# Pokémon Casino WebApp

## Requirements

1. Node JS.
2. Pokedex API (included in project).
3. XAMPP v3.3.0^
4. Pokedex database (included in project).

## How to deploy on development environment

1. Install all the required dependencies:

```
npm install
```

2. Run the development server:

```
npm start
```

3. Install XAMPP.
4. Start Apache Server and MySQL Server.
5. Import database in MySQL.
6. Move pokedex-api folder to /xampp/htdocs.

# App.js

```
import './App.css';
import Page from './components/page';

function App() {
  return (
    <div className="App">
      <Page/>
    </div>
  );
}

export default App;
```

This is a React component named “App” which is the main component.

The line import './App.css'; imports a CSS file called "App.css". This is used to apply specific styles to the "App" component. You can mention that it is a common practice in React to import CSS files to style components.

The line import Page from './components/page'; imports a component named "Page" from a file named "page.js" located in the "components" folder. This allows “Page” component to change it´s content in component “App”. Inside the App() function, a React element is returned that contains other components and HTML elements. In this case, a  is returned with the class "App" and the component "Page" is included within it.

The last line export default App; exports the "App" component so that it can be used in other files.

## Page.jsx

```jsx
import React from 'react';
import { BrowserRouter, Link } from 'react-router-dom';
import Content from './content/content';

/**
 * Main content of the app
 * @returns All the content of the page
 */
function Page() {
  const isSessionActive = localStorage.getItem('session');

  return (
    <BrowserRouter>
      <nav className="navbar navbar-expand-sm navbar-dark barra">
        <Link className="navbar-brand" id="linklogo" to="/">
          <img src='./img/logo/logo.png' alt='Logo'/>
        </Link>
        <div className="collapse navbar-collapse" id="links-navbar">
          <ul className="navbar-nav ms-auto mt-2 mt-lg-0">
            {/* Navbar changes when user is logged */}
            {isSessionActive ? (
              <>
                <li className="nav-item d-flex my-2 my-lg-0">
                  <Link className="nav-link" to="/pokedex">
                    Pokédex
                  </Link>
                </li>
                <li className="nav-item d-flex my-2 my-lg-0">
                  <Link className="nav-link" to="/logout">
                    Cerrar sesión
                  </Link>
                </li>
              </>
            ) : (
              <>
                <li className="nav-item d-flex my-2 my-lg-0">
                  <Link className="nav-link" to="/login">
                    Iniciar sesión
                  </Link>
                </li>
                <li className="nav-item d-flex my-2 my-lg-0">
                  <Link className="nav-link" to="/register">
                    Registrarse
                  </Link>
```

```
                </li>
              </>
            )}
          </ul>
        </div>
      </nav>
      <main>
        <Content />
      </main>

    </BrowserRouter>
  );
}

export default Page;
```

The React component “Page” contains the casino App content. It displays a navigation bar that changes when user is logged. Here's an explanation line by line:

The first line, import React from 'react';, imports the React library required to work with React components.

The line import { BrowserRouter, Link } from 'react-router-dom'; imports the router (BrowserRouter) and link (Link) components from the react-router-dom library.

The line import Content from './content/content'; imports the "Content" component from a file named "content.js" located in the "content" folder.

The Page() function is the React component.

Inside the Page() function, a constant isSessionActive is declared, which retrieves the session state from the local storage.

Next, the UI structure is defined using React JSX elements. The BrowserRouter is used to wrap the content and enable navigation between the application's routes.

Inside the nav element, a navigation bar is defined with a navigation brand (Link) that displays a logo.

A conditional structure (<>) is used to display different navigation elements based on the session state. If the session is active, links to "Pokédex" and "Logout" are displayed. Otherwise, links to "Login" and "Register" are displayed.

After the navigation bar, the main element is present, which contains the "Content" component.

Finally, the "Page" component is exported so that it can be used in other files.

