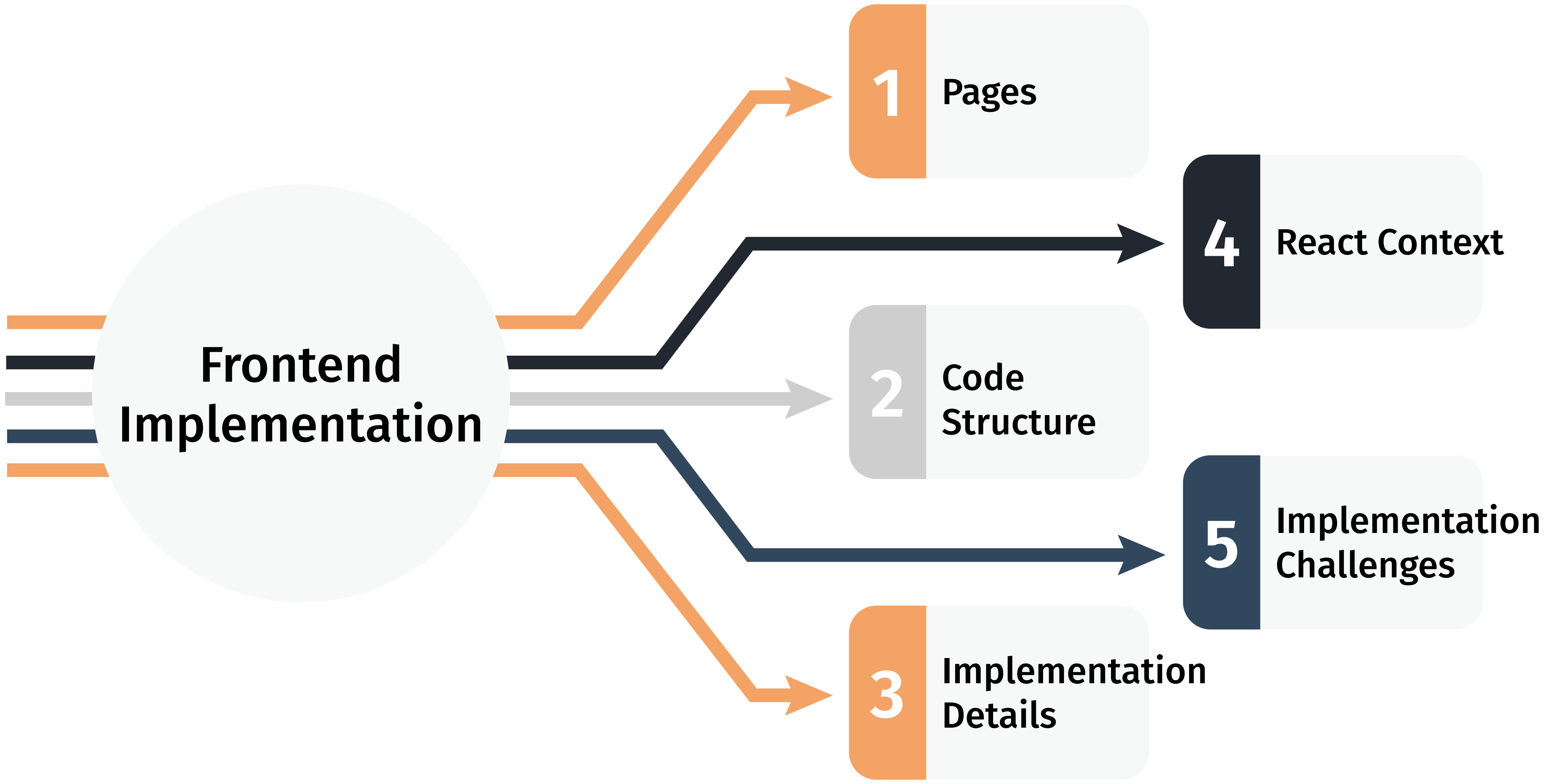
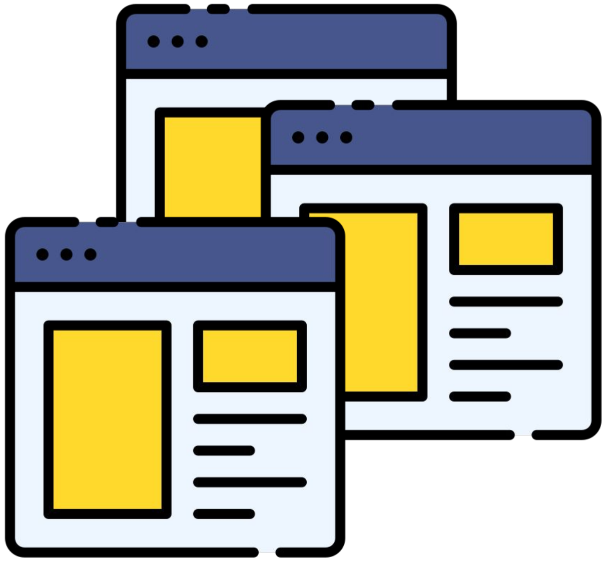
Web Applications Development

