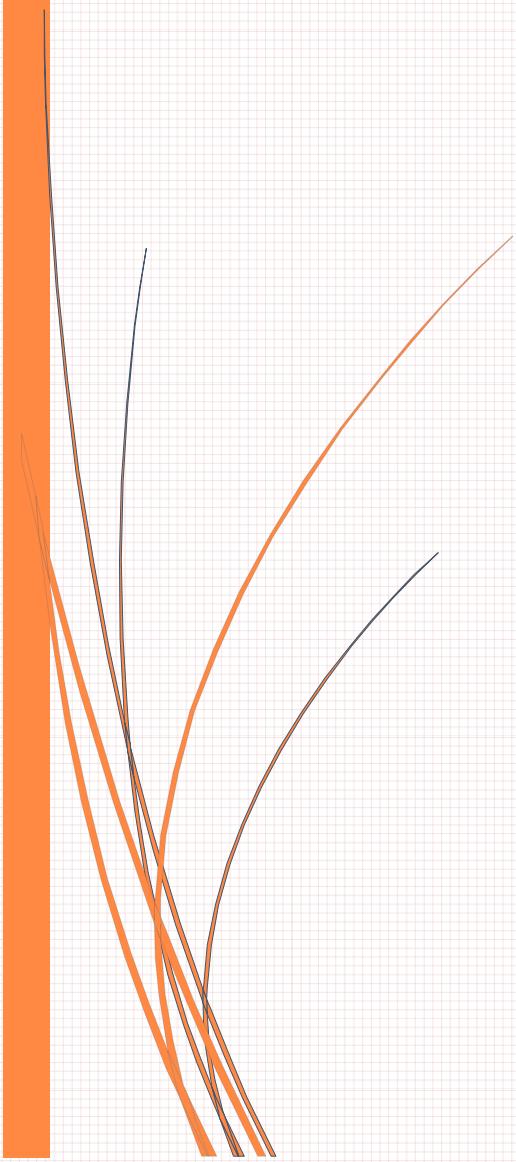
2023

Valorant

Primeros Pasos con React



Angie Garcia

Introducción

Contexto

Este proyecto de React es una aplicación web que muestra información sobre los agentes del juego "Valorant". Los agentes son personajes jugables en el juego, y la aplicación muestra sus nombres, roles y avatares. Además, la aplicación permite filtrar los agentes por rol y buscar agentes por nombre. Al seleccionar un agente, la aplicación muestra una vista detallada con más información sobre ese agente específico.

Es un proyecto sencillo para dar los primeros pasos con React, que he realizado con fines educativos.

Requisitos de Funcionalidad

Para este proyecto he tenido en cuenta los siguientes requisitos de funcionalidad:

- **Mostrar la lista de agentes:** La aplicación debe obtener datos de la API "https://valorant-api.com/v1/agents" al cargar y mostrar una lista de agentes en la página principal.

- **Filtrar por Rol:** La aplicación debe permitir al usuario filtrar la lista de agentes por su rol. Los roles disponibles son "Initiator", "Sentinel", "Duelist" y "Controller". Se debe proporcionar una opción "All Role" para mostrar todos los agentes sin filtrar por rol.

- **Búsqueda por Nombre:** La aplicación debe proporcionar una barra de búsqueda donde el usuario pueda ingresar el nombre de un agente. Al presionar el botón de búsqueda, la lista de agentes debe filtrarse y mostrar solo aquellos cuyos nombres coinciden (de forma parcial o completa) con el texto de búsqueda ingresado. Si no se encuentran resultados, se debe mostrar un mensaje de alerta indicando que el agente no se encontró.

- **Vista Detallada del Agente:** Al hacer clic en un agente de la lista, la aplicación debe mostrar una vista detallada con más información sobre ese agente específico. La vista detallada debe incluir el nombre del agente, su imagen/avatar, y su rol.

- **Cerrar Agentes:** En la vista detallada del agente, debe haber un botón "x" para cerrar la vista detallada y volver a la lista de agentes.

- **Estilo y Diseño:** La aplicación debe tener un diseño atractivo y amigable para el usuario. Se pueden utilizar gradientes de fondo en las tarjetas de los agentes para hacerlas más llamativas.

