Introduction

Sign Sync is a real-time translation system designed to bridge communication between spoken English and American Sign Language (ASL). The system allows users to input spoken or typed English and receive an accurate visual translation in ASL, using a combination of natural language processing, speech recognition, and gesture playback. It is developed as part of the COS301 Capstone Project to assist the Deaf and Hard-of-Hearing community.

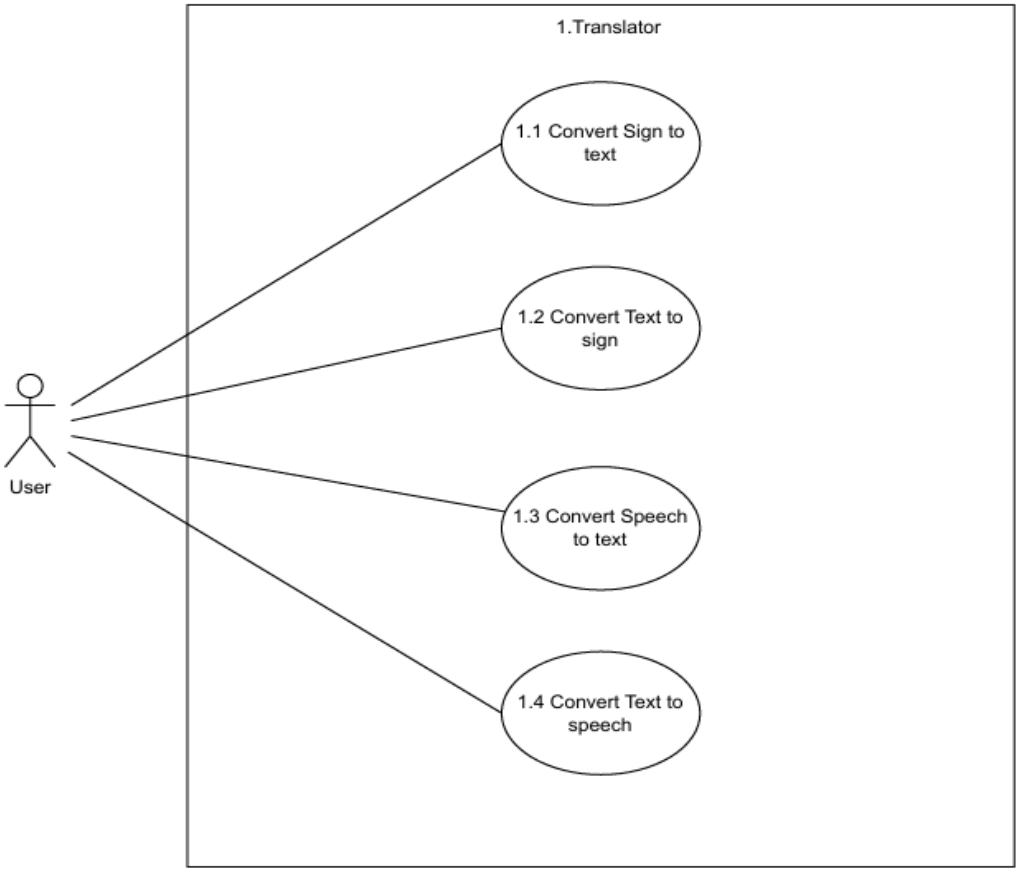
User Stories

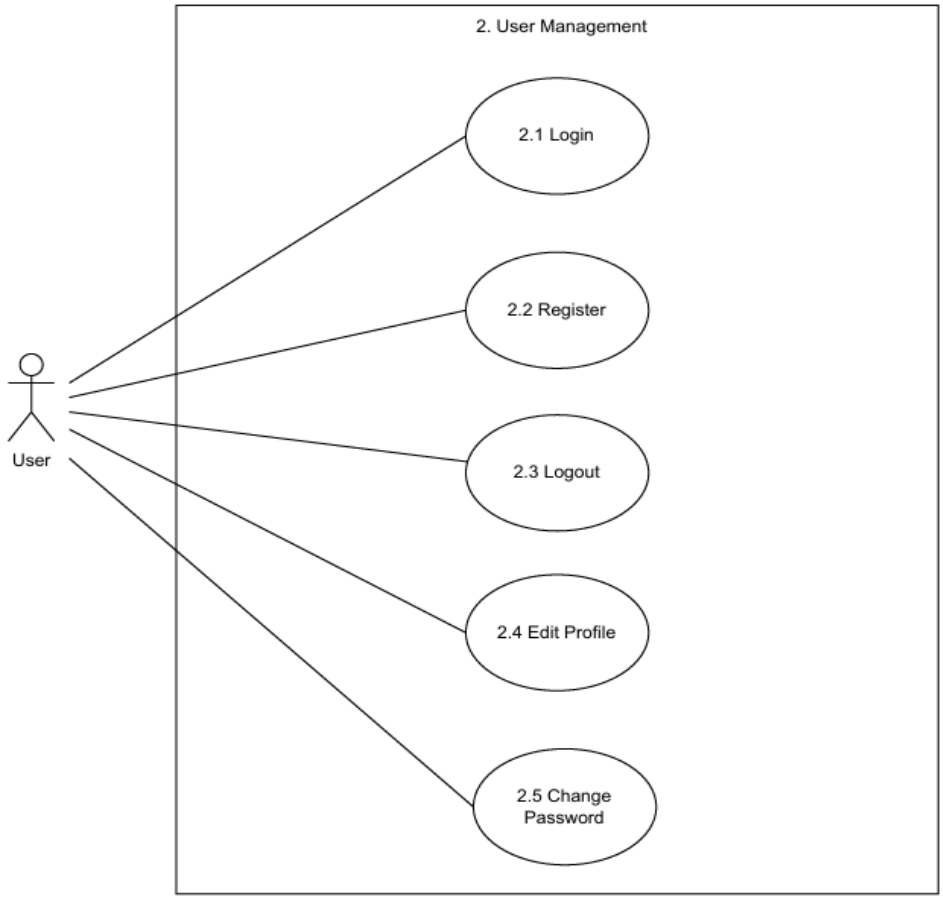
1. **Deaf or Hard-of-Hearing Users:**
    - View spoken language translated into sign animations.
    - Use sign language to communicate back via webcam input.

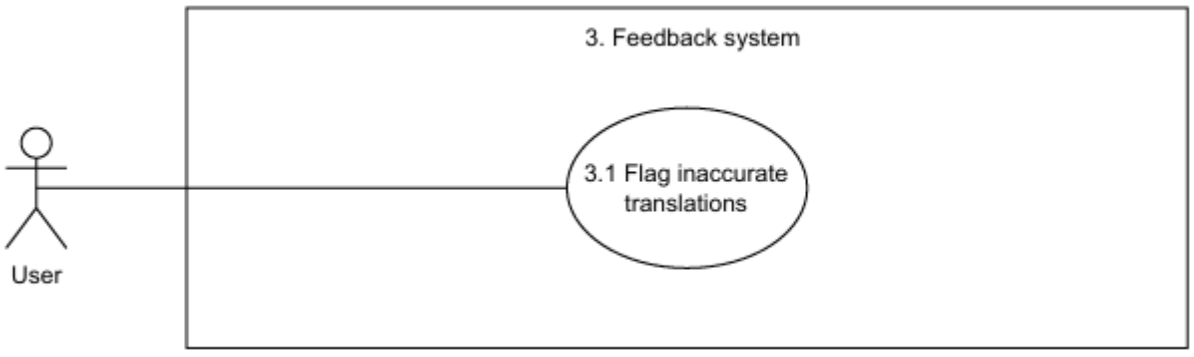
2. **Hearing Users:**
    - Speak naturally and have their speech translated into signs.
    - Read or hear signed responses translated to text or audio.

