Socket Chat Documentation

Version 1.0.0

Sabri Başoğlu

November 8, 2023

Contents

List of Code Listings

1 Introduction

The Socket Chat application is a real-time messaging platform developed using the Flutter framework. It facilitates instant communication through different chat rooms, employing Socketsbay's websocket service as the underlying messaging back-end. This app aims to provide a seamless and efficient chatting experience with an emphasis on simplicity and ease of use.

2 Key Features

- User Authentication: Users can choose and set a preferred username to identify themselves across chat rooms.
- Chat Room Management: Users can join between different chat rooms as per their interests or group affiliations.
- Real-Time Communication: Leveraging websockets, this app offers an authentic real-time messaging experience where users can see messages as they are being typed and immediately when they are sent.
- User Interface: A clean and intuitive interface ensures that users can navigate the app effortlessly and focus on their conversations.

3 Architecture

The Socket Chat app is built with a Model-View-ViewModel (MVVM) architecture, streamlined by the use of the 'stacked' package in Flutter. 'stacked' offers a structured approach to applying MVVM principles, enabling developers to focus on modular and maintainable code. It handles dependency injection, state management, and provides utilities for routing and service location. This allows for clean separation between user interface (Views), business logic (Models), and data presentation (ViewModels). The architecture ensures that each component is independently testable and easy to understand, facilitating agile development and future scalability.

- **Model layer** (Business Logic) handles data and state management, encapsulating the logic of how the application works. This includes managing websocket connections for real-time chatting, user authentication, and operations related to chat rooms and messages.

- **View layer** (User Interface) is solely responsible for the presentation and UI components, ensuring a seamless and responsive user experience. The ‘stacked‘ package ensures that the Views remain declarative and concise, reacting to changes in the ViewModel.
- **ViewModel layer** serves as an intermediary by binding the UI with the business logic. It reacts to user actions, retrieves/presents data from/to the Model layer, and triggers updates on the View layer.
- **Service layer** In addition to the MVVM structure, the application includes a robust Services Layer, which leverages the Singleton design pattern to manage shared resources and services. These Singleton services ensure that there is only one instance of each service throughout the app, providing centralized and efficient management of core functionalities such as networking, user preferences, and real-time communication. The ‘stacked‘ package facilitates the use of these services with its built-in dependency injection mechanism, which establishes and retrieves service instances as needed across different parts of the application. This approach enhances the modularity and reusability of the codebase.

4 Libraries and Dependencies

- stacked: Provides the foundational architecture for MVVM and dependency injection.
- stacked_services: A set of utilities for navigation, dialogue management, and bottom sheet functionality.
- cherry_toast: A package to show customizable toast notifications.
- web_socket_client: Enables real-time communication using websocket protocol.
- shared_preferences: Allows easy access to the device’s local storage for storing key-value pairs.
- build_runner: A powerful tool for generating source code files.
- flutter_test: Provides a testing framework to write unit tests for Flutter apps.
- flutter_lints: Ensures the code follows Dart’s and Flutter’s recommended lint rules.

- mockito: A Dart package that provides a way to mock dependencies in tests.
- stacked_generator: Generates code for dependency injection and routing based on stacked annotations.

5 Setup and Configuration

Detailed instructions on how to set up the development environment and run the application.

5.1 Clone the Repository

```
git clone https://github.com/sabreys/socket_chat.git
```

5.2 Install Dependencies

```
flutter pub get
```

5.3 Run the App

```
flutter run
```

6 Services

The Socket Chat app employs a Services Layer that consists of several key services, each responsible for a specific set of operations. Below is a detailed description of each service and its role within the app:

6.1 Cache Service

The Cache Service is responsible for storing messages received through the WebSocket connection on the device. This ensures that messages are preserved and can be retrieved at any time, preventing data loss and allowing users to access their chat history persistently.

6.2 Channel Service

The Channel Service keeps track of the details of various chat rooms or channels. It monitors the active channels and manages any updates or changes to them, ensuring the app's chat room data remains current and synchronized.

6.3 Socket Service

At the core of the real-time communication feature of the app is the Socket Service. This service handles WebSocket connections, managing the sending and receiving of messages between the client and the server. It is crucial for the app's ability to offer live interactions among users.

6.4 UserService

The UserService maintains the active user's information, providing convenient access to the user's data throughout the app. It holds details such as the username and any preferences or settings associated with the user, facilitating personalized experiences.

7 Chat Application Flowchart

Figure 1 presents the application's flowchart illustrating the complete workflow.

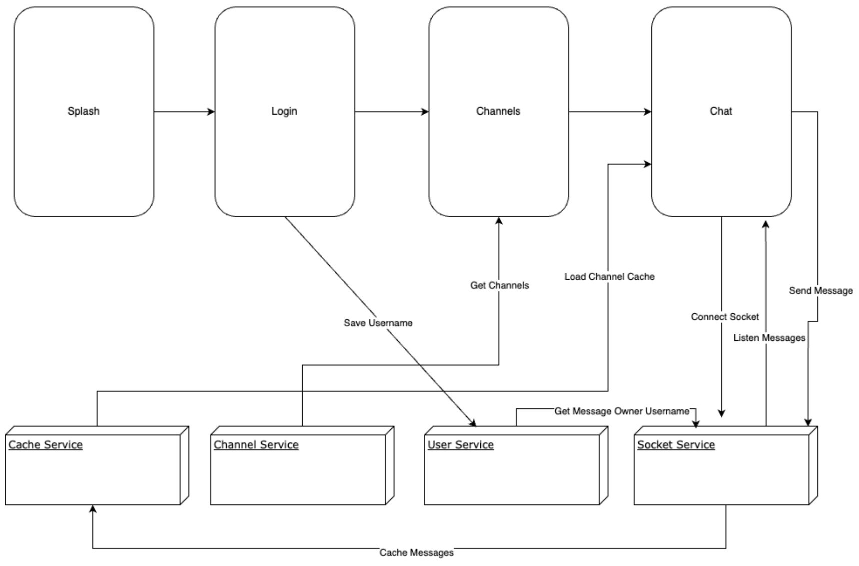

Figure 1: Application flowchart.