- **Navegación:** La aplicación debe utilizar la biblioteca de enrutamiento "react-router-dom" para permitir la navegación entre la página principal y la vista detallada del agente.

Nota: Es importante asegurarse de que la API "https://valorant-api.com/v1/agents" esté accesible y proporcionando datos adecuados. Además, se recomienda agregar manejadores de errores para manejar posibles fallos en la obtención de datos desde la API.

Angie Garcia

Instalación

Para probar este proyecto localmente, sigue los siguientes pasos:

1. Clona este repositorio en tu máquina local utilizando el siguiente comando:

   ```
   git clone https://github.com/angiegxg/valorant.git
   ```

2. Ve al directorio del proyecto

   ```
   cd valorant
   ```

3. Instala las dependencias utilizando npm:

   ```
   npm install
   ```

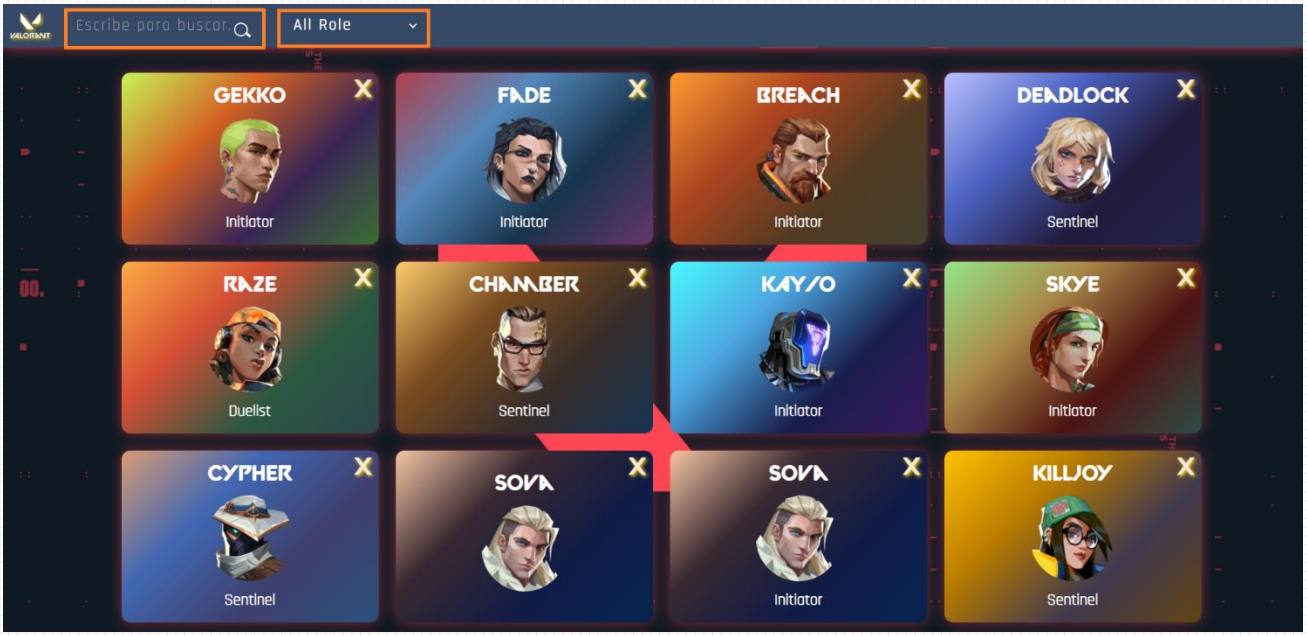
4. Inicia la aplicación

   ```
   npm start
   ```

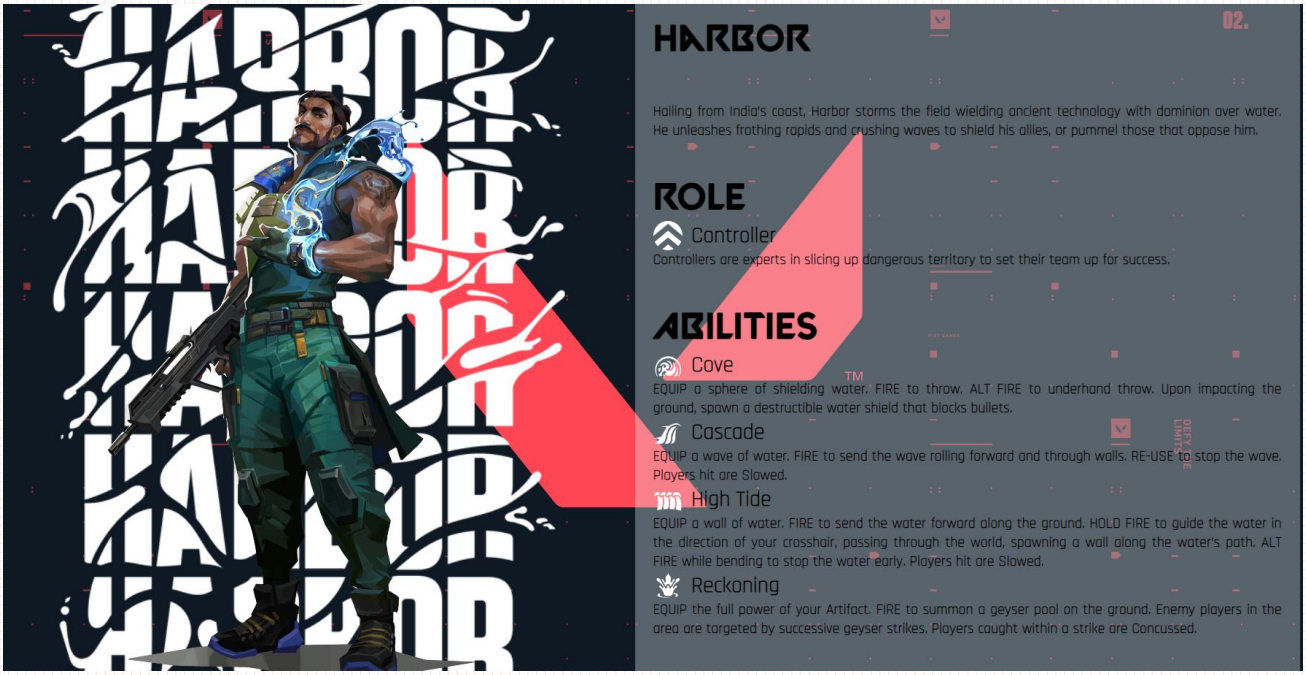
5. Abre tu navegador web y navega a `http://localhost:3000` para ver la aplicación en acción.

Uso

Una vez que la aplicación esté en funcionamiento, podrás explorar la lista completa de agentes de Valorant. Podrás buscar agentes por nombre utilizando la barra de búsqueda y filtrar la lista por rol utilizando el filtro desplegable.



Haz clic en una tarjeta de agente para ver una vista detallada de la información del agente, incluida su imagen, rol, descripción y habilidades.



Documentación

