*\*needs to be typed out in latex or wiki (according to the study guide)*

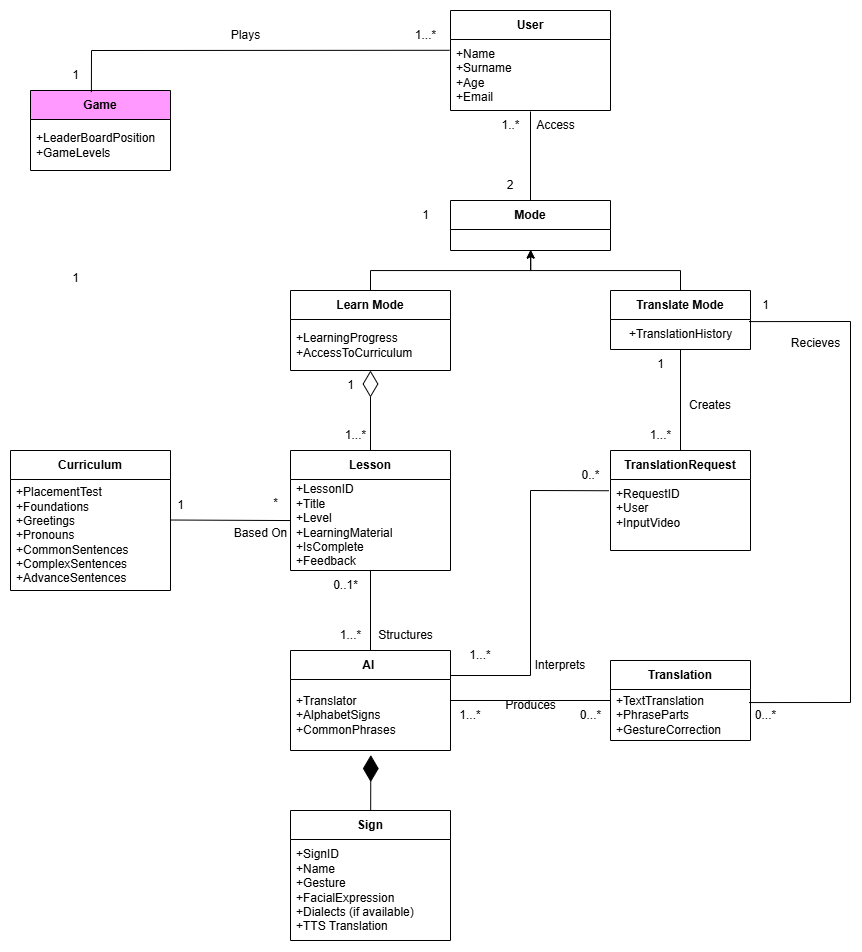
# Introduction

*State the business need for the application and summarize the project's scope.*

Imagine you are in a busy shopping centre and someone tries to get your attention — not with words, but through a series of hand gestures. You watch, trying to decipher the movements, but remain unsure. Gradually, you recognize it as sign language and find yourself wishing you understood, even just a little. You wish there were a way to translate their gestures. As TMKDT, we aim to turn those wishes into reality with Hands UP. A powerful application to help you — one sign at a time!

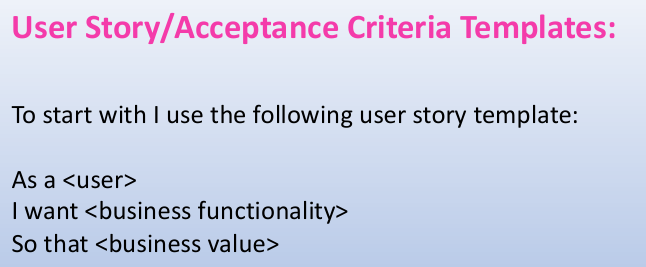
Hands UP is an innovative application that bridges the communication gap between signers and non-signers. Using advanced AI technology, the application detects and translates sign language in real-time through the device's camera, converting signs into both text and spoken language without significant delays. Beyond translation, it also serves as an interactive learning platform with structured lessons and immediate feedback on signing accuracy.

