11/1/2024

Language Translation Application

Software System Design Document

# 

Maya Ellison

December 11, 2024

Table of Contents

[0](#_Toc184799846)

[Project Goal: 2](#_Toc184799847)

[Requirements: 2](#_Toc184799848)

[Business Requirements: 2](#_Toc184799849)

[Functionalitites: 2](#_Toc184799850)

[Target Users and Needs: 2](#_Toc184799851)

[Business Goals: 2](#_Toc184799852)

[Non-Functional Requirements: 2](#_Toc184799853)

[Performance Requirements: 2](#_Toc184799854)

[Security Requirements: 3](#_Toc184799855)

[Maintainability Requirements: 3](#_Toc184799856)

[Other: 3](#_Toc184799857)

[Use Case Diagram 4](#_Toc184799858)

[Domain Model 5](#_Toc184799859)

[Class Diagram 6](#_Toc184799860)

[Sequence Diagrams 7](#_Toc184799861)

[State Diagram 8](#_Toc184799862)

[Activity Diagram 9](#_Toc184799863)

[Component Diagram 10](#_Toc184799864)

[Cloud Deployment Diagram 11](#_Toc184799865)

[Skeleton Classes 11](#_Toc184799866)

[User 11](#_Toc184799867)

[Settings 12](#_Toc184799868)

[TranslationRequest 12](#_Toc184799869)

[InputType 12](#_Toc184799870)

[Device 12](#_Toc184799871)

[Language 13](#_Toc184799872)

[LanguageModel 13](#_Toc184799873)

[Subscription 13](#_Toc184799874)

[Tables Definition 14](#_Toc184799875)

[Users Table 14](#_Toc184799876)

[Devices Table 14](#_Toc184799877)

[Languages Table 14](#_Toc184799878)

[TranslationRequests Table 15](#_Toc184799879)

[LanguageModels Table 15](#_Toc184799880)

[Design Patterns 15](#_Toc184799881)

# Project Goal:

The goal of this document is to design a **Language Translation Application** that enables users to communicate seamlessly across different languages. The app will accurately translate text, speech or images from one language to another, supporting multiple languages and dialects.

# Requirements:

## Business Requirements:

### Functionalitites:

* The system shall enable users to input text in one language and receive a translated output in their desired language.
* The system shall allow real-time translation of spoken language.
* The system shall automatically detect language input.
* The system shall translate text through camera view.

### Target Users and Needs:

* Travelers: In foreign countries they need quick access to language translations for assistance with directions, activities and casual conversations.
* Business Professionals: Require translations to facilitate business communications, meetings and contracts.
* General Users: Language translations for personal use or leisure.

### Business Goals:

* Provide reliable translations that users can trust.
* Support a diverse user base across multiple languages and countries.

## Non-Functional Requirements:

### Performance Requirements:

* The system shall handle a growing number of users and large volume of translation requests to scale horizontally to accommodate increased demand.
* Text translations should be delivered in 2 seconds and speech translation should achieve real-time latency.
* The system shall be capable of processing 100 translations per second in both text and speech translations.

### Security Requirements:

* The system shall authenticate users securely using multifactor authentication for account-based features, such as premium content.
* Only eligible users shall have access to additional functionalities, such as premium content.
* All data, including user queries and translation results, must be encrypted in transit and at rest.

### Maintainability Requirements:

* The app shall be developed using modular architecture to simplify updates and troubleshooting.
* Comprehensive documentation shall be maintained to aid in development and maintaining the system continuity.
* The app shall follow continuous integration and continuous deployment (CI/CD) pipeline with automated testing. Performance testing should also be done regularly.

### Other:

* The system shall have an uptime of at least 99.9%.
* The system shall be intuitive and easy to navigate, with clear labels.
* The app shall be compatible across major platforms with consistent user experience.

# Use Case Diagram

A diagram of a language translation application

Description automatically generatedMy Use Case Diagram outlines the main functionalities of the system and the interactions between the actors and the application. The primary actors involved are the *User*, who interacts with the application to request translations or other services, and potentially another *User*, who receives translations during conversations. Each use case represents a distinct feature offered by the app. The *Translate Text* and *Translate Speech* functionalities cater to users needing text or audio translation, respectively, while the *Translate Images* use case allows users to upload images containing text for translation. The *Detect Language* use case enables the app to identify the source language automatically, which enhances user experience by reducing manual input. Lastly, the *Translate Conversation* use case facilitates real-time translation between users during live interactions.

