

laSalle

UNIVERSITAT RAMON LLULL

**Escola Tècnica Superior d'Enginyeria
Electrònica i Informàtica La Salle**

Trabajo Final de Máster

Máster en Desarrollo y Arquitectura de Software

Vibrations Game

Alumno
Cristian Vega Sánchez

Profesor Ponente
Eduardo de Torres

Mentor
Alex Soto

ACTA DEL EXAMEN DEL TRABAJO FINAL DE MÁSTER

Reunido el Tribunal calificador en la fecha indicada, el alumno

D. Cristian Vega Sánchez

expuso su Trabajo Final de Máster, titulado:

Vibrations Game

Acabada la exposición y contestadas por parte del alumno las objeciones formuladas por los Sres. miembros del tribunal, éste valoró dicho Trabajo con la calificación de

Barcelona,

VOCAL DEL TRIBUNAL

VOCAL DEL TRIBUNAL

PRESIDENTE DEL TRIBUNAL

Contenido

1. Resumen	6
2. Summary.....	7
3. Introducción.....	8
4. Marco de trabajo	9
GitLab	10
Docker	11
Kubernetes	12
Office	14
Diagrama de contexto	15
5. Objetivos.....	16
Generales	16
Individuales	16
6. Frontend - Interfaz grafica.....	17
Introducción.....	17
Tecnologías utilizadas	17
6.1. Requerimientos	20
Requerimientos propuesta	20
Requerimientos funcionales	20
Requerimientos no funcionales	21
6.2. Especificaciones.....	22
Casos de uso del usuario Player	22
Casos de uso del usuario Administrador	23
6.3. Diseño	24
Diseño de la aplicación UI/UX.....	24
Responsivo	25
LottieFiles	26
Construcción del proyecto sobre React	27
6.4. Desarrollo.....	28
Arquitectura	28
Estructura de directorios	30
Sistema de navegación.....	31
Interacción con el servidor	31
Sistema de orientación en dispositivos móviles	34
Resultados de la implementación	36

6.5. Testing	44
Jest	44
Diseño del test	44
Diseño e implementación de pruebas	45
6.6. Generación imagen Docker:	46
7. Planificación y gestión del proyecto.....	47
Kubernetes	48
8. Análisis de resultados	50
9. Estudio económico	51
Coste desarrollo	51
Coste del software	51
Coste infraestructura.....	51
Coste total	52
10. Conclusiones	53
11. Líneas de futuro	54
12. Bibliografía	55

1. Resumen

Vibrations Game nace de la necesidad de tener una aplicación con la que demostrar las capacidades de Apache Kafka como plataforma para crear y procesar stream de datos. Para poder realizar este objetivo se ha pensado en implementar una aplicación que haga uso de esta plataforma.

La aplicación resultante trata de un juego con el que un jugador, mediante un frontend desplegado en un dispositivo móvil y haciendo uso de los sensores de movimiento y aceleración, capture coordenadas y las envíe a un backend usando el protocolo de comunicaciones Websocket. Este frontend será desarrollado con JavaScript usando el framework de React.

El backend será una aplicación desarrollada en Java con el uso de framework de Spring. Entre otras funcionalidades, tendrá que interactuar con el Apache Kafka para guardar, recuperar y procesar las coordenadas recibidas.

Para terminar, también habrá un módulo de Machine Learning desarrollado con la plataforma de código abierto de Tensorflow que se encargará de comprobar qué movimiento corresponde un conjunto de coordenadas. Este módulo será consumido por el backend. Este módulo se desarrollará usando Python como lenguaje de programación.

2. Summary

Vibrations Game was born from the need to have an application with which to demonstrate the capabilities of Apache Kafka as a platform to create and process data streams. In order to achieve this objective, it was decided to implement an application that makes use of this platform.

The resulting application is about a game with which a player, through a frontend deployed on a mobile device using the motion and acceleration sensors, captures coordinates and sends them to a backend using the Websocket communications protocol. This frontend will be developed in JavaScript using the React framework.

The backend will be an application developed in Java with the use of the Spring Framework. Among other functionalities, it will have to interact with Apache Kafka to save, retrieve and process the received coordinates.

Finally, there will also be a Machine Learning module developed with the Tensorflow, an opensource platform, that will oversee the checking of which movement corresponds a given set of coordinates. This module will be consumed by the backend. This module will be developed using Python as the programming language.

3. Introducción

Vibrations Game es un proyecto que desarrolla un juego para dispositivos móviles en el cual se muestran figuras a realizar en estos dispositivos y el jugador tiene que intentar de reproducirlas con el máximo de fidelidad posible.

Esta aplicación es un proyecto nacido de la necesidad de demostrar las capacidades de procesamiento de la plataforma de Apache Kafka. Apache Kafka es una plataforma de código abierto que permite crear y procesar streams de datos.

Para poder realizar esta operativa será necesario la implementación de varios módulos para que puedan interactuar entre ellos y así ofrecer la funcionalidad deseada.

El primero de los módulos es el frontend. Este módulo se desarrolla utilizando el framework de React, utilizando JavaScript como lenguaje de programación. Está pensado para ser usado en dispositivos móviles y será el que leerá los instrumentos del dispositivo para obtener la información de las coordenadas en el transcurso de un movimiento. También será el módulo con el que los usuarios interactúan directamente, tanto para operaciones administrativas como crear una partida o para operaciones no administrativas como unirse a una partida creada.

El siguiente módulo será el backend. Su función será la de procesar toda la información que se recibe del frontend. A más, será también el responsable de comunicarse con otros módulos para poder ejecutar la operativa que se espera de él. Esta operativa incluye tanto tareas administrativas como no administrativas. Estará implementado con Java con el soporte del framework de Spring.

Seguidamente tenemos el módulo de Apache Kafka. Este módulo irá también acompañado de la aplicación Zookeeper. La funcionalidad que se espera de este módulo es la de recibir los mensajes del módulo del backend y de guardarlos para poder ser procesados posteriormente. Estos mensajes contendrán las coordenadas capturadas por el módulo de frontend y que han estado enviadas al módulo de backend.

El último de los módulos es una aplicación de Machine Learning. Este módulo estará desarrollado con la plataforma de TensorFlow. Más concretamente se usará el lenguaje de programación Python. Su funcionalidad será la de poder detectar un patrón de movimiento dado un conjunto de coordenadas recibidas. Estas coordenadas necesitan estar ordenadas en el tiempo para poder hacer los cálculos adecuados. La obtención y el envío a este módulo del conjunto de coordenadas será tarea del módulo de backend. De esta manera este módulo solamente se deberá de centrar en la funcionalidad de Machine Learning.

4. Marco de trabajo

Este proyecto ha estado propuesto por Alex Soto, un profesor del Master de Desarrollo y Arquitectura de Software que se imparte en La Salle, en la Universidad Ramon Llull. Alex Soto también trabaja para la compañía Red Hat y este proyecto ha surgido de la necesidad de Red Hat de querer posicionarse como pioneros en arquitecturas basadas en eventos y el procesamiento de stream en tiempo real en entornos de Kubernetes. Para hacerlo necesitan crear aplicaciones que demuestren la usabilidad de esta arquitectura. Vibrations Game, la aplicación que se desarrolla en este proyecto, es una de ellas.

Vibrations Game tiene que ser una aplicación web multijugador que tiene que funcionar en dispositivos móviles como teléfonos móviles o tabletas en el que al usuario se le presenta un patrón de movimiento a realizar y este tendrá que ejecutar el movimiento en este dispositivo seguido de este patrón.

Para poder implementarlo, se ha dividido el proyecto en las siguientes partes:

- Frontend
- Backend + Apache Kafka
- Tensorflow
- Kubernetes

Estas cuatro partes se dividen entre los tres miembros que desarrollan este proyecto.

Uno de los puntos a destacar es que, a la hora de realizar el despliegue de la aplicación únicamente solo habrá una instancia por cada una de las partes desarrolladas. Esto implica que la parte del frontend, backend, Apache Kafka y Tensorflow únicamente tendrán una instancia desplegada en el entorno de Kubernetes.

GitLab

En cuanto a la localización del código implementado, se ha decidido utilizar GitLab como repositorio de código. Para hacer esto, se ha creado un grupo llamado MDAS TFM que se encuentra en el siguiente link: <https://gitlab.com/mdastfm>. Dentro de este grupo se ha creado tres proyectos y otro grupo. Estos son los siguientes:

- Front end Development: en este proyecto encontraremos el código de la parte frontal del proyecto.
- Backend: Aquí estará el código de la parte del backend del proyecto más la parte de Apache Kafka.
- Machine Learning Projects: en este grupo estarán los proyectos que se han desarrollado para la parte de Machine Learning. Dentro estarán los proyectos siguientes:
 - o MotionRecorder: es la aplicación que permite grabar los datos para poder entrar el modelo de Machine Learning.
 - o MotionPredictor: es el proyecto en el que se podrá entrenar un modelo de datos con los datos obtenidos en el MotionRecorder y a demás contiene una API para que el backend la pueda atacar.
- Kubernetes: en este proyecto encontraremos los scripts necesarios para poder desplegar la aplicación de Vibrations Game en un entorno de Kubernetes.

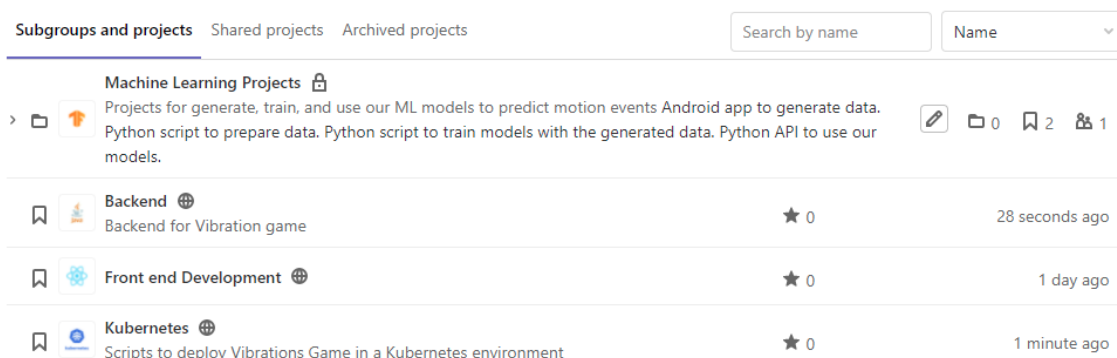


Fig. 1: Detalles del grupo MDAS TFM en GitLab

El GitLab no solo nos ofrece el recurso de repositorio de código, sino que también nos ofrece funcionalidades como la creación de tickets y pipelines CI/CD que también serán usadas en el desarrollo de este proyecto.

Docker

Docker es una plataforma de código abierto que nos permite desarrollar, empaquetar y ejecutar aplicaciones. Docker te permite separar tus aplicaciones de la infraestructura necesaria para poder ejecutarlas. Esto lo puede hacer, ya que te permite gestionar la infraestructura de la misma manera que lo haces con una aplicación, con código. Es lo que se dice IaaS (Infrastructure as a Code). Esta metodología de trabajo proporciona ventajas a la hora de empaquetar, testear y desplegar aplicaciones, ya que permite reducir significativamente el tiempo que se tarda entre escribir el código y desplegar la aplicación a producción.

La arquitectura de Docker está basada en cliente-servidor. Esto básicamente implica que el cliente se comunica con la parte servidor y es este último el que realiza todo el trabajo.

Aunque la parte del cliente y servidor se pueden comunicar remotamente, Docker también permite que las dos partes puedan estar en el mismo sistema.

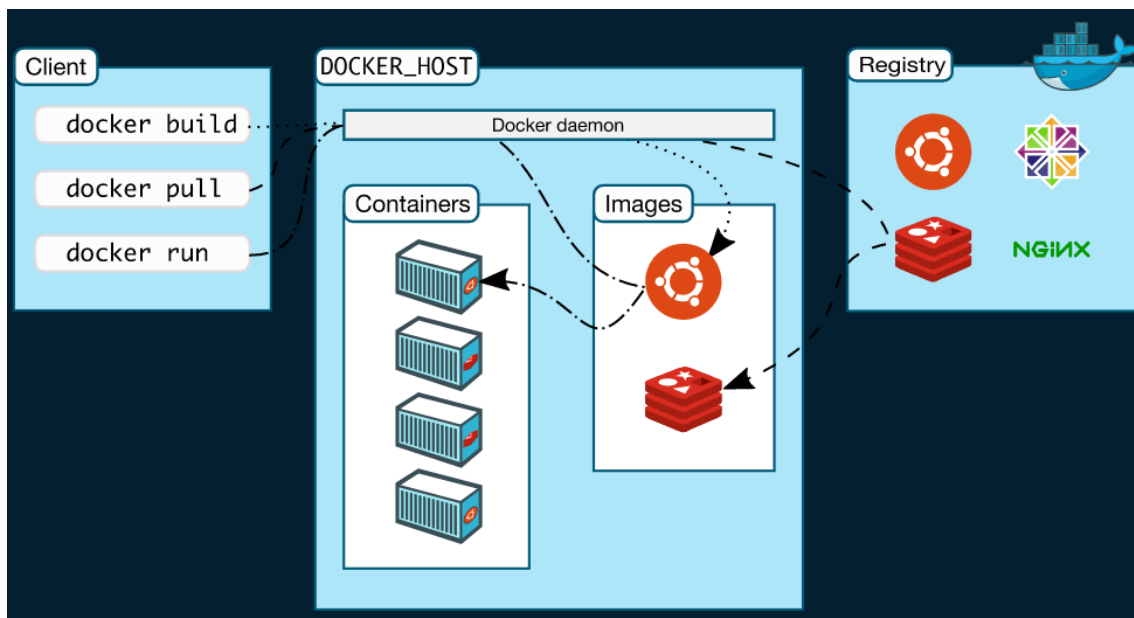


Fig. 2: Detalles de la arquitectura Docker

En la arquitectura Docker podemos encontrar los siguientes elementos:

- Docker Daemon: es la parte servidora. Procesa las peticiones recibidas en las API REST y también gestiona los objetos de Docker como a hora imágenes, contenedores, redes y volúmenes.
- Docker client: es la parte del cliente. Se encarga de la comunicación con el usuario y el servidor. Recibe las comandas como por ejemplo “Docker build” y las envía al servidor para que este las ejecute.
- Docker registry: el registro de Docker es el que guarda las imágenes Docker. Básicamente funciona como un repositorio de imágenes Docker. Estos registros pueden ser públicos o privados.
- Imágenes: se trata de un objeto de Docker de solo lectura que contiene un conjunto de instrucciones para crear un contenedor Docker. Docker te permite crear tantas imágenes propias como usar imágenes que ya estén presentes en los repositorios. Para crear tu propia imagen, es necesario un fichero Dockerfile que contenga las instrucciones necesarias para crear y ejecutar la imagen.

