MEMORIA DE PROYECTO FIN DE CICLO DAW

IES Bernaldo de Quirós

2023 – 2024

ROBERTO FERNÁNDEZ PÉREZ

INDICE

Pádel Mánager – Introducción

Presentación y objetivos

Ante el creciente sedentarismo que invade la actual sociedad en la que vivimos, surgen ideas de incentivar el movimiento, el deporte y los hábitos saludables. Aprovechando el auge de un deporte muy dinámico y apto para todos los públicos como es el pádel, apoyamos la creación y/o la promoción de un centro deportivo que albergue unas instalaciones para su práctica con una aplicación web dinámica y amigable.

La aplicación pretende ser una herramienta fácil de utilizar e intuitiva para el consumidor, a la vez que una manera dinámica de darse a conocer y gestionarse para la empresa. Contará con una página de presentación, donde se podrán recorrer las distintas funcionalidades mediante un navegador, observar las últimas noticias y avisos, o acceder a las redes sociales y contactos del centro. Se podrá realizar un login para realizar diversas funcionalidades o disfrutar servicios.

Contexto y planteamiento del problema

La idea de este proyecto surge de una afición profunda por el pádel, una especie de deporte más versátil y dinámico que otros deportes de raqueta. Cuando llevas cierto tiempo integrándote en este contexto, uno se da cuenta de que existe una gran cantidad de centros que no cuentan con las mejores tecnologías y medios para facilitar la vida a los jugadores. Poder acceder y comprobar rápidamente la disponibilidad de las pistas, poder adquirir el equipamiento necesario o ayudar a encontrar una pareja o un equipo para jugar resolvería los problemas más importantes de cualquier jugador.

En mi posición como deportista, y ante la frustración de llegar a una pista y encontrarse con que está ocupada, llegué a la conclusión de que en la época en la que vivimos, debería existir una tecnología que evite estos problemas.

Con la creación de esta aplicación, buscamos ayudar a los centros a ofertarse en el mercado digital, y facilitar la gestión del mismo, a la vez que mejoramos la interacción de los usuarios y damos una buena imagen.

Análisis de costes

El análisis de costes es un proceso crucial de la planificación y la gestión de proyectos, donde se determinan los recursos financieros necesarios para llevar a cabo el proyecto de manera eficiente. Se identifican y estiman los costos asociados directos e indirectos, con el fin de asegurar que los recursos sean empleados de la manera más eficaz posible y haciendo económicamente viable el proyecto.

Teniendo en cuenta varios factores, como el del desarrollo de la aplicación web por una persona desde su despacho en casa, se realizan ciertas estimaciones de los posibles gastos que se pretenden asumir:

- Mobiliario
  - Escritorio – 150€
  - Silla ergonómica – 100€
  - Armario pequeño – 80€
  - TOTAL: 330€
- Alquiler de piso
  - 300€/mes – 3.600€ anuales
- Teléfono + Internet
  - Plan de teléfono móvil y fibra óptica: 60€/mes – 720 anuales
- Luz
  - 50€/mes – 720€ anuales
- Material de oficina
  - 50€/mes – 600€
- Publicidad
  - Google Ads, redes sociales… - 100€/mes – 1.200€ anuales
- Servicio de hosting y Gestión de BDD
  - Hosting Web – 20€/mes
  - Gestión de BDD – 15€/mes
  - TOTAL: 420€ anuales
- Otros gastos
  - Licencias de software – 200€
  - Seguro y nóminas – 1.500€
  - PC – 2.000€
- TOTAL ANUAL – 11.170€

Tras el análisis, se estima un gasto anual total de aproximadamente 11.170 €. La viabilidad económica del proyecto parece favorable, siempre que se mantenga un control riguroso de los gastos y se optimicen los recursos disponibles. Con una adecuada planificación y gestión financiera, el proyecto puede desarrollarse de manera eficiente, permitiendo alcanzar los objetivos establecidos sin exceder el presupuesto estimado.