## Content.jsx

```
import React from 'react';
import { Route, Routes } from 'react-router-dom';
import Home from '../routes/home';
import Login from '../routes/login';
import Register from '../routes/register';
import Logout from '../routes/logout';
import Pokedex from '../routes/pokedex';

/**
 * Content changes dynamically depending on the path
 * @returns Content component
 */
function Content() {
    return (
        <Routes>
            <Route path="/" element={
                <div id="contenido">
                    <Home />
                </div>
            } />
            <Route path="/login" element={
                <div id="formulario">
                    <Login />
                </div>
            } />
            <Route path="/register" element={
                <div id="formulario">
                    <Register />
                </div>
            } />
```

```
            <Route path="/pokedex" element={
                <div id="contenido">
                    <Pokedex />
                </div>
            } />
            <Route path="/logout" element={
                <div id="contenido">
                    <Logout />
                </div>
            } />
        </Routes>
    );
}

export default Content;
```

“Content” component displays different components depending on the path.

The line import { Route, Routes } from 'react-router-dom'; imports the Route and Routes components from the react-router-dom library. Those components are used to change content between defined routes.

The Content() function is the React component that represents the content área of the application.

Inside the Content() function, the UI structure is defined using React JSX elements. The Routes component wraps multiple Route components.

Each Route component represents a specific route path and its corresponding element/component to render. The path prop specifies the URL path, and the element prop contains the JSX element/component to render when the path matches.

For example, when the path is '/', the Home component is rendered inside a div element with the id 'contenido'. Similarly, different routes such as /login, /register, /pokedex, and /logout render their respective components within different div elements.

Finally, the "Content" component is exported so that it can be used in other files.

# Home.jsx

```jsx
import React, { useEffect } from 'react';

function Home() {
  return (
    <div>
      <h2>Inicio</h2>
      <p>Bienvenido a la página de inicio.</p>
    </div>
  );
}

export default Home;
```

This code represents the "Home" component of a React application, which serves as the home page.

The line import React, { useEffect } from 'react'; imports the React library and the useEffect hook. You can mention that React is necessary to work with React components, and useEffect is a hook used for handling side effects in functional components.

The Home() function is the React component. It displays a welcome screen.

Finally, the "Home" component is exported so that it can be used in other files.

## Login.jsx

```
import React, { useRef, useEffect } from 'react';
import { validateLogin } from '../content/validations/login';
/**
 * Displays Login form
 * @returns Login form
 */
function Login() {
  const formRef = useRef(null);
/**
 * When clicked on submit it goes to login validation
 */
  useEffect(() => {
    if (formRef.current) {
      validateLogin();
    }
  }, []);

  return (
    <div className="contenido-formulario">
      <h2 className="titulo">Iniciar sesión</h2>
      <form id="login" ref={formRef}>
        <div class="mb-4">
          <label class="form-label">Usuario</label>
          <input
            type="text"
            class="form-control"
            name="usuario"
            id="usuario"
            aria-describedby="helpId"
            placeholder="Nombre de usuario"
            minLength={4}
            required
          />
          <p id="error-usuario" class="text-red-600 text-xs"></p>
        </div>
        <div class="mb-4">
          <label for="" class="form-label">Contraseña</label>
          <input
            type="password"
            class="form-control"
            name="contraseña"
            id="contraseña"
            aria-describedby="helpId"
            placeholder="Contraseña"
```

```
            minLength={6}
            required
          />
          <p id="error-contraseña" class="text-red-600 text-xs"></p>
        </div>
        <p id="error-servidor" class="text-red-600"></p>
        <button id="submit" type="submit" class="btn btn-
primary">Enviar</button>
      </form>
    </div>
  );

}

export default Login;
```

This code represents the "Login" component of a React application, which displays a login form.

The line import React, { useRef, useEffect } from 'react'; imports the React library and the useRef and useEffect hooks. useRef and useEffect are hooks used for managing references and side effects in functional components, respectively.

The line import { validateLogin } from '../content/validations/login'; imports a function named validateLogin from a file that handles login validation.

Inside the Login() function, a formRef is created using the useRef hook. This reference is used to access the form element in the useEffect hook.

The useEffect hook is used to execute the validateLogin function when the component is mounted. The validation is triggered when the submit button is clicked. The dependency array [ ] ensures that the effect runs only once.

The JSX code represents the structure of the login form. Various form elements such as input and label are used to collect user input.

The className attribute is used instead of class to define CSS classes for styling purposes.

Error messages are displayed using p elements with specific id attributes. These messages are conditionally rendered based on the validation results.

The form also includes a server error message and a submit button.

Finally, the "Login" component is exported so that it can be used in other files.