- Contenedores: un contenedor es una instancia de una imagen que está en ejecución. En otras palabras, un contenedor se crea a partir de una imagen Docker ya definida. Es por esto que un contenedor se define por su propia imagen.

Kubernetes

Kubernetes es una plataforma de código abierto utilizada para automatizar despliegues, escalar y gestionar aplicaciones desplegadas en contenedores.

En un entorno de producción necesitarás gestionar los contenedores que ejecutan las aplicaciones y asegurar de que no hay interrupciones en los servicios. Por ejemplo, si un contenedor se cae, otro contenedor se debería levantar para poder sustituirlo.

Kubernetes provee un framework que permite ejecutar sistemas distribuidos resistentes a fallos. Se encarga de escalar las aplicaciones dependiendo de la carga de peticiones, controla el fallo levantando nuevos contenedores y ofrece patrones de despliegue entre otros.

También te provee de los recursos siguientes:

- Detección de servicios y balanceo de carga: Kubernetes te permite exponer un contenedor, usando el nombre del DNS o su propia IP de tal manera que se pueda recibir peticiones. Además, si el tránsito de datos es alto, te permite hacer balanceo de carga y distribuirlo entre las diferentes instancias de una misma aplicación.
- Orquestación de almacenamiento: Kubernetes permite montar automáticamente un sistema de almacenamiento, como ahora en local o en proveedores cloud entre otros.
- Automatización de despliegues y retrocesos: se puede describir el estado deseado de un despliegue de contenedores. Kubernetes es entonces capaz de pasar del estado actual a un estado deseado automáticamente.
- Limitación de recursos: en Kubernetes puedes definir cuanta CPU y memoria RAM necesita cada contenedor.
- Recuperación automática: Kubernetes puedes reiniciar un contenedor que ha fallado, reemplazándolo o parar contenedores que no responden.
- Gestión de configuración y de secrets: Kubernetes permite almacenar y gestionar la información sensible, como por ejemplo contraseñas, tokens, etc. Además, puedes desplegar y modificar la configuración y los secrets de las aplicaciones sin tener que volver a generar imágenes de contenedores.

En la arquitectura de Kubernetes encontramos los elementos de la siguiente imagen:

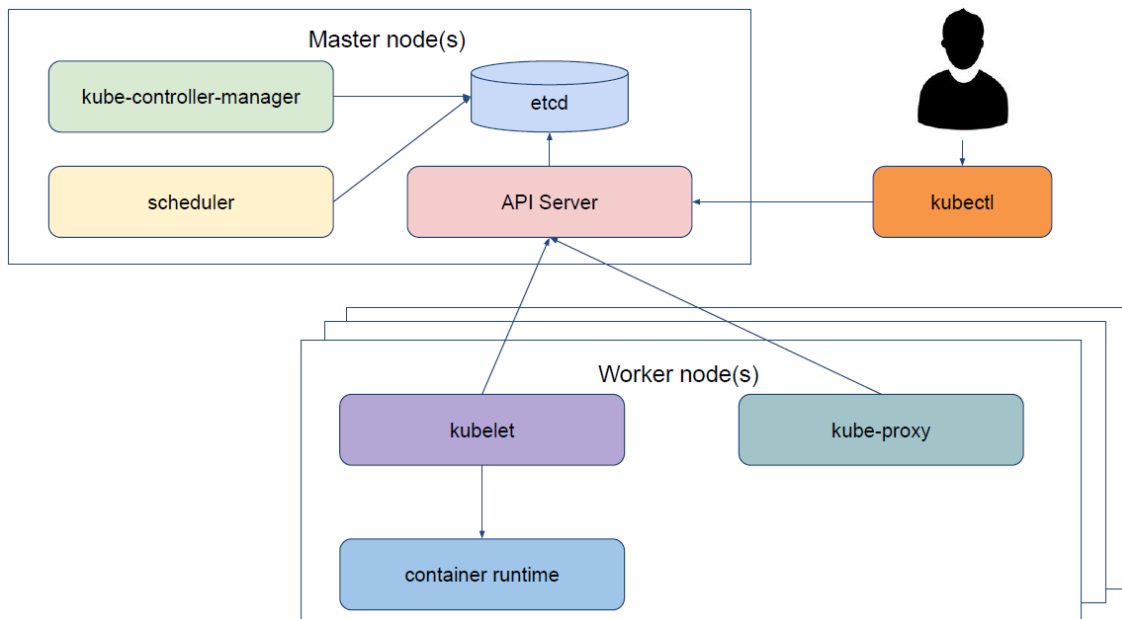


Fig. 3: Detalles de la arquitectura de Kubernetes

Un cluster de Kubernetes consiste en un conjunto de máquinas, llamadas nodos.

Estos nodos pueden ser:

- Nodos master: proveen el panel de control del cluster. Los componentes del node master toman decisiones globales sobre el cluster a más de detectar y responder los eventos del cluster como ahora levantar un nuevo pod.
- Nodos worker: son los responsables de correr los pods y de proveer el entorno de runtime en Kubernetes.

El **API Server** es el componente dentro del node master que recibe las peticiones REST, las valida y las procesa. El estado resultante del cluster después de ejecutar las peticiones es guardar una key-value store distribuida. El único componente que puede acceder a esta store es el API Server. Como resultado de esto, esta componente actúa como master de todo el cluster.

El **etcd** es la key-value store distribuida que guarda el estado deseado del cluster.

El **Controller Manager** es el componente que gestiona los ciclos de control. Su función consiste en comparar continuamente el estado deseado definido en el etcd con el estado actual que se encuentra en los nodes worker. En el caso de que exista una diferencia entre los dos estados, este componente actúa para que el estado actual llegue al estado deseado.

También hay un componente opcional, el **Cloud Controller Manager**. Este componente interactúa con los proveedores de cloud para gestionar los recursos como ahora los balanceos de carga y las rutas de la red.

El **Scheduler** es un componente del node master que controla los pods creados que no tienen un nodo asignado y los selecciona un nodo en el cual se puedan ejecutar. Para hacerlo, se tienen en cuenta factores como el requerimiento de recursos del pod, restricciones de hardware y software, carga de trabajo de los nodes worker para no sobrecargarlos, etc.

El **kubelet** es un componente que actúa como agente, ejecutando en cada node worker y se comunica con el node master. Este componente recibe la definición de pod y ejecuta el contenedor asociado a un pod. También asegura que los contenedores estén ejecutados en el pod, controlando así la salud de las aplicaciones desplegadas en los pods.

El **kube-proxy** es un proxy de red que se ejecuta en cada node worker, y su función es mantener las reglas de la red.

El **Container Runtime** es el software responsable de ejecutar los contenedores. Kubernetes soporta algunos, como ahora: Docker, containerd y CRI-O entre otros.

Para acabar tenemos **kubectl**, que es la CLI de Kubernetes. Las comandas de kubectl son transformadas en peticiones REST que se envían al componente de API Server.

Office

Para poder organizar y documentar este proyecto se ha utilizado la suite de Microsoft Office. De todas las aplicaciones suite, se han utilizado las siguientes:

- Outlook: para poder comunicarnos mediante correos electrónicos y poder organizar reuniones entre nosotros y con el tutor del proyecto.
- Teams: esta aplicación de mensajería nos ha permitido intercambiar mensajes y documentos en un chat. Además, se ha utilizado para realizar videoconferencias de las reuniones que se han organizado mediante Outlook.
- Word: con este programa hemos escrito documentos de texto como esta memoria del proyecto.
- Powerpoint: esta aplicación nos ha permitido preparar las diapositivas para la presentación de este proyecto.
- OneDrive: es un servicio de hosting que nos permite guardar ficheros en la nube y poderlos compartir con otros usuarios.

Diagrama de contexto

En la siguiente imagen se muestra el diagrama de contexto de la solución:

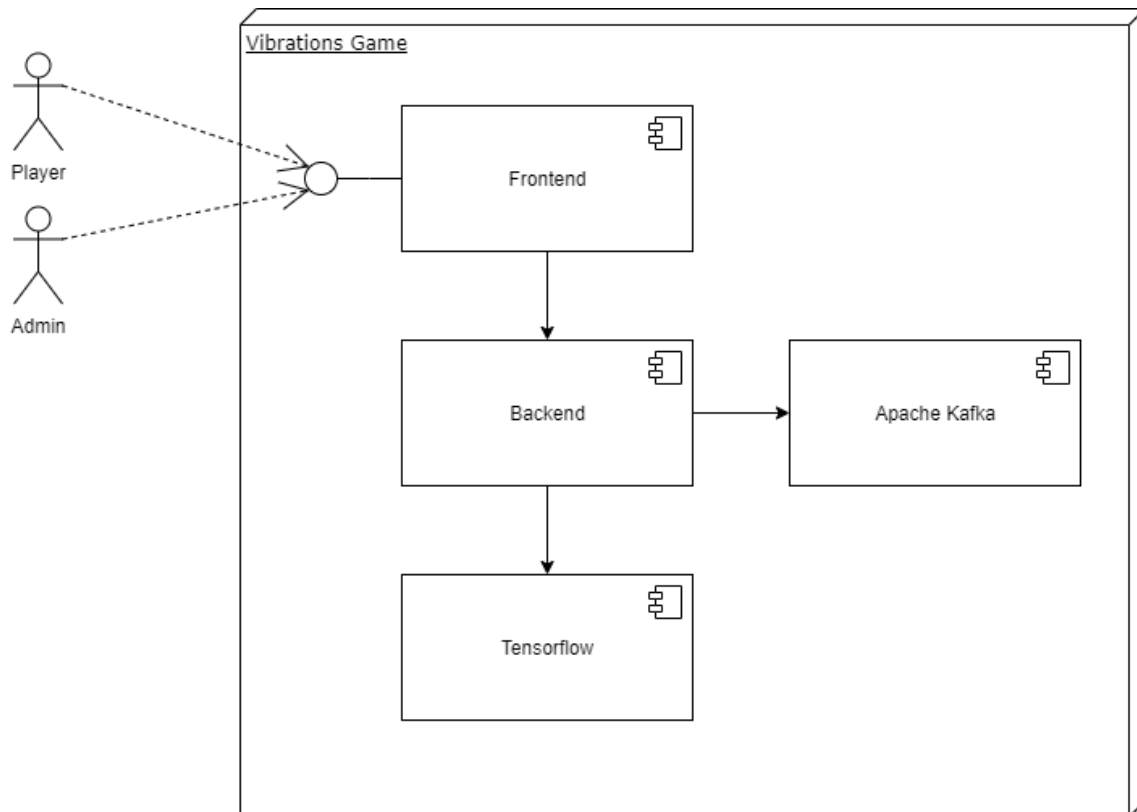


Fig. 4: Diagrama de contexto de la solución

En este diagrama podemos ver los diferentes elementos que conforman la solución.

Podemos ver dos tipos de usuario:

- Admin: será el que ejecutará las tareas administrativas.
- Player: este usuario será el que realizará las operativas relacionadas con la ejecución de una partida.

Estos dos usuarios utilizarán el componente frontend para realizar sus operaciones respectivas.

El componente del backend es el que recibe las peticiones enviadas por el frontend. También se comunica con el componente de Kafka y de Tensorflow para poder completar la operativa requerida por la aplicación.

Apache Kafka será el componente que guardará y procesará parte de la información recibida por el frontend.

Por último, teniendo el componente Tensorflow que recibe las peticiones del backend y, utilizando un modelo de datos de Machine Learning, devolver la respuesta del backend.

5. Objetivos

Generales

Los objetivos generales del proyecto se pueden definir a continuación:

- Utilizar Apache Kafka para conseguir datos en tiempo real.
- Utilizar Tensorflow para conseguir detectar patrones.
- Tener un frontend operativo en el cual los usuarios puedan interactuar.
- Disponer de un backend que pueda trabajar con el frontend y comunicarse con Apache Kafka y la aplicación de Tensorflow.
- Desplegar la aplicación en un entorno de Kubernetes.

Individuales

Los objetivos individuales del proyecto se pueden definir de la siguiente manera:

- Entender correctamente las necesidades de la aplicación de esta manera poder ofrecer aquello que se espera del aplicativo.
- Ser capaz de afrontar todas aquellas carencias de conocimiento y enfrentarse a todos los desafíos que vayan apareciendo a lo largo del proyecto.
- Tener una buena comunicación con el resto del equipo, porque es muy importante sincronizarnos, ya que existen una comunicación directa en los servicios desarrollados por cada miembro del equipo, así como una correcta integración.
- Realizar un diseño atractivo para el usuario final y que este se sienta cómodo a la hora de utilizar la aplicación.
- Definir y concretar correctamente la implementación a realizar para evitar al máximo la refactorización de código.
- Aprender buenas prácticas del lenguaje de este modo optimizar el código para obtener un producto de calidad.
- Planificar cumpliendo todos los deadlines planteados en la planificación del proyecto y haber realizado una buena estimación de las diferentes tareas a realizar.
- Conseguir configurar y montar todo el ecosistema en un entorno de producción real en algún servidor cloud de forma pública y accesible al mundo.

6. Frontend - Interfaz grafica

Introducción

La parte del frontend de la aplicación Vibrations Game es la parte visual de la solución encargada de interactuar con el usuario/jugador a través de un dispositivo móvil de manera responsiva y recogiendo los sensores de movimiento del dispositivo del usuario, está desarrollada con el framework de Javascript React y utilizando componentes visuales de Material-UI.

Ya que es la parte encargada de mostrar toda la funcionalidad del backend, esta comunicación se realiza principalmente mediante la utilización de sockets y también mediante unos endpoints (API REST) al servidor Java. Aquí podemos observar un diagrama a modo de resumen de la estructura de nuestro frontal que más adelante se explicara detalladamente:

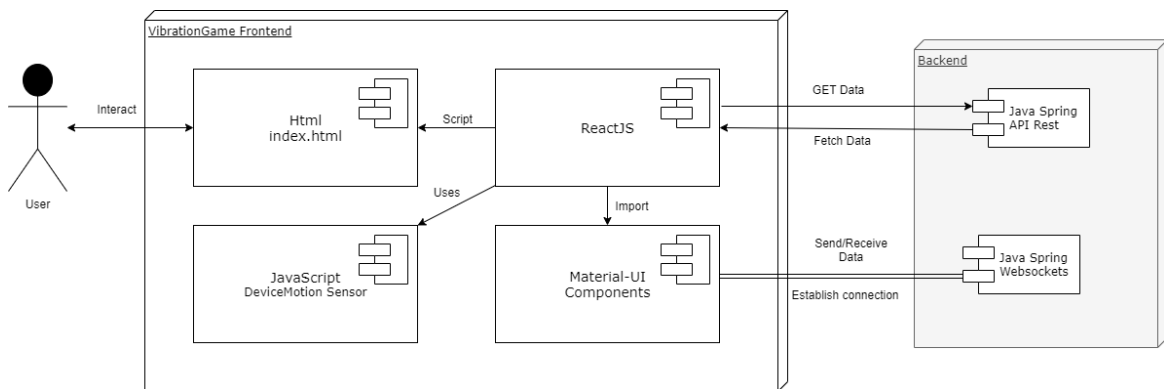


Fig. 5: Diagrama de componentes del frontend

Tecnologías utilizadas

Para poder desarrollar la parte de frontend, se han utilizado herramientas y aplicaciones específicas, distintas de las que se han usado en común por el resto de módulos, que se explicarán a continuación.



