📝 Design Doc

Overview

The goal of this project is to develop a calculator application with modern React-based frontend technology. The application should be capable of performing basic arithmetic operations and support additional functionalities like percentage calculations, memory management, history tracking, and date-specific history retrieval. User authentication is also considered to enable history saving features.

⚙️ Backend

Introduction

This backend service for the calculator application is built using NestJS. The architecture focuses on modularity, testability, and adherence to modern software engineering principles. NestJS encourages good architecture practices through its own built-in modularity features. The backend is organized into different modules to isolate concerns:

- **User**: Manages user data and user-related functionalities.
- **Authentication**: Handles JWT-based user authentication.
- **Calculations**: Deals with the calculator's operations storing and fetching logic.

Technologies Used

- **NestJS**
- **Mongoose**
- **JWT for Authentication**

Module Structure

Each module in the backend lives in its own folder which contains various related files:

- *.module.ts: The main module file that brings all the related code together.
- *.controller.ts: Handles incoming requests and returns responses.
- *.service.ts: Contains the business logic and calls the repository.
- *.repository.ts: Directly interacts with the database.
- *.schema.ts: Defines the Mongoose schema for the database collection.
- *.dto.ts: Data Transfer Object - An object that carries data between processes.
- *.entity.ts: A TypeScript class that defines the object and type structure for the database records.
- *.spec.ts: Spec files for testing. They are segregated in the same way as the other layers/files, so you should add one .spec for your service, another one for your controller and so on.

Why Use Entity?

The *.entity.ts is particularly important as it defines the type returned by the repository layer model methods. This avoids "leaking" database specific types to other layers, thereby reducing the coupling to the database provider.

🖥️ Frontend

Introduction

Inspired by the iPhone's calculator design, this project aims to develop a calculator application using modern NextJS/React-based frontend technology and a NestJS backend. The application performs basic arithmetic, percentage calculations, memory operations, and tracks user history. It provides a user-friendly UI and efficient functionality.

Tech Stack

- **Framework**: React.js
- **Language**: TypeScript
- **State Management**: React Hooks (useReducer, useState, useEffect)
- **Other Libraries**: big.js, react-query

Folder Structure Rationale

The folder structure is organized to isolate different aspects and responsibilities of the application:

- **components**: Holds reusable UI elements, further broken down by feature area (e.g., calculator, navbar).
- **contexts**: Manages global state and provides shared functionality.
- **hooks**: Contains custom React Hooks for managing local component states.
- **pages**: Holds the application's main views.
- **services**: Contains API calls.

Main Contents

- **Components**
  - **Calculator**: Root Calculator component
    - **Header**: Displays previous operations or results
    - **Body**: Button interface for digits and operations
    - **History**: Displays past calculations
  - **Navbar**: Navigation bar
  - **Layout**: Main layout wrapper
- **Contexts**
  - **AuthContext**: Manages authentication state
- **Hooks**
  - **useCalculator**: Manages calculator state and logic

Testing

- Tests are available and can be executed using `yarn test`.