## Logout.jsx

```
function Logout() {
  localStorage.removeItem('session');
  window.location.href = '/';
}

export default Logout;
```

The Logout function is a simple function that performs two actions: it removes the session item from the local storage using localStorage.removeItem('session'), and it redirects the user to the home page by setting window.location.href to '/'.

## Pokedex.jsx

```
import { PHPController as Controller } from
"../content/validations/controller/controller.js";
import { useEffect, useState } from "react";
/**
 * It displays current pokemon data in database
 * @returns Pokemon list from API
 */
function Pokedex() {
  const [pokemonList, setPokemonList] = useState([]);

  useEffect(() => {
    getPokemonList();
  }, []);
/**
 * Gets pokemon list from API
 */
```

```
  async function getPokemonList() {
    try {
      const response = await Controller.listPokemon();
      setPokemonList(response.data);
    } catch (error) {
      console.error(error.message);
    }
  }
/**
 * Displays pokemon list from API
 * @returns HTML table row
 */
  function showPokemonList() {
    return pokemonList.map((pokemon) => (
      <tr key={pokemon.id}>
        <td>{pokemon.id}</td>
        <td>
          <img src={pokemon.url_imagen} alt={pokemon.nombre} />
        </td>
        <td>{pokemon.nombre}</td>
        <td>{getTypeImage(pokemon)}</td>
        <td>{pokemon.PS}</td>
        <td>{pokemon.ATK}</td>
        <td>{pokemon.DEF}</td>
        <td>{pokemon.VEL}</td>
        <td>{pokemon.ATK_ESP}</td>
        <td>{pokemon.DEF_ESP}</td>
      </tr>
    ));
  }
/**
 * Gets an image based on pokemon type
 * @param {pokemon} pokemon
 * @returns HTML img tag
 */
  function getTypeImage(pokemon) {
    let url = "img/tipos/";
    if (pokemon.tipo_2) {
      return (
        <>
       <img src={url +
pokemon.tipo_1.toLowerCase().normalize('NFD').replace(/[\u0300-\u036f]/g,
"") + '.png'} alt={pokemon.tipo_1} />
       <img src={url +
pokemon.tipo_2.toLowerCase().normalize('NFD').replace(/[\u0300-\u036f]/g,
"") + '.png'} alt={pokemon.tipo_2} />
        </>
      );
    } else {
```

```
      return <img src={url +
pokemon.tipo_1.toLowerCase().normalize('NFD').replace(/[\u0300-\u036f]/g,
"") + '.png'} alt={pokemon.tipo_1} />
    }
  }

  return (
    <div id="tabla">
      <table id="pokedex">
        <thead>
          <tr>
            <th>#</th>
            <th></th>
            <th>Pokémon</th>
            <th>Tipos</th>
            <th>PS</th>
            <th>ATQ</th>
            <th>DEF</th>
            <th>VEL</th>
            <th>ATQ ESP</th>
            <th>DEF ESP</th>
          </tr>
        </thead>
        <tbody>{showPokemonList()}</tbody>
      </table>
    </div>
  );
}

export default Pokedex;
```

This code represents a Pokedex component that displays the Pokémon data fetched from an API.

The component makes use of the useState and useEffect hooks from React to manage the state and perform side effects.

The pokemonList state variable is initialized using useState([]) and will store the Pokémon data retrieved from the API.

The useEffect hook is used to fetch the Pokémon list when the component mounts. It calls the getPokemonList function, which is an asynchronous function that retrieves the data

from the API using the Controller.listPokemon method. The fetched data is then set in the pokemonList state using setPokemonList(response.data).

The showPokemonList function maps over the pokemonList array and generates the HTML table rows (<tr>) with the Pokémon data.

The getTypeImage function retrieves the appropriate image based on the Pokémon's type. It generates the HTML <img> tags with the necessary attributes.

Finally, the Pokedex component returns the rendered HTML table (<table>) with the Pokémon data displayed in rows. The showPokemonList function is invoked inside the <tbody> element to generate the table rows dynamically.