Plan de financiación

Se prevé que la aportación inicial de capital como los primeros compases de vida de la aplicación será un desembolso propio que abarque los gastos necesarios. Si fuese necesario, se puede recurrir a fuentes adicionales de financiación, como préstamos bancarios, ayudas y subvenciones y sobretodo la captación de nuevos inversores que vean posibilidades de crecimiento en el futuro de la aplicación.

Sería necesario realizar ciertas revisiones (trimestrales, en principio) del estado financiero y por ende, el ajuste de estrategia y presupuesto, asegurando la sostenibilidad financiera. <A su vez, mensualmente se hará un Plan de Control Presupuestario, donde figuren y se monitoricen los gastos mensuales y asegurar que se encuentren en los límites estimados.

Plan de recursos humanos

Al tratarse de un desarrollo de un proyecto individual, todas las responsabilidades recaen sobre una persona, que debe de cumplir una serie de roles y responsabilidades o valorar si deben ser externalizados, como:

- Desarrollador Web y Gráfico
  - Diseño y desarrollo de la aplicación web.
  - Implementación de funcionalidades y características principales.
  - Mantenimiento y actualización del código.
  - Gestión de la base de datos.
  - Aseguramiento de la calidad del software.
- Marketing y Publicidad
  - Desarrollo y ejecución de estrategias de marketing digital.
  - Gestión de campañas publicitarias en redes sociales y Google Ads.
  - Análisis de datos y optimización de campañas.
- Dirección y Soporte técnico
  - Visión y estrategia.
  - Planificación y organización.
  - Gestión de recursos, supervisión y control.
  - Innovación, adaptación y mejor.

La clave del éxito en este proyecto sea posiblemente la externalización de algún apartado de estilo gráfico (accesibilidad) y el marketing.

Plan de prevención de riesgos

Se busca garantizar la seguridad y la salud de los trabajadores involucrados en el desarrollo de la aplicación. Se abordan los riesgos más comunes en el entorno:

- Ergonómicos
  - Malas posturas .
  - Movimientos repetitivos.
- Psicosociales
  - Estrés laboral.
  - Asilamiento social.
- Físicos
  - Fatiga visual.
  - Iluminación inadecuada.
- Seguridad
  - Incendios.
  - Cables sueltos.

Evaluando cada apartado y sus posibles soluciones, se concluye que las medidas de prevención a tomar para estos posibles riesgos son:

- Sillas ergonómicas y pausas regulares para estiramientos y cambios de postura reduciendo así la fatiga muscular.
- Ajuste de iluminación en pantallas y uso de filtros antirreflejos.
- Fomentar ejercicios de relajación para manos y muñecas.
- Gestión de carga de trabajo adecuada y apoyo psicológico más programas de bienestar
- Fomento de la comunicación y el trabajo en equipo.
- Mantenimiento regular de equipos eléctricos y cables, aparte de uso de canaletas y pasacables.
- Instalación de detectores de humo.

Pese a estas medidas, se deben también llevar a cabo inspecciones regulares, a la vez que implementar formación continua en estos temas, debido a que son recurrentes y nunca se está demasiado preparado para lo inesperado.

Especificación de requisitos

Introducción y descripción general

La aplicación web está compuesta por tres capas principales: presentación, lógica de negocio y persistencia de datos. La capa de presentación utiliza React y Bootstrap para construir una interfaz de usuario intuitiva y atractiva. La capa de lógica de negocio, implementada con Node.js y Express, gestiona las reglas de negocio y la interacción con los datos. La capa de persistencia de datos utiliza MySQL, gestionada a través de Sequelize, para asegurar un almacenamiento de datos eficiente y seguro.

Requisitos específicos

Funcionales

- **Autenticación de Usuarios**: El sistema debe permitir a los usuarios registrarse, iniciar sesión.
- **Notificaciones**: El sistema enviará notificaciones automáticas vía email o dentro de la plataforma para eventos importantes.
- **Gestión de Contenidos**: Los usuarios podrán crear, editar, eliminar y visualizar contenido de forma intuitiva.

