# **טכנולוגיות אינטרנט מתקדמות - 61776 (WEB)**

**הגשת פרויקט**

**Interactive Infographics - B22 - Group9**

|  |  |
| --- | --- |
| שם חבר.ת הצוות | תז |
| רביע לחאם – Rabea Lahham | **209318419** |
| תאמר עאמר – Tamer Amer | **207418583** |
| בהאלדיין סויד – Bahaldin Swied | **207659392** |
| מובאדא שחאדה – Mbada Shehady | **212800080** |

**GraphiX** – A WEB application is designed to simplify the process of creating and visualizing data through interactive infographics. With a user-friendly interface, it offers users the ability to upload JSON files containing their data, which is then automatically transformed into informative graphs. This feature is ideal for quickly visualizing and analyzing existing datasets. For those who prefer to start from scratch, the application provides tools to create graphs tailored to specific needs. Users can input data manually or through JSON, and customize their graphs to represent data in the most effective way. This application is perfect for data analysts, researchers, and anyone interested in transforming raw data into visually compelling and interactive charts. Whether you’re looking to analyze data from a JSON file or design custom infographics, our platform makes the process intuitive and efficient.

**Key Features:**

* Chart Browsing: Users can explore a curated list of charts on the create page.
* Detailed Instructions: Each chart type comes with comprehensive explanations on how to prepare the JSON files.
* Responsive Design: Optimized for various device sizes, providing a consistent user experience across desktops, tablets, and mobile devices.
* Accessibility: Navigation controls and UI elements are designed to be accessible and user-friendly.

**Technologies Used:**

* React: A JavaScript library for building user interfaces with component-based architecture.
* Tailwind CSS: A utility-first CSS framework for styling the application.
* React Hooks: Functions that allow you to use state and other React features and making it easier to manage side effects and state within functional components.
* Charts.js: A JavaScript library for creating interactive and customizable charts and graphs in web applications.
* Database: A Json file used as a database for all the recipes information.

**Project Links:**

|  |  |
| --- | --- |
| Link To Website | <https://interactive-infographics.vercel.app/> |
| Link To Repository | [**https://github.com/oRABiiA/WEB-HW3**](https://github.com/oRABiiA/WEB-HW3) |
| Link To MTW | [**https://www.morethanwallet.com/app/870**](https://www.morethanwallet.com/app/870) |

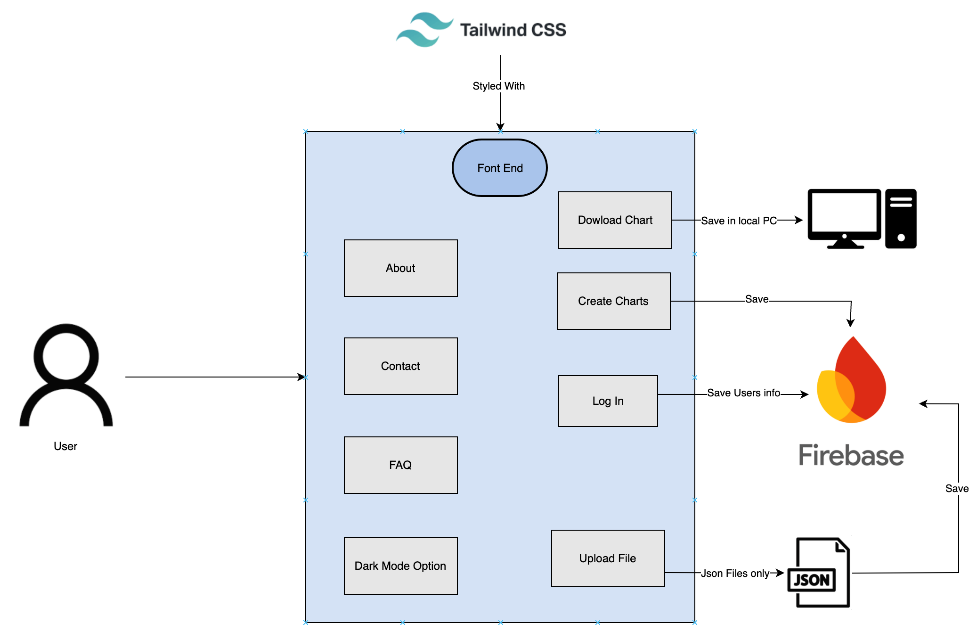
**Assignments:**

|  |  |  |
| --- | --- | --- |
| **Participants** | **Made Assignments** | **Completed Assignments** |
| **Rabea (Project Manager)** | 1. Website Architecture 2. Upload Page 3. Programmer Guide | Done |
| **Mbada** | 1. Website Architecture 2. Code Refactoring 3. Programmer Guide | Done |
| **Bahha** | 1. Website Architecture 2. Create Page 3. User Guide | Done |
| **Tamer** | 1. Website Architecture 2. Code Refactoring 3. User Guide | Done |

**Requirements:**

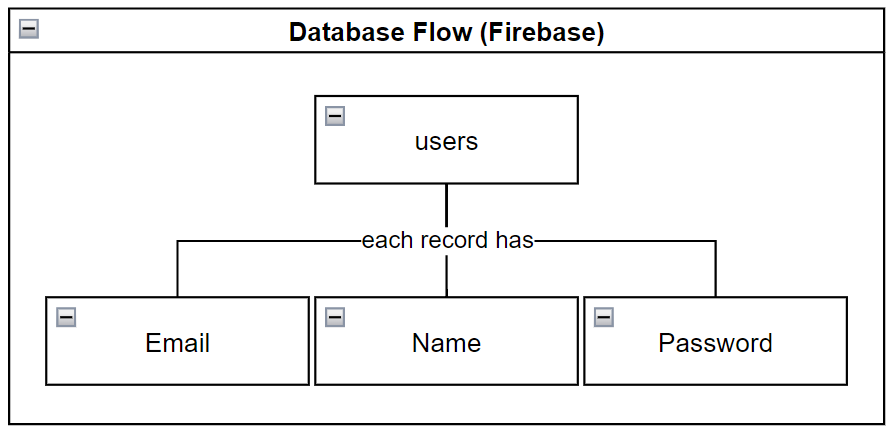
|  |  |
| --- | --- |
| **Functional Requirements** | |
| System allows users to upload JSON files containing data for visualization. | |
| System provides different types of interactive charts based on the uploaded data. | |
| System allows responsiveness on various devices. | |
| System allows to users to login \ register. | |
| System allows users to download graph as image. | |
| System allows users to pick colors for charts | |
| System allows users to adjust charts values. | |
| **Non- Functional Requirements** | |
| The website shall be user-friendly, with intuitive navigation and a responsive design suitable for various devices (desktop, tablet, smartphone). | Usability |
| The system shall feature a user-friendly interface, designed with intuitive navigation and clear visual hierarchy. | Usability |
| The website shall load within 3 seconds on standard connections, and uploading charts should happen smoothly. | Performance |
| The system shall support an increasing number of users and graphs without degradation in performance or usability. | Scalability |
| The website shall be available 24/7. | Availability |
| The website’s codebase shall be structured and documented to facilitate easy updates, bug fixes, and feature enhancements. | Maintainability |
| The website shall offer customization options for users, such as theme selection, colors for charts and notification settings to enhance the personal user experience. | Customization |
| Adjusting chart values or colors will based on the number of the user choosing. | Customization |
| The website must maintain high-quality, engaging, and error-free content to attract and retain visitors. | Content Quality |
| The design should focus on simplicity and ease of use, avoiding unnecessary elements. | Design |