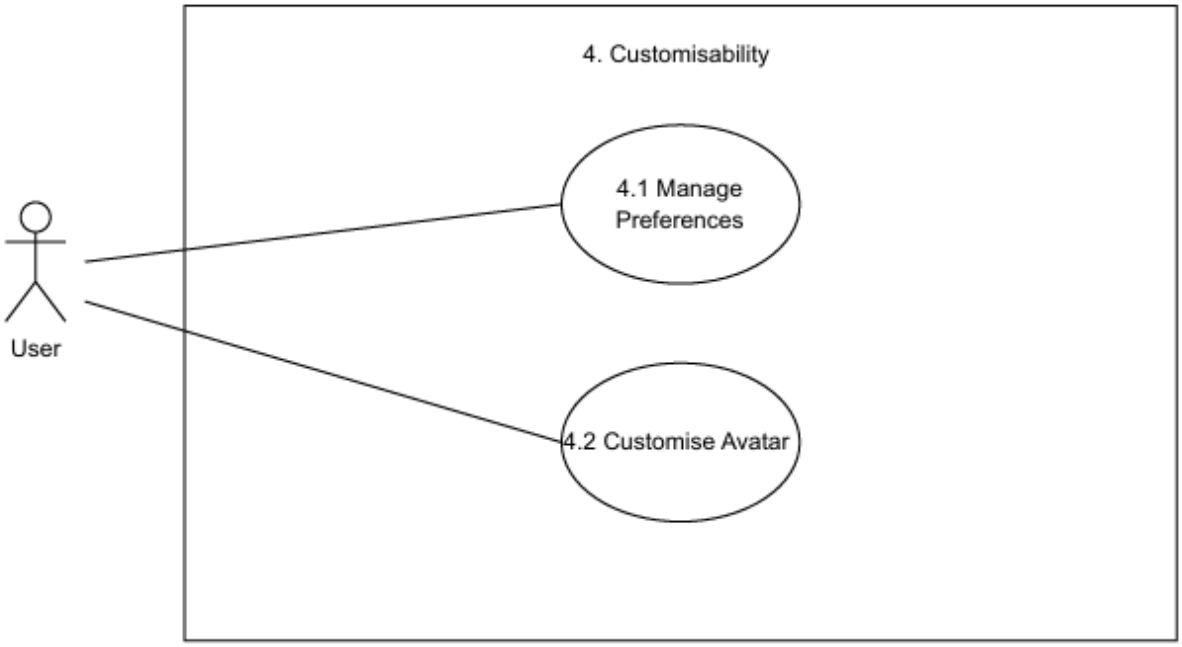
3. **Administrators/Researchers**
    - Monitor system performance.
    - Gather feedback data for AI retraining.
    - Manage user access and customization settings.

Use Case Diagrams




2. User Management
2.1 Login
2.2 Register
2.3 Logout
2.4 Edit Profile
2.5 Change Password
User


3. Feedback system
3.1 Flag inaccurate translations
User


4. Customisability
4.1 Manage Preferences
User
4.2 Customise Avatar

Functional Requirements

R1: text-to-Sign Translator
- R1.1: Capture text input from user.
- R1.2: Translate English text to Sign gloss.
- R1.3: Search word definition for appropriate sign.
- R1.4: Display sign through avatar.

R2: Sign-to-Text Translator
- R2.1: Capture webcam input and extract hand keypoints
- R2.2: Classify sign gesture sequences using trained AI model
- R2.3: Convert recognized signs to sign gloss
- R2.4: Convert sign gloss to English

R3: Feedback and AI Improvement System
- R3.1: Allow users to flag inaccurate translations
- R3.2: Log flagged data
- R3.3: Retrain AI models periodically using collected data

