Project Overview (Time spent 7h)

This project is a Flutter application that displays real-time data for both stock and cryptocurrency symbols. It fetches initial stock quotes via a REST API and updates their prices in real time using WebSocket connections. The user can switch between viewing stock and cryptocurrency data, with the UI reflecting the appropriate symbol data based on the selection.

Note: I was trying to pick up the flutter and dart on the fly and learn as I go. I mostly use React and Swift as my languages of choice. In this exercise, I tried my best to combine my experience and new learnings together to get to the final result.

Design Decisions

1. **Separation of Concerns:**
    - **Controller Layer:** The application employs a `HomeController` to manage business logic, separating it from the UI layer. This controller handles data fetching, WebSocket connections, and symbol subscription management.
        1. For data fetching stock_quote_service was created
        2. For socket connection, symbol_wss_service was created
    - **Presentation Layer:** The UI is composed of modular, reusable widgets such as `SymbolsView` and `PillsView`. These widgets are responsible for rendering the UI based on the data provided by the controller.
2. **Widget Composition:**
    - The UI is broken down into smaller widgets to promote reusability and maintainability. For instance, `SymbolsView` is a widget that handles the display of symbols, while `CircleRoundAvatar` is responsible for showing an avatar image or symbol's initials.
    - This modular approach allows easy testing and isolation of specific UI components.
3. **State Management:**
    - **Stateful Widgets:** The application uses `StatefulWidget` to manage local state within the `HomePage`. This approach is sufficient given the scale of the application, where state changes are limited to UI updates based on user interactions or incoming WebSocket data.
    - **Controller:** The `HomeController` manages the application's core logic, including fetching data and handling WebSocket connections. This controller is responsible for updating the UI by providing data to the `StatefulWidget`, ensuring a clear separation between UI and logic.
4. **Error Handling:**
    - The application includes basic error handling, particularly in the data fetching process. If an error occurs during the initial data fetch or WebSocket connection, the UI displays an error message with an option to retry.

State Management Approach

The state management in this project is handled primarily through `StatefulWidget` and a `Controller` (specifically, `HomeController`). Here's how state management is organized:

- **StatefulWidget:**
  - The `HomePage` widget is a `StatefulWidget` that manages the state for displaying stock and cryptocurrency data. It holds the state for loading indicators, error messages, and the actual symbol data.
- **Controller (HomeController):**
  - The `HomeController` acts as the business logic layer (can be further improved). It manages WebSocket connections and data fetching from the API. The controller updates the UI by passing data to the `StatefulWidget` through callback functions.
  - The controller also manages subscriptions to stock and cryptocurrency symbols, ensuring that only the necessary data is fetched and displayed.

Data flow

On the initial load, I decided to have a single API call to get the symbols before connecting to the data stream.
This makes the experience more user-focused and gives time for the connection to get established in the background. It also helps in times when the **stock market is closed**.

Because of the **limited data availability** on the stream, the first initial API call also gives me the percentage price change of the day that can be displayed in green or red color to indicate the trend movement.

I also added subscription and unsubscription to symbols **on demand**.
When “crypto” tab is chosen I unsubscribe from stock symbols and subscribe to crypto symbols.
The same happens if “stocks” tab is chosen.

I decided to keep the connection open at all times to avoid any potential delays and performance issues. The only time when the connection is closed is when the home page is disposed of.

Third-Party Libraries Utilized

1. **`web_socket_channel`:**

    - This library is used for managing WebSocket connections. It handles real-time updates for stock and cryptocurrency prices.
2. `http`:
    - This library is used for making HTTP requests. It handles the communication with external APIs.

Key Components

- **HomePage:** The main screen where users can toggle between stock and cryptocurrency views.
- **SymbolsView:** A widget that displays a list of symbols (either stocks or crypto) with their respective prices and percentage changes.
- **PillsView:** A widget that provides a UI for toggling between the stock and cryptocurrency views.
- **HomeController:** Manages the core business logic, including fetching data, managing WebSocket connections, and updating the UI.

Future Improvements (If I had more time)

- **More separation of concern:** I would separate the business logic more as a domain layer. The idea is to have the main logic independent from the tools and data sources. This creates a scalable and cost-effective solution that is easier to change and scale.
- **Advanced State Management:** For larger applications, it would be better to use more advanced state management solutions such as `Bloc` to handle complex state scenarios and make the application more scalable.
- **More Testing:** Good test coverage is important for many reasons. Some of them would be confidence in code, regular releases, and automated CI/CD.
- **Caching:** Implementing caching mechanisms for the initial stock data to reduce network calls and improve the app's performance.
- **Error Reporting:** Enhance error handling by integrating error logging and alerts for production monitoring.