Frontend API Documentation

Overview

The frontend component of the BeeBot system serves as the primary interface layer for user interaction, implemented as an Android application using Java and XML. This component acts as an intermediary between the system's backend AI functionalities and end-user input, providing a seamless, accessible, and efficient communication medium. Core features include:

- **Multi-modal Input Support:** Facilitates both text and voice-based interaction.
- **Dynamic Feedback Mechanisms:** Includes real-time response rendering and error handling.
- **Accessibility Enhancements:** Offers features such as live text responses and text-to-speech conversion to support diverse user needs.

The frontend emphasizes usability, inclusivity, and seamless integration with the backend system, ensuring a highly responsive and user-friendly experience.

Installations and Setup

Prerequisites

To develop, test, and deploy the application, the following tools and libraries are required:

Tools

- **Android Studio:** Integrated Development Environment (IDE) for Android development.
- **Gradle:** Dependency management and build automation system.
- **Android SDK:** Essential libraries and tools for Android app development.

Core Libraries

- **Google Play Services:** Enables integration with Android’s ecosystem.
- **Android SpeechRecognizer:** Supports voice recognition functionality.
- **Android TextToSpeech:** Converts textual responses into audio output for enhanced accessibility.
- **Retrofit:** A type-safe HTTP client for API communication.
- **Glide:** Efficient image loading and caching library for optimized app performance.

Comprehensive Environment Setup

1. **Install Android Studio**:
    - Download Android Studio from the official website.
    - Ensure the latest version of the Android SDK is installed.

2. **Import the Project**:
    - Clone the chatbot repository or download the source code.
    - Open the project in Android Studio.

3. **Configure Gradle**:
    - Allow Gradle to resolve dependencies automatically. Ensure a stable internet connection during the process.

4. **Set the API Base URL**:
    - Update the base URL in `ApiClient.java` to match the backend server address:

      ```
      private static final String BASE_URL =
      "https://api.mannychatbot.com/";
      ```

5. **Run the Application**:
    - Connect an Android device or use an emulator.
    - Build and deploy the application using Android Studio.

Detailed File Structure

- **MainActivity.java**: Governs user interactions, API requests, and the overall application flow.
- **ChatAdapter.java**: Manages the rendering of chat bubbles for user and AI responses.
- **VoiceHelper.java**: Implements speech-to-text and text-to-speech functionalities for voice interactions.

Resource Files

- **XML Layouts**: Define the structural and visual components of the application, including the main chat interface, user settings, and accessibility controls.
- **res/drawable**: Contains graphic assets such as icons, buttons, and backgrounds.
- **res/values**: Centralizes resource definitions like strings, dimensions, and themes to promote consistency and scalability.

Data Flow

The frontend application establishes communication with the backend via HTTPS requests, enabling secure data exchange. User input, primarily in the form of text, is transmitted from the frontend to the backend. The backend processes this input by invoking the ChatGPT API to generate responses. These responses are then returned to the frontend over the same secure channel, where they are rendered and presented within the user interface. This architecture ensures efficient and secure data handling between the frontend and backend systems.

Frontend API Integration

Interaction Workflow

The frontend uses Retrofit to manage HTTP-based interactions with the backend. The interaction flow includes:

1. Capturing user input (text or voice).
2. Transmitting the input to the backend for processing.
3. Displaying AI-generated responses in the chat interface.

Key Classes and Methods

ApiClient.java: Configures Retrofit for API communication.

```
public class ApiClient {
      private static Retrofit retrofit;
      public static Retrofit getClient() {
            if (retrofit == null) {
                  retrofit = new Retrofit.Builder()
                     .baseUrl(BASE_URL)
                     .addConverterFactory(GsonConverterFactory.create())
                     .build();
            }
            return retrofit;
      }
}
```

ChatService.java: Defines backend interaction endpoints.

```
public interface ChatService {
    @POST("/chatbot")
    Call<ChatResponse> sendMessage(@Body ChatRequest request);
}
```

MainActivity.java: Manages user interaction and API requests.

```
private void sendMessage(String message) {
    ChatRequest request = new ChatRequest(message);
    ChatService service =
ApiClient.getClient().create(ChatService.class);
     service.sendMessage(request).enqueue(new Callback<ChatResponse>()
{
          @Override
          public void onResponse(Call<ChatResponse> call,
     Response<ChatResponse> response) {
               if (response.isSuccessful()) {
                    displayMessage(response.body().getAiResponse());
               } else {
                    displayError("Error in fetching response.");
               }
          }
          @Override
          public void onFailure(Call<ChatResponse> call, Throwable t) {
               displayError("Connection error: " + t.getMessage());
          }
     });
}
```

Advanced Error Management

The application implements robust error-handling mechanisms to maintain reliability:

400 Bad Request: Indicates invalid user input.

```
{ "error": "User input is empty" }
```

500 Internal Server Error: Highlights backend processing failures.

```
{ "error": "Unexpected server error" }
```

Network Errors: Provides actionable feedback for connectivity issues:

```
displayError("Network unavailable. Please check your connection.");
```

Expanded Functional Capabilities

Initialization and Setup:

Core components like `SpeechRecognizer` and `TextToSpeech` are pre-initialized at app startup for immediate use.

Input Management:

Input validation and sanitization ensure error-free processing of both text and voice inputs.

Rendering Responses:

Custom adapters and `RecyclerView` dynamically render chat responses, ensuring an engaging user experience.

Error Resilience:

Fallback mechanisms maintain application functionality during offline scenarios or backend outages, with clear notifications guiding users through troubleshooting steps.

Thread Management:

Use of background threads for heaver operations like text-to-speech processing

User Experience (UX) Features

Ensures intuitive user experience:

- **Session History:** Retains chat history locally to allow users to send the chat log to their email.
- **Contextual Responses:** Allows for the use of a context file to cater the answers to the location of the kiosk
- **Intuitive Buttons:** Colored and symbolized buttons to indicate navigation without the need to read.

Future Enhancements

Future development efforts will focus on enhancing the frontend's versatility and user experience. Plans include implementing multi-company support with a modular UI architecture, allowing the application to be easily customized for various business needs. Multi-language support will be introduced to cater to a global user base, enabling seamless interaction across diverse linguistic audiences. Scalable text functionality will be enhanced to ensure readability

and accessibility across all device types and user preferences. Additionally, the integration of more robust animations will elevate the application's visual engagement, providing a smoother and more dynamic user experience. These enhancements aim to create a highly adaptable, inclusive, and visually compelling platform.

References

https://www.javatpoint.com/android-texttospeech-example
https://developer.android.com/reference/android/speech/tts/TextToSpeech
https://developer.android.com/reference/android/speech/SpeechRecognizer
https://leonardloo.medium.com/android-studio-detailed-introduction-1be9753a3230