 Not found– text/plain
En caso de que la actividad no exista.

/GET /instance/map?sector=x1,y1,x2,y2

Obtiene la sección del mapa global delimitada por dos vectores.

Headers Authorization {token} – Token obtenido al iniciar la sesión

URL Params `x1:number` – Coordenada x de la esquina superior izquierda
`y1:number` – Coordenada y de la esquina superior izquierda
`x2:number` – Coordenada x de la esquina inferior derecha
`y2:number` – Coordenada y de la esquina inferior derecha

Request body -

Response body 200 OK – application/json
`WorldMapSector` – Objeto con la información relevante de la sección del mapa solicitada: ancho y alto de la sección, lista de jugadores encontrados e información parcial de las celdas contenidas.

401 Unauthorized – text/plain
En caso de que el usuario no haya iniciado sesión
400 Bad request– text/plain
En caso de que los parámetros suministrados no sean válidos

/POST /instance/trades

Crea un acuerdo comercial con otro jugador.

Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	-
Request body	application/json `TradingAgreement` – Objeto con la información necesaria para crear el acuerdo: destinatario, oferta y requerimientos.
Response body	200 OK – text/plain En caso de éxito se devuelve un string con el identificador del acuerdo recién creado. 401 Unauthorized – text/plain En caso de que el usuario no haya iniciado sesión 409 Conflict– text/plain En caso de que la información suministrada no permita la creación del acuerdo: destinatario inexistente, valores negativos, recursos insuficientes, etc.

/DELETE /instance/trades/:id

Elimina (y cancela) un acuerdo comercial existente.

Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	{id} – Identificador del acuerdo que se desea cancelar.
Request body	-
Response body	200 OK – text/plain En caso de éxito devuelve una respuesta vacía. 401 Unauthorized – text/plain En caso de que el usuario no haya iniciado sesión 403 Forbidden – text/plain En caso de que el jugador no pueda borrar el acuerdo comercial elegido. 404 Not found– text/plain En caso de que el acuerdo comercial no exista. 409 Conflict– text/plain En caso de que el jugador origen o destino no exista

/GET	**/instance/trades/:id**
	Obtiene los datos de un tratado comercial
Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	{id} – Identificador del acuerdo.
Request body	-
Response body	200 OK – application/json `TradingAgreement` – Objeto con los datos del acuerdo 401 Unauthorized – text/plain En caso de que el usuario no haya iniciado sesión 403 Forbidden – text/plain En caso de que el jugador no esté involucrado en el acuerdo comercial elegido. 404 Not found– text/plain En caso de que el tratado comercial no exista o haya sido borrado.

/PATCH	**/instance/trades/:id**
	Marca un tratado comercial como "aceptado".
Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	{id} – Identificador del acuerdo.
Request body	application/json `Partial<TradingAgreement>` – Información parcial del acuerdo comercial, solo se aceptará un campo `accepted` con valor `true`
Response body	200 OK – text En caso de éxito devuelve una respuesta vacía. 401 Unauthorized – text/plain En caso de que el usuario no haya iniciado sesión 403 Forbidden – text/plain En caso de que el jugador no pueda alterar el acuerdo comercial elegido. 404 Not found– text/plain En caso de que el tratado comercial no exista o haya sido borrado. 409 Conflict - text/plain En caso de que se intente cambiar un atributo diferente del estado del acuerdo, o que el estado del acuerdo no sea válido

4.2.2.3 Gestión de usuarios

/GET	**/users?q=:criteria**
	Ejecuta una búsqueda de usuarios empleando los filtros definidos en los parámetros

de entrada.

Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	`SearchParams / b64str` – Objeto de tipo `SearchParams`, convertido a string, serializado a base64 y enviado como parámetro de consulta en la URL. Contiene los criterios de búsqueda de usuarios en formato MongoDB.
Request body	-
Response body	200 OK – application/json `SearchResult<User>` – Resultados paginados de la búsqueda correspondientes al índice solicitado. 401 Unauthorized – text/plain En caso de que el usuario no haya iniciado sesión 403 Forbidden – text/plain En caso de que el usuario no disponga del permiso `ListUsers`

/GET /users/:id

Obtiene los datos de un usuario

Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	{id} – Cadena de texto con el identificador del usuario
Request body	-
Response body	200 OK – application/json `User` – Información del usuario 401 Unauthorized – text/plain En caso de que el usuario no haya iniciado sesión 403 Forbidden – text/plain En caso de que el usuario no disponga del permiso `ViewUser`

/GET /users/:id/games

Obtiene los identificadores de los juegos a los que un usuario se encuentra vinculado.

Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	{id} – Cadena de texto con el identificador del usuario
Request body	-
Response body	200 OK – application/json `InstanceInfo[]` – Lista de estructuras con información de los juegos vinculados. 401 Unauthorized – text/plain En caso de que el usuario no haya iniciado sesión

	403 Forbidden – text/plain En caso de que se estén consultando los datos de un jugador diferente al vinculado a la sesión

/POST /users/check

Comprueba si los datos que se envían en el cuerpo de la solicitud son válidos de cara a registrar a un usuario

Headers	-
URL Params	-
Request body	User – Una instancia de un usuario con los valores del formulario de registro.
Response body	200 OK – application/json `Record<string,string>` – Conjunto de pares clave/valor donde cada clave se corresponde con un campo del formulario, y cada valor con un error de validación encontrado en el proceso o `undefined` en caso de no tuviera errores.

/POST /users/register

Crea un usuario jugador desde la web a partir del formulario de registro e inicia su sesión.

Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	-
Request body	`RegistrationRequest` – Objeto con los datos del usuario
Response body	200 OK – application/json `WithToken<User>`– Devuelve la información del usuario junto con el token de la sesión recién iniciada. 409 Conflict – text/plain En caso de que el correo electrónico facilitado ya esté registrado en la base de datos.

/POST /users

Crea un usuario jugador desde la web a partir del formulario de registro

Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	-
Request body	`User` – Objeto con los datos del usuario
Response body	200 OK – application/json `User` – Devuelve el usuario con su nuevo identificador automático tras la creación

403 Forbidden – text/plain
En caso de que el usuario no disponga del privilegio `AddUser`
409 Conflict – text/plain
En caso de que el correo electrónico facilitado ya esté registrado en la base de datos.

/PUT **/users/:id**

Reemplaza un usuario completamente con nuevos datos

Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	{id} – Cadena de texto con el identificador del usuario
Request body	`User` – Objeto con los datos del usuario
Response body	200 OK – application/json `User` – Devuelve el usuario con los nuevos valores. 403 Forbidden – text/plain En caso de que el usuario no disponga del privilegio `EditUser` 404 Not found– text/plain En caso de que el usuario que se intenta editar no exista.

/DELETE **/users/:id**

Elimina un usuario.

Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	{id} – Identificador del usuario
Request body	-
Response body	200 OK – text/plain En caso de éxito devuelve una respuesta vacía. 401 Unauthorized – text/plain En caso de que el usuario no haya iniciado sesión 403 Forbidden – text/plain En caso de que el usuario no disponga del privilegio `DeleteUser` 404 Not found– text/plain En caso de que el identificador no exista

/POST **/users/recoveryTokens**

Crea un token de recuperación de clave

Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	-
Request body	`{email:string}` – Objeto JSON con un único campo “email”.
Response body	200 OK – application/json

`PasswordRecoveryRequest` – Objeto con información relevante y el identificador único de la operación
409 Conflict – text/plain
En caso de que ya exista un token para este email.

4.2.2.1 Gestión de juegos

/GET	**/games?q=:criteria**
	Ejecuta una búsqueda de juegos empleando los filtros definidos en los parámetros de entrada.
Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	`SearchParams / b64str` – Objeto de tipo SearchParams, convertido a string, serializado a base64 y enviado como parámetro de consulta en la URL. Contiene los criterios de búsqueda de juegos en formato MongoDB.
Request body	-
Response body	200 OK – application/json `SearchResult<Partial<Game>>` – Resultados paginados de la búsqueda correspondientes al índice solicitado. Los datos devueltos no devuelven los juegos completos con la configuración, solo la información multimedia. 401 Unauthorized – text/plain En caso de que el usuario no haya iniciado sesión 403 Forbidden – text/plain En caso de que el usuario no disponga del permiso `ListGames`

/GET	**/games/:id**
	Obtiene los datos de un juego
Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	{id} – Cadena de texto con el identificador del juego
Request body	-
Response body	200 OK – application/json `Game` – Información completa del juego, incluyendo toda la configuración. 401 Unauthorized – text/plain En caso de que el usuario no haya iniciado sesión 403 Forbidden – text/plain En caso de que el usuario no disponga del permiso `ViewGame`

/GET	**/games/:id/stats**
	Obtiene los datos de un juego
Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	{id} – Cadena de texto con el identificador del juego
Request body	-
Response body	200 OK – application/json `GameStats` – Estructura de datos con metainformación del juego:nº de instancias, jugadores conectados, estadísticas, etc. 404 Not found – text/plain En caso de que no exista un juego con ese identificador.

/POST	**/games**
	Crea un nuevo juego con los datos y configuración indicados
Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	-
Request body	`Game` – Estructura de datos con la información del juego
Response body	200 OK – application/json `Partial<Game>` – Devuelve los datos parciales del juego recién creado. 401 Unauthorized – text/plain En caso de que el usuario no haya iniciado sesión 403 Forbidden – text/plain En caso de que el usuario no disponga del permiso `AddGame` 409 Conflict – text/plain En caso de que se envíe un juego con un identificador no nulo 500 Server error– text/plain En caso de error al guardar el juego.

/PUT	**/games/:id**
	Reemplaza un juego completamente con nuevos datos
Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	{id} – Cadena de texto con el identificador del juego
Request body	`Game` – Objeto con los datos del juego
Response body	200 OK – application/json `Partial<Game>` – Devuelve la información parcial del juego actualizado (sin configuración) 401 Unauthorized – text/plain En caso de que el usuario no haya iniciado sesión

403 Forbidden – text/plain
En caso de que el usuario no disponga del permiso `EditGame`

/DELETE /games/:id

Elimina un juego

Headers Authorization {token} – Token obtenido al iniciar la sesión

URL Params {id} – Identificador del juego

Request body -

Response body 200 OK – text/plain
En caso de éxito devuelve una respuesta vacía.
401 Unauthorized – text/plain
En caso de que el usuario no haya iniciado sesión
403 Forbidden – text/plain
En caso de que el usuario no disponga del privilegio `DeleteGame`
404 Not found– text/plain
En caso de que el identificador no exista
409 Conflict – text/plain
En caso de que el juego tenga alguna instancia ejecutándose en memoria.
En caso de que el juego tenga alguna instancia latente en la base de datos

/POST /games/:id/join

Vincula la sesión del jugador con la instancia correspondiente del juego seleccionado. Si el jugador ya estaba vinculado a una instancia, se le redirige hacia ella. De lo contrario, se vincula su cuenta con la primera instancia libre.

Headers Authorization {token} – Token obtenido al iniciar la sesión

URL Params {id} – Identificador del juego

Request body -

Response body 200 OK – application/json
`Asset[]` – Una colección de los recursos gráficos empleados en el juego seleccionado, que el cliente necesitará para su inicialización
401 Unauthorized – text/plain
En caso de que el usuario no haya iniciado sesión
403 Forbidden – text/plain
En caso de que el usuario no disponga del permiso `Play`
500 Server error– text/plain
En caso de que el servidor no disponga de instancias libres para

añadir al jugador.

/GET **/games/:id/style.css**

Devuelve el archivo con las reglas de estilo CSS definidas por configuración para el juego seleccionado.

Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	{id} – Cadena de texto con el identificador del juego
Request body	-
Response body	200 OK – text/css Devuelve un recurso CSS con todas las reglas de estilo necesarias para visualizar el cliente de acuerdo a la configuración del juego. 404 Not found – text/plain En caso de que no exista un juego con ese identificador.

/GET **/gamelist**

Obtiene una lista con información parcial de todos los juegos

Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	{id} – Cadena de texto con el identificador del juego
Request body	-
Response body	200 OK – application/json `Partial<Game>[]` – Estructura de datos con información multimedia del juego, sin la configuración.