## Pokedex.jsx

```
import React, { useRef, useEffect } from 'react';
import { validateRegister } from '../content/validations/register';
/**
 * Displays register form
 * @returns Register form
 */
function Register() {
  const formRef = useRef(null);
/**
 * When clicked on submit it goes to register validation
 */
  useEffect(() => {
    if (formRef.current) {
      validateRegister()
    }
  }, []);

  return (
    <div className="contenido-formulario">
      <h2 className="titulo">Registrarse</h2>
      <form id="register" ref={formRef}>
        <div class="mb-4">
          <label class="form-label">Usuario</label>
          <input
```

```
            type="text"
            class="form-control"
            name="usuario"
            id="usuario"
            aria-describedby="helpId"
            placeholder="Nombre de usuario"
            minLength={4}
            required
          />
          <p id="error-usuario" class="text-red-600 text-xs"></p>
        </div>
        <div class="mb-4">
          <label for="" class="form-label">Contraseña</label>
          <input
            type="password"
            class="form-control"
            name="contraseña"
            id="contraseña"
            aria-describedby="helpId"
            placeholder="Contraseña"
            minLength={6}
            required
          />
          <p id="error-contraseña" class="text-red-600 text-xs"></p>
        </div>
        <div class="mb-4">
          <label for="" class="form-label">Fecha de nacimiento</label>
          <input type="date"
            class="form-control"
            name="fecha"
            id="fecha"
            aria-describedby="helpId"
            required
          />
          <p id="error-fecha" class="text-red-600 text-xs"></p>
        </div>
        <div class="mb-4">
          <label for="" class="form-label"></label>
          <div class="form-check form-check-inline">
            <input class="form-check-input"
              type="checkbox"
              id="condiciones"
              value="condiciones"
              required
            />
            <label class="form-check-label">Acepta los términos y
condiciones de uso</label>
          </div>
          <p id="error-condiciones" class="text-red-600 text-xs"></p>
```

```
        </div>
        <p id="error-servidor" class="text-red-600"></p>
        <button id="submit" type="submit" class="btn">Crear
cuenta</button>
      </form>
    </div>
  );
}

export default Register;
```

This code represents a Register component that displays a registration form.

The component imports React as well as the useRef and useEffect hooks from the react package, and the validateRegister function from the ../content/validations/register module.

The Register component uses the useRef hook to create a reference to the form element (formRef). This reference will be used to access the form and perform validation.

The useEffect hook is used to trigger the registration validation when the component mounts. If the formRef.current exists, it calls the validateRegister function.

The component renders the registration form, which consists of various form inputs such as username, password, date of birth, and terms and conditions acceptance checkbox. Each input has specific attributes such as name, id, placeholder, minLength, and required. There are also corresponding error messages that are displayed when validation fails.

The form has an error message section (error-servidor) and a submit button.

# Validations

## Login.js

```
import { PHPController as Controller } from "./controller/controller.js";
/**
 * Validates login form
 */
export function validateLogin() {
    createListeners();
}
/**
 * Creates event listeners on each form field
 */
function createListeners() {
    document.getElementById("login").noValidate = true;

    for (let i = 0; i < document.getElementById("login").length; i++) {
        if (document.getElementById("login")[i].id !== "condiciones") {
            document.getElementById("login")[i].addEventListener("blur",
checkErrors, false);
            document.getElementById("login")[i].addEventListener("invalid
", displayErrors, false);
            document.getElementById("login")[i].addEventListener("input",
correctErrors, false);
        }
    }
    document.getElementById("submit").addEventListener("click",
validateForm, false);
}

const FIELD_ERROR_ID = "error-";
/**
 * Validates form when submit
 * @param {submit} e
 */
function validateForm(e) {
    e.preventDefault();
    let formValid = document.getElementById("login")[0].validity.valid;
    for (let i = 1; i < document.getElementById("login").length; i++) {
        formValid = document.getElementById("login")[i].validity.valid &&
formValid;
    }
    formValid = formValid;
```

```
    if (!formValid) {
        let formValid = true;
        const form = document.getElementById("login");
        for (let i = 0; i < form.length; i++) {
            const input = form[i];
            if (!input.validity.valid) {
                formValid = false;
                displayErrors(input);
            } else {
                removeErrors(input);
            }
        }
    } else {
        const username = document.getElementById("usuario").value;
        const password = document.getElementById("contraseña").value;
        login(username, password);
    }
}
/**
 * Sends form to API to check errors
 * @param {username} username
 * @param {password} password
 */
async function login(username, password) {
    const response = await Controller.login(username, password);
    if (response.status === "success") {
        localStorage.setItem('session', true);
        window.location.href = '/';
    } else {
        document.getElementById(FIELD_ERROR_ID + "servidor").innerHTML =
response.message;
    }
}
/**
 *
 * @param {formInput} e
 */
function checkErrors(e) {
    const input = e.target;
    removeErrors(input);
    displayErrors(input);
}
/**
 *
 * @param {formInput} e
 */
function correctErrors(e) {
    const input = e.target;
    if (input.validity.valid) {
```

```
        removeErrors(input);
    }
}
/**
 *
 * @param {formInput} input
 */
function removeErrors(input) {
    if (document.getElementById(FIELD_ERROR_ID + input.id)) {
        document.getElementById(FIELD_ERROR_ID + input.id).innerHTML =
"";
    }
}
/**
 *
 * @param {formInput} input
 */
function displayErrors(input) {
    if (input.validity.valueMissing) {
        document.getElementById(FIELD_ERROR_ID + input.id).innerHTML =
"El campo está vacío";
    } else if (input.validity.patternMismatch) {
        document.getElementById(FIELD_ERROR_ID + input.id).innerHTML =
input.title;
    } else if (input.validity.tooShort) {
        document.getElementById(FIELD_ERROR_ID + input.id).innerHTML =
"El campo es demasiado corto. Necesita " + input.minLength + " como
minimo";
    }
}
```

