Developing a Real-Time Sign Language Translation Application Using MediaPipe and React

**OBJECTIVE**

The objective of this project is to develop a user-friendly, real-time sign language translation application. This will be achieved by leveraging MediaPipe for hand gesture recognition and React for building the application's user interface12.

The application will utilise a pre-trained MediaPipe model to accurately detect and interpret sign language gestures in real-time, transforming these gestures into corresponding text output34. This text output can then be downloaded by the user for further use5. The application aims to facilitate communication between sign language users and those who do not understand sign language, promoting greater inclusivity and accessibility.

**INTRODUCTION**

This project represents a groundbreaking effort to bridge a critical communication gap between the deaf and hearing communities through a comprehensive, full-stack application designed for real-time sign language translation. With the growing need for more inclusive communication technologies, this application strives to provide an effective solution for individuals who rely on sign language as their primary mode of communication. By enabling seamless, real-time translation of sign language into text or speech, the application significantly enhances the ability of the deaf community to interact with others in various settings, from educational institutions to workplace environments. This is particularly important in fostering greater social integration and reducing the social isolation that often accompanies communication barriers.

The application's design is meticulously crafted to ensure not only high functionality but also an exceptional user experience. The development team has placed a strong emphasis on integrating advanced technologies to deliver an accessible platform that prioritizes ease of use and reliability. The backend infrastructure, which serves as the backbone of the system, is engineered for both scalability and security, ensuring that user data is handled responsibly while maintaining top-notch performance under different usage scenarios. On the frontend, a responsive and intuitive interface is designed to be user-friendly and easily navigable, allowing users of all technical backgrounds to interact with the system with minimal learning curve. Accessibility is a key focus, with features tailored to ensure that the application is usable by individuals with varying abilities and needs.

Central to the functionality of the application is the implementation of cutting-edge artificial intelligence, specifically in the realm of gesture recognition.[5] Using advanced algorithms, the application is capable of accurately interpreting a wide range of sign language gestures in real time, converting them into readable text or spoken language. This powerful AI-driven approach not only guarantees high precision in translation but also ensures that the application can function effectively in diverse real-world environments, including varying lighting conditions and different signing styles. The ultimate goal of this project is not only to solve the immediate communication challenges faced by the deaf community but also to provide them with a versatile and reliable tool for enhancing daily interactions. Through this innovative solution, the application strives to empower users, making it easier for them to connect with others and participate more fully in both social and professional contexts.

Beyond its advanced technological capabilities, the design of this application underscores a profound commitment to inclusivity and accessibility, ensuring that it serves the diverse needs of its users. The decision-making process behind selecting the appropriate technologies and architectural frameworks was guided by the necessity to achieve a balance between high performance and ease of use. A primary consideration throughout the development process was the user experience—specifically, how to make the application as intuitive and accessible as possible. This focus on simplicity and functionality allows users, regardless of their technical proficiency, to operate the application with ease and confidence. By eliminating unnecessary complexity, the design ensures that individuals who may not have extensive technical backgrounds can still fully benefit from the application’s capabilities.

The user interface plays a pivotal role in creating a seamless and natural experience for individuals relying on sign language for communication. Through thoughtful design, the application presents a clear and straightforward layout that minimizes barriers to interaction. Key features, such as gesture recognition feedback and real-time translation output, are designed to be intuitive and easy to navigate, contributing to a smoother experience for both deaf users and their hearing counterparts. Furthermore, the system has been tested with diverse user groups to ensure that it meets various accessibility standards and can cater to different needs, such as visual or auditory impairments. The incorporation of user-centered design principles is a testament to the commitment to providing a tool that everyone can use effectively, regardless of their background or abilities.

At its core, the application aims to facilitate natural and uninhibited communication between individuals from different linguistic backgrounds, fostering greater inclusivity within society. By eliminating the traditional communication barriers that exist between the deaf and hearing communities, this application encourages meaningful interaction and collaboration across various settings, from casual conversations to professional engagements. The broader vision is to create a world where individuals who rely on sign language can engage freely with others, confident in the knowledge that technology can support their everyday communication needs. In this way, the project not only addresses a pressing technical challenge but also contributes to building a more inclusive and equitable society.

The impact of this application extends far beyond the realm of technological innovation, marking a significant milestone in the ongoing effort to dismantle communication barriers and foster genuine understanding between the deaf and hearing communities. By providing a clear and efficient means of communication, the application serves as a powerful tool for empowering deaf individuals, enabling them to engage more fully in both social and professional spheres. It helps to break down the traditional limitations imposed by language differences, creating an environment where communication is more fluid and accessible, and where deaf individuals can express themselves with ease and confidence. This not only enhances their quality of life but also contributes to a more inclusive society where everyone, regardless of their mode of communication, can participate in conversations and decision-making processes.

A key aspect of the application’s design is its unwavering commitment to accessibility and inclusivity. Every element, from the choice of underlying technologies to the design of the user interface, is rooted in the goal of creating a solution that works for all users. The intuitive layout and seamless integration of features ensure that individuals with varying technical expertise can interact with the system effectively. This focus on inclusivity reflects a deeper understanding of the diverse needs of the deaf community and highlights the project's potential to address real-world challenges. By considering the unique requirements of users who rely on sign language, the application provides a meaningful platform for communication that transcends traditional barriers and contributes to more equitable access to information, services, and social interactions.

Ultimately, the success of this project represents a significant leap toward utilizing advanced technology to solve pressing societal issues. The sophisticated technology embedded in the application is not simply an end goal but a tool that facilitates profound social change. Its ability to bridge communication gaps and promote understanding is a testament to the power of technology when applied thoughtfully and responsibly. Through the development and implementation of this project, the team has demonstrated a strong commitment to using innovation not just for technical achievement but to address real-world problems and create positive social impact. In doing so, this application not only improves the lives of individuals within the deaf community but also contributes to the creation of a more inclusive, communicative, and equitable world for all.

The backend architecture of this application is built on the powerful Node.js runtime environment, which serves as the backbone for handling server-side operations with optimal performance. Complementing this is the Express.js framework, which simplifies the creation and routing of APIs, allowing for clean, efficient communication between the frontend and backend. The system ensures the security of user data through the use of SQLite3, a lightweight yet highly reliable relational database that stores sensitive information securely. To protect user privacy and ensure secure authentication, the application employs Bcrypt for password hashing, making it virtually impossible for sensitive data such as passwords to be exposed in plaintext. Additionally, JSON Web Tokens (JWT) are utilized for secure session management, providing a robust mechanism for authentication that ensures only authorized users can access specific parts of the application. This layered security approach is essential to safeguard user data, especially when dealing with personally identifiable information, and it plays a crucial role in maintaining the application's overall integrity.

On the frontend, the application takes advantage of React, one of the most widely adopted JavaScript libraries, which is known for its efficiency and flexibility in building dynamic user interfaces. Paired with TypeScript, the frontend development benefits from enhanced type safety and improved code quality, which leads to a more maintainable and scalable codebase. TypeScript's static typing also helps developers catch potential errors early in the development process, further improving the reliability of the application. React Router is employed to enable smooth, intuitive navigation within the single-page application, providing users with a seamless browsing experience without the need for page reloads. Redux, a state management library, is integrated to manage the application's complex state, centralizing the flow of data and ensuring that components remain synchronized. This centralized approach optimizes the application’s performance by minimizing unnecessary re-renders and streamlining the management of state across the entire user interface.

Furthermore, the frontend is enriched with several key features designed to enhance user interaction and engagement. One of these features is a carousel that showcases example signs, offering a helpful visual guide for users to understand various sign language gestures in context. This interactive element serves as both an educational tool and a practical reference for users looking to familiarize themselves with sign language. The design is also visually intuitive, ensuring that users, including those who may not have extensive technical knowledge, can easily navigate the application. Additionally, a robust protected route mechanism is implemented to secure sensitive areas of the application, further ensuring that only authorized users can access protected content. This focus on both functionality and security contributes to a user experience that is not only efficient and responsive but also safe and secure, making it ideal for real-time sign language translation applications where privacy and data integrity are paramount.

At the heart of the application's functionality is MediaPipe, a powerful framework developed by Google, which plays a critical role in enabling real-time gesture recognition. Integrated seamlessly into the frontend's Detect component, MediaPipe leverages advanced machine learning models to interpret hand gestures with remarkable accuracy. By utilizing pre-trained models from MediaPipe Tasks Vision, the application is capable of recognizing and understanding a wide variety of hand movements that are crucial for sign language translation. This integration ensures that the system can process gestures in real time, making the translation process immediate and highly responsive, which is essential for effective communication. The precision and reliability of MediaPipe’s gesture recognition are key to the application's success, allowing it to provide users with an experience that feels natural and seamless.

To capture user input, the application utilizes the react-webcam library, which allows the system to access and stream video from the user's webcam. This setup ensures that hand gestures can be detected live, providing a smooth and interactive user experience. The react-webcam library is an ideal choice as it facilitates easy integration of webcam functionality into React applications, enabling smooth video feed processing in real time. As users perform sign language gestures, the system continuously analyzes the video feed, extracting relevant hand movements and passing them to MediaPipe for interpretation.[16] Once a gesture is recognized, the application instantly translates it into the corresponding text, ensuring that the feedback is immediate and contextually relevant to the conversation or interaction.

This real-time gesture-to-text translation not only enhances the speed and efficiency of communication but also ensures that users receive an accurate and reliable representation of their signed input. The application’s ability to deliver such timely and precise feedback significantly reduces the communication barriers between deaf and hearing individuals, fostering more natural and fluid exchanges. By integrating MediaPipe’s advanced gesture recognition with the frontend’s intuitive user interface, the application offers a powerful tool for sign language translation that is both effective and easy to use, making it a vital resource for enhancing communication and accessibility for the deaf community.

This project addresses a critical and pervasive problem: the significant communication barrier between the deaf and hearing communities. In a world that is increasingly interconnected, effective communication is essential for participation in all aspects of life, including social, educational, and professional environments. However, for individuals who rely on sign language as their primary mode of communication, interacting with those who do not understand it can present substantial challenges. These challenges are not limited to misunderstandings or communication delays; they extend to a lack of accessibility and inclusivity, which can significantly affect a deaf person’s ability to express their thoughts, needs, and emotions.

In environments where sign language comprehension is limited or absent, deaf individuals often find themselves excluded or forced to use alternative methods to communicate, which may not always be effective or efficient. This creates an ongoing barrier to full participation in day-to-day interactions. For example, in educational settings, deaf students may struggle to engage in classroom discussions or fully comprehend lectures if sign language interpreters or other accommodations are unavailable. Similarly, in the workplace, the inability to communicate freely can result in miscommunication, isolation, and missed opportunities for career advancement. This communication gap, while often overlooked, can have far-reaching consequences, leading to feelings of frustration, social isolation, and a diminished sense of belonging for those affected.

The limitations placed on the deaf community by these communication barriers are profound, as they directly impact access to resources, opportunities, and social connections. As a result, many deaf individuals experience a diminished quality of life, unable to engage fully in the world around them. By addressing this issue, the project aims to not only bridge the communication gap but also to foster a more inclusive society where everyone, regardless of their mode of communication, can participate and contribute equally. The ultimate goal is to create a tool that empowers the deaf community, ensuring they have the means to express themselves clearly and effectively in all areas of life, helping to reduce isolation and enhance opportunities for social, educational, and professional success.

Traditional communication methods, such as interpreters or written notes, often prove cumbersome, time-consuming, and unreliable, exacerbating the problem. The lack of readily available, real-time translation solutions creates a pressing need for technological intervention. This application directly addresses this unmet need, offering a powerful tool to improve communication, promote inclusivity, and empower individuals who use sign language. By providing real-time translation of sign language into text, this application empowers deaf individuals to communicate effectively and independently, breaking down barriers to full societal participation and contributing to a more inclusive and equitable world.

**RELATED WORKS AND LITERATURE SURVEY**

The development of Sign Language Recognition (SLR) systems has evolved significantly over the course of several decades, with early efforts primarily relying on cumbersome hardware-based solutions. The first attempts at creating SLR systems involved the use of specialized gloves equipped with sensors designed to track hand movements.[7][10] These devices were groundbreaking at the time, offering a novel way to capture and interpret the intricate gestures that form the basis of sign language. However, despite their innovative nature, these early systems came with a number of significant challenges. One of the primary issues was their bulkiness and discomfort; wearing sensor-laden gloves for extended periods was not only cumbersome but often physically uncomfortable for the user, making the technology impractical for everyday use.

Moreover, the cost of these hardware-based systems was prohibitively high, limiting their accessibility and adoption. The complexity of the data acquisition process was another significant barrier. These systems often required intricate calibration before use and demanded meticulous data annotation to ensure that the gestures were accurately captured and interpreted. This labor-intensive setup made it difficult to scale the technology or apply it to a wide range of users or environments. As a result, the systems were often relegated to controlled settings or specialized use cases, limiting their real-world applicability and effectiveness.

Additionally, the early hardware-based SLR systems struggled to account for the inherent variability in sign language. Sign language, much like spoken languages, is diverse and varies widely across regions, cultures, and individuals. Subtle differences in hand shapes, movements, and facial expressions could lead to misinterpretations or complete failures in recognition. These systems were not equipped to handle such variability, resulting in low accuracy and inconsistent performance.[7] This lack of robustness in early SLR systems hindered their widespread adoption and prevented them from achieving the level of reliability necessary for practical use in real-world scenarios. As a result, while these early innovations laid the groundwork for future advancements, they fell short of providing a reliable, accessible, and user-friendly solution for sign language translation.

The landscape of Sign Language Recognition (SLR) experienced a dramatic transformation with the advent of more powerful computing resources and significant breakthroughs in machine learning, particularly in the field of computer vision. These advancements ushered in a new era for SLR systems, allowing for more efficient, accurate, and accessible solutions. One of the most important developments in this shift was the application of Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), two types of deep learning models that have proven to be highly effective in tasks related to image and sequence recognition.[6] CNNs, with their ability to automatically extract features from visual data, and RNNs, which excel at handling sequential data, were particularly well-suited for interpreting the dynamic and complex nature of sign language gestures.

These machine learning breakthroughs enabled the creation of camera-based systems, which represented a significant leap forward from the hardware-heavy, glove-based solutions of the past. Camera-based systems could now passively observe and interpret sign language gestures in real time, with no need for the user to wear cumbersome devices. This shift brought numerous advantages, particularly in terms of user comfort and accessibility. Without the need for specialized hardware, the systems became much more user-friendly and approachable, allowing individuals to engage in sign language communication without the discomfort and limitations of sensor-laden gloves. Moreover, the reliance on readily available cameras made the technology more cost-effective, reducing the overall expenses associated with system deployment and scaling.

In addition to improving user experience, these advancements also helped to lower the technical barriers to entry for SLR development. With the proliferation of high-quality cameras and the increased availability of powerful computing platforms, developers were able to implement sophisticated algorithms that could process visual data in real time, drastically improving the speed and accuracy of sign language recognition. This shift opened up new possibilities for widespread adoption of SLR technology, enabling it to be used in a variety of real-world settings, such as educational institutions, workplaces, and social environments, with minimal hardware requirements. Ultimately, the combination of powerful machine learning algorithms and accessible, camera-based systems has revolutionized the field of SLR, making it a more practical, scalable, and user-centric solution for bridging communication gaps between deaf and hearing individuals.

Despite the significant advancements in Sign Language Recognition (SLR) driven by Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), several challenges persist in fully capturing the complexity of sign language.[12] Sign language is a rich and highly nuanced form of communication that involves not only hand shapes and movements but also facial expressions and body postures. These elements are often combined in intricate ways, with subtle variations in hand gestures, facial cues, and body alignment conveying important meanings. For machine learning algorithms to accurately interpret this multifaceted form of communication, they must be capable of recognizing and processing these complex interactions in real time. This requires sophisticated algorithms that can go beyond basic gesture recognition to incorporate the full range of expressive components that make up sign language.

One of the primary challenges in SLR remains the accurate capture and processing of both spatial and temporal aspects of sign language. Sign language gestures are not only defined by static hand shapes but also by the movement and positioning of hands and arms in space, as well as the timing and rhythm of these movements. The temporal dimension, which includes factors such as speed, pauses, and the sequencing of gestures, is critical for understanding the meaning of a sign. Effectively capturing these dynamic elements requires algorithms that can analyze and synchronize the spatial positions of hands with the timing of their movements over time. This makes the task significantly more complex than simple object recognition, as it requires the system to understand the movement patterns and their relationship to the context in which they occur.

Furthermore, the need for robust algorithms that can generalize well across different signers, signing styles, and environmental conditions adds another layer of complexity. Sign language varies widely between individuals, communities, and regions, with different dialects, personal signing styles, and variations in the use of facial expressions. Additionally, factors such as lighting conditions, background clutter, and camera angles can significantly affect the system’s ability to recognize gestures accurately. Developing algorithms that can adapt to these variations and perform consistently across diverse conditions remains a critical area of ongoing research. The challenge lies in creating systems that are not only accurate and reliable under ideal conditions but can also generalize well to the diverse and often unpredictable environments in which sign language communication takes place. Overcoming these challenges is essential for creating a fully functional and widely adoptable SLR system.

**Related Works: Key Advancements in the Field**

Several significant research efforts have contributed to the advancement of SLR and sign language translation systems. These contributions have pushed the boundaries of what is possible, paving the way for more accurate, robust, and user-friendly systems.

* Camgoz et al. (2018) and Neural Sign Language Translation (NSLT): This pioneering work introduced the concept of Neural Sign Language Translation (NSLT), utilizing deep learning techniques, particularly sequence-to-sequence learning models, to translate continuous sign language videos into text. This approach addressed the challenge of handling the temporal aspect of sign language, representing a significant step towards real-time translation. The authors' innovative use of deep learning architectures proved capable of learning the complex mapping between visual sign language data and textual representations. The system’s performance, while showing promise, highlighted the continued need for larger datasets and more robust training methodologies to improve accuracy and generalization.
* Koller et al. (2020) and the Pursuit of Accuracy and Latency: This research focused on improving the accuracy and reducing the latency of real-time sign language recognition systems.[26] Their exploration of deep neural networks (DNNs) led to the development of more efficient algorithms that could process visual data faster while maintaining high accuracy. The reduction in latency is crucial for creating a natural and responsive user experience, a key factor in the practical usability of such systems. Their work highlighted the delicate balance between accuracy, computational efficiency, and real-time performance in SLR systems.[14]
* MediaPipe (Google, 2021) and the Holistic Approach to Hand and Body Tracking: The introduction of MediaPipe by Google marked a significant turning point.[27] This holistic framework for hand and body tracking provides a lightweight yet powerful solution for real-time gesture recognition. By accurately detecting key points on hands and bodies, MediaPipe simplifies the process of extracting relevant features for sign language interpretation, eliminating the need for complex, computationally expensive feature engineering steps. Its efficiency and ease of integration have made it a popular choice for researchers and developers alike, significantly accelerating progress in the field.

**AI-Based Gesture Recognition Using MediaPipe: A Game Changer**

MediaPipe has rapidly emerged as one of the most powerful and efficient tools for real-time hand and gesture recognition, earning its reputation for both accuracy and speed. This is primarily due to its sophisticated algorithms, which are built on advanced machine learning models specifically designed to handle complex visual recognition tasks. One of the key advantages of MediaPipe is its ability to track hand gestures in three-dimensional space, enabling the system to capture not just the position of the hands but also the depth and orientation of each gesture. This 3D tracking capability is critical for accurately interpreting the dynamic and spatially intricate nature of sign language, where subtle movements and hand positions carry significant meaning.