4.2.2.4 Gestión de instancias

/GET **/instances?q=:criteria**

Ejecuta una búsqueda de instancias empleando los filtros definidos en los parámetros de entrada.

Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	`SearchParams / b64str` – Objeto de tipo SearchParams, convertido a string, serializado a base64 y enviado como parámetro de consulta en la URL. Contiene los criterios de búsqueda de juegos en formato MongoDB.
Request body	-
Response body	200 OK – application/json `SearchResult<Partial<GameInstanceSummary>>` – Resultados paginados de la búsqueda correspondientes al índice solicitado.

	Los datos devueltos no devuelven estructuras de tipo instancia, sino un tipo `GameInstanceSummary` con información más relevante: nombre del juego, n.º de jugadores, información en memoria, etc. 401 Unauthorized – text/plain En caso de que el usuario no haya iniciado sesión 403 Forbidden – text/plain En caso de que el usuario no disponga del permiso `ListInstances`

/GET	**/instances/:id**
	Obtiene los datos básicos de una instancia, sin información de estado del juego.
Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	{id} – Cadena de texto con el identificador del juego
Request body	-
Response body	200 OK – application/json `GameInstance` – Información completa del juego, incluyendo toda la configuración. 401 Unauthorized – text/plain En caso de que el usuario no haya iniciado sesión 403 Forbidden – text/plain En caso de que el usuario no disponga del permiso `ViewInstance` 404 Not found – text/plain En caso de que no exista una instancia con ese identificador

/POST	**/instances**
	Crea una nueva instancia con los valores facilitados. Al invocar este método la aplicación de backend inicializará la instancia de acuerdo a las reglas del juego, creando el mapa inicial y las plazas para jugadores indicadas.
Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	-
Request body	`Instance` – Estructura de datos con la información inicial de la instancia.
Response body	200 OK – application/json `Partial<GameInstance>` – Devuelve la estructura parcial de la instancia, sin información del mapa, colas o jugadores. 401 Unauthorized – text/plain En caso de que el usuario no haya iniciado sesión 403 Forbidden – text/plain

	En caso de que el usuario no disponga del permiso `AddInstance` 409 Conflict – text/plain En caso de que el identificador de juego asociado a la instancia no se encuentre. En caso de que se envíe un identificador ya existente en la estructura GameInstance 500 Server error– text/plain En caso de error al guardar

/PATCH /instances/:id

Cambia el número de jugadores de una instancia existente.

Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	{id} – Cadena de texto con el identificador del juego
Request body	`Record<string,any>` – Diccionario con la clave `maxPlayers` y valor el nuevo número de jugadores
Response body	200 OK – application/json `Partial<GameInstance>` – Devuelve la estructura parcial de la instancia actualizada, sin información del mapa, colas o jugadores. 400 Bad request– text/plain En caso de que maxPlayers no se facilite 401 Unauthorized – text/plain En caso de que el usuario no haya iniciado sesión 403 Forbidden – text/plain En caso de que el usuario no disponga del permiso `EditInstance` 409 Conflict– text/plain En caso de que la instancia esté ejecutándose en el momento del cambio. En caso de que el nuevo número de jugadores se encuentre por debajo del total de jugadores registrados en esa instancia.

/PUT /instances/:id/status

Cambia el estado de ejecución de una instancia.Al parar y arrancar, el estado completo de la instancia se persiste en la base de datos.

Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	-
Request body	`Record<string,string>` – Diccionario con una única clave status cuyo valor debe ser start o stop, que indica el estado que debe tomar la instancia.

Response body 200 OK – text/plain
Si todo ha ido bien devolverá una cadena de texto con el nuevo estado de la instancia.
400 Bad request– text/plain
En caso de que no se haya facilitado el nuevo estado
401 Unauthorized – text/plain
En caso de que el usuario no haya iniciado sesión
403 Forbidden – text/plain
En caso de que el usuario no disponga del permiso `EditInstance`
409 Conflict – text/plain
En caso de que se intente parar una instancia que no está en memoria
En caso de que se intente arrancar una instancia que yá está en memoria.
500 Server error– text/plain
En caso de error al guardar

/DELETE /instances/:id

Elimina una instancia

Headers Authorization {token} – Token obtenido al iniciar la sesión

URL Params {id} – Identificador de la instancia

Request body -

Response body 200 OK – text/plain
En caso de éxito devuelve una respuesta vacía.
401 Unauthorized – text/plain
En caso de que el usuario no haya iniciado sesión
403 Forbidden – text/plain
En caso de que el usuario no disponga del privilegio `DeleteInstance`
404 Not found– text/plain
En caso de que el identificador no exista
409 Conflict – text/plain
En caso de que la instancia esté aún en memoria.

4.2.2.5 Otros

/POST /uploads

Codifica el contenido de una imagen en una cadena base64 y lo sube a una carpeta

junto al resto de recursos gráficos.

Headers	Authorization {token} – Token obtenido al iniciar la sesión
URL Params	-
Request body	`FileUpload` – Estructura de datos que contiene la representación en base64 de la imagen junto con su extensión original.
Response body	200 OK – text/plain Si todo ha ido bien devuelve la URL de la nueva imagen.

4.3 Diseño gráfico e interfaces

4.3.1 Estilos

4.3.1.1 Website

Los valores de personalización del *website* son sólo un estilo básico, ya que se espera que la empresa o particular que explote la plataforma lleve a cabo un proceso de personalización de la interfaz. Para tal efecto, el código fuente a partir del cual se generan los binarios de distribución, incorpora un archivo `.scss` que define las variable de estilo del sitio web. A continuación se puede consultar la lista junto con los valores empleados en el estilo de referencia:

Atributo	Variable	Valor por defecto
Color de fondo	`$site-background-color-dark`	#888888
Color de letra	`$site-front-color`	#FFFFFF
Color primario	`$site-primary-color`	#FFB300
Color secundario	`$site-secondary-color`	#00FFAA
Familia tipográfica	`$site-font-family`	Saira Semi Condensed

Tabla 5: Configuración de estilo por defecto del sitio web

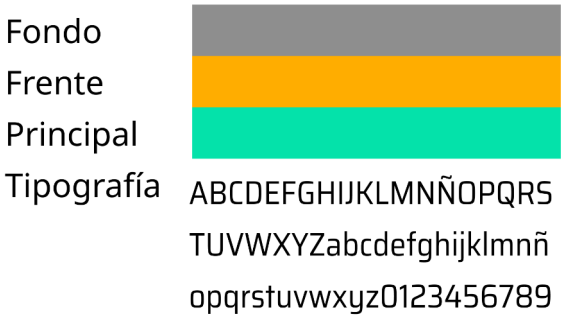


Figura 33: Configuración visual por defecto del sitio web

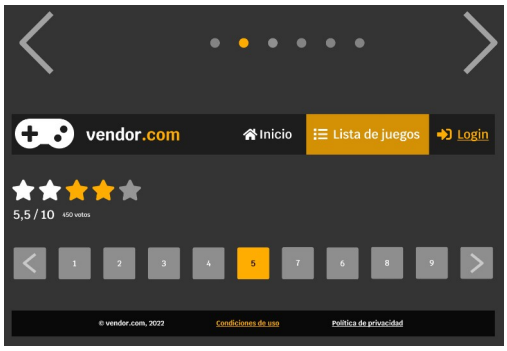


Figura 34: Muestra de los componentes del sitio web

4.3.1.2 Cliente del juego

La interfaz gráfica del juego ha sido diseñada desde cero para poder tomar el control total de la personalización de los elementos desde el *backoffice*, y está accesible desde la configuración de juego, sección interfaz, subsección “Estilo visual y diseño”. En la siguiente tabla se puede consultar la relación completa de atributos configurables:

Descripción	Atributo/Variable[26]
Familia tipográfica	`uiControlFontFamily`
Color de los controles (fondo)	`uiControlBackgroundColor`
Color de los controles (brillante)	`uiControlBackgroundColorBrilliant`
Color de los controles (frente)	`uiControlForegroundColor`
Color de los controles (hover)	`uiControlBackgroundPrimary`
Color de los controles (selected)	`uiControlBackgroundSecondary`
Tamaño del texto	`uiControlTextSize`
Tamaño de los encabezados	`uiControlTextHeadingSize`
Color de las sombras	`uiControlShadowColor`
Padding en elementos con borde	`uiControlPadding`
Color del borde	`uiControlBorderColor`
Radio de circunferencia del borde	`uiControlBorderRadius`
Color del texto (normal)	`uiControlFontColor`
Color del texto (desactivado)	`uiControlFontColorDisabled`
Color del texto (éxito/positivo/bien)	`uiSuccess`
Color del texto (fracaso/negativo/mal)	`uiDanger`
Color del texto (atención/alerta)	`uiWarning`

Tabla 6: Configuración de estilo de la interfaz del juego

Figura 35: Ejemplo de aplicación de dos estilos diferentes sobre la misma información

26 Toda la configuración de estilo está almacenada en la estructura `UIConfig`

4.3.2 Usabilidad / UX

De acuerdo con Nielsen (1994), hay diez criterios que atender a la hora de hacer un sistema usable. A continuación se enumeran los aspectos más relevantes del componente de principal de juego orientados a cumplir estos requisitos:

- Visibilidad del estado del sistema – Cronómetros informativos en las tareas en cola informando al usuario del tiempo restante para su finalización, ventanas emergentes informando de notificaciones entrantes.
- Control del usuario y libertad – El juego anima al usuario a explorar sin temor a equivocarse. Por un lado proporciona información suficiente antes de llevar a cabo las actividades (por ejemplo, informando de su coste en recursos y tiempo), por otro, porque permite cancelar las actividades antes de su finalización.
- Prevención de errores – Durante el diseño de la interacción se ha tenido presente este precepto. Así pues, los controles que permitan llevar a cabo acciones que puedan generar un error, efectúan una comprobación de las precondiciones necesarias, y llegado el caso se deshabilitan.
- Consistencia entre el mundo real y el sistema – Todas las acciones del juego están acompañadas de descripciones textuales y metáforas visuales (iconos) que ayuden al usuario a entender qué está haciendo en cada momento y cuales son las consecuencias de sus acciones.
- Estética y diseño minimalista – Se ha intentado crear una interfaz mínima que no distraiga al usuario de la tarea que debe realizar en cada momento.

4.4 Lenguajes de programación y APIs empleadas

4.4.1 Nodejs

La elección de node.js como plataforma de ejecución se ha tomado con dos criterios. En primer lugar, atendiendo a la amplia cantidad de *plugins*, *middlewares* y librerías disponibles para el, casi listos para utilizar desde su instalación. En segundo lugar, Javascript. Este lenguaje es casi un imperativo en la parte de cliente, y se deseaba una plataforma capaz de ejecutar una aplicación diseñada en las mismas condiciones que el cliente, de forma que pudieran compartir y reutilizar todo el código fuente posible.

4.4.2 Expressjs, WS y express-ws

Ambos son el estándar de facto en sus respectivos campos en node.js, el primero en tratamiento de solicitudes web y el segundo en comunicaciones vía *websocket*[27]. Además, están diseñados para poder trabajar de forma cooperativa mediante el middleware express-ws, que libera al programador de todo el proceso de configuración e instanciación.

4.4.3 Typescript

Se ha escogido como lenguaje principal de desarrollo por varios motivos. En primer lugar, porque ofrece la flexibilidad de Javascript junto con la robustez de un potente sistema de tipos estáticos. En segundo lugar, porque tiene un excelente soporte en los principales IDEs[28] y editores del mercado. En tercer lugar, porque al ser transpilable[29] a Javascript permite compartir el modelo de datos entre el cliente y el servidor, ahorrando tiempo y código duplicado.