Phase 2 - Frontend

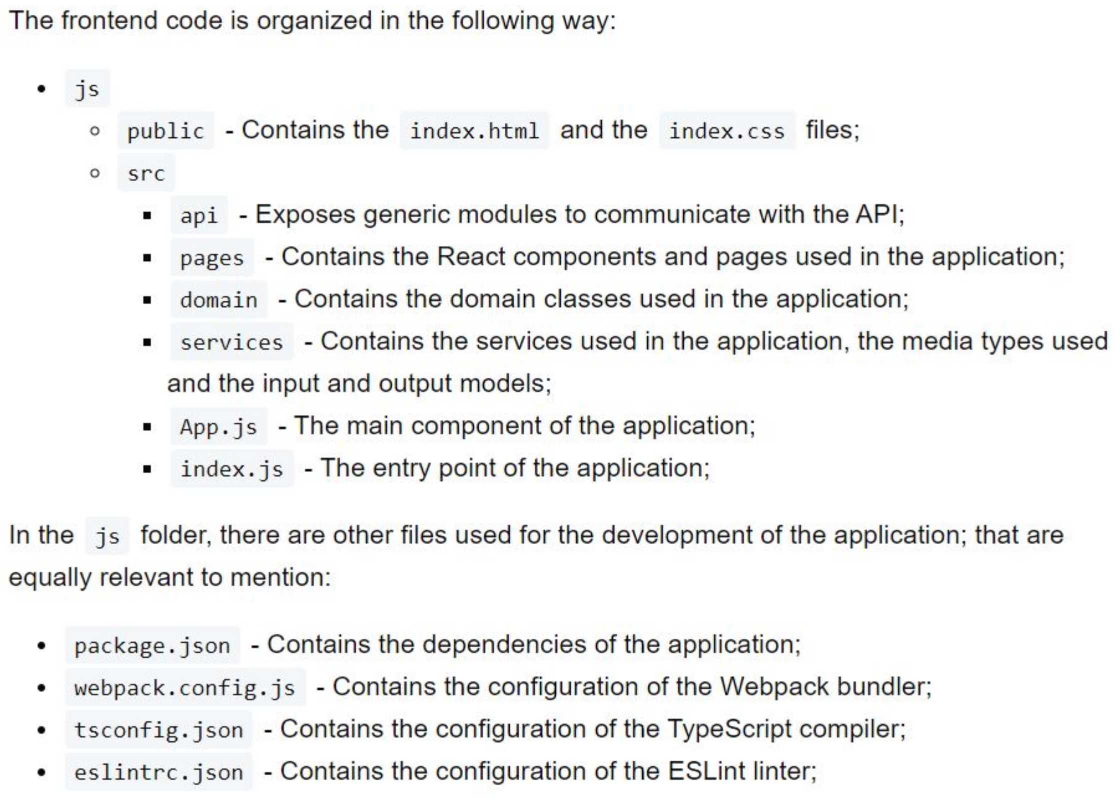

Frontend Implementation
1 Pages
4 React Context
2 Code Structure
5 Implementation Challenges
3 Implementation Details

Pages



- /
- /login
- /me
- /register
- /rankings
- /rankings/**:id**
- /games
- /games/**:gameId**
- /logout
- /about
- /error

Code Structure



The frontend code is organized in the following way:

