Real-Time Stock Watchlist Web Application

By Pranjal Sinha

Problem Statement

Round Requirements

Participants are required to select one of the following apps:

1. Upstox
2. Angle One
3. Groww

The task is to create a dummy application showcasing a single feature from the chosen app, using the specified tech stack:

- Frontend: Flutter and Dart
- Backend: TypeScript
- Database: PostgreSQL

Selected App

Groww has been selected for this project. The implemented feature in the dummy application is the ability to add and view stocks in a watchlist.

Application Overview

The Real-Time Stock Watchlist application allows users to manage and monitor stock prices in real-time. The frontend, built with Flutter and Dart, communicates with a TypeScript backend that interfaces with a PostgreSQL database. Docker is used for containerizing the backend service.

Technology Stack

Frontend

- Framework: Flutter
    - Purpose: Develops cross-platform applications with a single codebase.
    - Advantages: Rich set of pre-designed widgets, hot reload for quick iteration.
- Language: Dart
    - Purpose: Programming language for building Flutter applications.
    - Advantages: Strongly typed, supports asynchronous programming.

Backend

- Framework: Express.js
    - Purpose: Simplifies building RESTful APIs and handling HTTP requests.
    - Advantages: Minimal and flexible, integrates well with middleware.
- Language: TypeScript
    - Purpose: Adds static type definitions to JavaScript, improving code quality and maintainability.
    - Advantages: Type safety, enhanced IDE support, better refactoring.
- Database: PostgreSQL
    - Purpose: Stores and manages application data.
    - Advantages: ACID compliance, robust performance, advanced features like JSONB.

Dependencies

Frontend Dependencies

- Flutter SDK
    - Version: 3.7.0 (or latest stable version)
    - Purpose: Provides the framework and tools for building Flutter applications.
- Dart
    - Version: 3.7.0 (or latest stable version)
    - Purpose: Language used for writing Flutter code.
- Dependencies List (from `pubspec.yaml`):
    - `flutter`: Core Flutter framework package.
    - `provider`: State management solution for efficient state handling.
    - `http`: For making HTTP requests to the backend.
    - `flutter_test`: Provides testing capabilities for Flutter applications.

Backend Dependencies

- Node.js
    - Version: 18.x (or latest LTS version)
    - Purpose: JavaScript runtime for server-side code execution.
- Express.js
    - Version: 4.18.2 (or latest stable version)
    - Purpose: Framework for handling HTTP requests and routing.
- TypeORM
    - Version: 0.3.7 (or latest stable version)
    - Purpose: ORM for managing database entities and interactions.
- PostgreSQL
    - Version: 15.0 (or latest stable version)
    - Purpose: Relational database for storing application data.
- Dependencies List (from `package.json`):
    - `express`: Web framework for Node.js.
    - `typeorm`: ORM for interacting with PostgreSQL.

- `pg`: PostgreSQL client for Node.js.
- `typescript`: Superset of JavaScript for adding types.
- `ts-node`: TypeScript execution engine for Node.js

Flutter Modules

- `main.dart`
  - Purpose: Entry point for the Flutter application, setting up the app and routing.
  - Details: Initializes the app, sets up routing, and handles navigation.
- `widget_test.dart`
  - Purpose: Contains tests to ensure Flutter widgets render and behave correctly.
  - Details: Includes unit tests for UI components to verify functionality and appearance.
- Modules List (from `pubspec.yaml`):
  - `widgets`: Custom widgets created for the application's UI.
  - `screens`: Different pages or screens within the application.
  - `models`: Data models representing application entities, such as stocks.
  - `services`: Functions or classes responsible for managing API calls and application logic.