Interfaces

- **Interfaz de Usuario (UI)**: Debe ser intuitiva y fácil de navegar, con un diseño responsive que se adapte a dispositivos móviles y de escritorio.
- **Interfaz de Administración**: Herramientas para que los administradores gestionen usuarios, contenidos y configuraciones del sistema.
- **API Pública**: Debe existir una API bien documentada para que desarrolladores externos puedan interactuar con el sistema.
- **Panel de Control**: Un dashboard interactivo donde los usuarios puedan ver y gestionar todas sus actividades y configuraciones.

Rendimiento

- **Tiempo de Respuesta**: El tiempo de respuesta del sistema debe ser inferior a 2 segundos para todas las operaciones críticas.
- **Escalabilidad**: El sistema debe ser capaz de manejar un incremento del 100% en el número de usuarios sin degradación del rendimiento.
- **Disponibilidad**: El sistema debe estar disponible un 99.9% del tiempo, exceptuando mantenimientos programados.
- **Optimización**: El uso de recursos del sistema debe ser eficiente, minimizando el consu mo de CPU y memoria.

Obligaciones de diseño

- **Accesibilidad**: El diseño debe cumplir con las directrices WCAG 2.1 para garantizar que el sistema sea accesible para personas con discapacidades.
- **Consistencia**: Se debe mantener una coherencia visual y funcional en todas las interfaces del sistema.
- **Usabilidad**: Realizar pruebas de usabilidad con usuarios finales para asegurar que el sistema es fácil de usar y entender.
- **Seguridad**: El diseño debe incorporar medidas de seguridad desde el principio, incluye ndo el cifrado de datos y la protección contra ataques comunes.

Atributos

- **Seguridad**: Todos los datos sensibles deben ser almacenados y transmitidos de manera segura, cumpliendo con los estándares de la industria.
- **Mantenibilidad**: El código debe estar bien documentado y estructurado para facilitar el mantenimiento y la actualización del sistema.
- **Portabilidad**: El sistema debe ser compatible con múltiples plataformas y navegadores.
- **Reusabilidad**: Los componentes del sistema deben ser diseñados de manera que puedan ser reutilizados en diferentes partes del proyecto o en proyectos futuros.
- **Confiabilidad**: El sistema debe ser robusto y minimizar los errores y caídas durante su uso.

Análisis

Introducción

La aplicación web se estructura en capas: presentación, lógica de negocio y persistencia de datos, utilizando tecnologías modernas como React, Node.js, Express y MySQL gestionado con Sequelize. Este diseño asegura una arquitectura robusta, modular y escalable, facilitando el desarrollo y mantenimiento del software. En este documento se detallan los diagramas de clases y casos de uso para ofrecer una visión clara de la estructura y funcionalidad del sistema

Capa de clases

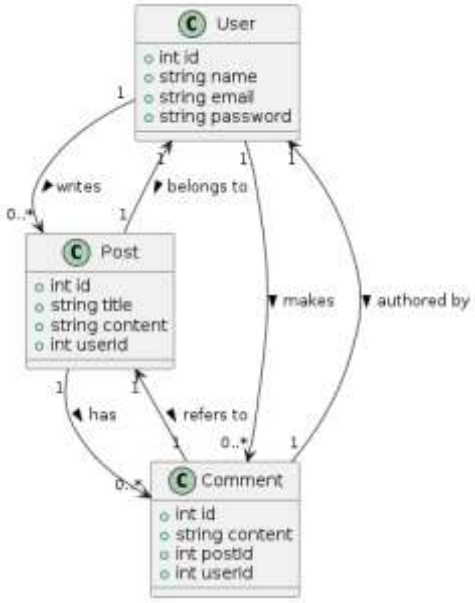
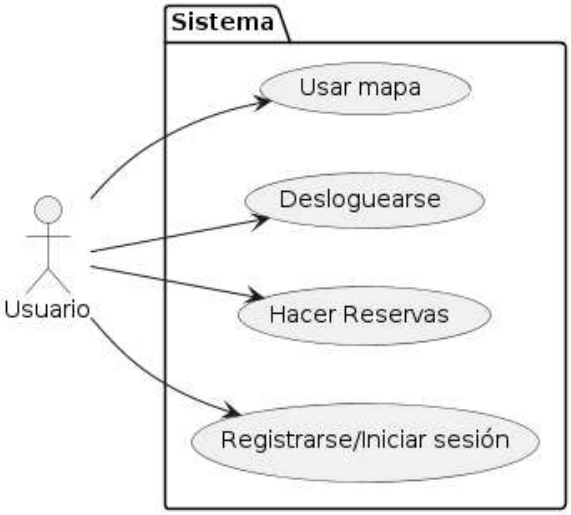