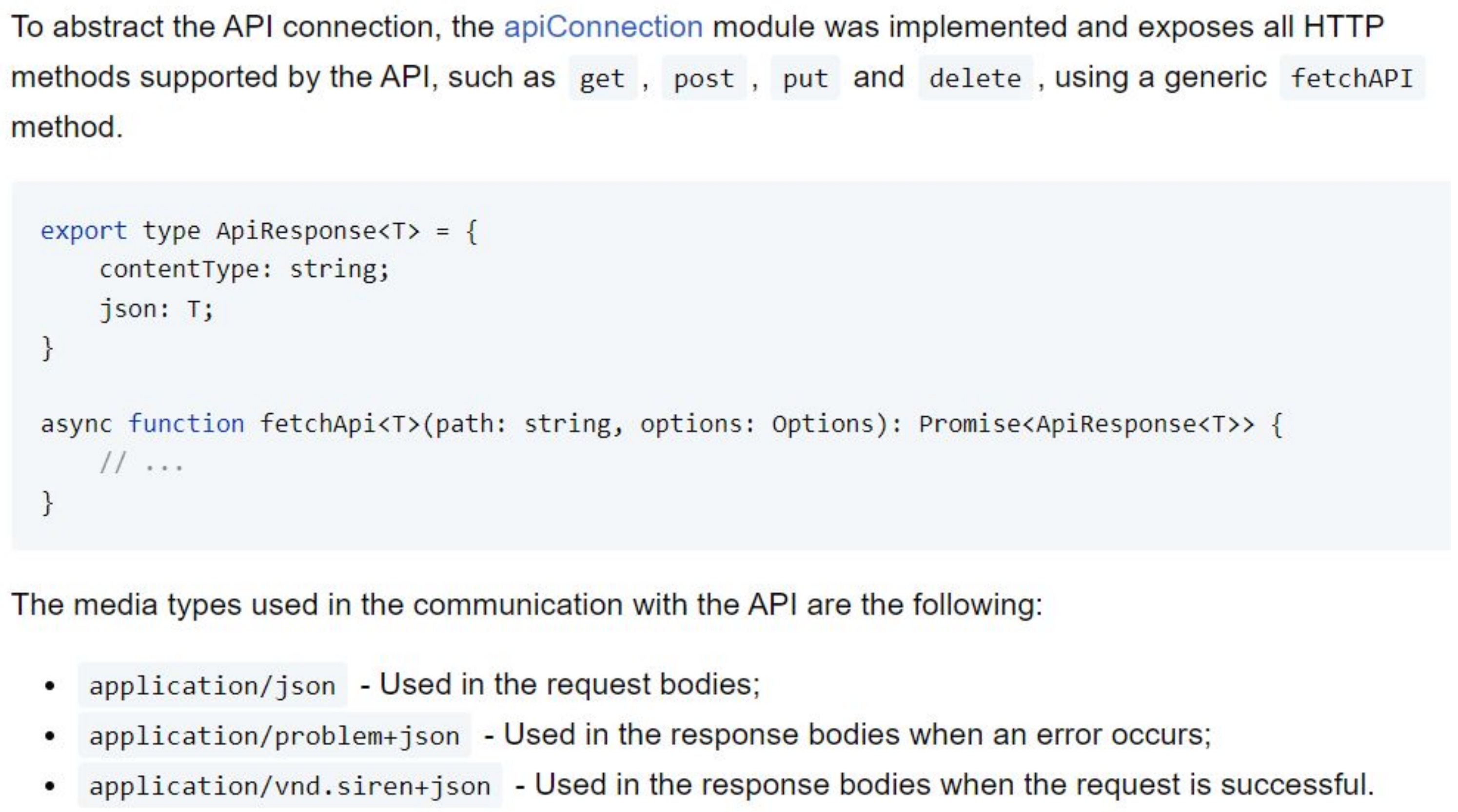
- `js`
  - `public` - Contains the `index.html` and the `index.css` files;
  - `src`
    - `api` - Exposes generic modules to communicate with the API;
    - `pages` - Contains the React components and pages used in the application;
    - `domain` - Contains the domain classes used in the application;
    - `services` - Contains the services used in the application, the media types used and the input and output models;
    - `App.js` - The main component of the application;
    - `index.js` - The entry point of the application;

In the `js` folder, there are other files used for the development of the application; that are equally relevant to mention:

- `package.json` - Contains the dependencies of the application;
- `webpack.config.js` - Contains the configuration of the Webpack bundler;
- `tsconfig.json` - Contains the configuration of the TypeScript compiler;
- `eslintrc.json` - Contains the configuration of the ESLint linter;

Implementation Details

API Connection



To abstract the API connection, the apiConnection module was implemented and exposes all HTTP methods supported by the API, such as `get` , `post` , `put` and `delete` , using a generic `fetchAPI` method.