# **Website Architecture:**

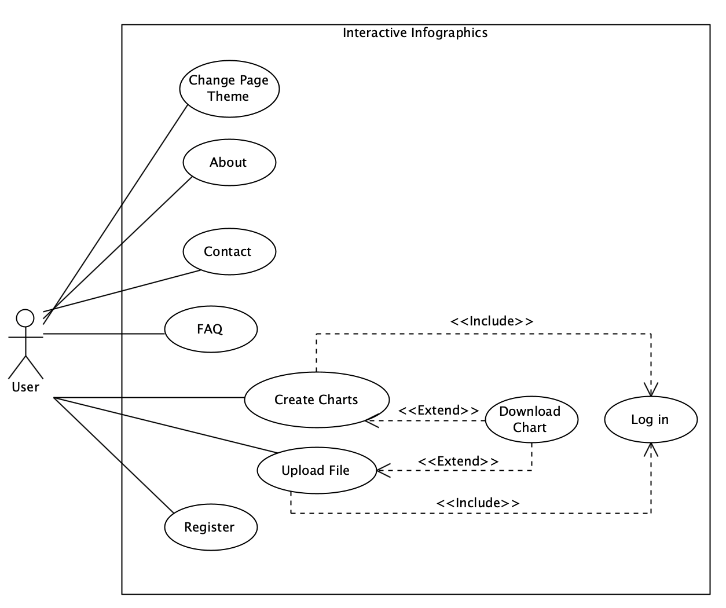


# **Database Architecture:**

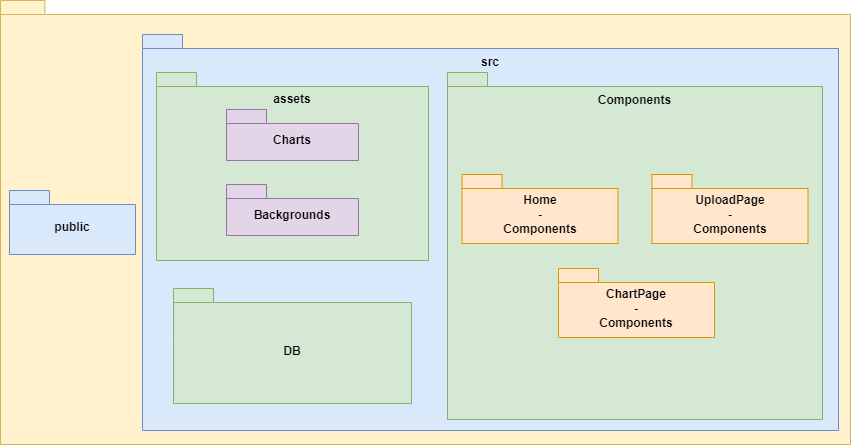
The Database includes information about the users inside users table. The users table has saved records with fields Email, Password and Name.  
The information is saved as the following:



# **Use Case Diagram:**



# **Components & Package Diagram:**



User Guide

*Tamer Amer – 207418583  
Rabea Lahham – 209318419  
Bahaldin Swied – 207659392  
Mbada Shehady – 212800080*

**Table of Contents**

Header …………………………………………………………….  
 Home button……………………………………………………….  
 About button………………………………………………………..  
 Contact button……………………………………………………..  
 FAQ button…………………………………………………………..  
 Dark/Light mode button…………………………………………

Footer ………………………………………………………………  
 Facebook button…………………………………………………..  
 Twitter button……………………………………………………….  
 Telegram button…………………………………………………….  
 Instagram button…………………………………………………..  
  
Home Page………………………………………………………..  
 Login/Register navigation button……………………………..  
 Login/Register window……………………………………………

About Page…………………………………………………………  
 Page content……………………………………………..

Contact Page……………………………………………………..  
 Page content……………………………………………..

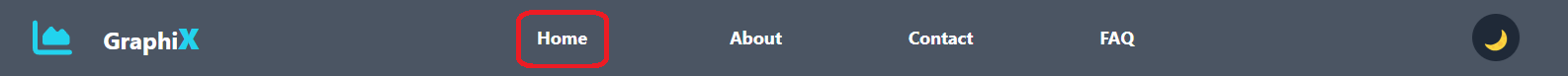
FAQ Page…………………………………………………………..  
 Page content……………………………………………..

Upload Page……………………………………………………….  
 Create/Upload window…………………………………………..  
 Guidance window………………………………………………….  
 Back navigation button…………………………………………..

Create Charts Page………………………………………………  
 Size Box……………………………………………………………….  
 Create button……………………………………………………….  
 Clear button…………………………………………………………  
 Back navigations button…………………………………………

**Header**

**Home Button:** it navigates you to the main landing page of the website(starting page).

****

**About Button:** it navigates you to the About page of the website.  
**Contact Button:** it navigates you to the Contact page of the website.  


**FAQ Button:** it navigates you to the FAQ page of the website.  
****

**Dark/Light Mode Button:** it swaps between dark and light mode of the website ****

**Footer**

**Facebook Button:** it navigates you to the Facebook website.  
A black circle with white x in it

Description automatically generated

**Twitter Button:** it navigates you to the Twitter website.

A red line with a black circle and a white x in a black circle and blue circle

Description automatically generated

**Telegram Button:** it navigates you to the Telegram website.

A black circle with a red line and a white x in the middle

Description automatically generated

**Instagram Button:** it navigates you to the Instagram website.

A black circle with white x in it

Description automatically generated

**Home Page**

A screenshot of a login form

Description automatically generated

**Login/Register Window:** Login in into the site with an existing account for further actions within the website, register with new account or reset your existing account password and you could also navigate to the Twitter and Facebook pages.

A screenshot of a login form

Description automatically generated

You can sign in using your Facebook account or your Twitter account

A screenshot of a login box

Description automatically generated

If you have an existing account already insert the Email address and the password and click login.

