Sportiva
Documentation of the sport event web application developed

This document explains from start to finish the development of the web application created for the Sportradar coding task.

1. Architecture Justification

The project is developed using React, which is a JavaScript library I have learned to use over the past few months and which has allowed me to create several personal applications. React was chosen for its ability to create reusable components, its versatile ecosystem, and its strong support.

Alongside React, TypeScript was used, which is a web language for making web pages interactive. TypeScript was chosen over JavaScript for its advantages such as static typing, autocompletion, and easier maintenance.

The project was created using Vite, a frontend build tool for web projects. Node.js was also used, which is a runtime environment that allows JavaScript to run outside the browser, on the server. The package manager for this environment is npm, which helps install dependencies and run the application on a local server.

Visual Studio Code was chosen as the code editor. Tailwind CSS was used for the design, which is a CSS framework that avoids writing long classes by using small, reusable ones.

Finally, for version control, GitHub was used with frequent commits during each development phase. GitHub is a platform for hosting and managing projects using Git, a version control system. My GitHub profile also contains other projects and applications I have developed, including this Sportradar coding task project:

https://github.com/mariosulee/Sportiva-FE-CodingTask.git

1. Project Start

The project began by creating a Vite project using the terminal command npm create vite@latest. The React framework and TypeScript language with the SWC compiler were selected, and npm dependencies were installed.

Next, the code editor was opened, in my case Visual Studio Code for its versatility. The project was linked to a GitHub repository to document the development process. Tailwind CSS was installed following the official documentation instructions.

The App component includes a <header>, <main>, and <footer> to structure the content. The Calendar.tsx component was then created in the components folder.

2. Creation of the Calendar Component

The Calendar.tsx component displays the current month in a grid layout. An array for the weekdays, an array for blank days, and an array for the days of the month were created. In the component's return, these arrays are iterated over with map and rendered.

To highlight the days with events in the calendar, the events.json file is imported as eventsData, and events for the current month and year are filtered. Using map, the event days are stored in the eventDays array, which is later rendered with a special condition.

3. Creation of the EventDetail Component

The next step was to complete the second task of the project. To display a detailed page for a clicked event, React Router is used, a library that handles navigation between different pages within a web application without reloading. The library is installed, the main component is wrapped with <BrowserRouter>, and <Routes> and <Route> are used in <main> to render each route.

The new EventDetail.tsx component is created. The local events state must be passed as global state in the main component and sent to other components via props. The types for Team and Event are also defined in types.tsx, assisted by AI to automatically generate each field from the provided JSON file due to its length.

The dynamic date parameter from the main component <Route> is used to filter events for that date. These events are stored in the dayEvents variable. If dayEvents is empty, a message shows that no events were found. Otherwise, it displays the teams, sport, season, time, stadium, and league, each accompanied by a React icon depending on the sport.

4. Creation of the EventForm Component

The EventForm.tsx component is a page containing a form to add a new event. A button in Calendar.tsx links to this page using <Link>, which allows navigation without reloading the page.

The form includes fields for team names, a dropdown for sport selection, and fields for date, time, stadium, and league. When the submit button is pressed, a new Event object is created and added to the global state, then displayed on the calendar. Local state is declared for each form field, and onChange handlers update the state as the user types.

In handleSubmit, validation ensures no fields are empty. A new SportEvent is created and added to the global state using setEvents, returning to the main component.

5. Ensuring Responsive Design

To ensure responsiveness, Google Chrome's Device Toolbar was used to test how the application behaves on mobile phones and tablets. Adjustments were made to day sizes in the calendar.

Centered texts and flexible spacing are prioritized. Text is small unless viewed on screens larger than 768px, handled with Tailwind CSS's md: prefix. Containers, buttons, text, headers, and more were adjusted accordingly.

6. Navigability and Local Storage

The home screen includes a visible button that takes the user to the event creation page, which also has a button to return to the calendar. Clicking a day on the calendar shows details of events for that day, or a message if none exist.

To save events in Local Storage, the project migrated to useReducer, making the state global and defining actions to add or delete events. Component props now use state.events and dispatch, improving modularity and extensibility.

In this reducer, initialState is initialized with localStorageEvents, which returns events from Local Storage if present, or example events from the JSON file if not.