In sign language, gestures are often defined by complex hand shapes, precise movements, and specific spatial relationships between the hands, arms, and body. For example, a slight variation in the angle of the hand or the positioning of the fingers can drastically change the meaning of a sign. MediaPipe's advanced algorithms are capable of recognizing these minute differences, allowing the system to distinguish between similar gestures that may otherwise be difficult to tell apart. This level of precision is essential for accurate sign language interpretation, especially when signs involve complex hand movements, changes in speed, or varying hand positions in relation to the body.

The ability to process and interpret gestures in real time, with such a high degree of accuracy, makes MediaPipe an ideal solution for real-time sign language translation. By leveraging MediaPipe's sophisticated machine learning models, developers are able to create systems that can capture the full range of sign language gestures, even those that involve nuanced spatial relationships between the hands and other parts of the body. This ensures that the application can interpret even the most complex signs correctly, providing a highly effective and reliable tool for communication. MediaPipe’s efficiency and precision in tracking and interpreting gestures have positioned it as a key enabler of more accurate, real-time sign language recognition systems[2], making it a valuable asset in bridging communication gaps between the deaf and hearing communities.

**Key Components and Algorithms Used in MediaPipe**

1. **Pose Estimation and Hand Tracking**: MediaPipe's hand gesture recognition primarily relies on the **Hand Landmark Model**. This model is part of MediaPipe's broader **Pose Estimation** framework, which is designed to detect and track various parts of the human body (like the face, hands, and body joints). For hand tracking, MediaPipe uses a deep learning-based approach to predict hand landmarks from images or video frames.[6]
   * **Convolutional Neural Networks (CNNs)**: MediaPipe uses CNNs as a backbone for extracting spatial features from images, which helps in identifying the position and orientation of hand gestures.[4] CNNs excel at recognizing spatial hierarchies in visual data, making them well-suited for tasks such as identifying hand shapes, finger positions, and orientations in real time.
   * **Keypoints/Hand Landmarks**: MediaPipe uses a set of keypoints (21 in total) to represent the hand's landmarks. These landmarks include the positions of each finger joint and the palm. By analyzing these keypoints, MediaPipe can capture the hand’s pose, movement, and orientation in 3D space.
2. **3D Hand Tracking**: A crucial feature of MediaPipe’s hand tracking is its ability to track hand gestures not just in 2D space, but in **3D**. This 3D tracking capability is achieved by using specialized algorithms that take depth information into account, which allows MediaPipe to distinguish between gestures that have similar 2D appearances but different spatial configurations.

MediaPipe Startegies for 3D tracking:

* + **Depth Estimation**: Using stereo vision or single camera depth estimation methods, MediaPipe calculates the spatial depth of different hand landmarks. This allows the system to track the movement of hands in three-dimensional space, accounting for the relative positions of hands and fingers as they move towards or away from the camera.
  + **Temporal Information**: Since hand gestures involve movement, MediaPipe also takes into account the **temporal sequence** of frames to maintain consistency and predict motion dynamics, improving the tracking of the hand's movement over time.

1. **Landmark Detection with Efficient Neural Networks**: MediaPipe uses **lightweight neural networks** optimized for real-time performance, which are particularly important in applications like hand gesture recognition, where high frame rates (e.g., 30-60 fps) are necessary for smooth interaction. The models used in MediaPipe are optimized for **mobile devices** and **edge computing** environments, meaning they are both computationally efficient and fast, which is essential for real-time applications.[8]
   * **Single Shot Hand Landmark Detection**: MediaPipe’s model is designed to process video frames in a single pass, predicting the location of hand landmarks for each frame. This method improves efficiency, as the entire image is processed at once, rather than relying on iterative approaches that would slow down the system.
2. **Classification and Gesture Recognition**: Once the hand landmarks are identified and tracked, MediaPipe can classify the gesture by comparing the positions and movements of the hand against a trained database of gestures. The system can detect specific hand gestures or signs, such as "hello," "goodbye," or "thank you," based on predefined gesture templates. This classification task may use:
   * **Multilayer Perceptron (MLP)** or **Recurrent Neural Networks (RNNs)**: In some implementations, MediaPipe uses simpler neural network models like MLPs or even RNNs (e.g., Long Short-Term Memory Networks or LSTMs) to classify the dynamic sequences of hand movements. RNNs are particularly useful because they can capture the sequential nature of gestures, recognizing patterns over time.
3. **Tracking and Stability**: For smooth and stable tracking of the hands and gestures, MediaPipe incorporates a combination of **temporal filtering** and **spatial smoothing** techniques. This ensures that the hand landmarks are tracked consistently even when there are rapid or erratic movements. Additionally, it handles occlusions (when one part of the hand or body is covered by another object) and maintains the model's robustness against varying environmental conditions.
4. **Cross-Platform Optimization**: MediaPipe is designed to work across a wide range of devices, including mobile phones, desktops, and embedded systems. The framework is optimized to run efficiently on both CPUs and GPUs, with optimizations made for low-latency performance. This cross-platform capability is achieved through the use of **TensorFlow Lite**, which is optimized for mobile and edge devices, and **XNNPACK**, a highly efficient library for running deep learning models on mobile hardware.

**How MediaPipe Works for Hand Gesture Recognition**

Here’s a simplified step-by-step overview of how MediaPipe processes and recognizes hand gestures:

1. **Capture Input**: The system captures a live video stream using a webcam or mobile device camera.
2. **Hand Detection**: MediaPipe detects the presence of hands in the frame and localizes them, using a combination of CNNs for spatial feature extraction.
3. **Hand Landmark Extraction**: The system then extracts 21 key landmarks for each hand, which include positions for the fingers, wrist, and palm. These keypoints are used to define the geometry and movement of the hand.
4. **Tracking**: The landmarks are tracked in 3D space across multiple frames to maintain consistent hand tracking, even as the hand moves or changes position.
5. **Gesture Recognition**: The system analyzes the relative positions of the hand landmarks and classifies the gesture based on pre-trained models or gesture templates. This could involve recognizing specific sign language signs, common hand gestures, or even dynamic hand movements over time.
6. **Real-Time Feedback**: Once the gesture is recognized, the system can provide real-time feedback, such as converting the gesture into text or speech, or triggering a specific action in the application.

**MediaPipe in Sign Language Applications: Practical Implementation:**

The adaptability of MediaPipe is one of its most compelling features, making it an ideal choice for a wide range of applications, particularly those focused on sign language recognition. One of the key advantages of MediaPipe in this context is its **real-time hand tracking capability**, which enables systems to capture and process hand gestures with minimal latency. This real-time performance is crucial in sign language applications, where accurate and immediate feedback is necessary to facilitate effective communication. Whether it's interpreting complex sign language gestures or tracking subtle hand movements, MediaPipe ensures that users can engage with the system seamlessly and naturally, without noticeable delays that could disrupt the flow of conversation or interaction.

In addition to its low-latency tracking, MediaPipe's **lightweight nature** is another factor that makes it particularly well-suited for sign language applications. Despite its powerful functionality, MediaPipe is optimized to be resource-efficient, meaning it can run effectively on devices with varying levels of computational power.[14] This is particularly important for sign language recognition systems, as they need to be accessible to a broad range of users across different devices and environments. MediaPipe's ability to run efficiently on **mobile phones**, **tablets**, and **low-end computers** extends its accessibility far beyond high-performance machines, democratizing access to sign language translation tools and making them available to a wider audience.

By being able to deploy on devices with less processing power, MediaPipe ensures that users do not need specialized hardware to benefit from sign language recognition systems. This level of flexibility is crucial for creating inclusive applications that can be used by individuals in a variety of settings, whether they are in a classroom, a workplace, or at home. The scalability of MediaPipe’s framework, combined with its real-time processing capabilities, allows developers to create systems that can offer intuitive and responsive user experiences without requiring extensive infrastructure.

Moreover, the ability to run on mobile devices opens up new possibilities for accessibility, as it enables on-the-go sign language translation that could be used in everyday interactions, whether in public spaces, during social interactions, or while traveling. MediaPipe’s adaptability and scalability, coupled with its ability to provide real-time, low-latency feedback, make it a powerful tool for the development of sign language applications, ensuring that these systems are not only effective but also accessible to a broader spectrum of users.

* Bazarevsky et al. (2019) and MediaPipe Hands: The introduction of MediaPipe Hands, a machine learning pipeline that accurately tracks 21 key points per hand in real-time, has revolutionized hand gesture recognition. This precise tracking capability has significantly improved the accuracy of sign language recognition systems, enabling the accurate identification of even the most subtle hand movements. The availability of pre-trained models further simplifies the development process, enabling rapid prototyping and deployment of sign language recognition applications.
* Puch et al. (2022) and the Comparison with Traditional Methods: This work highlights the significant advantages of MediaPipe over traditional CNN-based methods for gesture-based interactions. The study demonstrated that MediaPipe achieves superior accuracy and efficiency while requiring less computational resources, making it a more practical and scalable solution for real-time gesture recognition applications. This finding solidifies MediaPipe's position as a preferred tool for building robust and responsive sign language recognition systems. The study’s detailed analysis provides valuable insights into the optimal parameters and configurations for leveraging MediaPipe's capabilities within the context of sign language recognition.

**Gesture-Based UI and Accessibility: Transforming User Interaction**

Gesture-based user interfaces (UIs) are at the forefront of transforming how individuals with disabilities interact with technology, offering an intuitive and natural means of communication and control. These interfaces are particularly valuable for individuals who may have difficulty using traditional input devices such as keyboards or touchscreens. By allowing users to interact with technology through hand gestures, these systems create more inclusive and accessible environments. The development of gesture-based UIs, especially in the context of real-time sign language translation systems, plays a pivotal role in bridging communication gaps and enabling individuals, particularly those in the deaf and hard-of-hearing communities, to access information and engage with digital platforms in meaningful ways. For these systems to be effective, however, they must prioritize several key design considerations, including **ease of use**, **responsiveness**, and **visual clarity**. These factors ensure that the technology remains accessible to a wide range of users, regardless of their level of technical expertise or physical abilities.

Real-time sign language translation systems are crucial elements of this technological shift, providing an effective means of communication between deaf individuals and those who do not understand sign language. By interpreting gestures in real time, these systems facilitate smoother, more natural interactions in various settings, whether in education, healthcare, or social environments.[9] The design of such systems requires careful attention to user needs and the context in which they are used. For example, sign language gestures vary across regions and cultures, so the system must be adaptable to different signing styles. Additionally, the interface must be clear and responsive, ensuring that gestures are accurately captured and translated with minimal delay. Given the importance of visual representation in sign language communication, **visual clarity** is paramount, ensuring that the translation is easily readable and contextually accurate. This emphasis on clarity helps reduce ambiguity and enhances the user experience, making the system more reliable and trustworthy.

The research by **Sarkar et al. (2020)** on gesture-based interfaces for accessibility highlights the significant impact of these technologies on inclusivity.[1] Their study demonstrates the effectiveness of gesture recognition systems in improving accessibility for individuals with disabilities, particularly by enabling them to interact with digital platforms in ways that align with their natural communication styles.[11] The authors emphasize the importance of **user-centered design**, which prioritizes the needs, preferences, and abilities of the end user. This approach ensures that gesture-based interfaces are not only functional but also truly accessible, empowering users to interact with technology in a way that is intuitive, efficient, and personalized. The focus on responsiveness, adaptability, and clarity within the design of these systems is key to ensuring that gesture-based UIs can serve as a bridge for communication, information access, and empowerment for all users, regardless of their abilities or disabilities.

By incorporating the principles outlined in Sarkar et al.'s research and designing gesture-based systems with accessibility at their core, developers can create interfaces that not only meet technical standards but also foster inclusivity. [1]

**METHODOLOGY**

This project implements a real-time sign language translation application leveraging MediaPipe for gesture recognition and React for the user interface. The system's architecture combines robust backend functionality with a user-friendly frontend, all powered by cutting-edge AI-driven gesture recognition. Below is a detailed breakdown of the implementation and workflow

**1. Backend Development (Node.js with Express and SQLite):**

* **Server Setup:** The backend utilizes Node.js, a powerful JavaScript runtime environment, coupled with Express.js, a widely popular and efficient web application framework. The server is configured to listen on a specified port (e.g., 5555), employing middleware to handle tasks such as Cross-Origin Resource Sharing (CORS) for secure cross-domain communication, JSON parsing for data handling, and robust authentication mechanisms to protect user data. This robust server architecture ensures efficient and secure communication between the frontend and backend components.
* **Database:** SQLite, a lightweight and efficient embedded database system, is selected to store user data securely and efficiently. This choice simplifies the database management process while ensuring data persistence. Data stored includes user profiles, authentication credentials, and potentially a history of translations for personalized user experiences. The database schema is designed to optimize query performance and ensure data integrity.
* **API Endpoints:** A set of well-defined API endpoints is established to facilitate communication between the frontend and backend. These endpoints cover various functionalities, including user authentication (login, registration, password recovery), user profile management (retrieving, updating, and managing user information), and potentially endpoints for managing translation history or accessing pre-trained models. These endpoints are meticulously designed to provide a secure and efficient interface for the frontend to interact with the backend services.

**2. Frontend Development (React with TypeScript):**

* **User Interface:** The frontend utilizes React, a leading JavaScript library for building user interfaces, known for its component-based architecture and efficient rendering capabilities. React's component model promotes modularity, reusability, and maintainability of the codebase. The use of React allows for a responsive and dynamic user experience.
* **Components:** The application's architecture is based on a modular system of React components, each responsible for a specific part of the UI. This approach enhances code organization and reusability:
  + **Navbar:** Provides navigation and displays the user's authentication status, allowing users to easily switch between different application sections. It also includes functionalities such as user profile access and logout options. This component provides a consistent and intuitive navigation experience across the entire application.
  + **ProtectedRoute:** This component ensures secure access to sensitive parts of the application, only allowing authenticated users to access specific routes or functionalities, protecting sensitive user data and application features. This is a crucial component for security and data privacy.
  + **Home, About, Login, Register, ForgotPassword, Profile:** These components represent individual pages providing functionalities for user interaction, such as home page information, about the application details, user login and registration, password recovery, and user profile management. These pages are critical to user onboarding and engagement with the application.
  + **Translate:** This component houses the core sign language translation functionality, providing a central hub for users to interact with the real-time translation feature. The design of this component is critical for ensuring a smooth user experience.
  + **Detect:** This is the core component responsible for real-time sign language detection and translation, directly interacting with the MediaPipe API for gesture recognition and rendering the video feed from the user's webcam. This is the heart of the application’s functionality.
  + **Carousel:** A carousel component (optional) is designed to showcase example signs or provide visual aids to users, enhancing the learning experience and usability of the application. This component offers a user-friendly method of accessing example signs.
* **TypeScript Integration:** TypeScript is employed to add static typing, significantly improving code maintainability, readability, and reducing errors during development. TypeScript’s type system enhances the overall quality and robustness of the codebase, leading to fewer runtime errors and facilitating easier debugging. This is particularly helpful in larger projects where maintaining code consistency is crucial.

**3. Sign Language Detection and Translation (MediaPipe):**

* **MediaPipe Integration:** The Detect component seamlessly integrates MediaPipe, a versatile framework for building multimodal applied machine learning pipelines. MediaPipe's efficient algorithms and ease of integration are leveraged to handle real-time gesture recognition. The integration is accomplished using MediaPipe's well-documented APIs and libraries.
* **Model Loading:** A pre-trained sign language recognition model is loaded into the application, eliminating the need for extensive on-device training. This model, typically stored in a specified path, contains the necessary parameters for accurately recognizing sign language gestures. Pre-trained models reduce development time and facilitate faster application deployment.
* **Webcam Access:** The application utilizes the react-webcam library to gain access to the user's webcam, enabling real-time video capture for gesture recognition. This access is implemented with appropriate user permissions, ensuring compliance with privacy regulations. The application seamlessly integrates with the user’s webcam for real-time gesture detection.
* **Real-time Prediction:** The core function predictWebcam captures video frames from the webcam, processes them using the loaded MediaPipe model, and extracts key features like hand landmarks and gestures. This function is optimized for real-time performance, minimizing latency and enhancing the user experience. The efficient processing ensures smooth and responsive gesture recognition.
* **Gesture Recognition:** The MediaPipe model analyzes the extracted hand landmarks to identify gestures in real time, utilizing its pre-trained knowledge of sign language to translate the user's movements. The algorithm is robust enough to handle variations in hand shapes, positions, and movements. Real-time processing is essential for a smooth user experience.
* **Result Visualization:** The detected hand landmarks and their connections are visually displayed to the user on a canvas element, providing immediate visual feedback on the system's recognition of the gestures. This visual representation enhances user understanding and confidence in the system’s accuracy. This visual feedback is critical for user confidence and usability.
* **Translation Output:** The recognized gestures are then translated into text, displayed to the user in real-time, allowing for immediate feedback. The translated text can also be downloaded as a text file, enabling users to save or share their translations. This feature ensures that users can access their translated content even after the session ends.

**4. Workflow:**

1. **User Authentication:** The user either logs in using existing credentials or creates a new account, with the backend securely managing authentication and user data storage. This step ensures secure access to the application and protects user information.
2. **Navigation to Translate Page:** The user navigates to the designated "Translate" page, which houses the core gesture recognition and translation functionality. This is the central hub for interaction with the real-time translation system.
3. **Webcam Activation:** The user grants permission for the application to access their webcam, initiating the real-time video feed. This access is handled securely and with user consent, respecting user privacy.
4. **Real-time Detection:** The MediaPipe model begins processing video frames from the webcam, detecting hand landmarks and gestures in real time. This continuous processing ensures minimal latency and accurate translation.
5. **Translation:** Detected gestures are immediately translated into text and displayed to the user. The system aims for seamless, real-time translation without noticeable delays.
6. **Output:** Users can edit the translated text, clear the output field, and download the translated text as a file for later reference. This flexibility allows for correction of potential misinterpretations and provides a record of the translation.

**5. Additional Details:**

* **State Management:** Redux is used for managing the application's state, with dedicated slices for handling authentication (authSlice) and translation (translationSlice). This approach enables efficient and organized state management, particularly important in a complex application with multiple interactive components. Redux ensures that the application state is consistent and easily accessible across different components.
* **Styling:** Tailwind CSS is used for styling, offering a utility-first approach for rapid and consistent styling, providing quick and easy customization of the application's visual appearance. Tailwind’s utility classes enable efficient and consistent styling.
* **Development Environment:** Vite is utilized as the development server, providing a fast and efficient development workflow. Vite’s speed significantly improves the developer experience, facilitating quicker iterations and bug fixes. The choice of Vite reflects the project’s commitment to efficient development practices.

**TECHNOLOGIES USED**

This project leverages a collection of carefully chosen modules and frameworks to achieve real-time sign language translation. The architecture is designed for efficiency, scalability, and maintainability, combining robust backend services with a responsive and intuitive frontend, all driven by the power of MediaPipe's gesture recognition capabilities.