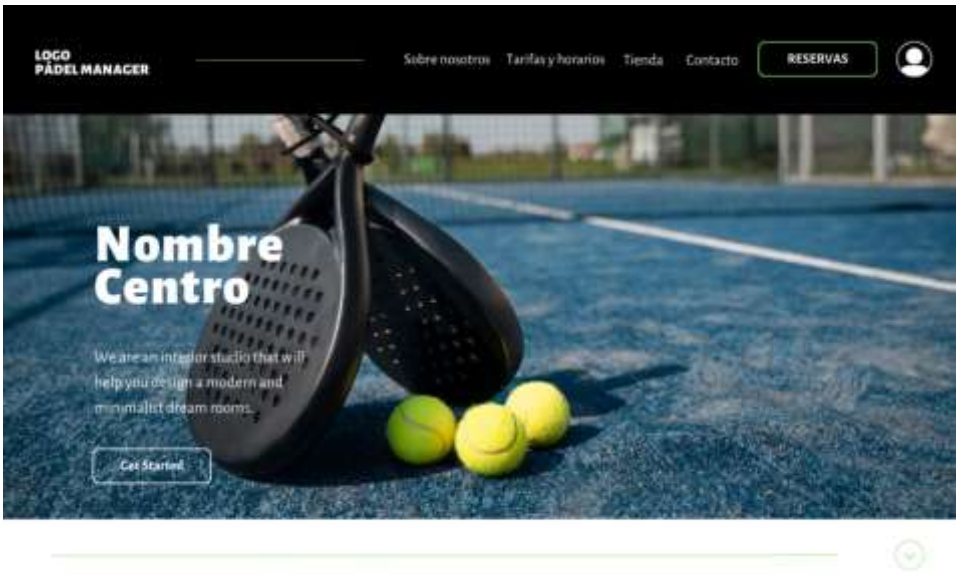
Diagrama de casos de uso



Diseño

Introducción

Para llevar a cabo el diseño de esta aplicación, se ha intentado utilizar colores que fuesen llamativos y fáciles de ver, que combinasen juntos y que a la vez dieran una sensación de estar relacionados con el pádel. Implementando una interfaz visual suave con contrastes grandes entre fondos y texto, la interfaz ha terminado por ser fácil e intuitiva, a la vez que agradable y que llama la atención. A continuación, figura un boceto de lo que se pretendía obtener.



Por otro lado, el diseño del backend y de la capa de datos resultó bastante sencillo, dado que no cuenta con gran complejidad ni cantidad de elementos o interconexiones.

Capa de presentación

En el diseño de la interfaz de usuario de este proyecto, se ha optado por utilizar Bootstrap como el marco principal debido a su capacidad para facilitar la creación de interfaces modernas, responsivas y coherentes. Bootstrap proporciona un conjunto robusto de componentes preestilizados y un sistema de rejilla flexible que permite un desarrollo ágil y eficiente. Además, se han utilizado clases auxiliares de CSS personalizadas para ajustar y mejorar ciertos aspectos visuales específicos que no se podían cubrir únicamente con Bootstrap. Este enfoque combinado asegura que la aplicación no solo sea visualmente atractiva, sino también accesible y fácil de usar en una variedad de dispositivos y tamaños de pantalla.

