**TRANSLATE UTILITY**

***FOR AUTOMOTIVE AFTERMARKET***

A hand pouring water onto a plant

Description automatically generated

**University: Kumaraguru College of Technology**

**Mentor Name @BGSW:**

Ms. Sangeetha Jayaprakash (SX/EAA4),

Mr.Vinaykumar M S (SX/EAA4) and

Mr. Nagare Kunal (SX/EAA6)

**Project Guide @University:**

Mr. Kishore R, Asst. Professor. – Automobile Engineering

**Project Team:**

* 1. Kamalaparvathi T M [20BAU015], IV Year – B.E Automobile Engineering
  2. Karthikeyan N [21BIT022], III Year – B.Tech Information Technology
  3. Charan A A[21BAD014], III Year – B.Tech Artificial Intelligence and Data Science
  4. Janarthanan P [21BAD027], III Year – B.Tech Artificial Intelligence and Data Science

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **TITLE** | **PAGE.NO** |
| **1** | **Introduction** | 3 |
| **2** | **Objectives** | 3 |
| **3** | **Solution Concept** | 4 |
|  | 3.1 Architecture Diagram | 5 |
|  | 3.2 Sequence Diagram | 5 |
| **4** | **User Interface** | 6 |
|  | 4.1 UI Components | 6 |
| **5** | **Technological Stack** | 7 |
| **6** | **Features** | 8 |
| **7** | **Use Cases** | 8 |
| **8** | **Implementation** | 9 |
| **9** | **Testing** | 14 |
| **10** | **Future Improvements** | 15 |
| **11** | **Conclusion** | 15 |

**1. INTRODUCTION:**

Positioned as an advanced and versatile solution, the Translate Utility emerges as a powerful tool, granting users the ability to effortlessly engage in seamless string conversion across multiple languages. Tailored with precision to meet the distinctive requirements of BOSCH's esteemed Automotive department, this cutting-edge tool is strategically designed to not only facilitate but elevate multilingual communication and operational efficiency. Its adaptability as a cross-platform solution guarantees accessibility on a diverse range of devices, ensuring that users can harness its capabilities irrespective of their preferred platform. The Translate Utility's commitment to user convenience is further emphasized by its intuitive and user-friendly interface. Additionally, the tool boasts an extensive repertoire of supported languages, making it a comprehensive resource poised to significantly optimize and streamline language-related tasks within the dynamic landscape of the Automotive department at BOSCH.

**2. OBJECTIVES:**

The overarching objective is to create a robust and versatile cross-platform tool expressly designed to streamline and enhance translation tasks. A key highlight is the tool's dedicated focus on file translation, demonstrating exceptional flexibility by seamlessly handling a diverse range of file formats. Significantly, the Translate Utility's functionality is fortified by strategic database support, strategically deployed to mitigate translation costs and catalyse the efficiency of the entire translation process. This deliberate integration of database support is geared towards optimizing both time and resources associated with language translation, positioning the tool as an invaluable asset for BOSCH's Automotive department. In essence, the Translate Utility emerges not merely as a linguistic tool but as a dynamic solution that addresses the multifaceted demands of the Automotive department, promising heightened cost-effectiveness and operational expediency.