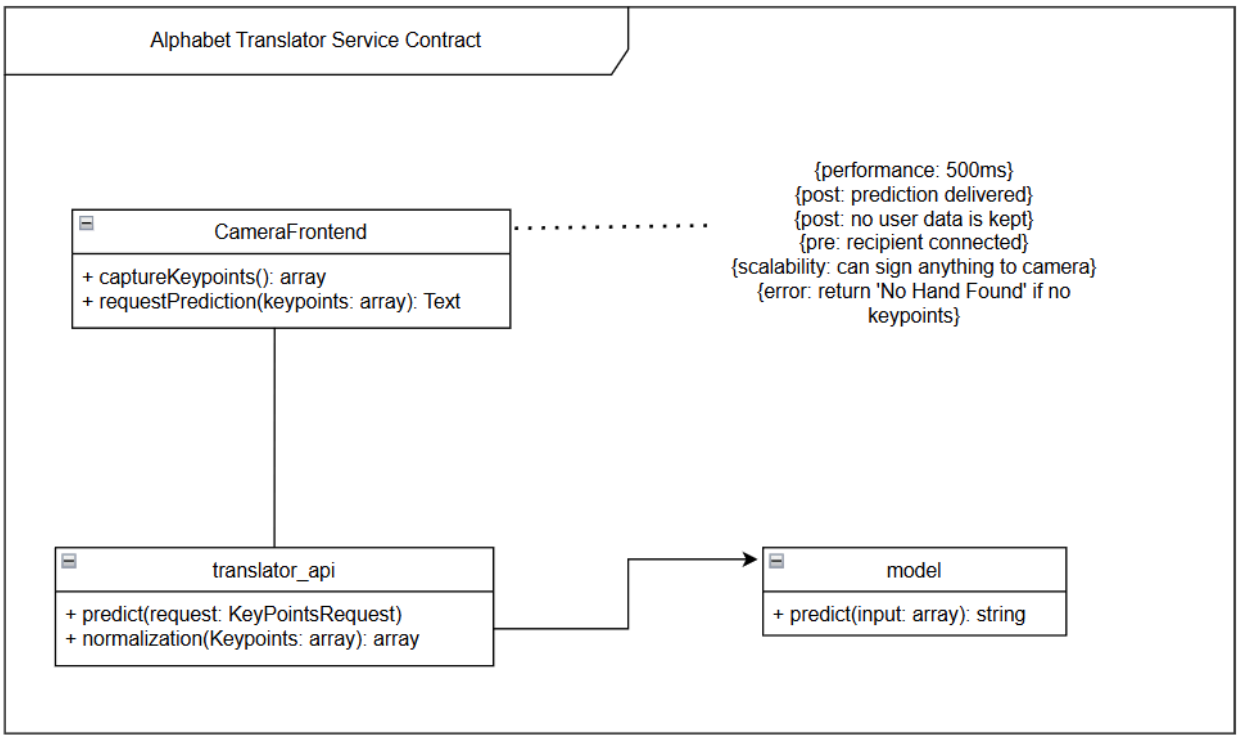
R4: User Interface
- R4.1: Display live avatar animations based on translation output
- R4.2: Show translated text and/or play voice feedback
- R4.3: Offer accessibility options (high-contrast mode, font scaling, voice personas)

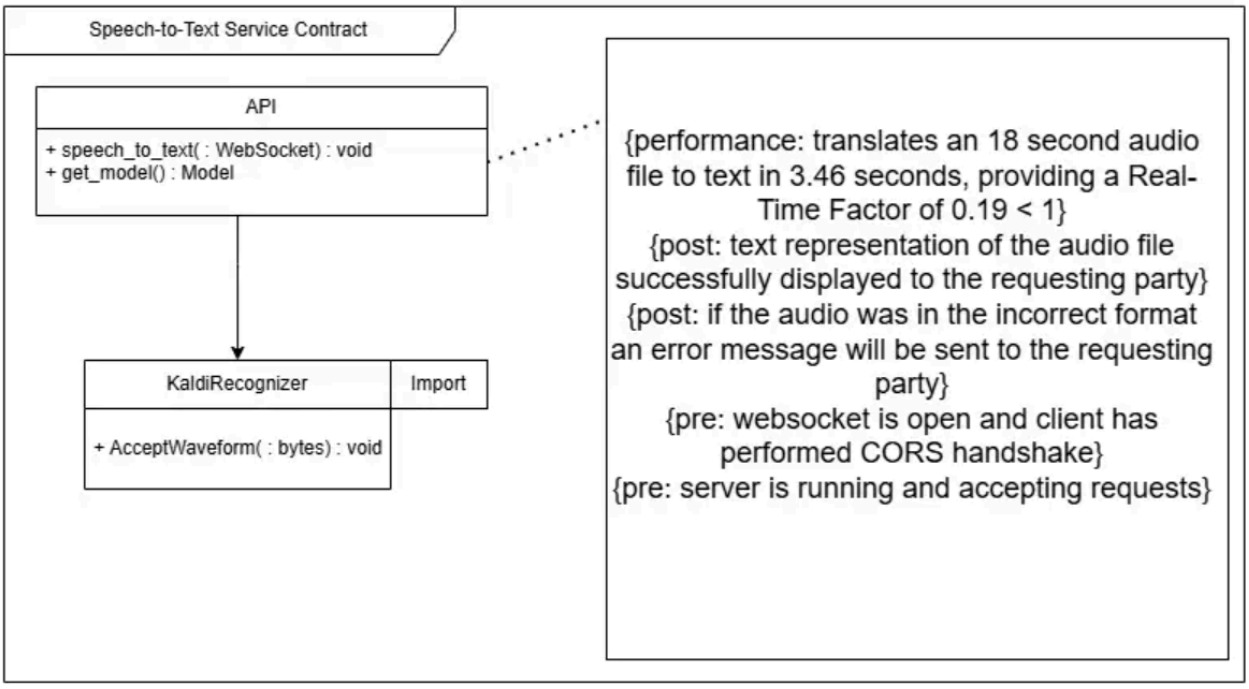
R5: User Management and Settings
- R5.1: Authenticate users
- R5.2: Store and retrieve user preferences and settings
- R5.3: Allow users to login
- R5.4: Allow users to register