index.js:

- **Descripción:** Este archivo es el punto de entrada de la aplicación React. Aquí se inicializa el enrutador de la aplicación utilizando BrowserRouter de react-router-dom y se renderiza el componente principal App dentro de un componente StrictMode. StrictMode ayuda a identificar posibles problemas en la aplicación y advierte sobre el uso de prácticas desaconsejadas.
- **Implementación:** Se importa React y ReactDOM para la renderización de la aplicación, además de los estilos y el componente App. También se importa BrowserRouter para habilitar el enrutamiento de la aplicación. Luego, se crea el nodo raíz de la aplicación utilizando ReactDOM.createRoot y se renderiza el componente App dentro de React.StrictMode y BrowserRouter.

App.js

- **Descripción:** En este archivo, se define el componente principal App, que actúa como el contenedor de la aplicación. Aquí se configuran las rutas y se asocian con los componentes Home y Detail. El componente App es responsable de manejar la navegación entre estas dos páginas.
- **Implementación:** Se importa React y los estilos necesarios. Además, se importan los componentes Home y Detail, que representan las páginas principales y de detalle respectivamente. El componente App contiene un componente Routes, que define las rutas y sus componentes asociados. En este caso, la ruta '/' está asociada con el componente Home, y la ruta '/detail/:uuid' está asociada con el componente Detail.