VSCoDe IDEA es un IDE que ha desarrollado la empresa de Microsoft y que permite desarrollar aplicaciones en diferentes lenguajes de una manera muy rápida y portable. En este caso lo hemos utilizado tanto para HTML, CSS, React y Docker. Esto es posible, ya que dispone de extensiones fácilmente descargables a través de su Marketplace, este incluye des de lenguajes, visualizadores de ficheros, formateadores de texto y mucho más. Remarcar que hemos utilizado Snippets para el autocompletado y Lottie Viewer para previsualizar nuestras animaciones.



ReactJS es un framework de Javascript de código abierto que nos facilita el desarrollo de interfaces de usuario de tal forma que el desarrollador se puede centrar más en el desarrollo de una sola página. Nos permite utilizar componentes reusables para nuestra interfaz.



JavaScript como principal lenguaje de toda nuestra aplicación, ya que React es un framework de este y también lo utilizamos para obtener los sensores del dispositivo móvil a través de un evento de JavaScript nativo (DeviceMotionEvent).



Lenguaje de marcas en su última versión utilizado como standard para referenciar las páginas web dentro de la World Wide Web. En nuestro caso lo utilizamos únicamente como punto de entrada de la aplicación en el cual le pasamos el script de React con todo el contenido además de definir el contenido del header como el title y metadatos.



Hojas de estilo en cascada también en su última versión, utilizadas para definir los estilos de los componentes utilizados, nos permite separar el contenido de la representación visual de estos.



Node Package Manager es un gestor de paquetes desarrollado en JavaScript. Nos permite obtener cualquier librería con tan solo una línea de código, lo cual nos permitirá agregar dependencias de forma simple, distribuir paquetes y administrar eficazmente tanto los módulos como el proyecto a desarrollar en general.



MATERIAL-UI

Material-UI para un desarrollo rápido y sencillo de componentes con una base de diseño de carácter profesional. Dispone de una API muy fácil e intuitiva con todo el listado de componentes que ofrece, además de iconos y temas.



LottieFiles

LottieFiles es un formato de archivo de animación de código abierto que es extremadamente ligero, de alta calidad, interactivo y se puede manipular en tiempo de ejecución. Eliminando la sensación de página estática a través de unos JSON rápidos de cargar y manipular.

OpenSSL
Cryptography and SSL/TLS Toolkit

OpenSSL para la creación de certificados obteniendo una capa de seguridad adicional necesaria para obtener nuestro dominio HTTPS para poder solicitar al usuario acceso a la orientación de su dispositivo móvil.



Jest es un marco de pruebas de JavaScript mantenido por Facebook, enfocado en la simplicidad y el soporte al testing. Facilitando y mejoran la experiencia con la realización del test del proyecto.

6.1. Requerimientos

Antes de comenzar a realizar la aplicación nos reunimos para definir los requerimientos que queríamos que cumpliera nuestra aplicación, estos son una mezcla entre los requerimientos de la propuesta de proyecto más aquellos que queríamos añadir para aportar un valor adicional.

Requerimientos propuesta

Los requerimientos principales propuestos para la realización del trabajo son los siguientes:

- Aplicación web.
- Orientada a dispositivos móvil.
- Recolección de sensores del dispositivo.
- Utilización de websockets.
- Presentar patrones al usuario.
- Mostrar un ranking al finalizar el juego (Top 5).

Tienen un carácter amplio y no muy concreto lo cual nos permite tener bastante flexibilidad en la toma de decisiones, así como tampoco se especifica una tecnología concreta para el frontal de nuestro aplicativo.

Requerimientos funcionales

Los requisitos funcionales principales que decidimos de forma general y que queríamos cumplir son los siguientes:

- El administrador y el jugador pueda acceder al servidor web desde cualquier dispositivo móvil ya sea móvil o Tablet.
- La aplicación sea en Inglés, tanto animaciones como textos literales.
- El jugador pueda unirse a partidas ya creadas no empezadas.
- El jugador pueda visualizar los patrones que tiene que realizar en la pantalla.
- El dispositivo pueda capturar todos los datos de los sensores de movimiento del jugador.
- Se muestre una puntuación final con el listado de jugadores y sus correspondientes puntuaciones.
- El administrador pueda crear partidas definiendo todas las condiciones de una partida.
- El administrador pueda eliminar partidas.
- El administrador pueda visualizar el estado y ranking de la partida en curso.

Requerimientos no funcionales

A continuación, los requerimientos no funcionales definidos al comenzamiento del proyecto:

- El sistema debe ser capaz de recoger los datos de los sensores en unos tiempos entre 10-20ms.
- El sistema debe disponer de componentes visuales reutilizables.
- El sistema debe proporcionar mensajes de error que sean informativos y orientados al usuario final.
- La interfaz de usuario debe ser userfriendly y minimalista con un carácter de videojuego para mejorar la experiencia del jugador.
- Utilizar animaciones para no tener un carácter tan estático y parecer más dinámico.
- Todas las comunicaciones con los sensores de orientación del dispositivo tienen que estar encriptadas mediante un protocolo HTTPS.
- Los ficheros deberán estar optimizados para minimizar el espacio del servidor.

6.2. Especificaciones

En este apartado definiremos los casos de uso que pueden realizar nuestros actores principales Player y Administrador y después mostraremos en más detalles las diferentes pantallas que aplican cada uno de ellos.

Casos de uso del usuario Player

El usuario Player es aquel que accede a nuestra aplicación desde su móvil/tableta para ejecutar un juego.

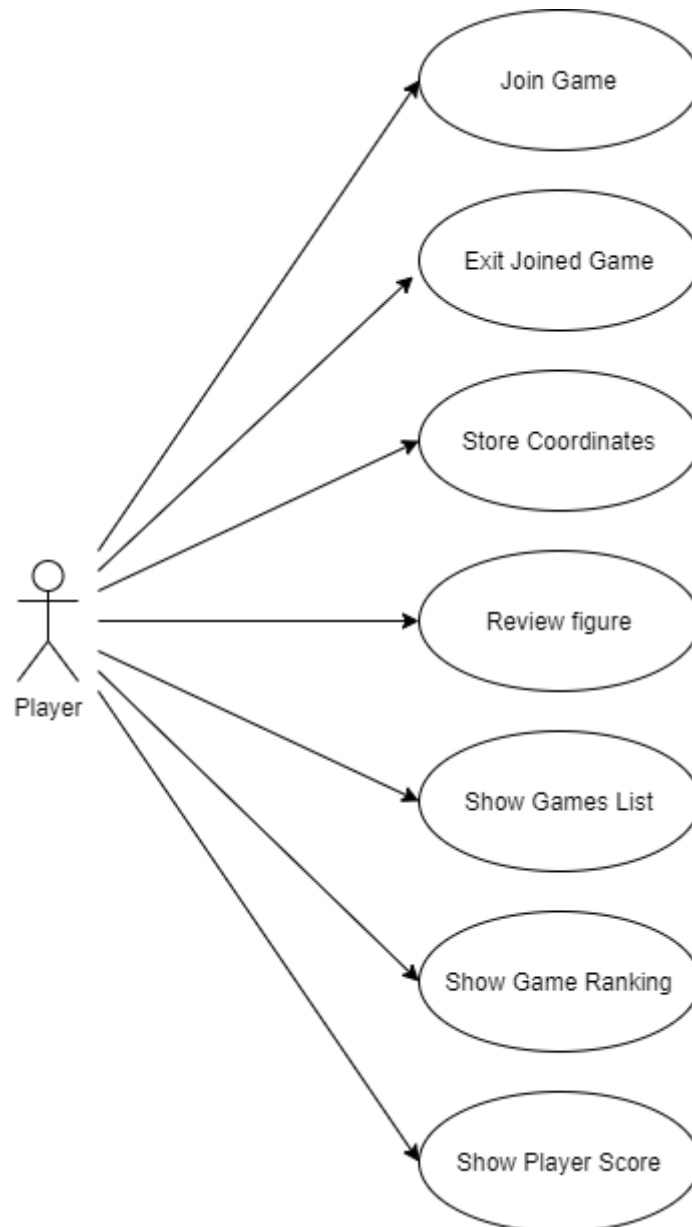


Fig. 5: Diagrama de los casos de uso del usuario Player

Casos de uso del usuario Administrador

El usuario Administrador es aquel que realiza operaciones administrativas a través de su portal exclusivo, orientado a la configuración y visualización de los datos generales sobre una partida.

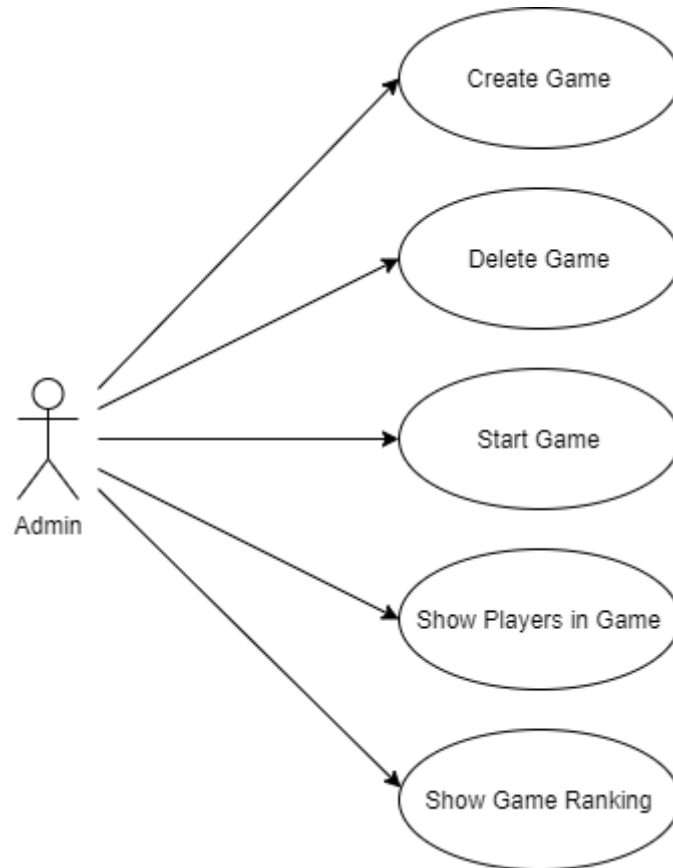


Fig. 6: Diagrama de los casos de uso del usuario Administrador

6.3. Diseño

Diseño de la aplicación UI/UX

La idea principal del diseño visual de la aplicación está basada en un concepto minimalista, en el cual se prima tener aquellos componentes básicos sin sobrecarga de elementos visuales. También otro punto fuerte es la idea de hacerla lo más user friendly posible, ya que se trata de un juego interactivo, para conseguir dicho objetivo hemos utilizado emojis y LottieFiles.

Gracias a las animaciones que nos proporciona LottieFiles hace que nuestra página siempre esté en constante movimiento y no de la sensación de una página estática. Estas tienen un peso mínimo lo cual no hace que la página no tarde mucho en cargar todos los assets.

Siempre orientando el diseño hacia nuestro target principal que es el usuario móvil, teniendo muy en cuenta el concepto responsivo en todos los elementos considerando los diferentes tamaños de dispositivos móviles y tableta. Para realizar estas comprobaciones nos hemos ayudado de la herramienta de DevTools de Chrome que dispone de una gran cantidad de dispositivos y sus tamaños reales, esta también nos permite simular los sensores del dispositivo.

Con el fin de plantear los diferentes prototipos y mockups en las fases más primitivas del desarrollo nos hemos reforzado con la ayuda de herramientas como App-Moqups o Draw.IO y también hemos utilizado diseños a papel para las primeras aproximaciones.

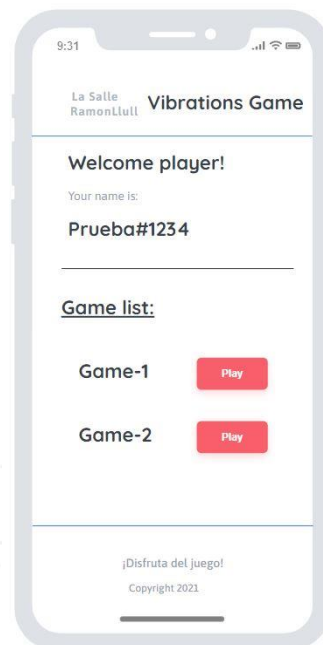
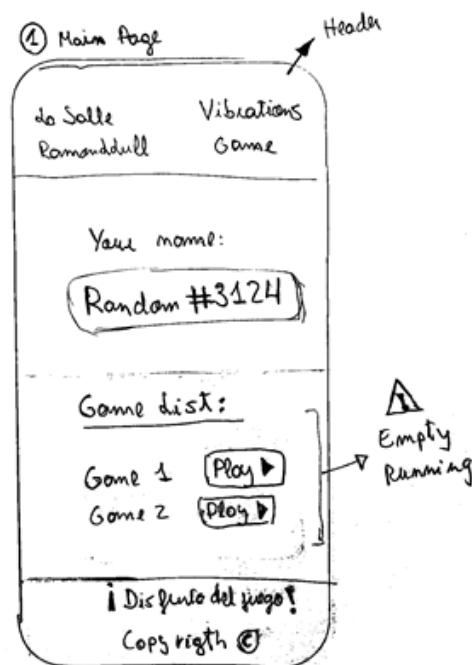


Fig. 7: Primera aproximación de la página principal papel (izquierda)

Fig. 8: Wireframe de la página principal realizado con la aplicación Moqups (derecha)

Responsivo

Material-UI

La utilización de la librería de Material-UI como soporte para la creación de componentes supone un gran ahorro de esfuerzo de trabajo. Material-UI nos permite de una manera rápida y sencilla creación de páginas web con resultado de alta calidad y un producto formal.

Esta librería también nos permite solucionar fácilmente la problemática de nuestra web responsiva y orientada a dispositivos móviles, ya que prácticamente todos los componentes cuentan con un Grid, que sería el equivalente a un flexbox en CSS.

Este Grid se encarga de reajustar según la pantalla del dispositivo y conseguir mantener una apariencia adaptativa sin importar el tamaño y la orientación de la pantalla, lo que garantiza coherencia entre los diseños.

El Grid se divide en dos layouts: containers e ítems.

Los anchos de los elementos se establecen en porcentajes, por lo que siempre son fluidos y tienen un tamaño en relación con su elemento principal.

Los elementos tienen relleno para crear el espacio entre elementos individuales.