# Domain Model

A screenshot of a computer

Description automatically generatedMy Domain Model focuses on representing the key entities, their attributes, and relationships within the system to provide a conceptual view of the applications structure. The *User* entity captures user-specific such as name, email and preferred language*. Device* represents the hardware used to capture and display input and output. *TranslateRequest* encapsulates the data and logic necessary for processing a translation, linking to *Language* for source and target language details. *TextTranslation, SpeechTranslation,* and *ImageTranslation* are linked to *TraslateRequest* as well, in the form of composition as they cannot exist independently of this class.  *Settings* maintain user preferences, such as default languages and preferences, while *Subscription* tracks billing and usage details. The *LanguageModel* entity encapsulates translation algorithms, supporting various languages and dialects defined in the Language entity. Relationships between these entities, such as a User having multiple Devices, are modeled with appropriate multiplicities as well.

# Class Diagram

A diagram of a computer program

Description automatically generated with medium confidence My Class Diagram provides a static representation of the language translation app, detailing the core classes, their attributes, methods and relationships. Key classes include *User,* which manages user-specific information and actions such as selecting input types and viewing translations, and *Device* which interfaces with the hardware to capture and send input data. The *TranslationRequest* class represents the translation process, containing information about source and target languages*. Settings* manages user preferences like default languages and notifications, while *Subscription* tracks the user’s plan and billing details. The *Language* and *LanguageModel* classes handle language data and translation logic, respectively, and *InputType* defines the different input modalities of text, speech and image. Relationships between these classes illustrate their interactions, such as *User* linking to *Device, TranslationRequest*, and *Settings*, and *TranslationRequest* relying on *Language* and *LanguageModel* for processing.

# Sequence Diagrams

A screenshot of a computer

Description automatically generatedMy Sequence Diagram illustrates the dynamic interaction between components during the translation process, detailing the flow of messages and the order of operations. It begins with the *User* selecting an input type via the *Device*, which captures the input data and forwards it to *TranslationRequest*. The *TranslationRequest* then interacts with the *LanguageModel* to process the translation. Once complete, the translated output is sent back to the *Device* for display to the user.

# State Diagram

A computer screen shot of a diagram

Description automatically generatedMy State Diagram outlines the lifecycle of a translation request, showing the states and transitions triggered by user actions or system processes. The app starts in an *Idle* state, awaiting input. Based on the input type, it transitions to states such as *Text Input, Speech Input*, or *Image Input*. Once input is captured, it is validated, fetched from cache, and processed. Upon successful translation, the system transitions to a completed state, displaying the output to the user. This process can then be repeated.

# Activity Diagram

A screenshot of a computer screen

Description automatically generatedMy Activity Diagram is organized with swim lanes to visualize the workflow across different actors of User, Device, TranslationRequest, and LanguageModel. It begins with the user initiating a request, followed by the device capturing input and creating a translation request. The translation process is handled by the AppController and processed by the LanguageModel. The translated result is then returned to the device for display.

# Component Diagram

A diagram of a language

Description automatically generatedMy Component Diagram represents the logical architecture of the language translation app, showing the components involved in the system and their interactions. The *Translation Input Component* is responsible for capturing user input in various formats, such as text, speech, or images. It works closely with the *Device Manager Component* which abstracts device-specific operations and provides a unified interface for handling input and displaying output. At the heart of the system lies the *Translation Service Component*, which orchestrates the translation process. It receives input from the *Translation Input Component*, processes it through the *Language Model Manager Component*, and sends the results to the *Translation Output Component*. Finally, the *Translation Output Component* formats and delivers the translated content to the user.

# Cloud Deployment Diagram

A computer screen shot of a diagram

Description automatically generatedMy Deployment Diagram models the system’s physical architecture, highlighting the cloud-based infrastructure. The user interacts with the app via a mobile or web client, which connects to an API Gateway. The gateway routes requests to an Application Server hosted in a cloud environment, with data storage handled by a database server. The inclusion of artifacts like translation data, user preferences, and subscription details ensures alignment with the system’s functionality and supports the app’s distributed architecture.

# Skeleton Classes

## User

class User:

def \_init\_(self, userId, name, email, subscriptionId):

self.userId = userId

self.name = name

self.email = email

self.preferredLanguage = preferredLanguage #Instance of Language

self.device = device #Instance of Device

def selectInputType(self, inputType):

def provideInput(self, inputData):

def viewTranslation(self):

## Settings

class Settings:

def \_init\_(self, defaultLanguage, defaultTargetLanguage, notificationsEndabled):

self.defaultLanguage = defaultLanguage #Instance of Language

self.defaultTargetLanguage = defaultTargetLanguage #Instance of Language

self.notificationsEndabled = notificationsEndabled #Bool

def toggleNotifications(self):

def setDefaultTargetLanguage(self, language):

self.defaultTargetLanguage = language

## TranslationRequest

class TranslationRequest:

def \_init\_(self, sourceLanguage, targetLanguage, inputText, inputType):

self.sourceLanguage = sourceLanguage #Instance of Language

self.targetLanguage = targetLanguage #Instance of Language

self.inputText = inputText

self.inputType = inputType # Instance of InputType

def generateTranslation(self):

## InputType

class InputType:

def\_init\_(self, inputType):

self.inputType = inputType

def text(self):

def speech(self):

def image(self):