**3. SOLUTION CONCEPT:**

Diagram of a project

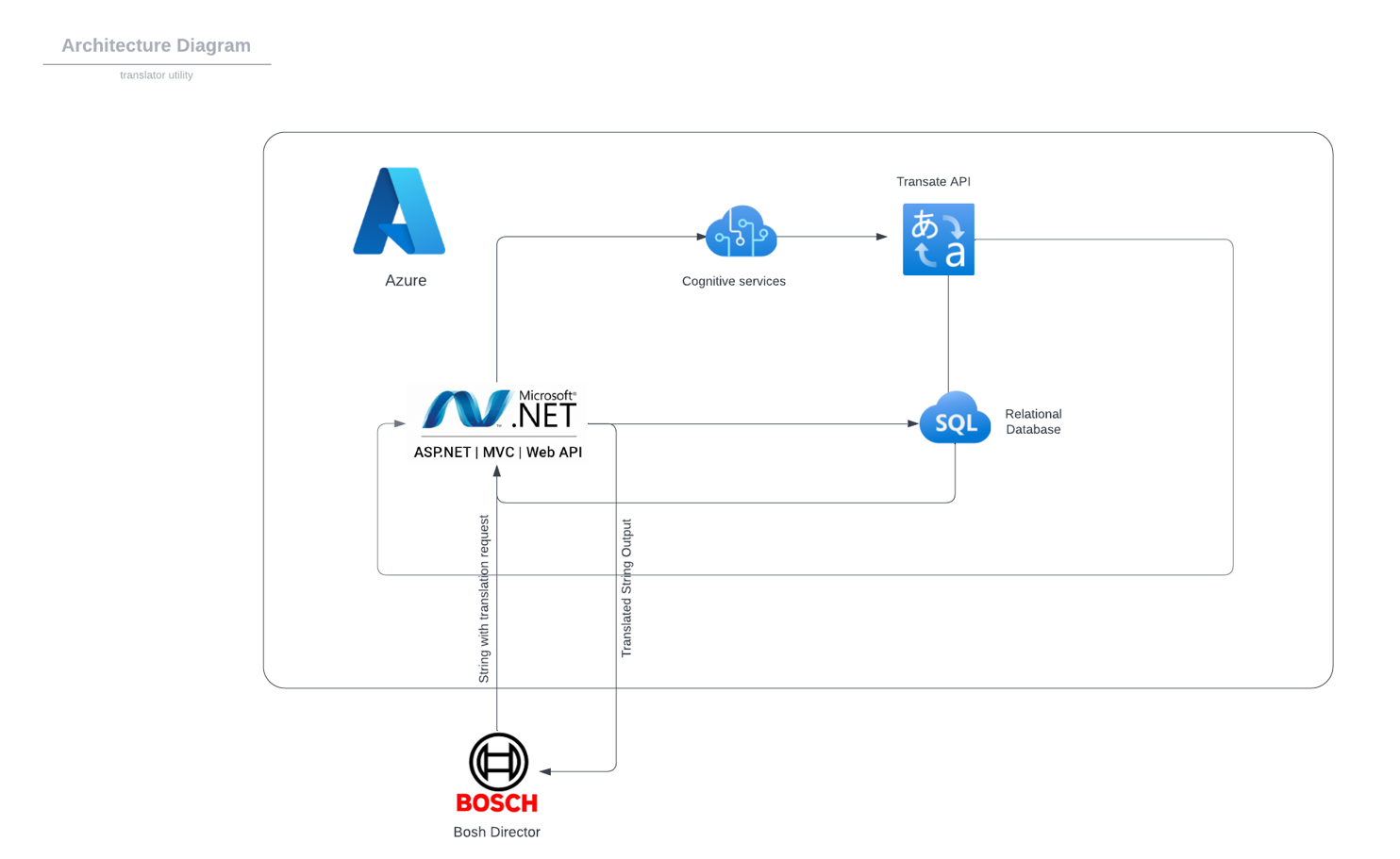
Description automatically generated

Users initiate translation requests within the application, utilizing the integrated API/DLL/SDK. The application checks its local database to determine whether the requested translation is already available. If the requested translation is found in the local database, the application expeditiously retrieves the pre-existing translated string and promptly delivers it to the user. In scenarios where the local database lacks the requested translation, the application dynamically responds forwarding a request to the Microsoft Azure Translation API, tapping into its powerful language translation capabilities. The translated string is obtained from the Azure Translation API, ensuring a precise and accurate rendition of the requested content.

The translated string is promptly dispatched to the user, ensuring a seamless and responsive translation experience. Then a copy of the translated string is intelligently stored back in the local database. This caching mechanism optimizes subsequent requests for the same translation, minimizing reliance on external services.

