Top of Form

**Briefing Document: SignLang Translator Application**

This document provides an overview of a web application for sign language translation named "SignLang Translator". The app utilises AI-powered hand gesture recognition to translate sign language into text in real-time. The analysis is based on the provided source code and focuses on the key functionalities, technologies used, and the application's structure.

**1. Core Functionality & User Flow**

The application allows users to translate sign language into text using a webcam. The core functionalities include:

* **Real-time Sign Language Detection:** The app captures video from the user's webcam and utilises MediaPipe's Gesture Recognizer model to detect hand gestures in real-time.
* **Text Translation:** Detected gestures are translated into corresponding words or phrases, which are displayed on the screen.
* **Text Editing & Download:** Users can edit the translated text and download it as a .txt file.
* **User Authentication:** The app provides user registration, login, password reset, and profile management features.
* **Sign Language Learning Resources:** The app features a carousel displaying common sign language gestures, possibly to aid users in learning.

**2. Technologies Used**

* **Frontend:** The frontend is built using React, TypeScript, and Vite. It incorporates libraries like React Router for routing, Redux for state management, and Framer Motion for animations. Tailwind CSS is used for styling.
* **Backend:** The backend is built using Node.js with Express.js as the web framework. SQLite is used as the database for storing user data and potentially translation history.
* **AI/ML:** The application leverages MediaPipe's Tasks Vision library for hand tracking and gesture recognition. The core functionality relies on a pre-trained Gesture Recognizer model:
* **modelAssetPath:** "./Trained Model/sign\_language\_recognizer\_25-04-2023.task",

**3. Application Structure**

The application follows a typical client-server architecture.

* **Client:** The React application handles user interactions, displays the webcam feed and translated text, and communicates with the backend API.
* **Server:** The Node.js server provides API endpoints for user authentication, profile management, and potentially for storing and retrieving translation data.

**4. Security Considerations**

The application implements security measures including:

* **Password Hashing:** Passwords are hashed using bcrypt before storing them in the database.
* const hashedPassword = await bcrypt.hash(password, 10);
* **JWT Authentication:** User authentication is handled using JSON Web Tokens (JWT), providing a secure way to manage user sessions.
* const token = jwt.sign({ id: user.id, email: user.email}, JWT\_SECRET, {expiresIn: '1h'})
* **CORS Configuration:** Cross-Origin Resource Sharing (CORS) is configured to allow requests from specific origins, enhancing security.

**5. Potential Improvements & Observations**

* **Accessibility:** Further attention to accessibility features would enhance usability for individuals with disabilities.
* **Model Accuracy & Data Augmentation:** The accuracy and robustness of the gesture recognition model could be improved through further training and data augmentation, ensuring a wider range of signs and individual variations can be recognised accurately.
* **User Feedback System:** Implementing a system for users to provide feedback on translation accuracy would facilitate ongoing model improvement and refinement.

**6. Conclusion**

The SignLang Translator application demonstrates a promising approach to utilising AI for breaking down communication barriers. The application's use of modern web technologies and integration with powerful AI/ML libraries provides a solid foundation. Further development, particularly in areas of model accuracy, accessibility, and user feedback mechanisms, would enhance its impact and real-world applicability.

Bottom of Form