4.4.4 Modelo SPA

Tanto el *backoffice* como el *frontoffice* se han diseñado siguiendo un modelo SPA, o aplicación de página única. La elección de esta metodología frente a uno normativo de página única se ha tomado atendiendo a tres criterios. En primer lugar, el de la separación de responsabilidades: las SPAs se encargan de todos los aspectos relacionados con la presentación de la información y la gestión de la interacción, liberando a la aplicación de servidor de estas tareas. En segundo lugar el de la experiencia del usuario, ya que este no tiene que esperar el tiempo completo que tarda en recargar todo el contenido de la página, sino solo la actualización de la información que solicita.

27 Un websocket es una tecnología que permite enviar y recibir mensajes hacia un servidor sin necesidad de que haya una solicitud previa por parte del navegador. Más información en https://developer.mozilla.org/es/docs/Web/API/WebSockets_API (Consultado el 07/09/2022)
28 Acrónimo de *Integrated Development Environment*, o entorno integrado de desarrollo.
29 Proceso que transforma un código fuente de un lenguaje en otro diferente (Wikipedia contributors, 2022a).

4.4.5 Vue3

Sabiendo que el tipo de aplicación que se quiere construir tiene un componente de interacción muy importante, la elección de un *framework* de presentación estaba condicionado a que estuviera basado en Javascript. La elección de Vue frente a Angular o React se toma exclusivamente en base a un criterio, el de la familiaridad. Esta decisión se fundamenta en que el uso de un *framework* conocido agilizará el desarrollo, dado que ya se conocen sus peculiaridades, carencias y problemas.

4.4.6 vue-router / vuex

Estas dos librerías se encargan de la gestión las rutas y del estado de la aplicación, respectivamente. La elección de ambas frente a otras alternativas se efectúa, además de por la familiaridad, por el grado de integración con vue, ya que todas forman parte del mismo ecosistema.

4.4.7 HTML Canvas

Se ha elegido trabajar directamente sobre canvas en lugar de emplear otras librerías (Phaser, Pixi.js) porque todas las características que ofrecen son accesorias, mientras que las curvas de aprendizaje muy superiores. Canvas cumple exactamente con las necesidades del proyecto facilitando un componente sobre el que poder dibujar controlando la posición y el tamaño de los elementos.

4.4.8 Bootstrap 5

La elección del *framework* CSS se toma en base a varios criterios: facilidad de uso, documentación, número de componentes disponibles, integración con el resto del *stack* y la familiaridad. De los candidatos evaluados (Material Design, Bootstrap, Bulma, CSS3 *vanilla* y), Bootstrap 5 es el que mejor cumple los requisitos establecidos.

4.4.9 CSS3

El cliente de juego embebido dentro del *website* está escrito haciendo uso únicamente de CSS3. Esto es así porque las reglas de estilo del juego se deben poder suministrar en tiempo de ejecución. Por el contrario, Bootstrap y otros *frameworks* CSS requieren que las reglas estén precargadas en la cabecera del código HTML. Además, durante la fase de pruebas se corroboró que Bootstrap contaminaba el espacio de reglas CSS del juego, modificando el aspecto visual de muchos elementos, y afectando a la experiencia de usuario. Este es uno de los motivos que ha propiciado la separación lógica del *frontoffice* y el *backoffice* en dos proyectos diferenciados.

4.4.10 MongoDB

La elección de MongoDB es evidente debido a la sinergia que se crea entre esta base de datos y el lenguaje de programación subyacente (Javascript). El uso de cualquier tipo de base de datos relacional (MySQL, PostgreSQL, HSQLDB) habría obligado a normalizar cada tipo de dato antes de pasarlo a una tabla, de cara a poder indexar las búsquedas. Por otro lado, los almacenes de datos en memoria (memcached, redis, etc.) carecen de soporte a largo plazo (requieren una base de datos de respaldo) ni de soporte nativo para JSON, por lo que tampoco han llegado a ser una opción.

Durante la fase de desarrollo se empleará una instancia local de MongoDB instalada en el propio equipo. De cara a la puesta en marcha en producción, en lugar de un servicio local se optará por conectar con un servicio en la nube ubicado en MongoDB Atlas, dejando más capacidad de proceso libre para la aplicación de backend. El cambio entre la instancia local y el servicio remoto es transparente para la aplicación, y debe poder llevarse a cabo tan solo variando una línea de la configuración.

5 Implementación

5.1 Requisitos de la instalación

- Node.js >= v14.16.0
- Carpeta con los proyectos y los assets iniciales
- Conexión a internet

5.2 Instrucciones de instalación

Todas las instrucciones que se detallen a continuación suponen que se trabaja sobre una carpeta preexistente denominada `$PROJECT_DIR`, que contendrá tres carpetas: `$SERVER_DIR`, `$WEB_DIR` y `$BACKOFFICE_DIR`. Cada una de estas carpetas contendrá el código fuente del proyecto correspondiente. Existe además una cuarta carpeta `cdn` con los recursos necesarios para la inicialización automática de los datos, y debe estar al mismo nivel que las otras tres.

5.2.1 App backend

Antes de construir y arrancar la aplicación de backend, hay que tener acceso a una instancia de MongoDB. En general basta con descargar los binarios[30] de internet y ejecutar el servicio `mongod`, o seguir las instrucciones del fabricante.

Con la instancia de la base de datos levantada, se procede a construir el servicio. Para ello, se abre una terminal en la carpeta `$PROJECT_DIR/$SERVER_DIR` y se ejecutan los siguientes comandos:

```
1    npm install -g typescript
2    npm install
```

Estos dos comandos instalan el compilador de Typescript y todas las librerías necesarias para construir e instalar el proyecto. A continuación se editará el archivo .env ubicado en la raíz de la carpeta del servicio y se cambian (si procede) las siguientes líneas:

```
1    CONNECTION_STRING=mongodb://localhost
2    DATABASE=unnamed_project
3    CDN_URL=http://localhost:3000/public/
4    STATIC_FOLDER=c:/dev/unnamed-project-ts/cdn/
5    API_URL=http://localhost:3000/
```

30 https://fastdl.mongodb.org/windows/mongodb-windows-x86_64-5.0.9-signed.msi (Recuperado el 02/06/2022)

Donde:

- `CONNECTION_STRING` – es la cadena de conexión con la base de datos, se puede dejar como está para conectar con una instancia local de mongodb. Si fuera necesario especificar credenciales habrá que introducirlas en esta cadena de la manera apropiada.
- `DATABASE` –. Nombre de la base de datos desde la que se cargarán los datos al arrancar y donde se guardarán las instancias y jugadores. El nombre es irrelevante.
- `CDN_URL` – La aplicación de backend almacena en local los ficheros subidos desde el backoffice y la web a modo de micro CDN[31] . Esta cadena de texto determina la URL base que se usará como prefijo para los assets a la hora de mandarlos al cliente, de forma que sepa el host al que debe hacer las solicitudes. En caso de usar un CDN externo, habrá que modificar este valor en consecuencia.
- `STATIC_FOLDER` – Ruta del sistema de archivos donde se encuentran los ficheros subidos por los usuarios. En un CDN local esta carpeta estará accesible vía express.js en la URL definida en el punto anterior. Debe apuntar a la carpeta `$PROJECT_DIR/cdn` (incluida con el resto de proyectos)
- `API_URL` – Apunta al punto de acceso de la propia API. Es necesario porque algunos recursos (style.css) se genera dinámicamente haciendo uso de esta URL.

Una vez modificados los valores, se ejecutarán los siguientes comandos:

```
1    tsc
2    node out/src/index.js
```

Si todo ha ido bien, el servicio estará activo y se podrá continuar con el resto de elementos del proyecto. Por defecto el puerto es el 3000, y el servicio estará disponible en http://localhost:3000

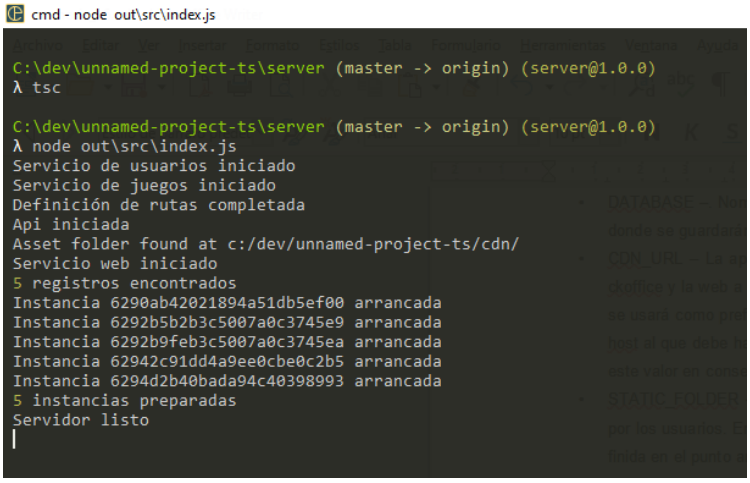


Figura 36: La aplicación de backend tras un arranque exitoso

31 Content Delivery Network, más información en https://www.cloudflare.com/es-es/learning/cdn/what-is-a-cdn/ (Recuperado el 02/06/2022)

5.2.2 Backoffice

Se comienza abriendo un terminal en la carpeta `$PROJECT_DIR/$BACKOFFICE_DIR`. A continuación se verifica el contenido del archivo `.env`:

```
1  VUE_APP_REST_ENDPOINT=http://localhost:3000
2  VUE_APP_CDN_ASSETS_URL=http://localhost:3000/public/
```

donde:

- `VUE_APP_REST_ENDPOINT` – Se corresponde con la URL donde se encuentra el servicio de backend.
- `VUE_APP_CDN_ASSETS_URL` – Se corresponde con la ruta donde el backoffice espera encontrar los recursos gráficos, y debe coincidir con el valor de `CDN_URL` establecido en el servicio de *backend*.

A continuación se debe verificar que en el archivo `tsconfig.json`, el atributo `server` apunta a la ruta relativa correcta donde se encuentra el proyecto del servicio de *backend*. Por ejemplo, en el siguiente fragmento se mostraría la configuración correcta para un valor de `$SERVER_DIR`=”server”:

```
1  ...
2  "server/*":["../server/src/*"]
3  ...
```

Verificadas las rutas para vincular el código fuente del servidor (necesario para compartir los modelos de datos), se procede a instalar, construir y ejecutar el servicio:

```
1  npm install
2  npm run serve
```

Si todo ha ido bien, el servicio estará activo y se podrá continuar con el resto de elementos del proyecto. El puerto de conexión al servicio suele ser el 8080, de no ser así el servidor de desarrollo dará un aviso del nuevo puerto.

Figura 37: La aplicación de backoffice tras un arranque exitoso

5.2.3 Web

Se comienza abriendo un terminal en la carpeta `$PROJECT_DIR/$WEB_DIR`. A continuación se verifica el contenido del archivo `.env`:

```
1  VUE_APP_REST_ENDPOINT=http://localhost:3000
2  VUE_APP_WS_ENDPOINT=ws://localhost:3000/websocket
```

donde:

- `VUE_APP_REST_ENDPOINT` – Es la URL de la aplicación de backend
- `VUE_APP_WS_ENDPOINT` – Es la URL del servicio de websockets para la comunicación asíncrona servidor/cliente. En general tiene la misma estructura que la URL de la api, pero usando el esquema "ws" para *websockets* sobre HTTP o "wss" para *websockets* sobre HTTPS.

A continuación habrá que verificar el archivo tsconfig.json de igual forma que se hizo en el apartado anterior. Una vez satisfechos con el valor, se ejecutan los siguientes comandos:

```
1  npm install
2  npm run serve
```

Si todo ha ido bien, el servicio estará activo y toda la plataforma estará iniciada. El puerto de conexión al servicio suele ser el 8080 o sucesivos, de no ser así el servidor de desarrollo dará un aviso del nuevo puerto.

5.3 Usuario administrador

Para terminar, se ha introducido una llamada en la API para crear un usuario administrador por defecto. Basta con ejecutar http://localhost:3000/management/install y acceder con las credenciales:

Email	root@super.user
Password	root

6 Demostración

6.1 Instrucciones de uso

Tras la instalación, el arranque y la inicialización, al abrir el navegador en la URL indicada por el proyecto web debería aparecer el home con los juegos destacados. No obstante, la instalación no incorpora assets ni datos de muestra. Para solventar esta carencia, se ha dispuesto una versión online[32] accesible en la dirección https://www.toomanyhex.es

Figura 38: Página principal de la versión online

Desde aquí se puede acceder a la página de login y crear una nueva cuenta siguiendo el enlace "Crear cuenta", o identificarse si ya se dispone de un usuario.