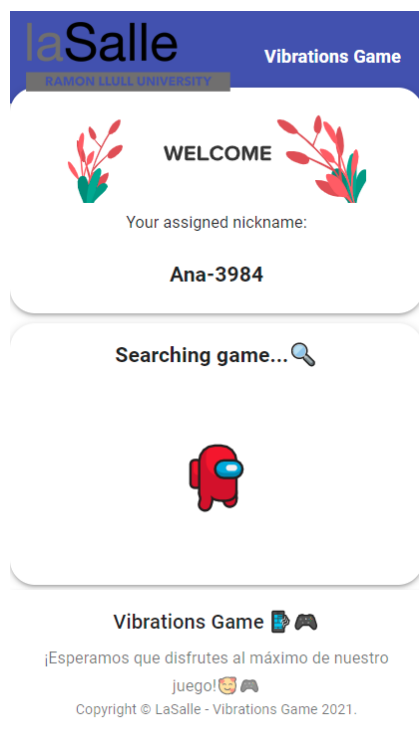


Fig. 9 Izquierda: Pantalla principal iPhone 6/7/8 Plus

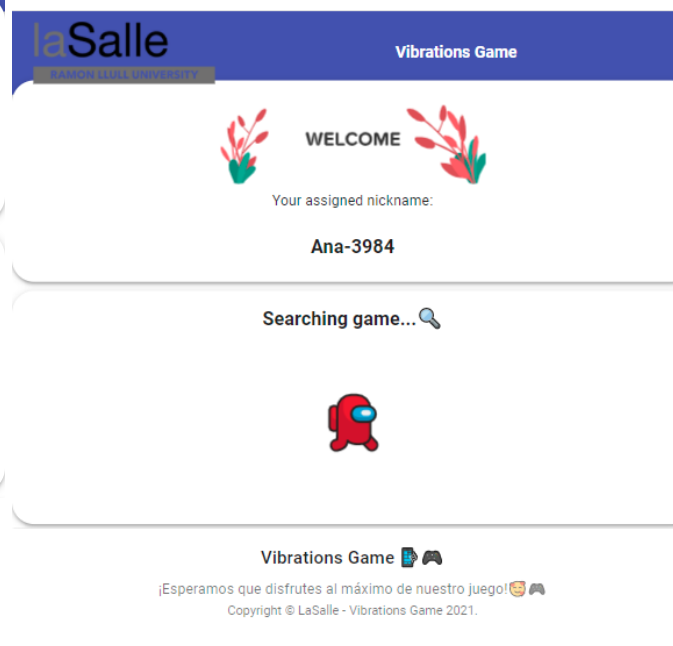


Fig. 10 Izquierda: Pantalla principal iPad

LottieFiles

LottieFiles también creemos que ha sido uno de los mayores aciertos a la hora de realizar el frontal de nuestra aplicación, permitiéndonos cambiar de una página web estática inanimada a una que está en constante movimiento y más amigable con el usuario.

Normalmente este tipo de decisiones suponen un alto incremento del tiempo de descarga del contenido de la página web, empeorando la experiencia de usuario, pero LottieFiles es de las mejores animaciones web que existe en la web con respecto a la optimización, ya que estas son creadas a través de un fichero JSON haciéndolo hasta un 600% más pequeño que un GIF, además de una fácil portabilidad también nos permite modificar las diferentes capas de nuestra animación de una manera muy fácil



Fig. 11: Ejemplo de LottieFile utilizado en la pantalla del ranking

Cuenta con un MarketPlace lleno de animaciones creadas por diseñadores profesionales de todo el mundo, facilitando las labores de diseño si no se dispone de un diseñador particular, como es el caso.

Construcción del proyecto sobre React

En este apartado vamos a justificar el uso de React como el framework base de nuestra aplicación frontal.

Para ello no nos basaremos en comparaciones con otros frameworks, puesto que existen varios como Angular o Vue con características similares entre los cuales sería complicado justificar de manera absoluta el uso de un framework frente a otro en un proyecto de pequeñas dimensiones como es Vibrations Game.

Nos centraremos en mostrar las ventajas que ofrece React para un proyecto como el nuestro y el porqué es adecuado su uso en nuestro caso.

¿Que nos aporta React?

React nos aporta una manera más ordenada y con menos código de desarrollar que si usásemos JavaScript puro, ya que se trata de un framework de este, aportándonos mayor abstracción. Además de esto permite que las vistas se asocien con los datos, de modo que, si cambian los datos, también cambian las vistas.

En el caso de Vibrations Game se tenía como requisito de diseño la creación de componentes reutilizables para nuestra aplicación, esto lo podemos hacer fácilmente con funciones dentro de React.

Pero el punto más importante que nos aporta React son sus clases, ya que nos ofrece un state en el que almacenar variables, así como el estado actual de la página y el apartado de render en el que devolvemos el componente visual. Gracias al ciclo de React podemos cargar los datos antes de devolver las vistas al usuario con la función `componentDidMount` haciendo muy fácilmente un `fetch` (HTTP Request) de la API o de los websockets.

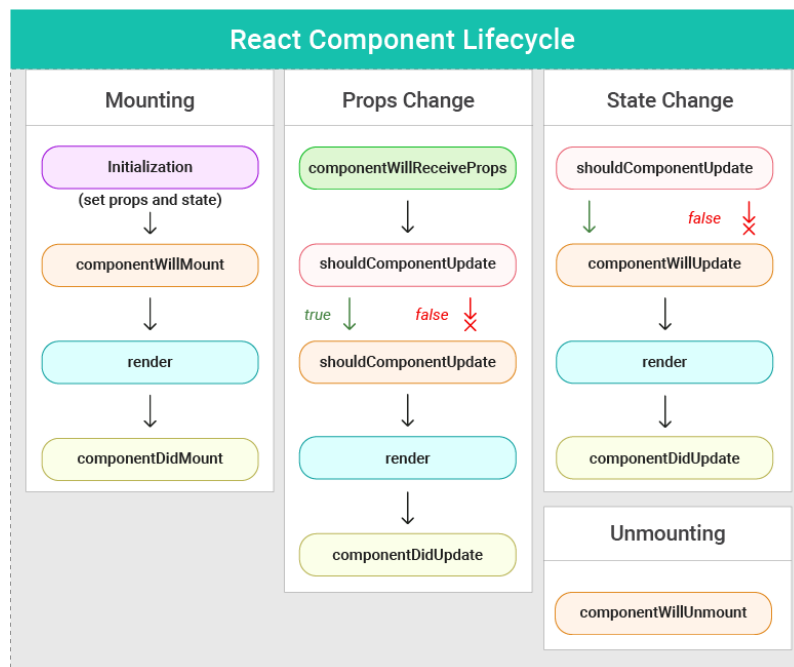


Fig. 12: Ciclo de vida de los componentes de React

6.4. Desarrollo

Arquitectura

En cuanto a la arquitectura interna de nuestro frontend, podemos decir que hemos separado nuestra aplicación en dos grandes módulos:

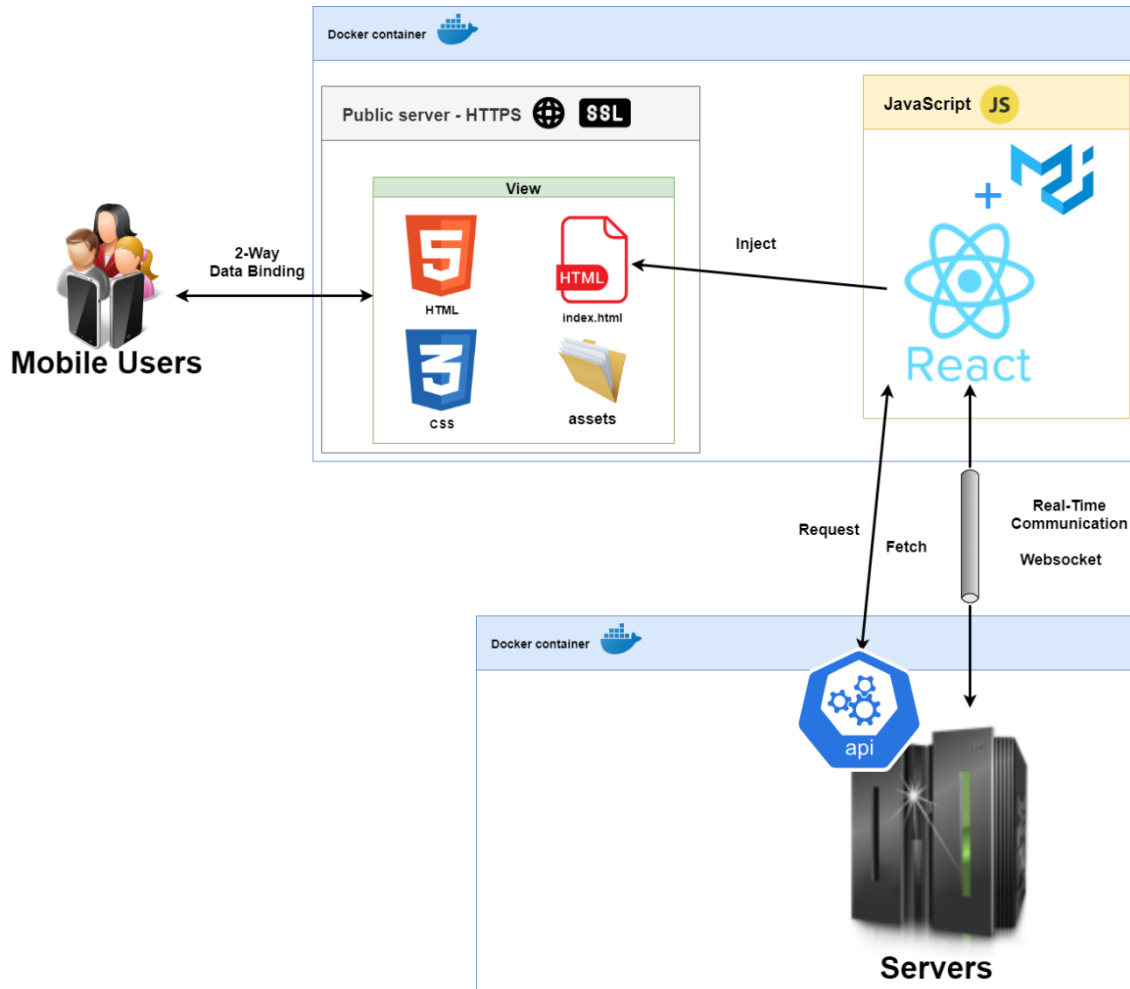


Fig. 13: Arquitectura del frontal, comunicación con el servidor y usuarios.

- **View o Servidor Público:** Se trata del punto de acceso desde la World Wide Web para los usuarios finales. Dicho punto de entrada es accesible mediante nuestro fichero `index.html`. Dicho módulo cuenta con todos los ficheros que se tendrán que alojar en el servidor hosting, la aplicación compilada. También contiene la carpeta `assets` con todos los ficheros que la aplicación necesita, principalmente imágenes.

```
index.html
public > index.html > html > head > script
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="utf-8" />
6   <script src="https://cdn.jsdelivr.net/npm/sockjs-client@1/dist/sockjs.min.js"></script>
7   <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Roboto:300,400,500,700&display=swap" />
8   <meta name="viewport" content="width=device-width, initial-scale=1" />
9   <meta name="theme-color" content="#000000" />
10  <meta name="description" content="Web site created using create-react-app" />
11  <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
12
13  <title>Vibrations Game - LaSalle</title>
14 </head>
15
16 <body>
17
18   <noscript>
19     <h2 style="color: #ff0000">Seems your browser doesn't support Javascript! Websocket relies on Javascript being
20     enabled. Please enable
21     Javascript and reload this page!</h2>
22   </noscript>
23
24   <div id="root"></div>
25
26 </body>
27 </html>
```

Fig. 14: Index.html con el apartado root marcado donde se inyectará React

- JavaScript o Núcleo de la aplicación: todo nuestro desarrollo de React utilizando Material-UI y JavaScript. El fichero index.js será el encargado de inyectar al DOM de la aplicación (index.html del módulo view) toda nuestra App.js. Este módulo contiene todos los componentes utilizados por el frontal, así como todas las vistas de la aplicación, apoyándose en Material-UI.

```
JS index.js
src > JS index.js
1 import React from "react";
2 import ReactDOM from "react-dom";
3 import reportWebVitals from "./reportWebVitals";
4 import "./index.css";
5 import App from "./App";
6
7 ReactDOM.render(<App />, document.getElementById("root"));
```

Fig. 15: Index.js inyectando a la aplicación React (App.js) a nuestro root html.

Finalmente comentar que nuestra aplicación React, internamente se comunica constantemente con la parte del servidor backend mediante API REST y websockets, en el siguiente apartado Interacción con el servidor (en el apartado 6.4. Desarrollo) se explica más en detalle la comunicación entre estos.

Estructura de directorios

A continuación, se muestra la estructura de carpetas utilizada y los archivos más importantes del proyecto con sus correspondientes explicaciones:

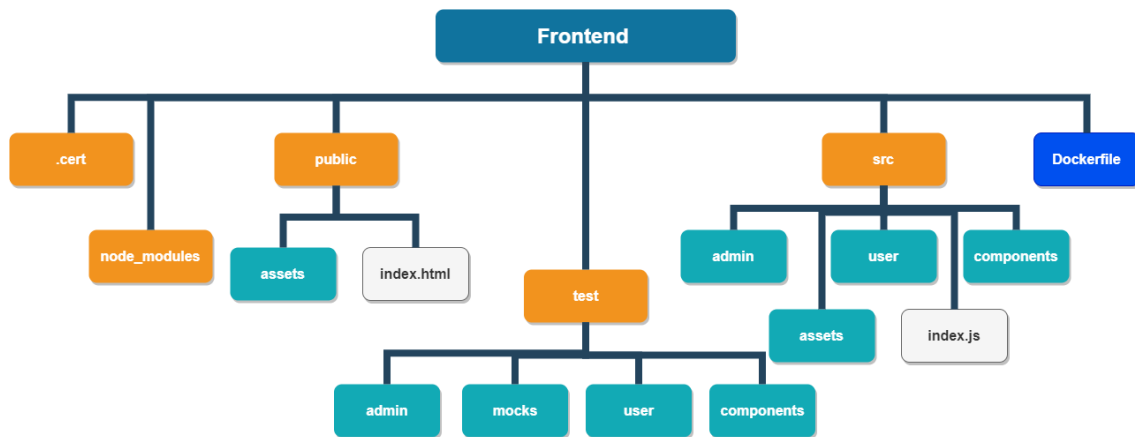


Fig. 16: Estructura de carpetas frontend Vibrations Game

- **.cert** : carpeta que contiene el certificado y la clave (generado con OpenSSL), ya que nuestra aplicación accede a los sensores del dispositivo del usuario y es requerida una conexión https para poder obtenerlos.
- **node_modules**: carpeta que contiene todos los módulos externos de los que depende el proyecto (todas las instalaciones de npm descargadas).
- **public**: contiene todo el contenido público del servidor para ser publicado en un servidor, html principal, assets y favicon.
- **src**: toda la carpeta de origen. Dividido en:
 - **admin**: contener todos los diseños relacionados con la página de administración ("/admin").
 - **user**: contienen todos los diseños relacionados con la página del usuario, la aplicación principal ("/").
 - **components**: contiene todos los elementos mínimos de la aplicación como: encabezado, pie de página, títulos ... etc. Usado por el usuario y el administrador. Intentando reutilizarlos al máximo.
 - **assets**: todos los archivos requeridos de la aplicación, en este caso solo utilizamos imágenes con extensiones: ".jpg", ".png" y ".json".
 - **index.js**: responsable de inyectar toda nuestra aplicación React ("App") en el html.
- **Test**: todos los test de la aplicación. Divido en:
 - **mocks**: Encargados de mockear todos los canvas estilos y ficheros, como por ejemplo imágenes o lottiefiles.
 - **admin**: contiene todos los test relacionados con la página administración.
 - **user**: contiene todos los test relacionados con las páginas de usuario.
 - **components**: contiene todos los test de los componentes visuales de la aplicación.
- **.Dockerfile**: generar el contenedor Docker a partir de las carpetas del proyecto actual, copiando e instalar todas las dependencias escritas en "package-lock.json".