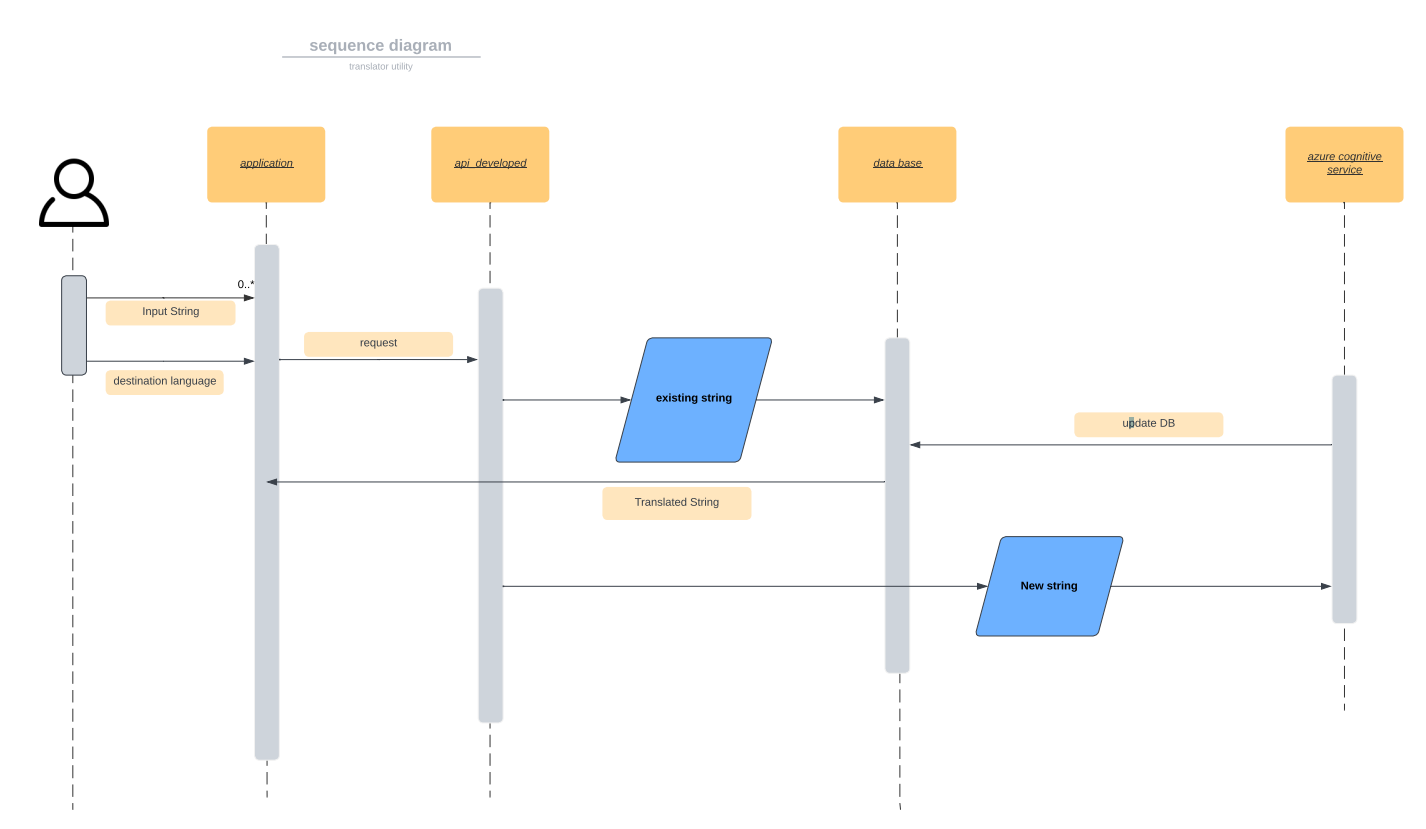
**3.1. ARCHITECTURE DIAGRAM:**

Software architecture establishes structure, enabling scalability and maintainability. A well-designed architecture optimizes resource use, boosts system reliability, and supports adaptability to evolving requirements, forming the bedrock of a robust software product.