32 Las credenciales de administrador de esta página son las mismas que para el servidor local

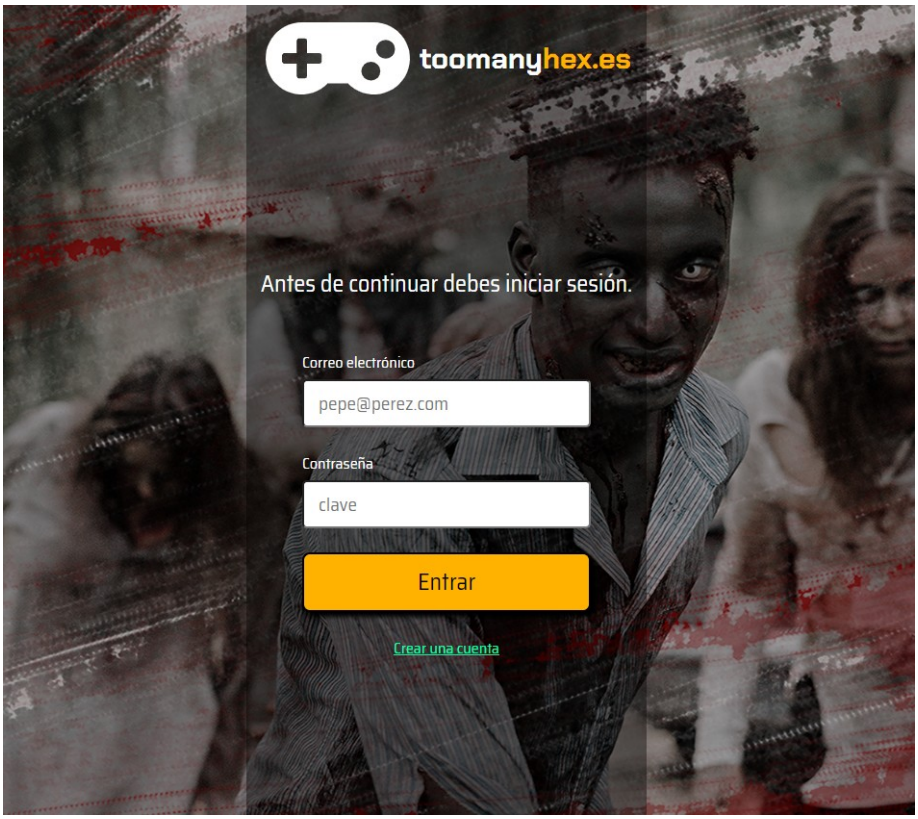


Figura 39: Pantalla de login

Figura 40: Pantalla de registro

Una vez que se ha acreditado la identidad del usuario, solo resta entrar en uno de los juegos para vincular la cuenta del jugador a dicho título, empezando así su aventura en el mundo persistente asociado.

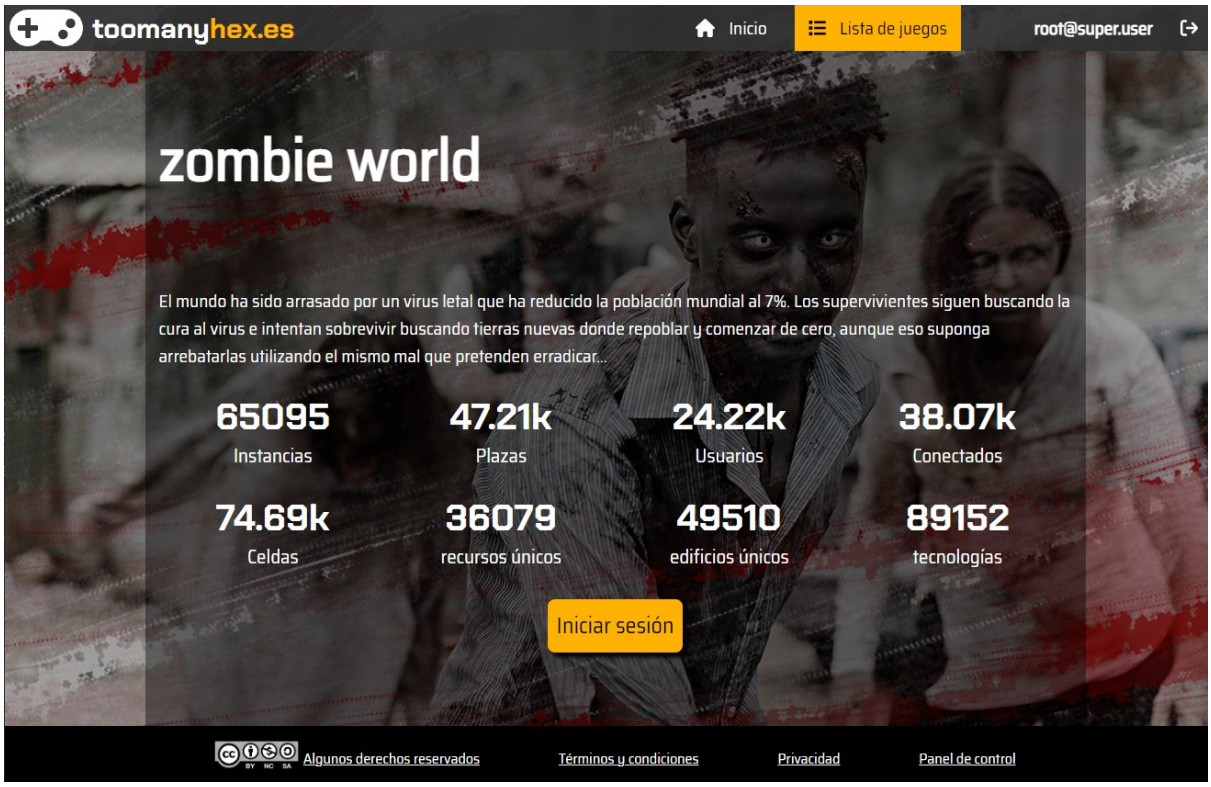


Figura 41: Pantalla de ficha del juego

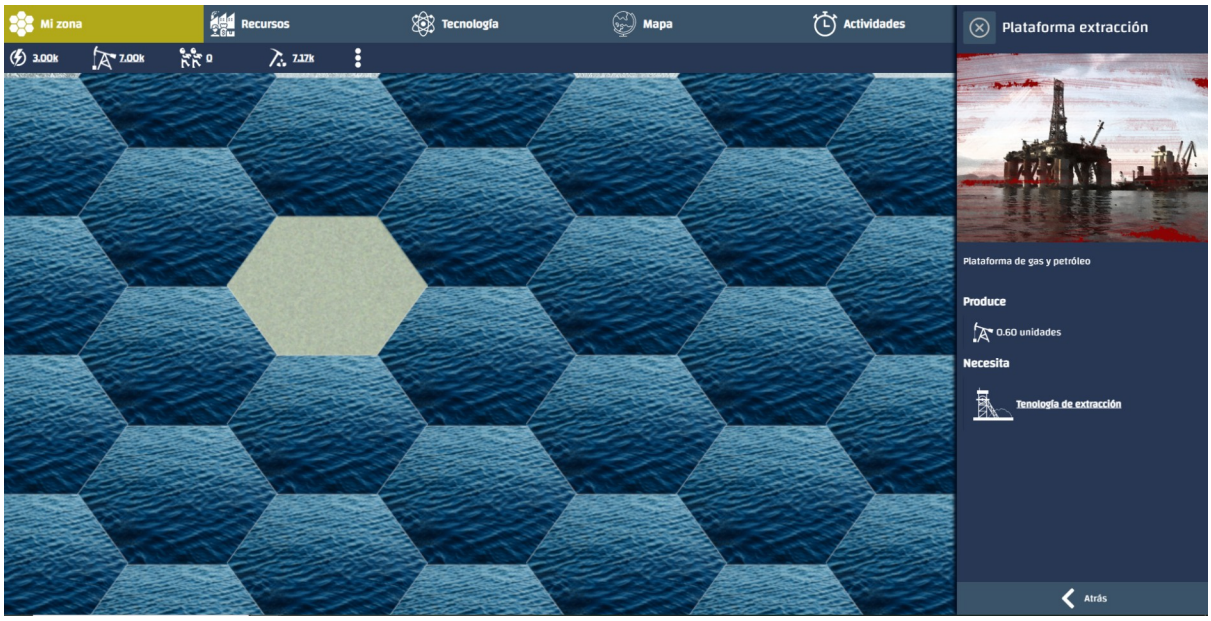


Figura 42: Área del jugador

6.2 Prototipos

6.2.1 Prototipos Hi-Fi

A continuación se expondrán los prototipos de alta calidad sobre los que se basará la interfaz de los dos entregables web: el sitio web y el backoffice.

6.2.1.1 Sitio web

Landing page / Home

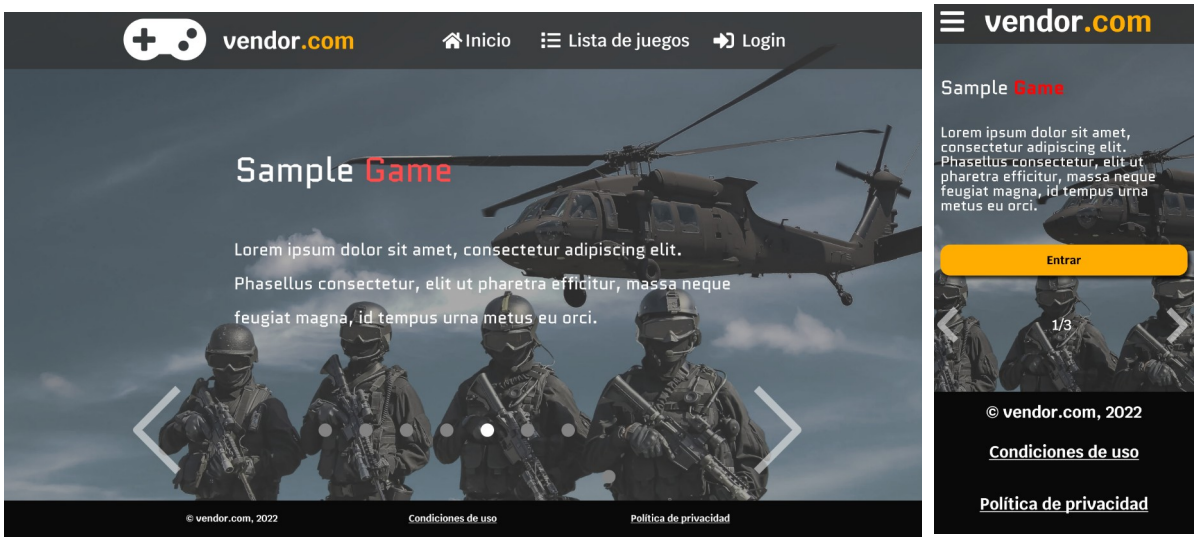


Figura 43: Página principal, versión escritorio(izquierda) y móvil(derecha)

En la imagen se observa cómo el fondo cambia para mostrar la imagen principal del juego seleccionado. Además, se ve cómo el título toma prestados los colores principales asignados al juego en el *backoffice*. Los controles permiten la navegación en horizontal a través de los juegos destacados, así como ir a uno en concreto de la lista.

