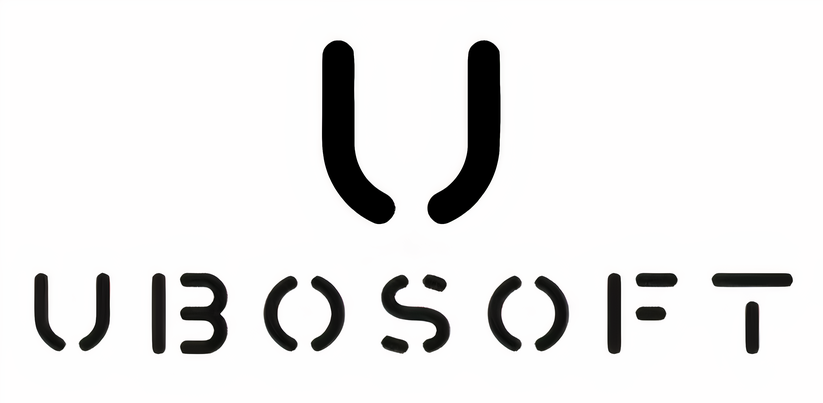
**Ubosoft, CIF:A60695816, Calle María de Luna 3**



Plan de gestión, análisis, diseño y memoria del proyecto Battleship Online

Grupo 09. Sally Floyd

Organización de GitHub: <https://github.com/UNIZAR-30226-2021-09>

Fecha de presentación de la propuesta: 15/3/2021

# Tabla de contenidos

[**Tabla de contenidos**](#_imwwcodok763) **1**

[**1. Introducción**](#_21nazzr2nxdp) **3**

[**2. Organización del proyecto**](#_agsh0lvdvou) **5**

[**3. Plan de gestión del proyecto**](#_5z5r80vxh7j6) **5**

[3.1 Procesos](#_a4rnwe4gpqz9) 5

[3.1.1. Procesos de inicio](#_2hft4y6ysfqy) 5

[3.1.2. Procesos de ejecución y control](#_inkzoo6b5fon) 6

[3.1.3. Procesos técnicos](#_2ub44wwwienk) 7

[3.2. Planes](#_4cv8ws2rgsdh) 8

[3.2.1. Plan de gestión de configuraciones](#_a0v1byj0pwsz) 8

[3.2.2. Plan de construcción y despliegue del software](#_ydwd8sbz1id3) 9

[3.2.3. Plan de aseguramiento de la calidad](#_d7z5j8w1fv9k) 9

[3.2.4. Calendario del proyecto y división del trabajo](#_1k64ndw3p5nl) 11

[**4. Análisis y diseño del sistema**](#_cplb04a3umt5) **13**

[4.1 Análisis de requisitos](#_8w39rigzsfbu) 13

[4.2. Diseño del sistema](#_nm0f1md3t2i1) 15

[**Anexo I. Reglas de juego de Batalla Naval**](#_5ccu2o3zfeze) **21**

[**Anexo II. Prototipo de pantallas y mapa de navegación del juego.**](#_pjviwi9tfe35) **22**

# 

# 1. Introducción

El sistema a desarrollar consiste en un software que permita a sus usuarios jugar a Batalla Naval (ver anexo I) en distintas modalidades de juego, guardar y consultar su desempeño pasado en el juego y mantener una lista de amigos con los que jugar. Se podrán jugar partidas contra amigos añadidos previamente, contra contrincantes aleatorios mediante un sistema de emparejamiento, y se podrán organizar torneos entre usuarios registrados en la aplicación.

Este sistema se desarrollará versiones para dos plataformas, web y móvil (Android). Los jugadores las podrán usar alternativamente, y se guardará el progreso de los mismos sin importar desde qué plataforma accedan.

Alcance del proyecto:

El trabajo requerido para desarrollar el proyecto debe satisfacer lo siguiente:

1. El sistema debe permitir al usuario registrarse en la aplicación e identificarse cada vez que quiera entrar.
2. Los usuarios deben poder jugar a Batalla Naval (ver Anexo I), donde dos jugadores se turnan para intentar hundir la flota del rival.
3. El usuario dispondrá de un perfil que incluirá distintas estadísticas (partidas jugadas en total, ganadas, perdidas…).
4. Se debe poder agregar a otros usuarios a una lista de amigos.
5. Debe permitir varios modos de juego (emparejamiento a ciegas, online con amigos, local, contra IA, torneo con amigos).
6. Se debe poder guardar una partida a medias para retomarla después.
7. Una partida pausada se debe poder continuar desde otro dispositivo.
8. Se podrá abandonar una partida iniciada.
9. Al finalizar una partida, el sistema debe notificar los resultados.
10. Debe proporcionar un listado de las partidas creadas y no finalizadas.
11. Se podrá guardar un histórico de resultados de las distintas partidas y poder consultarlo.
12. Se utilizará una interfaz intuitiva y usable.
13. No habrá una diferencia significativa en el rendimiento de la aplicación entre las versiones de la misma.
14. Los datos de la aplicación deberán estar almacenados por un sistema gestor de bases de datos.
15. El sistema modificará los puntos de un jugador según los resultados de las partidas.
16. El sistema permitirá visualizar un ranking de los jugadores por puntos.
17. El usuario podrá compartir su perfil.
18. El sistema debe permitir configurar el aspecto del juego (colores de los barcos y del fondo para el tablero).
19. El sistema debe permitir configurar el idioma del juego (español, inglés).
20. El sistema contará con efectos de sonido.

Como resultado de la realización del proyecto se entregarán:

* Los fuentes del proyecto
* La aplicación desplegada en AWS.
* El plan de gestión, documentos de análisis y diseño y la memoria del proyecto

Hitos principales:

* 19 de febrero de 2021: Propuesta técnica y económica.
* 15 de marzo de 2021: Plan de gestión, análisis y diseño y memoria del proyecto v1.
* 14 de abril de 2021: Plan de gestión, análisis y diseño y memoria del proyecto v2.
* 31 de mayo de 2021: Actas de las reuniones con el cliente.
* 1 de junio de 2021: Plan de gestión, análisis y diseño y memoria del proyecto final.

# 2. Organización del proyecto

* Alberto Carmona Estradera:
  + Responsable de backend
  + Responsable de despliegue
  + Responsable de la documentación
* Binhui Chen Zhou:
  + Responsable de frontend Android
  + Responsable de la documentación
* Alonso del Rincón de la Villa:
  + Responsable de backend
  + Responsable de despliegue
  + Responsable de la documentación
  + Director
* Adrian Espino Candalija:
  + Responsable de frontend web
  + Responsable de la documentación
* Sergio Atilano Gabete César:
  + Responsable de frontend Android
  + Responsable de las actas
  + Responsable de la documentación
* Santiago Jiménez Navarro:
  + Responsable de frontend web
  + Responsable de la documentación
  + Responsable del control de esfuerzos

# 3. Plan de gestión del proyecto

## 3.1 Procesos

En esta sección se describe cómo se llevarán a cabo distintas tareas que hay que realizar en distintos momentos del proyecto.

### 3.1.1. Procesos de inicio

#### Proceso de identificación y asignación de recursos

Contamos con 6 *smartphones* Android y 6 ordenadores personales, uno por integrante del grupo. No se requiere asignación de estos pues cada uno cuenta con uno. Uno de los integrantes se registrará en AWS y contratará un servicio EC2 (máquina virtual Ubuntu 18.04 LTS con 1 GB de memoria) que todos los desarrolladores compartiremos para desplegar el sistema para ser probado y donde se llevará a producción.

#### Proceso de formación

Utilizaremos la pila tecnológica MEVN, compuesta de MongoDB, Express.js, Vue y Node.js para implementar el backend y el frontend web, así como React Native para el desarrollo del frontend Android. El equipo no tiene experiencia con ninguna de estas herramientas aunque algunos componentes tienen un conocimiento básico del lenguaje en el que estas se basan: JavaScript. Se planean un total de 11 horas de media por persona para la formación en estas tecnologías. Los desarrolladores se autoformarán a través de tutoriales y consultando la documentación, tanto en las etapas previas a la implementación como durante la misma.

|  |  |
| --- | --- |
| ENLACES DE INTERÉS | |
| **Link** | **Descripción del enlace y utilidad del mismo** |
| <https://www.youtube.com/watch?v=Z4TuS0HEJP8&list=PLPl81lqbj-4I11QPam9ApoT7tGbmyBg9P&index=6> | Vídeo donde se explican algunas características y opciones de JavaScript. En la misma playlist se explican otras funcionalidades del lenguaje |
| <https://www.youtube.com/watch?v=GAQB7Y4X5fM&list=PLPl81lqbj-4J-gfAERGDCdOQtVgRhSvIT&index=2> | Playlist de un tutorial de Vue.js que se está siguiendo |
| <https://www.youtube.com/watch?v=ARIzrNwA5HQ> | Ejemplo completo de implementación de una aplicación sencilla usando MEVN |
| <https://vuejs.org/v2/guide> | Guía oficial para el manejo de Vue.js |
| <https://reactnative.dev/docs/getting-started> | Documentación oficial de React Native |

### 3.1.2. Procesos de ejecución y control

#### Proceso de comunicaciones y registro de decisiones y actas

Semanalmente, los sábados (salvo excepciones) tendrá lugar una videoconferencia con Google Meet en la que todos los integrantes estarán presentes. En está se revisará el desarrollo en esa semana y se discutirán los cambios a realizar. Además se usará un grupo de Whatsapp conjunto para la coordinación, pregunta de dudas y comunicación de incidencias. Las decisiones tomadas en las reuniones se reflejarán en las actas. Cualquier decisión individual será comentada en las reuniones y por lo tanto quedará reflejada en el acta correspondiente. Estas actas se redactarán durante el transcurso de cada reunión.