Sistema de navegación

Utilización de 'react-router-dom' para el redireccionamiento de componentes basándose en la URL, como hemos comentado anteriormente nuestra web esta dividida principalmente en dos servicios:

- Administrador "/admin": Gestión y administración de las diferentes partidas, así como crearlas (número 1 de la Figura 16).
- Página principal jugador "/": Accesible para todos aquellos jugadores que quieren unirse a una partida (número 2 de la Figura 16).



```
<Router>
  <div>
    <header className="App-header">
      <Header />
    </header>
    <Switch>
      1 <Route path="/admin">
        <Admin />
      </Route>
      2 <Route path="/">
        <UserMain
          randomName={this.state.name}
          socket={this.state.socket}
          isJoined={this.state.isJoined}
          isWaiting={this.state.isWaiting}
          actualFigure={this.state.actualFigure}
          figureIndex={this.state.figureIndex}
          isDeleted={this.state.isDeleted}
          isEnd={this.state.isEnd}
          gameId={this.state.gameId}
        />
      </Route>
    </Switch>
    <Paper>
      <Footer />
    </Paper>
  </div>
</Router>
```

Fig. 17: App.js mostrando las diferentes rutas de la aplicación

Como se puede observar se trata de un Switch el cual únicamente se encarga de ir alternando en función de la ruta actual (URL), dentro de una misma ruta el contenido se va actualizando dependiendo en el estado del juego en el que se encuentre el usuario que este es suministrado por el backend a través de los websocket, así como por cada estado se mostrara una pantalla diferente.

Podríamos decir que la aplicación está basada en dos rutas principales, pero dentro de cada una de ellas existen diferentes estados.

Interacción con el servidor

Websockets

El usuario ira interactuando con la aplicación e ira pasando por los siguientes estados y podrá realizar diferentes acciones dependiendo del estado en el que se encuentra, así

como actualizar la pantalla actual en función de estos. Comentar que esta comunicación de estados y acciones está directamente conectada con el websocket de la parte del servidor, lo hace con la siguiente línea de comando:

```
var socket = new WebSocket("wss://192.168.1.53:8090/vibration/websocket");
```

Fig. 18: Línea de código para establecer conexión websocket

El estado actual de la aplicación se coge de los mensajes que recibe este socket en App.js y este se va pasando por los diferentes componentes de la aplicación utilizando la herencia.

Usuario:

- **CONNECTED:** Cuando se consigue establecer la conexión websocket con el backend, automáticamente recibe un nombre de usuario aleatorio y lo muestra en el frontal. Antes de que se establezca aparece el LottiFile de cargando.
 - **JOIN:** el usuario puede seleccionar una partida ya creada en la lista de la página principal y clicar el botón de JOIN para unirse. Si no existe ninguna partida se mostrará “Searching game...” en la aplicación del usuario.
- **WAITING:** Es la pantalla de carga en la que se encuentra el usuario antes de que el administrador inicie la partida. También aparecen animaciones para hacer la espera un poco más amena.
- **PLAYING:** Este sería el estado central de la aplicación, automáticamente al entrar en una partida al usuario se le presentará el primer patrón de movimiento obtenido por el servidor.

El usuario realiza el movimiento y se lanza una petición al servidor para que lo compruebe REVIEW, acto seguido si el usuario acierta pasara a la siguiente pantalla, incrementado el número de la página actual y mostrando el siguiente patrón también obtenido por el backend.

También existe una pantalla asociada a la finalización de todos los patrones, la cual muestra un mensaje al jugador informándole de su estado.
- **END:** automáticamente el usuario es redirigido a una página donde podrá observar una tabla con todos los jugadores de la partida y las puntuaciones obtenidas por cada uno de ellos.

Administrador:

- **CREATE:** El administrador coloca todas las condiciones de una partida en los inputs y presiona el botón de CREATE. Estos campos están validados para que cumplan con el formato correcto.
- **START:** Botón que lanza la opción al servidor de iniciar la partida actual.
- **DELETE:** Botón que lanza la opción al servidor de borrar la partida actual.

API REST

React nos permite de una forma muy sencilla extraer información de las APIs, cuenta con una función `fetch` la cual recibe una URL y esta función nos permite recoger el contenido de la petición GET al servidor.

```
fetch("https://192.168.1.53:8090/game/list")
  .then((response) => response.json())
  .then((gameList) => this.setState({ gameList: gameList.gamelist }));
```

Fig. 19: Ejemplo de obtención de datos del backend

En el transcurso del juego se realizan diferentes llamadas a la API del servidor, estos endpoints son:

- **"/game/list/":** Encargado de devolver todas las partidas. Nos devuelve el identificador de partida (único), el nombre de esta y si se encuentra en curso.

```
{
  "number": 2,
  "gameList": [
    {
      "id": "4cf590f4-dfd7-4d56-9400-7c831e5f331f",
      "name": "Prueba1",
      "playing": false
    },
    {
      "id": "43bf17c8-74aa-4938-8093-60dccf802f1e",
      "name": "GameRoom1",
      "playing": false
    }
  ]
}
```

Fig. 20: Respuesta de la petición de obtención de partidas.

- **"/game/players" + identificador de partida:** Devuelve listado de jugadores asociados a una partida.

```
{
  "number": 1,
  "playerList": [
    {
      "username": "Jaume-124"
    }
  ]
}
```

Fig. 21: Respuesta de la petición de obtención de partidas.

- **"/ranking/game/" + identificador de partida:** Devuelve listado de jugadores y su puntuación asociados a una partida.

```
{
  "ranking": {
    "Jaume-124": 12
  }
}
```

Fig. 22: Respuesta de la petición de obtención de partidas.

Sistema de orientación en dispositivos móviles

Para el reconocimiento de patrones necesitamos acceder a los datos de aceleración de los dispositivos, para obtener dichos datos lo realizamos con DeviceMotionEvent que extiende de los eventos del DOM. Para recoger estos datos periódicamente le asignamos un listener que detecta los valores de movimiento con un intervalo de 20 ms aproximadamente. También disponemos de la orientación y del giroscopio, pero en este caso únicamente utilizaremos la aceleración “devicemotion”:

```
updateSensors() {  
  
  // Request permission for iOS 13+ devices  
  var deviceMotionEvent = new DeviceMotionEvent();  
  
  if (  
    deviceMotionEvent &&  
    typeof deviceMotionEvent.requestPermission === "function"  
  ) {  
    deviceMotionEvent.requestPermission();  
  }  
  
  if (!this.state.is_running) {  
    window.addEventListener(  
      "devicemotion",  
      (event) => {  
        this.handleOrientation(event);  
      },  
      true  
    );  
    this.setState({ is_running: true });  
  }  
}
```

Fig. 23: Generación de la escucha activa de la aceleración del dispositivo

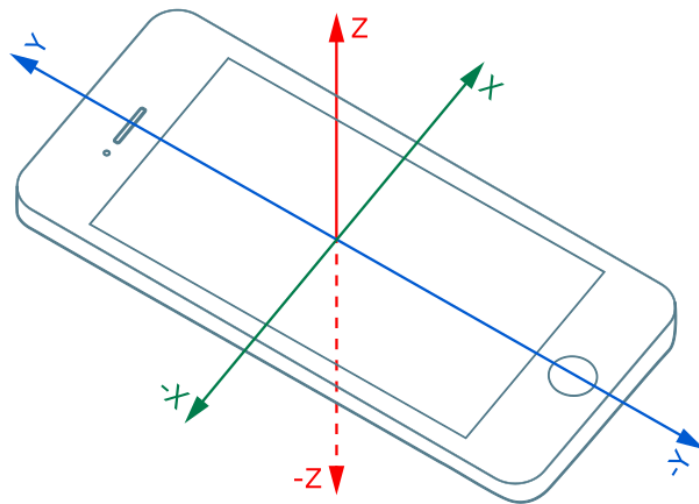


Fig. 24: Sistema de coordenadas acelerómetro

A continuación, se enseña la secuencia de pasos que realiza la captura de los sensores del dispositivo de nuestro usuario:

1. Pedir permiso al usuario (HTTPS). Al usuario le aparece un pop-up con la opción de permitir o cancelar la utilización de sus sensores de orientación.
2. Detectar el inicio del movimiento del dispositivo (Detención de inicio de movimiento).
3. Capturar aceleración en los ejes X, Y, Z, sin incluir la gravedad y con un máximo de 6 decimales.
4. Envío de coordenadas constante vía websockets al servidor backend para su posterior lanzamiento al stream de Apache Kafka.
5. Enviar al servidor solicitud de revisión de figura (Detención de fin de movimiento).
6. Validación de figura mediante comparación. Comparando el valor que nos devuelve el backend de Tensorflow con el patrón que tiene el frontend, ya que este devuelve un String con el movimiento con mayor ponderación.

Observación: La detención de inicio y final de un movimiento se realiza cuando se obtiene una variación grande de movimiento, ya que los sensores son muy sensibles. Esto lo conseguimos almacenando los últimos 10 valores, realizando un sumatorio absoluto entre ellos y haciendo una media, si supera cierto umbral podemos decir que los movimientos del dispositivo son significativos.

A continuación, se muestran los mensajes que le aparecen al jugador en pantalla para realizar los diferentes patrones, con un total de 6 movimientos:

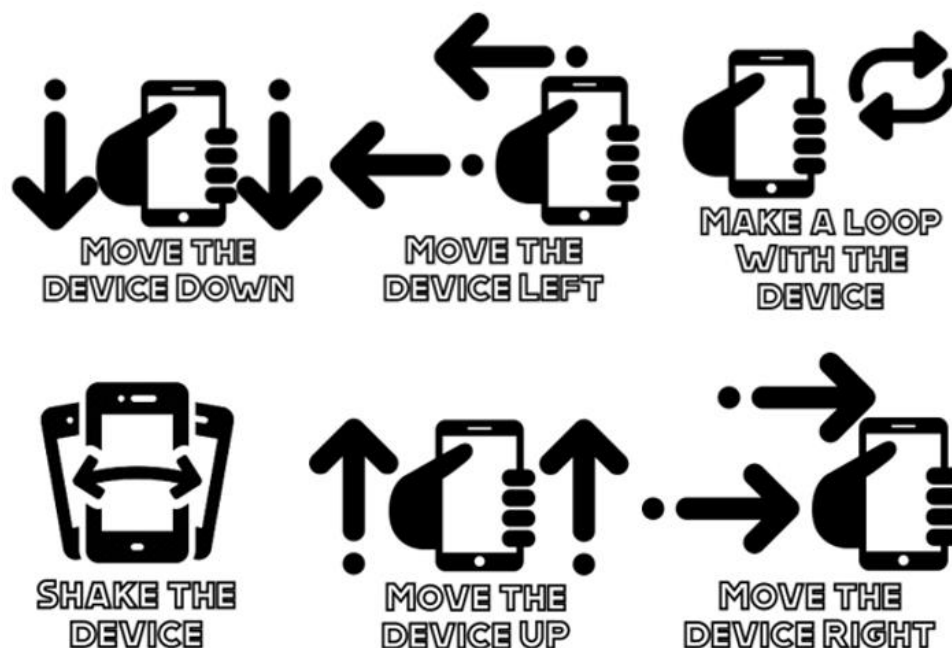


Fig. 25: Imágenes de movimiento mostradas al jugador

Están acompañados de componentes visuales de movimiento (flechas) y texto para un mayor entendimiento del patrón a realizar.

Resultados de la implementación

A continuación, se mostrarán capturas tomadas de Vibrations Game tanto por la parte del jugador como administrativa. Las siguientes capturas de pantallas están en un orden normal de ejecución para mayor entendimiento del flujo del juego.

Pantallas jugador

- **Página de establecimiento de conexión:** Muestra un LottieFile animado para que sea más interactivo mientras el usuario espera a que la aplicación establezca la conexión con el servidor.



Fig. 26: Página de carga: LottieFile animación loading

- **Página principal jugador:** Si el servidor backend se encuentra activo y el frontal ha podido establecer la conexión con el websocket para obtener así su nombre aleatorio aparecerá la pantalla principal. Esta pantalla puede variar dependiendo si se existe alguna partida en curso/creada o todavía no se ha creado ninguna partida.