Login

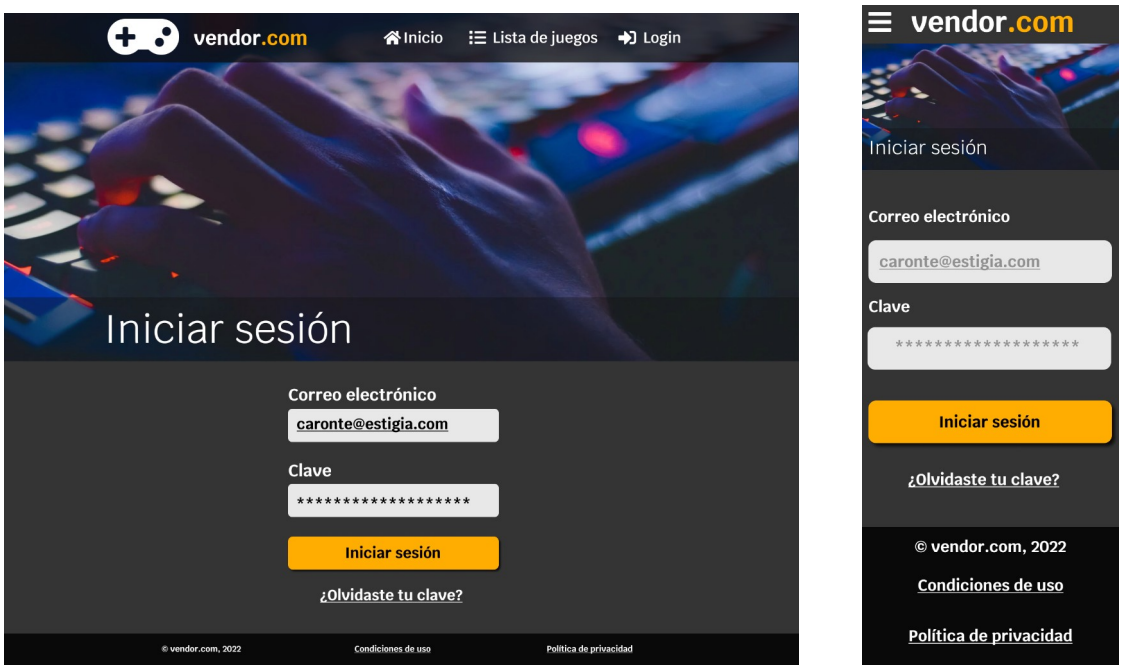


Figura 44: Página de login, versión escritorio(izquierda) y móvil(derecha)

La página de *login* identifica al jugador y le permite acceder a los diferentes juegos de la plataforma. Si el usuario o la clave introducidos no son válidos, la página mostrará un error indicando este hecho. Existe, además, un enlace que llevará al usuario a iniciar el proceso de recuperación de clave.

Registro

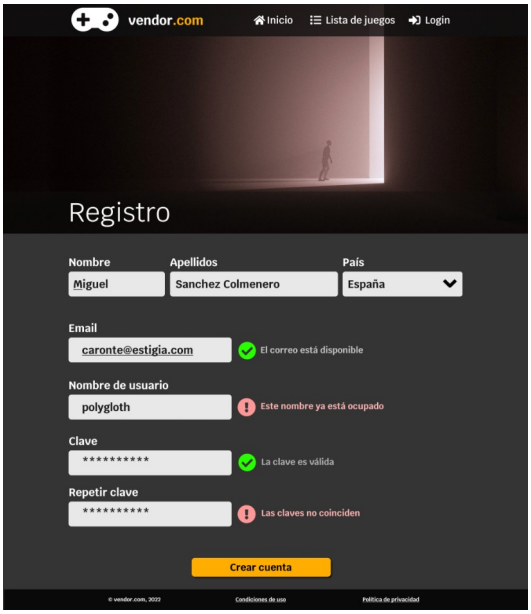


Figura 45: Página de registro, versión escritorio(izquierda) y móvil(derecha)

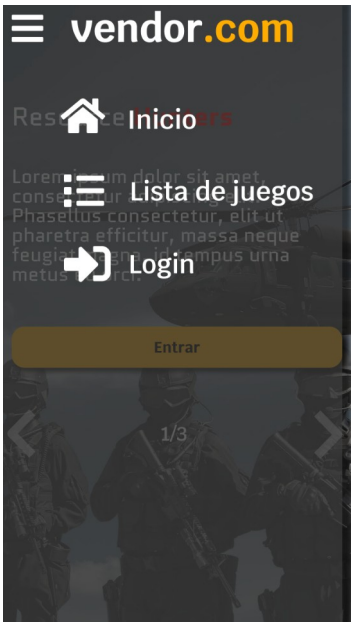


Figura 46: Menú principal desplegado

La página de registro toma los datos del jugador necesarios para crear la cuenta de usuario. Los valores de *email* y nombre de usuario son únicos en toda la plataforma, por lo que si se introduce un valor duplicado, se generará un error. Para mejorar la experiencia de usuario, los valores de los campos se comprobarán sobre la marcha, esto es, tan pronto como el usuario cambie de un campo al siguiente. De esta manera, si un campo toma un valor no válido, se informará visualmente al usuario de este hecho. Al terminar el registro correctamente, el usuario queda automáticamente identificado en el sistema.

Información del juego

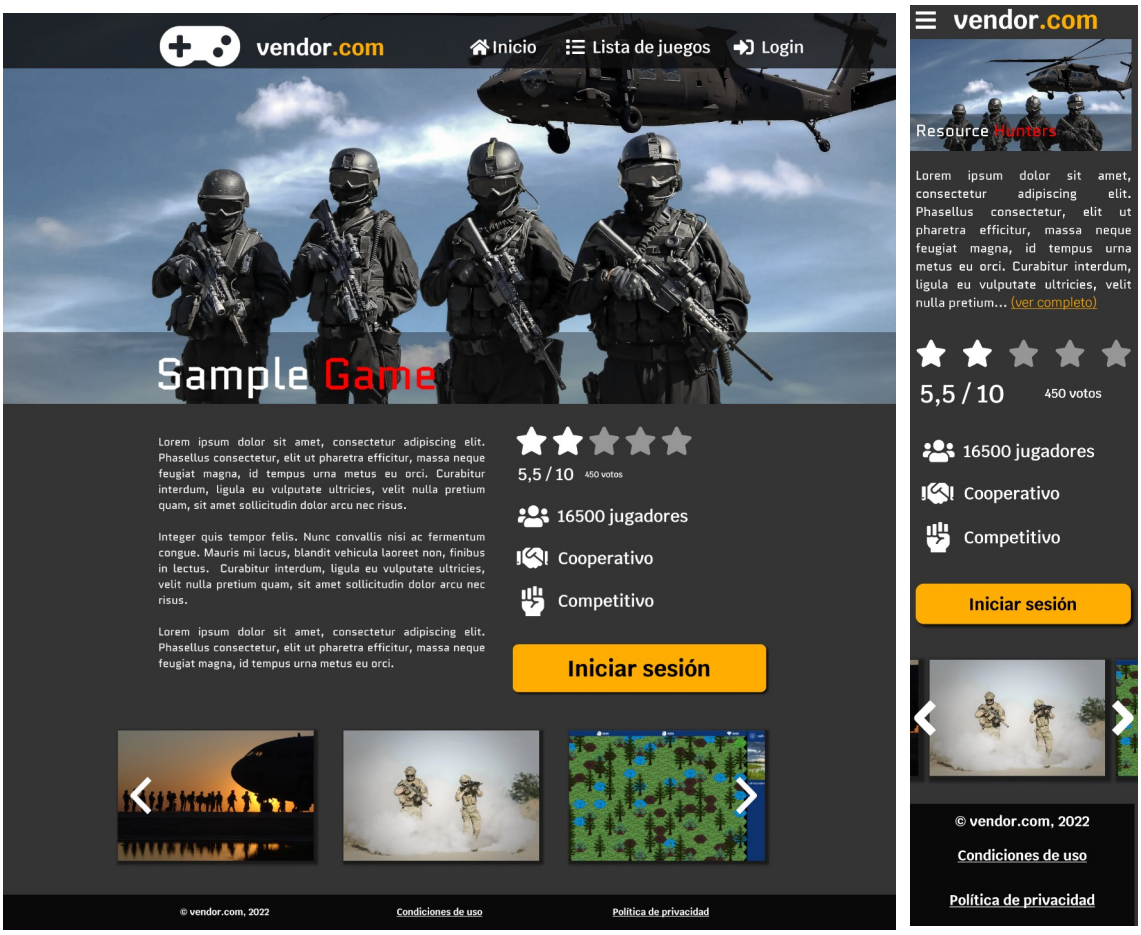


Figura 47: Información del juego, versión escritorio(izquierda) y móvil(derecha)

Esta página muestra las propiedades básicas del juego, así como el contenido multimedia asociado. Tiene un carrusel de imágenes que permite explorar el contenido oculto, así como un control para asignar una valoración al juego. En caso de que el usuario esté registrado, el botón “iniciar sesión” cambia a “Entrar”.

Listado / Buscador

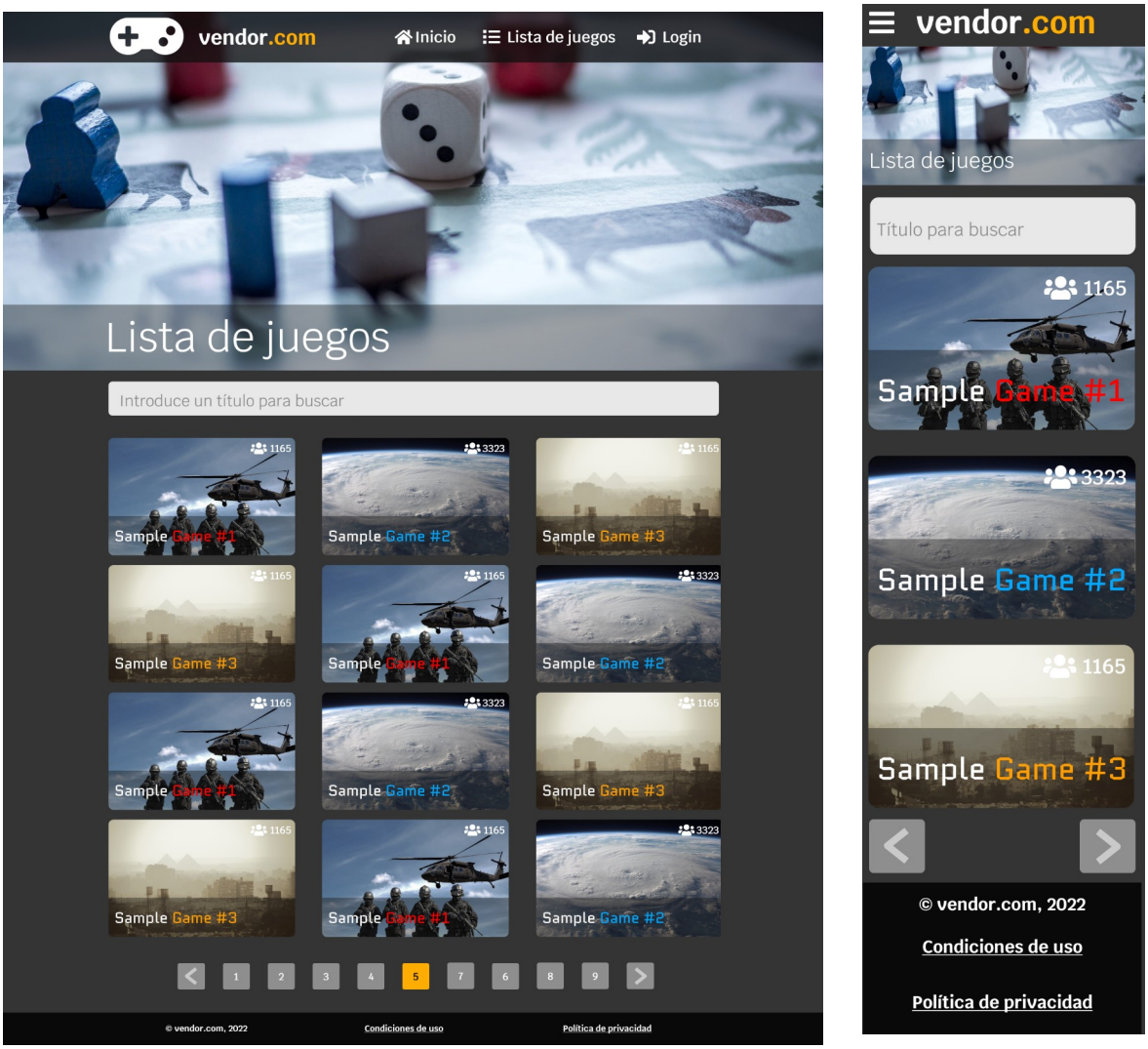


Figura 48: Listado y buscador, versión escritorio(izquierda) y móvil(derecha)

Esta página tiene tres partes importantes. En primer lugar, un control de búsqueda que permite filtrar la lista de juegos en función del nombre. En segundo lugar, la lita de resultados, presentada como una colección de “tarjetas”, donde cada elemento lleva a la información del juego mostrado. Finalmente, un paginador que permite limitar el número de resultados por búsqueda y navegar entre las diferentes páginas.