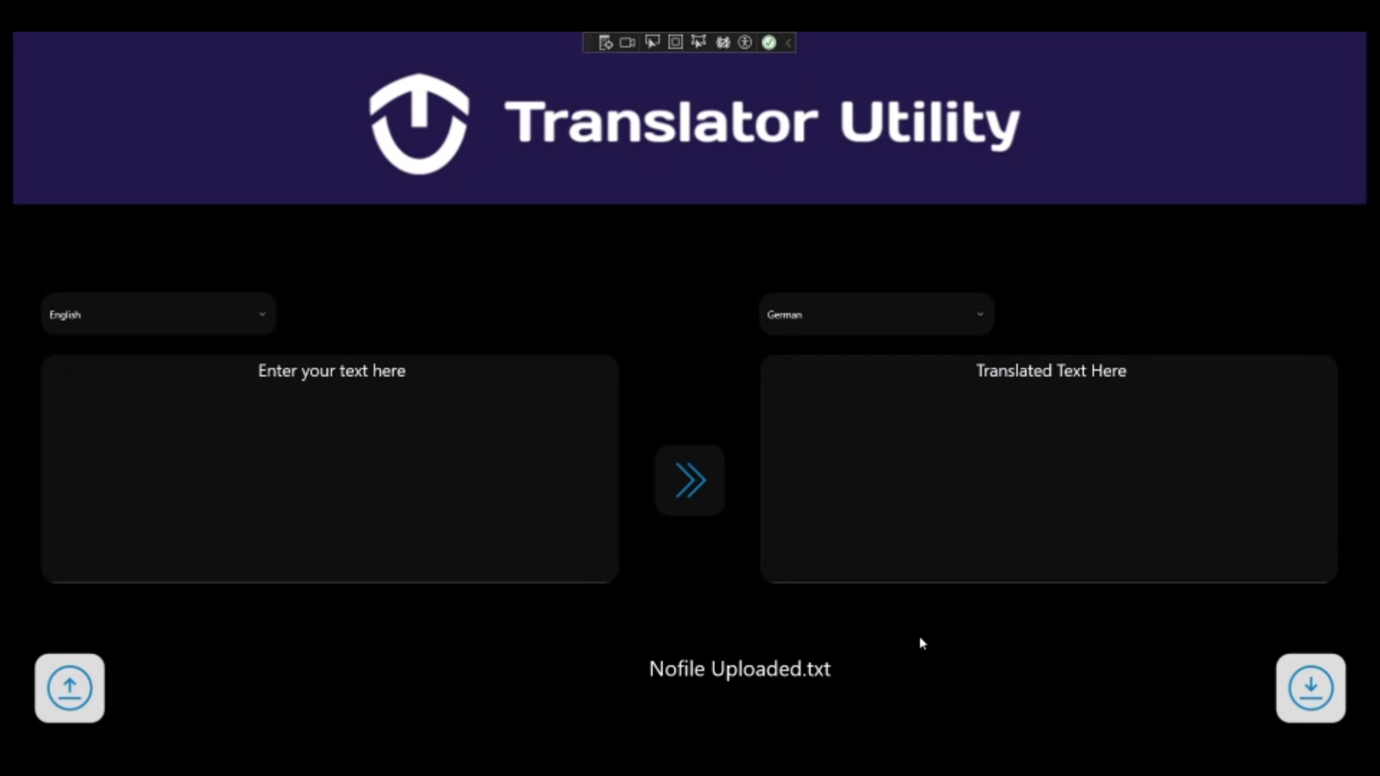
The diagram ensures the architecture of our translate utility with all the resources and technology used.

**3.2. SEQUENCE DIAGRAM:**



The above sequence diagram ensures the sequence of the working of the translate utility from request made to the developed API to the retrieval of