The import statement import { PHPController as Controller } from "./controller/controller.js"; allows access to the functions of the PHP controller to perform user authentication.

The validateLogin() function is the main function that is called to initiate the validation of the login form. This function invokes createListeners(), which creates event listeners on each form field.

The event listeners are responsible for detecting events like "blur" (when losing focus), "invalid" (when the field is invalid), and "input" (when a valid value is entered) on the form fields.

The validateForm() function is executed when clicking the submit button of the form. It checks the validity of each field and displays corresponding errors if the form is not valid. If the form is valid, it retrieves the username and password values and invokes the login() function to log in.

The login() function sends the user data to the PHP controller to perform authentication. If the response is successful, it saves the session and redirects to the main page. If the response contains an error, it displays the error message in the corresponding field.

The checkErrors(), correctErrors(), removeErrors(), and displayErrors() functions are used to check and display errors in the form fields in different situations.

## Register.js

```
import { PHPController as Controller } from "./controller/controller.js";
/**
 * Validates register form
 */
export function validateRegister() {
    createListeners();
}
/**
 * Creates event listeners on each form field
 */
function createListeners() {
    document.getElementById("register").noValidate = true;

    for (let i = 0; i < document.getElementById("register").length; i++)
{
        if (document.getElementById("register")[i].id !== "condiciones")
{
            document.getElementById("register")[i].addEventListener("blur
", checkErrors, false);
            document.getElementById("register")[i].addEventListener("inva
lid", displayErrors, false);
            document.getElementById("register")[i].addEventListener("inpu
t", correctErrors, false);
        }
    }
```

```
    document.getElementById("fecha").addEventListener("change",
validateAge);
    document.getElementById("fecha").addEventListener("blur",
validateAge);
    document.getElementById("condiciones").addEventListener("click",
checkErrors, false);
    document.getElementById("submit").addEventListener("click",
validateForm, false);
}

const FIELD_ERROR_ID = "error-";

/**
 * Validates form when submit
 * @param {submit} e
 */
function validateForm(e) {
    e.preventDefault();
    let formValid =
document.getElementById("register")[0].validity.valid;
    for (let i = 1; i < document.getElementById("register").length; i++)
{
        formValid = document.getElementById("register")[i].validity.valid
&& formValid;
    }
    formValid = document.getElementById("condiciones").checked &&
formValid;

    let ageValid = true;
    let ageError = false;

    if (!formValid || !ageValid) {
        const form = document.getElementById("register");
        for (let i = 0; i < form.length; i++) {
            const input = form[i];
            if (input.id !== "condiciones") {
                if (!input.validity.valid) {
                    formValid = false;
                    displayErrors(input);
                } else {
                    removeErrors(input);
                }
            }
        }

        formValid = false;

        if (ageError) {
```

```
            document.getElementById(FIELD_ERROR_ID + "fecha").innerHTML =
"Debes tener al menos 18 años";
        }

        if (!document.getElementById("condiciones").checked) {
            document.getElementById(FIELD_ERROR_ID +
"condiciones").innerHTML = "Acepta los términos de servicio";
        } else {
            document.getElementById(FIELD_ERROR_ID +
"condiciones").innerHTML = "";
        }
    } else {
        const username = document.getElementById("usuario").value;
        const password = document.getElementById("contraseña").value;