#### 

#### Proceso para determinar las tareas a realizar y su reparto

Al final de cada reunión semanal se fijarán unos objetivos para tener cumplidos para la siguientes reuniones. Estos se reflejaran como *issues* en la herramienta de gestión de incidencias de GitHub para mantener un registro claro de las tareas, aunque pueda comunicarse también por otros canales como el grupo de Whatsapp.

#### Proceso de gestión del equipo humano

Si los involucrados en una disputa no son capaces de llegar a un acuerdo en un primer momento, se hará partícipe al resto del equipo y se tratará de llegar a la resolución más beneficiosa para el proyecto.

#### Proceso de medida de progreso y monitorización del estado del proyecto

El progreso y estado del proyecto se evaluará comparando los *issues* que se esperaban resolver con los resueltos. Si uno de los desarrolladores estima que no podrá acabar a tiempo la tarea en la que trabaja, podrá notificarlo por cualquiera de los canales de comunicación mencionados y el director podrá destinar a esa tarea a otro miembro que esté disponible.

Si, por la naturaleza de la tarea, esta no facilita la colaboración de otros miembros para acabarla, podrá retrasarse su fecha de finalización y desviar parte de otras tareas previstas de su encargado a otro miembro (preferiblemente otro encargado de la misma *layer*).

En el caso de que esto último no sea posible, se barajará renunciar a alguna funcionalidad secundaria para cumplir los plazos e intentar implementarla al final, si es que queda tiempo.

#### Proceso de la entrega de resultados

En las fechas de entrega previstas se entregarán los fuentes desarrollados hasta el momento, además de la documentación (plan de gestión, análisis, diseño y memoria). Además se desplegará el sistema en el servicio EC2 para que el cliente pueda ser testigo de y probar la aplicación en las reuniones programadas previas a las entregas.

### 3.1.3. Procesos técnicos

#### Métodos, herramientas y técnicas

Respecto a los IDEs a utilizar, cada subgrupo de desarrolladores (back-end, front-end web y front-end android), definirán cual es el IDE que van a utilizar. Esto se hace así porque es preferible que las personas que están trabajando en el mismo código utilicen la misma herramienta de desarrollo para evitar incompatibilidades ocasionadas por un cambio de IDE, pero por otra parte se permite que los distintos grupos usen IDEs diferentes para flexibilizar el desarrollo y porque entre estos grupos no se comparte el código hasta que no se llega a la etapa de integración.

Así pues, el entorno de desarrollo que se va a utilizar es el editor Visual Studio Code, junto a las librerías de Live Server (para observar los cambios del código a tiempo real), Prettier (para ayudar a formatear el código), y Live Share en el caso de que haga falta compartir el código entre los desarrolladores.

Para el control de versiones se usará Git y mediante un workflow centralizado, cada equipo (backend, frontend web y frontend Android) subirá los cambios al que le corresponda de los tres repositorios alojados en Github.

### 3.2. Planes

### 3.2.1. Plan de gestión de configuraciones

#### Convenciones de nombres y guías de estilo

Para mantener la homogeneidad entre los diferentes documentos desarrollados por todo el equipo y los contenidos de los mismos, se van a definir qué convenciones de nombres se van a utilizar.

El estilo de nombrado que se va a utilizar va a ser el Camel Case (poner la primera palabra en minúsculas, y las siguientes palabras también en minúsculas excepto la primera letra, uniendo todas las palabras. Por ejemplo, camel case se escribiría *camelCase*), ya que es uno de los estilos de nombrado más usados en proyectos de JavaScript.

* Los nombres de los documentos serán lo suficientemente descriptivos como para detallar cuál es su contenido, y se podrá diferenciar los diferentes archivos simplemente observando su nombre.
* Las variables tendrán nombres descriptivos que ayuden a comprender su objetivo. Existen variables especiales como los iteradores, los cuales podrán llamarse i, j, k, o alguna otra letra minúscula que tenga relación con lo que está representando.
* Las clases se escribirán siempre capitalizadas: La primera letra en mayúscula y el resto en minúsculas.
* Las constantes irán identificadas por un nombre completamente escrito en mayúsculas. Por ejemplo *MAX*, o *DIM*.

Para el desarrollo del proyecto se usarán guías de estilo de Google para HTML, CSS y JavaScript para el front-end. En cuanto al back-end se usarán las guías de estilo de Google para JavaScript.

#### Responsables

El encargado de las actas será Sergio Gabete César, el jefe de proyecto será Alonso del Rincón y Santiago Jimenez el encargado del control de esfuerzos. Alonso del Rincón y Alberto Carmona serán responsables del despliegue, aunque seguramente todos los miembros terminarán desempeñando esta tarea. El resto de tareas se realizarán por todos los componentes del proyecto.

#### Recursos

Se trabajará sobre tres repositorios públicos alojados en github: uno para backend, otro para frontend web y otro para frontend Android.. Todos los integrantes tendrán permisos de administración sobre él. Estos cuentan con una rama principal (main) donde se llevan el código una vez una funcionalidad está completa y otra rama (development) donde se va guardando los cambios periodicamente. La documentación se subirá a un cuarto repositorio.

#### Procedimientos para modificar los fuentes

Se usará un flujo de trabajo centralizado. Cada equipo (frontend web, frontend Android y backend) tendrá una rama propia donde subir sus cambios. Estas ramas solo se fusionaran con la principal cuando se complete una funcionalidad. Es decir, la rama principal siempre deberá contener código compilable y probado.

Las incidencias podrán ser bugs a solucionar o tareas a realizar. Estas podrán estar en tres estados abiertos, cerrados (soportados por Github Issues) y siendo resuelta (mediante etiquetas).

### 3.2.2. Plan de construcción y despliegue del software

El despliegue del sistema se realizará tanto en nuestras máquinas individuales como en un servidor externo.

En nuestros ordenadores personales se probarán las primeras versiones y haremos los tests locales antes de realizar cambios en nuestro repositorio común, todos los integrantes usaremos una misma imagen de una máquina virtual para hacer las pruebas de despliegue.

En un servidor externo donde estará desplegada siempre la última versión del sistema y será donde accedan los usuarios. Pretendemos usar los servicios de AWS para desplegar ahí una máquina virtual igual a las de nuestras máquinas personales. Todos tendremos acceso a esta máquina por medio de un mismo usuario y contraseña. Tendremos que hacer visible esta máquina a todo el mundo pero solo para conectarse a nuestro videojuego usando la ip y dominio suministrado por Amazon.

Intentaremos seguir el procedimiento básico para realizar una integración continua. Para ello utilizaremos la herramienta de github actions y realizaremos diferentes scripts para:

-Comprobar que el sistema está en ejecución y que no esté en un estado de fallo (script que se conecta remotamente al servidor y que compruebe que el servicio del servidor está en marcha e intentar comprobar que todas las funciones del sistema funcionan).

-Montar el sistema en el servidor (copia el repositorio en el servidor remoto y lo despliega ).

-Scripts de tests automáticos.

Esto nos ayudará en la práctica de integración continua. Cualquier integrante podrá desplegar una nueva versión desde un repositorio de github en el servidor externo, pero también deberá ser posible volver a una versión anterior del sistema de forma rápida usando ese mismo repositorio.

### 3.2.3. Plan de aseguramiento de la calidad

Se diseñarán pruebas unitarias de caja negra para detectar aspectos críticos y prevenir errores en los siguientes componentes:

* Usuario
  + Registro de un nuevo jugador
  + Inicio de sesión
  + Modificación de la información del usuario
  + Cálculo de estadísticas
  + Añadir un jugador a un listado de amigos.
  + Eliminar un jugador de un listado de amigos.
* Partida:
  + Creación de una partida.
  + Eliminación de una partida.
  + Guardar una partida a medias.
  + Modificación de los puntos tras una partida.
* Lógica del juego:
  + Colocar barcos en el tablero.
  + Marcar un disparo en el tablero.
  + Comprobar barcos restantes en el tablero.