A screenshot of a login form

Description automatically generated

If you have an existing account already but you forgot the password, click “Forget Password” to rest your password and then login in.

A screenshot of a login form

Description automatically generated

If you don’t have an account, you can register with a new account and login into the website with the new account.

**About Page**

**About Page:** in this page you can find all the information about the team who participated creating this website in the **“About Us”** section, all the key features of the website and what we offer under the **“Key Features"** section , an explanation why our website is needed now days under the **“Why Interactive Infographics”**, what our mission as a team under **“Out Mission”** section and more information about our team under the **“Our Team”** , in case you didn’t find the information you are looking for you can contact us for more information using the **“Contact”** Page.

A screenshot of a website

Description automatically generated

**Contact Page**

**Contact Page:** Contact page provides you with multiple methods to contact out team and working hours and expected respones waiting time.

**Contact Us Box:** easy to use method where you insert your personal details with the provided questions.

**Other Ways to Reach Us:** More options for the user to contact us, we are also reachable via the official Email, Phone number and mail box.

**Business Hours:** provies you with the working hours of the team.

**Respone Time:** information about the expected time for us to respone back to your request

A screenshot of a computer

Description automatically generated

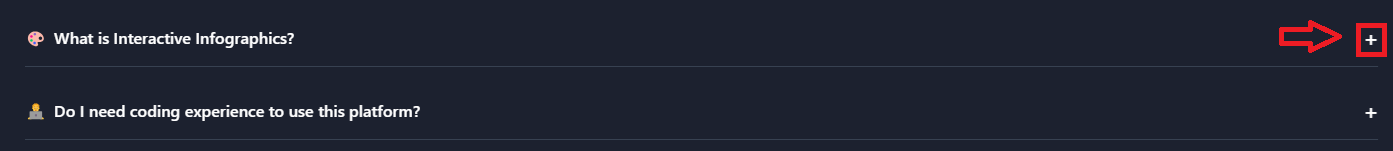
**FAQ Page**

**FAQ Page:** this page provides the user with the “Frequently Asked Questions”, if you are looking for an answer for a question you have about the website, you might the find the answer in this Page.

A screenshot of a computer

Description automatically generated

You simply click on the plus as shown below for the desired questions



The Answer you are looking for

A blue and red rectangle with white text

Description automatically generated

**Upload Page**

**Upload Page:** on this page you can create and design your wanted charts using out website, it contains 3 main elements, Create-Charts/Upload-File window, guidelines window on how to upload JSON files and a back navigation button.

**Create-Charts/Upload-File window:** in this window you could choose whether to create charts manually or to upload a JSON file with the desired information according to the structure that we provide information about in the **“guidelines window”.**

**Guidelines window**: in this window you get all the information needed about the structure of the JSON file needed to create your own desired charts with a provided example for extra clarification about the instructions.

**Back navigation button:** using this button allows you to come back with to the “Home page”, clicking this button will not log you out from the website and keeps you connected

**Create-Charts/Upload-File window:**

**A blue background with white text

Description automatically generated**clicking on “Create Charts” allows you to create your own charts by inserting your data manually

clicking on “Upload File” allows you to upload a JSON file that contains your data and automatically create your charts based on the type of the charts you choose

A screenshot of a computer

Description automatically generated

**Guidelines window**:

In this window, you will find all the information you need about the process of uploading a JSON file, we also provide a reliable example, you simply click on the desired chart type you are looking for and an example will show up for you

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Back navigation button:** if you need to navigate to the “Home Page”, clicking on this button will do the work!

A white circle with a black dot in a red frame

Description automatically generated

**Create Charts Page**

**Create Charts Page:** this page comes into play after the user clicks on **“Create Charts”** from the **“Upload page”**, it’s for the user that decides to create his own charts by inserting the data manually, it contains 4 main elements, the “Size Bar”, “Create”, “Clear” and the “navigation back button”.

**Size bar:** it allows the user to choose the amount and the size of the charts.

A screenshot of a computer

Description automatically generated

After clicking on the box a table of options will open:

A screenshot of a computer

Description automatically generated

After the user chooses a desired size, another box will open according to the user’s decision:

A screenshot of a computer

Description automatically generated

A screenshot of a bar chart

Description automatically generated

After the user chooses the chart type, boxes will open for the user to insert his data, an example is showing below:



The user inserts his data in the “Data” boxes and gives names to the “Labels”



A screenshot of a computer

Description automatically generated

The user has the option also to choose a color for his charts:

A white rectangular object with a black background

Description automatically generated

A screen shot of a computer

Description automatically generated

When the user is done with inserting his data, giving names for the labels and chooses the color the user, the user clicks “create” and his chart will be created:

A red green and yellow circles

Description automatically generated

A graph of data and a group

Description automatically generated with medium confidence

When the user gets his chart according to the inserted data, the user can “clear” the plot to create more charts:

Clear:

A red green and yellow circles

Description automatically generated

A screenshot of a computer

Description automatically generated

Clicking on “Clear” will reset the page to the default settings where the user could start the whole process again, as explained above

navigation back button: this button will bring back the user to the “Upload Page”

A white circle with a black dot in a red frame

Description automatically generated

Programmer Guide

*Tamer Amer – 207418583  
Rabea Lahham – 209318419  
Bahaldin Swied – 207659392  
Mbada Shehady – 212800080*

Table Of Contents

[1. Introduction](#_Toc163214355) 25

* 1. [System Architecture Overview](#_Toc163214356) 25
     1. [Key Components](#_Toc163214357) 25

[1.1.2 Key Libraries and Frameworks](#_Toc163214358) 25

[2. Key Files and Folders](#_Toc163214359) 26

[2.1 Index.html](#_Toc163214360) 26

[2.2 JSON Files Folder](#_Toc163214361) 26

[2.3 public folder](#_Toc163214362) 26

[2.4 src folder](#_Toc163214363) 26

[2.5 package.json](#_Toc163214364) 27

[3. Components](#_Toc163214365) 28

[3.1 Always loaded components](#_Toc163214367) 28

[3.1.1 Header.jsx](#_Toc163214368) 28

[3.1.2 Footer.jsx](#_Toc163214369) 30

[3.2 Specific page components](#_Toc163214371) 31

[3.2.1 About.jsx](#_Toc163214372) 31

[3.2.2 Contact.jsx](#_Toc163214373) 32

* + 1. [CreatePage.jsx](#_Toc163214374) 33
    2. FAQ.jsx ……………………………………………………………………………………… 34
  1. ChartPage components ……………………………………………………………… 35
     1. ChartHeader.jsx ………………………………………………………………………… 35
     2. ChartPage.jsx …………………………………………………………………………… 36
  2. Home components ……………………………………………………………………… 37
     1. ForgotPasswordCard.jsx …………………………………………………………… 37
     2. Home.jsx …………………………………………………………………………………… 38
     3. RegisterCard.jsx ………………………………………………………………………… 39
     4. SuccessMessageCard.jsx …………………………………………………………… 40
  3. UploadPage components ……………………………………………………………… 41
     1. InfoCard.jsx …………………………………………………………………………………41
     2. UploadPage.jsx ……………………………………………………………………………42