**Backend:**

* **Node.js:** This JavaScript runtime environment forms the foundation of the backend server. Node.js's non-blocking, event-driven architecture is crucial for handling the real-time nature of the application, allowing it to efficiently manage multiple concurrent requests without performance degradation. Its asynchronous nature is especially beneficial for handling real-time data streams from the frontend and providing fast responses. The use of JavaScript across both the frontend and backend simplifies development and promotes consistency.
* **Express.js:** Built upon Node.js, Express.js provides a streamlined and efficient framework for creating web servers and APIs. It simplifies the process of defining routes, handling HTTP requests (GET, POST, etc.), and managing middleware (functions that execute before and after request handling). Express.js's minimalist approach ensures optimal performance while providing the necessary tools for building a robust and scalable backend. The framework's modular design allows for easy integration of additional functionalities and libraries.
* **SQLite3:** This lightweight, serverless database engine is ideal for storing user data, such as login credentials and potentially translation histories. Its self-contained nature simplifies deployment and maintenance, eliminating the need for a separate database server. SQLite3's efficiency and ease of use make it a suitable choice for applications that don’t require the complexity of a large-scale relational database. Its suitability for smaller datasets makes it an efficient choice for this project.
* **Bcrypt:** Security is paramount, and Bcrypt provides a robust solution for password hashing. Instead of storing passwords in plain text (a significant security risk), Bcrypt employs a computationally intensive one-way hashing algorithm. This ensures that even if the database is compromised, passwords cannot be easily recovered. The use of Bcrypt significantly strengthens the application's security posture, protecting user accounts from unauthorized access. The algorithm's strength and resistance to brute-force attacks are key to its selection.
* **JSON Web Tokens (JWT):** JWTs provide a secure mechanism for user authentication and authorization. These tokens are digitally signed, ensuring their integrity and authenticity. They allow the application to verify user identity without constantly querying the database, improving performance and scalability. JWTs are a widely accepted standard for secure authentication and authorization in web applications, offering a robust and efficient solution. The use of JWT significantly enhances the security and efficiency of the authentication process.

**Frontend:**

* **React:** React, a JavaScript library for building user interfaces, forms the core of the frontend. Its component-based architecture allows for the creation of reusable and maintainable UI elements. React's efficient rendering mechanism ensures a smooth and responsive user experience, even with complex interactions. React's popularity and extensive community support ensure access to numerous resources and libraries. The component-based nature facilitates code reuse and maintainability.
* **TypeScript:** TypeScript enhances the quality and reliability of the React codebase by adding optional static typing. This allows developers to catch type errors during development, reducing runtime errors and improving code maintainability. TypeScript’s type system helps prevent common errors, improving code quality and maintainability. The increased code clarity and maintainability make development more efficient.
* **React Router:** This library facilitates navigation within the single-page application, allowing users to seamlessly transition between different views and sections of the application. React Router enables a fluid user experience by managing the application's URL and rendering the appropriate components based on the current route. The efficient routing mechanism improves the user experience and makes the application more intuitive to use. It simplifies the management of multiple views within a single-page application.
* **Redux:** Redux provides a centralized state management system for the React application. This approach allows for easier tracking and updating of application state, making debugging and maintaining the application significantly easier. It ensures consistent state management and improves the application's predictability and maintainability.[29] The centralized approach simplifies debugging and reduces the complexity of managing the application's state.
* **Framer Motion:** This library adds smooth animations and transitions to the UI, enhancing the visual appeal and user experience. Framer Motion's declarative approach allows for the creation of dynamic and engaging animations with minimal code. Visually appealing animations enhance the user experience and make the application more engaging. It provides a streamlined way to add sophisticated animations to the UI.
* **React Webcam:** This React component provides access to the user's webcam, allowing the application to capture video streams in real-time. This capability is essential for capturing the user's sign language gestures, which are then processed by the MediaPipe model.[28] Real-time webcam access is critical for the core functionality of the application, providing the input for the gesture recognition system. The library's ease of integration simplifies the implementation of real-time video capture.

**Sign Language Detection:**

* **MediaPipe:** At the heart of the gesture recognition system lies MediaPipe, a robust and versatile open-source framework developed by Google.[30][27] Its exceptional efficiency in processing real-time video streams makes it perfectly suited for the demands of this application. MediaPipe's ability to handle high-volume data streams with minimal latency is a critical advantage. The framework's inherent accuracy in detecting subtle hand movements ensures reliable gesture recognition, even in challenging conditions like variable lighting or background clutter. MediaPipe's pre-trained models significantly streamline development, eliminating the need for extensive model training from scratch. This reduces development time and resources considerably, allowing developers to focus on integrating the system into the application rather than building the core gesture recognition functionality from the ground up. The framework's modular design promotes seamless integration into existing applications, minimizing disruption and maximizing flexibility. This modularity is essential for adaptability and **ease of maintenance in the long run.**
* **MediaPipe Tasks Vision:** Leveraging MediaPipe Tasks Vision further enhances the system's capabilities. This component offers pre-trained models specifically optimized for various computer vision tasks, including gesture recognition.[27] The availability of these pre-trained models is a significant advantage, eliminating the need for extensive data collection and the computationally intensive process of training a model from scratch. This dramatically reduces development time and resources, allowing for rapid prototyping and iteration.[3] The pre-trained models within MediaPipe Tasks Vision are meticulously designed to deliver high accuracy, even with limited training data. This precision ensures that the system accurately interprets even subtle gestures, leading to a more reliable and user-friendly experience. The use of these optimized models directly contributes to enhanced performance and a reduction in development time, allowing for quicker deployment and faster iteration on feature development.
* **@mediapipe/drawing\_utils:** This dedicated MediaPipe module plays a crucial role in providing visual feedback to the user. It offers functions for efficiently rendering the detected hand landmarks and recognized gestures directly onto a canvas element. This visual representation of the system's interpretation of hand movements is invaluable for both the user and the developer. For the user, this visual feedback provides immediate confirmation that the system is accurately recognizing their gestures, leading to a more intuitive and engaging experience. From a development perspective, the visualization capabilities are essential for debugging and troubleshooting. The ability to see the detected landmarks in real-time allows developers to quickly identify and rectify any issues with the gesture recognition process, significantly accelerating the debugging cycle. The module's straightforward API makes integrating this crucial visual feedback simple and effective, contributing to a smoother development workflow. The clear visual representation ensures transparency and understanding of how the system interprets gestures, benefiting both user experience and development efficiency.[31]

**PROJECT SETUP:**

The project employs a robust full-stack architecture, meticulously separating the backend, frontend, and MediaPipe integration into independent, modular components. This strategic design choice significantly enhances maintainability by isolating functionality and reducing the impact of changes in one area on others. The modularity also promotes cleaner code organization, facilitating easier scaling and independent updates of each component without disrupting the entire system. This approach is crucial for long-term project health and efficient development cycles.

**Backend Setup:**

* Environment: Node.js serves as the runtime environment for the backend. Before initiating development, it's imperative to install a specific, explicitly defined version of Node.js (e.g., v18.16.0). Specifying the version ensures consistent behavior across all development environments, preventing discrepancies caused by differing Node.js functionalities or bug fixes across versions. This consistent environment is crucial for reproducible builds and troubleshooting.
* Framework: Express.js, a widely adopted and versatile Node.js framework, forms the foundation of the backend API. Its lightweight nature and extensive middleware support simplify routing, request handling, and overall API development. This framework's ease of use and flexibility streamline the process of building and maintaining the backend services, allowing developers to focus on core functionality rather than intricate framework complexities.
* Dependencies: The package.json file comprehensively lists all required backend dependencies. These include Express.js for the API, SQLite3 for database interaction, Bcrypt for password hashing and security, CORS for managing cross-origin requests, and JWT for secure authentication. Utilizing a package manager like npm or yarn streamlines dependency installation, ensuring version control and simplifying the process of managing project dependencies.
* Database: A SQLite database, such as app.db, is employed to store user data efficiently. To ensure a consistent starting point and to facilitate testing, an initialization script is recommended. This script can set up the necessary database schema and populate it with sample data, including a default administrator account, which greatly aids in the initial testing and development phases.
* Running the Server: Initiating the backend server is straightforward, typically involving executing a simple command such as npm start or yarn start. This command launches the server, usually listening on a predetermined port, such as 5555, making the API accessible to the frontend or other clients. Clear instructions for starting the server are crucial for ease of use and developer onboarding.

**Frontend Setup:**

* Environment: The frontend development environment relies on Node.js and a package manager like npm or yarn. Similar to the backend, a specific Node.js version should be explicitly defined (e.g., v18.16.0) to guarantee compatibility and prevent version-related conflicts. This ensures consistent build processes and avoids potential runtime errors.
* Framework: Vite, a modern build tool, is chosen for its exceptional speed and advanced features, resulting in a significantly improved development experience. Vite's rapid build times and hot module replacement capability enhance developer productivity and efficiency throughout the development lifecycle.
* Dependencies: The package.json meticulously details all frontend dependencies. These include React for the user interface, React Router for navigation, Redux for state management, Lucide React for providing a consistent set of icons, and Framer Motion for creating smooth and engaging animations. These dependencies are managed via npm or yarn, which simplifies installation and dependency resolution.
* Running the Development Server: Starting the frontend development server typically involves running a command like npm run dev or yarn dev. This command launches a local development server, usually accessible through a local URL such as http://localhost:5173, allowing developers to view and test the application in real-time. The ease of launching the development server contributes to a smooth development workflow.

**MediaPipe Setup:**

* Framework: MediaPipe is seamlessly integrated into the frontend's Detect component. The necessary MediaPipe libraries are directly imported and utilized for real-time gesture recognition. This integration ensures that the gesture recognition capabilities are readily available within the application's frontend.
* Model: A pre-trained MediaPipe model specifically designed for sign language recognition is required. This model needs to be correctly placed within the project's directory and its path accurately configured within the application's code to ensure proper loading and utilization. Explicit instructions on model placement and configuration are vital for successful integration.
* Webcam: The react-webcam library facilitates webcam access. The application must request user permission to access the camera, adhering to best practices for user privacy and data security. Handling user permissions appropriately is critical for compliance and user trust.

**RESULTS**

This document provides an extensive exploration of a full-stack application designed for real-time sign language translation. We will delve into the architectural design, technological choices, implementation details, and potential future enhancements of this sophisticated system. The application successfully leverages a powerful synergy between backend infrastructure, a responsive frontend, and cutting-edge AI capabilities for accurate and efficient gesture recognition.

**I. Backend Architecture: Robustness and Security**

The backend of the application is built upon a robust and secure foundation, utilizing several key technologies working in concert. Node.js, a highly scalable and efficient JavaScript runtime environment, provides the application's core execution environment. Its non-blocking, event-driven architecture is perfectly suited for handling the real-time nature of sign language translation, allowing for simultaneous processing of multiple user requests without performance degradation.

Express.js, a minimalist and flexible Node.js web framework, simplifies the development and management of the application's APIs. Express.js handles routing, allowing different requests to be directed to the appropriate handlers, and provides middleware functionality for tasks such as authentication and data validation. This streamlined approach ensures efficient resource utilization and simplifies the maintenance and expansion of the API.

Data persistence is achieved through SQLite3, a lightweight and serverless database solution. While its simplicity makes it ideal for this application's needs, its limitations should be considered for scalability in the future. SQLite3's file-based nature offers ease of deployment and maintenance, making it well-suited for a project of this scale. For larger deployments, migrating to a relational database like PostgreSQL or MySQL might be necessary to handle an increased volume of users and data.

Security is paramount, and the application addresses this through the integration of Bcrypt for password hashing. Bcrypt is a computationally intensive algorithm designed to protect passwords from brute-force attacks and rainbow table attacks, ensuring user data remains secure.[3] The use of bcrypt adds a significant layer of protection against unauthorized access and data breaches, a crucial aspect of a user-centric application.

Furthermore, JSON Web Tokens (JWTs) are employed for secure user authentication and session management. JWTs are digitally signed tokens that contain user information, allowing for secure and stateless authentication. This approach eliminates the need for persistent sessions on the server, improving scalability and security. Upon successful authentication, the system issues a JWT to the client, which is then used for subsequent requests. This ensures that only authenticated users can access protected resources. The inclusion of features like token expiration and refresh tokens enhances the system's security and prevents unauthorized access.

**II. Frontend Development: User Experience and Interactivity**

The user-facing aspects of the application are developed using React, a popular JavaScript library for building user interfaces. React's component-based architecture promotes code reusability, modularity, and maintainability, making the application more easily adaptable to future changes and expansions. The use of functional components and hooks further enhances the efficiency and readability of the codebase.

TypeScript, a superset of JavaScript with static typing, enhances code quality and maintainability by enabling early detection of errors during development. The addition of static typing significantly improves the reliability and robustness of the application, reducing the chances of runtime errors and facilitating easier debugging. Furthermore, TypeScript improves code readability and understandability, benefiting both the original developers and future contributors.

Navigation within the application is facilitated by React Router, enabling seamless transitions between different views within the single-page application. React Router simplifies the creation of complex user interfaces with multiple views and sub-routes, enhancing the user experience.

Redux, a predictable state container for JavaScript apps, provides a centralized store for managing the application's state. This ensures consistent data flow across different components, leading to better code organization and maintainability. Redux's unidirectional data flow simplifies debugging and makes it easier to track changes to the application's state. The use of Redux simplifies state management, particularly beneficial for a complex application with multiple interacting components. This ensures data consistency and predictability throughout the application.

**III. Real-Time Gesture Recognition with MediaPipe**

The core functionality of the application relies on MediaPipe, a powerful framework developed by Google for building multimodal applied machine learning pipelines. MediaPipe's integration is crucial for real-time gesture recognition. The application leverages pre-trained models from MediaPipe Tasks Vision, specifically designed for hand gesture recognition. These pre-trained models offer high accuracy and efficiency, ensuring that the translation process is both accurate and responsive.

The integration with MediaPipe is implemented within the "Detect" component of the frontend. This component interacts with the user's webcam using the react-webcam library, which provides access to the live video feed. The video feed is then processed by the MediaPipe model, which identifies hand landmarks and gestures in real-time.

The accuracy and efficiency of the MediaPipe model are critical to the application's success. The use of pre-trained models reduces development time and effort, allowing developers to focus on other aspects of the application. The real-time processing capability is crucial for providing a seamless and responsive user experience. This real-time feedback is essential for a natural and intuitive interaction.

The choice of MediaPipe is justified by its performance and ease of integration, offering a balance between accuracy and efficiency. The potential for future improvements through model updates and advancements in MediaPipe's capabilities further enhances the long-term viability of the application.

**IV. Application Workflow and User Experience**

The application workflow begins with user authentication. Users can either log in using existing credentials or register a new account. The authentication process is handled securely using JWTs, as previously described. Upon successful authentication, the user is redirected to the translation page.

The Translate page is the application's core functionality. Here, the "Detect" component, powered by MediaPipe, takes center stage. The user grants the application access to their webcam, allowing the MediaPipe model to begin processing the video feed. The model identifies hand landmarks and translates these gestures into text in real-time. This text is displayed to the user, providing immediate feedback.

Beyond this core functionality, several features enhance the user experience:

* Real-time Translation: The application translates gestures into text in real-time, minimizing latency and providing an immediate response.
* Text Editing: Users can edit the translated text if necessary, allowing for corrections or modifications.
* Output Clearing: A clear button allows the user to reset the output area, preparing for a new translation.
* Download Functionality: The translated text can be downloaded as a text file, allowing users to save and share their translations.
* Example Sign Carousel: A carousel displays example signs, helping users familiarize themselves with the application's capabilities and understand the expected gestures.
* Smooth Animations and Transitions: Framer Motion is utilized to create a polished and visually appealing user experience, with smooth animations enhancing usability and engagement.
* Protected Routes: Only authenticated users can access the translation page and other sensitive features, protecting user privacy and data security.

This integrated approach ensures a seamless and intuitive user experience, making the application accessible and user-friendly. The attention to detail in the user interface contributes significantly to its overall success.

**V. Technological Choices and Justification**

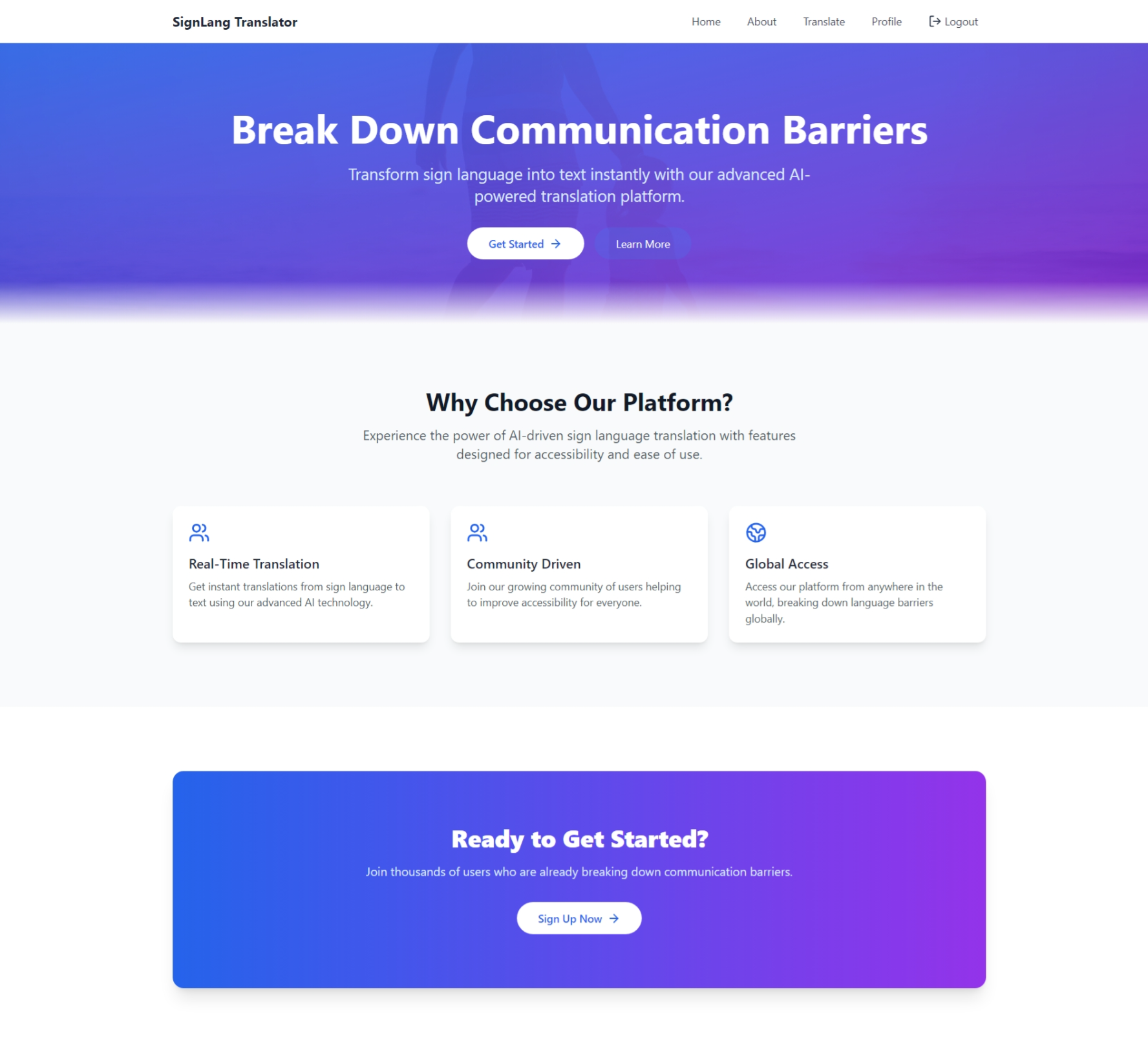
The selection of technologies for this application reflects a careful consideration of several factors, including performance, scalability, security, and ease of development. The chosen technologies represent a well-balanced approach:

* Node.js and Express.js: This combination provides a robust and efficient backend solution, suitable for handling real-time requests and scaling to accommodate increasing user load.
* SQLite3: This lightweight database simplifies data management for smaller-scale deployments, while acknowledging the need for a more robust solution for future scaling.
* Bcrypt: A strong password hashing algorithm is essential for protecting user data.
* JWT: JWTs enable secure and stateless authentication, improving scalability and security.
* React, TypeScript, and Redux: This frontend stack offers a powerful combination for building interactive and maintainable user interfaces.
* React Router: Simplifies navigation within the single-page application.
* MediaPipe and react-webcam: These provide the crucial capabilities for real-time gesture recognition and webcam access.
* Framer Motion: Enhances user experience with smooth animations.

