



# SportSee

Build an analytics dashboard  
with React

Developer at sportsee, a fast growing coaching startup. The company is launching a new version of the "profile" page for a user. It will allow the user to track the number of sessions performed as well as the number of calories burned

# Summary of Prerequisites

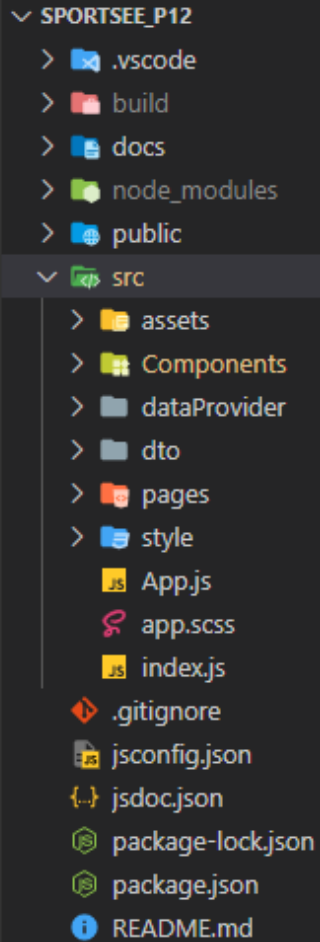
- **Language:** React, JavaScript library for building interactive user interfaces. React allows for the development of responsive web applications using reusable components.
- **Dependencies:**
  - **Axios:** JavaScript library used for making HTTP requests from a browser or a server. . Axios is used in the project to make API calls and retrieve data.
  - **Recharts:** Recharts is a data visualization library based on React. Recharts is used in the project to display data in the form of charts.
- **npm:** (Node Package Manager) It is used to install project dependencies, manage package versions, and handle build and start scripts for the project.
- **Node js**

He is should be installed using the npm install command in the project's root directory. This will download and install the required packages in the project's node\_modules folder.

# OBJECTIVES SPECIFICATIONS

- make the profile page with React
- Graphics integration
- Responsive Desktop 1024 x 780 pixels
- Retrieve data from the backend
- Use node js
- Realize the data mock
- Integrate the api
- Documentation in english
- Integrate PropTypes for each component

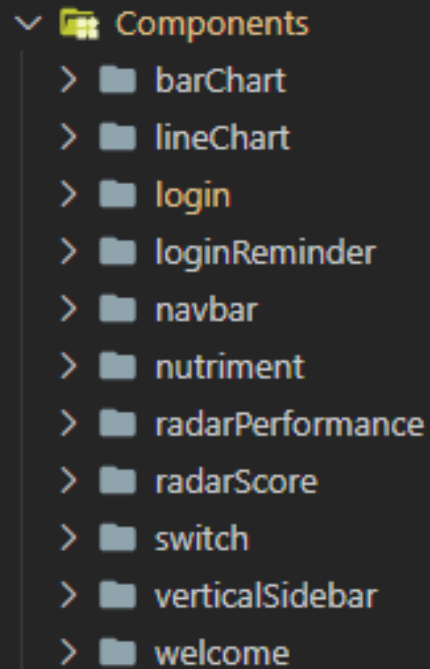
# PROJECT ARCHITECTURES



```

SPORTSEE_P12
├── .vscode
├── build
├── docs
├── node_modules
├── public
├── src
│   ├── assets
│   ├── Components
│   ├── dataProvider
│   ├── dto
│   ├── pages
│   ├── style
│   ├── App.js
│   ├── app.scss
│   ├── index.js
│   ├── .gitignore
│   ├── jsconfig.json
│   ├── jsdoc.json
│   ├── package-lock.json
│   ├── package.json
│   └── README.md

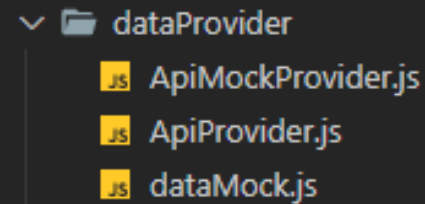
```



```

Components
├── barChart
├── lineChart
├── login
├── loginReminder
├── navbar
├── nutriment
├── radarPerformance
├── radarScore
├── switch
├── verticalSidebar
└── welcome

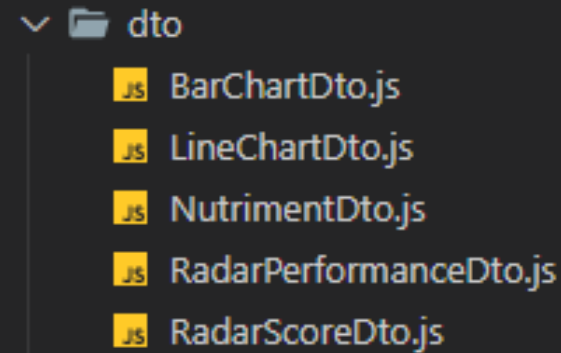
```



```

dataProvider
├── ApiMockProvider.js
├── ApiProvider.js
└── dataMock.js

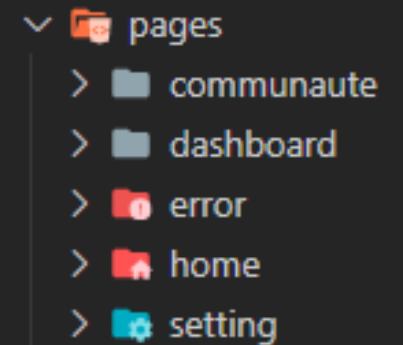
```



```

dto
├── BarChartDto.js
├── LineChartDto.js
├── NutrimentDto.js
├── RadarPerformanceDto.js
└── RadarScoreDto.js