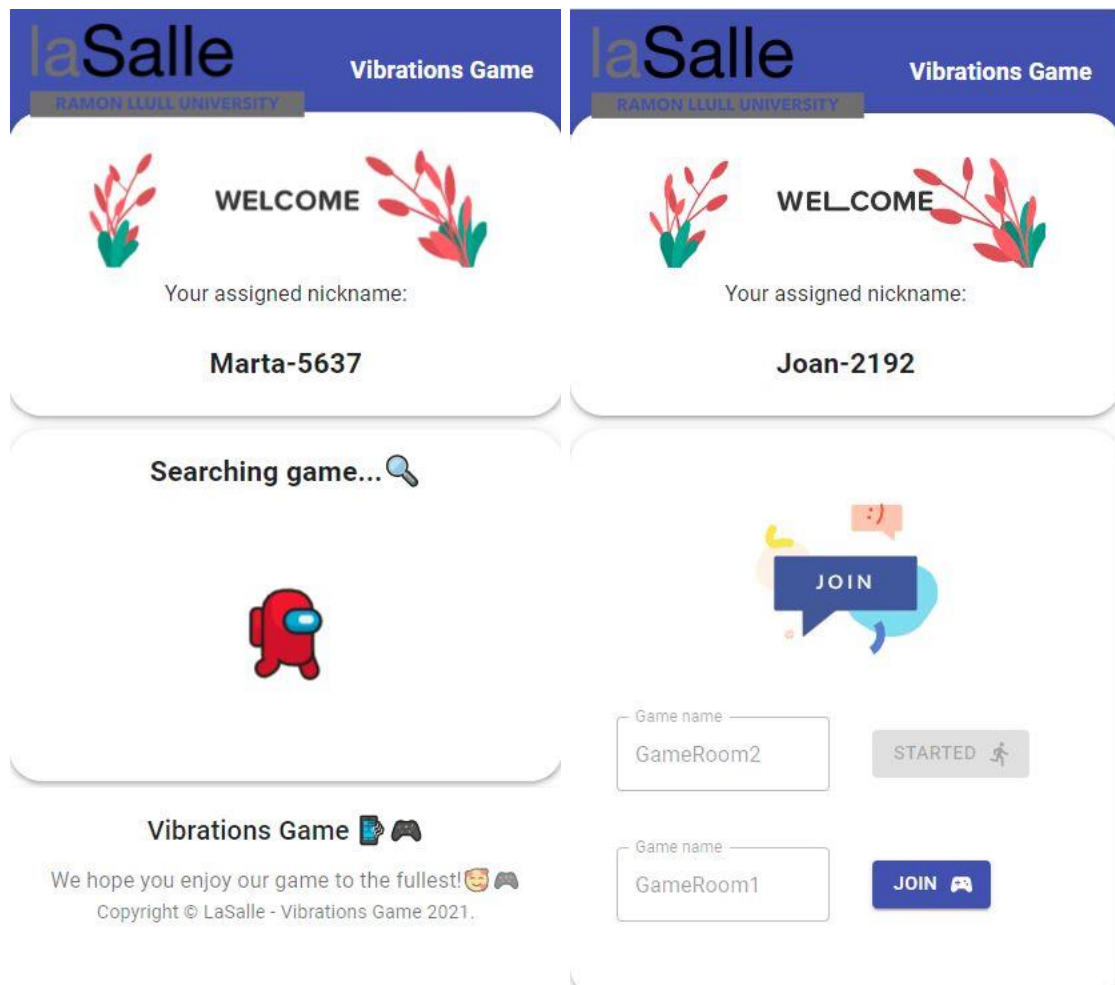


Fig. 27: Página principal buscando partida y con partidas en curso/creadas

- **Página de espera jugador:** Página en la que se encuentra el jugador una vez ha pulsado el botón de unirse a una partida no empezada. Para hacer más amena la espera mientras el administrador inicia la partida tiene una animación de carga.

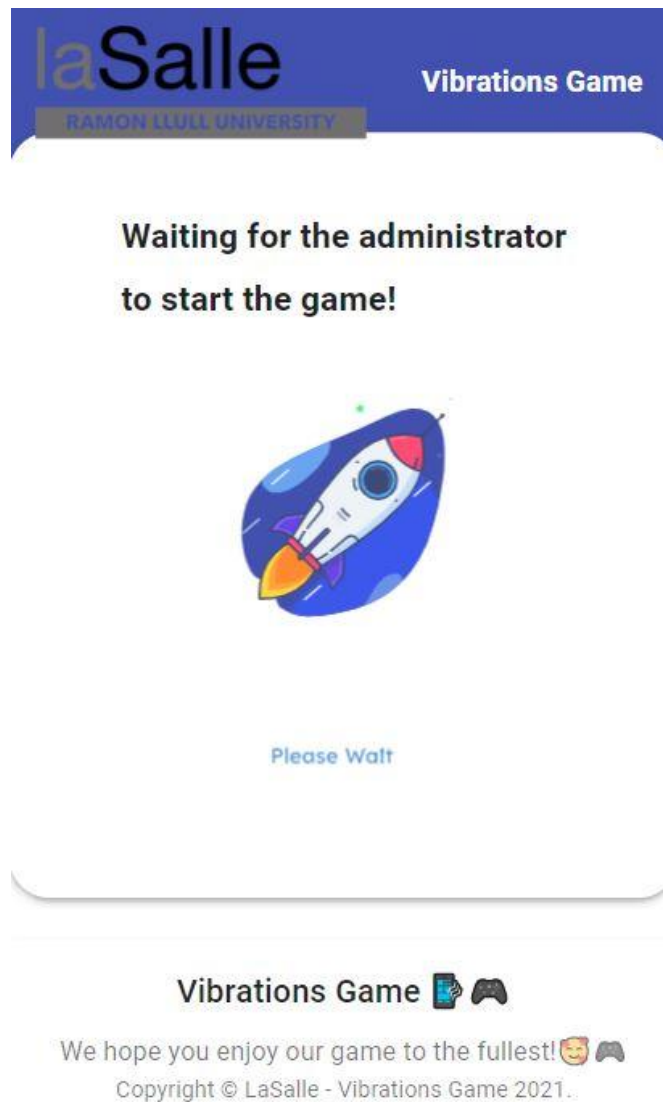


Fig. 28: Página de espera jugador

- **Página de detención de figuras:** Automáticamente una vez el administrador inicia la partida el jugador es redireccionado al juego mostrándole la primera figura, una vez este comience a realizar movimientos con el móvil considerables (tolerancia de movimiento) empezarán a grabarse y enviar directamente al servidor, una vez se detecte que el móvil vuelve a estar en un rango estable (finalización de figura) se lanza una petición al servidor para que este revise si la figura es correcta.

Si la revisión marca que la figura no es correcta el usuario deberá volver a realizar dicho movimiento hasta completar el patrón, de lo contrario avanzará al siguiente paso mostrándole otra figura nueva e incrementando el número de figura en el que se encuentra (bolas numeradas en la parte superior).

Una vez llegada a la última figura número 6 si la completa correctamente le mostrara una pantalla la cual le indica que ha finalizado completamente todos los patrones propuestos, pero debe esperar a que el juego acabe para avanzar a la pantalla de resultados final (Fig. 30).

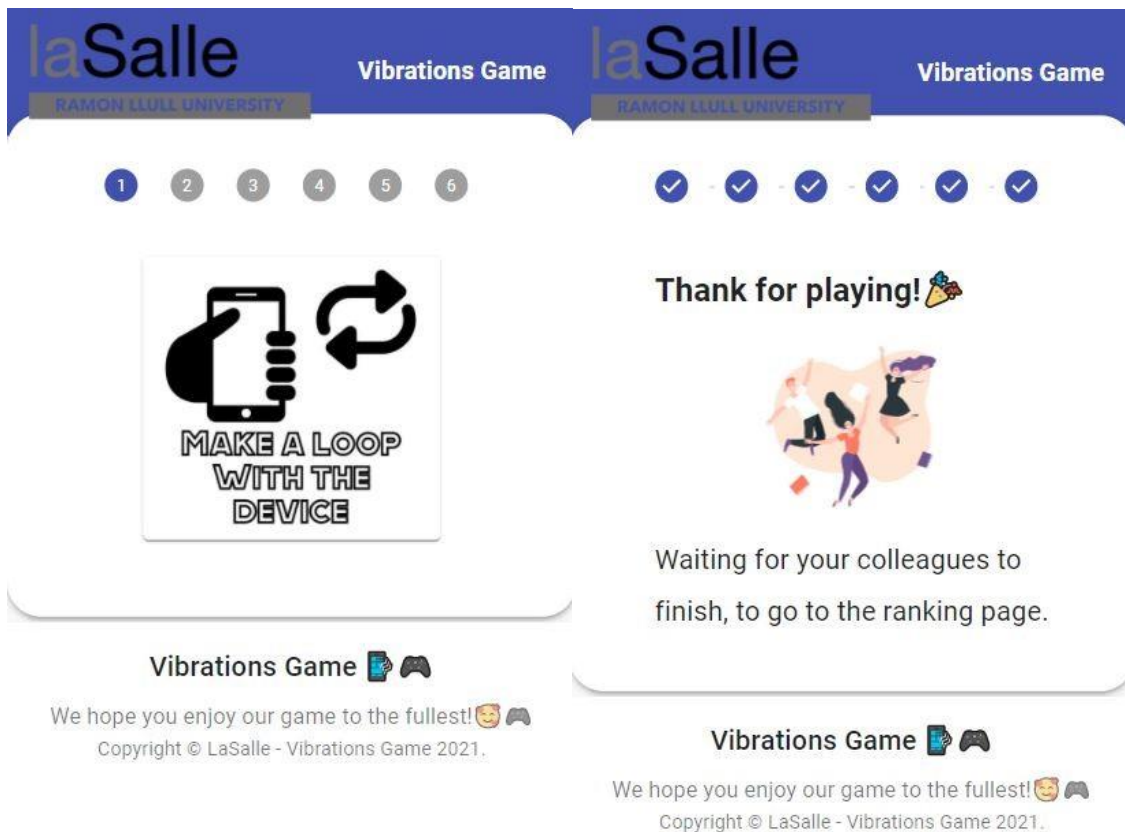
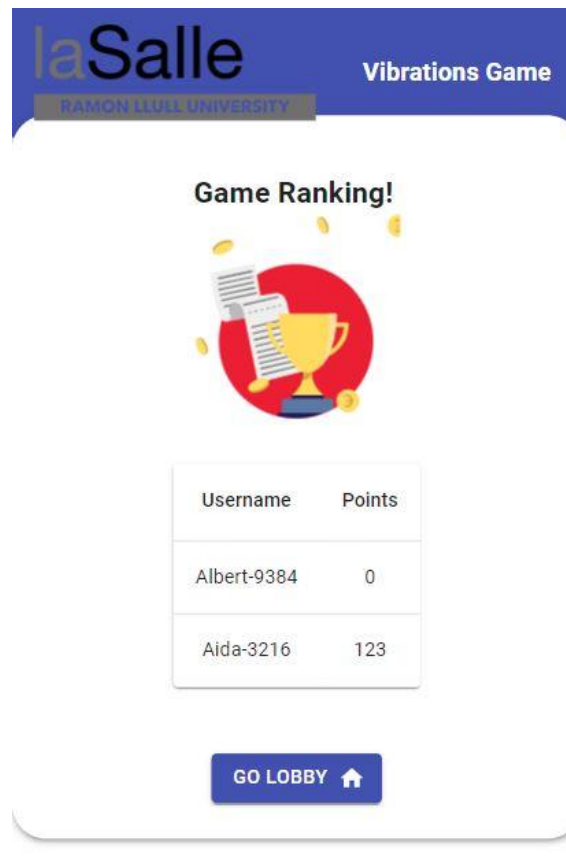


Fig. 29: Detección de figuras

Fig. 30: Esperando a que el juego finalice

- **Página de resultados:** Una vez finalizado el juego por la condición de victoria establecida por el administrador (puntos o tiempo). Automáticamente todos los jugadores serán redireccionados a la pantalla de ranking donde podrán observar sus puntuaciones. Dándoles las opciones de regresar a la pantalla principal del jugador



Vibrations Game 🎮
We hope you enjoy our game to the fullest! 🤖
Copyright © LaSalle - Vibrations Game 2021.

Fig. 31: Página de resultados partida

Pantallas administrador

- **Página de creación de partidas:** Página principal con la que se encuentra el administrador, permitiéndole crear una partida. Una partida se puede definir por un nombre de partida, número máximo de jugadores, condición de victoria (tiempo o puntos) y el valor de dicha condición de victoria.

laSalle

RAMON LLULL UNIVERSITY

Vibrations Game

Administrator Page

Create game:

Match name *

GameRoom1

Max
Players

5 ▾

Victory
Conditions

Time ▾

Seconds/points

100

CREATE GAME ✎

Vibrations Game 📱🎮

We hope you enjoy our game to the fullest! 😊🎮

Copyright © LaSalle - Vibrations Game 2021.

Fig. 32: Página de espera usuario

- **Página de espera de jugadores (sala de espera):** Página en la que nos encontramos una vez creada la partida. En esta nos muestra un listado de los jugadores que se han unido y nos da la posibilidad de Iniciar el juego o eliminar la sala.

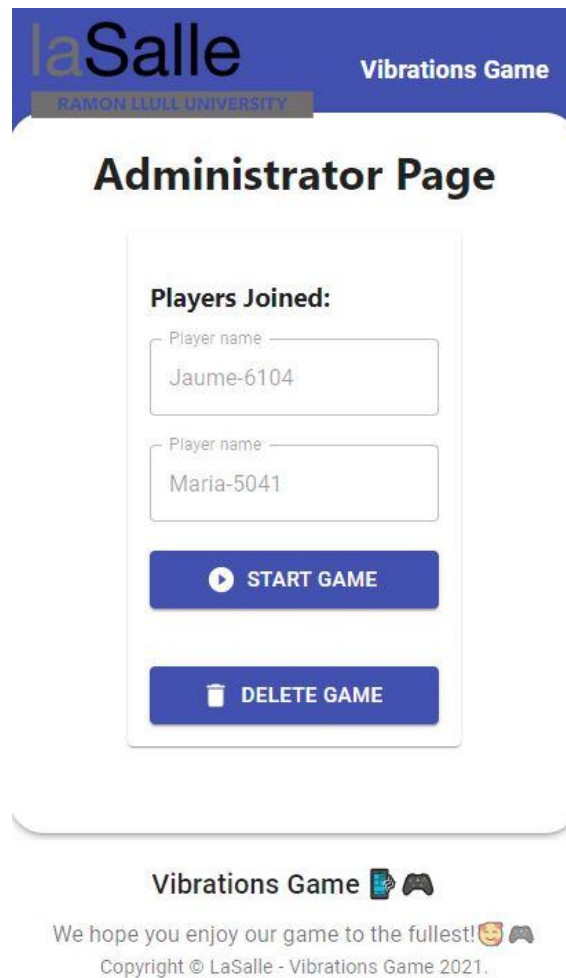


Fig. 33: Página de espera de jugadores

- **Página de partida en marcha:** Página en la que nos encontramos una vez se ha iniciado la partida. En esta página podemos observar el estatus de la partida: iniciada o finalizada, así como un listado de los diferentes usuarios y su puntuación casi a tiempo real (demora de 3seg). También nos brinda la oportunidad una vez la partida ha sido finalizada de crear una nueva partida.