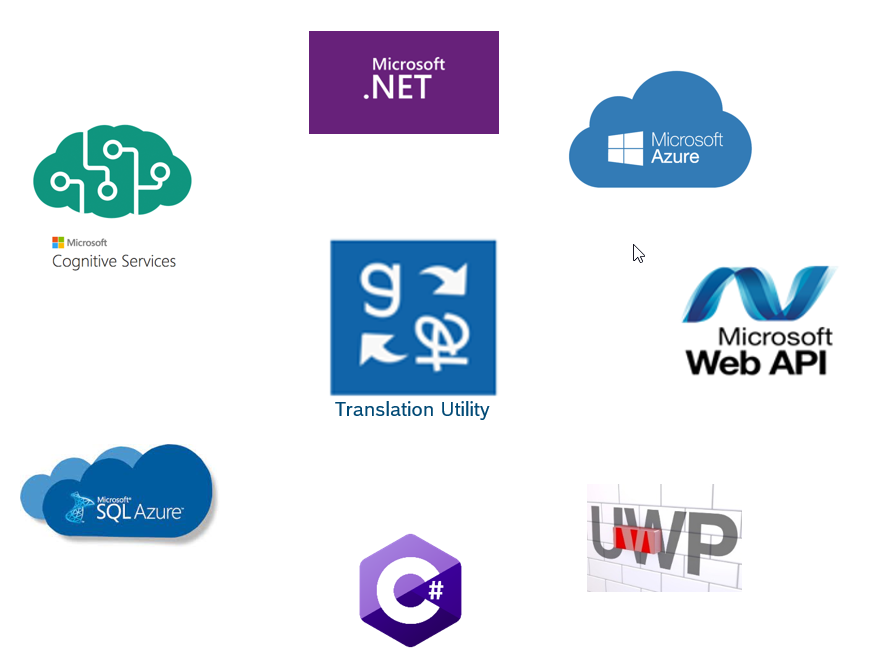
**4. USER INTERFACE:**



**4.1 UI** **Components:**

* Input Language: The drop down on the top-left ensure the selection of the input language of the string for some languages there is a feature of autodetect.
* Output Language: The drop down on the top-right ensure the selection of the output language of the string.
* Input text or string: This input box enables the user to type the text to be translated.
* Translated Text: This is output box where the translated String will appear.
* Translate button: When the button is clicked, the input text is translated into translated text of target language.
* File Upload: The icon on bottom-left ensures the user to upload the file to translate the content in the file.
* File Download: The icon in the bottom-centre ensures the user to download the file which contains the translated text for the input text.

**5. TECHNOLOGICAL STACK:**



* Cognitive Service: The service used in our application is provides by Microsoft Azure.
* Azure Translate API: Azure Translation API is used for the translation of the new translation request.
* SQL Database: The translations are stored in the database to reduce the cost and time and Microsoft Azure SQL is used as a database.
* C#: The functional part or backend of the API is written in C# and hence it could be easily integrated with the different applications.
* UWP: Universal Window Platform is used for the UI components in the demo application.
* Microsoft .NET: The framework used in the API development is .NET framework of Microsoft.

**5.1 API Integration**

Absolutely, here's the revised version of the code with titles and descriptions as requested:

Startup.cs

public void ConfigureServices(IServiceCollection services)

{

services.AddMvc().AddNewtonsoftJson();

services.AddDbContext<TranslationDbContext>(options => options.UseSqlServer(connectionString));

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

app.UseAuthentication();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

}

Description for Startup.cs:

This file sets up the necessary services and middleware for the application. It includes configuration for MVC with JSON serialization, database context for SQL Server, and optionally, authentication and authorization.

TranslationController.cs

[ApiController]

[Route("api/[controller]")]

public class TranslationController : ControllerBase

{

private readonly TranslationDbContext \_dbContext;

public TranslationController(TranslationDbContext dbContext)

{

\_dbContext = dbContext;

}

[HttpPost]

public async Task<IActionResult> Translate([FromBody] TranslateRequest request)

{

string translatedText = await TranslateTextAsync(request.Text, request.SourceLanguage, request.DestinationLanguage);

string existingTranslation = CheckTranslationExists(request.Text, request.SourceLanguage, request.DestinationLanguage);

if (existingTranslation != null)

{

return Ok(existingTranslation);

}

await AddTranslationAsync(request.Text, request.SourceLanguage, translatedText, request.DestinationLanguage);

return Ok(translatedText);

}

}

Description for TranslationController.cs:

This controller handles translation requests. It includes a method to translate text, check for existing translations in the database, and add new translations if they don't already exist.

TranslationDbContext.cs

public class TranslationDbContext : DbContext

{

public TranslationDbContext(DbContextOptions<TranslationDbContext> options) : base(options) { }

public DbSet<Translation> Translations { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Translation>(entity =>

{

entity.HasKey(t => t.Id);

entity.Property(t => t.Text).IsRequired();

entity.Property(t => t.SourceLanguage).IsRequired();

entity.Property(t => t.DestinationLanguage).IsRequired();

entity.Property(t => t.TranslatedText).IsRequired();

});

}

}

Description for TranslationDbContext.cs:

This class defines the database context for the application, specifying the `Translation` entity and its required properties. It sets up the model configurations necessary for the database schema.