Las clases que han sido implementadas por CSS de apoyo se encuentran en la App.js, y cuentan con nombres bastante indicativos. Se emplea el uso de variables globales para ciertos colores, que facilitan el mantenimiento o si se desea realizar algún cambio futuro.

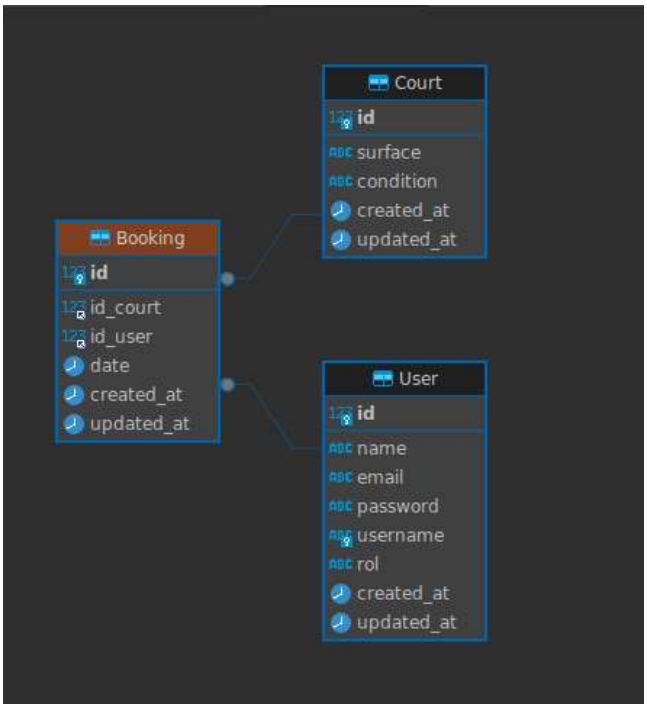
Capa lógica y capa de datos

La capa de datos está diseñada utilizando MySQL, un sistema de gestión de bases de datos relacional ampliamente reconocido por su rendimiento, fiabilidad y facilidad de uso. Esta capa

es responsable de almacenar, gestionar y recuperar los datos necesarios para el funcionamiento de la aplicación.

La lógica de esta aplicación se ha diseñado utilizando Node.js con el framework Express y el ORM Sequelize. Esta capa es responsable de gestionar la lógica de negocio y la interacción con la base de datos, proporcionando una estructura sólida y escalable para el desarrollo de la aplicación.

Cuando una solicitud HTTP llega a la aplicación, se procesa por el controlador correspondiente. El controlador valida la solicitud y llama a los métodos necesarios en los servicios. Los servicios, a su vez, interactúan con los modelos de Sequelize para realizar las operaciones en la base de datos. Finalmente, la respuesta se envía de vuelta al cliente. Este diseño modular y bien estructurado asegura que la aplicación sea fácil de mantener, escalar y probar, proporcionando una base sólida para el desarrollo de nuevas funcionalidades.



Implementación

Tecnologías implementadas

Para el desarrollo del proyecto, se ha optado por el uso de JavaScript utilizando React para el frontend y Express en Node.js para el backend. React es una biblioteca de JavaScript ampliamente utilizada para construir interfaces de usuario, conocida por su enfoque basado en componentes, que facilita la creación de aplicaciones dinámicas y de alto rendimiento. Entre las ventajas de React se encuentran la capacidad de reutilizar componentes y su eficiente gestión del DOM virtual, lo que mejora el rendimiento. Sin embargo, React tiene una curva de aprendizaje empinada y puede resultar complejo para los desarrolladores nuevos en su ecosistema.

Por otro lado, Express es un framework minimalista para Node.js que simplifica la creación de servidores web y API. Su flexibilidad y simplicidad lo convierten en una opción popular para desarrollar aplicaciones web rápidas y escalables. Las ventajas de usar Express incluyen su rapidez, facilidad de configuración y la gran cantidad de middleware disponible. No obstante, su minimalismo puede ser una desventaja en proyectos más grandes, donde puede ser necesario implementar características adicionales manualmente.

