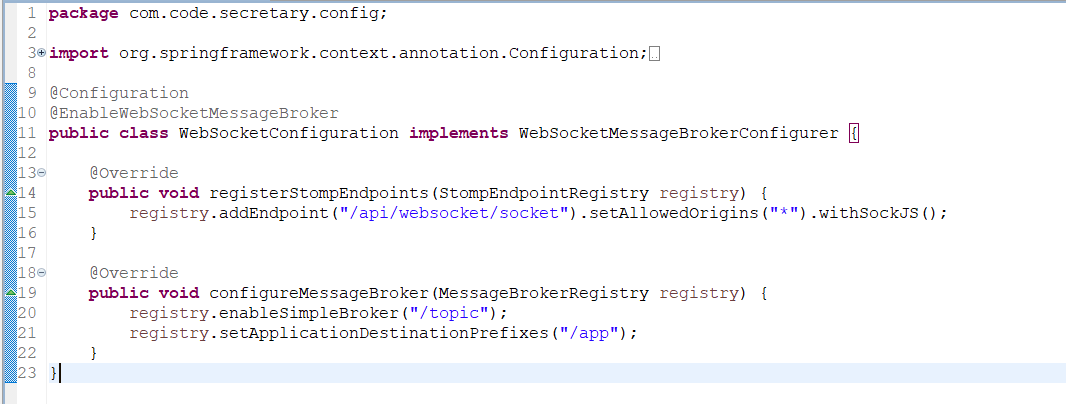
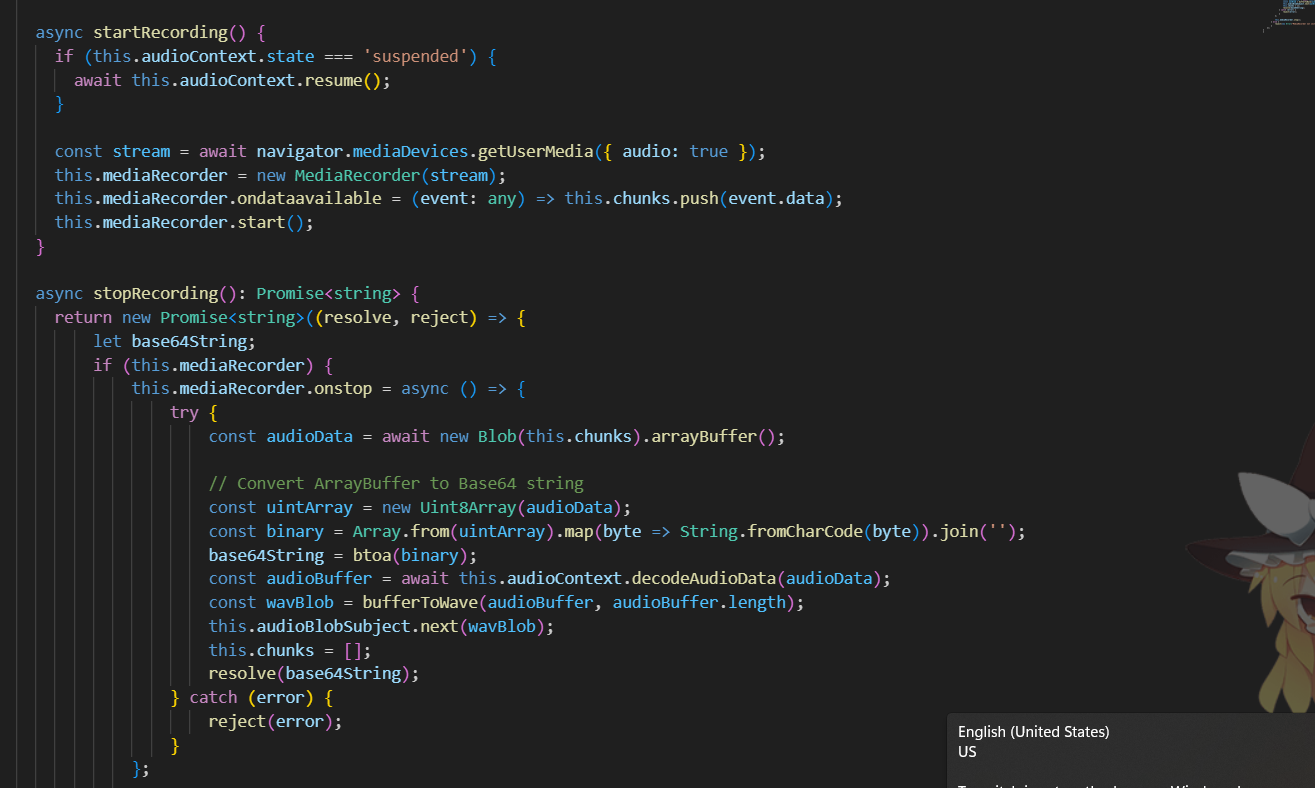
Name: Lina Samer Saied