## Device

class Device:

def \_init\_(self, deviceId, deviceType, inputType, inputData):

self.deviceId = deviceId

self.deviceType = deviceType

self.inputType = inputType

self.inputData = inputData

def captureInput(self):

def sendInput(self):

def displayResult(self):

## Language

class Language:

def \_init\_(self, languageId, languageName, languageCode):

self.languageId = languageId

self.languageName = languageName

self.languageCode = languageCode

def getLanguageCode(self):

return self.languageCode

def getLanguageName(self):

return self.languageName

def getSupportedLanguages():

## LanguageModel

class LanguageModel:

def \_init\_(self, modelType):

self.modelType = modelType

def processTranslation(self, sourceText, sourceLanguage, targetLanguage):

def isLanguageSupported(self, language

## Subscription

class Subscription:

def \_init\_(self, subscriptionId, status, plan, renewalDate):

self.subscriptionId = subscriptionId

self.status = status

self.plan = plan

self.renewalDate = renewalDate #Date

def upgradePlan(self, plan):

def cancelSubscription(self):

def getBillingDetails(self):

# Tables Definition

## Users Table

CREATE TABLE Users (

userId VARCHAR(255) PRIMARY KEY,

name VARCHAR(255) NOT NULL,

email VARCHAR(255) UNIQUE NOT NULL,

preferredLanguageId VARCHAR(10),

deviceId VARCHAR(255),

FOREIGN KEY (preferredLanguageId) REFERENCES Languages(languageId),

FOREIGN KEY (deviceId) REFERENCES Devices(deviceId)

);

## Devices Table

CREATE TABLE Devices (

deviceId VARCHAR(255) PRIMARY KEY,

deviceType VARCHAR(50) NOT NULL,

inputTypeId VARCHAR(50) NOT NULL,

inputData TEXT

);

## Languages Table

CREATE TABLE Languages (

languageId VARCHAR(10) PRIMARY KEY,

languageName VARCHAR(100) NOT NULL,

languageCode VARCHAR(10) NOT NULL

);

## TranslationRequests Table

CREATE TABLE TranslationRequests (

requestId VARCHAR(255) PRIMARY KEY,

userId VARCHAR(255) NOT NULL,

sourceLanguageId VARCHAR(10) NOT NULL,

targetLanguageId VARCHAR(10) NOT NULL,

inputText TEXT,

inputTypeId VARCHAR(50),

timestamp DATETIME DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (userId) REFERENCES Users(userId),

FOREIGN KEY (sourceLanguageId) REFERENCES Languages(languageId),

FOREIGN KEY (targetLanguageId) REFERENCES Languages(languageId)

);

## LanguageModels Table

CREATE TABLE LanguageModels (

modelId VARCHAR(255) PRIMARY KEY,

modelType VARCHAR(50) NOT NULL,

supportedLanguageId VARCHAR(10) NOT NULL,

FOREIGN KEY (supportedLanguageId) REFERENCES Languages(languageId)

);

# Design Patterns

The design of the Language Translation Application incorporates several ley design principles and patterns to ensure scalability, flexibility and maintainability. For GRASP, the Creator pattern is applied where classes such as *User* create related objects like *TranslationRequest,* establishing high cohesion. The design also emphasizes High Cohesion and Low Coupling since each class has a focused purpose, such as *Subscription* to manage billing and *Device* to handle user input, reducing dependencies between components.

The app adheres to SOLID principles to promote robustness. The Single Responsibility Principle (SRP) is reflected in classes like *Settings* for managing user preferences and *TranslationRequest* for processing translations. The Open/Closed Principle (OCP) enables the system to adapt to new input types without modifying existing code. For example, using polymorphism in class InputType. Polymorphic behavior also ensures compliance with the Liskov Substitution Principle (LSP), where subclasses such as TextInput or SpeechInput can replace the base InputType. To avoid unnecessary dependencies, the Interface Segregation Principle (ISP) ensures that interfaces expose only relevant methods, as seen in LanguageModel. Finally, the Dependency Inversion Principle (DIP) ensures that abstractions are prioritized, allowing translation workflows to remain decoupled from implementation details.

The design leverages several GOF patterns as well. The Factory Method is used for creating *InputType* objects dynamically based on user preferences, while the Strategy pattern supports flexible selection of translation algorithms in *LanguageModel*. Additionally, for global configuration management, the Singleton pattern could be applied to *Settings* or *LanguageModel*.

Microservices best practices are woven into the design to support scalability. Functionalities such as translation processing, subscription management, and user settings can be split into individual services, adhering to the principle of Service Separation. An API Gateway routes client requests to appropriate services to simplify client-server communication. Each service can maintain its database which follows the Database per Service principle for data isolation. By implementing these principles and patterns, the application achieves a modular, scalable, and user-friendly architecture that can adapt to future requirements while maintaining reliability and performance.