# Domain Model

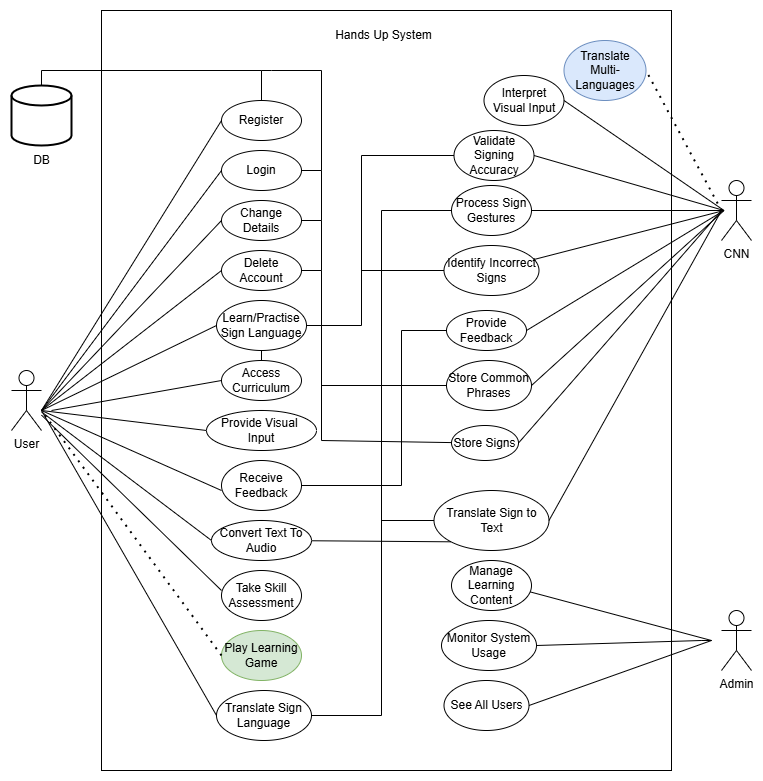


# User Stories / Characteristics

*List all intended users and explain the system's usage for each.*

**

# Use Case Diagram



# Functional Requirements

*Specify the functional requirements to satisfy the use cases. Assign the requirements to sub systems.*FR1. Application must be able to translate sign language

FR 1.1. Users must be able to provide visual input

FR 1.2. Application must provide text output

FR 1.3. Application must provide audio output

FR2. Users should see their learning progress and be able to track it  
 FR 2.1. Application should provide comprehensive progress analysis

FR 2.2. Progress should be presented in a simple (probably graphical manner)

FR3. Application must support real-time feedback and correction

  FR 3.1. Users must receive immediate feedback on incorrect signs

  FR 3.2. Application must suggest correct hand gestures or movements

FR4. Application must allow user profile creation and customization

  FR 4.1. Users must be able to create and manage personal account

  FR 4.2. Users must be able to set learning goals and preferences

# Architectural Requirements

## Quality Requirements

|  |  |
| --- | --- |
| Quality Attribute | Description |
| Usability | The system will feature an intuitive user interface designed for users of varying tech literacy. Accessible navigation and real-time visual feedback will ensure a smooth learning and translation experience. |
| Performance | Real-time translation of sign language must occur with minimal latency (<200ms response time). The system should maintain responsiveness under concurrent user loads. |
| Scalability | The architecture must support horizontal scaling to accommodate increasing numbers of users and data (especially for cloud-based ML inference). |
| Availability | System uptime must be ≥ 99%, especially for real-time features. Cloud hosting and CDN caching will support this. |
| Security | Secure user authentication (e.g., Firebase Auth), HTTPS communication, and data privacy compliance (e.g., POPIA/GDPR) will be implemented. |
| Maintainability | Codebase will be modular with proper documentation and CI/CD integration (e.g., GitHub Actions, Docker) for seamless updates and bug fixes. |
| Offline Support | As a PWA, the system will provide limited offline functionality (e.g., cached lessons, recent translations) using service workers. |

## Architectural Patterns

**Client-Server Architecture**:  
 The core of the application will follow a traditional client-server model, where the frontend (React/Flutter Web) interacts with backend APIs (Node.js or FastAPI).

**Microservices Architecture (Optional Future Enhancement)**:  
 For scaling specific components (e.g., translation service, learning analytics), consider migrating to microservices. Useful if sign language models are containerized and deployed separately.

**Event-Driven Architecture (for real-time feedback)**:  
 WebSockets or similar technologies may be used to push real-time detection and feedback from the backend to the client.

**MVC (Model-View-Controller):**

Separates the UI (View), data handling (Model), and app logic (Controller) for clear organization.

## Design Patterns

|  |  |
| --- | --- |
| Pattern | Purpose |
| Observer Pattern | Used to trigger UI updates in real-time when sign language is recognized (e.g., translating input frames to words). |
| Strategy Pattern | Enables switching between different sign language models or learning modules dynamically. |
| Singleton Pattern | For managing global app state like authenticated user, current lesson progress, or AI model instances. |
| Factory Pattern | Instantiate different AI model types (learning, translation, gamification) |

NB Strategy is optional

## Constraints

**Web-Based Delivery:**

The application must be accessible through modern web browsers. This limits hardware-level access but ensures platform independence.

**Progressive Web App (PWA):**

The system must support PWA features such as:

* Add to home screen
* Offline support using service workers
* Background sync and push notifications for learning reminders

**Budget Limit (R5000):**

All technologies must be open-source or free-tier friendly. Preference for Google Cloud, Firebase, and GitHub for deployment and services.

**Dataset Limitations**

For the initial phase of the project, we will use publicly available datasets such as the Sign Language MNIST and ASL alphabet datasets available on Kaggle. These datasets cover basic static hand gestures for American Sign Language (ASL).

# Service Contracts

**Service contracts** define the **interface** and **expected behavior** of services in a software system—especially in systems following **Service-Oriented Architecture (SOA)** or **microservices**.

**In simpler terms:**

A **service contract** is an agreement that specifies:

* What a service **does**
* How clients should **interact** with it
* What **inputs** it accepts
* What **outputs** or **responses** it returns
* Any **preconditions**, **postconditions**, and **error conditions**

**Example**

Service Name: UserService

Operation: getUserDetails(userId)

Inputs: userId (String, required)

Outputs: User object (id, name, email)

Preconditions: userId must exist in the system

Postconditions: returns the correct user details

Errors: 404 if userId not found

# Technology Requirements