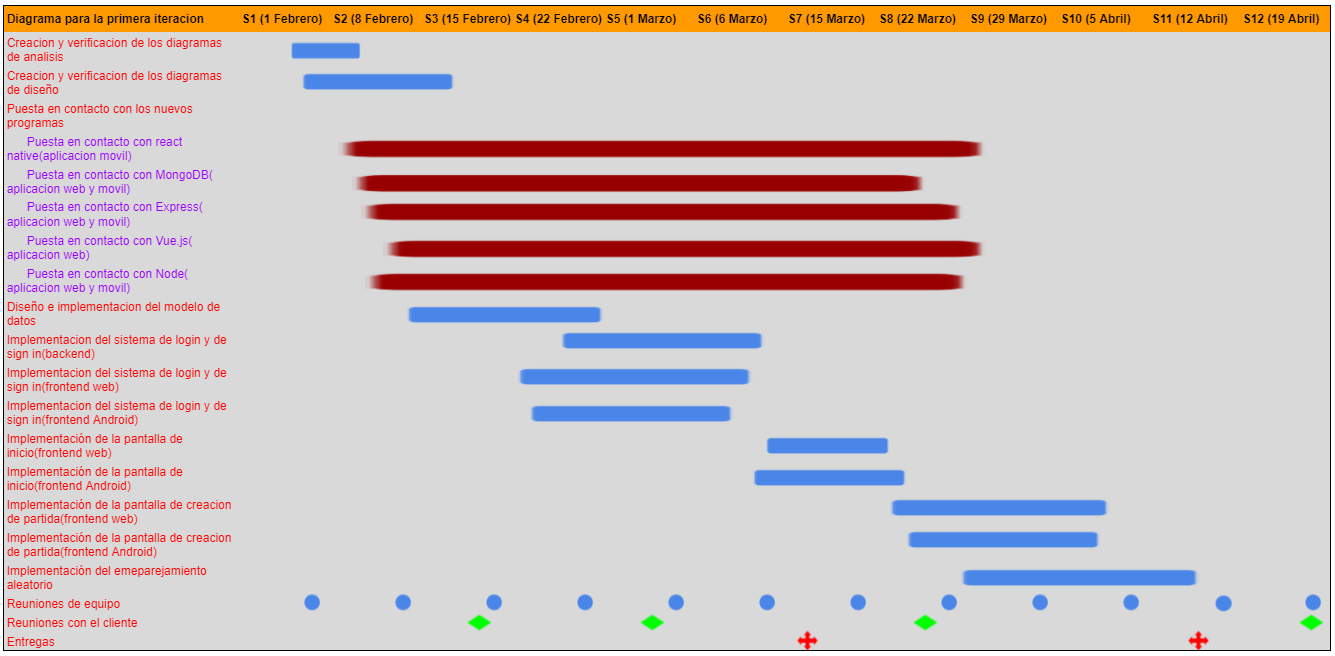
Para verificar el correcto funcionamiento del API y su integración en el frontend, se realizarán pruebas de integración agrupando los componentes por funcionalidad.

Finalmente, sobre el sistema en conjunto se plantearán pruebas manuales de sistema basados en los casos de uso:

* Registrarse.
* Iniciar sesión.
* Modificar y visualizar el perfil.
* Modificar y visualizar el listado de amigos.
* Modo de juego: Contra IA.
* Modo de juego: Emparejamiento a ciegas.
* Modo de juego: Online con amigos.
* Modo de juego: Local.
* Retomar partida a medias en distintos dispositivos.
* Abandonar una partida.
* Consultar partidas creadas y no finalizadas.
* Consultar el histórico de resultados.
* Consultar el ranking de jugadores.
* Compartir perfil.
* Configurar el aspecto del juego.
* Configurar el idioma.
* Activar efectos de sonido.

### 

### 3.2.4. Calendario del proyecto y división del trabajo

El diagrama de Gantt a seguir durante el proyecto es el siguiente:

Requisitos a completar en la primera iteración(22 Abril):

* El sistema debe permitir al usuario registrarse en la aplicación e identificarse cada vez que quiera entrar.
* Los usuarios deben poder jugar a Batalla Naval (ver Anexo I), donde dos jugadores se turnan para realizar movimientos e intentar hundir la flota del rival.
* Los usuarios podrán buscar partidas.
* Debe permitir los modos de juego: emparejamiento a ciegas, local y contra IA.
* Los jugadores podrán rendirse durante una partida.
* Al finalizar una partida, el sistema notificará los resultados a los jugadores.

Requisitos a completar en la segunda iteración:

* El usuario dispondrá de un perfil que incluirá distintas estadísticas (partidas jugadas en total, ganadas, perdidas…).
* Se debe poder agregar a otros usuarios a una lista de amigos.
* Debe permitir los modos de juego: torneo con amigos y online con amigos.
* Se debe poder guardar una partida a medias para retomarla después desde ese u otro dispositivo.
* Si una partida está pausada, una vez que un usuario realice un nuevo movimiento, se le enviará una notificación al oponente.
* Después de cada partida el sistema modificará los puntos asociados a cada uno de los jugadores, sus estadísticas de partidas ganadas y perdidas y sus históricos de partidas.
* Debe proporcionar un listado de las partidas creadas y no finalizadas.
* Cada usuario tendrá un histórico de partidas con sus respectivos resultados el cual podrá consultar.
* El sistema permitirá visualizar un ranking de los jugadores por puntos.
* El usuario podrá compartir su perfil generando un código especial asociado a dicho perfil.
* Se podrá usar un código de usuario para acceder a perfiles de otros usuarios.
* El sistema debe permitir configurar el aspecto del juego (colores de los barcos y del fondo para el tablero).
* El sistema debe permitir configurar el idioma del juego (español, inglés).
* Ciertas partes del sistema contarán con efectos de sonido para aumentar el feedback con el usuario.
* La interfaz deberá ser intuitiva y usable.
* No habrá una diferencia significativa en el rendimiento de la aplicación entre las versiones de la misma.

Las tareas a realizar pueden ser divididas en las siguientes partes:

* Los módulos del software a desarrollar(participan todos los integrantes del grupo)
* La documentación(participan todos los integrantes del grupo)
* El diseño gráfico(los integrantes del frontend son Binhui, Sergio, Santiago y Adrian)
* Instalaciones o despliegues(los encargados del backend son Alberto y Alonso)
* Pruebas manuales(participan todos los integrantes del grupo)

# 

# 

# 4. Análisis y diseño del sistema

## 4.1 Análisis de requisitos

|  |  |
| --- | --- |
| Requisito | Descripción |
| RF1 | El sistema debe permitir al usuario registrarse en la aplicación e identificarse cada vez que quiera entrar. |
| RF2 | Los usuarios deben poder jugar a Batalla Naval (ver Anexo I), donde dos jugadores se turnan para realizar movimientos e intentar hundir la flota del rival. |
| RF3 | Los usuarios podrán buscar partidas |
| RF4 | El usuario dispondrá de un perfil que incluirá distintas estadísticas (partidas jugadas en total, ganadas, perdidas…). |
| RF5 | Se debe poder agregar a otros usuarios a una lista de amigos. |
| RF6 | Debe permitir varios modos de juego (emparejamiento a ciegas, torneo con amigos, online con amigos, local, contra IA). |
| RF7 | Se debe poder guardar una partida a medias para retomarla después desde ese u otro dispositivo. |
| RF8 | Si una partida está pausada, una vez que un usuario realice un nuevo movimiento, se le enviará una notificación al oponente |
| RF9 | Los jugadores podrán rendirse durante una partida. |
| RF10 | Al finalizar una partida, el sistema notificará los resultados a los jugadores. |
| RF11 | Después de cada partida el sistema modificará los puntos asociados a cada uno de los jugadores, sus estadísticas de partidas ganadas y perdidas y sus históricos de partidas. |
| RF12 | Debe proporcionar un listado de las partidas creadas y no finalizadas. |
| RF13 | Cada usuario tendrá un histórico de partidas con sus respectivos resultados el cual podrá consultar |
| RF14 | El sistema permitirá visualizar un ranking de los jugadores por puntos. |
| RF15 | El usuario podrá compartir su perfil generando un código especial asociado a dicho perfil |
| RF16 | Se podrá usar un código de usuario para acceder a perfiles de otros usuarios |
| RF17 | El sistema debe permitir configurar el aspecto del juego (colores de los barcos y del fondo para el tablero). |
| RF18 | El sistema debe permitir configurar el idioma del juego (español, inglés). |
| RF19 | Ciertas partes del sistema contarán con efectos de sonido para aumentar el feedback con el usuario |
| RNF1 | La interfaz deberá ser intuitiva y usable. |
| RNF2 | No habrá una diferencia significativa en el rendimiento de la aplicación entre las versiones de la misma. |

## 4.2. Diseño del sistema

En este apartado se van a detallar diferentes diagramas arquitecturales. Estos diagramas describirán una parte fundamental del proyecto software: Sus elementos, propiedades, y las relaciones entre ellos y con el entorno. La construcción de estos diagramas durante la fase de análisis del proyecto es importante en la medida en que de esta forma los integrantes podrán discutir sobre la estructura del proyecto de forma eficiente, y como resultado se obtendrá un esquema que se utilizará para obtener un proyecto final más organizado.