[4. Key functions](#_Toc163214375) 43

[4.1 State Management For Page Rendering](#_Toc163214376) 43

[4.2 Theme Context Provider](#_Toc163214377) 44

[4.3 Data Context Provider](#_Toc163214378) 44

## Introduction

This manual serves as a guide for developers working on the GraphiX – Interactive Infographics Website stored on GitHub repository and hosted on Vercel. It details the Web architecture, code organization, interaction between the components of React, and how to use and modify the React components with tailwind for changing the UI.  
This platform is designed to empower users with the ability to create and customize charts effortlessly. Whether you have data stored in JSON files or want to start from scratch, our website provides a user-friendly interface for generating a wide range of visualizations. Key features include:

* **Chart Generation**: Import JSON files or manually input data to generate various types of charts, including bar charts, line graphs, pie charts, and more.
* **Customization Options**: Tailor your charts with different styles, colors, and configurations to match your specific needs.
* **Interactive Interface**: Intuitive tools and controls make it easy to modify charts on the fly, ensuring you can achieve the exact look and functionality you require.

### 1.1 System Architecture Overview

The application integrates ReactJS for web coding, a json file as a database for storing and retrieving data, and tailwind as a UI design framework.

#### Key Components

* **ReactJS:** Using components architecture with ReactJS Framework.
* **Database:** User data stored in Firebase to retrieve and use users login/register.
* **UI**: Designed using tailwind framework.

#### Key Libraries and Frameworks

* **Vite**: For creating the web application.
* **ReactJS:** For components development.
* **Tailwind:** UI elements design.
* **Vercel:** For hosting the website.

**Key Files and Folders**

We used Vite to create our project. Vite makes a template for the project which we started developing our website from this basis.

**Index.html:**

The index.html file serves as the skeleton of the web application. It provides the basic HTML structure required to bootstrap the React app and includes important meta tags for responsiveness and proper rendering. The <div id="root"></div> acts as the mounting point for the React application, where all the React components will be injected.

**JSON Files Folder:**

A folder which holds a sample JSON files for usage in case the user does not know how to upload a JSON file in the upload page. Use these files to understand how to upload JSON files.

**Public Folder:**

The public folder contains static assets that are directly served to the client. In this project, it includes the SVG file used for the website's tab icon.

**Src Folder:**

The src folder contains the core source code for the application. It is organized into several subfolders and files that define the structure and functionality of the React application. Below is a detailed description of each component and folder within the src directory:

1. assets Folder:

The assets folder holds static resources such as images used in the application. This folder provides a centralized location for managing assets that are imported and used within components.

1. components Folder:

The components folder contains React components that define the structure and behavior of different parts of the user interface.  
Organizes reusable UI components such as buttons, forms, and layout elements. Each component is typically defined in its own file, which may include associated styles and logic.

1. DB Folder:

The DB folder is used for storing data-related for users, such as users login info. It includes the firebase configuration details to connect to the database and be able to retrieve and write data from the database.  
It includes firebase.js which includes the database configurations and instantiation of the database.

1. App.jsx Component:

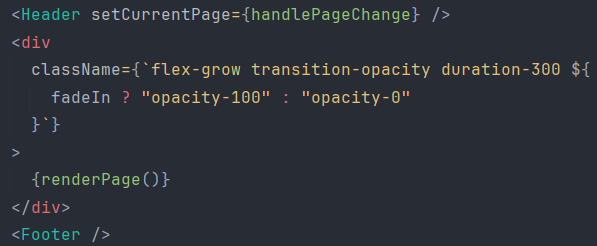
The App.jsx component serves as the root component of the React application. It typically contains the main application layout and hooks.  
Acts as the central hub for rendering other components and managing the application’s routing and state.

1. index.css:  
   The index.css file contains the tailwind CSS configurations applied throughout the application.
2. main.jsx Component:  
   The main.jsx file is the entry point for the React application. It renders the root component (App) into the DOM. It initializes the React application and attaches it to the <div id="root"></div> element in the index.html file. This file sets up the rendering process and may include configuration for React Router or global providers.

**package. json:**

The package.json file is a crucial part of any project. It contains metadata about the project, as well as scripts and dependencies required for development and production. It ensures that the project’s setup, configuration, and package management are consistent and reproducible across different environments.

**Components**

**Always Loaded Components:**

These components are always rendered on the screen, they are called by the App.jsx component across all the pages in the application. They are not affected by the user interactions in the application.

**Header.jsx:**

The Header.jsx component is responsible for rendering the website’s header, including navigation, dark mode toggling, and responsiveness features. It utilizes React hooks and conditional rendering to provide a dynamic and interactive user interface.  
The Header.jsx component provides a responsive and interactive navigation bar for the application. It features a logo, navigation links, dark mode toggle, and a mobile menu for smaller screens. This component ensures a consistent and user-friendly navigation experience across different devices and themes.

**Component Structure**

**Props**:

* setCurrentPage: Function to set the current page being displayed.

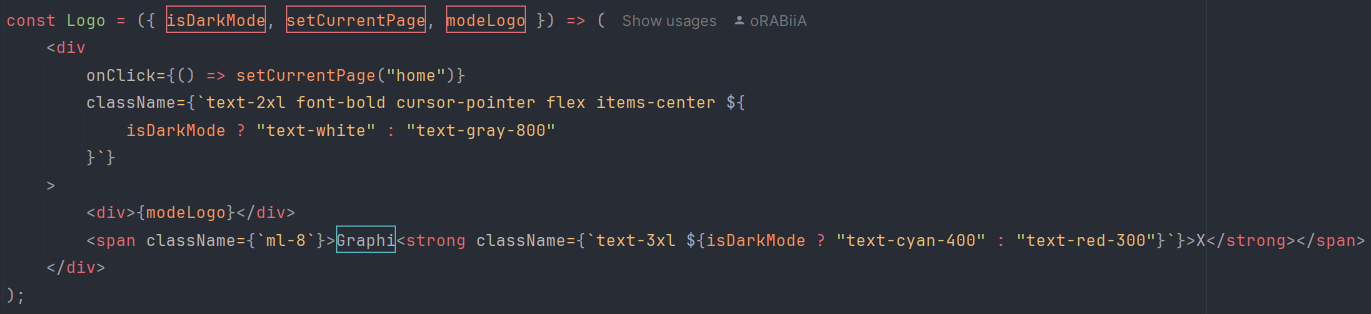
**State**:

* isMenuOpen: Boolean state to track if the mobile menu is open or closed.