6.2.1.2 Pantallas del juego

Área de jugador

Figura 49: Vista de área de jugador, versión escritorio(izquierda) y móvil(derecha)

Esta vista presenta la vista principal del juego. Aquí se puede interactuar con las celdas del mapa que son propiedad del jugador, navegar entre las secciones del juego y llevar a cabo actividades.

Panel de información

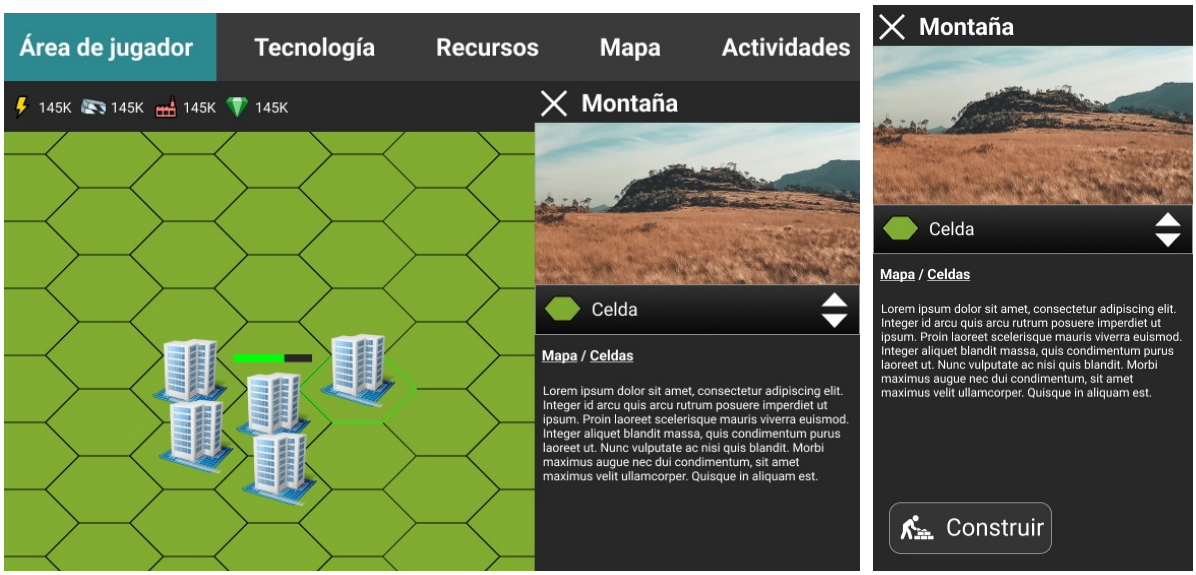


Figura 50: Panel de información, versión escritorio(izquierda) y móvil(derecha)

El panel de información da acceso al contenido multimedia de cada elemento del juego, así como a las actividades asociadas. Desde aquí se pueden investigar tecnologías, construir edificios y llevar a

cabo todas las acciones que el juego permite. Se puede ver como el panel muestra un *dropdown* cuando el objeto seleccionado contiene diversos elementos informativos.

Modal de ejecución de actividad

Figura 51: Ventana modal de actividad, versión escritorio

Este prototipo muestra el sistema de pantallas modales y el modelo de ejecución de actividades. Al ejecutar una actividad, un *popup* aparece para preguntar al usuario los parámetros que debe tomar la actividad antes de ejecutarse. En este caso, se está llevando a cabo una actividad de construcción sobre una celda, y el parámetro que debe decidir el jugador es el tipo de edificio a construir. La versión móvil es idéntica pero con el modal a pantalla completa.

Modal de ejecución de actividad

Figura 52: Vista de tecnología, escritorio(izquierda) y móvil(derecha)

Este prototipo muestra el sistema el mapa tecnológico diseñado por el creador del juego. El árbol se crea de forma dinámica a partir de las relaciones entre las diferentes tecnologías introducidas. El panel informativo ofrece información relevante al seleccionar una tecnología investigada o investigable. La actividad asociada a una tecnología es la de investigación, y mostrará el modal de confirmación correspondiente.

6.2.1.3 Backoffice

Sección / Buscador

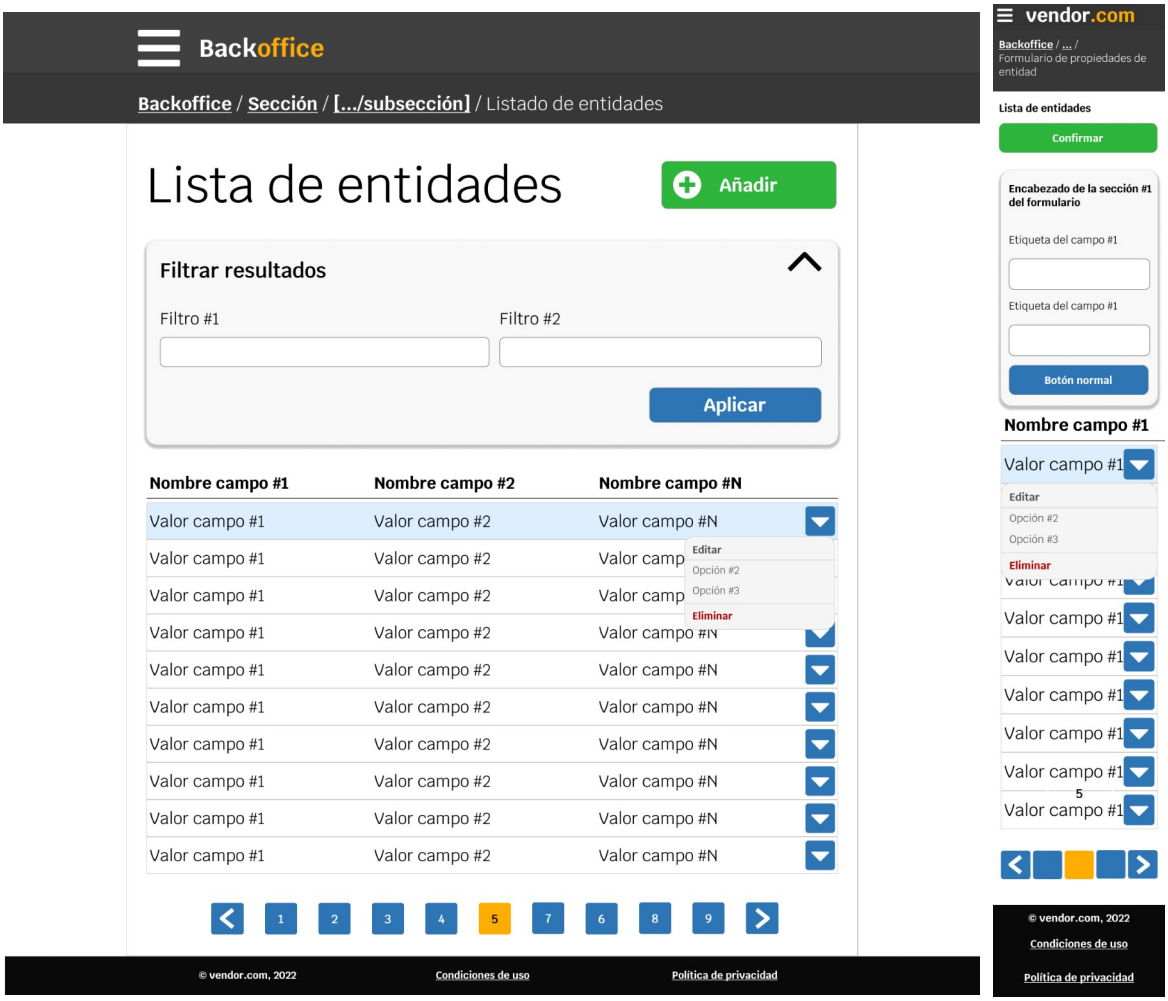


Figura 53: Vista de sección, versión escritorio(izquierda) y móvil(derecha)

La sección, por defecto, muestra el listado de entidades con las que está asociada. El formulario del filtro tiene un número variable de campos que viene determinado por el `SectionHandler` que controla la sección en cuestión. Al hacer clic en “aplicar” se ejecuta la función de búsqueda designada y se devuelven los resultados a la sección, que los pasará al listado y este al componente designado para presentarlos. Por último, se observa un componente de paginado que permite navegar entre los resultados.

Cada ítem presenta, además, un control contextual para llevar a cabo acciones sobre el. Por lo general, se encontrarán las acciones de “Abrir”, “Editar” y “Eliminar”, correspondiéndose con la vista de entidad, el formulario de edición y la ventana modal de borrado, respectivamente.

Sección / Formulario de alta y edición

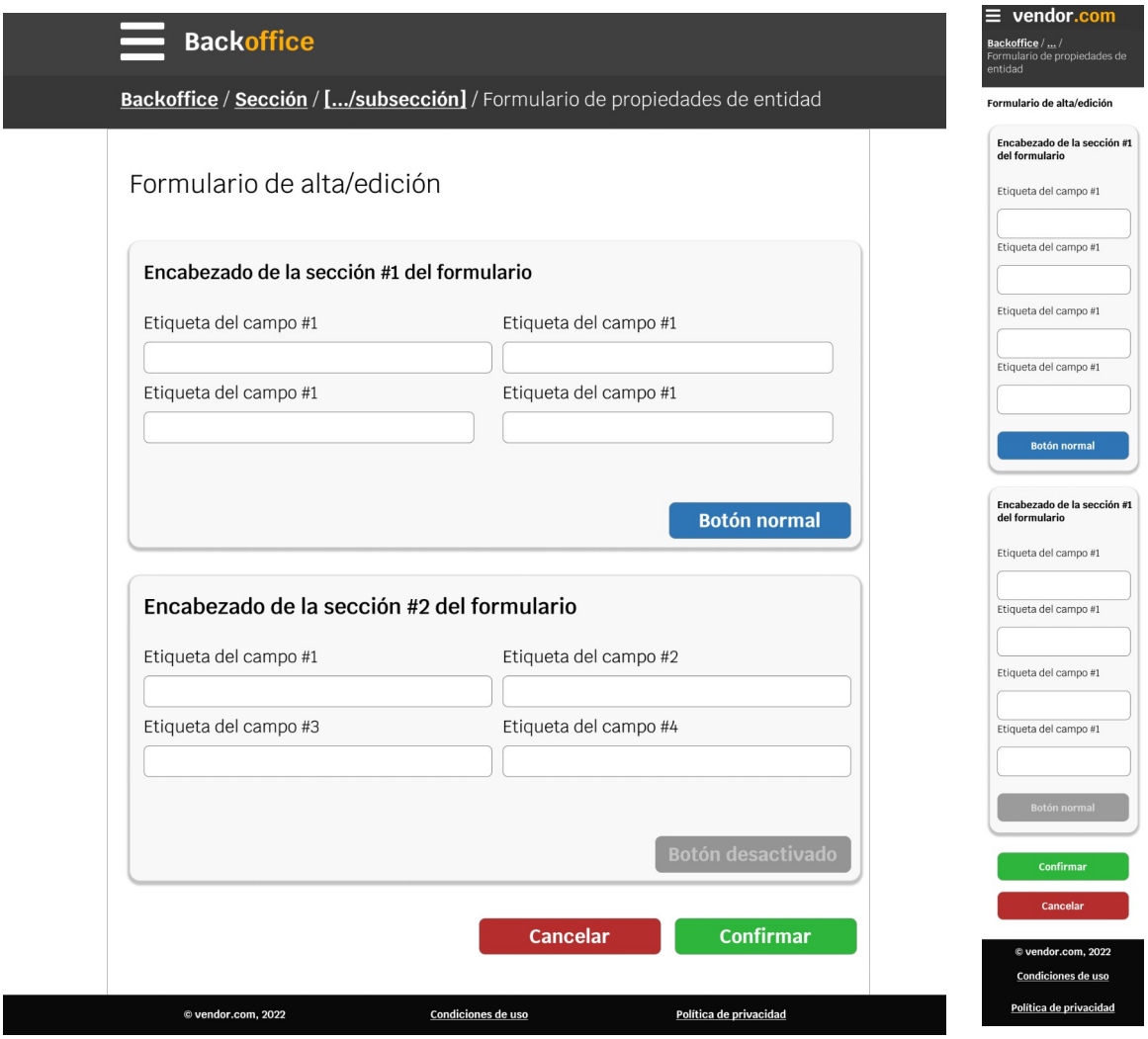


Figura 54: Vista de formulario de edición, versión escritorio(izquierda) y móvil(derecha)

El formulario de edición que se carga en cada momento depende de la entidad que se gestiona. En el comportamiento general, el botón de cancelar simplemente vuelve hacia atrás en el historial del navegador, mientras que el botón de confirmar invoca el *listener* `@save` del componente pasando al *callback* la entidad presentada en el formulario.