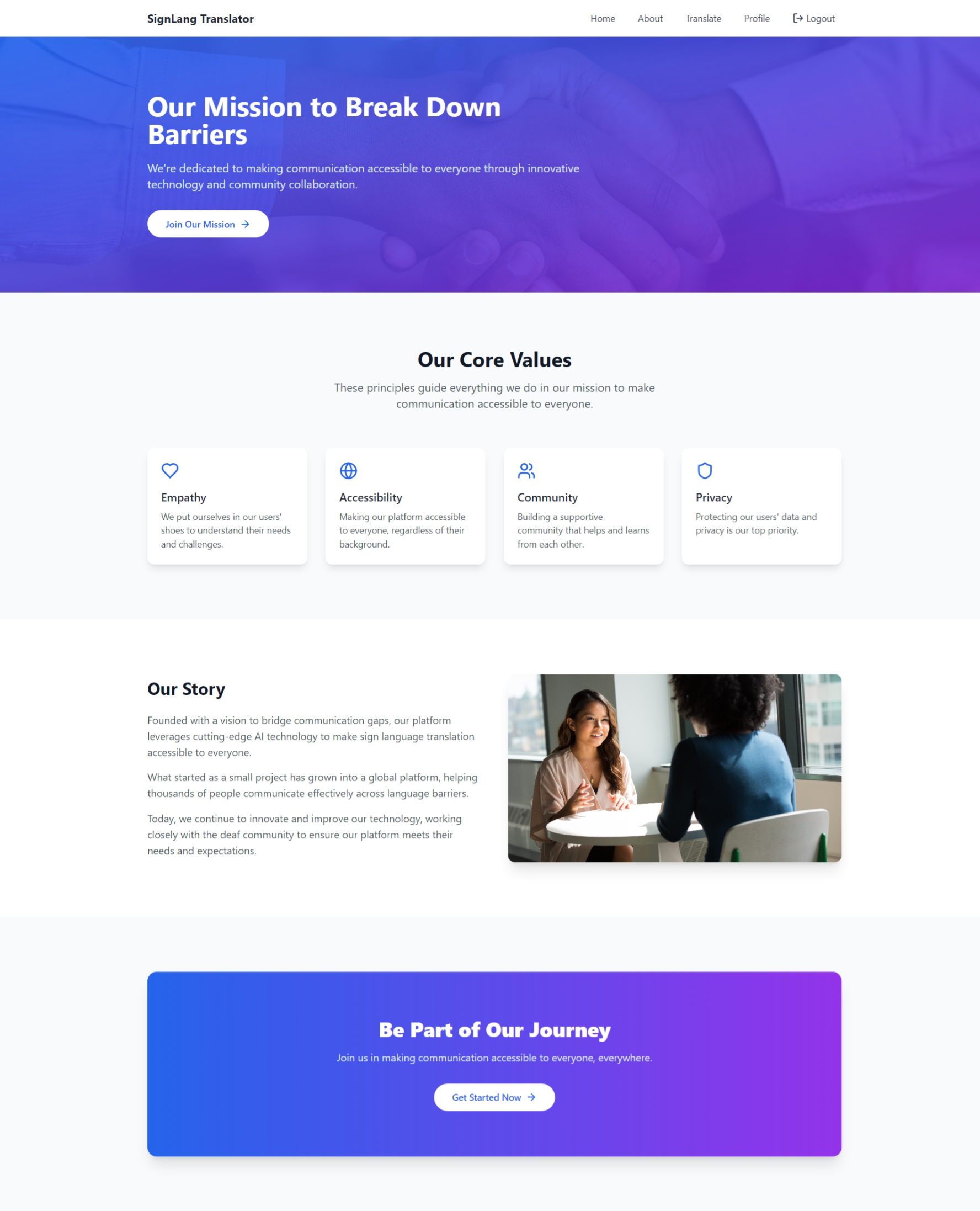
The modular design of the application allows for easy maintenance and scalability. The use of industry-standard tools and frameworks ensures the long-term maintainability of the project. The well-defined separation of concerns between the frontend, backend, and AI processing components facilitates independent development and testing.

**PROJECT OUTPUT**

Home Page:

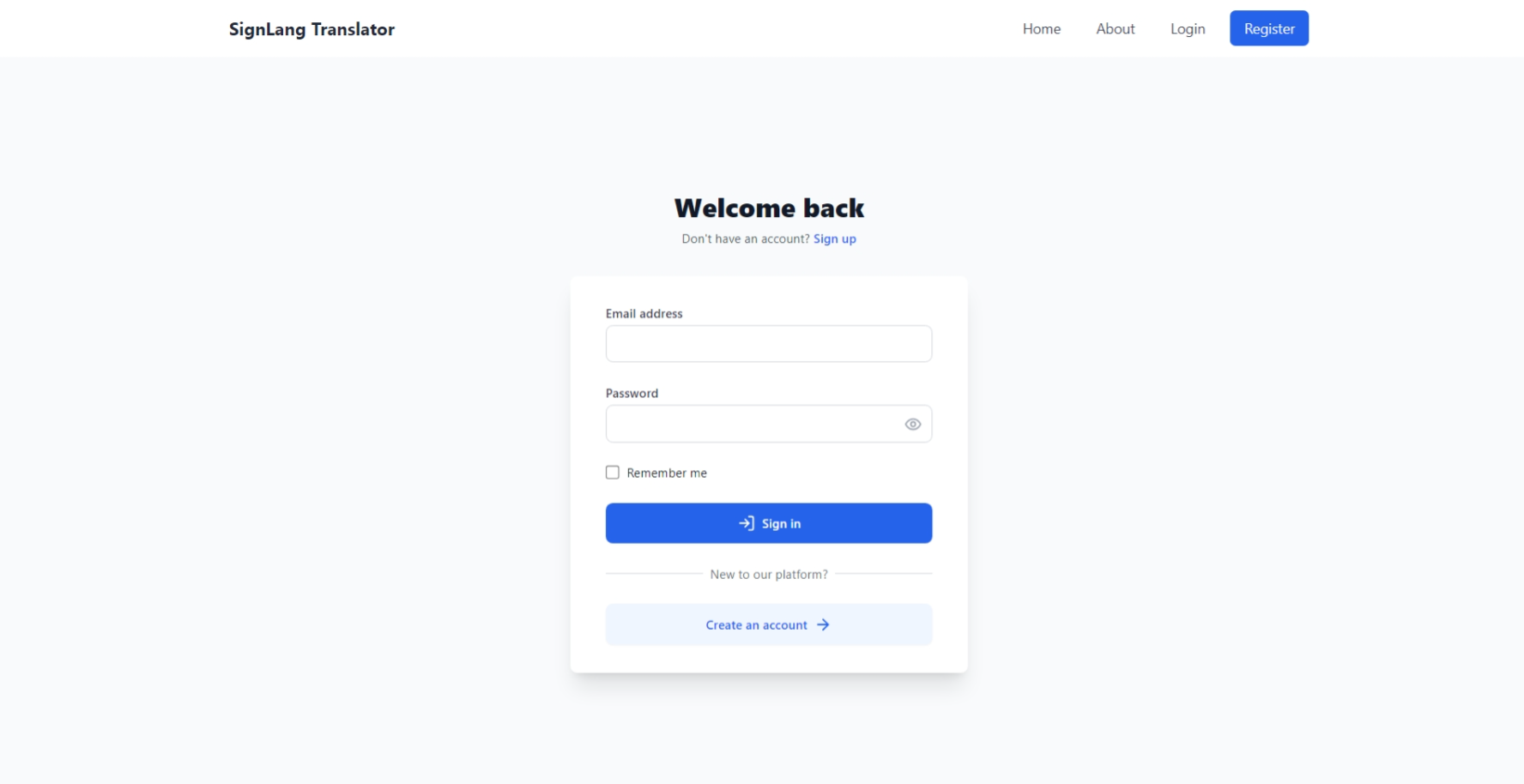
****

Fig(i) Home Page

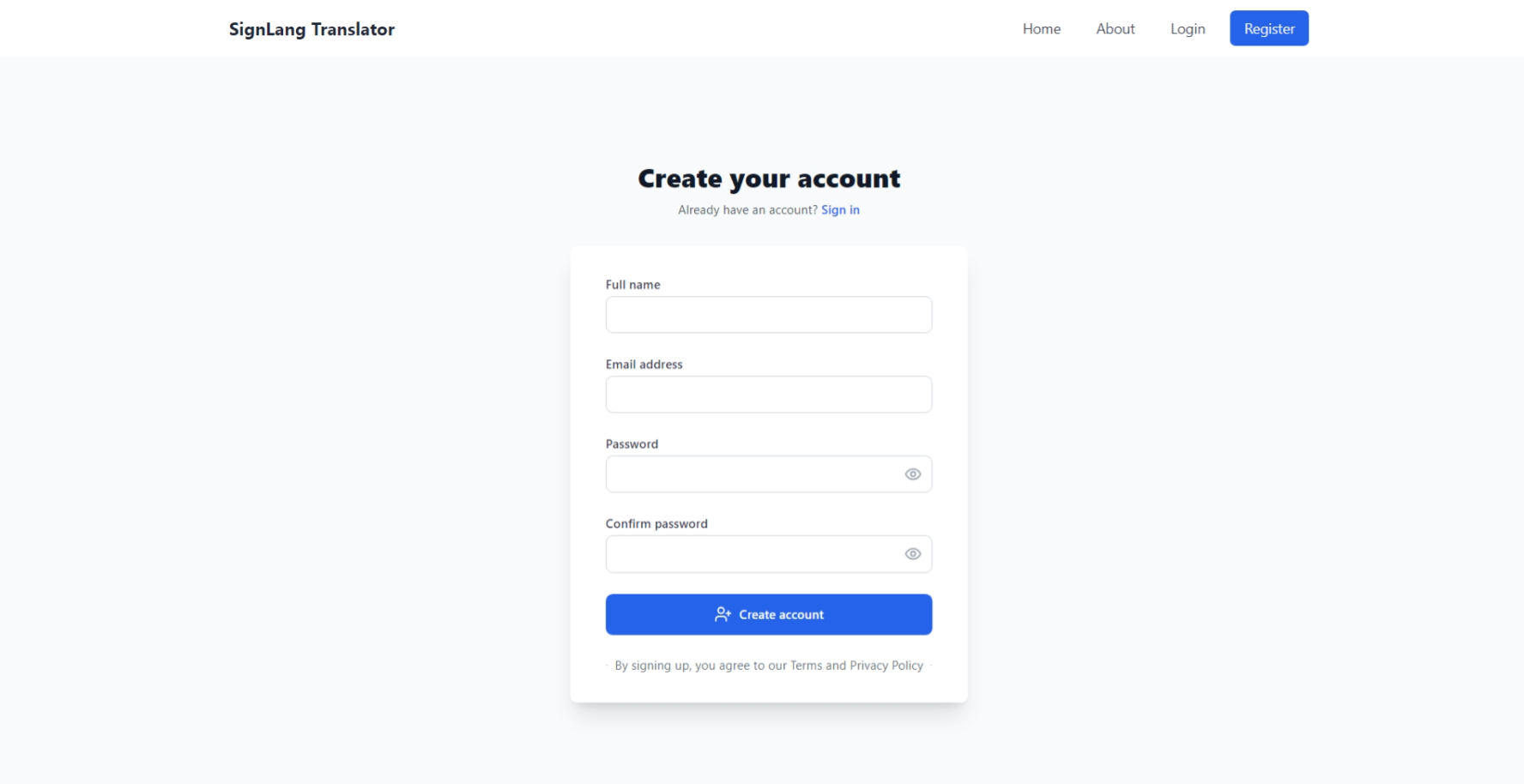
About Us Page:****

Fig(ii) About Us Page

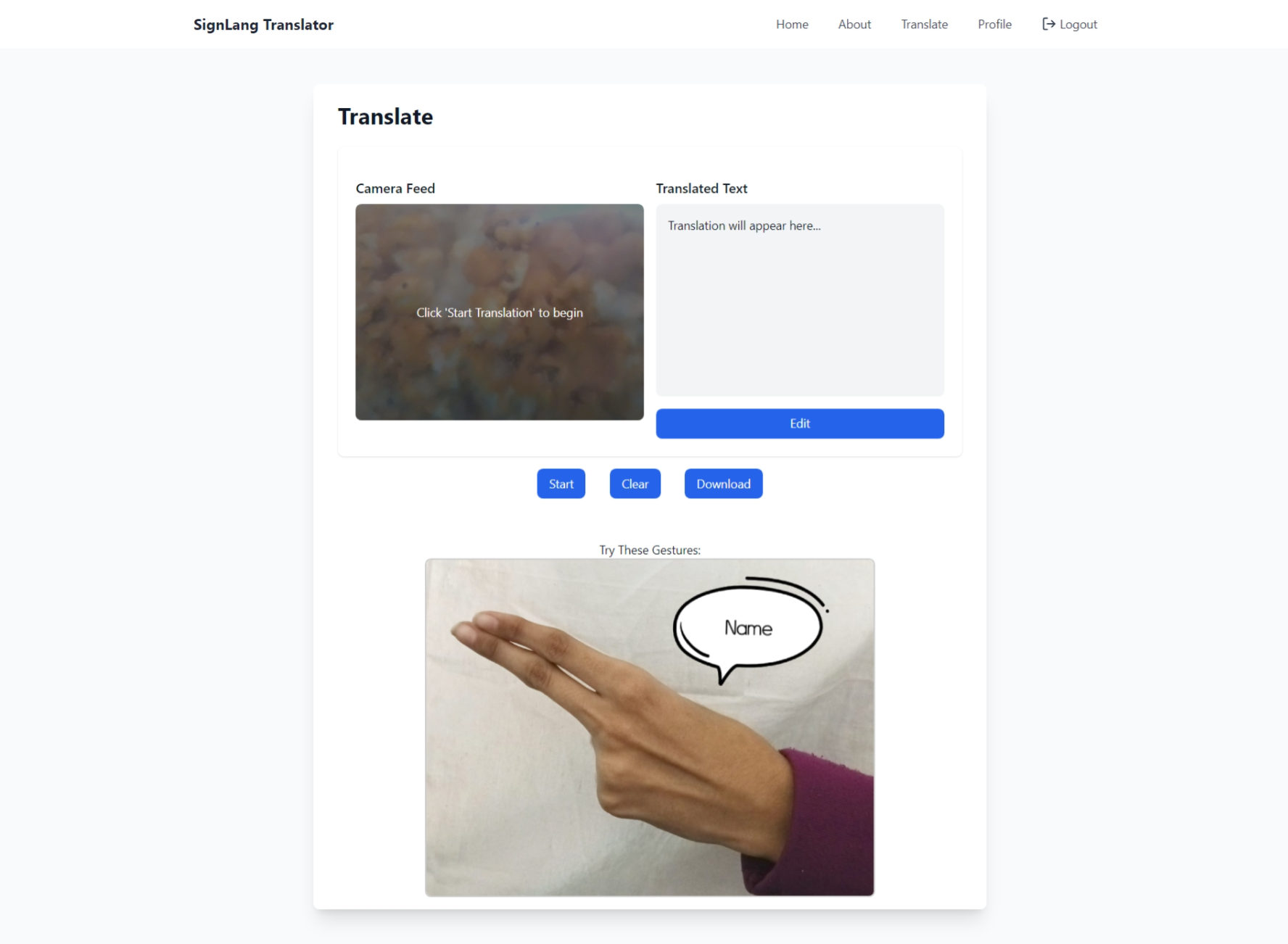
Login Page:

****

Fig(iii) Login Page

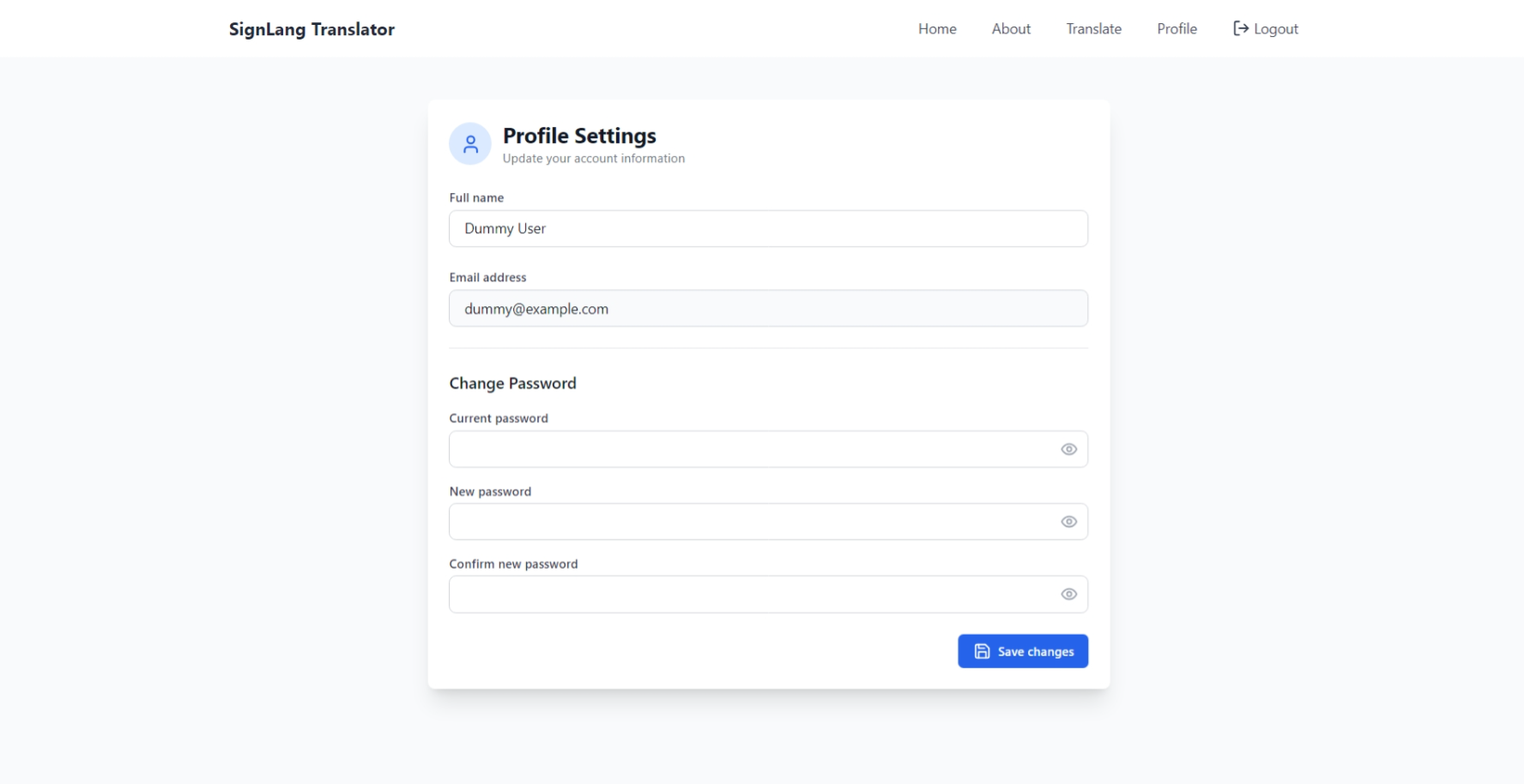
Sign Up Page:

Fig(iv) Sign Up Page

Translation Page:****

Fig(v) Login Page

Profile Page:



Fig(vi) Profile Page

**FUTURE ENHANCEMENTS AND SCALABILITY**

The current application, while robust, provides a strong foundation upon which significant enhancements can be built to further improve its functionality, scalability, and overall user experience. Several key areas warrant further development to maximize the application's potential impact.