Translation.cs

public class Translation

{

public int Id { get; set; }

public string Text { get; set; }

public string SourceLanguage { get; set; }

public string DestinationLanguage { get; set; }

public string TranslatedText { get; set; }

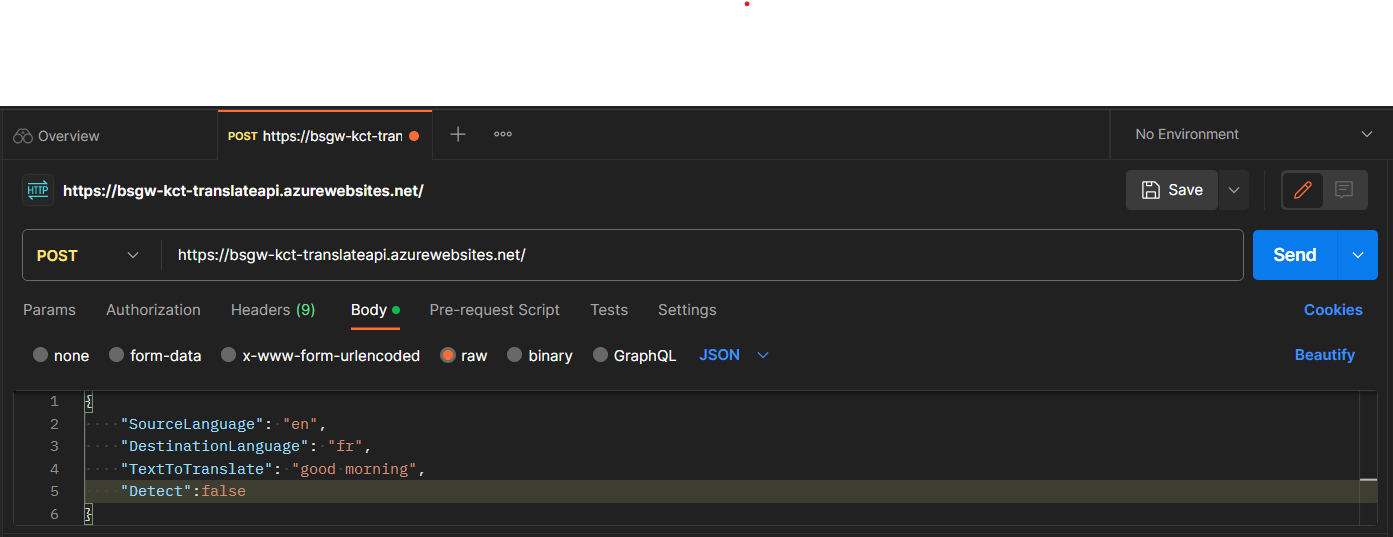
}

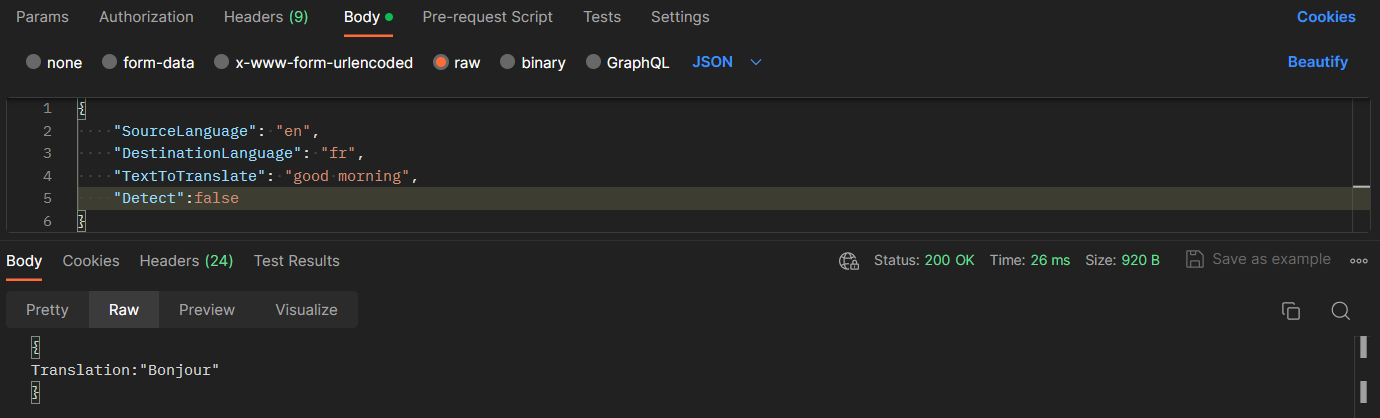
Description for Translation.cs:

This is the model representing a translation. It includes properties such as the text to be translated, source and destination languages, and the translated text.