        register(username, password);
    }
}
/**
 * Sends form to API to check errors
 * @param {username} username
 * @param {password} password
 */
async function register(username, password) {
    const response = await Controller.register(username, password);
    if (response.status === "success") {
        localStorage.setItem('session', true);
        window.location.href = '/';
    } else {
        document.getElementById(FIELD_ERROR_ID + "servidor").innerHTML =
response.message;
    }
}

let ageValid = true;
let ageError = false;
/**
 * Check if age is greater than 18
 */
function validateAge() {
    const dateInput = document.getElementById("fecha");
    const selectedDate = new Date(dateInput.value);
    const currentDate = new Date();
    const minimumAge = 18;

    const ageDifference = currentDate.getFullYear() -
selectedDate.getFullYear();

    if (ageDifference < minimumAge) {
```

```
        document.getElementById(FIELD_ERROR_ID + dateInput.id).innerHTML
= "Debes tener al menos 18 años";
        ageValid = false;
        ageError = true;
    } else {
        removeErrors(dateInput);
        ageValid = true;
        ageError = false;
    }
}
/**
 *
 * @param {formInput} e
 */
function checkErrors(e) {
    const input = e.target;
    removeErrors(input);
    displayErrors(input);
}
/**
 *
 * @param {formInput} e
 */
function correctErrors(e) {
    const input = e.target;
    if (input.validity.valid) {
        removeErrors(input);
    }
}
/**
 *
 * @param {formInput} input
 */
function removeErrors(input) {
    if (document.getElementById(FIELD_ERROR_ID + input.id)) {
        document.getElementById(FIELD_ERROR_ID + input.id).innerHTML =
"";
    }
}
/**
 *
 * @param {formInput} input
 */
function displayErrors(input) {
    if (input.validity.valueMissing) {
        document.getElementById(FIELD_ERROR_ID + input.id).innerHTML =
"El campo está vacío";
    } else if (input.validity.patternMismatch) {
```

```
        document.getElementById(FIELD_ERROR_ID + input.id).innerHTML =
input.title;
    } else if (input.validity.tooShort) {
        document.getElementById(FIELD_ERROR_ID + input.id).innerHTML =
"El campo es demasiado corto. Necesita " + input.minLength + " como
minimo";
    }
}
```

The import statement import { PHPController as Controller } from "./controller/controller.js"; allows access to the functions of the PHP controller to perform user registration.

The validateRegister() function is the main function that is called to initiate the validation of the registration form. This function invokes createListeners(), which creates event listeners on each form field.

The event listeners are responsible for detecting events like "blur" (when losing focus), "invalid" (when the field is invalid), and "input" (when a valid value is entered) on the form fields.

The createListeners() function also adds event listeners to additional elements like the "fecha" (date of birth) field, the "condiciones" (terms and conditions) checkbox, and the submit button.

The validateForm() function is executed when clicking the submit button of the form. It checks the validity of each field, including checking if the "condiciones" checkbox is checked. If the form is not valid, it displays corresponding errors. If the form is valid, it retrieves the username and password values and invokes the register() function to register the user.

The register() function sends the username and password to the PHP controller for registration. If the registration is successful, it saves the session and redirects to the main page. If there is an error, it displays the error message in the corresponding field.

The validateAge() function checks if the selected date of birth is at least 18 years ago. It compares the selected date with the current date and displays an error message if the age is not valid.

The checkErrors(), correctErrors(), removeErrors(), and displayErrors() functions handle the display of error messages for each form field based on their validity.