Debido a la extensión del proyecto, se va a dividir la arquitectura del sistema en diferentes vistas, las cuales conjuntamente darán una visión general de dicho proyecto. Entre estas vistas se encuentran las de módulos, las de componentes y conectores, y las de distribución.

#### Vista de módulos

Esta vista refleja el sistema en tiempo de desarrollo, y permite explicar la estructura del código, enumerando para ello los paquetes o clases y algunas de sus relaciones.

Una primera versión de la vista de módulos del proyecto se encuentra en la Figura 1:

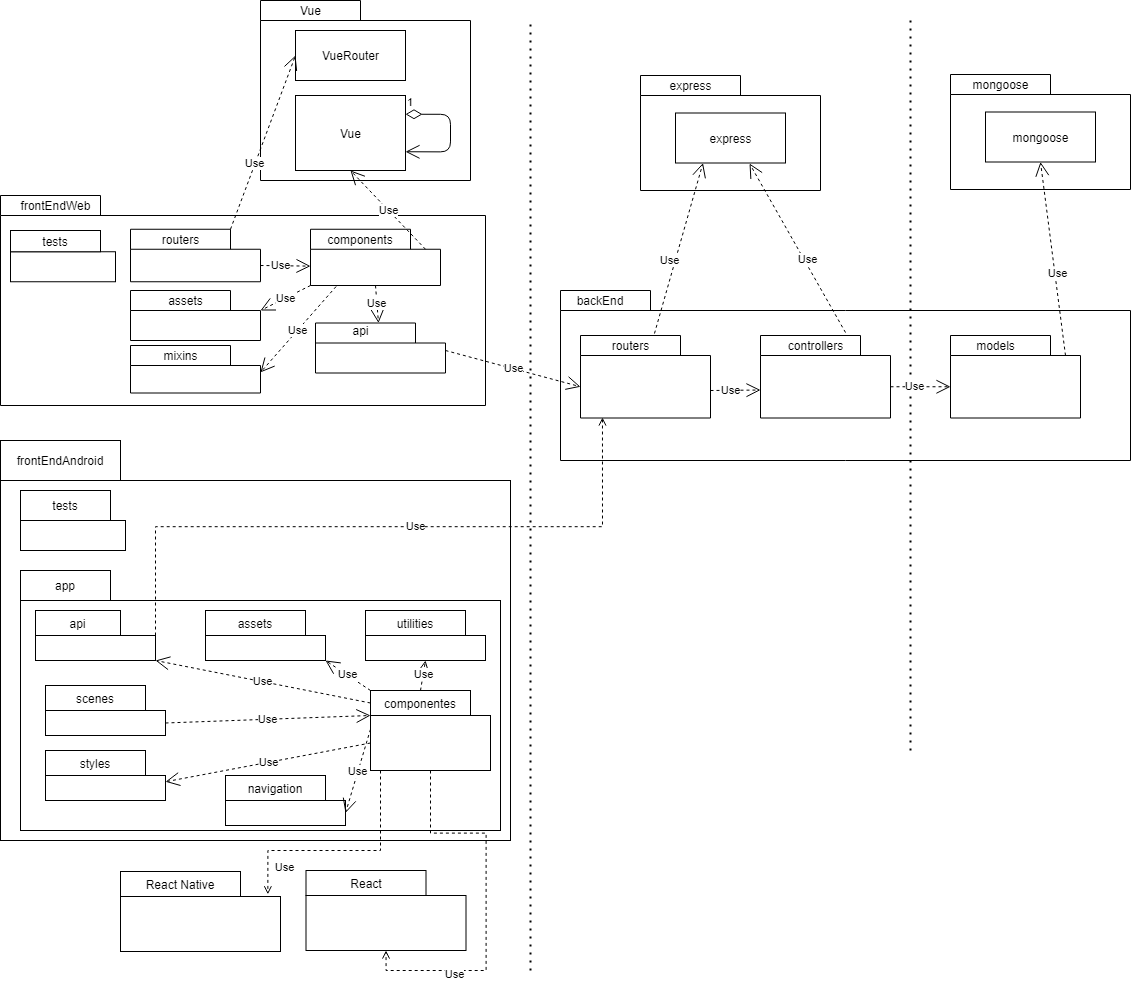


Figura 1: Vista de módulos del sistema

**Paquetes**

* frontEndWeb: contiene los ficheros referentes al frontend web.
  + test: contiene los test unitarios para componentes del frontend web.
  + routers: contiene ficheros que definen la lógica a aplicar cuando el usuario pida ciertas URLs.
  + assets: imágenes y archivos de audio usados por los componentes de Vue.
  + mixins: código común a distintos componentes.
  + components: componentes de Vue que componen la interfaz.
  + api: ficheros con las peticiones http a la API del backend.
* frontEndAndroid: contiene los ficheros referentes al frontend Android.
  + test: contiene los test unitarios para componentes del frontend Android.
  + api: ficheros con las peticiones http a la API del backend.
  + assets: imágenes y archivos de audio usados por los componentes de React.
  + scenes: ficheros que hacen uso de los componentes para conformar las pantallas
  + utilities: código común a distintos componentes.
  + styles: fuentes y colores.
  + navigations: ficheros que contienen la navegación del proyecto.
* backEnd:
  + tests: contiene los tests unitarios para componentes del backend
  + routers: ficheros con la lógica ante distintos métodos http.
  + controllers: ficheros con la lógica de negocio de la aplicación,
  + model: ficheros con los esquemas del modelo de datos y las funciones de interacción con el gestor.

#### Vista de componentes y conectores

Esta vista describe el sistema en tiempo en ejecución, pero no concreta dónde está desplegado. Para ello utiliza los objetos, procesos, etc., y de esta manera se permite razonar sobre cómo se comportará el sistema cuando esté en ejecución, sin importar el hardware sobre el que esté desplegado.

A continuación se encuentra un diagrama donde se especifican los componentes (y sus interfaces), y los conectores del sistema:

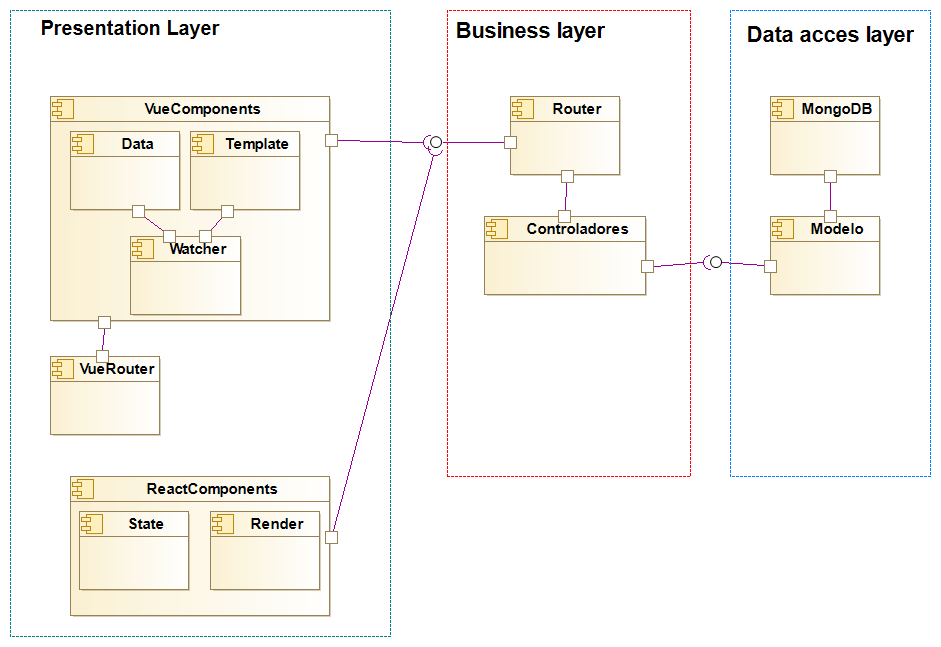
**

Figura 2: Vista de componentes y conectores

**Componentes**

* VueComponents: componentes que encapsulan un componente visual (Template), un dato asociado a este (Data) y una instancia del watcher de Vue, que detecta cambios en Data y llama al renderizador para que vuelva a renderizar el componente. Con estos componentes se consituye la interfaz web.
* VueRouter: componente que, ante una petición de un recurso via URL, intercepta esta para darle respuesta directamente desde el navegador. Esta es la forma que tiene Vue de crear páginas. Desde los routers pueden instanciarse componentes para que al introducir una URL al usuario se le devuelva una página concreta.
* ReactComponents: de forma similar a los componentes de Vue los componentes de react encapsulan datos (State) y un componente gráfico (Render). Un cambio en State provoca un rerenderizado del componente. El componente gráfico, que en Vue era html y css, en este caso son “core components” de React Native (elementos de GUI).
* Router: componentes que actúan con la llegada de peticiones http, definiendo las llamadas a controladores requeridas para responder a un método http con una URL específica.
* Controladores: Componentes que piden a Modelo los datos requeridos por los *routers* y los transforma,
* Modelo: Componente que recibe peticiones de los controladores y realizan consultas al gestor de bases de datos para atenderlas.
* MongoDB: Gestor de bases de datos.

#### Interfaces importantes

Destaca la importancia de la API RPC que ofrece el backend a los frontend. Mediante métodos http POST a las siguientes URLs, el usuario tendrá a su disposición las siguientes operaciones con objetos JSON:

* /user/login: el cuerpo de la petición debe contener un objeto de tipo User. Si no existe un usuario con ese nombre o correo lo crea y devuelve una confirmación. De existir devuelve un mensaje de error.
* /user/signin: el cuerpo de la petición debe contener un objeto de tipo User. Si existe el usuario y la contraseña es correcta devuelve una confirmación. Si no, devuelve un mensaje de error.
* /game/random: el cuerpo de la petición debe contener un objeto de tipo User. Devuelve un dato de tipo Game cuando se encuentra un rival.
* /game/friend: el cuerpo de la petición debe contener dos objetos de tipo User. El propio y el del amigo al que se quiere retar. Devuelve un objeto de tipo Game cuando el rival acepta.
* /game/tournament; .el cuerpo de la petición debe contener un número potencia de 2 de objetos de tipo User. Devolverá un objeto de tipo Game con el primer enfrentamiento.
* /game/ia: el cuerpo de la petición debe contener un objeto de tipo User. Devuelve un objeto de tipo Game.
* /game/movement: el cuerpo de la petición debe contener un objeto de tipo Game con un movimiento posible para el Game con ese id almacenado en el servidor. Si es el movimiento ganador devolverá un mensaje informando de la victoria. Si es el movimiento ganador de una partida de torneo devolverá además un objeto de tipo Game con el siguiente enfrentamiento. Si es el movimiento ganador de una última partida de torneo se enviará un mensaje informando de esto.
* /user/user: el cuerpo de la petición debe contener un objeto de tipo User. Se devuelve el objeto user del mismo id almacenado con todas sus propiedades relevantes.
* /user/friendlist: el cuerpo de la petición debe contener un objeto de tipo User. Se devuelve una lista de objetos User correspondiente a los amigos del usuario de la petición.
* /user/addfriend: el cuerpo de la petición debe contener dos objetos de tipo User, uno correspondiente al solicitante y otro al destinatario de la petición de amistad. Se devuelve un mensaje cuando el destinatario acepte o rechace la invitación.
* /user/accept: el cuerpo de la petición debe contener dos objetos de tipo User, uno correspondiente al solicitante y otro al destinatario de la petición de amistad.
* /user/dismiss: el cuerpo de la petición debe contener dos objetos de tipo User, uno correspondiente al solicitante y otro al destinatario de la petición de amistad.
* /user/ranking: el cuerpo de la petición debe contener un objeto de tipo User. Se devolverá una lista de objetos de tipo User con los objetos usuario correspondientes a los 5 jugadores que están por encima del de la petición seguidos de los 5 que están inmediatamente debajo de él.
* /user/share: el cuerpo de la petición debe contener un objeto de tipo User. Se devolverá una URL que proporcionará el perfil del usuario.

#### Vista de distribución en estilo de despliegue

En este último tipo de vista sí que se tiene en cuenta el hardware sobre el que va a correr la aplicación, ya que se refleja el comportamiento del sistema en tiempo de ejecución en un despliegue en concreto.

En la figura 3 se encuentra una primera versión de la vista de distribución del sistema:

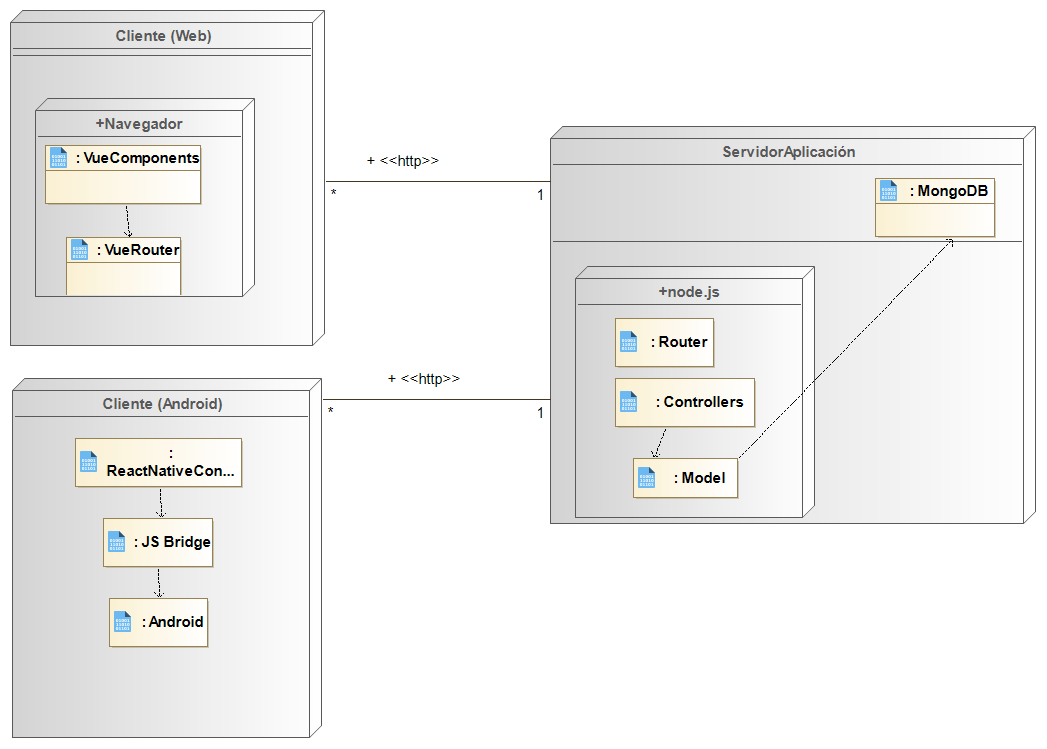


Figura 3: Vista de distribución del sistema

# Anexo I. Reglas de juego de Batalla Naval

### **Tableros**

Los jugadores manejan un tablero de océano y un tablero de tiro; cada uno divididos en 100 casillas (10 filas x 10 columnas).​ Cada tablero representa una zona diferente del mar abierto: la propia y la contraria. En el primer tablero, el jugador coloca sus barcos y registra los «tiros» del oponente; en el otro, se registran los tiros propios contra el otro jugador, diferenciando los impactos y los que dan al agua. Al tiempo, se deduce la posición de los barcos del contrincante.

### **Naves**

Cada jugador dispondrá de una flota de 5 barcos:

* 1 portaaviones: ocupa 5 casillas
* 1 buque: ocupa 4 casillas
* 2 submarinos: ocupa 3 casillas cada uno
* 1 crucero: ocupa 2 casillas

Al comenzar una partida, cada jugador posiciona sus barcos, ocupando las casillas correspondiente a un tipo de barco horizontal o verticalmente, en el primer tablero, de forma secreta, invisible al oponente. No se podrán colocar barcos pegados entre sí en la misma dirección.

### **Resultados de un tiro**

Agua: cuando disparas sobre una casilla donde no está colocado ningún barco enemigo, disparas al agua.

Tocado: cuando disparas en una casilla en la que está ubicado un barco enemigo que ocupa 2 o más casillas y destruyes sólo una parte del barco, le has tocado.

Hundido: si disparas en una casilla en la que está ubicado un barco con el resto de casillas tocadas, le has hundido, es decir, has eliminado ese barco del juego.

### **Desarrollo del juego**

Una vez todas las naves han sido posicionadas, se inicia una serie de rondas. En cada ronda, cada jugador en su turno «dispara» hacia la flota de su oponente indicando una posición (las coordenadas de una casilla), la que registra en el segundo tablero.

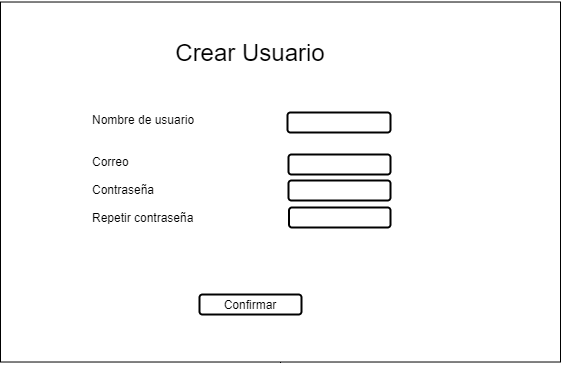
En el segundo tablero se marcará, el resultado del tiro, en caso de ser Agua, pasa el turno a su oponente y en caso de Tocado o Hundido, vuelves a disparar, siempre y cuando no hayas hundido toda la flota de tu enemigo, en cuyo caso habrás ganado.

# Anexo II. Prototipo de pantallas y mapa de navegación del juego.

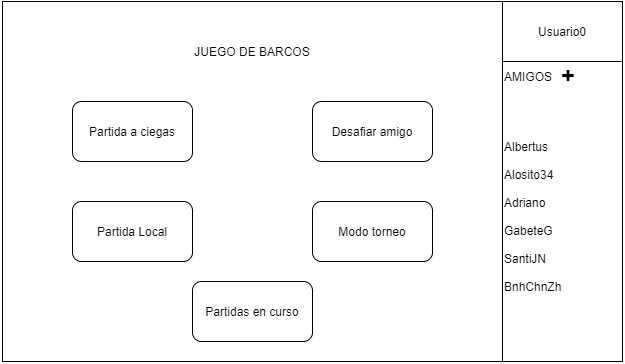
## Pantallas

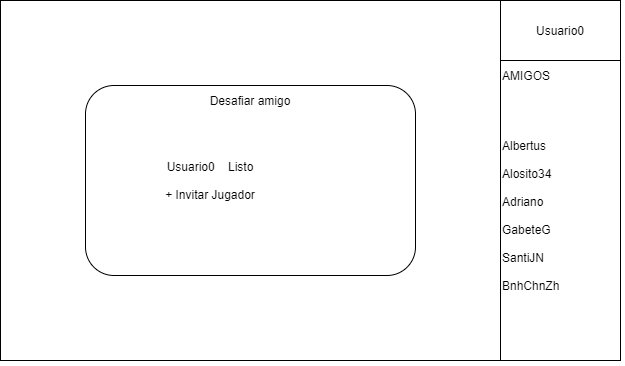
Pantalla de inicio de sesión, es la primera pantalla del sistema.



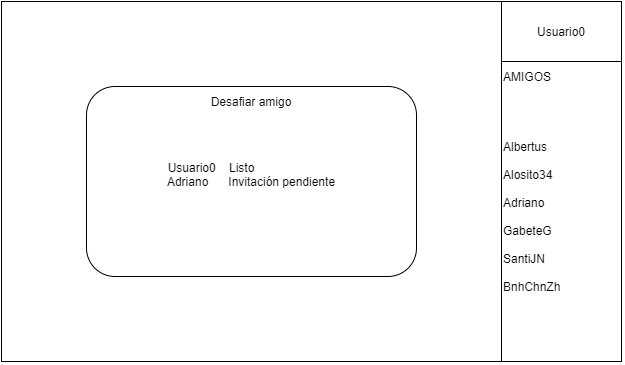
Pantalla de creación de usuario

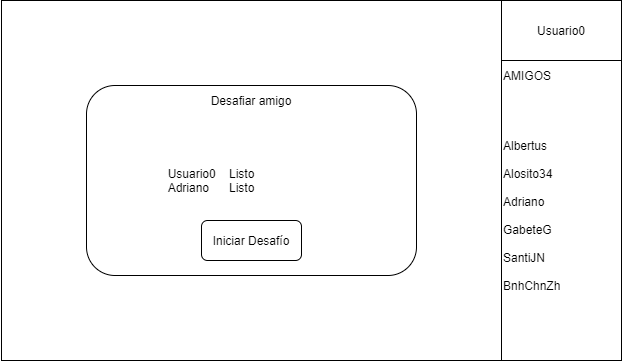
Pantalla que hace de menú principal, en él se muestran los distintos modos de juego.



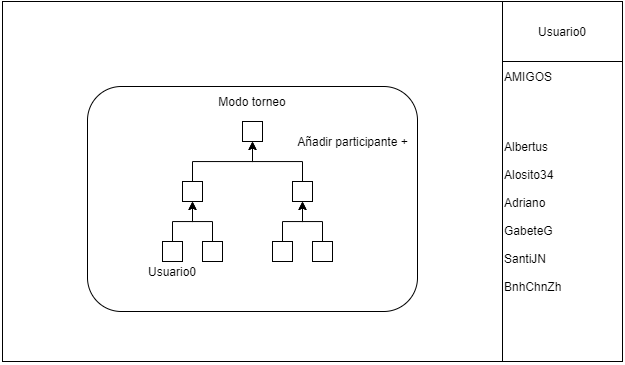
Pantalla de desafío amistoso en la que seleccionas un usuario de tu lista de amigos para retarle a un duelo.

Pantalla de confirmación de petición de duelo. Esperando a que el jugador desafiado acepte tu desafío.

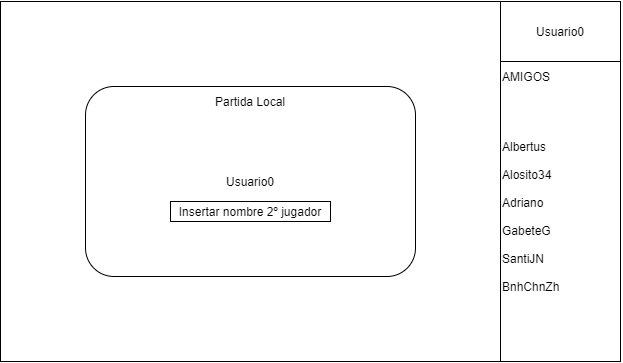


Una vez el rival haya aceptado el desafío, puedes iniciar la partida.

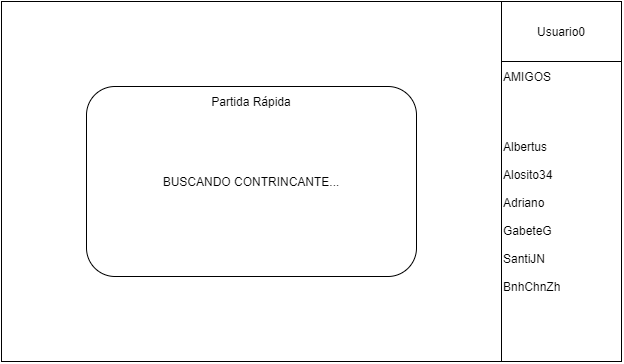
Pantalla de creación de torneo, puedes invitar a otros jugadores a unirse al torneo.



Pantalla de creación de partida local.

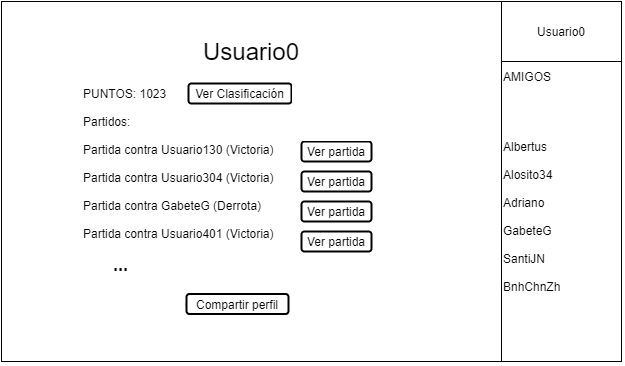


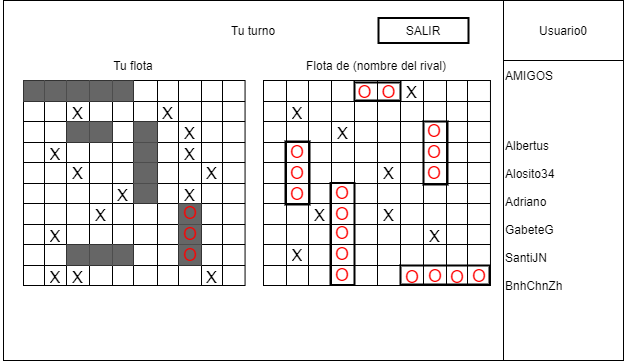
Pantalla de búsqueda de contrincante para una partida rápida.



Pantalla de contrincante encontrado para partida rápida

Pantalla de consulta de perfil de usuario.



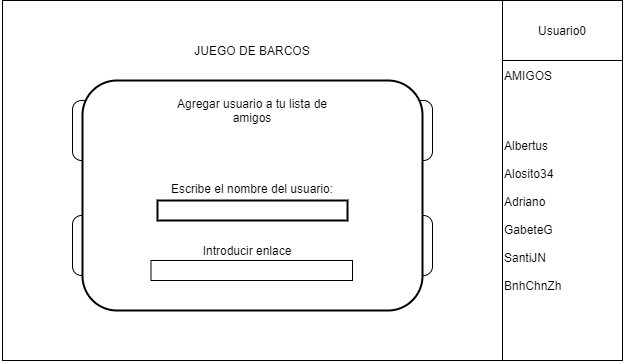


Pantalla de visualización de una partida acabada.

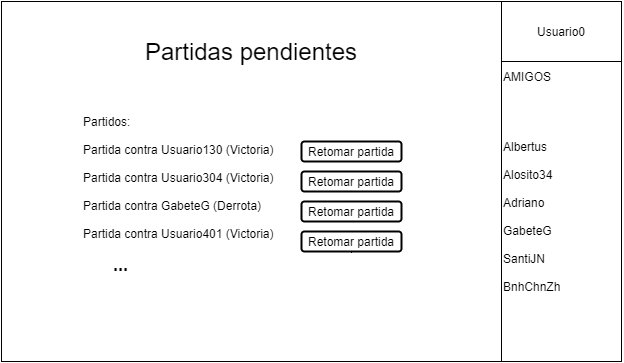
Pantalla de clasificación donde se muestran la lista de jugadores con más puntos.



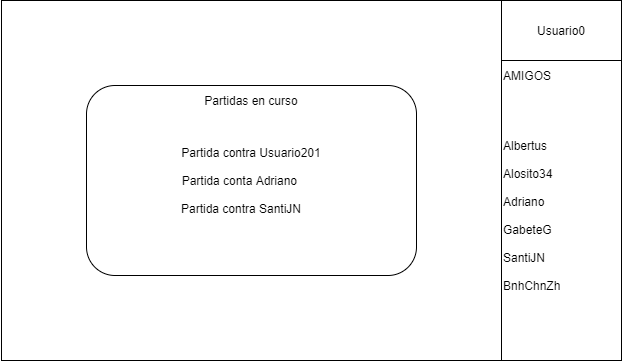
Pantalla de solicitud de amistad.



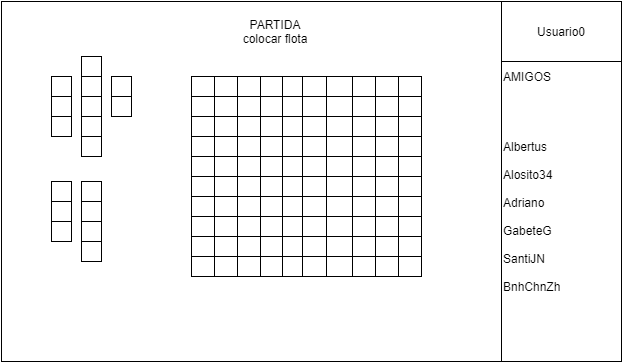
Pantalla de lista de partidas pendientes de que juegues tu turno.



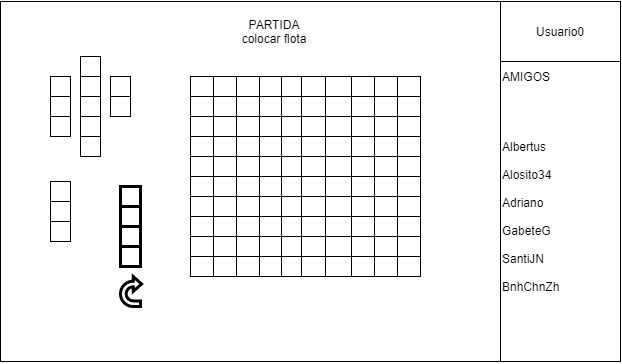
Pantalla de lista de partidas pendientes de que tu rival juegue su turno.



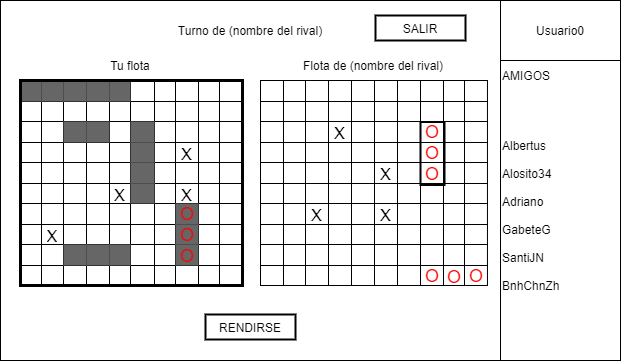
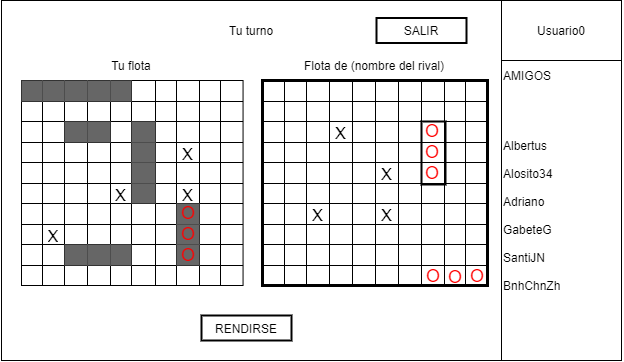
Primera pantalla de una partida en la que colocas tus barcos en el tablero.



Puedes rotar un barco seleccionado



Pantalla de turno del usuario.