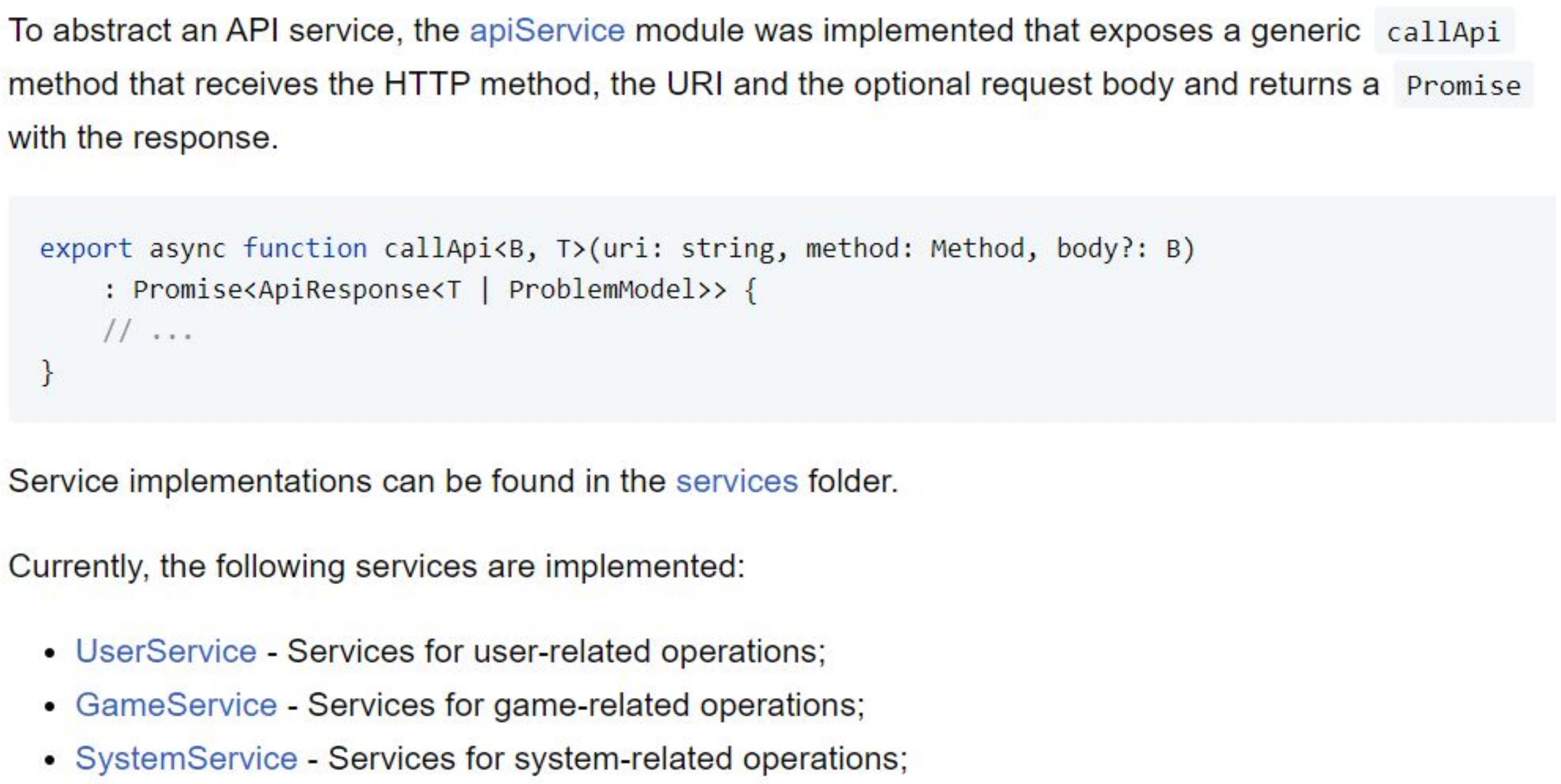
```
export type ApiResponse<T> = {
    contentType: string;
    json: T;
}

async function fetchApi<T>(path: string, options: Options): Promise<ApiResponse<T>> {
    // ...
}
```

The media types used in the communication with the API are the following:

- `application/json` - Used in the request bodies;
- `application/problem+json` - Used in the response bodies when an error occurs;
- `application/vnd.siren+json` - Used in the response bodies when the request is successful.