**Logic:**

* isDarkMode: Determines if the current theme is dark.
* navItems: Array of navigation items.
* darkModeLogoSVG and lightModeLogoSVG: SVG elements representing logos for dark and light modes, respectively.

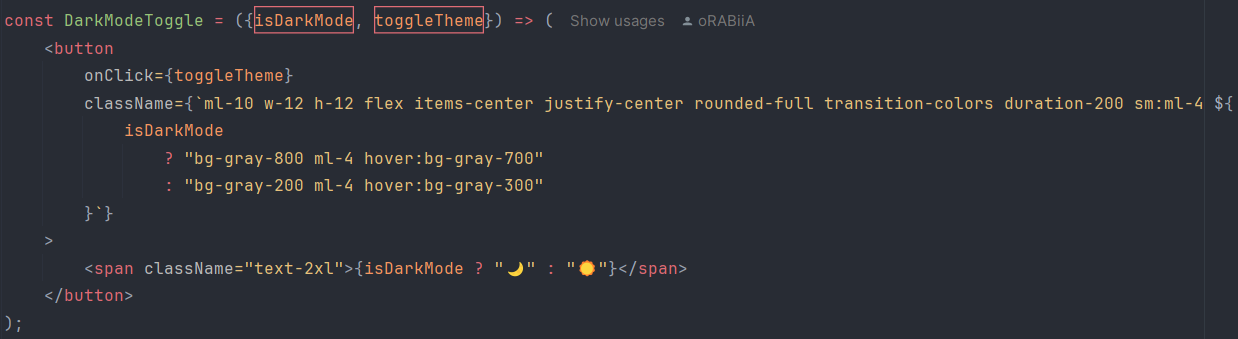
The Header.jsx component also features a Logo, NavItem, DarkModeToggle and MobileMenu components.

Logo component:

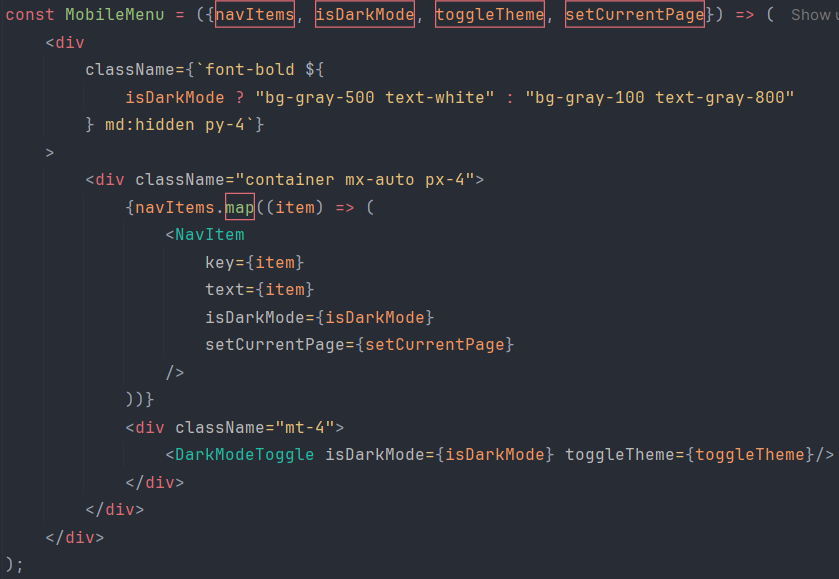
It renders the Logo inside the header which displays the "GraphiX" Logo as presented in the website.

NavItem component:

Is responsible for rendering the menu in the header which navigates to "Home", "About", "Contact" & "FAQ" pages.

DarkModeToggle component:

Is responsible for showing the button for toggling the Light/Dark mode in the header.

MobileMenu component:

Is responsible for rendering the header for mobiles or small screens when the screen size is small. It also compresses the navigation items in a hamburger element which shows the navigation items as well as the DarkModeToggle when it get pressed.

**Footer.jsx:**

The Footer component renders the footer section of the application. It includes social media links and a copyright notice. The appearance of the footer changes based on the current theme (light or dark mode) of the application.

**Component Structure**

**Props:**

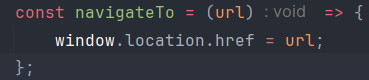
* No props are passed to this component.

**State:**

* No local state is used in this component.

**Functions:**

* navigateTo(url): Navigates to the specified URL by setting window.location.href.

  
  
Navigation to a URL inside the footer:

**Specific Page Components**

**About.jsx:**

The About component provides information about the platform, its features, mission, and team. It also includes an animated introduction to the platform using CSS animations and Tailwind CSS styling.

**Component Structure**

**Props:**

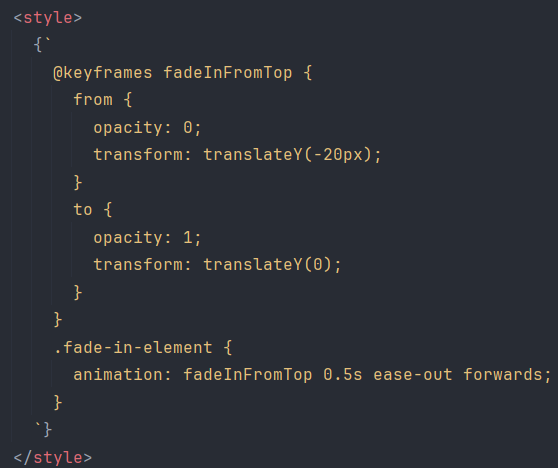
* setCurrentPage: A function to set the current page, although it's not used in this component.

**State:**

* No local state is used in this component.

**Functions:**

* navigateTo(url): Navigates to the specified URL by setting window.location.href.



fadeInFromTop: Custom keyframe animation that animates elements from opacity 0 and translation up to full opacity and no translation.

Each section of content has a fade-in animation with different delays to create a sequential appearance effect.

**Contact.jsx:**

The Contact.jsx component provides users with a contact form and additional contact information. It allows users to get in touch with the team for support, feedback, or inquiries. The component includes animations for a smooth introduction and adapts to the dark or light theme.