En conjunto, la combinación de React y Express proporciona una arquitectura robusta y escalable para el desarrollo de aplicaciones web modernas, aunque requiere una buena comprensión tanto del frontend como del backend para aprovechar al máximo sus beneficios.

Descripción del proyecto

Capa de presentación

La presentación del proyecto viene en el árbol de archivos en la parte web, donde se encuentra toda la parte de React. Principalmente, encontramos primero los módulos de node que se hayan instalado con el fin de agilizar y facilitar la creación de la aplicación (ya sean componentes ya creados previamente, librerías, etc) y la carpeta public, donde se guardan todos los archivos multimedia o que necesariten ser accedidos por la aplicación. Por último, src almacena la parte dinámica del frontend. Dentro de ella, la distribución de las carpetas viene dada por:

- components: Las piezas que se utilizan para construir las páginas, y que se definen de esta manera para poder ser reutilizadas cuando sean necesarias, o para que su mantenimiento sea más eficiente.
- pages: Los scripts donde se llama a los componentes que sean necesarios y se conforma la estructura que será renderizada y posteriormente mostrada al usuario en la aplicación.
- images: Las imágenes que necesiten ser utilizadas por la aplicación en el estilo, por ejemplo, no pueden ser guardadas en public, por eso se encuentran aquí.

- App.js: Es el script encargado de renderizar la página y generar y mostrar todas las páginas. En él se define la página de carga principal, los estilos que puedan ser utilizados y se realiza todo el enrutamiento para la carga de las páginas requeridas.

- config.dev.json: Es un archivo auxiliar, donde se definen características del entorno de desarrollo o la URL del servidor contra el que se vuelcan las peticiones de la aplicación (conexión con el backend).

El resto de archivos dados en esta carpeta son implementados de base por React, no se han llegado a editar o utilizar.

Entrando más profundamente, los componentes empleados son:

- Calendar: Es el componente que permite realizar la funcionalidad más importante del proyecto, dado que muestra un calendario con los días restringidos, en el que podemos hacer click y ver las fechas disponibles con las que cuente cada pista y realizar una reserva. Si el usuario no se encuentra logueado, será redireccionada a dicha página, y si lo está, se confirma y se realiza dicha reserva a nombre del usuario

logueado. Realiza dos peticiones al servidor, una que recibe todas las reservas almacenadas en la base de datos (GET) y otra para guardar la reserva (POST).

- Footer: Define el pie de página de todas las páginas que se muestran, con dos elementos informativos y dos enlaces relevantes.
- Header: Similar al footer, pero más dinámico. Un navbar que muestra un logo de la app y una serie de enlaces, los cuales todos redireccionan a su respectiva parte. Se muestra en todas la páginas al igual que el footer.
- Login: Es el encargado de verificar que el usuario introducido en un form sea validado en la base de datos, y actúa en consecuencia. Da la posibilidad de acceder al registro de usuarios. Una vez se haga un login correctamente, redirecciona a la página que muestra los datos del usuario.
- Register: Parecido a login, es un formulario donde se introducen los datos necesarios para registrar un nuevo usuario (tipo user) y valida que tanto el correo como el nombre de usuario no estén repetidos en la base de datos.
- Map: Elemento de apoyo que proporciona una funcionalidad interactiva al usuario, donde se muestra la zona en la que se ubica el centro. Se dan varias posibilidades, como elegir entre diferentes capas, ampliar, reducir, volver al inicio o desplegar el marcador.
- titleUpdater: Pequeño componente que utiliza el App.js para actualizar el título del navegador según donde se encuentre.

