Technical Design Document

Purpose

The purpose of this project is to create a calculator with basic arithmetic functions along with user authentication.

Requirements

The calculator should have a browser-based user interface, a number pad with digits 0-9, a decimal point, buttons for addition, subtraction, multiplication, and division, and a display that shows the input and the result of the calculation. Additionally, the calculator should follow the order of operations. The calculator should also have a way to sign up with a username and password and a way to log in with username and password. However, the calculator should be usable with or without authentication and the calculator should have a navigation bar or panel which displays the user's authentication status. In additional, the calculator show have memory functions (M+, M-, MR, MC), a percentage function (%), a square root function (√), an exponential function (^), a history function, a positive/negative function (+/-), and clear functions (AC, CE, C, Delete). Finally, the calculator should have saveable user sessions.

Implementation

This application was implemented using Next.js, one of the most widely known fullstack React frameworks and one of the few frameworks recommended for use in the React documentation. This framework allows for frontend and backend code to exist in a single repository for an easier deployment, and allows for simple page routing which is useful for navigating between the login, signup, and home pages. TypeScript is also used in order to ensure type safety throughout the program. Additionally, in the styles folder there is a single CSS file which uses TailwindCSS for styling.

This application also uses a PostgreSQL database to store user information and Prisma to act as an object relational mapper between the database and the application. The database contains one table called "users" which contains an auto incrementing, integer ID field used as a primary key, a unique string username field, and a password string field. Since SQL databases are ACID compliant, they help ensure that user information is saved correctly, and PostgreSQL is simply one of the more well-known SQL database options. Additionally, Prisma helps make working with databases safer, and assists with converting between database types and TypeScript types. The schema and a Prisma Client instantiation are both located in the Prisma folder of the code.

There are two API endpoints located in the pages/api folder called signup and login. The signup endpoint takes in a POST request with a username and password as part of the request body. The function checks that both the username and password fields were sent and non-empty and sends an error message back if otherwise. Then, the user's password is hashed using the bcrypt library by generating a salt and using the salt to create a hashed password for the user. This ensures the user's credentials are kept safe. It then stores the username and the hashed password in the database. If an error occurs, such as the username already existing in the database, then an error message is sent back to the user.

The login endpoint also takes in a POST request with a username and a password as part of the request body. It checks that the username and password were sent and non-empty and sends an error message back if otherwise.

Then it retrieves the user's information from the database. If the username is not found, an error message is sent back to the user. If the username is found, then bcrypt is used to check if the password sent in the request body matches the one stored in the database, sending an error message to the user if they do not match.

In the pages folder but outside of the api folder exist five files. The _app and the _document are both created when making a new Next.js project. The _app.tsx file is used as the root of all the components. The _document.tsx file contains the HTML structure of the application. The other three files are used to define the three pages found within the application.

The signup.tsx file is used for the signup page. When this page first renders, a useEffect hook is used to check if a user session exists in local storage. If a user session exists, the user is taken directly to the home page. Otherwise, they stay on the signup page. This page contains an input for the username, an input for the password, and a button to send a POST request to the API for signup. If signup is successful, the user's session is stored in local storage and the user is taken to the homepage. Local storage is used because it is easy to access and make changes to and only the username and some session data related to the calculator application itself are stored in it, but no other confidential information. If signup fails, an error message is shown beneath the button.

The login.tsx file is used for the login page. It functions exactly the same as the signup page except it sends a POST request to the API for login rather than signup when the button is pressed.

The index.tsx file is used for the home page. This is where the calculator is located. When the page renders, a useEffect hook is used to retrieve the user's session from local storage if it exists. At the top there is a navigation bar that shows the title of the page in the center. To the left it states whether the user is currently signed in or not. To the right it shows login and signup buttons if the user is not signed in, or a sign out button if the user is logged in. When the user signs out, their session is removed from local storage.

Beneath the navigation bar is the user interface for the calculator. The calculator uses JavaScript's built-in eval() function to perform the mathematical operations inputted into the calculator and follow the order of operations. Logic is implemented to ensure that any arguments passed into the eval() function are valid, such as making sure the input only has at most one decimal point, no operators side by side (e.g.. 2++ is not a valid expression), no leading zeros, and operators requiring numbers to the left and right of them. Every time a button is pressed, the value state and display state are updated. The value state is the state of the current number being inputted and the display state is the full expression being displayed on the calculator, including both numbers and operators. For example, if the user inputs 24 + 40, then the value state would be 40 but the display state would be 24 + 40. For the memory functions, a memory state is used and initialized to be 0 and changes depending on the buttons the user presses. The positive/negative function, square root function, and exponential function only apply to the number stored in the value state. However, the exponential function is treated similarly to other operators like addition which require numbers both to the left and right of it. The AC button clears the screen and resets the memory state back to 0, the C button clears the screen, the CE button removes the current operand, and the DEL button removes the last digit or operator inputted. Expressions are evaluated with the eval() function whenever the = button is pressed. Additionally, whenever the = button is pressed, the expression is saved into a history state which is an array that can contain up to ten expressions, which removes the oldest one to add new ones once it reaches ten expressions. The user can navigate their history using the buttons to the right of the screen. Whenever the = button or a memory function button are pressed, the user's session is saved to local storage, which includes their username, the value state, the display state, the memory state, and the history state.