**Component Structure**

**Props:**

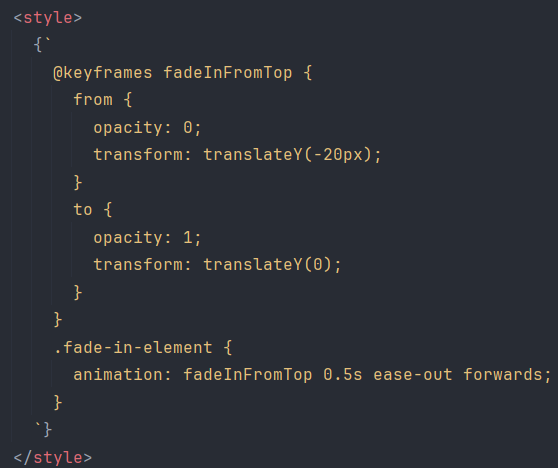
* setCurrentPage: A function to set the current page, used here to navigate to the FAQ page when the corresponding link is clicked.

**State:**

* No local state is used in this component.

**Functions:**

* navigateTo(url): This function is not used directly but the setCurrentPage function is utilized to navigate to the FAQ page.



fadeInFromTop: Custom keyframe animation that animates elements from opacity 0 and translation up to full opacity and no translation.

Each section of content has a fade-in animation with different delays to create a sequential appearance effect.

**CreatePage.jsx:**

The CreatePage.jsx component allows users to create and customize various types of charts (e.g., bar, line, pie) based on their selections. Users can choose the chart size, type, and color, input data, and labels for each chart. The component dynamically generates and displays the charts based on the provided configurations.  
The component renders a back button, dropdown menus for size and chart type selection, inputs for data and labels, color pickers, and buttons for creating and clearing charts. It also conditionally displays a chart size selector if all charts are of PDP type and renders the charts based on user inputs.

#### **Dependencies**

* useTheme from App.jsx for theme management
* BackArrow image for navigation
* Select component from react-tailwindcss-select for dropdown menus
* TEChart from tw-elements-react for rendering charts
* PropTypes for type checking
* React hooks: useState, useRef

#### **Props**

* setCurrentPage (Function): A function to update the current page in the application. It is used to navigate back to the upload page.

#### **Handlers**

* **handleChange(value)**: Updates the chart size and initializes additional selections based on the chosen size.
* **handleAdditionalChange(index, value)**: Updates the chart type for a specific chart.
* **handleDataInputChange(chartIndex, dataIndex, newValue)**: Updates the data input for a specific chart.
* **handleNameInputChange(selectIndex, inputIndex, value)**: Updates the label input for a specific chart.
* **handleClearButton()**: Clears all selections and resets the state.
* **handleCreateButton()**: Creates charts based on the selected configurations and scrolls to the charts section. Also determines if all charts are of PDP type.
* **handleBLRcolorChange(selectIndex, value)**: Updates the color for Bar, Line, and Radar charts.
* **handlePDPcolorChange(selectIndex, inputIndex, value)**: Updates the color for Pie, Doughnut, and Polar Area charts.
* **handleChartSizeChange(index, event)**: Updates the size of individual charts.

**FAQ.jsx:**

The FAQ.jsx component displays a list of frequently asked questions (FAQs) with expandable answers. It adapts its styling based on the current theme (dark or light mode) and includes animation effects for a smooth user experience.

**Component Structure**

**Props:**

* No props are used for this component.

**State:**

* No local state in this component.

**Functions:**

* **No specific functions**: Handles rendering and styling.

The component also renders a FAQItem component as follows:

FAQItem component:

A subcomponent used to render each individual FAQ item.

**Props:**

* faq (object): Contains question, answer, and icon for the FAQ item.
* isDarkMode (boolean): Indicates whether the dark mode is active.
* animationDelay (string): Animation delay for the fade-in effect.

**State:**

* isOpen (boolean): Determines whether the answer is visible or not.

**Functions:**

* **setIsOpen**: Toggles the visibility of the answer when the question is clicked.

**ChartPage Components**

**ChartHeader.jsx:**

The ChartHeader.jsx component displays a header for a chart with dynamic text based on the chart type. It includes separator lines and a label that adjusts its styling based on the provided mode (dark or light). It is intended to be used as a part of a chart visualization interface.

#### **Key Features**

* **Dynamic Chart Name**: Converts chart type from the data prop into a human-readable format.
* **Dynamic Styling**: Adjusts background and text color based on the mode prop (dark or light).
* **Visual Separators**: Includes horizontal lines on either side of the chart name for visual separation.

**Component Structure**

**Props:**

* data (array): An array of chart data objects. The type of the chart is derived from data[0].type.
* mode (string): The current theme mode, which affects the styling of the component. Expected values are "dark" or "light".

**Functions:**

* **chartName**: Used to get the human-readable chart name based on the type in data.

**ChartPage.jsx:**

The ChartPage.jsx component is a React functional component responsible for rendering a chart based on the provided data and allowing users to download the chart as an image. It uses the Chart.js library to generate various types of charts based on the chart type specified in the data prop.

**Props:**

* data (array, required): An array containing chart data. The first element of this array is expected to have the following properties:
  + type (string): The type of chart to be rendered (e.g., "bar", "line", "pie", etc.).
  + x (array): An array of labels for the chart.
  + y (array of arrays): An array where each element represents the dataset for the chart. Each dataset should correspond to x labels.
  + label (array): An array of labels for the datasets.
* setCurrentPage (function, required): A function to change the current page. It is expected to be called with the string "uploadPage" to navigate back to the upload page.

**Hooks:**

* useRef:
  + chartRef: Holds a reference to the Chart.js instance.
  + canvasRef: Holds a reference to the canvas element where the chart is rendered.
* useEffect:
  + Creates and destroys the Chart.js instance when the component mounts and unmounts, respectively. It also updates the chart when data or isDarkMode changes.

**Functions:**

* generateChart(): Determines the type of chart to be rendered and returns the configuration object based on the chart type.
* generateBarChart(), generateHorizontalBarChart(), generateLineChart(), generatePieChart(), generateDoughnutChart(), generatePolarChart(), generateRadarChart(), generateBubbleChart(), generateScatterChart(): Each of these functions generates the chart configuration for a specific type of chart. They customize the appearance of the chart based on the theme (dark or light mode).
* showChartNotSupported(): Returns a JSX element displaying a message when the chart type is not supported.
* downloadChart(): Allows users to download the rendered chart as a PNG image. It creates a temporary canvas, draws the chart on it, and triggers a download.

**Effect:**

* useEffect hook initializes the Chart.js instance with the chart configuration and sets the background color based on the theme. It also ensures that the chart instance is destroyed when the component unmounts.

**Home Components**

**ForgotPasswordCard.jsx:**

The ForgotPasswordCard.jsx component allows users to reset their password by first verifying their email and then updating their password. It handles the email submission and password update processes and provides feedback to the user through success and error messages. The component also includes a toggle for showing/hiding the password field.