Menú de secciones

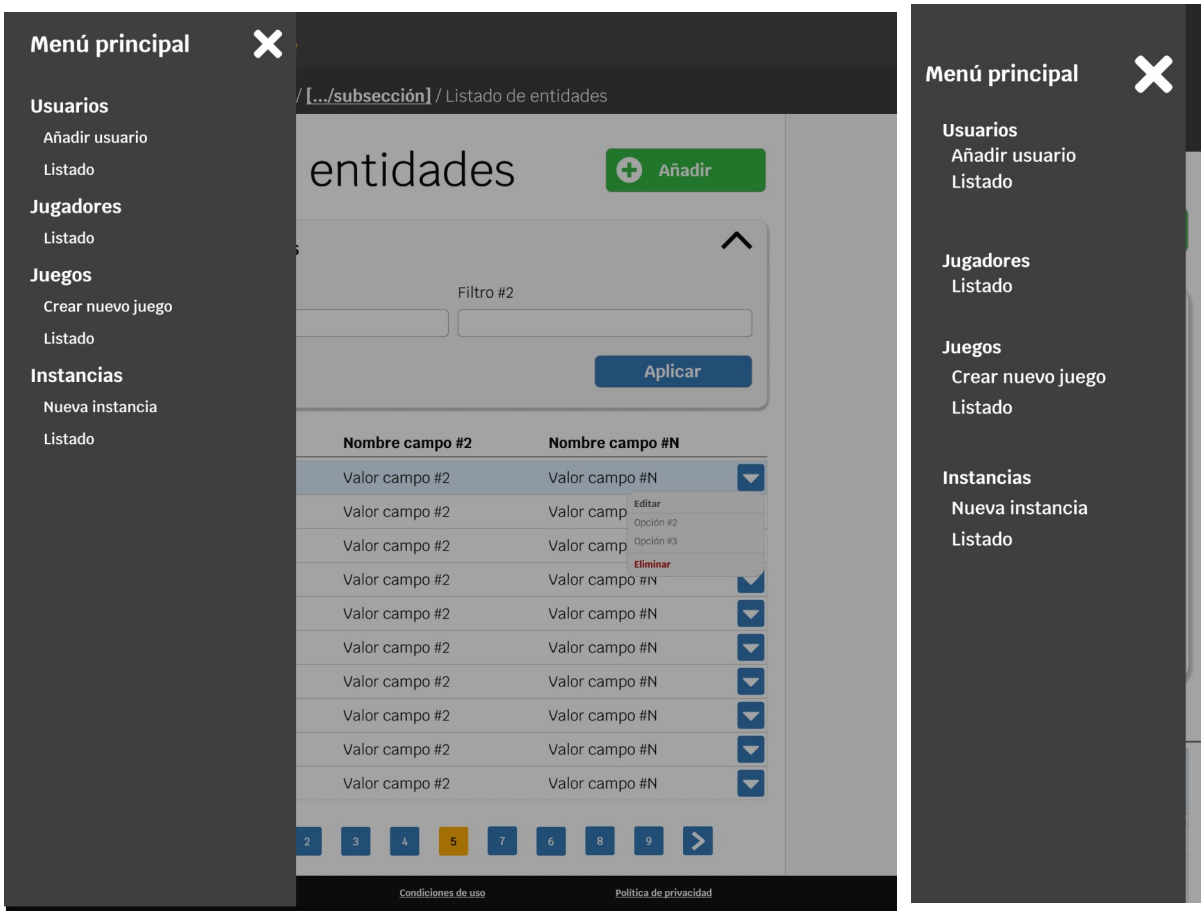


Figura 55: Menú desplegable, versión escritorio(izquierda) y móvil(derecha)

El menú de secciones está oculto por defecto. Es un componente *offcanvas* de Bootstrap que al mostrarse arroja un velo sobre la aplicación, convirtiendo el menú en una ventana modal. Al seleccionar un ítem o pulsar el control de cierre, el modal desaparece, devolviendo el control a la aplicación.

Contenido modal

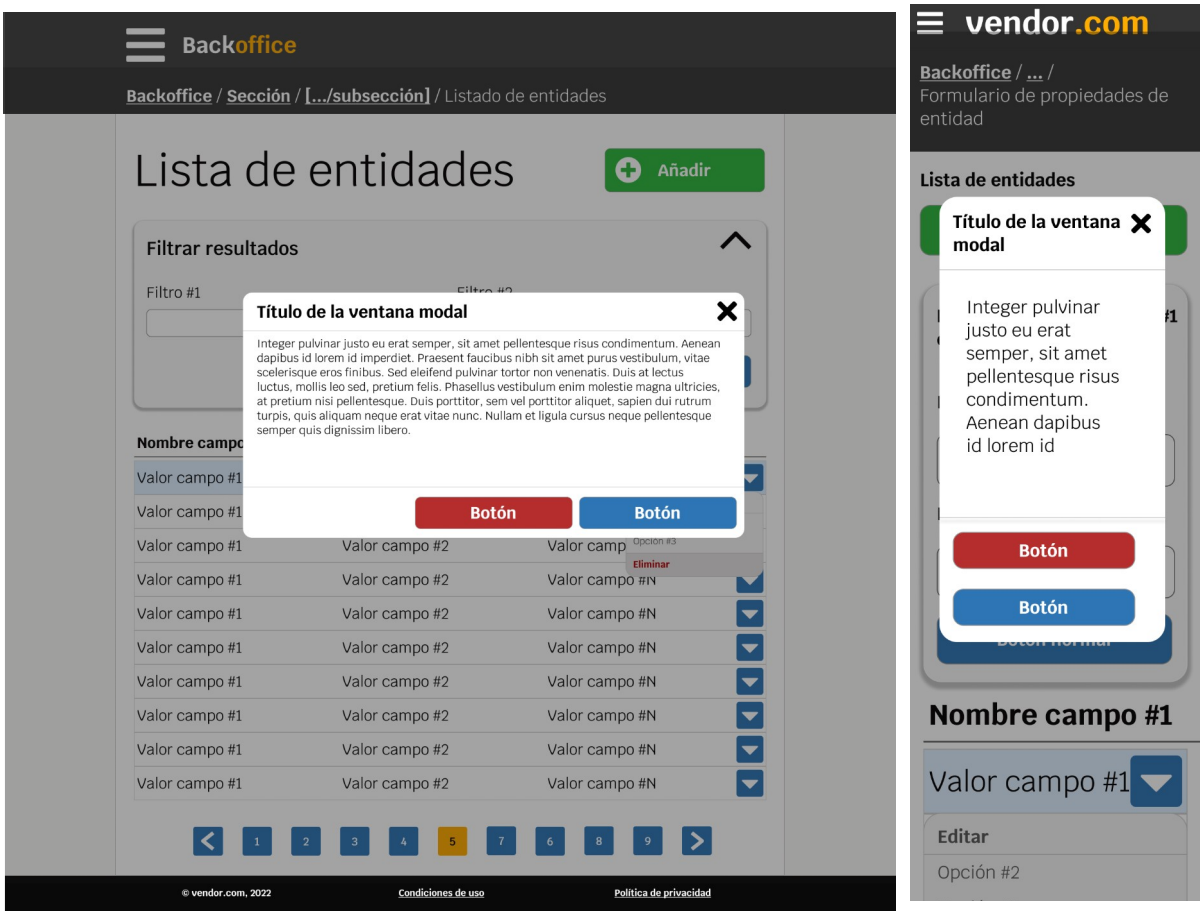


Figura 56: Ventana modal, versión escritorio(izquierda) y móvil(derecha)

El componente de contenido modal permite embeber otros componentes de manera que el usuario no pueda llevar más acciones a cabo hasta que concluya la que está en marcha.

Editor de juegos

Figura 57: Editor de juegos, versión escritorio(izquierda) y móvil(derecha)

Este componente presenta las diferentes categorías de propiedades que pueden ser parametrizadas dentro de un juego. Cada categoría navega al elemento de edición correspondiente y permite cambiar las propiedades especificas. El botón “aplicar cambios” toma todas las ediciones efectuadas y las envía al *backend* para efectuar los cambios necesarios en bloque.

A partir de aquí, al hacer clic en cada sección se ocultan todos los elementos y se visualiza el editor de sección, que agrupa diferentes subsecciones en un componente en “acordeón” que mantiene oculto el contenido que no es necesario en cada momento. Para aquellos elementos que necesiten un tercer nivel de agrupamiento, se dispone de un componente modal capaz de cargar sub-subsecciones, aumentando la profundidad de la navegación hasta 4 niveles desde el menú de edición de juego.