# API Requests

## Controller.js

```
import axios from 'axios';
/**
 * Creates request to API
 */
export class PHPController {
/**
 * Gets pokemon list from API
 * @returns Pokemon list, types and stats
 */
  static async listPokemon() {
    let jsonResponse = null;
    try {
      const data = {
        method: 'generateList'
      };

      const response = await axios.post('http://localhost:80/pokedex-
api/app/controllers/pokemon_controller.php', data, {
        headers: {
          'Content-Type': 'application/json'
        }
      });

      jsonResponse = response.data;
    } catch (error) {
      console.error(error.message);
    }
    return jsonResponse;
  }
/**
 * Unused yet. Shows details of a specific pokemon
 * @param {id} id
 * @returns Pokemon name, types, description and stats
```

```
 */
  static async showPokemon(id) {
    let jsonResponse = null;
    try {
      const response = await axios.post('', null, {
        headers: {
          'Content-Type': 'application/json'
        }
      });
      jsonResponse = response.data;
    } catch (error) {
      console.error(error.message);
    }
    return jsonResponse;
  }
/**
 * Checks if an user is registered
 * @param {username} user
 * @param {password} password
 * @returns API response
 */
  static async register(user, password) {
    let jsonResponse = null;
    try {
      const data = {
        username: user,
        password: password,
        method: 'register'
      };

      const response = await axios.post('http://localhost:80/pokedex-
api/app/controllers/trainer_controller.php', data, {
        headers: {
          'Content-Type': 'application/json'
        }
      });

      jsonResponse = response.data;
    } catch (error) {
      console.error(error.message);
    }
    return jsonResponse;
  }
/**
 * Checks if form user data is correct
 * @param {username} user
 * @param {password} password
 * @returns API response
 */
```

```
  static async login(user, password) {
    let jsonResponse = null;
    try {
      const data = {
        username: user,
        password: password,
        method: 'login'
      };

      const response = await axios.post('http://localhost:80/pokedex-
api/app/controllers/trainer_controller.php', data, {
        headers: {
          'Content-Type': 'application/json'
        }
      });

      jsonResponse = response.data;
    } catch (error) {
      console.error(error.message);
    }
    return jsonResponse;
  }

}
```

import axios from 'axios'; is an import statement that allows you to use the Axios library in your code. Axios is a popular JavaScript library used for making HTTP requests from the browser or Node.js. listPokemon(): This method sends a POST request to the specified API endpoint (pokemon_controller.php) to retrieve a list of Pokémon, including their types and stats. It uses the axios.post() function with the specified URL, request data, and headers. The response data is stored in jsonResponse and returned.

showPokemon(id): This method is currently unused (Unused yet.). It would be responsible for showing details of a specific Pokémon identified by the given id. The code makes a POST request without specifying a URL or request data, so it would need further implementation to work correctly.

register(user, password): This method sends a POST request to the specified API endpoint (trainer_controller.php) to register a user with the provided username and password. It includes the method parameter set to 'register' in the request data. The response data is stored in jsonResponse and returned.

login(user, password): This method sends a POST request to the specified API endpoint (trainer_controller.php) to authenticate a user with the provided username and password. It includes the method parameter set to 'login' in the request data. The response data is stored in jsonResponse and returned.

In both the register() and login() methods, the code uses axios.post() to send the request, specifying the URL, request data, and headers. The response data is then stored in jsonResponse and returned.

Overall, this code demonstrates how to use Axios to make HTTP requests to interact with an API for tasks like retrieving Pokémon data, user registration, and user authentication.

# While developing

I preferred using Axios over the Fetch API when making server requests to try a new way of communicating with other servers. It is necessary to handle CORS policies to allow communication between servers.

The navigation bar did not update the content when changing routes when it was divided into two components, 'Navbar' and 'Content'. This was solved by keeping 'Page' as the container for Navbar, allowing 'Content' to modify its content. Validations took much longer than expected because a second structure that communicates with the application was not taken into account. This was resolved by creating pokedex-api.

The team assembly system, combat system, linking the account with PayPal, and payments with a PayPal account could not be implemented. The team assembly system exists partially in the code of pokedex-api, where it returns the result of the "plantillas" table in the pokedex database as $data, which would be used in the '/pokedex' route to select the user's 6-team. The combat system would be based on strategy and luck, taking into account that the higher the betting risk, the higher the money to be received. The strategy part lies in the statistics of the Pokémon that will fight against the trainer of the chosen server event in '/', considering who attacks first, who can withstand more hits, and who hits harder by using random numbers after comparing the Pokémon's statistics.

Once the original idea is fully implemented, the next step would be to add more payment methods to make the application more versatile and constantly add events to keep the user community active in the application while improving security and optimizing the code.