These files collectively create a backend API for a translation service using ASP.NET Core and a SQL Server database, enabling the storage and retrieval of translations.

Preview:





**6. FEATURES:**

**6.1 String Translation:** OurTranslate utility provides the option of single string translation from one language to other language.

**6.2 File Translation:** Our Translate utility also provides an option of file translation from one language to other language.

(files formats supported: excel, .txt)

excel file format: constraint – document with first column as input string with the first row as source language.

**7. USE CASES:**

The following are the use cases specific to automobile after market:

* Translate vehicle user manuals, maintenance guides, and documentation into multiple languages to ensure that customers worldwide can easily understand and follow instructions.
* Facilitate communication within the global supply chain by translating documentation related to parts, manufacturing processes, and logistics, fostering collaboration with international suppliers.
* Implement multilingual customer support services, including translated service request forms and hotline assistance to communicate effectively with customers from diverse language backgrounds.
* Translate diagnostic tools, software interfaces, and error code descriptions to enable technicians to troubleshoot and identify issues accurately, regardless of language differences.
* Translate parts ordering systems, inventory management interfaces, and supply chain communications to streamline international procurement processes and collaboration with global suppliers.

**8. IMPLIMENTATION:**

**8.1 Database Creation:**

The below code helps to create the table in the database for the first time if it is not found found in the database

public void DatabaseCreate()

{

string connectionString = "Data Source=(local);Initial Catalog=master;Integrated Security=SSPI;Encrypt=false";

using SqlConnection connection = new(connectionString);

connection.Open();

// Check if the table exists

string checkTableQuery = @"

IF NOT EXISTS (SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = N'TXT01\_TRANSLATED\_TEXTS')

CREATE TABLE TXT01\_TRANSLATED\_TEXTS

(

TXT01\_TRANSLATION\_TEXT\_ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY,

TXT01\_TEXT\_ID INT NOT NULL,

TXT01\_LANG\_ID VARCHAR(20) NOT NULL,

TXT01\_TEXT NVARCHAR(3000) NOT NULL,

TXT01\_CREATED\_DATE DATETIME NOT NULL,

CONSTRAINT UC\_Id\_Language UNIQUE (TXT01\_TEXT\_ID, TXT01\_LANG\_ID)

);

";

using SqlCommand checkTableCommand = new(checkTableQuery, connection);

checkTableCommand.ExecuteNonQuery();

}

**8.2 Code to check the Database for the Existing Translation:**

The below code checks for the translations in the database and to return the string if it is present .

public string CheckTranslationExists(string stringValue, string sourceLanguage, string destinationLanguage)

{

string connectionString = "Data Source=(local);Initial Catalog=master;Integrated Security=SSPI;Encrypt=false";

using SqlConnection connection = new(connectionString);

connection.Open();

string query = @"

DECLARE @textId INT;

SELECT @textId = TXT01\_TEXT\_ID

FROM TXT01\_TRANSLATED\_TEXTS

WHERE TXT01\_TEXT = @stringValue AND TXT01\_LANG\_ID = @sourceLanguage;

IF @textId IS NULL

BEGIN

SELECT '-1' AS TXT01\_TEXT;

END

ELSE IF EXISTS (

SELECT 1

FROM TXT01\_TRANSLATED\_TEXTS

WHERE TXT01\_TEXT\_ID = @textId AND TXT01\_LANG\_ID = @destinationLanguage

)

BEGIN

SELECT TXT01\_TEXT

FROM TXT01\_TRANSLATED\_TEXTS

WHERE TXT01\_TEXT\_ID = @textId AND TXT01\_LANG\_ID = @destinationLanguage;

END

ELSE

BEGIN

SELECT '{' + CAST(@textId AS VARCHAR(10)) + '}' AS TXT01\_TEXT;

END

";

using SqlCommand command = new(query, connection);

command.Parameters.AddWithValue("@stringValue", stringValue);

command.Parameters.AddWithValue("@sourceLanguage", sourceLanguage);

command.Parameters.AddWithValue("@destinationLanguage", destinationLanguage);

object result = command.ExecuteScalar();

return Convert.ToString(result);