In App.tsx, this global state is declared with the reducer, and a useEffect hook updates Local Storage whenever the events state changes, keeping the latest array of events. Added or deleted events persist on page reload.

7. Filtering by Sport Category

To filter events in Calendar by sport, a local state selectedSport is added using useState. A variable filters events if selectedSport has a value. The calendar now displays filteredEvents instead of the entire events state, showing only events for the selected sport.

(Spanish Version)

El presente documento explicará de principio a fin el desarrollo de la aplicación web que se ha desarrollado para la coding task de Sportradar.

0. Justificación de arquitectura

El proyecto a desarrollar se hará con **React**, que es una biblioteca de JavaScript que he aprendido a utilizar en estos últimos meses y que me ha permitido crear varias aplicaciones propias. Se ha elegido React por su capacidad de utilizar componentes reutilizables, tiene un ecosistema con muchas posibilidades, y un gran soporte.

Junto a React se ha utilizado **Typescript**, que es un lenguaje web para hacer interactivas las páginas web. Se ha usado TypeScript en vez de JavaScript por sus ventajas como el tipado estático, el autocompletado, o el mejor mantenimiento.

El proyecto se ha creado usando **Vite**, que es una herramienta de construcción de frontend para proyectos web. Se ha usado también **Node.js** que es un entorno de ejecución que permite correr el lenguaje de programación que se usa (JavaScript) fuera del navegador, haciéndolo correr en el servidor. El gestor de paquetes de este entorno es **npm** y hace de herramienta para instalar dependencias y hacer correr la aplicación en un servidor local.

Como editor de código se ha elegido **Visual Studio Code**. Para el diseño de la aplicación se ha usado **TailWind CSS**, que es un framework de CSS que permite evitar escribir clases largas a cambio de otras pequeñas y reutilizables.

Finalmente, para el control de versiones, se ha usado **GitHub** y se han ido haciendo commits eventuales por cada fase de desarrollo. GitHub es una plataforma para alojar y gestionar fácilmente proyectos usando Git, que es un sistema de control de versiones. En mi perfil de GitHub se pueden ver también otros proyectos y aplicaciones que he desarrollado, así como el proyecto para esta coding task:

https://github.com/mariosulee/Sportiva-FE-CodingTask.git

1. Comienzo del proyecto

Se ha comenzado creando un proyecto en Vite usando el comando en la terminal del ordenador **npm create vite@latest**. Se elige entonces el framework de React y el lenguaje TypeScript con el compilador SWC y la instalación de dependencias de npm.

A continuación, se abre un editor de código fuente, en mi caso he escogido Visual Studio Code por su polivalencia. Se abre el proyecto y se vincula al repositorio de GitHub que se debe crear para ir documentando el desarrollo de la aplicación. Seguidamente, se instala TailWind CSS con las instrucciones que se aportan en la página oficial de documentación.

El componente App incluirá un <header> , un <main>, y un <footer> para estructurar el contenido. Entonces, se crea el componente en la carpeta components de **Calendar.tsx.**

2. Creación del componente Calendar

Este componente llamado **Calendar.tsx** mostrará el calendario del mes actual en un display de grid. Para ello se crea un array que contiene los días de la semana, otro de los días en blanco del calendario, y otro de los días del mes. Dentro del return del componente se recorren estos componentes con el método map y se van renderizando.

Para mostrar de distinto color en el calendario aquellos días que hayan eventos, se importa el archivo events.json bajo el nombre eventsData, y se filtran los eventos del mes y año actual en el que se está. Con el método map se van metiendo esos días del mes en el array eventDays, que posteriormente se recorre y renderiza con una condición especial.

3. Creación del componente EventDetail

A continuación, se dispone a completar la segunda tarea del proyecto. Para mostrar una página detallada del evento en el que se clicke, se puede utilizar **React Router**, que es una librería de React que sirve para manejar la navegación entre distintas páginas dentro de una aplicación web sin recargar. Primeramente se instala la librería, para luego modificar el componente principal envolviendo lo renderizado en la etiqueta <BrowserRouter> y en el main renderizar cada ruta con <Routes> y <Route>.