API Service



To abstract an API service, the apiService module was implemented that exposes a generic `callApi` method that receives the HTTP method, the URI and the optional request body and returns a `Promise` with the response.

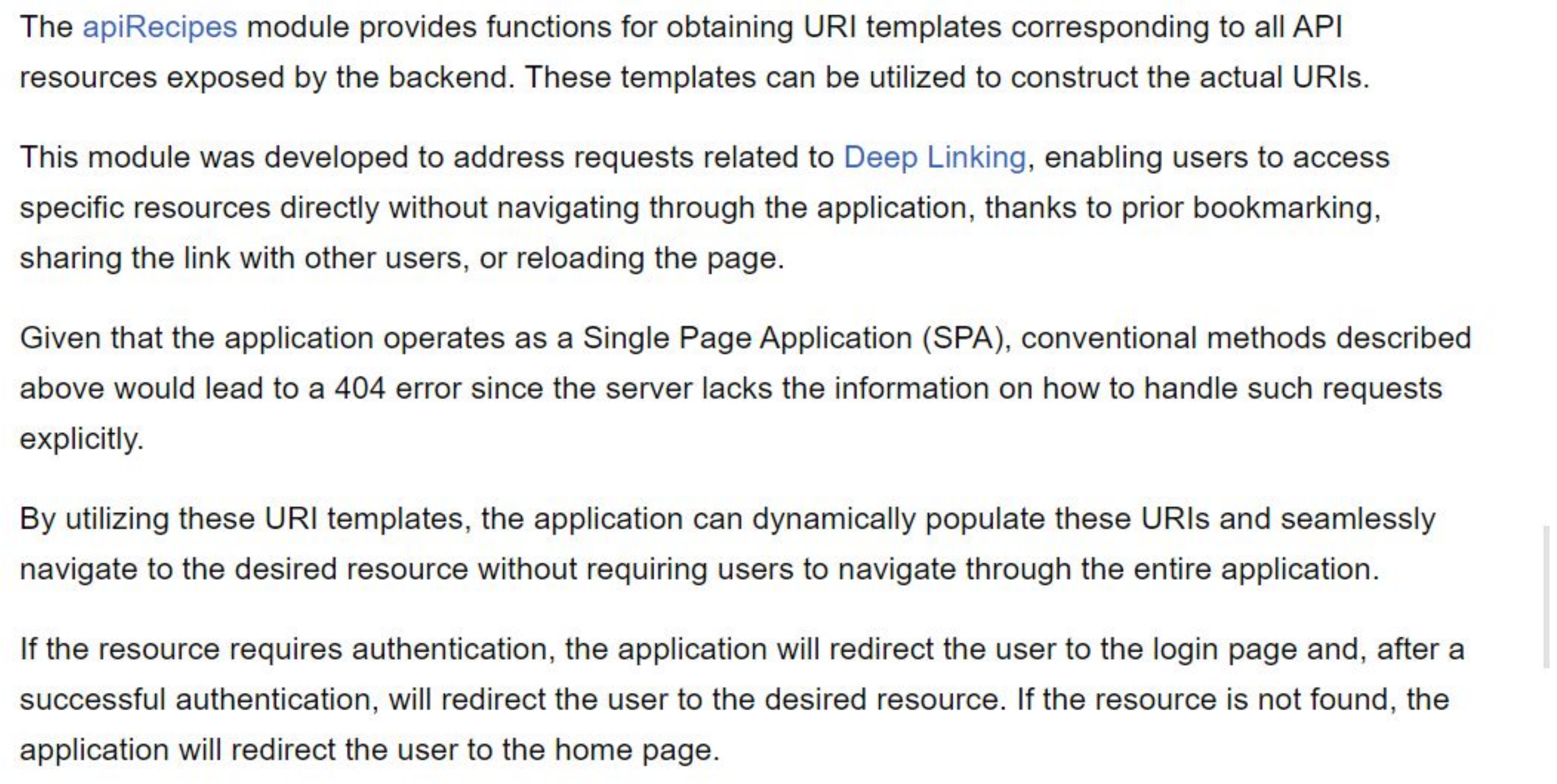
```
export async function callApi<B, T>(uri: string, method: Method, body?: B)
    : Promise<ApiResponse<T | ProblemModel>> {
    // ...
}
```

Service implementations can be found in the services folder.