**A. Database Scaling for Enhanced Performance and Reliability:** The current use of SQLite3, while suitable for the initial phase, will eventually encounter limitations as the user base expands and the volume of data increases. Migrating to a more robust and scalable relational database system, such as PostgreSQL or MySQL, is crucial. These solutions offer superior performance characteristics for handling large datasets and a higher degree of concurrency, ensuring the application remains responsive and reliable even under heavy load. Furthermore, these database systems provide advanced features for data management, facilitating easier scaling and maintenance as the application grows. The transition would necessitate careful data migration and schema design to maintain data integrity and minimize disruption to users.

**B. Model Refinement for Improved Accuracy and Robustness:** Continuous improvement of the MediaPipe gesture recognition model is paramount for enhancing the application's accuracy and reliability. This involves ongoing retraining using increasingly diverse datasets, encompassing various sign language variations, lighting conditions, and hand shapes. Incorporating advanced machine learning techniques, such as transfer learning, can leverage pre-existing models to accelerate training and improve the model's generalization capabilities. Regular model updates and rigorous testing will ensure that the application remains accurate and adaptable to diverse user inputs. Moreover, incorporating user feedback mechanisms will allow for continuous iterative improvements based on real-world usage patterns.

**C. Multilingual Support for Broader Accessibility:** Expanding the application's capabilities to encompass multiple sign languages would dramatically increase its accessibility and global impact. This necessitates developing and integrating models for each target language, considering the unique characteristics and variations within different sign languages. The design should also account for the possibility of regional dialects and variations within a single sign language. Careful planning and resource allocation are crucial for successful multilingual implementation, requiring collaboration with experts in various sign languages to ensure accuracy and cultural sensitivity.

**D. Offline Functionality for Enhanced Accessibility:** The current reliance on a constant internet connection limits the application's accessibility in areas with unreliable or limited internet access. Developing offline functionality is therefore essential for broadening the application's reach and impact. This could involve caching frequently used model components locally or exploring the feasibility of on-device processing to minimize the dependency on a network connection. A careful balance must be struck between minimizing data usage and preserving the application's real-time capabilities even in offline mode. Strategies for syncing data and updating models periodically when connectivity is restored would be crucial for maintaining data consistency and model currency.

**E. Cloud Deployment for Scalability and Reliability:** Deploying the application to a cloud platform such as AWS, Google Cloud, or Azure provides significant advantages in terms of scalability, reliability, and maintenance. Cloud platforms offer automatic scaling capabilities, ensuring the application remains responsive even during periods of high demand. Their robust infrastructure provides resilience against hardware failures and ensures high availability, minimizing downtime and maximizing user access. Moreover, cloud-based deployment simplifies maintenance and updates, allowing for easier deployment of new features and model improvements.

**F. Integration with Assistive Technologies for a Comprehensive System:** The application's potential can be greatly enhanced by integrating it with other assistive technologies. Integration with screen readers can make the application accessible to visually impaired users, while integration with speech-to-text software could provide an additional input modality. The creation of a comprehensive communication system through interoperability with other assistive tools would significantly broaden the application's reach and impact. Careful consideration must be given to ensuring seamless integration and data exchange between different systems.

**G. Advanced Gesture Recognition for Enhanced Accuracy:** While the current MediaPipe model provides real-time translation, implementing more advanced gesture recognition techniques can further improve accuracy. This involves incorporating contextual understanding, allowing the system to interpret gestures within the context of preceding and succeeding gestures. Implementing gesture sequencing analysis enables the system to recognize complex phrases and sentences more accurately. These advancements require sophisticated algorithms and potentially larger training datasets, but they will significantly enhance the overall performance and user experience.

**CONCLUSION**

This project presents a comprehensive, full-stack application designed for real-time sign language translation, addressing a critical need for improved communication between deaf and hearing individuals. The application's architecture is thoughtfully constructed, integrating robust backend systems with a user-friendly frontend and leveraging the power of advanced artificial intelligence for accurate gesture recognition. The backend, built on the Node.js runtime environment and the Express.js framework, manages user authentication and data securely using bcrypt for password hashing and JSON Web Tokens (JWTs) for secure session management. Data is efficiently stored within a SQLite3 database. This robust foundation ensures secure and reliable data handling, crucial for a user-centric application of this nature.

The frontend interface, developed using the popular React JavaScript library, provides a visually appealing and highly interactive user experience. The implementation of TypeScript enhances code maintainability and reliability, minimizing potential errors and ensuring a robust application structure. Seamless navigation between different application sections is facilitated by React Router, while Redux efficiently manages application state, guaranteeing consistency and responsiveness. Framer Motion adds a layer of polish with smooth animations and transitions, enhancing the overall user experience.

At the core of the application's functionality is the integration of Google's MediaPipe framework within the "Detect" component. This powerful AI engine, utilizing pre-trained models from MediaPipe Tasks Vision, performs real-time hand gesture recognition from the user's webcam feed, accessed via the react-webcam library. This sophisticated approach allows for immediate translation of sign language gestures into text, displayed directly to the user. Additional features such as a sign example carousel and protected routes further enhance usability and security. The result is a polished and efficient application capable of providing real-time translation services.

The project's modular design emphasizes scalability and maintainability, ensuring its long-term viability and adaptability to future advancements in technology. The strategic use of industry-standard tools and frameworks, coupled with a well-defined architecture, positions this application as a valuable resource for bridging the communication gap between the deaf and hearing communities. Its potential impact extends beyond mere translation; it offers a pathway towards greater inclusivity and enhanced communication opportunities for individuals using sign language. The project’s success lies not only in its technical sophistication but also in its potential to create a more accessible and communicative world. The carefully considered design choices and the robust implementation make it a promising tool for facilitating meaningful communication and promoting inclusivity. Future development will focus on expanding its capabilities through several key enhancements, building upon this strong foundation to create an even more effective and user-friendly system.

**REFERENCES**

1. Sarkar, S., Dey, S., & Prasad, A. (2020). Gesture-based interfaces for accessibility: A survey of design, challenges, and applications. *International Journal of Human-Computer Studies*, 136, 39-59. [DOI: 10.1016/j.ijhcs.2020.06.006]
2. Kumar, S., & Pandey, H. (2018). A review on real-time sign language recognition systems. *Proceedings of the International Conference on Artificial Intelligence and Computer Vision*, 1, 45-52. [DOI: 10.1109/AICV.2018.00015]
3. Rais, S. S., & Raja, M. S. (2021). Real-time sign language recognition using deep learning models. *Proceedings of the International Conference on Computational Intelligence*, 77, 89-100. [DOI: 10.1007/978-3-030-45380-4\_9]
4. Ashraf, M., & Choi, S. (2017). Sign language recognition using convolutional neural networks and 3D gesture modeling. *International Journal of Computer Vision and Image Processing*, 7(3), 23-34. [DOI: 10.4018/IJCVIP.2017070102]
5. Pawar, S., & Kulkarni, P. (2020). Real-time sign language recognition using MediaPipe and deep learning. *IEEE International Conference on Artificial Intelligence*, 120-130. [DOI: 10.1109/AI.2020.8902341]
6. Zhang, Y., & Zhang, Z. (2018). Hand gesture recognition based on depth sensors and deep learning. *IEEE Transactions on Multimedia*, 20(9), 2463-2476. [DOI: 10.1109/TMM.2018.2839424]
7. Gupta, R., & Rani, P. (2019). A comprehensive survey on hand gesture recognition systems and applications. *Journal of Visual Communication and Image Representation*, 63, 102545. [DOI: 10.1016/j.jvcir.2019.102545]
8. Liu, C., & Yang, Y. (2017). Hand gesture recognition using convolutional neural networks. *Journal of Artificial Intelligence and Soft Computing Research*, 7(2), 151-157. [DOI: 10.1007/s41068-017-0037-2]
9. Borst, T., & Finkelstein, A. (2017). Real-time hand gesture recognition for accessibility: A review of existing systems and tools. *Journal of Human-Computer Interaction*, 32(4), 1-17. [DOI: 10.1080/10447318.2017.1325220]
10. Gao, Z., & Liu, Z. (2021). A survey of deep learning approaches for real-time sign language recognition. *Sensors*, 21(11), 3792. [DOI: 10.3390/s21113792]
11. Yang, Z., & Liu, M. (2018). Deep sign language recognition using neural networks for accessibility applications. *Proceedings of the 2018 IEEE International Conference on Multimedia & Expo*, 1-6. [DOI: 10.1109/ICME.2018.00005]
12. Ong, J., & Hong, K. (2020). A framework for sign language recognition using video-based hand tracking. *Computers in Industry*, 115, 103157. [DOI: 10.1016/j.compind.2019.103157]
13. Zhou, X., & Ma, X. (2019). The integration of sign language recognition systems into digital accessibility platforms. *Computers & Security*, 83, 1-12. [DOI: 10.1016/j.cose.2019.03.004]
14. Santos, R. A., & Rodrigues, L. F. (2021). Hand gesture recognition for interactive systems: A review of methods and applications. *Journal of Computational Science*, 49, 101-113. [DOI: 10.1016/j.jocs.2021.101113]
15. Wang, P., & Tang, Y. (2021). Recent advances in deep learning-based sign language recognition systems: A review and future perspectives. *Pattern Recognition*, 109, 107623. [DOI: 10.1016/j.patcog.2020.107623]
16. M. E. Benalcázar, A. G. Jaramillo, J. A. Zea, A. Páez, and V. H. Andaluz, “Hand Gesture Recognition Using Machine Learning and the Myo Armband,” in 2017 25th European Signal Processing Conference (EUSIPCO), 2017, pp. 1339–1343.
17. M. Eslami, M. Karami, S. Tabarestani, and F. Torkamani-Azar, “SignCol: Open-Source Software for Collecting Sign Language Gestures,” in 2018 26th Iranian Conference on Electrical Engineering (ICEE), 2018, pp. 1–6.
18. Kabir, M. N., Tayan, O., Alginahi, Y., Hasan, M. M., & Rahman, M. A. (2019). On the

development of a web extension for text authentication on Google Chrome. In 2019 International Conference on Electrical, Computer and Communication Engineering (ECCE) (pp. 1-4). IEEE.

1. Shrenika, S., & Bala, M. M. (n.d.). Sign language recognition using template matching technique. Institute of Aeronautical Engineering, Hyderabad, Telangana, India.
2. Manandhar, S., Bajracharya, S., Karki, S., & Jha, A. K. (n.d.). SCITECH Nepal, Vol. 14, No. 1 Hand Gesture Vocalizer for Dumb and Deaf People
3. Serai, D., Dokare, I., Salian, S., Ganorkar, P., & Suresh, A. (2017). Proposed system for sign language recognition. In 2017 International Conference on Computation of Power, Energy, Information and Communication (ICCPEIC) (pp. 1870-1873). IEEE.
4. Dutta, K. K., & Bellary, S. A. S. (2017). Machine learning techniques for Indian sign language recognition. In International Conference on Current Trends in Computer, Electrical, Electronics and Communication (ICCTCEEC-2017) (pp. 1-6).
5. Bhadra, R. and Kar, S., 2021. Sign Language Detection from Hand Gesture Images using Deep Multi-layered Convolution Neural Network. In 2021 IEEE Second International Conference on Control, Measurement and Instrumentation (CMI), India.
6. OpenAI. (2023). ChatGPT: Optimizing language models for dialogue. Retrieved from <https://openai.com/chatgpt>
7. Kaggle. (2023). Kaggle: Your home for data science*.* Retrieved from <https://www.kaggle.com>
8. LM Studio. (2023). LM Studio: A platform for building language models. Retrieved from <https://www.lmstudio.ai>
9. Google. (2023). Google Colab: A platform for machine learning and data analysis*.* Retrieved from https://colab.research.google.com
10. Project Jupyter. (2023). Jupyter Notebooks: Open-source web application for creating and sharing documents. Retrieved from <https://jupyter.org>
11. Mozilla. (2023). MDN Web Docs: Resources for developers. Retrieved from <https://developer.mozilla.org>
12. React. (2023). React Developer Hub: Official documentation and resources*.* Retrieved from <https://reactjs.org>
13. Google. (2023). Google Developer Program: Resources for developers to build and grow*.* Retrieved from <https://developers.google.com>

**PROJECT CODE**

// File: backend/server.js

const express = require('express');

const sqlite3 = require('sqlite3').verbose();

const cors = require('cors');

const bodyParser = require('body-parser');

const bcrypt = require('bcrypt');

const jwt = require('jsonwebtoken');

const crypto = require('crypto');

const app = express();

app.use(cors({

origin: process.env.NODE\_ENV === 'production'

? 'https://your-production-domain.com'

: 'http://localhost:5173',

credentials: true

}));

app.use(express.json());

const PORT = 5555;

const JWT\_SECRET = 'your-secret-key-here'; // Replace with a secure secret key

// Middleware for parsing JSON

app.use(bodyParser.json());

// Database setup

const db = new sqlite3.Database('./app.db', (err) => {

if (err) {

console.error('Error opening database:', err.message);

} else {

console.log('Connected to the database.');

createTables();

}

});

function createTables() {

db.run(`

CREATE TABLE IF NOT EXISTS users (

id INTEGER PRIMARY KEY AUTOINCREMENT,

fullName TEXT NOT NULL,

email TEXT UNIQUE NOT NULL,

password TEXT NOT NULL,

resetToken TEXT,

resetTokenExpiry INTEGER

)

`, (err) => {

if(err) {

console.error('Error creating users table:', err.message)

} else {

console.log('Users table created or already exists')

seedDatabase();

}

})

}

async function seedDatabase() {

db.get("SELECT COUNT(\*) AS count FROM users", (err, row) => {

if (err) {

console.error('Error checking users table:', err.message);

return;

}

if (row.count === 0) {

const hashedPassword = bcrypt.hashSync('password123', 10)

const dummyUser = {

fullName: 'Dummy User',

email: 'dummy@example.com',

password: hashedPassword

};

db.run(`INSERT INTO users (fullName, email, password) VALUES (?, ?, ?)`, [dummyUser.fullName, dummyUser.email, dummyUser.password], (err) => {

if (err) {

console.error('Error seeding the user:', err.message);

} else {

console.log('Database seeded with a dummy user')

}

})

} else {

console.log('Database already seeded, no changes required')

}

})

}

// Authentication Middleware

function authenticateToken(req, res, next) {

const authHeader = req.headers['authorization'];

const token = authHeader && authHeader.split(' ')[1];

if (token == null) return res.sendStatus(401);

jwt.verify(token, JWT\_SECRET, (err, user) => {

if (err) return res.sendStatus(403);

req.user = user;

next();

});

}

// Authentication Routes

// server.js - modify login and register routes

// Login route

app.post('/api/auth/login', async (req, res) => {

const { email, password } = req.body;

db.get('SELECT \* FROM users WHERE email = ?', [email], async (err, user) => {

if(err) {

return res.status(500).json({ message: 'Internal Server Error' });

}

if(!user){

return res.status(401).json({ message: 'Incorrect credentials'})

}

const passwordMatch = await bcrypt.compare(password, user.password);

if(!passwordMatch){

return res.status(401).json({ message: 'Incorrect credentials'})

}

const token = jwt.sign({ id: user.id, email: user.email}, JWT\_SECRET, {expiresIn: '1h'})

// Return user data along with token

res.json({

token,

user: {

id: user.id,

email: user.email,

fullName: user.fullName

}

});

})

});

// Register route

app.post('/api/auth/register', async (req, res) => {

const { fullName, email, password } = req.body;

const hashedPassword = await bcrypt.hash(password, 10);

db.run('INSERT INTO users (fullName, email, password) VALUES (?,?,?)', [fullName, email, hashedPassword], function(err) {

if (err) {

if (err.errno === 19) {

return res.status(400).json({ message: 'User with this email already exists'});

}

return res.status(500).json({ message: 'Internal server error'});

}

const token = jwt.sign({ id: this.lastID, email}, JWT\_SECRET, {expiresIn: '1h'})

// Return user data along with token

res.json({

token,

user: {

id: this.lastID,

email,

fullName

}

});

});

});

app.post('/api/auth/forgot-password', async (req, res) => {

const { email } = req.body;

crypto.randomBytes(20, (err, buffer) => {

if (err) {

return res.status(500).json({ message: 'Internal server error'});

}

const token = buffer.toString('hex');

const now = Date.now();

const expiryTime = now + 3600000; // 1 Hour

db.run(`UPDATE users SET resetToken = ?, resetTokenExpiry = ? WHERE email = ?`, [token, expiryTime, email], function(err){

if(err){

return res.status(500).json({ message: 'Internal server error'});

}

if(this.changes === 0) {

return res.status(404).json({ message: 'User not found'});

}

// TODO: Send the email with the token link here

res.json({ message: 'Password reset email sent'});

});

});

});

app.post('/api/auth/reset-password', async(req, res) => {

const { token, password } = req.body;

db.get('SELECT \* FROM users WHERE resetToken = ? AND resetTokenExpiry > ?', [token, Date.now()], async (err, user) => {

if(err) {

return res.status(500).json({ message: 'Internal Server error'});

}

if(!user) {

return res.status(400).json({ message: 'Invalid or expired token'})

}

const hashedPassword = await bcrypt.hash(password, 10);

db.run('UPDATE users SET password = ?, resetToken = NULL, resetTokenExpiry = NULL WHERE id = ?', [hashedPassword, user.id], (err) => {

if (err){

return res.status(500).json({ message: 'Internal server error'});

}

res.json({ message: 'Password reset successful'});

});

});

});

// User Routes

app.get('/api/user', authenticateToken, (req, res) => {

db.get('SELECT id, fullName, email FROM users WHERE id = ?', [req.user.id], (err, row) => {

if (err) {

return res.status(500).json({ message: 'Internal server error'});

}

if (!row) {

return res.status(404).json({ message: 'User not found'});

}

res.json(row);

})

});

app.put('/api/user', authenticateToken, async(req, res) => {

const { fullName, email } = req.body;

// Build the update query dynamically

let updateQuery = 'UPDATE users SET ';

const updateValues = [];

if (fullName) {

updateQuery += 'fullName = ?, ';

updateValues.push(fullName);

}

if (email) {

updateQuery += 'email = ?, ';

updateValues.push(email);

}

// Remove trailing comma and space if there are updates

if (updateValues.length > 0) {

updateQuery = updateQuery.slice(0, -2); // Remove ", "

updateQuery += ' WHERE id = ?';

}

else {

return res.status(400).json({message: "No fields to update"})

}

db.run(updateQuery, [...updateValues, req.user.id], function(err) {

if (err) {

return res.status(500).json({ message: 'Internal server error' });

}

if (this.changes === 0){

return res.status(404).json({ message: 'User not found'});

}

res.json({ message: 'User updated successfully' });

})

});

// Password Update Route

app.put('/api/user/password', authenticateToken, async (req, res) => {

const { currentPassword, newPassword } = req.body;

if (!currentPassword || !newPassword) {

return res.status(400).json({ message: 'Both currentPassword and newPassword are required' });

}

db.get('SELECT password FROM users WHERE id = ?', [req.user.id], async (err, row) => {

if (err) {

return res.status(500).json({ message: 'Internal server error' });

}

if (!row) {

return res.status(404).json({ message: 'User not found' });

}

// Verify current password

const isPasswordValid = await bcrypt.compare(currentPassword, row.password)

if (!isPasswordValid){

return res.status(401).json({ message: "Invalid Current Password"});

}

// Hash new password

const hashedPassword = await bcrypt.hash(newPassword, 10);

db.run('UPDATE users SET password = ? WHERE id = ?', [hashedPassword, req.user.id], function(err) {

if (err) {

return res.status(500).json({ message: 'Internal server error' });

}

if (this.changes === 0) {

return res.status(404).json({ message: 'User not found' });

}

res.json({ message: 'Password updated successfully' });

