RICK AND MORTY EXPLORER

This document provides an overview of Rick and Morty Explorer, a React project, offering users to explore characters, locations and episodes of Rick and Morty animated sitcom.

API
The project utilizes REST architecture with the base URL set to https://rickandmortyapi.com/api where all available API resources can be accessed via GET requests and returning data in JSON format for all responses.

DEPLOYMENT
The project is deployed and accessible at https://rick-morty-info.vercel.app/

PROJECT USAGE
For instructions on how to use the project code, please refer to the README file included in the public project repository on GitHub: https://github.com/festonamu/rick-morty

TECHNOLOGIES USED
- React 18
- TypeScript
- SCSS

CREATION
This project was created using create-react-app with the --template typescript flag.

PACKAGES
Due to the nature of the project and its simplicity, not many packages outside of default ones were required. Besides:
- react-router-dom, for routing purposes;
- sass, for styling;
- @fortawesome/react-fontawesome and @fortawesome/free-solid-svg-icons, for icons;
- jest libraries, for unit tests.

STYLING
To enhance maintainability and reusability, SCSS was used for styling, applying the BEM (Block Element Modifier) methodology for class naming.
Stylesheets are organized based on components so each component imports respective styles from these files, with separate files for main and common style.
Mixins are used for creating reusable style, particularly for flex display management and link styling; and variables, mostly used for colors, to enable theme changes and usage all around.

VIEWS
- **App.tsx**: Is importen in the main index.tsx (from which the app is rendered). It holds all the views (Home, Cards, Details); and Navbar and Footer, which are static (top and bottom)
- **Navbar.tsx**: Contains a theme changer button and navigation links (characters, locations, episodes) which also are setting the type of the overview Cards.
- **Home.tsx**: Displays a slider showing random character avatar and the navigation links too.

- **Cards.tsx**: Displays cards of a specific type (characters, locations, episodes) with filters and dynamic pagination included. Structure is uniform across types but details are passed as properties to the Overview.tsx component.
- **Details.tsx**: Shows a specific character, location, or episode with all associated data. Structure is uniform across types but details are passed as properties to corresponding components (Characters.tsx, Location.tsx, Episode.tsx). From here navigating options to other type details is included, and with next and previous buttons for navigating to same type data.

COMPONENTS

- **Filters.tsx**: Contains inputs according to the type of cards which can be used in Cards.tsx to filter the elements by their overview informations.
- **Pagination.jsx**: Also used in Cards.tsx with the purpose of navigating through pages, and is dynamic based on the type of data and the current page.
- **Loader.tsx**: Used in both Cards.tsx and Details.tsx, displays until the data is fetched. If there is no data or an error occurs, it receives a property showing a message for why there is no data fetched (not found or error fetching).

UTILITIES

- **extractId.ts**: is used to obtain specified or random avatars for locations and episodes, ensuring characters are residents of locations and played in episodes.

HOOKS

- **useFetch.ts**: One of the most important roles, it fetches the data every time its parameter (url – the API) changes, setting the data results and pagination information, loader, and errors (if any).

UNIT TESTS

- **useFetch.test.ts**: Is created to verify the functionality of the useFetch hook, ensuring that it correctly fetches data from the specified API endpoint and handles loading states and errors appropriately. It tests various scenarios such as successful data fetching, loading state transitions, and error handling.
- **extractId.test.ts**: Is created to verify the correctness of the extractId function by checking its behavior against different inputs, ensuring that it generates valid IDs for images.
- **Pagination.test.tsx**: Is verifying the correctness of pagination functionality, testing scenarios such as navigating to the next and previous pages, proper rendering of page numbers, and handling edge cases like reaching the first or last page.