Angie Garcia

Home.js

- **Descripción:** El componente Home representa la página principal de la aplicación, donde se muestran las tarjetas de los agentes. Utiliza el estado local para mantener la lista de agentes, el rol seleccionado y la copia original de la lista de agentes para facilitar las operaciones de filtrado y búsqueda.
- **Implementación:** El componente Home importa los componentes Nav y CardList, que son responsables de mostrar la barra de navegación y la lista de tarjetas de agentes, respectivamente. Cuando el componente se monta, se utiliza la función useEffect para hacer una llamada a la API "https://valorant-api.com/v1/agents" y obtener los datos de los agentes. Estos datos se almacenan en el estado local characters y también se guarda una copia en originalCharacters para facilitar la búsqueda. Las funciones searchHandler, handleRoleChange y closeHandler se utilizan para filtrar la lista de agentes según el texto de búsqueda, el rol seleccionado y para cerrar un agente específico.

Nav.js

- **Descripción:** El componente Nav representa la barra de navegación de la aplicación, que contiene los componentes Search y FilterRole. Estos componentes son responsables de la funcionalidad de búsqueda y filtrado por rol de los agentes.
- **Implementación:** El componente Nav importa los componentes Search y FilterRole y los renderiza dentro de un contenedor. También contiene el logotipo del juego "Valorant". El componente Nav recibe las funciones onSearch y onRoleChange como props y las pasa a los componentes hijos Search y FilterRole para que puedan manejar la búsqueda y el cambio de rol, respectivamente.

FilterRole.js

- **Descripción:** El componente FilterRole representa el filtro desplegable que permite al usuario seleccionar un rol específico para filtrar la lista de agentes. Los roles disponibles son "All Role", "Initiator", "Sentinel", "Duelist" y "Controller".
- **Implementación:** El componente FilterRole utiliza el estado local para mostrar las opciones de roles disponibles y manejar el cambio de rol seleccionado. Cuando se selecciona un rol, se llama a la función onRoleChange pasada como prop para comunicar el cambio al componente Home y realizar el filtrado adecuado.

Search.js

- **Descripción:** El componente Search representa la barra de búsqueda que permite al usuario buscar agentes por nombre. Cuando el usuario ingresa texto en la barra de búsqueda y presiona el botón de búsqueda, se ejecuta la función onSearch pasada como prop para realizar la búsqueda y filtrar la lista de agentes según el texto ingresado.
- **Implementación:** El componente Search utiliza el estado local para mantener el texto de búsqueda. Cuando el usuario ingresa texto en la barra de búsqueda, se actualiza el estado y el valor se pasa a la función onSearch cuando se presiona el botón de búsqueda.

CardList.js

- **Descripción:** El componente CardList representa la lista de tarjetas de agentes en la página principal. Recibe la lista de agentes, el rol seleccionado y la función closeHandler como props y filtra los agentes según el rol seleccionado antes de mostrar la lista de tarjetas.
- **Implementación:** El componente CardList recibe la lista de agentes y el rol seleccionado como props. Luego, filtra la lista de agentes según el rol seleccionado y muestra las tarjetas de los agentes restantes utilizando el componente Card.

Card.js

- **Descripción:** El componente Card representa una tarjeta de agente en la lista de agentes. Cada tarjeta muestra el nombre del agente, su avatar y su rol. También tiene un botón "x" para cerrar la tarjeta y eliminar el agente de la lista.
- **Implementación:** El componente Card recibe información sobre el agente y la función closeHandler como props. Cuando el usuario hace clic en el botón "x", se llama a la función closeHandler para cerrar la tarjeta y eliminar el agente de la lista. Cuando el usuario hace clic en una tarjeta, se navega a la vista detallada del agente utilizando useNavigate de react-router-dom.