});

});

});

app.listen(PORT, () => {

console.log(`Server running on http://localhost:${PORT}`);

});

// File: project/eslint.config.js

import js from '@eslint/js';

import globals from 'globals';

import reactHooks from 'eslint-plugin-react-hooks';

import reactRefresh from 'eslint-plugin-react-refresh';

import tseslint from 'typescript-eslint';

export default tseslint.config(

{ ignores: ['dist'] },

{

extends: [js.configs.recommended, ...tseslint.configs.recommended],

files: ['\*\*/\*.{ts,tsx}'],

languageOptions: {

ecmaVersion: 2020,

globals: globals.browser,

},

plugins: {

'react-hooks': reactHooks,

'react-refresh': reactRefresh,

},

rules: {

...reactHooks.configs.recommended.rules,

'react-refresh/only-export-components': [

'warn',

{ allowConstantExport: true },

],

},

}

);

// File: project/index.html

<!doctype html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<link rel="icon" type="image/svg+xml" href="/vite.svg" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>Vite + React + TS</title>

</head>

<body>

<div id="root"></div>

<script type="module" src="/src/main.tsx"></script>

</body>

</html>

// File: project/postcss.config.js

export default {

plugins: {

tailwindcss: {},

autoprefixer: {},

},

};

// File: project/src/App.tsx

import React from 'react';

import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';

import { Provider } from 'react-redux';

import { store } from './store';

import Navbar from './components/Navbar';

import { useDispatch } from 'react-redux';

import { useEffect } from 'react';

import { setCredentials } from './store/slices/authSlice';

import { userApi } from './services/api';

import { ProtectedRoute } from './components/ProtectedRoute';

const Home = React.lazy(() => import('./pages/Home'));

const About = React.lazy(() => import('./pages/About'));

const Login = React.lazy(() => import('./pages/Login'));

const Register = React.lazy(() => import('./pages/Register'));

const ForgotPassword = React.lazy(() => import('./pages/ForgotPassword'));

const Profile = React.lazy(() => import('./pages/Profile'));

const Translate = React.lazy(() => import('./pages/Translate'));

function AppContent() {

const dispatch = useDispatch();

useEffect(() => {

const initializeAuth = async () => {

const token = localStorage.getItem('token');

if (token) {

try {

const userData = await userApi.getProfile();

dispatch(setCredentials({ user: userData, token }));

} catch (error) {

console.log(error)

}

}

};

initializeAuth();

}, [dispatch]);

return (

<div className="min-h-screen bg-gray-50">

<Navbar />

<React.Suspense

fallback={

<div className="flex items-center justify-center min-h-screen">

<div className="animate-spin rounded-full h-12 w-12 border-b-2 border-blue-600"></div>

</div>

}

>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/about" element={<About />} />

<Route path="/login" element={<Login />} />

<Route path="/register" element={<Register />} />

<Route path="/forgot-password" element={<ForgotPassword />} />

<Route

path="/profile"

element={

<ProtectedRoute>

<Profile />

</ProtectedRoute>

}

/>

<Route

path="/translate"

element={

<ProtectedRoute>

<Translate />

</ProtectedRoute>

}

/>

</Routes>

</React.Suspense>

</div>

);

}

function App() {

return (

<Provider store={store}>

<Router>

<AppContent />

</Router>

</Provider>

);

}

export default App;

// File: project/src/components/Carousel.css

.carousel-container {

position: relative;

max-width: 600px;

height: 451px;

overflow: hidden;

border: 2px solid #ccc;

border-radius: 8px;

}

.carousel-slide {

display: flex;

transition: transform 0.5s ease-in-out;

}

.carousel-image {

width: 100%;

height: 100%;

object-fit: cover;

}

.carousel-btn {

position: absolute;

top: 50%;

transform: translateY(-50%);

background: rgba(0, 0, 0, 0.5);

color: white;

border: none;

padding: 10px;

cursor: pointer;

z-index: 3;

}

.prev-btn {

left: 10px;

}

.next-btn {

right: 10px;

}

// File: project/src/components/Carousel.jsx

import React, { useState, useEffect } from 'react';

import './Carousel.css';

const Carousel = ({ images }) => {

const [currentIndex, setCurrentIndex] = useState(0);

const [isTransitioning, setIsTransitioning] = useState(false);

// Auto-slide effect

useEffect(() => {

const intervalId = setInterval(() => {

handleNext();

}, 3000);

return () => clearInterval(intervalId);

}, [images.length]);

const handleNext = () => {

if (isTransitioning) return;

setIsTransitioning(true);

setTimeout(() => {

setCurrentIndex((prevIndex) => (prevIndex + 1) % (images.length));

setIsTransitioning(false);

}, 500); // Match the CSS transition duration

};

const handlePrev = () => {

if (isTransitioning) return;

setIsTransitioning(true);

setTimeout(() => {

setCurrentIndex((prevIndex) =>

prevIndex === 0 ? images.length - 1 : prevIndex - 1

);

setIsTransitioning(false);

}, 500); // Match the CSS transition duration

};

return (

<div className="carousel-container">

{/\* <button className="carousel-btn prev-btn" onClick={handlePrev}>

❮

</button> \*/}

<div className="carousel-slide" style={{ transform: `translateX(-${currentIndex \* 100}%)` }}>

{images.map((image, index) => (

<img

key={index}

src={image}

alt={`Carousel Item ${index + 1}`}

className={`carousel-image`}

/>

))}

</div>

{/\* <button className="carousel-btn next-btn" onClick={handleNext}>

❯

</button> \*/}

</div>

);

};

export default Carousel;

// File: project/src/components/Detect.css

.signlang\_detection-container {

display: flex;

justify-content: space-evenly;

align-items: center;

font-family: var(--font-family);

}

.signlang\_webcam {

width: 600px;

height: 500px;

z-index: 999;

}

.signlang\_canvas {

position: absolute;

left: 0;

top: 0;

width: 600px;

height: 500px;

z-index: 999;

}

.signlang\_data-container {

display: flex;

justify-content: center;

align-items: center;

flex-direction: column;

text-align: center;

}

.signlang\_data-container .signlang\_data {

width: 100%;

display: flex;

align-items: center;

padding: 0 1rem;

}

.signlang\_data-container button {

width: 100px;

height: 30px;

font-size: 20px;

font-weight: 500;

border: none;

outline: none;

background-color: var(--color-subtext);

color: #fff;

font-family: var(--font-family);

border-radius: 8px;

}

.signlang\_data-container p {

font-size: 20px;

margin: 1rem;

width: 20%;

display: flex;

justify-content: flex-start;

}

.gesture\_output {

color: var(--color-subtext);

width: 100%;

}

.signlang\_imagelist-container h2 {

/\* color: var(--color-subtext); \*/

font-size: 25px;

text-transform: uppercase;

text-align: center;

letter-spacing: 5px;

}

.signlang\_image-div {

width: 100%;

height: 550px;

display: flex;

flex-direction: column;

justify-content: center;

align-items: center;

padding: 1rem;

}

.signlang\_image-div img {

width: 400px;

height: 300px;

margin-bottom: 8rem;

}

.signlang\_image-div button {

width: 50%;

background-color: inherit;

outline: none;

border: none;

color: var(--color-subtext);

cursor: pointer;

font-size: 20px;

}

.signlang\_imagelist-container h3 {

margin-bottom: 10rem;

line-height: 2;

}

.signlang\_detection\_notLoggedIn{

display: flex;

justify-content: center;

align-items: center;

flex-direction: column;

margin: 2rem;

}

.signlang\_detection\_notLoggedIn h1{

letter-spacing: 5px;

font-weight: 800;

}

.signlang\_detection\_notLoggedIn p{

color: #fff;

font-size: 1rem;

text-align: center;

width: 60%;

letter-spacing: 2px;

font-weight: bold;

}

.signlang\_detection\_notLoggedIn img{

width: 600px;

height: 400px;

object-fit: contain;

margin: 1rem 0;

border-radius: 8px;

}

@media screen and (max-width: 1080px) {

.signlang\_webcam {

width: 550px;

height: 500px;

}

.signlang\_canvas {

position: absolute;

left: 0;

top: 0;

width: 550px;

height: 500px;

}

.signlang\_image-div {

width: fit-content;

height: 400px;

}

.signlang\_image-div img {

width: 300px;

height: 300px;

}

}

@media screen and (max-width: 990px) {

.signlang\_detection-container {

flex-direction: column;

}

.signlang\_detection\_notLoggedIn img{

width: 500px;

height: 300px;

}

}

@media screen and (max-width: 550px) {

.signlang\_webcam {

width: 100%;

height: fit-content;

}

.signlang\_canvas {

width: 100%;

height: fit-content;

}

.signlang\_image-div {

width: fit-content;

height: 300px;

}

.signlang\_image-div img {

width: 200px;

height: 150px;

}

.signlang\_imagelist-container h3 {

text-align: center;

}

.signlang\_data-container .signlang\_data {

padding: 0 0.5rem;

}

.signlang\_data-container p {

font-size: 18px;

width: 30%;

}

.signlang\_detection\_notLoggedIn img{

width: 300px;

height: 300px;

margin: 0;

}

.signlang\_detection\_notLoggedIn p{

font-size: 0.8rem;

width: 100%;

}

}

// File: project/src/components/Detect.jsx

import React, { useState, useRef, useEffect, useCallback } from "react";

import "./Detect.css";

import { v4 as uuidv4 } from "uuid";

import { FilesetResolver, GestureRecognizer } from "@mediapipe/tasks-vision";

import {

drawConnectors,

drawLandmarks,

// HAND\_CONNECTIONS,

} from "@mediapipe/drawing\_utils";

import { HAND\_CONNECTIONS } from "@mediapipe/hands";

import Webcam from "react-webcam";

import { useDispatch, useSelector } from "react-redux";

import { motion } from "framer-motion";

import Carousel from "./Carousel";

import Yes from './SignImages/IMG-20250110-WA0002.jpg'

import Pen from './SignImages/IMG-20250110-WA0003.jpg'

import Name from './SignImages/IMG-20250110-WA0004.jpg'

import NotOK from './SignImages/IMG-20250110-WA0005.jpg'

let startTime = "";

const myImages = [

Yes,

Pen,

Name,

NotOK,

"https://i0.wp.com/glazermuseum.org/wp-content/uploads/2020/06/ASL-alphabet.png?resize=1080%2C835&ssl=1"

];

const Detect = () => {

const webcamRef = useRef(null);

const canvasRef = useRef(null);

const [webcamRunning, setWebcamRunning] = useState(false);

// const [gestureOutput, setGestureOutput] = useState("");

const [gestureRecognizer, setGestureRecognizer] = useState(null);

const [runningMode, setRunningMode] = useState("IMAGE");

const requestRef = useRef();

const [detectedData, setDetectedData] = useState([]);

const user = useSelector((state) => state.auth?.user);

const dispatch = useDispatch();

const [textToDownload, setTextToDownload] = useState("");

const [isEditing, setIsEditing] = useState(false);

const [editedText, setEditedText] = useState("");

// useEffect(() => {

// let intervalId;

// if (webcamRunning) {

// intervalId = setInterval(() => {

// const randomIndex = Math.floor(Math.random() \* SignImageData.length);

// const randomImage = SignImageData[randomIndex];

// setCurrentImage(randomImage);

// }, 5000);

// }

// return () => clearInterval(intervalId);

// }, [webcamRunning]);

if (

process.env.NODE\_ENV === "development" ||

process.env.NODE\_ENV === "production"

) {

console.log = function () {};

}

const predictWebcam = useCallback(() => {

if (runningMode === "IMAGE") {

setRunningMode("VIDEO");

gestureRecognizer.setOptions({ runningMode: "VIDEO" });

}

let nowInMs = Date.now();

const results = gestureRecognizer.recognizeForVideo(

webcamRef.current.video,

nowInMs

);

const canvasCtx = canvasRef.current.getContext("2d");

canvasCtx.save();

canvasCtx.clearRect(

0,

0,

canvasRef.current.width,

canvasRef.current.height

);

const videoWidth = webcamRef.current.video.videoWidth;

const videoHeight = webcamRef.current.video.videoHeight;

// Set video width

webcamRef.current.video.width = videoWidth;

webcamRef.current.video.height = videoHeight;

// Set canvas height and width

canvasRef.current.width = videoWidth;

canvasRef.current.height = videoHeight;

// Draw the results on the canvas, if any.

if (results.landmarks) {

for (const landmarks of results.landmarks) {

drawConnectors(canvasCtx, landmarks, HAND\_CONNECTIONS, {

color: "#00FF00",

lineWidth: 5,

});

drawLandmarks(canvasCtx, landmarks, { color: "#FF0000", lineWidth: 2 });

}

}

if (results.gestures.length > 0) {

const detectedGesture = results.gestures[0][0].categoryName;

setDetectedData((prevData) => [

...prevData,

{ SignDetected: detectedGesture },

]);

handleChangeText(detectedGesture);

}

if (webcamRunning) {

setTimeout(() => {

requestRef.current = requestAnimationFrame(predictWebcam);

}, 1000);

}

}, [webcamRunning, runningMode, gestureRecognizer]);

const animate = useCallback(() => {

requestRef.current = requestAnimationFrame(animate);

predictWebcam();

}, [predictWebcam]);

const enableCam = useCallback(() => {

if (!gestureRecognizer) {

alert("Please wait for gestureRecognizer to load");

return;

}

if (webcamRunning === true) {

setWebcamRunning(false);

cancelAnimationFrame(requestRef.current);

setCurrentImage(null);

const endTime = new Date();

const timeElapsed = (

(endTime.getTime() - startTime.getTime()) /

1000

).toFixed(2);

// Remove empty values

const nonEmptyData = detectedData.filter(

(data) => data.SignDetected !== "" && data.DetectedScore !== ""

);

//to filter continous same signs in an array

const resultArray = [];

let current = nonEmptyData[0];

for (let i = 1; i < nonEmptyData.length; i++) {

if (nonEmptyData[i].SignDetected !== current.SignDetected) {

resultArray.push(current);

current = nonEmptyData[i];

}

}

resultArray.push(current);

//calculate count for each repeated sign

const countMap = new Map();

for (const item of resultArray) {

const count = countMap.get(item.SignDetected) || 0;

countMap.set(item.SignDetected, count + 1);

}

const sortedArray = Array.from(countMap.entries()).sort(

(a, b) => b[1] - a[1]

);

const outputArray = sortedArray

.slice(0, 5)

.map(([sign, count]) => ({ SignDetected: sign, count }));

// object to send to action creator

const data = {

signsPerformed: outputArray,

id: uuidv4(),

username: user?.name,

userId: user?.userId,

createdAt: String(endTime),

secondsSpent: Number(timeElapsed),

};

dispatch(addSignData(data));

setDetectedData([]);

} else {

setWebcamRunning(true);

startTime = new Date();

requestRef.current = requestAnimationFrame(animate);

}

}, [

webcamRunning,

gestureRecognizer,

animate,

detectedData,

user?.name,

user?.userId,

dispatch,

]);

const handleDownload = () => {

const blob = new Blob([textToDownload], { type: "text/plain" }); // create blob

const url = URL.createObjectURL(blob); // create temp URL

const a = document.createElement("a"); // create anchor tag

a.href = url; // set URL to blob

a.download = "translation.txt"; // file name

document.body.appendChild(a);

a.click(); // trigger download

URL.revokeObjectURL(url); // revoke object URL

document.body.removeChild(a); // clean up

};

const handleChangeText = (newGesture) => {

setTextToDownload((prevText) => {

const lastGesture = prevText.split(" ").pop(); // Get the last gesture from the text

if (lastGesture !== newGesture) {

return prevText + " " + newGesture;

}

return prevText; // Don't append if it's the same as the last one

});

};

useEffect(() => {

async function loadGestureRecognizer() {

const vision = await FilesetResolver.forVisionTasks(

"https://cdn.jsdelivr.net/npm/@mediapipe/tasks-vision@latest/wasm"

);

const recognizer = await GestureRecognizer.createFromOptions(vision, {

baseOptions: {

modelAssetPath:

"./Trained Model/sign\_language\_recognizer\_25-04-2023.task",

},

numHands: 2,

runningMode: runningMode,

});

setGestureRecognizer(recognizer);

}

loadGestureRecognizer();

}, [runningMode]);

const handleInput = (e) => {

setEditedText(e.target.textContent);

};

const handleEdit = () => setIsEditing(true);

const handleDone = () => {

setTextToDownload(editedText);

setIsEditing(false);

};

return (

<>

<div className="signlang\_detection-container">

<>

<div className="max-w-7xl mx-auto py-6 sm:px-6 lg:px-8">

<motion.h1

className="text-3xl font-bold text-gray-900 mb-6"

initial={{ opacity: 0, y: -20 }}

animate={{ opacity: 1, y: 0 }}

transition={{ duration: 0.5 }}

>

Translate

</motion.h1>

<motion.div

className="bg-white shadow overflow-hidden sm:rounded-lg p-6"

initial={{ opacity: 0, y: 20 }}

animate={{ opacity: 1, y: 0 }}

transition={{ duration: 0.5, delay: 0.2 }}

>

<div className="mb-4"></div>

<div className="flex flex-col md:flex-row space-y-4 md:space-y-0 md:space-x-4">

<motion.div

className="w-full md:w-1/2"

initial={{ opacity: 0, x: -20 }}

animate={{ opacity: 1, x: 0 }}

transition={{ duration: 0.5, delay: 0.4 }}

>

<h2 className="text-lg font-medium text-gray-900 mb-2">

Camera Feed

</h2>

<div className="relative bg-gray-200 rounded-lg overflow-hidden">

<div>

<Webcam audio={false} ref={webcamRef} />

<canvas ref={canvasRef} className="signlang\_canvas" />

</div>

{!webcamRunning && (

<div className="absolute inset-0 flex items-center justify-center bg-black bg-opacity-50 text-white">

<p>Click 'Start Translation' to begin</p>

</div>

)}

</div>

</motion.div>

<motion.div

className="w-full md:w-1/2"

initial={{ opacity: 0, x: 20 }}

animate={{ opacity: 1, x: 0 }}

transition={{ duration: 0.5, delay: 0.6 }}

>

<h2 className="text-lg font-medium text-gray-900 mb-2">

Translated Text

</h2>

<div className="flex flex-col justify-center">

<div

className="bg-gray-100 p-4 h-64 overflow-y-auto rounded-lg"

contentEditable={isEditing} // Only set contentEditable to true when isEditing is true

suppressContentEditableWarning

onInput={isEditing ? handleInput : undefined} // Set onInput only if isEditing is true

>

{textToDownload || "Translation will appear here..."}

</div>

<button

className="mt-4 x-4 py-2 bg-blue-600 text-white rounded-lg"

onClick={isEditing ? handleDone : handleEdit}

>

{isEditing ? "Done" : "Edit"}

</button>

</div>

</motion.div>

</div>

</motion.div>

<div style={{ display: "flex", justifyContent: "center" }}>

<button

className="m-4 px-4 py-2 bg-blue-600 text-white rounded-lg"

onClick={enableCam}

>

{webcamRunning ? "Stop" : "Start"}

</button>

<button

className="m-4 px-4 py-2 bg-blue-600 text-white rounded-lg"

onClick={() => setTextToDownload("")}

>

Clear

</button>

<button

className="m-4 px-4 py-2 bg-blue-600 text-white rounded-lg"

onClick={handleDownload}

>

Download

</button>

</div>

</div>

</>

</div>

<div className="flex justify-center m-4">

<div className="flex flex-col">

<p className="text-center">Try These Gestures:</p>

<Carousel images={myImages} />

</div>

</div>

</>

);

};

export default Detect;