```



```

pages
├── communaute
├── dashboard
├── error
├── home
└── setting

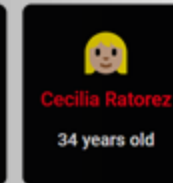
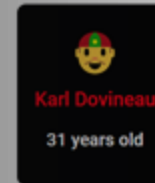
```

```
const App = () => {  
  return (  
    <header className="routePage">  
      <Navbar />  
      <VerticalSidebar />  
      <section className="content">  
        <Routes>  
          <Route path="/" element={<Home />} />  
          <Route path="/dashboard/:userId/:isDemo" element={<Dashboard />} />  
          <Route path="/setting/" element={<Setting />} />  
          <Route path="/communaute/" element={<Communaute />} />  
          <Route path="*" element={<Error />} />  
          <Route path="/login-reminder" element={<LoginReminder />} />  
        </Routes>  
      </section>  
    </header>  
  );  
};
```

# Routes - App.js



Choose and log in



Datas  
Mockees



Choose and log in



Datas  
API

# Home

# LOADER Dashboard page

HOOK - CREATE STATE



```
21 const [isLoading, setIsLoading] = useState(true);
```

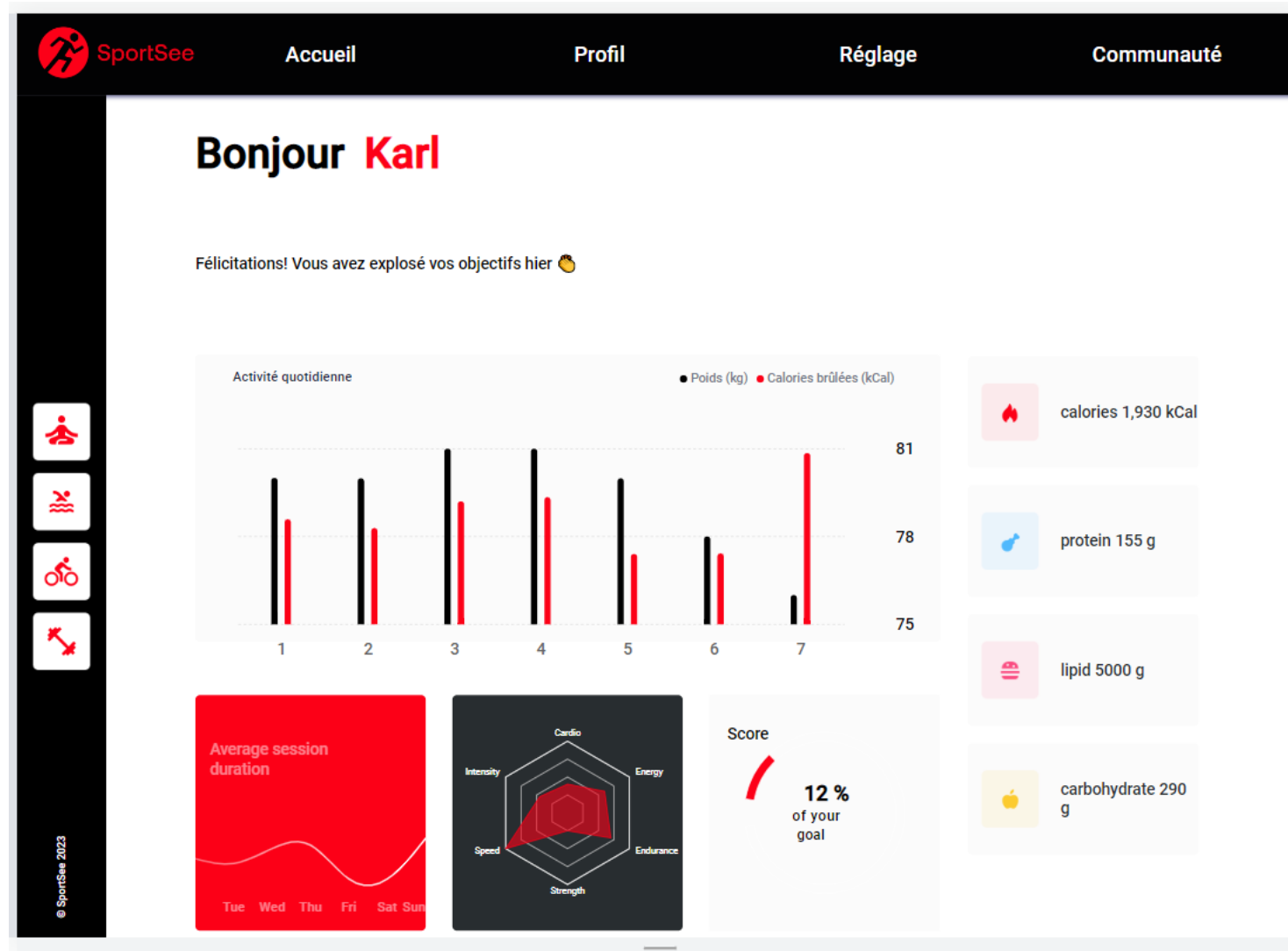
```
61 // Simulate loading for 5 seconds
62 await new Promise(resolve) => setTimeout(resolve, 1000);
63 setFirstName(firstName);
```