Software Versions

Frontend

- Flutter: Version 3.7.0 (or latest stable version)
- Dart: Version 3.7.0 (or latest stable version)

Backend

- Node.js: Version 18.x (or latest LTS version)
- Express.js: Version 4.18.2 (or latest stable version)
- TypeORM: Version 0.3.7 (or latest stable version)
- PostgreSQL: Version 15.0 (or latest stable version)

Features

1. Add Stocks: Allows users to add new stocks to their watchlist.
    - Details: Users can input stock name and price, which are then saved in the database.
2. View Stocks: Displays the user's watchlist with the current prices of the added stocks.
    - Details: Retrieves stock data from the database and displays it in a list format.

Architecture

Frontend

The frontend is developed using Flutter, which communicates with the backend through REST API calls. It handles user interactions and displays stock data.

Backend

The backend, built with Express.js and TypeScript, handles business logic and interacts with the PostgreSQL database using TypeORM. It processes requests from the frontend, performs CRUD operations, and sends responses.

TypeORM Overview

TypeORM provides a TypeScript-based API for interacting with PostgreSQL, simplifying CRUD operations and entity management.

Database Schema

The PostgreSQL database schema includes a `stock` table with columns for stock ID, name, and price. This schema supports efficient storage and querying of stock data.

Backend and Database Integration

Database Configuration

1. Database Connection:
    - The backend server connects to PostgreSQL using TypeORM configuration settings for host, port, username, password, and database name.
2. Entity Management:
    - Entities: Represent database tables as TypeScript classes.
    - Repositories: Provide methods for performing CRUD operations on entities.
3. Data Operations:
    - Creating Records:
        - Pseudocode:
        - `function createStock(stockData): connectToDatabase() executeSQL("INSERT INTO stock (name, price) VALUES (stockData.name,`
    - Retrieving Records:
        - Pseudocode:
        - `function getStocks(): connectToDatabase() result = execute`
        -
        - `SQL("SELECT * FROM stock") return result`

Backend-Frontend Communication

RESTful API

1. API Definition:
    - The backend exposes endpoints for adding and retrieving stocks.
2. Request Handling:
    - The backend processes HTTP requests from the frontend, performs the required operations, and sends JSON responses.
3. Response Formatting:
    - Responses include status codes and data to indicate the result of operations.
4. Example Workflow:
    - Frontend:
        - Action: Send a request to add a new stock.
        - Pseudocode:
        - `request = createHTTPRequest("POST", "http://localhost:3001/stocks", stockData) response = sendHTTPRequest(request) displayResponse(response)`
    - Backend:
        - Action: Process the request and interact with the database.
        - Pseudocode:
        - `function handleRequest(request): stockData = parseRequestBody(request) result = addStockToDatabase(stockData) response = createResponse(result) sendResponse(response)`
    - Database:
        - Action: Insert the stock data into the database.
        - Pseudocode:

- `function addStockToDatabase(stockData): connectToDatabase() executeSQL("INSERT INTO stock (name, price) VALUES (stockData.name, stockData.price)") return successStatus`

Error Handling and Validation

1. Error Handling:
    - The backend includes mechanisms to catch and handle errors, sending appropriate status codes and messages to the frontend.
2. Data Validation:
    - Ensures incoming data is valid before processing or saving it to the database, reducing the risk of data inconsistencies.

Running the Application

Frontend

1. Install Flutter and Dart SDK.
2. Run `flutter pub get` to install necessary dependencies.
3. Start the application with `flutter run` to see the app in action.

Backend

1. Ensure Docker is installed and running.
2. Navigate to the project directory.
3. Start the backend service and PostgreSQL database.

Troubleshooting

Common Issues

- Flutter Tests: If encountering issues, ensure all dependencies are listed in `pubspec.yaml` and are up to date. Check for any unresolved package versions or compilation errors.
- Database Connection: Ensure the backend service can connect to the PostgreSQL database.

Future Enhancements

1. User Authentication: Adding authentication to manage watchlists for different users securely.
2. Real-Time Updates: Implement WebSocket support for live updates on stock prices.
3. Enhanced UI: Expand the frontend UI to include features like stock charts, historical data, and improved user experience.