Currently, the following services are implemented:

- UserService - Services for user-related operations;
- GameService - Services for game-related operations;
- SystemService - Services for system-related operations;

API Recipes



The apiRecipes module provides functions for obtaining URI templates corresponding to all API resources exposed by the backend. These templates can be utilized to construct the actual URIs.

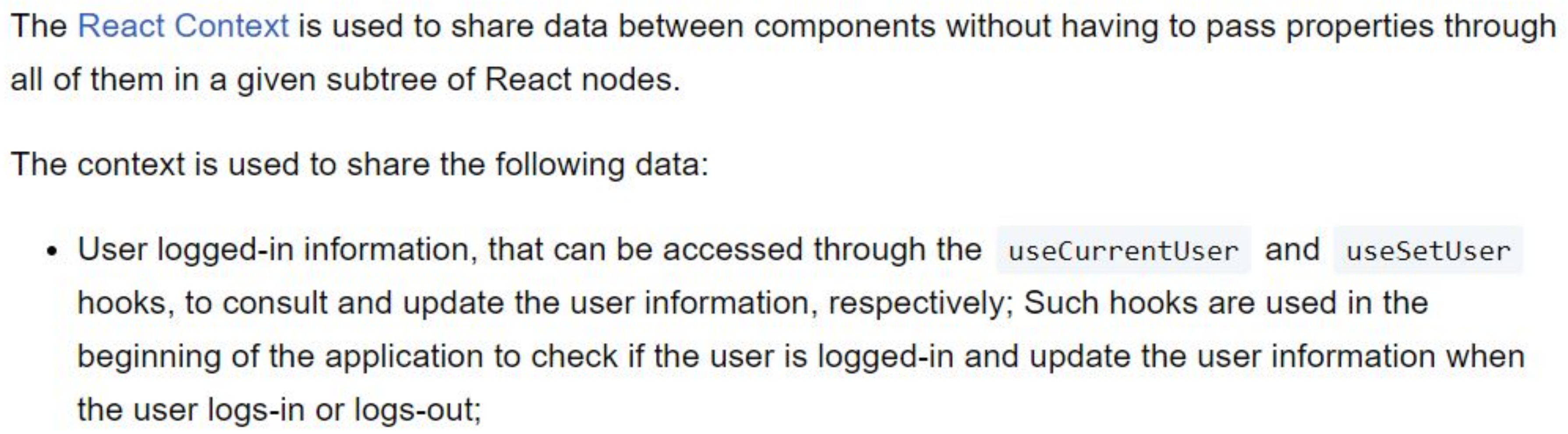
This module was developed to address requests related to Deep Linking, enabling users to access specific resources directly without navigating through the application, thanks to prior bookmarking, sharing the link with other users, or reloading the page.

Given that the application operates as a Single Page Application (SPA), conventional methods described above would lead to a 404 error since the server lacks the information on how to handle such requests explicitly.

By utilizing these URI templates, the application can dynamically populate these URIs and seamlessly navigate to the desired resource without requiring users to navigate through the entire application.

If the resource requires authentication, the application will redirect the user to the login page and, after a successful authentication, will redirect the user to the desired resource. If the resource is not found, the application will redirect the user to the home page.

React Context



The React Context is used to share data between components without having to pass properties through all of them in a given subtree of React nodes.

The context is used to share the following data:

- User logged-in information, that can be accessed through the `useCurrentUser` and `useSetUser` hooks, to consult and update the user information, respectively; Such hooks are used in the beginning of the application to check if the user is logged-in and update the user information when the user logs-in or logs-out;

Implementation Challenges

API Integration

Concurrency

Further Improvements

CSS

Improve User Experience

Revisit and Write more Tests

More Pages

Demo

1.0.1