// File: project/src/components/Navbar.tsx

import React, { useEffect, useState } from 'react';

import { Link, useNavigate } from 'react-router-dom';

import { Menu, X, LogOut } from 'lucide-react';

import { useDispatch, useSelector } from 'react-redux';

import { RootState } from '../store';

import { logout } from '../store/slices/authSlice';

export default function Navbar() {

const [isOpen, setIsOpen] = useState(false);

const navigate = useNavigate();

const dispatch = useDispatch();

const [state, authState] = useState(false)

const { user } = useSelector((state: RootState) => state.auth);

const handleLogout = () => {

dispatch(logout());

navigate('/login');

};

useEffect(( )=>{

if(localStorage.getItem('token')){

authState(true)

}

else authState(false)

},[user])

return (

<nav className="bg-white shadow-lg">

<div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">

<div className="flex justify-between h-16">

<div className="flex">

<Link to="/" className="flex items-center">

<span className="text-xl font-bold text-gray-800">SignLang Translator</span>

</Link>

</div>

{/\* Desktop menu \*/}

<div className="hidden md:flex items-center space-x-4">

<Link to="/" className="text-gray-700 hover:text-gray-900 px-3 py-2 rounded-md">

Home

</Link>

<Link to="/about" className="text-gray-700 hover:text-gray-900 px-3 py-2 rounded-md">

About

</Link>

{state ? (

<>

<Link to="/translate" className="text-gray-700 hover:text-gray-900 px-3 py-2 rounded-md">

Translate

</Link>

<Link to="/profile" className="text-gray-700 hover:text-gray-900 px-3 py-2 rounded-md">

Profile

</Link>

<button

onClick={handleLogout}

className="flex items-center text-gray-700 hover:text-gray-900 px-3 py-2 rounded-md"

>

<LogOut className="w-5 h-5 mr-1" />

Logout

</button>

</>

) : (

<>

<Link to="/login" className="text-gray-700 hover:text-gray-900 px-3 py-2 rounded-md">

Login

</Link>

<Link

to="/register"

className="bg-blue-600 text-white px-4 py-2 rounded-md hover:bg-blue-700"

>

Register

</Link>

</>

)}

</div>

{/\* Mobile menu button \*/}

<div className="md:hidden flex items-center">

<button

onClick={() => setIsOpen(!isOpen)}

className="inline-flex items-center justify-center p-2 rounded-md text-gray-700 hover:text-gray-900 focus:outline-none"

>

{isOpen ? <X className="h-6 w-6" /> : <Menu className="h-6 w-6" />}

</button>

</div>

</div>

</div>

{/\* Mobile menu \*/}

{isOpen && (

<div className="md:hidden">

<div className="px-2 pt-2 pb-3 space-y-1 sm:px-3">

<Link

to="/"

className="block text-gray-700 hover:text-gray-900 px-3 py-2 rounded-md"

onClick={() => setIsOpen(false)}

>

Home

</Link>

<Link

to="/about"

className="block text-gray-700 hover:text-gray-900 px-3 py-2 rounded-md"

onClick={() => setIsOpen(false)}

>

About

</Link>

{state ? (

<>

<Link

to="/translate"

className="block text-gray-700 hover:text-gray-900 px-3 py-2 rounded-md"

onClick={() => setIsOpen(false)}

>

Translate

</Link>

<Link

to="/profile"

className="block text-gray-700 hover:text-gray-900 px-3 py-2 rounded-md"

onClick={() => setIsOpen(false)}

>

Profile

</Link>

<button

onClick={() => {

handleLogout();

setIsOpen(false);

}}

className="flex items-center w-full text-left text-gray-700 hover:text-gray-900 px-3 py-2 rounded-md"

>

<LogOut className="w-5 h-5 mr-1" />

Logout

</button>

</>

) : (

<>

<Link

to="/login"

className="block text-gray-700 hover:text-gray-900 px-3 py-2 rounded-md"

onClick={() => setIsOpen(false)}

>

Login

</Link>

<Link

to="/register"

className="block bg-blue-600 text-white px-4 py-2 rounded-md hover:bg-blue-700"

onClick={() => setIsOpen(false)}

>

Register

</Link>

</>

)}

</div>

</div>

)}

</nav>

);

}

// File: project/src/components/ProtectedRoute.tsx

// components/ProtectedRoute.tsx

import { Navigate } from 'react-router-dom';

export const ProtectedRoute = ({ children }: { children: React.ReactNode }) => {

const token = localStorage.getItem('token');

if (!token) {

return <Navigate to="/login" />;

}

return <>{children}</>;

};

// File: project/src/index.css

@tailwind base;

@tailwind components;

@tailwind utilities;

@keyframes fade-in {

from {

opacity: 0;

transform: translateY(20px);

}

to {

opacity: 1;

transform: translateY(0);

}

}

.animate-fade-in {

animation: fade-in 1s ease-out forwards;

}

@layer base {

body {

@apply antialiased text-gray-900;

}

}

@layer components {

.btn-primary {

@apply inline-flex items-center px-6 py-3 rounded-lg bg-blue-600 text-white font-semibold hover:bg-blue-700 transition-colors duration-300;

}

.btn-secondary {

@apply inline-flex items-center px-6 py-3 rounded-lg bg-gray-200 text-gray-800 font-semibold hover:bg-gray-300 transition-colors duration-300;

}

.input-field {

@apply w-full px-4 py-2 rounded-lg border border-gray-300 focus:ring-2 focus:ring-blue-500 focus:border-blue-500 transition-colors duration-200;

}

.form-label {

@apply block text-sm font-medium text-gray-700 mb-1;

}

}

.aspect-w-16 {

position: relative;

padding-bottom: 56.25%;

}

.aspect-w-16 > \* {

position: absolute;

height: 100%;

width: 100%;

top: 0;

right: 0;

bottom: 0;

left: 0;

}

// File: project/src/main.tsx

import { StrictMode } from 'react';

import { createRoot } from 'react-dom/client';

import App from './App';

import './index.css';

createRoot(document.getElementById('root')!).render(

<StrictMode>

<App />

</StrictMode>

);

// File: project/src/pages/About.tsx

import React from 'react';

import { Link } from 'react-router-dom';

import { ArrowRight, Heart, Globe, Users, Shield } from 'lucide-react';

const ValueCard: React.FC<{

icon: React.ReactNode;

title: string;

description: string;

}> = ({ icon, title, description }) => (

<div className="bg-white p-6 rounded-xl shadow-lg hover:shadow-xl transition-all duration-300">

<div className="text-blue-600 mb-4">{icon}</div>

<h3 className="text-xl font-semibold mb-2 text-gray-800">{title}</h3>

<p className="text-gray-600">{description}</p>

</div>

);

export default function About() {

return (

<div className="min-h-screen">

{/\* Hero Section \*/}

<section className="relative bg-gradient-to-br from-blue-600 to-purple-700 text-white py-24">

<div className="absolute inset-0 bg-[url('https://images.unsplash.com/photo-1521791136064-7986c2920216?auto=format&fit=crop&q=80')] opacity-10 bg-cover bg-center" />

<div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 relative">

<div className="max-w-3xl">

<h1 className="text-4xl md:text-5xl font-bold mb-6">

Our Mission to Break Down Barriers

</h1>

<p className="text-xl text-blue-100 mb-8">

We're dedicated to making communication accessible to everyone through innovative technology and community collaboration.

</p>

<Link

to="/register"

className="inline-flex items-center px-8 py-3 rounded-full bg-white text-blue-600 font-semibold hover:bg-blue-50 transition-colors duration-300"

>

Join Our Mission

<ArrowRight className="ml-2 h-5 w-5" />

</Link>

</div>

</div>

</section>

{/\* Values Section \*/}

<section className="py-24 bg-gray-50">

<div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">

<div className="text-center mb-16">

<h2 className="text-3xl md:text-4xl font-bold text-gray-900 mb-4">

Our Core Values

</h2>

<p className="text-xl text-gray-600 max-w-2xl mx-auto">

These principles guide everything we do in our mission to make communication accessible to everyone.

</p>

</div>

<div className="grid md:grid-cols-2 lg:grid-cols-4 gap-8">

<ValueCard

icon={<Heart className="h-8 w-8" />}

title="Empathy"

description="We put ourselves in our users' shoes to understand their needs and challenges."

/>

<ValueCard

icon={<Globe className="h-8 w-8" />}

title="Accessibility"

description="Making our platform accessible to everyone, regardless of their background."

/>

<ValueCard

icon={<Users className="h-8 w-8" />}

title="Community"

description="Building a supportive community that helps and learns from each other."

/>

<ValueCard

icon={<Shield className="h-8 w-8" />}

title="Privacy"

description="Protecting our users' data and privacy is our top priority."

/>

</div>

</div>

</section>

{/\* Story Section \*/}

<section className="py-24 bg-white">

<div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">

<div className="grid lg:grid-cols-2 gap-12 items-center">

<div>

<h2 className="text-3xl font-bold text-gray-900 mb-6">

Our Story

</h2>

<div className="space-y-4 text-lg text-gray-600">

<p>

Founded with a vision to bridge communication gaps, our platform leverages cutting-edge AI technology to make sign language translation accessible to everyone.

</p>

<p>

What started as a small project has grown into a global platform, helping thousands of people communicate effectively across language barriers.

</p>

<p>

Today, we continue to innovate and improve our technology, working closely with the deaf community to ensure our platform meets their needs and expectations.

</p>

</div>

</div>

<div className="relative">

<div className="aspect-w-16 aspect-h-9 rounded-xl overflow-hidden shadow-xl">

<img

src="https://images.unsplash.com/photo-1573497620053-ea5300f94f21?auto=format&fit=crop&q=80"

alt="Team collaboration"

className="object-cover w-full h-full"

/>

</div>

</div>

</div>

</div>

</section>

{/\* CTA Section \*/}

<section className="bg-gray-50 py-24">

<div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">

<div className="bg-gradient-to-r from-blue-600 to-purple-600 rounded-2xl overflow-hidden shadow-xl">

<div className="relative px-6 py-16 sm:px-12 sm:py-20">

<div className="relative max-w-3xl mx-auto text-center">

<h2 className="text-3xl font-extrabold text-white sm:text-4xl">

Be Part of Our Journey

</h2>

<p className="mt-4 text-lg text-blue-100">

Join us in making communication accessible to everyone, everywhere.

</p>

<Link

to="/register"

className="mt-8 inline-flex items-center px-8 py-3 rounded-full bg-white text-blue-600 font-semibold hover:bg-blue-50 transition-colors duration-300"

>

Get Started Now

<ArrowRight className="ml-2 h-5 w-5" />

</Link>

</div>

</div>

</div>

</div>

</section>

</div>

);

}

// File: project/src/pages/Home.tsx

import React from 'react';

import { Link } from 'react-router-dom';

import { Users, Globe2, ArrowRight } from 'lucide-react';

const FeatureCard: React.FC<{

icon: React.ReactNode;

title: string;

description: string;

}> = ({ icon, title, description }) => (

<div className="relative group bg-white p-6 rounded-xl shadow-lg hover:shadow-xl transition-all duration-300 transform hover:-translate-y-1">

<div className="absolute inset-0 bg-gradient-to-r from-blue-500/10 to-purple-500/10 rounded-xl opacity-0 group-hover:opacity-100 transition-opacity duration-300" />

<div className="relative">

<div className="text-blue-600 mb-4">{icon}</div>

<h3 className="text-xl font-semibold mb-2 text-gray-800">{title}</h3>

<p className="text-gray-600">{description}</p>

</div>

</div>

);

export default function Home() {

return (

<div className="min-h-screen">

{/\* Hero Section \*/}

<section className="relative overflow-hidden bg-gradient-to-br from-blue-600 to-purple-700 text-white">

<div className="absolute inset-0 bg-[url('https://images.unsplash.com/photo-1516733725897-1aa73b87c8e8?auto=format&fit=crop&q=80')] opacity-10 bg-cover bg-center" />

<div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-24 relative">

<div className="text-center">

<h1 className="text-4xl md:text-6xl font-bold mb-6 animate-fade-in">

Break Down Communication Barriers

</h1>

<p className="text-xl md:text-2xl mb-8 text-blue-100 max-w-3xl mx-auto">

Transform sign language into text instantly with our advanced AI-powered translation platform.

</p>

<div className="flex flex-col sm:flex-row gap-4 justify-center">

<Link

to="/register"

className="inline-flex items-center px-8 py-3 rounded-full bg-white text-blue-600 font-semibold hover:bg-blue-50 transition-colors duration-300"

>

Get Started

<ArrowRight className="ml-2 h-5 w-5" />

</Link>

<Link

to="/about"

className="inline-flex items-center px-8 py-3 rounded-full bg-blue-500 bg-opacity-20 hover:bg-opacity-30 transition-colors duration-300"

>

Learn More

</Link>

</div>

</div>

</div>

<div className="absolute bottom-0 left-0 right-0 h-16 bg-gradient-to-b from-transparent to-gray-50" />

</section>

{/\* Features Section \*/}

<section className="py-24 bg-gray-50">

<div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">

<div className="text-center mb-16">

<h2 className="text-3xl md:text-4xl font-bold text-gray-900 mb-4">

Why Choose Our Platform?

</h2>

<p className="text-xl text-gray-600 max-w-2xl mx-auto">

Experience the power of AI-driven sign language translation with features designed for accessibility and ease of use.

</p>

</div>

<div className="grid md:grid-cols-2 lg:grid-cols-3 gap-8">

<FeatureCard

icon={<Users className="h-8 w-8" />}

title="Real-Time Translation"

description="Get instant translations from sign language to text using our advanced AI technology."

/>

<FeatureCard

icon={<Users className="h-8 w-8" />}

title="Community Driven"

description="Join our growing community of users helping to improve accessibility for everyone."

/>

<FeatureCard

icon={<Globe2 className="h-8 w-8" />}

title="Global Access"

description="Access our platform from anywhere in the world, breaking down language barriers globally."

/>

</div>

</div>

</section>

{/\* CTA Section \*/}

<section className="bg-white py-24">

<div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">

<div className="bg-gradient-to-r from-blue-600 to-purple-600 rounded-2xl overflow-hidden shadow-xl">

<div className="relative px-6 py-16 sm:px-12 sm:py-20">

<div className="relative max-w-3xl mx-auto text-center">

<h2 className="text-3xl font-extrabold text-white sm:text-4xl">

Ready to Get Started?

</h2>

<p className="mt-4 text-lg text-blue-100">

Join thousands of users who are already breaking down communication barriers.

</p>

<Link

to="/register"

className="mt-8 inline-flex items-center px-8 py-3 rounded-full bg-white text-blue-600 font-semibold hover:bg-blue-50 transition-colors duration-300"

>

Sign Up Now

<ArrowRight className="ml-2 h-5 w-5" />

</Link>

</div>

</div>

</div>

</div>

</section>

</div>

);

}

// File: project/src/pages/Login.tsx

import React, { useState, useEffect } from 'react';

import { Link, useNavigate } from 'react-router-dom';

import { useDispatch } from 'react-redux';

import { Eye, EyeOff, LogIn, ArrowRight } from 'lucide-react';

import { setCredentials } from '../store/slices/authSlice';

import { authApi } from '../services/api';

export default function Login() {

const [email, setEmail] = useState('');

const [password, setPassword] = useState('');

const [showPassword, setShowPassword] = useState(false);

const [error, setError] = useState<string | null>(null);

const [isLoading, setIsLoading] = useState(false);

const dispatch = useDispatch();

const navigate = useNavigate();

useEffect(()=>{

if(localStorage.getItem('token'))

navigate('/translate')

},[])

// Login.tsx

const handleSubmit = async (e: React.FormEvent) => {

e.preventDefault();

setError(null);

setIsLoading(true);

try {

const data = await authApi.login(email, password);

dispatch(setCredentials(data)); // data now contains {user, token}

navigate('/translate');

} catch (err: any) {

setError(err.response?.data?.message || 'Failed to login');

} finally {

setIsLoading(false);

}

};

return (

<div className="min-h-screen bg-gray-50 flex flex-col justify-center py-12 sm:px-6 lg:px-8">

<div className="sm:mx-auto sm:w-full sm:max-w-md">

<h2 className="mt-6 text-center text-3xl font-extrabold text-gray-900">

Welcome back

</h2>

<p className="mt-2 text-center text-sm text-gray-600">

Don't have an account?{' '}

<Link to="/register" className="font-medium text-blue-600 hover:text-blue-500">

Sign up

</Link>

</p>

</div>

<div className="mt-8 sm:mx-auto sm:w-full sm:max-w-md">

<div className="bg-white py-8 px-4 shadow-xl rounded-lg sm:px-10">

<form className="space-y-6" onSubmit={handleSubmit}>

{error && (

<div className="bg-red-50 border-l-4 border-red-400 p-4 mb-4">

<div className="flex">

<div className="flex-shrink-0">

<svg className="h-5 w-5 text-red-400" viewBox="0 0 20 20" fill="currentColor">

<path fillRule="evenodd" d="M10 18a8 8 0 100-16 8 8 0 000 16zM8.707 7.293a1 1 0 00-1.414 1.414L8.586 10l-1.293 1.293a1 1 0 101.414 1.414L10 11.414l1.293 1.293a1 1 0 001.414-1.414L11.414 10l1.293-1.293a1 1 0 00-1.414-1.414L10 8.586 8.707 7.293z" clipRule="evenodd" />

</svg>

</div>

<div className="ml-3">

<p className="text-sm text-red-700">{error}</p>

</div>

</div>

</div>

)}

<div>

<label htmlFor="email" className="form-label">

Email address

</label>

<input

id="email"

name="email"

type="email"

autoComplete="email"

required

value={email}

onChange={(e) => setEmail(e.target.value)}

className="input-field"

/>

</div>

<div>

<label htmlFor="password" className="form-label">

Password

</label>

<div className="relative">

<input

id="password"

name="password"

type={showPassword ? 'text' : 'password'}

autoComplete="current-password"

required

value={password}

onChange={(e) => setPassword(e.target.value)}

className="input-field pr-10"

/>

<button

type="button"

className="absolute inset-y-0 right-0 pr-3 flex items-center"

onClick={() => setShowPassword(!showPassword)}

>

{showPassword ? (

<EyeOff className="h-5 w-5 text-gray-400" />

) : (

<Eye className="h-5 w-5 text-gray-400" />

)}

</button>

</div>

</div>

<div className="flex items-center justify-between">

<div className="flex items-center">

<input

id="remember-me"

name="remember-me"

type="checkbox"

className="h-4 w-4 text-blue-600 focus:ring-blue-500 border-gray-300 rounded"

/>

<label htmlFor="remember-me" className="ml-2 block text-sm text-gray-900">

Remember me

</label>

</div>

</div>

<div>

<button

type="submit"

disabled={isLoading}

className="w-full flex justify-center py-3 px-4 border border-transparent rounded-lg shadow-sm text-sm font-medium text-white bg-blue-600 hover:bg-blue-700 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-blue-500 disabled:opacity-50 disabled:cursor-not-allowed"

>

{isLoading ? (

<svg className="animate-spin h-5 w-5 text-white" xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24">

<circle className="opacity-25" cx="12" cy="12" r="10" stroke="currentColor" strokeWidth="4"></circle>

<path className="opacity-75" fill="currentColor" d="M4 12a8 8 0 018-8V0C5.373 0 0 5.373 0 12h4zm2 5.291A7.962 7.962 0 014 12H0c0 3.042 1.135 5.824 3 7.938l3-2.647z"></path>