Creates an asynchronous promise that makes the code wait for one second (1000 milliseconds) before continuing

```
70 setIsLoading(false);
71 } catch (error) {
72   setIsLoading(false);
73 }
74 }
```

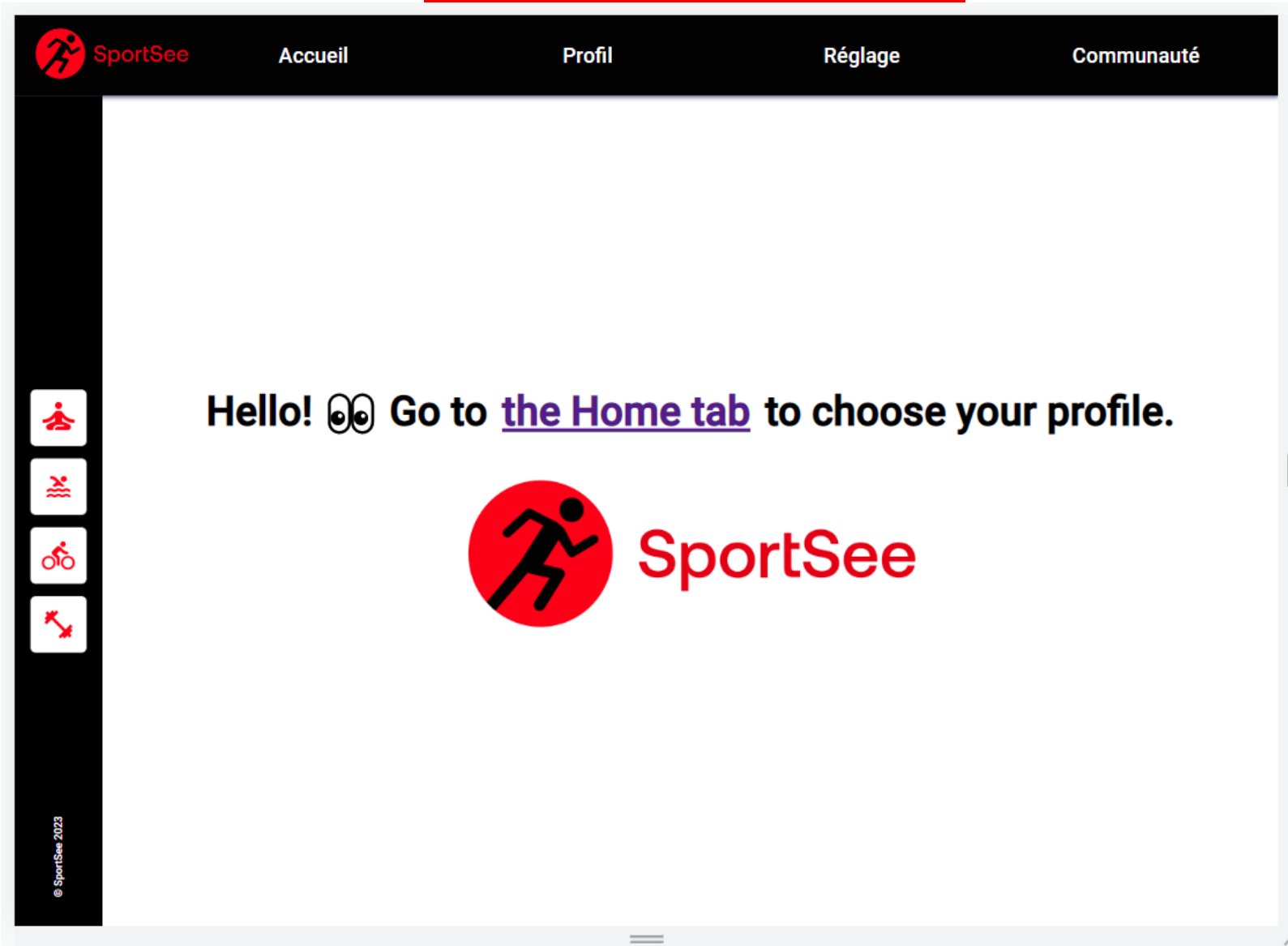
Finally, the line `setIsLoading(false)` sets the variable `isLoading` to false, indicating that the data loading is finished.