Por su parte, los scripts de las páginas llaman a los componentes que necesitan y componen la estructura principal de cada apartado de la página:

- about: Hace referencia a la página que habla del club que promociona y lo que desee la empresa que se muestre. Agrega también un componente map.
- Booking: Compone el elemento de las reservas con el header y el footer.
- Contact: Es el apartado en el que se pone un poco más de atención al cliente, se dan redes sociales y elementos que permitan la comunicación con el centro. Agrega un formulario que permite mandar un correo al centro.
- Legalwarning: Es una pequeña página que habla sobre datos legales que podría interesar agregar a la empresa.
- Main: Es el home de la aplicación, donde se muestra un eslogan o lo que le interese mostrar primero a la empresa. Cuenta con un fondo referido a la actividad vistoso que la diferencia de las demás.
- Rates: Página que muestra la tabla de datos importantes sobre tarifas y reservas de la empresa.
- Register/userlog: Compone la página para el elemento de registro/login.
- User: Muestra la información referida al usuario, como su username, nombre y reservas.

Capa lógica

En el otro lado de la aplicación, la parte del backend se ayuda de Sequelize para gestionar la base de datos, el cuál permite abstraer sentencias SQL a JS a la hora de trabajar con modelos de objetos, haciendo más legible y mantenible el código.

Sequelize, por defecto, implementa un sistema de archivos y carpetas en el que se gestionan todos los datos de la siguiente manera:

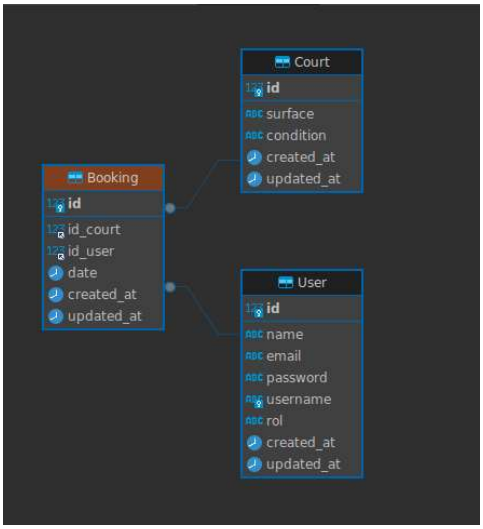
- Fixtures: Almacena los archivos en formato .json de los datos de prueba predefinidos que se cargan al lanzar la aplicación. Se hace referencia al modelo según el archivo.
- Models: Se definen la estructura de los tipos de objetos y sus respectivos datos usando sequelize, que los transmite luego a la base de datos.
- Routes: Almacena el archivo index.js, que se encarga de entrelazar las peticiones que hace el frontend en forma de ruta con la función del controlador respectivo.
- Controllers: Cada archivo de controller se encarga de crear la lógica de petición que puede ser realizada a la base de datos, tanto los parámetros que recibe como los que quiere buscar y el tipo de respuesta que puede dar.
- Index: Parte muy importante del traspaso de modelos. El archivo viene definido predeterminado por sequelize, pero es necesario agregar las relaciones existentes entre los modelos manualmente en este archivo. Se encarga de generar en la base de datos los objetos.

Middleware y migrations podrían ser utilizados más adelante si fuese necesario, por eso no son eliminados.

Server.js es el núcleo del backend, donde se hace el llamado a express, se aplica el CORS para que no existan problemas de peticiones CRUD, que se abra la ruta original y se tramitan las fixtures (los datos de prueba). Se aportan datos extra sobre la BDD y la configuración.

Contiene también al igual que el frontend los node modules, y en este caso un Dockerfile que define la versión de node especificada, el directorio de trabajo y ciertos comandos necesarios para desplegarse.

Por último, el archivo run-server.sh debe sr lanzado con la aplicación, ya que ejerce de servidor que recibe las peticiones, en el que se definen los datos necesarios de la base de datos y luego lanza un npx nodemon (entorno de desarrollo).



Capa de datos

La implementación de la capa de datos en este proyecto va completamente de la mano de la capa lógica, dado que el script que genera la base de datos de MySQL la genera el docker cuando se despliega,y todos los objetos, modelos, atributos y datos son generados por el backend y el docker.