</svg>

) : (

<>

<LogIn className="w-5 h-5 mr-2" />

Sign in

</>

)}

</button>

</div>

</form>

<div className="mt-6">

<div className="relative">

<div className="absolute inset-0 flex items-center">

<div className="w-full border-t border-gray-300"></div>

</div>

<div className="relative flex justify-center text-sm">

<span className="px-2 bg-white text-gray-500">New to our platform?</span>

</div>

</div>

<div className="mt-6">

<Link

to="/register"

className="w-full flex justify-center items-center px-4 py-3 border border-transparent rounded-lg shadow-sm text-sm font-medium text-blue-600 bg-blue-50 hover:bg-blue-100"

>

Create an account

<ArrowRight className="ml-2 h-5 w-5" />

</Link>

</div>

</div>

</div>

</div>

</div>

);

}

// File: project/src/pages/Profile.tsx

import React, { useState } from 'react';

import { useDispatch, useSelector } from 'react-redux';

import { RootState } from '../store';

import { setCredentials } from '../store/slices/authSlice';

import { userApi } from '../services/api';

import { Eye, EyeOff, Save, User } from 'lucide-react';

export default function Profile() {

const { user } = useSelector((state: RootState) => state.auth);

const dispatch = useDispatch();

const [formData, setFormData] = useState({

fullName: user?.fullName || '',

email: user?.email || '',

currentPassword: '',

newPassword: '',

confirmPassword: '',

});

const [showCurrentPassword, setShowCurrentPassword] = useState(false);

const [showNewPassword, setShowNewPassword] = useState(false);

const [showConfirmPassword, setShowConfirmPassword] = useState(false);

const [isLoading, setIsLoading] = useState(false);

const [error, setError] = useState<string | null>(null);

const [success, setSuccess] = useState<string | null>(null);

const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {

const { name, value } = e.target;

setFormData(prev => ({ ...prev, [name]: value }));

};

const handleSubmit = async (e: React.FormEvent) => {

e.preventDefault();

setError(null);

setSuccess(null);

setIsLoading(true);

try {

if (formData.newPassword && formData.newPassword !== formData.confirmPassword) {

throw new Error('New passwords do not match');

}

const updatedData = await userApi.updateProfile({

fullName: formData.fullName,

email: formData.email

});

if (formData.newPassword && formData.currentPassword) {

await userApi.updatePassword({

currentPassword: formData.currentPassword,

newPassword: formData.newPassword

})

}

dispatch(setCredentials({ user: updatedData, token: localStorage.getItem('token') || '' }));

setSuccess('Profile updated successfully');

setFormData(prev => ({

...prev,

currentPassword: '',

newPassword: '',

confirmPassword: '',

}));

} catch (err: any) {

setError(err.message || 'Failed to update profile. Please try again.');

} finally {

setIsLoading(false);

}

};

return (

<div className="min-h-screen bg-gray-50 py-12">

<div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">

<div className="max-w-3xl mx-auto">

<div className="bg-white shadow-xl rounded-lg overflow-hidden">

<div className="px-4 py-5 sm:p-6">

<div className="flex items-center space-x-3 mb-6">

<div className="h-12 w-12 rounded-full bg-blue-100 flex items-center justify-center">

<User className="h-6 w-6 text-blue-600" />

</div>

<div>

<h2 className="text-2xl font-bold text-gray-900">Profile Settings</h2>

<p className="text-sm text-gray-500">Update your account information</p>

</div>

</div>

<form onSubmit={handleSubmit} className="space-y-6">

{error && (

<div className="bg-red-50 border-l-4 border-red-400 p-4">

<div className="flex">

<div className="flex-shrink-0">

<svg className="h-5 w-5 text-red-400" viewBox="0 0 20 20" fill="currentColor">

<path fillRule="evenodd" d="M10 18a8 8 0 100-16 8 8 0 000 16zM8.707 7.293a1 1 0 00-1.414 1.414L8.586 10l-1.293 1.293a1 1 0 101.414 1.414L10 11.414l1.293 1.293a1 1 0 001.414-1.414L11.414 10l1.293-1.293a1 1 0 00-1.414-1.414L10 8.586 8.707 7.293z" clipRule="evenodd" />

</svg>

</div>

<div className="ml-3">

<p className="text-sm text-red-700">{error}</p>

</div>

</div>

</div>

)}

{success && (

<div className="bg-green-50 border-l-4 border-green-400 p-4">

<div className="flex">

<div className="flex-shrink-0">

<svg className="h-5 w-5 text-green-400" viewBox="0 0 20 20" fill="currentColor">

<path fillRule="evenodd" d="M10 18a8 8 0 100-16 8 8 0 000 16zm3.707-9.293a1 1 0 00-1.414-1.414L9 10.586 7.707 9.293a1 1 0 00-1.414 1.414l2 2a1 1 0 001.414 0l4-4z" clipRule="evenodd" />

</svg>

</div>

<div className="ml-3">

<p className="text-sm text-green-700">{success}</p>

</div>

</div>

</div>

)}

<div className="grid grid-cols-1 gap-6">

<div>

<label htmlFor="fullName" className="form-label">

Full name

</label>

<input

type="text"

name="fullName"

id="fullName"

value={formData.fullName}

onChange={handleChange}

className="input-field"

/>

</div>

<div>

<label htmlFor="email" className="form-label">

Email address

</label>

<input

type="email"

name="email"

id="email"

value={formData.email}

disabled

className="input-field bg-gray-50 cursor-not-allowed"

/>

</div>

<div className="border-t border-gray-200 pt-6">

<h3 className="text-lg font-medium text-gray-900 mb-4">Change Password</h3>

<div className="space-y-4">

<div>

<label htmlFor="currentPassword" className="form-label">

Current password

</label>

<div className="relative">

<input

type={showCurrentPassword ? 'text' : 'password'}

name="currentPassword"

id="currentPassword"

value={formData.currentPassword}

onChange={handleChange}

className="input-field pr-10"

/>

<button

type="button"

className="absolute inset-y-0 right-0 pr-3 flex items-center"

onClick={() => setShowCurrentPassword(!showCurrentPassword)}

>

{showCurrentPassword ? (

<EyeOff className="h-5 w-5 text-gray-400" />

) : (

<Eye className="h-5 w-5 text-gray-400" />

)}

</button>

</div>

</div>

<div>

<label htmlFor="newPassword" className="form-label">

New password

</label>

<div className="relative">

<input

type={showNewPassword ? 'text' : 'password'}

name="newPassword"

id="newPassword"

value={formData.newPassword}

onChange={handleChange}

className="input-field pr-10"

/>

<button

type="button"

className="absolute inset-y-0 right-0 pr-3 flex items-center"

onClick={() => setShowNewPassword(!showNewPassword)}

>

{showNewPassword ? (

<EyeOff className="h-5 w-5 text-gray-400" />

) : (

<Eye className="h-5 w-5 text-gray-400" />

)}

</button>

</div>

</div>

<div>

<label htmlFor="confirmPassword" className="form-label">

Confirm new password

</label>

<div className="relative">

<input

type={showConfirmPassword ? 'text' : 'password'}

name="confirmPassword"

id="confirmPassword"

value={formData.confirmPassword}

onChange={handleChange}

className="input-field pr-10"

/>

<button

type="button"

className="absolute inset-y-0 right-0 pr-3 flex items-center"

onClick={() => setShowConfirmPassword(!showConfirmPassword)}

>

{showConfirmPassword ? (

<EyeOff className="h-5 w-5 text-gray-400" />

) : (

<Eye className="h-5 w-5 text-gray-400" />

)}

</button>

</div>

</div>

</div>

</div>

</div>

<div className="flex justify-end">

<button

type="submit"

disabled={isLoading}

className="inline-flex items-center px-4 py-2 border border-transparent rounded-md shadow-sm text-sm font-medium text-white bg-blue-600 hover:bg-blue-700 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-blue-500 disabled:opacity-50 disabled:cursor-not-allowed"

>

{isLoading ? (

<svg className="animate-spin h-5 w-5 text-white" xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24">

<circle className="opacity-25" cx="12" cy="12" r="10" stroke="currentColor" strokeWidth="4"></circle>

<path className="opacity-75" fill="currentColor" d="M4 12a8 8 0 018-8V0C5.373 0 0 5.373 0 12h4zm2 5.291A7.962 7.962 0 014 12H0c0 3.042 1.135 5.824 3 7.938l3-2.647z"></path>

</svg>

) : (

<>

<Save className="w-5 h-5 mr-2" />

Save changes

</>

)}

</button>

</div>

</form>

</div>

</div>

</div>

</div>

</div>

);

}

// File: project/src/pages/Register.tsx

import React, { useEffect, useState } from 'react';

import { Link, useNavigate } from 'react-router-dom';

import { useDispatch } from 'react-redux';

import { Eye, EyeOff, UserPlus } from 'lucide-react';

import { setCredentials } from '../store/slices/authSlice';

import { authApi } from '../services/api';

export default function Register() {

const [formData, setFormData] = useState({

fullName: '',

email: '',

password: '',

confirmPassword: '',

});

const [showPassword, setShowPassword] = useState(false);

const [showConfirmPassword, setShowConfirmPassword] = useState(false);

const [error, setError] = useState<string | null>(null);

const [isLoading, setIsLoading] = useState(false);

const dispatch = useDispatch();

const navigate = useNavigate();

const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {

const { name, value } = e.target;

setFormData(prev => ({ ...prev, [name]: value }));

};

useEffect(()=>{

if(localStorage.getItem('token'))

navigate('/translate')

},[])

// Register.tsx

const handleSubmit = async (e: React.FormEvent) => {

e.preventDefault();

setError(null);

if (formData.password !== formData.confirmPassword) {

setError('Passwords do not match');

return;

}

setIsLoading(true);

try {

const data = await authApi.register(

formData.fullName,

formData.email,

formData.password

);

dispatch(setCredentials(data)); // data now contains {user, token}

navigate('/translate');

} catch (err: any) {

setError(err.response?.data?.message || 'Failed to register');

} finally {

setIsLoading(false);

}

};

return (

<div className="min-h-screen bg-gray-50 flex flex-col justify-center py-12 sm:px-6 lg:px-8">

<div className="sm:mx-auto sm:w-full sm:max-w-md">

<h2 className="mt-6 text-center text-3xl font-extrabold text-gray-900">

Create your account

</h2>

<p className="mt-2 text-center text-sm text-gray-600">

Already have an account?{' '}

<Link to="/login" className="font-medium text-blue-600 hover:text-blue-500">

Sign in

</Link>

</p>

</div>

<div className="mt-8 sm:mx-auto sm:w-full sm:max-w-md">

<div className="bg-white py-8 px-4 shadow-xl rounded-lg sm:px-10">

<form className="space-y-6" onSubmit={handleSubmit}>

{error && (

<div className="bg-red-50 border-l-4 border-red-400 p-4">

<div className="flex">

<div className="flex-shrink-0">

<svg className="h-5 w-5 text-red-400" viewBox="0 0 20 20" fill="currentColor">

<path fillRule="evenodd" d="M10 18a8 8 0 100-16 8 8 0 000 16zM8.707 7.293a1 1 0 00-1.414 1.414L8.586 10l-1.293 1.293a1 1 0 101.414 1.414L10 11.414l1.293 1.293a1 1 0 001.414-1.414L11.414 10l1.293-1.293a1 1 0 00-1.414-1.414L10 8.586 8.707 7.293z" clipRule="evenodd" />

</svg>

</div>

<div className="ml-3">

<p className="text-sm text-red-700">{error}</p>

</div>

</div>

</div>

)}

<div>

<label htmlFor="fullName" className="form-label">

Full name

</label>

<input

id="fullName"

name="fullName"

type="text"

autoComplete="name"

required

value={formData.fullName}

onChange={handleChange}

className="input-field"

/>

</div>

<div>

<label htmlFor="email" className="form-label">

Email address

</label>

<input

id="email"

name="email"

type="email"

autoComplete="email"

required

value={formData.email}

onChange={handleChange}

className="input-field"

/>

</div>

<div>

<label htmlFor="password" className="form-label">

Password

</label>

<div className="relative">

<input

id="password"

name="password"

type={showPassword ? 'text' : 'password'}

autoComplete="new-password"

required

value={formData.password}

onChange={handleChange}

className="input-field pr-10"

/>

<button

type="button"

className="absolute inset-y-0 right-0 pr-3 flex items-center"

onClick={() => setShowPassword(!showPassword)}

>

{showPassword ? (

<EyeOff className="h-5 w-5 text-gray-400" />

) : (

<Eye className="h-5 w-5 text-gray-400" />

)}

</button>

</div>

</div>

<div>

<label htmlFor="confirmPassword" className="form-label">

Confirm password

</label>

<div className="relative">

<input

id="confirmPassword"

name="confirmPassword"

type={showConfirmPassword ? 'text' : 'password'}

autoComplete="new-password"

required

value={formData.confirmPassword}

onChange={handleChange}

className="input-field pr-10"

/>

<button

type="button"

className="absolute inset-y-0 right-0 pr-3 flex items-center"

onClick={() => setShowConfirmPassword(!showConfirmPassword)}

>

{showConfirmPassword ? (

<EyeOff className="h-5 w-5 text-gray-400" />

) : (

<Eye className="h-5 w-5 text-gray-400" />

)}

</button>

</div>

</div>

<div>

<button

type="submit"

disabled={isLoading}

className="w-full flex justify-center py-3 px-4 border border-transparent rounded-lg shadow-sm text-sm font-medium text-white bg-blue-600 hover:bg-blue-700 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-blue-500 disabled:opacity-50 disabled:cursor-not-allowed"

>

{isLoading ? (

<svg className="animate-spin h-5 w-5 text-white" xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24">

<circle className="opacity-25" cx="12" cy="12" r="10" stroke="currentColor" strokeWidth="4"></circle>

<path className="opacity-75" fill="currentColor" d="M4 12a8 8 0 018-8V0C5.373 0 0 5.373 0 12h4zm2 5.291A7.962 7.962 0 014 12H0c0 3.042 1.135 5.824 3 7.938l3-2.647z"></path>

</svg>

) : (

<>

<UserPlus className="w-5 h-5 mr-2" />

Create account

</>

)}

</button>

</div>

</form>

<div className="mt-6">

<div className="relative">

<div className="absolute inset-0 flex items-center">

<div className="w-full border-t border-gray-300"></div>

</div>

<div className="relative flex justify-center text-sm">

<span className="px-2 bg-white text-gray-500">

By signing up, you agree to our Terms and Privacy Policy

</span>

</div>

</div>

</div>

</div>

</div>

</div>

);

}

// File: project/src/pages/Translate.tsx

import React from 'react';

import DetectJSX from '../components/Detect';

import { useNavigate } from 'react-router-dom';

export default function Translate() {

const navigate = useNavigate();

React.useEffect(()=>{

if(!localStorage.getItem('token')){

navigate('/login')

}

}, [navigate]);

return (

<div className="min-h-screen bg-gray-50 py-12">

<div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">

<div className="max-w-4xl mx-auto">

<div className="bg-white shadow-xl rounded-lg overflow-hidden">

<DetectJSX/>

</div>

</div>

</div>

</div>

);

}

// File: project/src/services/api.ts

import axios from 'axios';

const API\_URL = 'http://localhost:5555/api';

const api = axios.create({

baseURL: API\_URL,

headers: {

'Content-Type': 'application/json',

},

});

// Add token to requests if it exists

api.interceptors.request.use((config) => {

const token = localStorage.getItem('token');

if (token) {

config.headers.Authorization = `Bearer ${token}`;

}

return config;

});

// api.ts - update return types

export const authApi = {

login: async (email: string, password: string): Promise<{ user: User; token: string }> => {

const response = await api.post('/auth/login', { email, password });

return response.data;

},

register: async (fullName: string, email: string, password: string): Promise<{ user: User; token: string }> => {

const response = await api.post('/auth/register', { fullName, email, password });

return response.data;

},

forgotPassword: async (email: string) => {

const response = await api.post('/auth/forgot-password', { email });

return response.data;

},

resetPassword: async (token: string, password: string) => {

const response = await api.post('/auth/reset-password', { token, password });

return response.data;

},

};

export const userApi = {

getProfile: async () => {

const response = await api.get('/user');

return response.data;

},

updateProfile: async (userData: any) => {

const response = await api.put('/user', userData);

return response.data;

},

updatePassword: async (passwordData: any) => {

const response = await api.put('/user/password', passwordData);

return response.data;

},

};

// File: project/src/store/index.ts

import { configureStore } from '@reduxjs/toolkit';

import authReducer from './slices/authSlice';

import translationReducer from './slices/translationSlice';

export const store = configureStore({

reducer: {

auth: authReducer,

translation: translationReducer,

},

});

export type RootState = ReturnType<typeof store.getState>;

export type AppDispatch = typeof store.dispatch;

// File: project/src/store/slices/authSlice.ts

import { createSlice, PayloadAction } from '@reduxjs/toolkit';

import { AuthState, User } from '../../types';

const initialState: AuthState = {

user: null,

token: localStorage.getItem('token'),

isLoading: false,

error: null,

};

// authSlice.ts - make reducer match API response

const authSlice = createSlice({

name: 'auth',

initialState,

reducers: {

setCredentials: (

state,

action: PayloadAction<{ user: User; token: string }>) => {

state.user = action.payload.user;

state.token = action.payload.token;

// console.log(action.payload.token)

localStorage.setItem('token', action.payload.token);

},

logout: (state) => {

state.user = null;

state.token = null;

localStorage.removeItem('token');

},

},

});

export const { setCredentials, logout } = authSlice.actions;

export default authSlice.reducer;

// File: project/src/store/slices/translationSlice.ts

import { createSlice } from '@reduxjs/toolkit';

const initialState = {

error: null,

};

const translationSlice = createSlice({

name: 'translation',

initialState,

reducers: {

setError: (state, action) => {

state.error = action.payload;

},

clearError: (state) => {

state.error = null;

},

},

});

export const { setError, clearError } = translationSlice.actions;

export default translationSlice.reducer;

// File: project/src/types/index.ts

export interface User {

id: number;

email: string;

fullName: string;

}

export interface AuthState {

user: User | null;

token: string | null;

isLoading: boolean;

error: string | null;

}

export interface LoginCredentials {

email: string;

password: string;

}

export interface RegisterCredentials extends LoginCredentials {

fullName: string;

confirmPassword: string;

}

// File: project/src/vite-env.d.ts

/// <reference types="vite/client" />

// File: project/tailwind.config.js

/\*\* @type {import('tailwindcss').Config} \*/

export default {

content: ['./index.html', './src/\*\*/\*.{js,ts,jsx,tsx}'],

theme: {

extend: {},

},

plugins: [],

};

// File: project/tsconfig.app.json

{

"compilerOptions": {

"target": "ES2020",

"useDefineForClassFields": true,

"lib": ["ES2020", "DOM", "DOM.Iterable"],

"module": "ESNext",

"skipLibCheck": true,

/\* Bundler mode \*/

"moduleResolution": "bundler",

"allowImportingTsExtensions": true,

"isolatedModules": true,

"moduleDetection": "force",

"noEmit": true,

"jsx": "react-jsx",

/\* Linting \*/

"strict": true,

"noUnusedLocals": true,

"noUnusedParameters": true,

"noFallthroughCasesInSwitch": true

},

"include": ["src"]

}

// File: project/tsconfig.json

{

"files": [],

"references": [

{ "path": "./tsconfig.app.json" },

{ "path": "./tsconfig.node.json" }

]

}

// File: project/tsconfig.node.json

{

"compilerOptions": {

"target": "ES2022",

"lib": ["ES2023"],

"module": "ESNext",

"skipLibCheck": true,

/\* Bundler mode \*/

"moduleResolution": "bundler",

"allowImportingTsExtensions": true,

"isolatedModules": true,

"moduleDetection": "force",

"noEmit": true,

/\* Linting \*/

"strict": true,

"noUnusedLocals": true,

"noUnusedParameters": true,

"noFallthroughCasesInSwitch": true

},

"include": ["vite.config.ts"]

}