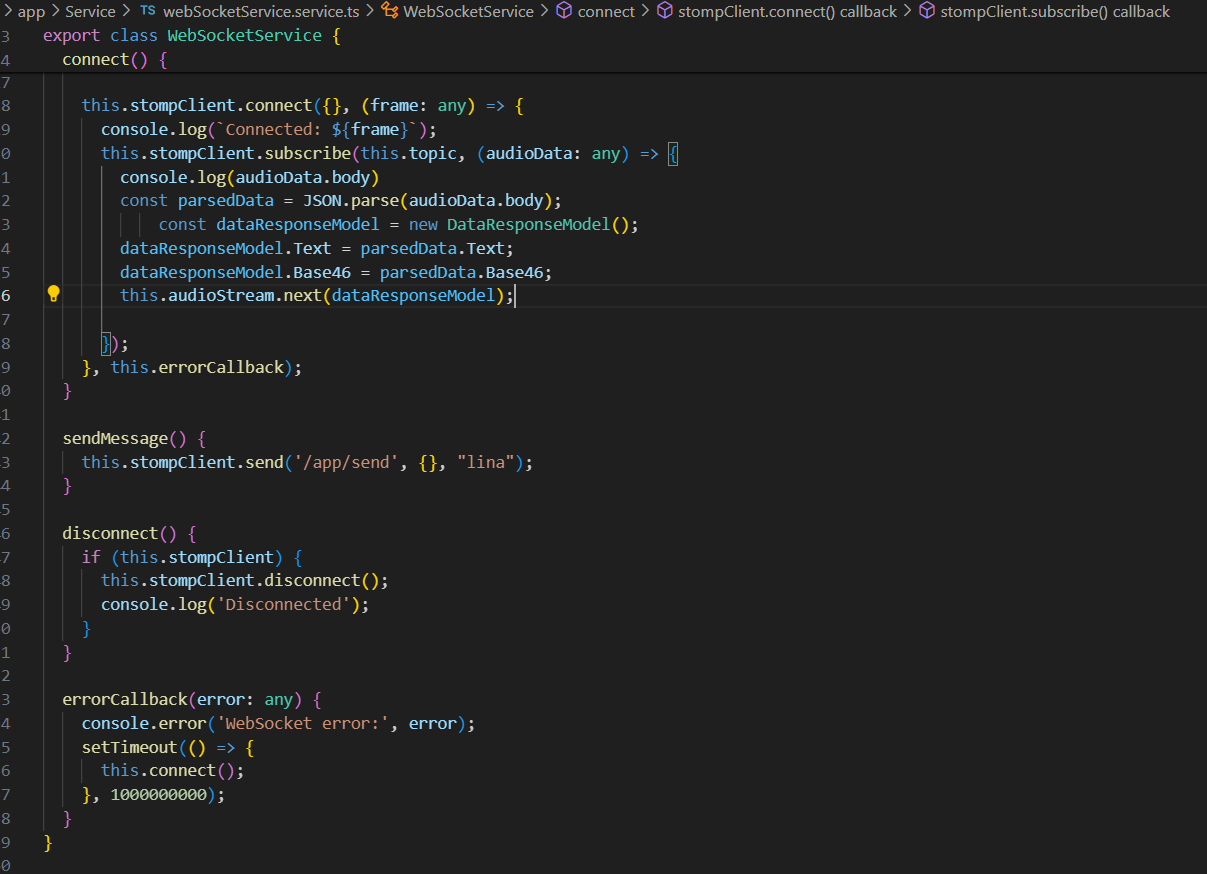


This configuration enables WebSocket communication in the Spring Boot application. It defines the endpoint for client connections, sets up a simple message broker, and configures message routing for both client subscriptions and application-specific messages.



 startRecording(): This method first checks if the audio context is suspended and resumes it if necessary. Then, it requests access to the user's microphone and creates a MediaRecorder instance. The recorded audio chunks are pushed into an internal array (chunks). Finally, the recording starts using mediaRecorder.start().

 stopRecording(): This method returns a promise that resolves with a Base64 encoded string representation of the recorded audio. It stops the media recorder and waits for the onstop event. Inside the event handler, it creates a Blob from the recorded chunks, converts the ArrayBuffer to a Base64 string, and decodes the audio data using the AudioContext. It then utilizes a helper function bufferToWave (likely defined elsewhere) to convert the decoded audio into a WAV Blob format. Both the Base64 string and the WAV Blob are made available: the Base64 string is resolved through the promise, and the WAV Blob is emitted through the audioBlobSubject observable for any subscribers to listen to. Finally, the internal chunk array is cleared.



**Key functionalities:**

* **connect():** This method initiates the WebSocket connection by creating a SockJS connection and using StompJS to connect over that socket. Upon successful connection, it subscribes to the /topic/receive topic on the server to receive audio data messages.
* **audioStream:** This BehaviorSubject serves as an observable that emits received audio data to any subscribed components.
* **audioData Subscription:** Inside the connection logic, the service subscribes to the /topic/receive topic. When a message arrives, it parses the JSON-formatted body, creates a DataResponseModel object with extracted text and Base64 data, and pushes it through the audioStream. This allows components to listen for and react to incoming audio data.
* **sendMessage():** This method sends a message ("lina") to the server endpoint /app/send. The purpose of this message and its handling on the server-side would depend on the specific application logic.
* **disconnect():** This method gracefully disconnects the WebSocket connection if it exists.
* **errorCallback():** This callback handles any WebSocket errors that might occur during connection or messaging. It logs the error and attempts to reconnect after a long delay (100 seconds in this example). This delay can be adjusted based on the application's requirements and resilience strategy.

Overall, this service provides a foundation for building real-time audio streaming applications in Angular by enabling two-way communication with a WebSocket server.