Conclusión

Valoración Personal

Como conclusión como uno de los primeros proyectos personales llevados a cabo a esta escala, la satisfacción de haber conseguido en gran parte sacar adelante ideas que no pensé ser capaz de llevar a cabo ha sido ampliamente gratificante y alentador para mi futuro. Aunque siento que sigue siendo una aplicación bastante simple y con un amplio abanico de mejora y falta de implementación, como seguridad o legibilidad de código y funcionamiento, siento que cumple con mis expectativas.

Análisis DAFO y CAME

Análisis CAME

Corregir:

- **Documentación y Comentarios**: Mejorar la documentación y comentarios del código para facilitar el mantenimiento y la colaboración.
- **Pruebas Automatizadas**: Implementar pruebas automatizadas para asegurar la calidad y funcionalidad del código.

Afrontar:

- **Seguridad**: Reforzar las medidas de seguridad, como la protección contra ataques XSS y CSRF.
- **Rendimiento**: Optimizar consultas SQL y gestionar de manera más eficiente los recursos del servidor.

Mantener:

- **Modularidad del Código**: Continuar con la buena práctica de modularidad para facilitar la extensión y mantenimiento del proyecto.
- **Uso de Tecnologías Modernas**: Mantener el uso de React, Express y Sequelize, que ofrecen una base sólida y escalable.

Explotar:

- **Escalabilidad**: Aprovechar la arquitectura actual para escalar la aplicación y soportar más usuarios y funcionalidades.
- **Interfaz de Usuario**: Potenciar la interfaz responsiva y atractiva desarrollada con React y Bootstrap para mejorar la experiencia del usuario.

Análisis DAFO

Debilidades:

- **Falta de Pruebas**: Ausencia de pruebas automatizadas que aseguren la calidad y funcionalidad del código.
- **Documentación Insuficiente**: Necesidad de mejorar la documentación para facilitar la comprensión y mantenimiento del proyecto.

Amenazas:

- **Seguridad**: Riesgos potenciales de seguridad que podrían comprometer la integridad y privacidad de los datos.
- **Dependencia de Terceros**: Posibles problemas con servicios externos que afectan a la funcionalidad de la aplicación.

Fortalezas:

- **Tecnologías Modernas**: Uso de tecnologías actuales y robustas como React, Express y Sequelize.
- **Arquitectura Modular**: Diseño modular que facilita la extensión y el mantenimiento del proyecto.
- **Interfaz Responsiva**: UI atractiva y adaptable a diferentes dispositivos, mejorando la experiencia del usuario.

Oportunidades:

- **Escalabilidad**: Potencial para escalar la aplicación y añadir nuevas funcionalidades.
- **Adopción de Nuevas Tecnologías**: Incorporar nuevas tecnologías y prácticas que mejoren el rendimiento y la seguridad.
- **Expansión de Funcionalidades**: Añadir características adicionales que aumenten el valor y utilidad de la aplicación.

Futuro de la aplicación o posibles ampliaciones

Para el futuro de esta aplicación, siento que mejorar el sistema de seguridad, tokens y cookies es primordial. Por otro lado, la idea de implementar una tienda sería lógico y es lo que más me falta en este primer sprint de aplicación

Referencias y Fuentes bibliográficas

- https://www.freepik.es/vector-gratis/fondo-geometrico-degradado-tecnologia_18954486.htm#fromView=search&term=informatica&page=4&position=36&track=sph&regularType=vector
- Pirámide de población de Asturias 2023 | Datosmacro.com (expansion.com)
- https://es.wikipedia.org/wiki/PHP
- https://www.fiverr.com/hamzah_malik/fix-or-do-anything-in-html-css-javascript-bootstrap
- https://en.wikipedia.org/wiki/Spring_Boot
- https://hoplasoftware.com/mysql-sistema-de-gestion-de-bases-de-datos-relacionales/

- https://www.ine.es/
- Image by Freepik
- https://www.padelnest.com/clubs/principado-de-asturias/es-AS?map=true