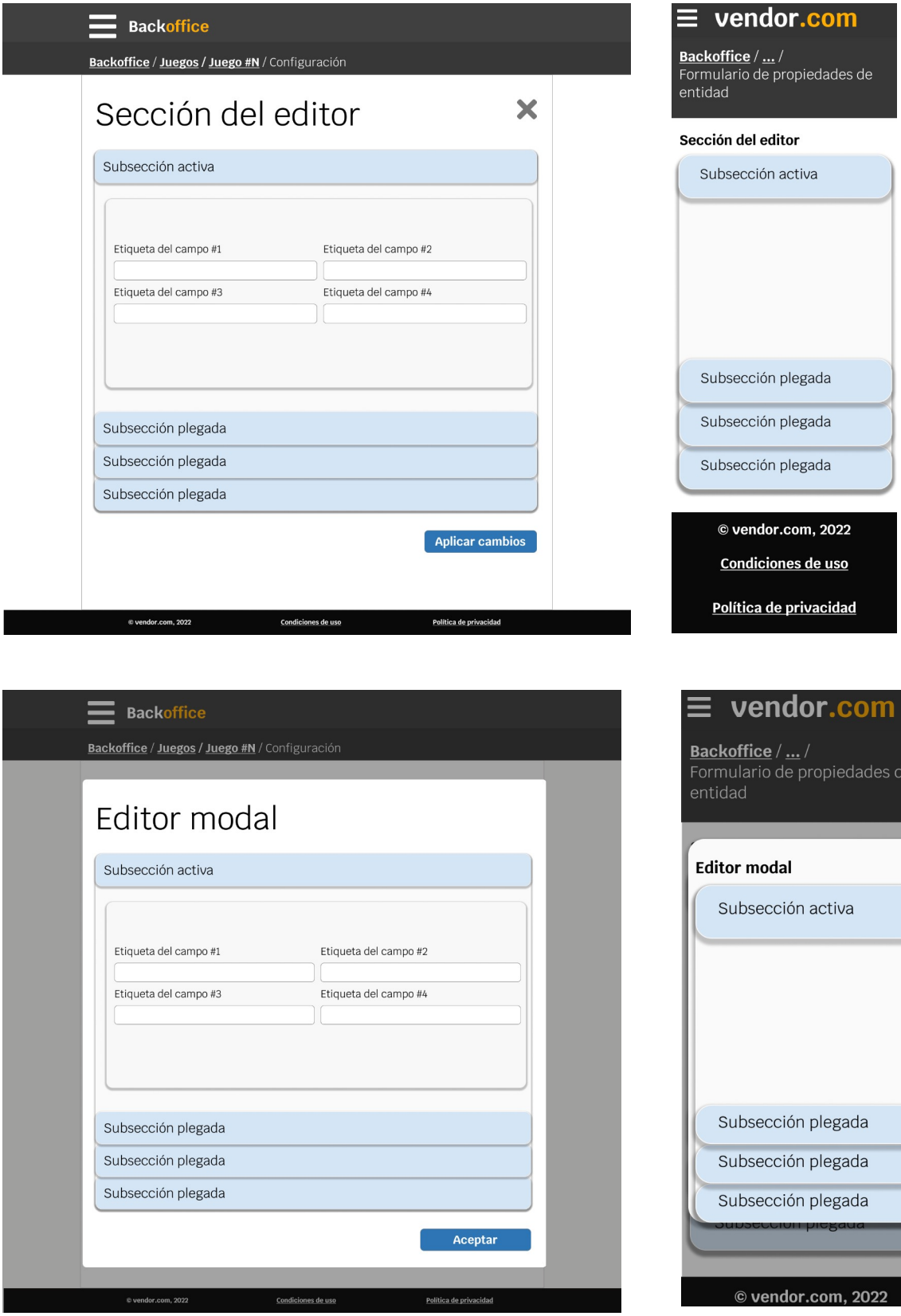


Figura 58: Editor de juegos, editor de sección, subsección y editor modal.

7 Conclusiones y líneas de futuro

7.1 Conclusiones

7.1.1 Lecciones aprendidas

Este proyecto ha supuesto un desafío en el aspecto técnico, organizativo y personal. En primer lugar, en relación al aspecto técnico se ha afrontado un proyecto que involucra más de una docena de tecnologías diferentes entre lenguajes de programación, *frameworks* web, servidores y servicios en la nube, bases de datos y entornos de desarrollo. Conseguir que todo funcione a través de múltiples capas de software, *hardware* y servicios ha requerido incontables horas de estudio, investigación y ensayo y error hasta llegar a la solución correcta del problema que se plantea en cada momento.

En segundo lugar, a la hora de definir y llevar a cabo las tareas se ha planteado un enfrentamiento interior constante entre el aspecto organizativo y el personal. Mientras que el primero es consciente del compromiso y el alcance del proyecto, el segundo se niega a a completar una tarea a sabiendas de que el fragmento de software o producto entregado aún puede mejorarse, generalizarse, o pulir en algún aspecto. Como en otros aspectos de la vida, ha sido necesario llegar a un acuerdo de mínimos y recordar que en el término medio está la virtud.

7.1.2 Objetivos cumplidos

En relación a los objetivos planteados al inicio de este proyecto, se pueden considerar cumplidos en el sentido de que los tres entregables (*frontoffice*, *backoffice* y aplicación de *backend*) producidos cubren el espectro de casos de uso que satisfacen los objetivos propuestos. Así pues, el backoffice facilita la creación de juegos de forma sencilla mediante la parametrización de los elementos del juego y la interfaz. Además, permite gestionar instancias de juegos que evolucionan a lo largo del tiempo, dando lugar a mundos persistentes. Finalmente, incorpora funciones de administración para todos los aspectos de la plataforma, facilitando su gestión.

Por su parte, la web pone a disposición de los jugadores todos los juegos producidos de forma fácil e intuitiva, sin trabas ni complicaciones, de manera que puedan participar en la experiencia de juego con el resto de participantes. Por otro lado, los objetivos personales se han visto satisfechos, pues el proyecto ha sido completado y publicado online y la impresión general con respecto a la calidad del código es positiva.

7.1.3 Cumplimiento de la planificación

A grandes rasgos, la planificación se ha ido cumpliendo en los plazos estipulados, de manera que el grueso del trabajo en cada fase se ha visto completado en tiempo y forma. Sin embargo, la falta de tiempo ha sido una constante a lo largo del proyecto. En consecuencia, al final de cada uno de los seis sprints previstos ha habido que elegir entre dejar una funcionalidad importante a medias para abarcar todas las previstas, o postergar una de menor prioridad para no dejar el trabajo incompleto.

El resultado es que al terminar el proyecto, algunas de las tareas de menor prioridad han ido saltando de sprint en sprint hasta no quedar tiempo disponible para ejecutarlas. Es el caso, por ejemplo, del proceso de recuperación de contraseñas, el *rating* de los juegos o la inclusión de música de fondo. Aunque de menor prioridad, son funcionalidades que aportan valor al producto, de manera que aunque no han sido completadas en el actual ciclo de desarrollo, se incluyen como parte del proceso de mejora.

El hecho de que no haya habido tiempo suficiente para completar todas las tareas previstas tiene dos motivos principales: errores de estimación en las actividades y retrasos derivados de imprevistos. En primer lugar, los errores de estimación se deben tanto a la falta de definición inicial del proyecto como a errores humanos introducidos al efectuar la valoración de las tareas. En segundo lugar, los retrasos debidos a imprevistos, que aunque predecibles, no son cuantificables a priori. En el mejor de los casos, el problema es un fallo humano fácilmente subsanable. En el peor, puede ser obstáculo infranqueable que requiere estudiar vías alternativas para garantizar la viabilidad del proyecto.

Al comienzo del proyecto se estableció una metodología iterativa que permitiera afrontar el proyecto en grupos de funcionalidades transversales que aportaran valor progresivamente. Este proceso se ha revelado extremadamente útil durante la fase de desarrollo, pues ha permitido entregar la funcionalidad mínima requerida en cada momento, al tiempo que evita caer en antipatrones de diseño como la parálisis por análisis[33].

Por último cabe señalar algunos cambios que se han introducido en el proyecto para corregir deficiencias que se han observado a lo largo del proceso de desarrollo:

- Formularios modales – Algunas ventanas modales se han reformulado a pestañas, acordeones o vistas navegables. Esto se debe a que aunque son controles muy útiles en escritorio, las versiones móviles son poco usables debido al espacio que ocupa el velo posterior, además de complicar considerablemente el desarrollo. No obstante, se ha mantenido el nombre original en las tareas aunque su desarrollo haya cambiado.

33 Escenario en el que un proyecto no pasa de la fase de análisis porque se intentan cubrir todos los posibles problemas antes de tener siquiera que enfrenarse a ellos. Más información en https://es.wikipedia.org/wiki/Par%C3%A1lisis_del_an%C3%A1lisis (Recuperado el 02/06/2022)

- Unificación de usuarios y jugadores – En el diseño original existían los conceptos separados de jugadores y usuarios, tal y como atestigua la planificación. Sin embargo, a lo largo del desarrollo se ha observado que ambas entidades podían generalizarse en una sola. Así pues, a la hora de ejecutar las tareas relacionadas con usuarios, no se crea contenido nuevo sino que se refactoriza el anterior con todas sus consecuencias: cambios en las secciones del *backoffice*, gestión de privilegios extra para usuarios jugadores, etc.

- Formularios de búsqueda – Sobre el papel, los formularios de búsqueda de entidades en el *backoffice* iban a ser mucho más potentes de lo que han resultado ser. Esto se debe a que el motor de persistencia elegido (MongoDB) no permite cruzar datos entre colecciones con la misma facilidad que un sistema relacional, aunque no es imposible ni mucho menos. El resultado es que los formularios sólo permiten filtrar por campos incluidos en la entidad gestionada.

7.2 Líneas de futuro

De cara a mejorar la experiencia de usuario, la calidad y la cantidad del contenido ofertado, se prevén las siguientes mejoras:

- Incorporar funcionalidad pendiente – Recuperación de contraseñas, *rating* de los juegos y lista de reproducción de música de fondo.
- Mejora de la estabilidad – La naturaleza distribuida y asíncrona de la aplicación la hace susceptible a fallos de sincronización entre el cliente y el servidor. Se han observado algunos comportamientos extraños en determinados momentos que habría que analizar y corregir.
- Corrección de *bugs* – Hay docenas de pequeños aspectos que pulir o mejorar.
- Gráficos animados – A pesar de que cada elemento del juego se puede alimentar con gran cantidad de recursos gráficos, el resultado final a veces puede parecer un poco estático y falto de vida, especialmente en el mapa del área del jugador. Se prevé la inclusión de assets gráficos que soporten animaciones interactivas, para lo que habría que estudiar la inclusión de Spine[34] o Dragonbones[35] como motores de animación.
- Mensajería instantánea – No hay gran diferencia entre el actual sistema de mensajería en diferido un sistema de chat en tiempo real. Bastaría con introducir los cambios necesarios en la interfaz para visualizar la información de forma usable.
- Nuevos motores de juego – Una vez estabilizadas las características de la plataforma, el siguiente paso es refactorizar el juego y el backoffice para permitir elegir entre diferentes motores de juego, mejorando así la oferta de juegos y la experiencia de usuario.

34 http://es.esotericsoftware.com/ (Recuperado el 02/06/2022)

35 https://docs.egret.com/dragonbones/en (Recuperado el 02/06/2022)

Bibliografía

- de Oliveira, D., Bongers, C., & Nnamdi, C. (2021, 29 junio). Different State Management Patterns for VueJS. daily.dev. Recuperado 1 de abril de 2022, de https://daily.dev/blog/different-state-management-patterns-for-vuejs
- Hieatt, E., & Mee, R. (s. f.). P of EAA: Repository. martinfowler.com. Recuperado 1 de abril de 2022, de https://martinfowler.com/eaaCatalog/repository.html
- Ignjatovic, A. (2021, 15 diciembre). Understanding and fixing N+1 query - Doctolib. Medium. Recuperado 1 de abril de 2022, de https://medium.com/doctolib/understanding-and-fixing-n-1-query-30623109fe89
- MongoDB Documentation Team. (s. f.). Structure your Data for MongoDB. Mongodb.Com. Recuperado 1 de abril de 2022, de https://www.mongodb.com/docs/guides/server/introduction/
- Nielsen, J. (1994, 24 abril). 10 Usability Heuristics for User Interface Design. Nielsen Norman Group. Recuperado 2 de abril de 2022, de https://www.nngroup.com/articles/ten-usability-heuristics/
- Pla, A. (2021, 22 septiembre). Branded content: el contenido de marca que todo target desea. Blog de Comunicación e Información. Recuperado 3 de abril de 2022, de https://blogs.uoc.edu/comunicacio/es/branded-content-el-contenido-de-marca-que-todo-target-desea/
- Rosenfeld, L., Morville, P., Nielsen, J., & ProQuest. (2002). Information Architecture for the World Wide Web. O'Reilly.
- Stafford, R. (s. f.). P of EAA: Service Layer. martinfowler.com. Recuperado 2 de abril de 2022, de https://martinfowler.com/eaaCatalog/serviceLayer.html
- Statista. (2021, 20 agosto). Average age of U.S. video game players in 2019. Recuperado 3 de abril de 2022, de https://www.statista.com/statistics/189582/age-of-us-video-game-players/
- Wikipedia contributors. (2022a, marzo 11). Source-to-source compiler. Wikipedia. Recuperado 1 de abril de 2022, de https://en.wikipedia.org/wiki/Source-to-source_compiler
- Wikipedia contributors. (2022b, marzo 31). Database normalization. Wikipedia. Recuperado 28 de marzo de 2022, de https://en.wikipedia.org/wiki/Database_normalization