Se crea entonces el nuevo componente **EventDetail.tsx** . Antes de ello se debe pasar el estado local **events** como un estado global en el componente principal y pasarlo al resto de componentes vía props. También se definen los tipos en types.tsx de Team y Event, CON INTELIGENCIA ARTIFICIAL HABIENDOLE PASADO EL ARCHIVO JSON PROPORCIONADO DE FORMA QUE AUTOMATIZASE TODA LA ESCRITURA DE CADA CAMPO, YA QUE ES EXTENSA.

Se usa el parámetro dinámico date de cuando se uso en el componente principal dentro de <Route> y se filtran los eventos de ese parámetro date. Estos eventos se guardan en la variable dayEvents, y si esta vale 0, se mostrará en pantalla un mensaje de que no se han encontrado eventos para esa fecha. En caso contrario, mostrará los equipos enfrentados, el

deporte, la temporada, la hora, el estadio en el que jugarán, y la liga. Todo ello acompañado de un icono de React distinto dependiendo del deporte del que se trate el evento.

4. Creación del componente EventForm

Se crea el componente **EventForm.tsx** que será una página de la web que contenga un formulario a rellenar para añadir un nuevo evento. Se incluye un botón en Calendar.tsx para que nos lleve a esta página. Esto se hace con <Link>, que es un componente para navegar entre páginas (rutas) dentro de tu aplicación sin recargar la página completa.

El formulario incluye campos para los nombres de los equipos, una lista de selección para los posibles deportes, y campos para rellenar de fecha, hora, estadio, y liga. Se requiere que una vez se pulse el botón de submit, que se cree un nuevo objeto de tipo Event, se añada al estado global declarado en el componente principal, y luego se añada al calendario. Para ello, se declara un estado local para cada uno de los campos a rellenar en el formulario, y se usa el parámetro onChange junto con su función setX para cambiar estos estados locales a lo que se escribe.

Posteriormente en la función handleSubmit se alerta de que alguno de los estados locales estén vacíos para tener todos los campos rellenos, y se crea un nuevo evento de tipo SportEvent con estos estados locales. Finalmente este nuevo evento se añade al estado global con setEvents y se vuelve al componente principal.

5. Aseguración del diseño responsivo

Para esta cuarta tarea del coding task, se ha procedido a utilizar la funcionalidad de Google Chrome de Device ToolBar, con el objetivo de examinar cómo se comportaría la aplicación en teléfonos móviles y dispositivos tablets. Habiendo hecho una examinación, se procede a modificar parámetros como el tamaño de los días en el calendario.

Se deben priorizar textos centrados y espacios que no sean fijos. Los textos deben de ser pequeños a menos que se vean en pantallas superiores a 768px, esto se hace con md: en TailWind CSS. Se ajustan de esta forma contenedores, botones, textos, headers..etc.

6. Navegabilidad y Local Storage

Respecto a la navegabilidad, en la pantalla de inicio de la aplicación se encuentra un botón visible que lleva al usuario a la ventana de creación del evento, en la cual también hay un botón para volver al calendario. Si se hace click en algún día del calendario, se mostrarán los detalles de los eventos en ese día, y en caso de que no haya, se muestra por pantalla.

Para guardar los eventos introducidos en Local Storage, primero, se opta por migrar el proyecto a un useReducer, cambiando así el estado global y declarando un set de acciones para borrar y añadir un evento al calendario. Cambiarán entonces las props que se les pasa

a los componentes, usando state.events y dispatch. Esto perfecciona el proyecto y lo hace más modular y extensible.

En este reducer el initialState de events se inicializa a la función localStorageEvents, la cual hace que si hay eventos en el localStorage, los devuelve, y si no, devuelve los eventos de ejemplo que hay en el json.

En App.tsx se declara entonces este estado global con el reducer y se usa un hook useEffect que haga que cada vez que cambie el estado de events, actualice automáticamente el contenido del localStorage, guardando la versión más reciente del array de eventos. De esta forma, los eventos añadidos o eliminados persisten al recargar la aplicación.

7. Filtrado por categoría de deporte

Para mostrar un filtrado en el componente Calendar y poder mostrar en el calendario solo los eventos del deporte seleccionado, se añade un state local con useState llamado **selectedSport**, que guarde el deporte seleccionado. Se declara una variable que filtre los eventos si hay algún valor en el estado selectedSport, y se cambia la forma en la que se ha mostrado los eventos en el calendario, en vez de mostrando el estado events, mostrando lo que contenga la variable filteredEvents.