**Props:**

* onClose (function): A callback function to close the forgot password card when the user clicks the "Close" button or after a successful password update.
* isDarkMode (boolean): A boolean indicating whether dark mode is enabled or not, affecting the component's styling.

**State:**

* email (string): Stores the email address entered by the user.
* newPassword (string): Stores the new password entered by the user.
* step (number): Indicates the current step in the password reset process (1 for email submission, 2 for password update).
* message (string): Stores error or success messages to display to the user.
* showSuccessCard (boolean): Controls the visibility of the success message card.
* showForgotPasswordCard (boolean): Controls the visibility of the forgot password card.
* showPassword (boolean): Toggles the visibility of the new password input field.

**Functions:**

* handleEmailSubmit (async function): Handles the email submission to check if the email exists in the database. If found, proceeds to the next step for password update; otherwise, displays an error message.
* handlePasswordUpdate (async function): Handles the password update process. Validates the new password length and updates it in the database. Displays a success message upon successful update.
* togglePasswordVisibility (function): Toggles the visibility of the new password input field between plain text and password input types.

It also includes a SuccessCard sub component:

A component that displays a success message indicating the password update was successful. Appears as an overlay with a "Close" button to hide it.

**Home.jsx:**

The Home.jsx component serves as the main entry point for users. It provides a login interface for users to sign in, options to navigate to Facebook or Twitter, and handles user registration and password recovery. The component conditionally renders different views based on whether a user is logged in or not and includes modals for registration, password recovery, and success messages.

**Props:**

* setCurrentPage (function): A function to change the current page view within the application.
* user (object): The current logged-in user object, or null if no user is logged in.
* setUser (function): A function to update the user state.

**State:**

* isPassword (boolean): Toggles between showing the password as plain text or obscured.
* inputValue (string): Stores the value of the password input field.
* email (string): Stores the value of the email input field.
* error (string): Stores error messages related to login and registration.
* showRegister (boolean): Controls the visibility of the registration modal.
* showForgotPassword (boolean): Controls the visibility of the forgot password modal.
* showSuccessMessage (boolean): Controls the visibility of the success message modal.

**Refs:**

* inputRef (ref): Reference to the password input field for toggling visibility.
* uploadButtonRef (ref): Reference to the upload button.
* footerRef (ref): Reference to the footer element.

**Effects:**

* Handles cleanup of user data on page unload and retrieves stored user data from session storage.
* Adds and removes an event listener for the beforeunload event to handle user logout.

**Functions:**

* handleLogout: Clears user data from storage and sets the user state to null.
* toggleVisibility: Toggles the visibility of the password input field.
* handleInputChange: Updates the state for the password input field.
* handleEmailChange: Updates the state for the email input field.
* navigateToFacebook: Redirects the user to Facebook.
* navigateToTwitter: Redirects the user to Twitter.
* handleLogin: Validates user credentials, retrieves user data from Firebase, and sets the user state or displays an error.
* handleRegisterSuccess: Closes the registration modal and shows the success message.

**RegisterCard.jsx:**

The RegisterCard.jsx component provides a user registration form that allows users to create an account by entering their name, email, password, and confirmation password. It includes input validation, toggles for password visibility, and interacts with Firebase Realtime Database to store new user data. The component also includes error handling and customizable styling based on the theme.

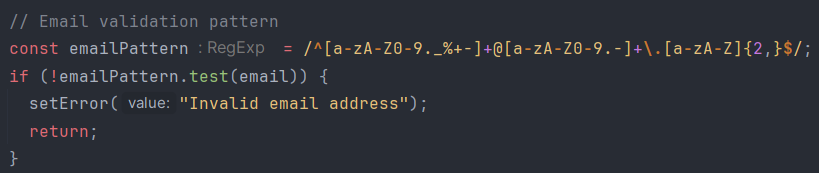
**Props:**

* onClose (function): A function to be called when the user closes the registration modal.
* isDarkMode (boolean): A boolean indicating whether dark mode is enabled, which affects the component's styling.
* onRegisterSuccess (function): A function to be called upon successful registration.

**State:**

* isPassword (boolean): Toggles the visibility of the password input field.
* isConfirmPassword (boolean): Toggles the visibility of the confirmation password input field.
* name (string): Stores the user's name.
* email (string): Stores the user's email address.
* password (string): Stores the user's password.
* confirmPassword (string): Stores the confirmation password.
* error (string): Stores error messages related to registration.

**Functions:**

* togglePasswordVisibility: Toggles the visibility of the password field.
* toggleConfirmPasswordVisibility: Toggles the visibility of the confirmation password field.
* handleRegister: Validates user input, checks for existing users in Firebase Realtime Database, and registers a new user if the input is valid using regular expressions.  
  Example:

**SuccessMessageCard.jsx:**

The SuccessMessageCard.jsx component displays a modal overlay with a success message and a close button. It is typically used to provide feedback to users, such as confirming successful actions (e.g., successful registration or operation completion).

**Props:**

* onClose (function): A callback function that is called when the close button is clicked. This function is used to handle the closing of the success message modal.
* message (string): The success message to be displayed within the modal.

**UploadPage Components**

**UploadPage.jsx:**

The UploadPage.jsx component is responsible for rendering the page where users can upload a JSON file to visualize their charts. It allows users to either upload a JSON file or create charts from scratch. The component also includes user-specific greeting and error handling for file uploads.

**Props:**

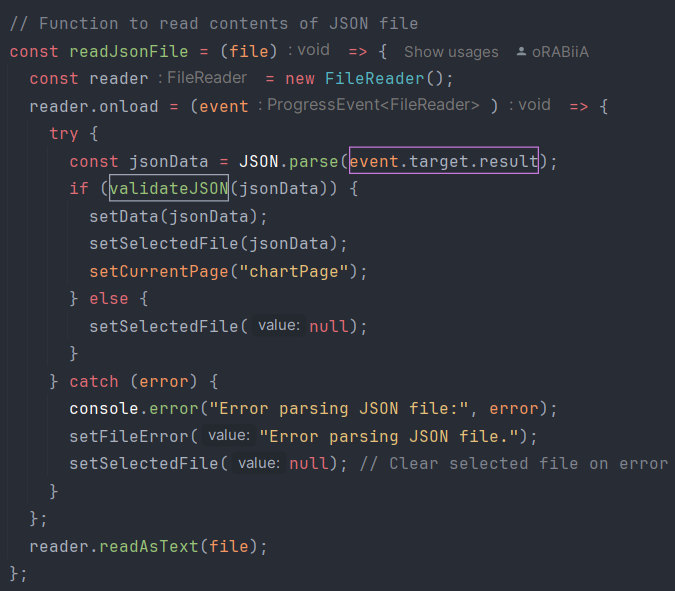
* setCurrentPage (function): A function to update the current page. Required.
* user (object): An object representing the user with the following properties:
  + name (string): The user's name. Required.
  + email (string): The user's email. Required.