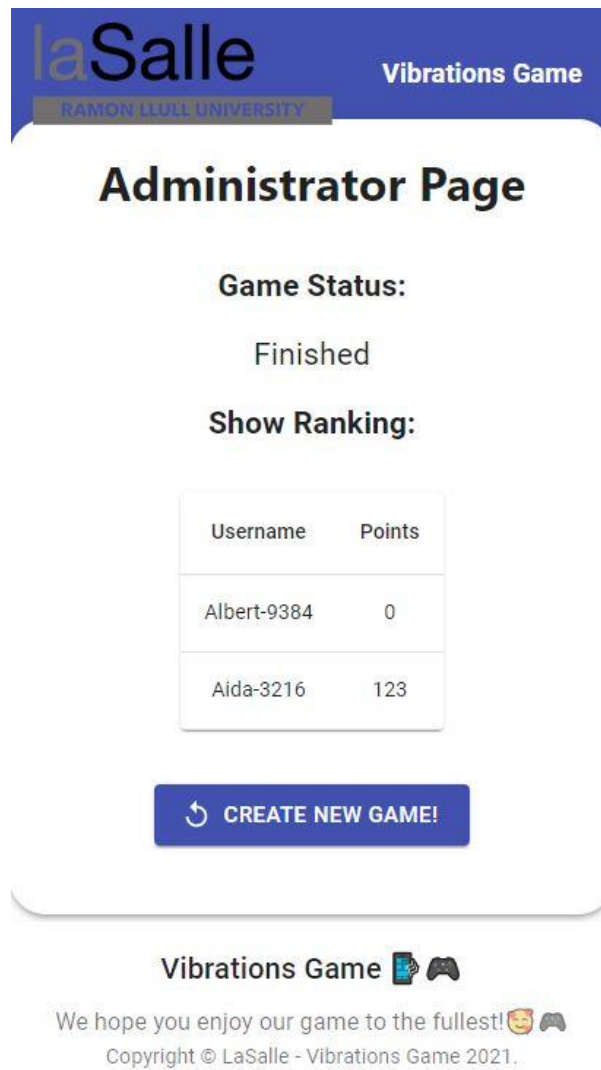


Fig. 34: Página de partida en marcha

6.5. Testing

Actualmente es muy difícil ver proyectos en los que no se apoye el desarrollo en uno o varios frameworks, tanto en la implementación funcional como en el desarrollo de otras partes complementarias como módulos de pruebas o de infraestructura. Por ello vamos a hablar ahora del framework utilizado que se han usado para la implementación de pruebas.

Jest

Jest es una librería que nos permite escribir y ejecutar tests, focalizándose en el concepto de simplicidad y rapidez. El test está completamente paralelizado de la ejecución en su propio proceso para maximizar el rendimiento, nos muestra indicadores de coverage y también dispone de un conjunto de excepciones y mocks muy fáciles de utilizar. Además de tener una gran comunidad y documentación, se nota que está altamente enfocado al desarrollador intentando de facilitarle la labor al máximo.

Diseño del test

La estructura de carpetas para el testing sigue la misma jerarquía de carpetas para facilitar la implementación de los test al desarrollador.

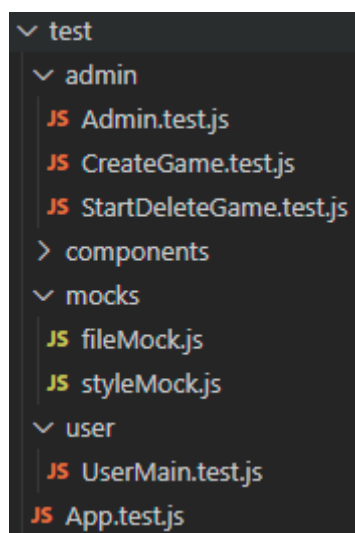


Fig. 35: Carpeta de testing Vibrations Game

Con el fin de poder realizar test sobre nuestra aplicación hemos configurado Jest para que automáticamente detecte todos los ficheros que acaben con la extensión “.test.js” para que los reconozca como ficheros de testing.

También para poder realizar el test correctamente sobre las diferentes vistas de la aplicación hemos necesitamos mockear todos los canvas, estilos e imágenes, así como t. Todas estas configuraciones se pueden encontrar en el fichero “jest.config.json”

Diseño e implementación de pruebas

En este apartado se explicarán los diferentes tipos de pruebas realizadas, así como que herramientas o metodologías se han usado en cada parte del proceso:

- Test Unitarios o Testing de componentes: comprueba el funcionamiento de una pieza o componente de forma aislada al resto de la aplicación. En esta parte debemos diferenciar entre la lógica y los componentes visuales. Para llevarlo a cabo hemos tenido que utilizar Enzyme el cual nos permite testear un componente sin tener en cuenta los posibles hijos de dicha clase. Y ayudándonos de las funciones que nos ofrece Jest.

```
PASS test/components/NicknameBoard.test.js
PASS test/components/slides/Figures/LeftPattern.test.js
PASS test/components/GameList.test.js
PASS test/components/slides/Figures/CirclePattern.test.js
PASS test/components/slides/Figures/DownPattern.test.js
PASS test/components/Title.test.js
PASS test/components/slides/Figures/UpPattern.test.js
PASS test/components/Footer.test.js
PASS test/components/slides/Figures/RightPattern.test.js
PASS test/components/Loading.test.js
PASS test/components/slides/Figures/ShakePattern.test.js
PASS test/components/Header.test.js
PASS test/admin/CreateGame.test.js
PASS test/components/slides/FigureManager.test.js (5.054 s)
PASS test/user/UserMain.test.js (5.57 s)
PASS test/admin/Admin.test.js (5.573 s)
PASS test/admin/StartDeleteGame.test.js (5.505 s)
PASS test/App.test.js (5.731 s)

Test Suites: 18 passed, 18 total
Tests:       18 passed, 18 total
Snapshots:   0 total
Time:        8.04 s
Ran all test suites.
```

Fig. 36: Ejecución de todos los test unitarios de la aplicación

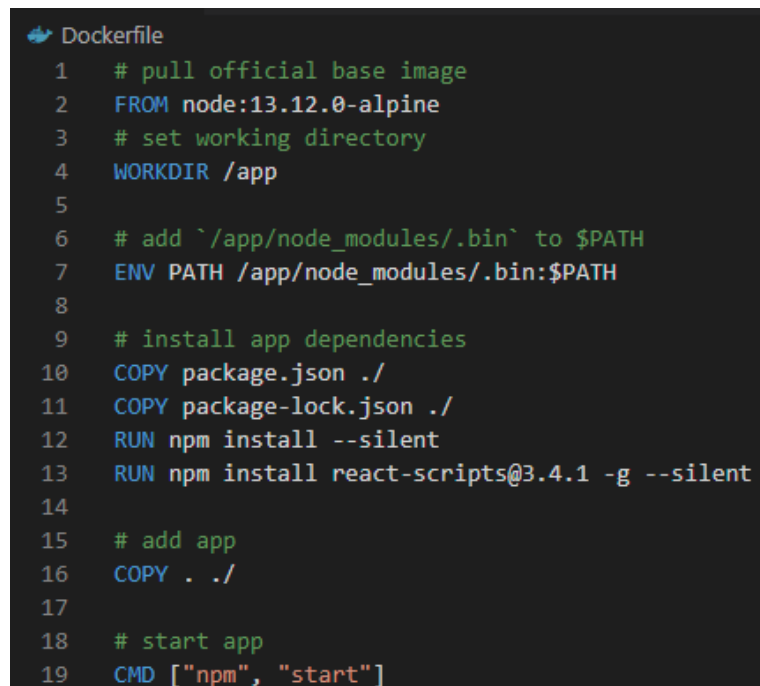
- Integración (end-to-end): Recorremos de forma manual todo el flujo de la aplicación de principio a fin, pasando por todas las rutas que nos ofrece el árbol de flujo, ya que la complejidad no es muy elevada.
- Aceptación (comportamiento): Se comprueba la aceptabilidad del sistema. Se ha evaluado punto por punto la conformidad del sistema revisando los requisitos descritos al principio del proyecto, con el fin de cerrar la aceptación de este para conseguir esto se han realizado reuniones con todos los integrantes del grupo para validarlo.

6.6. Generación imagen Docker:

Un Dockerfile es un documento de texto que contiene todos los comandos que un usuario podría llamar en la línea de comandos para montar una imagen. Usando docker build podemos crear una construcción automatizada que ejecuta varias instrucciones de línea de comandos en sucesión, es lo que se conoce como IaaS (Infraestructure as a Code).

A continuación, se explicará que pasos se realiza en el Dockerfile para generar nuestra imagen de nuestro aplicativo frontend en React:

1. Primero descargamos la imagen oficial base de Node.
2. Creamos un directorio, en el que se volcara toda la aplicación
3. Definimos las variables de entorno PATH para un acceso rápido.
4. Copiamos nuestro “package.json” y “package.lock.json” (contiene el listado de dependencias del proyecto) al contenedor.
5. Se instalan dichas dependencias con el comando “npm install”.
6. También se instala la librería de ‘react-scripts’.
7. Se copia todo el contenido del proyecto hacia la carpeta app del contenedor.
8. Lanzamos por línea de comandos “npm start” para que levante la aplicación.



```
Dockerfile
1  # pull official base image
2  FROM node:13.12.0-alpine
3  # set working directory
4  WORKDIR /app
5
6  # add `/app/node_modules/.bin` to $PATH
7  ENV PATH /app/node_modules/.bin:$PATH
8
9  # install app dependencies
10 COPY package.json ./
11 COPY package-lock.json ./
12 RUN npm install --silent
13 RUN npm install react-scripts@3.4.1 -g --silent
14
15 # add app
16 COPY . ./
17
18 # start app
19 CMD ["npm", "start"]
```

Fig. 37: Dockerfile encargado de generar la imagen de Docker del frontend

A la hora de levantar dicho contenedor podemos especificarle el puerto por el cual queremos que se muestre el frontal de Vibrations Game.

Más adelante se explicará como dicho contenedor es levantado, gestionado y orquestado con el resto de componentes (backend y tensorflow) utilizando Kubernetes para así conseguir unificar todo el ecosistema de nuestra aplicación.

7. Planificación y gestión del proyecto

Como se ha comentado anteriormente, este proyecto dispone de 4 partes:

- Frontend
- Backend + Apache Kafka
- Tensorflow
- Kubernetes

Cada uno de los tres miembros asignados a este proyecto desarrollará una parte.

La asignación de las partes queda de la siguiente manera:

- Frontend: Cristian Vega
- Backend + Apache Kafka: Joan Temprano
- Tensorflow: Carlos Martínez

Con respecto a la parte de Kubernetes como afecta a las tres partes, se ha repartido el trabajo entre los tres miembros del proyecto.

Sobre la planificación del frontend podríamos definir que se ha realizado en las siguientes etapas:

- Extracción, comprensión y definición de requerimientos: aterrizar la idea y hacerla propia, reuniones periódicas con el tutor del proyecto para entender perfectamente las necesidades de la aplicación requerida. Así como definir también entre los miembros del grupo las necesidades finales del proyecto.
- Diseño UI/UX: realización de primeras aproximaciones, primeramente, se realizó a papel para obtener una idea básica de forma rápida y poco a poco con herramientas de diseño ir definiendo cada vez más los componentes visuales. Esta parte era bastante flexible, ya que no teníamos que validar la parte visual con el cliente final, ya que teníamos libertad de decisión.
- Estudio de las tecnologías y creación de las bases del proyecto: Decidir el stack tecnológico que más se adecuaba a nuestras necesidades de proyecto, realizando una correcta investigación previa. Además de definir la estrategia de testing a seguir. Acto seguido se definió la estructura de carpetas y descargaron todas las dependencias para establecer el esqueleto del código.
- Implementación: Desarrollo directo del código. Este es el apartado con más horas invertidas.
- Testing: Ejecutar la estrategia de testing. Realizando códigos unitarios parcialmente por cada implementación y finalmente realizar los test de integración y aceptación.
- Refactorización y limpieza de código: tratar de optimizar la solución generada, de este modo tratar de obtener una solución limpia y funcional.
- Verificación: comprobar que se cumplían todos los requisitos previamente definidos tanto funciona como no funcionales y los objetivos del proyecto.
- Documentación: redactar todo el trabajo realizado tanto para la memoria final como para el Readme de GitLab.

Comentar que en todas las etapas también ha habido una fuerte integración y comunicación constante con el resto del equipo para poder alinearnos y establecer correctamente la comunicación entre los diferentes servicios.

Kubernetes

Seguidamente podemos observar el diagrama de contenedores para tener una visión global de la orquestación realizada con kubernetes, así como los diferentes contenedores:

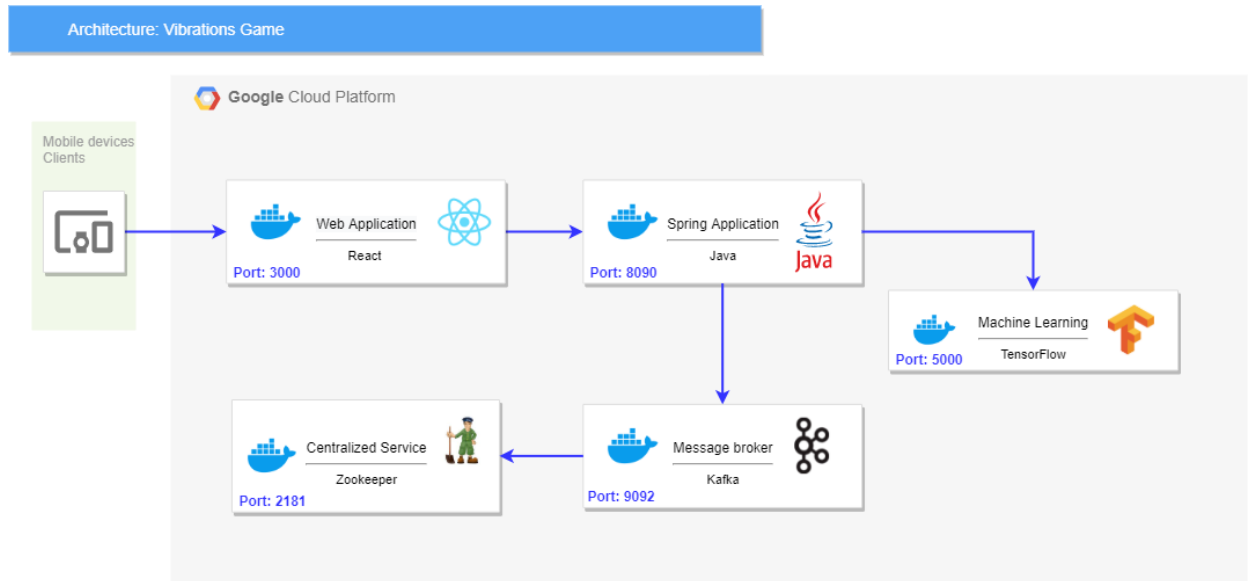


Fig. 38: Diagrama de contenedores Vibrations Game

Servicios

A continuación, se listarán todos los servicios con sus correspondientes puertos de acceso, así como también la dirección web en la cual se encuentra la imagen alojada. Dicha dirección de las imágenes Docker, Kubernetes la utilizará para descargar y poder levantar todo el ecosistema:

Nombre del Servicio	Puerto	Repositorio de imágenes Docker
Frontend React	3000	registry.gitlab.com/mdastfm/front-end-development:master
Backend Java	8090	registry.gitlab.com/mdastfm/backend:1.0.0
Backend Kafka	9092	docker.io/bitnami/kafka:2
Backend Zookeeper	2181	docker.io/bitnami/zookeeper:3
Backend Tensorflow- Motion Predictor	5000	registry.gitlab.com/mdastfm/machinelearning/motionpredictor:v1.0.0

Recordar que las imágenes de Docker son generadas automáticamente cuando se origina un merge en master o se crea un tag en GitLab. De esta manera siempre tendremos nuestras imágenes actualizadas con los últimos cambios realizados.

Despliegue

Hemos decidido apostar por la técnica de despliegue blue-green, ya que reduce el tiempo de inactividad de la aplicación, así como también el riesgo, utilizando dos entornos de producción idénticos.

Esta técnica consiste en tener dos entornos activos, normalmente formado por:

- Blue container: versión publica corriendo actualmente, normalmente una versión estable del entorno.
- Green container: Nueva implementación de código que se quiere lanzar al público.

El tráfico de usuarios accede al entorno estable de producción (blue container) y cuando se decide cambiar a la nueva implementación (green container) se redirecciona el tráfico a la nueva versión, manteniendo los dos entornos activos.

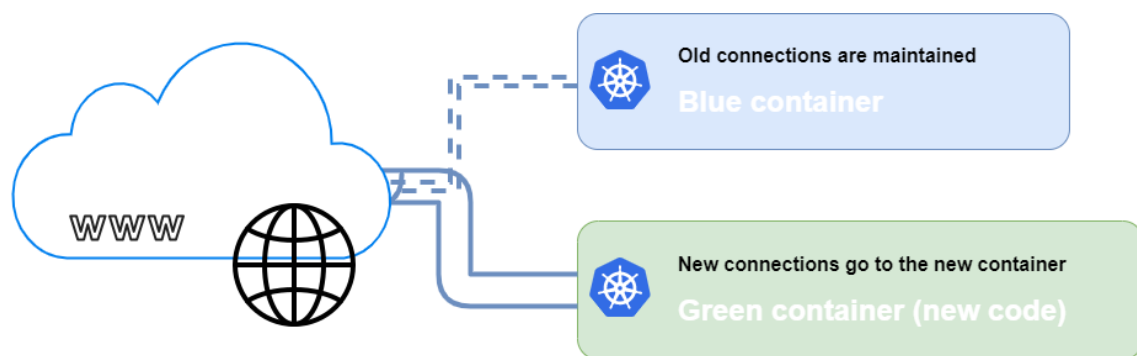


Fig. 39: Diagrama despliegue blue/green

Esta técnica nos permite fácilmente realizar un rollback cambiando el direccionamiento del tráfico a la versión estable anterior si se detecta alguna anomalía en el comportamiento de la nueva implementación.

8. Análisis de resultados

Por lo que respecta a los objetivos generales podemos decir que se han cumplido todos. Se ha conseguido lo siguiente:

- Integrar Kafka en la solución. El backend es capaz de enviar mensajes a un tópico de Kafka. También se ha configurado un stream de Kafka en el backend que consume y transforma este mensaje.
- Se ha implementado una aplicación usando la plataforma de Tensorflow con el lenguaje Python. Esta aplicación realiza la funcionalidad esperada y forma parte de la solución del proyecto.
- El módulo del frontend se ha implementado con la funcionalidad deseada. Con el frontend operativo para los usuarios, tanto para administradores como para jugadores, pueden interactuar con el juego de Vibrations Game.
- La parte del backend también cumple con la funcionalidad esperada. Puede recibir y procesar mensajes del frontal, como también conectarse con los módulos de Apache Kafka y Tensorflow para poder completar las funcionalidades deseadas.
- La solución dispone de scripts que permiten desplegar Vibrations Game en un entorno de Kubernetes.

A nivel más personal podríamos decir:

- Se han comprendido las necesidades de la aplicación, puesto que se han validado con el tutor del trabajo.
- Hemos sido capaces de afrontar todas aquellas carencias de conocimiento y arreglar todos los problemas que han ido surgiendo a lo largo de la implementación.
- La comunicación ha fluido perfectamente entre los diferentes miembros y el tutor, con reuniones periódicas constante a lo largo del proyecto. Más concretamente la relación entre el backend y el frontal ha sido muy estrecha, ya que teníamos que ir validando el correcto funcionamiento constantemente.
- El diseño es bastante atractivo y amigable, ya que se han colocado muchas animaciones.
- Se ha realizado una definición y diseño inicial bastante bueno, lo cual no ha hecho aumentar mucho el tiempo de refactorización.
- Hemos conseguido llegar a tiempo con los deadlines marcados y hemos estimado el tiempo bastante bien.
- Finalmente hemos conseguido desplegar todo el ecosistema en un entorno de producción abierto al público.

9. Estudio económico

En este apartado veremos el coste en horas y económico del proyecto. Este coste estará dividido entre el coste del desarrollo, coste del software utilizado y el coste de la infraestructura necesaria para desplegarlo.

Coste desarrollo

El coste del desarrollo corresponde al coste de la mano de obra del proyecto. En la tabla siguiente encontraremos detallados los costes de cada una de las partes desarrolladas. También hay una parte de análisis de proyecto en el que se incluye un estudio de las tecnologías utilizadas y sus limitaciones juntamente con la definición de la especificación y el diseño a utilizar. Hay incluido también un apartado para redactar la documentación del proyecto.

Concepte	Hores	Preu/hora	Import
Anàlisis del projecte	164	20	3280€
Desenvolupament React	200	15	3000€
Desenvolupament Java	168	15	2520€
Desenvolupament Tensorflow	124	15	1860€
CI/CD	40	15	600€
Documentació	154	15	2310€
Preu sense IVA			13570€
IVA 21%			2849,70€
Preu total			16419,70€

Coste del software

En este apartado encontraremos el coste del software utilizado en el transcurso del desarrollo del proyecto.

Concepte	Quantitat	Preu
Llicencia 1 any IntelliJ	1	499€
Llicencia 1 any PyCharm	1	199€
GitLab Premium 1 any	3	456,06€
Preu sense IVA		1154,06€
IVA 21%		242,35€
Preu total		1396,41€

Coste infraestructura

Por esta parte, se tendrá en cuenta que la aplicación se desplegará en el proveedor de cloud de Google. El datacenter que se ha seleccionado para la siguiente estimación es el que está situado en Bélgica. El coste que se mostrará en la siguiente tabla será mensual.

Concepte	Quantitat	Preu
Llicencia 1 any IntelliJ	1	499€
Llicencia 1 any PyCharm	1	199€
GitLab Premium 1 any	3	456,06€
Preu sense IVA		1154,06€
IVA 21%		242,35€
Preu total		1396,41€

Coste total

Finalmente, en la siguiente tabla se muestra el coste total del proyecto teniendo en cuenta los apartados anteriores.

Concepte	Preu
Cost desenvolupament	16419,70€
Cost software	1396,41€
Cost d'infraestructura (1 any)	1665,72€
Cost Total	19481,83€

10. Conclusiones

En el momento de elegir este proyecto para el trabajo final de master, se decidió este proyecto debido a su temática. Hacer una aplicación de una temática lúdica donde interactuasen elementos como websockets, programación de eventos con Apache Kafka y Machine Learning con Tensorflow lo hacía interesante y una novedad respecto a otros proyectos que estábamos acostumbrados hasta ahora.

Después de los esfuerzos y dedicación del equipo, se ha conseguido una aplicación funcional que cumple con los requerimientos que se esperaban de esta. Se ha conseguido un módulo de frontend funcional con el que los usuarios interactúan. También tener un módulo de backend que es capaz de procesar las peticiones realizadas por el frontend y de integrarse con la plataforma de Apache Kafka y el módulo hecho con la plataforma de Tensorflow. Finalmente tenemos un módulo de Tensorflow que contiene el modelo que le permite, dado un listado de coordenadas capturadas, saber qué tipo de figura corresponde.

Esta aplicación dispone también de herramientas para generar imágenes Docker y hacer que estas se desplieguen en un entorno de Kubernetes.

En líneas generales, se ha conseguido todos los objetivos que se había propuesto al principio del proyecto a pesar de tener que trabajar con tecnologías que no habíamos utilizado antes o que hacía tiempo que no tocábamos.

A título personal decir que me siento bastante orgulloso con el resultado final de nuestra aplicación, ya que como se ha comentado anteriormente había una carencia importante a nivel de conocimiento de las tecnologías, el cual ha supuesto una formación autodidacta aplicada a un proyecto real de una cantidad de horas elevada.

Creo que hemos sabido colaborar muy bien entre nosotros para salir adelante, mediante grupos de chat y reuniones periódicas, para integrar los diferentes servicios. Más concretamente en mi caso particularmente estaba más interconectado con el backend ya que teníamos que ir probando lo que el otro realizaba y coordinarnos.

En muchos momentos parecía que no llegaríamos al deadline, debido a que teníamos mucha carga de trabajo por parte del Master, pero gracias al esfuerzo de todos y la ayuda de nuestro tutor Alex Soto lo hemos conseguido.

También se me queda una sensación de querer mejorar la aplicación, porque debido al poco tiempo se ha tenido que desarrollar buscando una funcionalidad aceptable, pero pienso que si disponemos de más tiempo se puede obtener mucha más calidad, esto se comentara en el siguiente apartado más detalladamente.

11. Líneas de futuro

Como se ha comentado en el apartado anterior, por falta de tiempo hay muchas cosas que nos gustaría añadir a nuestro proyecto para aumentar así la calidad de este. Seguidamente comentaremos los puntos que nos gustaría añadir de cara a un futuro:

- Securizar el endpoint del Administrador para limitar su acceso, utilizando algún tipo de login o algún redireccionar de tráfico basado en el proxy.
- Crear animaciones para las figuras que se le muestra al usuario para que realice, quizás un avatar sosteniendo el móvil realizando el patrón. De esta manera quedaría mucho más claro el movimiento a realizar y mejoraría la experiencia del usuario.
- Utilización de frameworks o herramientas para las pruebas de aceptación. Hoy en día sabemos que existe muchos frameworks que facilitan mucho la automatización de las pruebas, alguno de estos podría ser NightmareJS junto con Jest.
- También se había pensado crear una página sencilla que hablase de ¿Quiénes somos? En la cual encontrar información de todos los integrantes del equipo, así como del tutor y la universidad.
- Añadir más componentes Snackbars de ayuda al usuario. Los snackbars proporcionan mensajes breves sobre los procesos de la aplicación. Este componente también se conoce como toast. Actualmente solo disponemos de uno en la creación de la partida, otros casos podrían ser cuando la sala es eliminada, cuando el usuario acierta/falla una figura, etc. Añadir esto no supondrá mucho tiempo ya que sería replicar el existente.

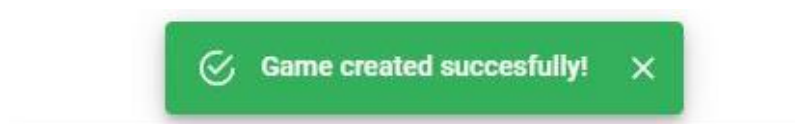


Fig. 40: Snackbars que le aparece al administrador al crear la partida

- Quizás un cambio que implicaría muchas horas de trabajo, pero que me gustaría realizar tanto como para aprender como para mejorar la calidad del código sería utilizar TypeScript para el tipado de todos los componentes de la aplicación.
- Otra propuesta sería realizar la aplicación multi lenguaje, ya que como tiene un diseño tan minimalista sería un cambio bastante sencillo.
- Una propuesta también orientada en mejorar la experiencia de usuario y bastante utilizado en el sector de los videojuegos sería añadirle música tanto a las pantallas de carga como al juego.

12. Bibliografía

- [1] Node.js – Información de NodeJS [consulta: 1 julio 2021]. Disponible en: <https://nodejs.org/>
- [2] React – Biblioteca de JavaScript [consulta: 1 julio 2021]. Disponible en: <https://reactjs.org/>
- [3] Material-UI – ReactUI Framework [consulta: 1 julio 2021]. Disponible en: <https://material-ui.com/>
- [4] JavaScript MDN – Información [consulta: 1 julio 2021]. Disponible en: <https://developer.mozilla.org/es/docs/Web/JavaScript>
- [5] Draw.IO – Diagrama Software [consulta: 1 julio 2021]. Disponible en: <https://app.diagrams.net/>
- [6] SensorJS – Example JS device sensor [consulta: 1 julio 2021]. Disponible en: <https://sensor-js.xyz/demo.html>
- [7] VSCode – Code IDE [consulta: 2 julio 2021]. Disponible en: <https://code.visualstudio.com/>
- [8] HTML – HTML Tutorial [consulta: 2 julio 2021]. Disponible en: <https://www.w3schools.com/html/>
- [9] CSS – CSS Tutorial [consulta: 2 julio 2021]. Disponible en: <https://www.w3schools.com/css/>
- [10] NPM – Node JavaScript Platform [consulta: 2 julio 2021]. Disponible en: <https://www.npmjs.com/>
- [11] LottieFiles – Free Lottie Animation Files [consulta: 2 julio 2021]. Disponible en: <https://lottiefiles.com/>
- [12] OpenSSL – Cryptography and SSL/TLS Toolkit [consulta: 2 julio 2021]. Disponible en: <https://www.openssl.org/>
- [13] GitLab – Code versión control repository [consulta: 2 julio 2021]. Disponible en: <https://gitlab.com/>
- [14] Docker – Development containers [consulta: 2 julio 2021]. Disponible en: <https://www.docker.com/>
- [15] Kubernetes – Management and container orchestration [consulta: 2 julio 2021]. Disponible en: <https://kubernetes.io/>
- [16] Microsoft Office – Office application package [consulta: 2 julio 2021]. Disponible en: <https://www.microsoft.com/en-us/microsoft-365>
- [17] Chrome Dev – Google Chrome Development tools [consulta: 2 julio 2021]. Disponible en: <https://www.google.com/intl/ca/chrome/dev/>
- [18] Moqups – Design web app [consulta: 2 julio 2021]. Disponible en: <https://moqups.com/>

- [19] Websocket – API Web JavaScript [consulta: 2 julio 2021]. Disponible en: https://developer.mozilla.org/es/docs/Web/API/WebSockets_API/Writing_WebSocket_client_applications
- [20] Device Motion – Window: devicemotion event [consulta: 2 julio 2021]. Disponible en: https://developer.mozilla.org/en-US/docs/Web/API/Window/devicemotion_event
- [21] Apache Kafka – Window: devicemotion event [consulta: 2 julio 2021]. Disponible en: <https://kafka.apache.org/>
- [22] Zookeeper – Apache zookeeper [consulta: 2 julio 2021]. Disponible en: <https://zookeeper.apache.org/>
- [23] Google Cloud – Google Cloud: Cloud Computing Services [consulta: 2 julio 2021]. Disponible en: <https://cloud.google.com/>
- [24] PyCharm – Python IDE [consulta: 2 julio 2021]. Disponible en: <https://www.jetbrains.com/pycharm/>
- [25] IntelliJ – IntelliJ IDE [consulta: 2 julio 2021]. Disponible en: <https://www.jetbrains.com/idea/>
- [26] Tensorflow – Machine Learning platform [consulta: 2 julio 2021]. Disponible en: <https://www.tensorflow.org/>
- [27] Jest – Testing library support [consulta: 2 julio 2021]. Disponible en: <https://jestjs.io/>
- [28] Nightmare– Acceptance testing library [consulta: 3 julio 2021]. Disponible en: <https://codecept.io/nightmare/>