# Dashboard





# Profile



Login to be directed  
to your profile

```

class ApiMockProvider {
  /**
   * Retrieves user activities by user ID.
   * @param {number} userId - The user ID.
   * @returns {BarChartDto} - User activities in the form of BarChartDto
   * @memberof ApiMockProvider
   */
  getActivitiesByUserId(userId) {
    const userSessions = [];
    const currentUser = USER_ACTIVITY.find(
      (user) => user.userId === parseInt(userId)
    );
    if (currentUser) {
      currentUser.sessions.forEach((session) => {
        const day = session.day.split("-");
        const formattedDate = `${day}`;
        userSessions.push({
          day: formattedDate,
          kilogram: session.kilogram,
          calories: session.calories,
        });
      });
    }
    return new BarChartDto(
      userSessions,
      "Day",
      "Kilograms",
      "Calories"
    );
  }
}

```

# Class ApiMockProvider

- The ApiMockProvider class is a mock data provider that contains several methods for retrieving user data. These methods include
- Each method takes a user ID as a parameter and returns the corresponding data in the form of a specific data transfer object (DTO).
- The class imports several DTOs, including RadarPerformanceDto, NutrimentDto, LineChartDto, BarChartDto, and RadarScoreDto.
- In summary, the ApiProvider class provides convenient methods to retrieve various user information from an API using Axios and DTO objects to structure the retrieved data.

# Class DataProvider

The ApiProvider class is a class that provides functions for retrieving user data from an API. It contains several methods for retrieving user data.

Each method takes a user ID as a parameter and returns the corresponding data in the form of a specific data transfer object (DTO). The class imports several DTOs, including

The class uses Axios to make HTTP requests to retrieve user data from the API. It also contains a handleError method for handling errors when retrieving user data.

In summary, the ApiMockProvider class provides functions to retrieve simulated user data. These functions are used to test and simulate calls to a real API, providing mock data for development and debugging purposes.

```
class ApiProvider {
  constructor() {
    this.baseUrl = BASE_URL;
  }

  /**
   * Handles errors when retrieving user data.
   * @param {Error} error - The error generated when retrieving the data.
   * @throws {Error} - An error indicating that user data cannot be fetched.
   */
  handleError(error) {
    console.log("Error fetching user data: ", error);
    console.error("Error fetching user data: ", error);
    throw new Error("Unable to fetch user data");
  }

  /**
   * Retrieves the user's first name by user ID.
   * @param {string} userId - The user ID.
   * @returns {Promise<string|null>} - A promise that resolves with the user's first name
   */
  async getUsernameByUserId(userId) {
    // Check if the request URL is correct
    console.log("Request URL: ", this.baseUrl + userId);

    // Make a GET request using axios
    return axios
      .get(this.baseUrl + userId)
      .then((response) => {
        console.log("response: ", response);

        return response.data &&
          response.data.data &&
          response.data.data.userInfos &&
          response.data.data.userInfos.firstName
            ? response.data.data.userInfos.firstName
            : null;
      })
  }
}
```