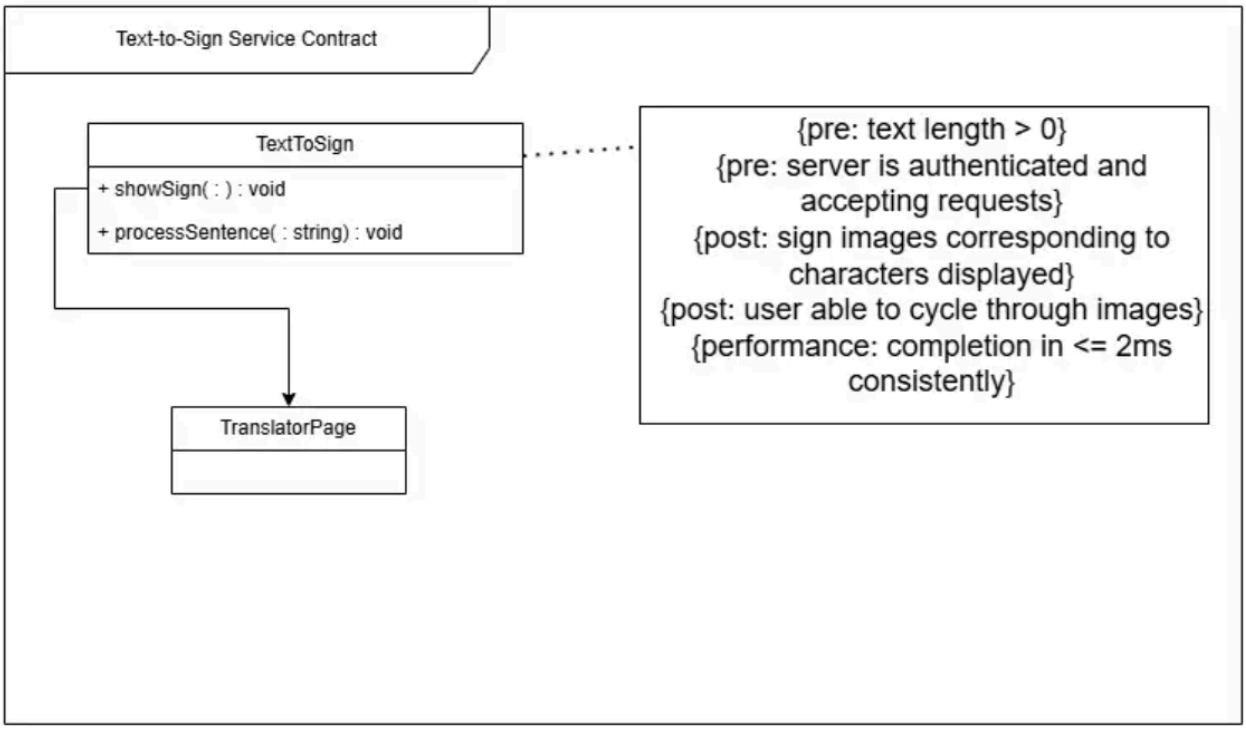
R6: Speech to Text
- R6.1: Capture user speech
- R6.2: Convert speech to text
- R6.3: Display text on screen

R7: Text to Speech
- R7.1: Capture text input by user
- R7.2: Convert text to Speech
- R7.3: Play speech for user to hear