Detail.js

- **Descripción:** El componente Detail representa la vista detallada de un agente específico. Se utiliza la información del agente con el identificador único uuid proporcionado en la URL para realizar una llamada a la API "https://valorant-api.com/v1/agents/{uuid}" y obtener los detalles completos del agente. Luego, se muestra la imagen del agente, su rol, descripción y habilidades asociadas en una presentación detallada.

- **Implementación:** El componente Detail importa useEffect, useState, useParams y useNavigate de react-router-dom. Utiliza useParams para obtener el uuid proporcionado en la URL, que se utiliza para realizar una llamada a la API y obtener los detalles completos del agente. El estado local detail se inicializa con un objeto que tiene una propiedad abilities para almacenar la lista de habilidades del agente. El efecto de useEffect se utiliza para realizar la llamada a la API y actualizar el estado local detail con la información del agente. La función goBack utiliza useNavigate para regresar a la página anterior cuando se hace clic en el botón "back".

Flujo de la Aplicación

1. Al cargar la aplicación, **index.js** es el punto de entrada, y se inicializa el enrutador utilizando **BrowserRouter** de **react-router-dom**. Luego, se renderiza el componente App dentro de un componente **StrictMode**.

2. En **App.js**, se define el componente principal App, que actúa como el contenedor de la aplicación. Aquí se configuran las rutas y se asocian con los componentes **Home** y **Detail**. La ruta **'/'** está asociada con el componente Home, que representa la página principal de la aplicación. La ruta **'/detail/:uuid'** está asociada con el componente **Detail,** que representa la vista detallada de un agente específico.

Angie Garcia

3. En **Home.js**, el componente Home se monta y utiliza el estado local para mantener la lista de agentes, el rol seleccionado y la copia original de la lista de agentes. Se realiza una llamada a la API **"https://valorant-api.com/v1/agents"** para obtener los datos de los agentes al cargar la página. Estos datos se almacenan en **characters** y **originalCharacters.**

4. **Nav.js** muestra la barra de navegación en la parte superior de la página principal. La barra de navegación contiene dos componentes hijos: **Search** y **FilterRole**. **Search** es una barra de búsqueda donde el usuario puede ingresar el nombre de un agente para buscarlo, y **FilterRole** es un filtro desplegable que permite al usuario seleccionar un rol específico para filtrar la lista de agentes.

5. Cuando el usuario interactúa con la barra de búsqueda (Search), el componente **Search** maneja el cambio de texto de búsqueda. Al hacer clic en el botón de búsqueda, el componente **Search** invoca la función **onSearch** pasada como prop, y esta función se ejecuta en el componente **Home**. La función **onSearch** en Home filtra la lista de agentes según el texto de búsqueda y actualiza el estado local **characters**.

6. Cuando el usuario selecciona un rol en el filtro desplegable (FilterRole), el componente **FilterRole** maneja el cambio de rol seleccionado. Al cambiar el rol, se invoca la función **onRoleChange** pasada como prop, y esta función se ejecuta en el componente **Home**. La función **onRoleChange** en **Home** actualiza el estado local **selectedRole**, lo que provoca que la lista de agentes se filtre por el rol seleccionado.

7. **CardList.js** recibe la lista de agentes y el rol seleccionado como props. El componente filtra los agentes según el rol seleccionado y muestra la lista de tarjetas de agentes restantes utilizando el componente **Card.**

8. Cuando el usuario hace clic en una tarjeta de agente (Card), se navega a la vista detallada del agente (Detail) utilizando **useNavigate** de **react-router-dom.**

9. En **Detail.js**, se muestra la vista detallada del agente seleccionado utilizando el **uuid** proporcionado en la URL. El componente realiza una llamada a la API **"https://valorant-api.com/v1/agents/{uuid}"** para obtener los detalles completos del agente. Luego, se muestra la imagen del agente, su rol, descripción y habilidades asociadas en una presentación detallada.

10. En la vista detallada, el usuario puede regresar a la página anterior haciendo clic en el botón "**back**", lo que utiliza **useNavigate** para realizar esta acción.

11. El flujo de la aplicación vuelve al paso 4, y el usuario puede seguir interactuando con la barra de búsqueda y el filtro por rol para continuar explorando los agentes disponibles.