Machine Translation Service

~ SpringBoot API ~

Developed by :

Tomislav Krstić

1. Introduction

The Machine Translation API Service is a Spring Boot project developed to introduce an adapter service that exposes an API for machine translation. The goal is to reduce costs by validating requests before sending them to a third-party machine translation service. This documentation provides an overview of the project, details about the implemented API, assumptions, design choices, and recommendations for making the project production-ready.

1. Code design description

The project follows the Model-View-Controller architecture for organizing code into distinct layers of models, view and controllers, providing separation of concerns and modularity.

Spring Data JPA enables integration with the database layer, with MySQL serving as the relational database management system for storing application data.

Beside the basic Maven dependencies for web APIs, dependencies such as *spring-boot-starter-validation*, *lombok* and *modelmapper* are included to handle validation ,boilerplate code and object mapping.

1. First request – get all Languages

**Endpoint:** /language

**Response:** Returns the list of all language codes.

**Sample Response:** {"en-US", "en-GB", "fr-FR", "de-DE", ...}

**Description:**

When JPA creates the lang\_code table, an SQL query is used to insert all the languages into the database.

When a GET request arrives, the controller requests data from the service which retrieves them from the database using the DTO class. In this case, the DTO class is used to fetch only the necessary data from the database. ModelMapper is used to implicitly map an instance to a new DTO class.

1. Second request – get all Domains

**Endpoint:** /domains

**Response:** Returns the list of all current domains

**Sample Response:** {"academic", "business", "general", "causal", "creative"}

**Description:**

The similar principle was applied in this request as in the previous one, but there was no need to use the DTO class.

1. Third request – receive all data for content translation

**Endpoint:** /translate

**Response:** Receive all required data from client side and store in database table

**Sample Request Body:**

{

"sourceLanguage":"English (United States)",

"targetLanguage":"French (France)",

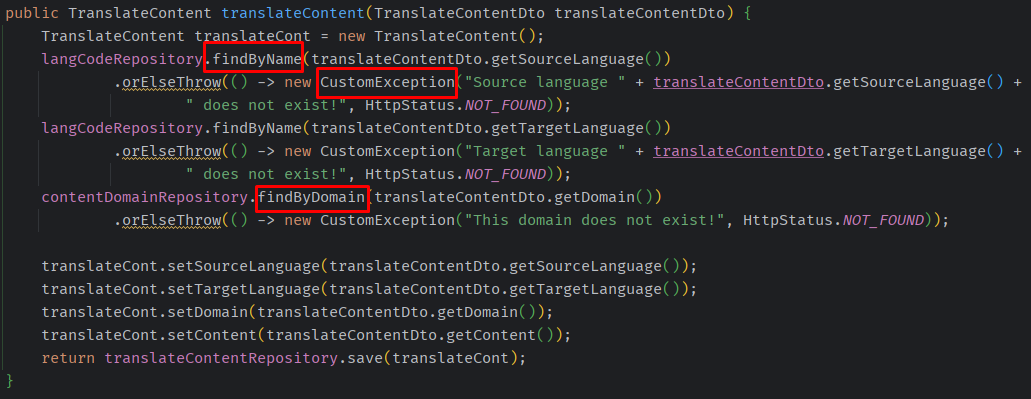
"domain":"academic",

"content":"Where u are ?"

}

**Description:**

To ensure that all fields must not be empty when initiating a POST request, we utilized the *NotBlank* annotation in the DTO class, which will throw an appropriate message whenever a field is empty. The requirement to use only already supported data was addressed by utilizing JPA naming conventions. Methods named findByName and findByDomain were created in the repository, which autonomously generate queries to check if the provided data matches any data in the database. In case the data does not exist in the database, a custom exception will be thrown. (*picture 1.1*)



*Picture 1.1 –method for validation*

1. Fourth request - receive all data for content translation only if content contain maximum 30 words

Endpoint: /validated-translate

Response: Receive all required data from client side and store in database table. Check if the content contains less than 30 words.

Sample Request Body:

{

"sourceLanguage":"English (United States)",

"targetLanguage":"French (France)",

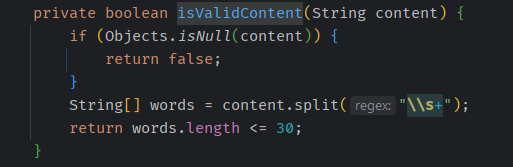
"domain":"academic",

"content":"Where u are ?"

}

**Description:**

Similar implementation to the previous POST request, with the addition of a check if the data exists in the database, there is a method *isValidContent()* (*picture 1.2*)that verifies if the text contains more than 30 words. If it contains more words, an exception will be thrown, providing an appropriate message.



*Picture 1.2 – method for validation words in content*