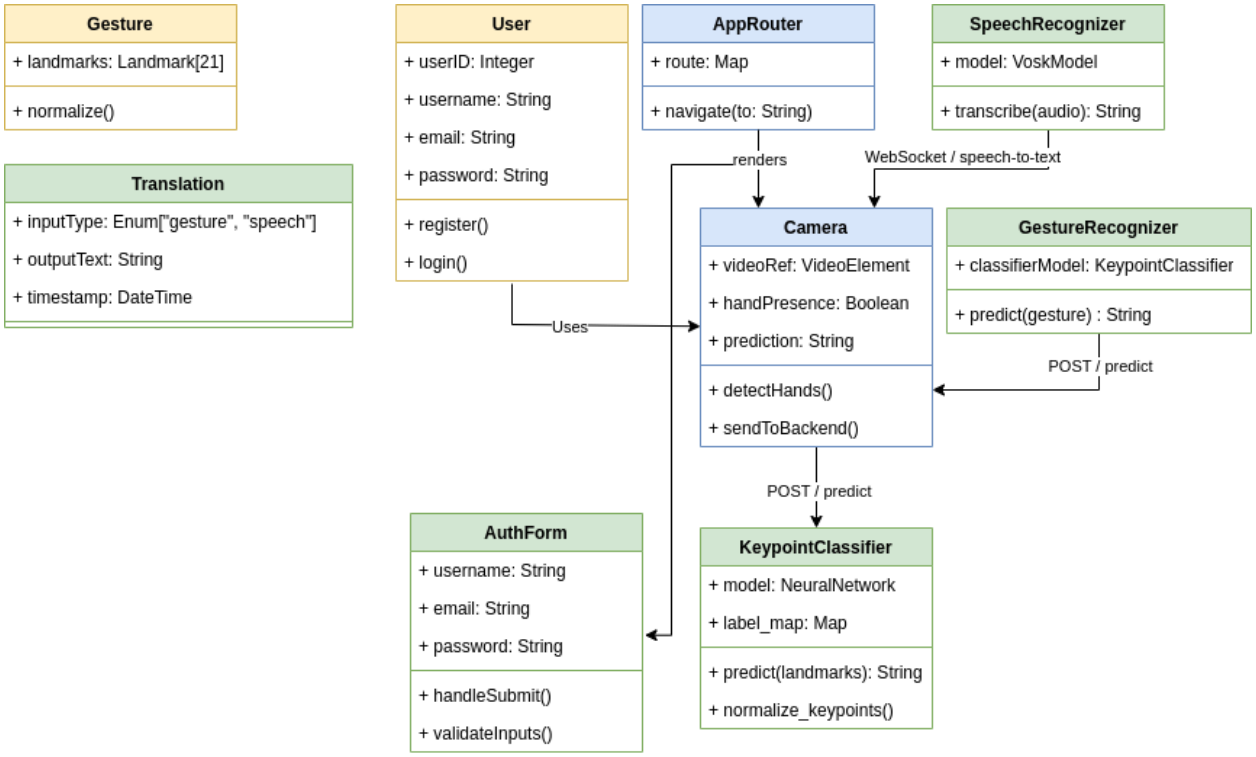
Service Contracts






Text-to-Sign Service Contract
TextToSign
+ showSign(:) : void
+ processSentence(: string) : void
TranslatorPage
{pre: text length > 0}
{pre: server is authenticated and accepting requests}
{post: sign images corresponding to characters displayed}
{post: user able to cycle through images}
{performance: completion in <= 2ms consistently}

Domain Model



Architectural Requirements

- **Architectural Patterns**
  - Micro-Service Architecture
    - Scalability
      - can scale important services independently
    - Incremental development fits well with Agile-Scrum methodology
    - Allows each team member to develop within preferred framework
    - Independent services allows to split the translation system into independent parts, each with clear responsibilities
      - ease of debugging, less code per module
      - independent updates
      - simplifies complex system
      - fault isolation

● Quality Requirements

- ○ Performance:
  - The web app needs to meet performance requirements such as little latency between translations (<= 2s).
  - An accuracy of at least 85% must be met
- ○ Scalability:
  - Independent services that can operate without affecting other parts
- ○ Modularity:
  - Componentization in order for efficient maintenance.

● Design Patterns

- ○ Observer Pattern: Real time updates when services are used and frontend components are scanned.
- ○ Strategy Pattern: Pick specific model to make predictions, etc.

● Constraints

- ○ Stateless microservices to allow easy deployment and horizontal scaling

Technology Requirements

- Technology Stack
    - Front-end (languages)
        - JavaScript
        - HTML
        - CSS
    - Front-end (frameworks)
        - React
        - Tailwind
    - Back-end
        - Python
        - MongoDB