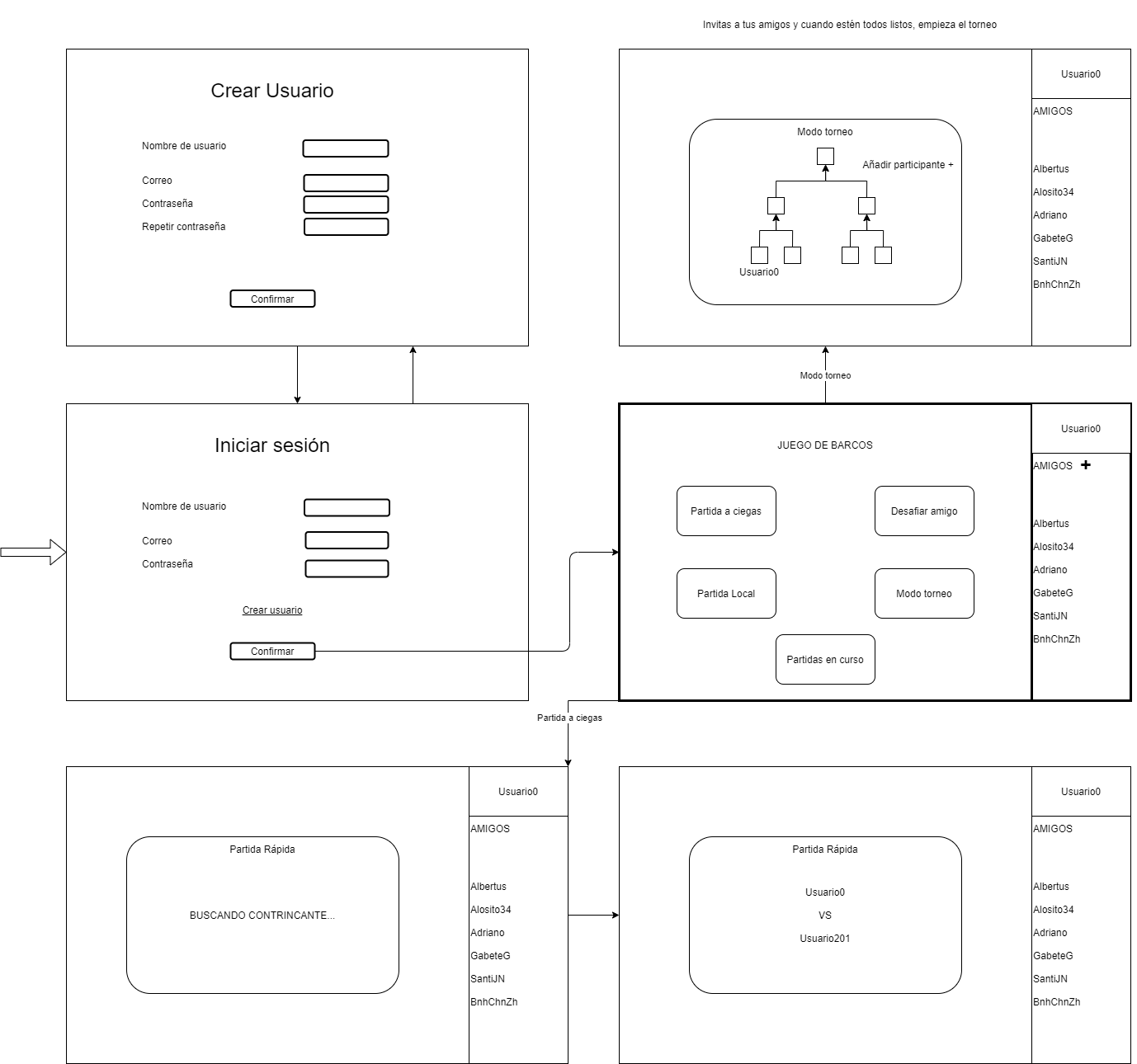


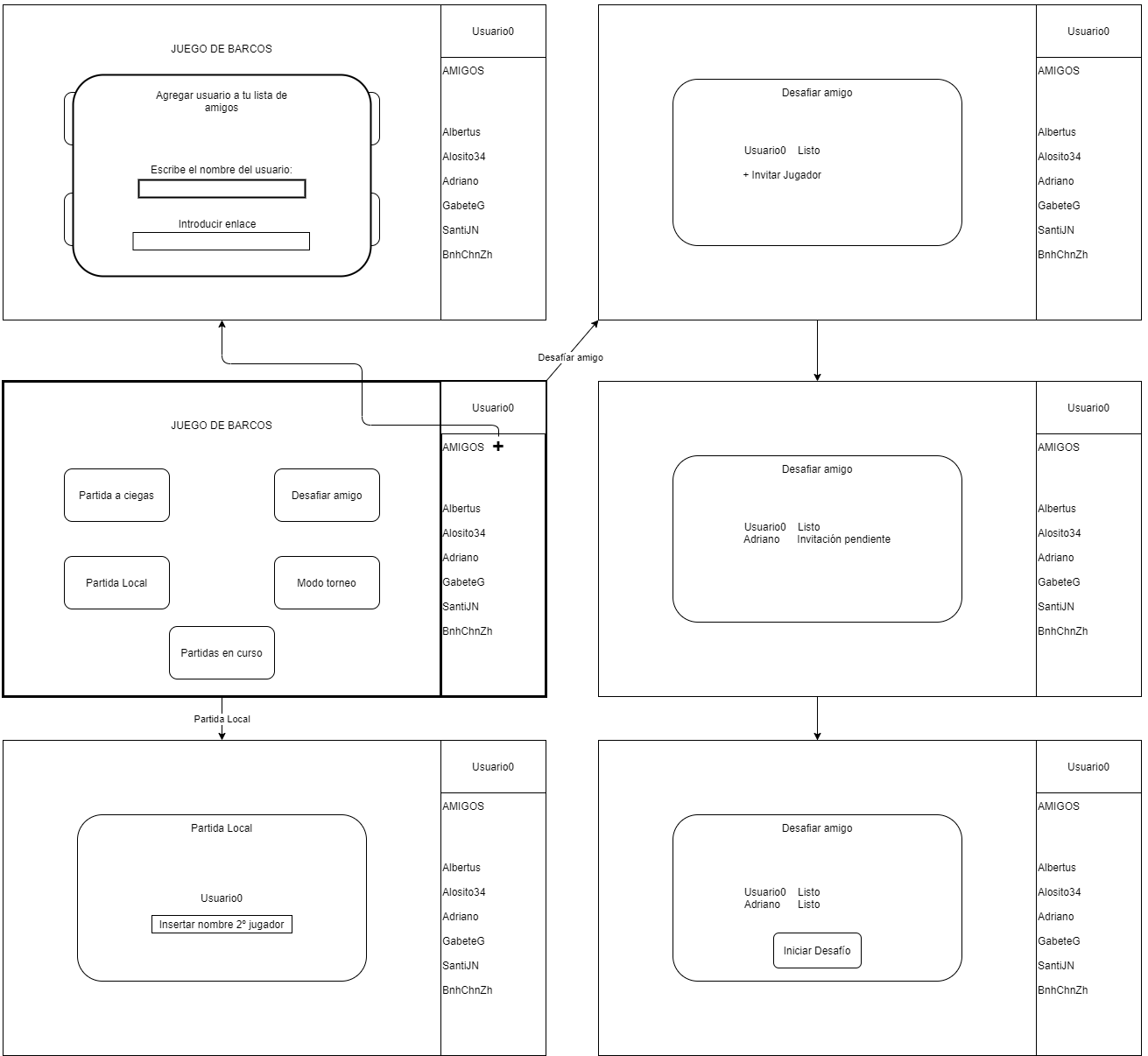
Pantalla de turno del rival.

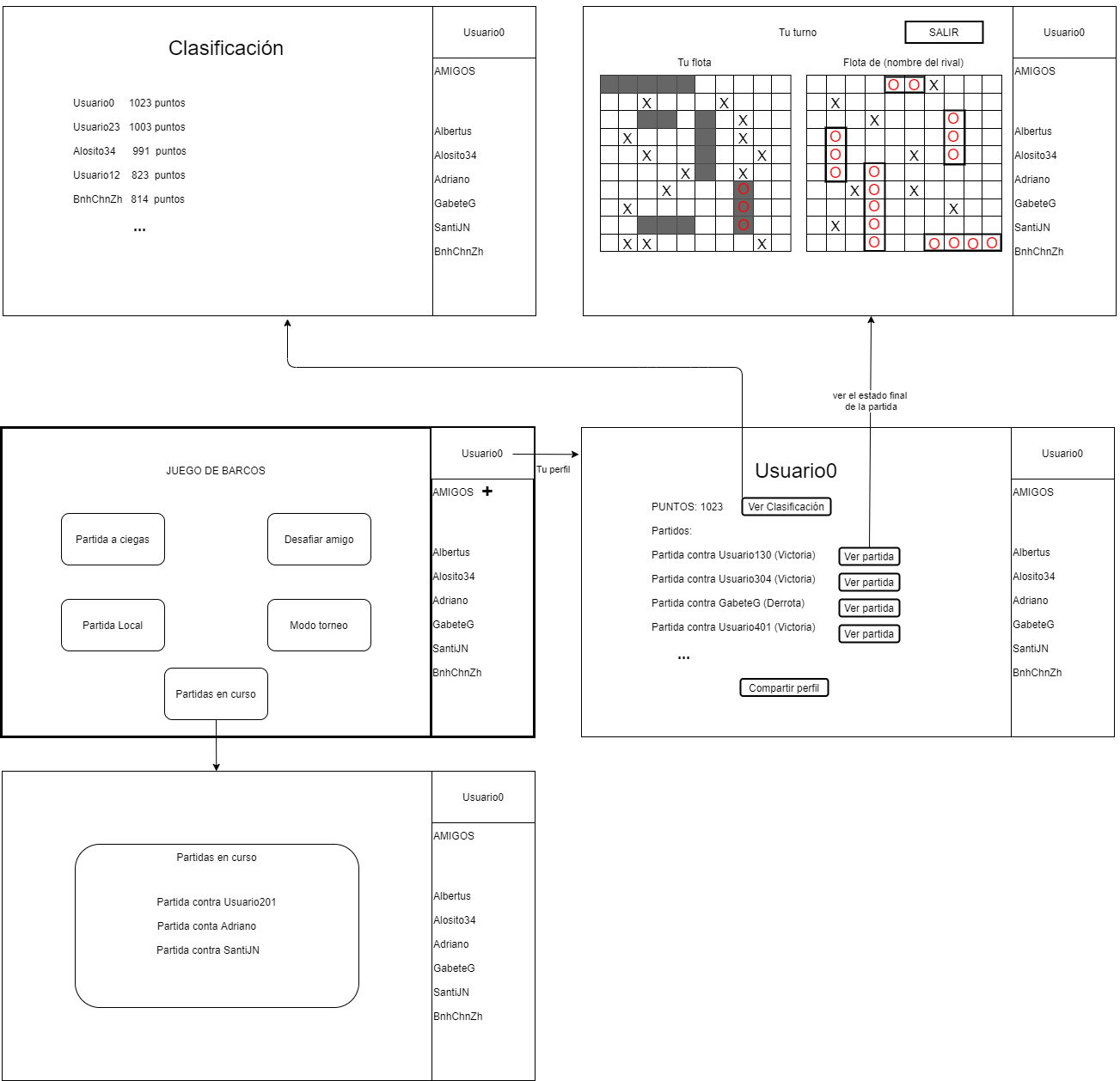
## Mapa de navegación

Ya que el mapa completo no cabe en una página, estará dividido en varias partes.

Inicio de sesión, creación de usuario, modo torneo y partida rápida



Agregar amigo, desafiar amigo y partida local

Ver perfil, clasificación, partidas en curso y ver una partida finalizada

Pantallas de una partida de hundir la flota