}

**8.3 Code to request to the Azure Translate API if not found in database:**

The below code sends the request to the Azure API if the translation is not found in the database.

public async Task<string> TranslateString(string stringValue)

{

string sourceLanguage = ((ComboBoxItem)SourceComboBox.SelectedItem)?.Tag?.ToString() ?? SourceComboBox.Items[2].ToString();

string destinationLanguage = ((ComboBoxItem)DestinationComboBox.SelectedItem)?.Tag?.ToString() ?? DestinationComboBox.Items[2].ToString();

try

{

// null string

if (string.IsNullOrEmpty(stringValue)) { return stringValue; }

// source and destination same

if ( sourceLanguage == destinationLanguage) { return stringValue; }

// Trim the empty space

stringValue = stringValue.Trim();

// Remove any <p> tags.

stringValue = Regex.Replace(stringValue, @"<p>", "");

// Remove any </p> tags.

stringValue = Regex.Replace(stringValue, @"</p>", "");

// Trim the empty space

stringValue = stringValue.Trim();

DatabaseCreate();

string CheckedTranslation;

if (sourceLanguage == "detect")

{

// Just detect language

string route = "/detect?api-version=3.0";

string textToLangDetect = stringValue;

object[] body = new object[] { new { Text = textToLangDetect } };

var requestBody = JsonConvert.SerializeObject(body);

using (var client = new HttpClient())

using (var request = new HttpRequestMessage())

{

// Build the request.

request.Method = HttpMethod.Post;

request.RequestUri = new Uri(endpoint + route);

request.Content = new StringContent(requestBody, Encoding.UTF8, "application/json");

request.Headers.Add("Ocp-Apim-Subscription-Key", key);

request.Headers.Add("Ocp-Apim-Subscription-Region", location);

// Send the request and get response.

HttpResponseMessage response = await client.SendAsync(request).ConfigureAwait(false);

// Read response as a string.

string result = await response.Content.ReadAsStringAsync();

JArray jsonArray = JArray.Parse(result);

string detectedLanguage = jsonArray[0]["language"].ToString();

sourceLanguage = detectedLanguage.ToUpper();

}

}

**9. TESTING:**

|  |  |  |  |
| --- | --- | --- | --- |
| **s.no** | **Test name** | **Test description** | **Status** |
| 1 | Language selection | Select the input and output languages from the drop down and proceed for the translations | Successful |
| 2 | String Translation | Give the input strings in the input box after language selection and hit on translate (>> button) and the translated string is showed in the output box | Successful |
| 3 | File translation | Using the upload button on the bottom upload the input file after choosing the suitable file format and then click on translate for translation. Then download the translated file for the future usage using the file download button | Successful |
| 4 | Database check | Check the requested translation by the user in the database and return the translated text to the user | Successful |
| 5 | Request to API | If the translation request is not present in the database, then request to the Azure translation service | Successful |
| 6 | Updating database | Add the new translated response from the Azure Api to the database for future use | Successful |
| 7 | Auto detection | The given string is sent to MS translate for language detection and then string is translated to desired language. | Successful |
| 8 | Special Characters & Other Language | Special Characters or other language alphabets are identified appropriately. | Successful |

These are some of the tests and many test cases are tested for those tests in the translate utility and verified the results.

**10. FUTURE IMPROVEMENTS:**

**10.1 Numeric and special characters:** The translation tool also translates the numeric and special characters if given as input.

(ex: @#$^ -> @#$^, 325734 -> 325734)

It can be improved in future.

**10.2 Uploading and progress animations:** The applications does not show any animations for the process of uploading or translating, it gives a message only after the file upload and translation completion. In future any animations could be developed to track the progress of the process to make it easier for the user.

**10.3 Dual time Check:** Till now the translations from database are only two ways, for example it can only apply (English-> German, German -> English) but it cannot provide options of transitive translations.

(ex: English -> German, German -> French is present in database for same text it could not find translation of English -> French, it could be solved in the future by updating the database two times, before and after translations)

**10.4 Multiple Translations:** The ability to translate the same string to different languages at the same time.

**CONCLUSION:**

Thus, the developed API satisfies many use cases of customers as well as Sellers, the key features of the API is file translation and multiple translation, so it has a great impact on the translation process. The solution provides an effective way of translation thus reduce the cost and time with the help of the integrated database and it can be integrated with different applications.