**State:**

* selectedFile (object or null): Stores the selected file if valid, or null otherwise.
* fileError (string): Stores error messages related to file processing.

**Key Functions:**

* handleFileChange(event): Handles file selection, validates the file type, and reads the file content if it's a JSON file.
* readJsonFile(file): Reads and parses the JSON file, validates its structure, and updates the state or displays errors.
* validateJSON(jsonData): Validates the JSON data structure based on the type of chart and its requirements.

It parses the JSON file in the following way:

**InfoCard.jsx:**

The InfoCard.jsx component provides an informational card with collapsible sections (accordions) to guide users on how to upload different types of charts using JSON files. The component dynamically adjusts its styling based on the current theme (light or dark).

**Dependencies:**

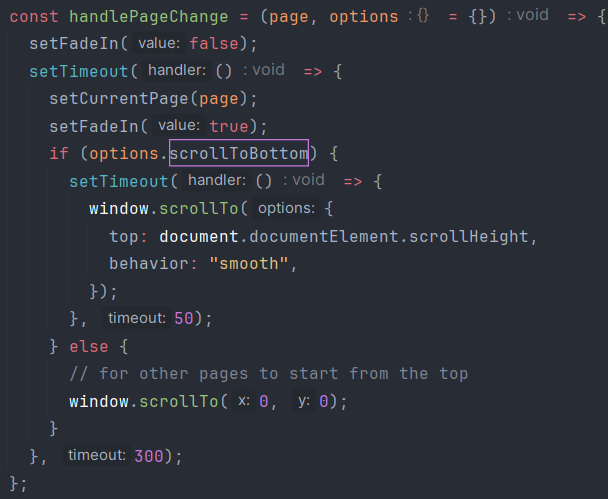
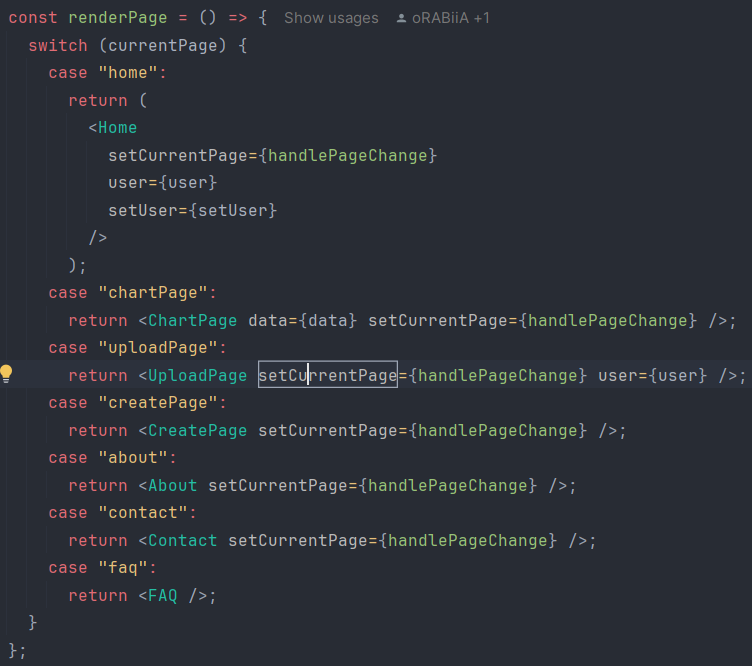
* useTheme from ../../App.jsx: Custom hook to get the current theme.
* useState from React: To manage the state of the active accordion item.
* TECollapse from tw-elements-react: A collapsible component to handle the accordion behavior.

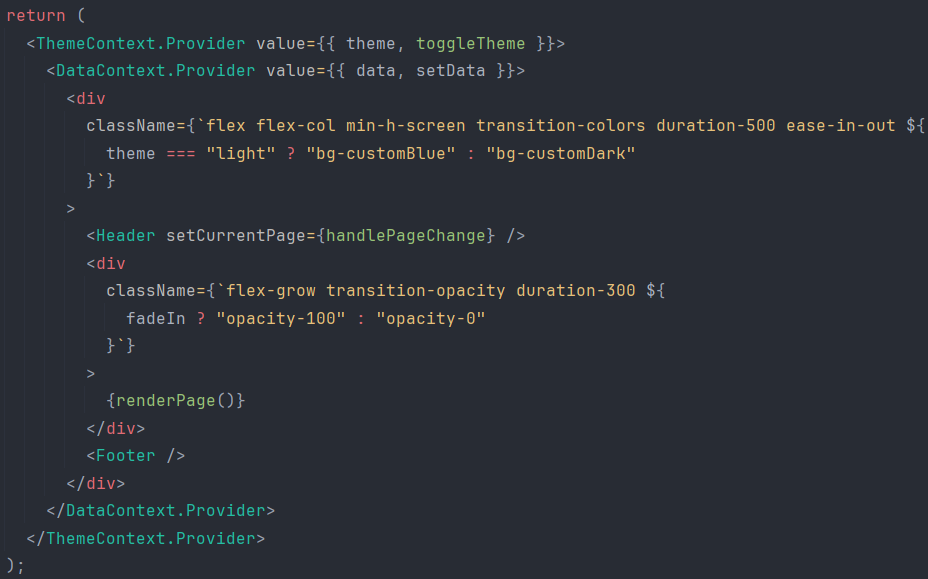
**Functional Behavior:**

* The component uses the theme value from the useTheme hook to apply theme-specific styles.
* The activeElement state tracks which accordion item is currently expanded.
* The handleClick function toggles the expansion state of an accordion item.

**Key Features**

**State Management for Page Rendering:**

Inside the App.jsx there is a state management functions which handles the rendering of the pages. The App.jsx Component renders as default the Home page to be the main page using "renderPage" function. In addition, the App.jsx sends as a prop to every other component a "handlePageChange" function which handles the switch to the requested page.



The renderPage function is called in the return statement of the App.jsx:

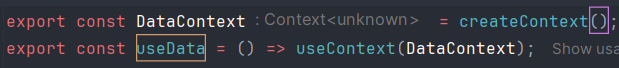
**Theme Context Provider:**

Manages and provides the current theme (light or dark) and a function to toggle the theme across the application.

Context Creation & Custom Hook:

**Data Context Provider:**

Manages and provides the data (e.g., chart data) and a function to update this data across the application.

Context Creation & Custom Hook: