

**MS Office Add-in**

Using

**Javascript**

**(Text Translation in Regional language)**

|  |  |  |
| --- | --- | --- |
| **Content** | **Student 1** | **Student 2** |
| Name | Atul Kumar | Ganduri Sreeshanth |
| Roll No | 19CS8163 | 19CS8169 |
| Reg No | 19U10649 | 19U10690 |
| Mentor | Prof Abhijit Sharma | Prof Abhijit Sharma |

**ACKNOWLEDGEMENT**

We wish to express our heartfelt gratitude to the all the people who have played a crucial role in the research for this project, without their active cooperation the preparation of this project could not have been completed within the specified time limit.

We are thankful to Professor *Abhijit Sharma,* dept. CSE NIT Durgapur, for supporting and motivating us to complete this project with complete focus, attention, utmost cooperation and patience.

We would like to thank our parents, friends, and classmates for their encouragement throughout our project period. At last, but not the least, we thank everyone for supporting us directly or indirectly in completing this project successfully.

*Atul Kumar (19CS8163)*

*Ganduri Sreeshanth (19CS8169)*

**TABLE OF CONTENT**

**1. Context**

**2. About the Project**

**3. Project prerequisite**

**4. Project structure**

**5. Steps involved in MS Office Add-in**

**A. Search and replace data in word document**

**B. REST API**

**C. Translation**

**D. HTML Parts**

**E. Apply Custom Style**

**F. Text translator Output**

**G. Complete Code**

**6. Use Case in Enterprise**

**1. Context :**

Google Translate is one of the earliest language translation services around. Initially available as a web app to detect and translate between languages, Google Translate is now also available as an API. Google Translate API supports over a hundred languages. Further, you can use the MS Office Add-ins platform to build solutions that extend Office applications and interact with content in Office documents.

Add-ins provides the following:

**● Add new functionality to Office clients** - Bring external data into Office, automate Office documents, expose functionality from Microsoft and others in Office clients, and more.

**● Create new rich, interactive objects that can be embedded in Office documents** - Embed maps, charts, and interactive visualizations that users can add to their own Excel spreadsheets and PowerPoint presentations.

With Office Add-ins, we can use familiar web technologies such as HTML, CSS, and JavaScript to extend and interact with Outlook, Excel, Word, PowerPoint, OneNote, and Project.

**2. About the project:**

The goal of this project is to develop Office Add-in, that can be added in Microsoft Word and Microsoft PowerPoint (This add-in would also work with other MS office Applications such as Outlook, Excel, OneNote, and Project. However, you need to ensure that it works properly with Word and PowerPoint). Whenever a new text document is created or an existing text document is opened in Word, the add-in will start interacting with content in the documents.

The task of the add-in is to translate the parts of the text document into regional languages. The user will enclose the intended text to be translated into specified symbols. At least three set of symbols have to be selected for specific language [For e.g., $$Lang-1$$, €€Lang-2€€, and ££Lang-3££]. Based on the symbol pairs, the add-in will decide the target language. The text document can have the source text to be translated (enclosed in the symbols) in any location of the document and can occur multiple times. The functionality of the add-in will remain same for the PowerPoint.

Note:

1. The text translation service needs to be accessed from Google’s Translate API. However, you are free to consider any other interesting web content instead of ‘Text Translation’. You can use any other service provider or better if you develop your own resource provider. The condition is to access the web content through Rest API and the number of services be at least three.
2. The add-in should be allowed to update the content automatically or when the user opts to do so.
3. A suitable interface for add-in configuration by the user may also be provided.

### **3. Project Prerequisites:**

This project requires knowledge of basic HTML, CSS, and JavaScript. We install the following modules for the project –

Open command prompt as administrator and start installing –

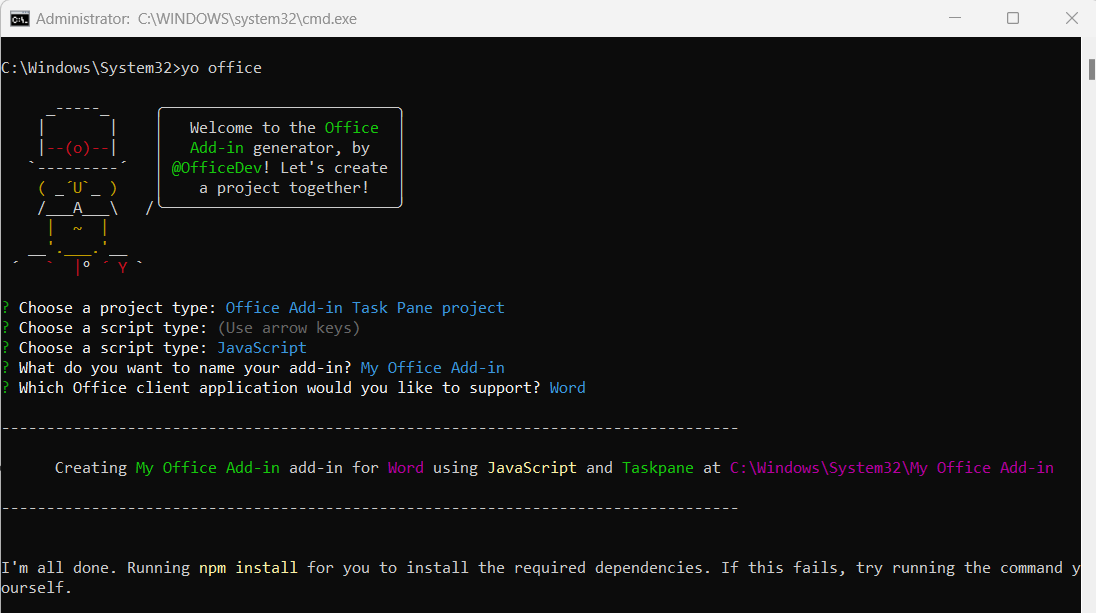
npm install -g yo generator-office

Run the following command to create an add-in project using the Yeoman generator.

yo office

When prompted, provide the following information to create your add-in project.

* **Choose a project type:** Office Add-in Task Pane project
* **Choose a script type:** Javascript
* **What do you want to name your add-in?** My Office Add-in
* **Which Office client application would you like to support?** Word



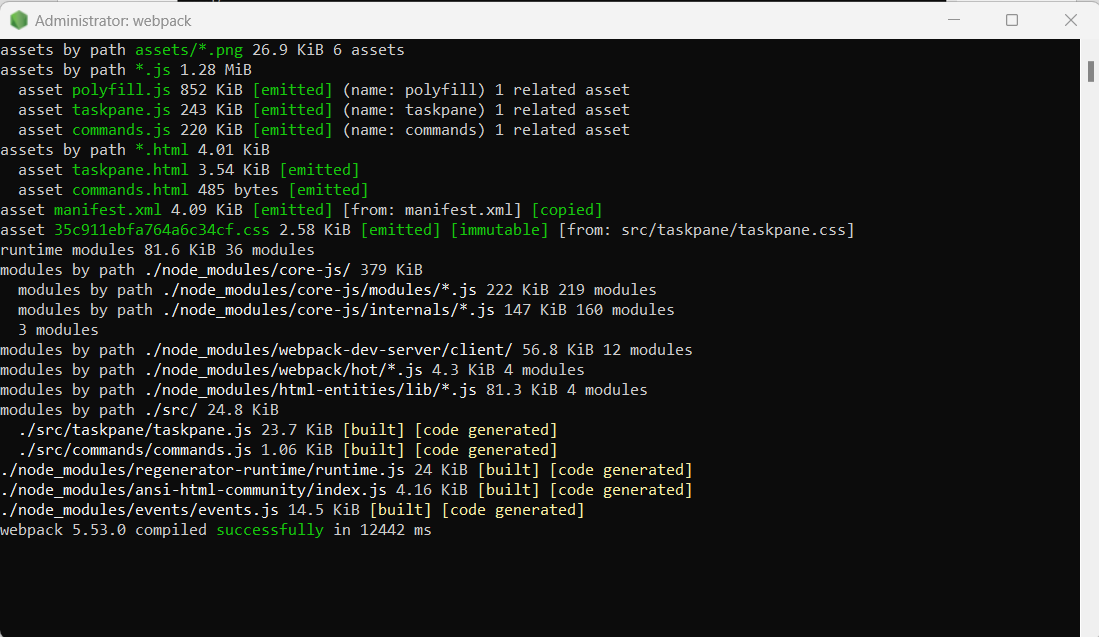
### **4. Project Structure :**

1. Navigate to the root folder of the project.

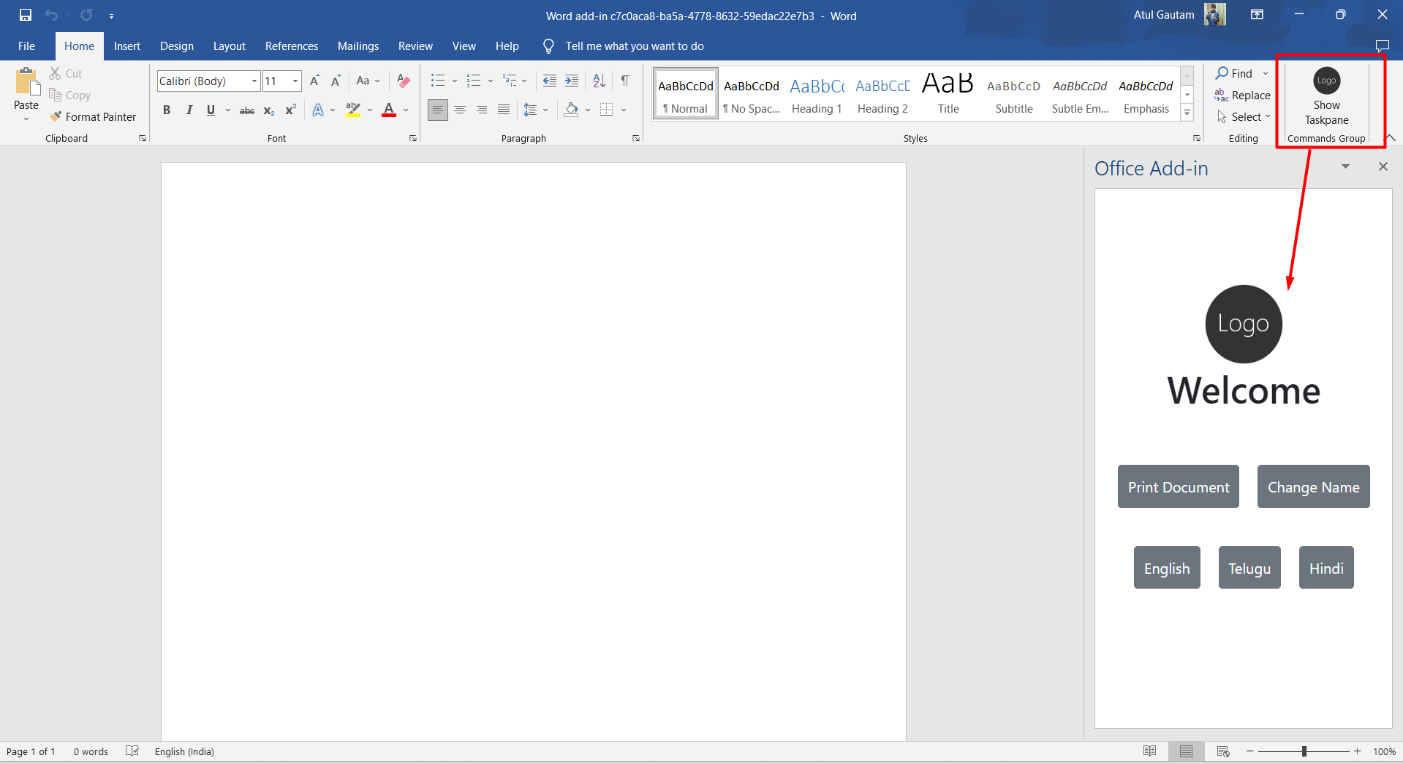
cd "C:\Users\a234g\Office Add-in"

2. Complete the following steps to start the local web server and sideload your add-in.

npm start



**Now this is how our office add-in will look like**.



### **5. Steps involved in MS Office Add-in:**

#### **Search and replace data in word document**

Firstly we will open the folder in vs code and import all the libraries which have been shared in the prerequisites section, and that is how our code will look like.

**Code:**

async function find() {

  return Word.run(async (context) => {

    var results = context.document.body.search("World"); //Search for the text to replace

    context.load(results);

    return context

      .sync()

      .then(function () {

        for (var i = 0; i < results.items.length; i++) {

          results.items[i].insertHtml("Shan", "replace");

          results.items[i].font.color = "blue";

          results.items[i].font.size = 20;

        }

      })

      .then(context.sync);

  })

  .catch(function(e){

    console.log(e.message);

    if (error instanceof OfficeExtension.Error) {

      console.log("Debug info: " + JSON.stringify(error.debugInfo));

    }

  });

}

#### **REST API.**

An API (application programming interface) as we know, is a set of rules that facilitates communication between different application programs. APIs play a critical role in optimizing application performance and aid in delivering consistent user experiences across different screens. The key attribute of APIs is providing a standardized way for two applications to interact and exchange data

## **a) What Is REST API?**

Representational State Transfer (REST) is an architectural style and methodology that is frequently used in developing web-based applications and services. To better understand the concept of REST or RESTful APIs, let’s first get familiar with the following fundamental terms:

### **b) Client**

A client is a person or program that uses the API to perform various tasks or actions. To perform an action or retrieve data, the client must hit the API and request the desired output. For instance, your web browser is a client that interacts with the APIs of different websites to get you the desired page content.

### **c) Resource**

A resource is any piece of information that the client gets when the API is called. For instance, the information that you get upon clicking on a random website button is a resource. Each source has a distinct name or identity commonly referred to as the resource identifier.

### **d) Server**

A server is where the application is stored along with all its resources. It is the most crucial element that receives client requests, processes them (calls the API), and delivers the desired output (resource). Rather than providing complete access to the application resources, a server only grants access to the representational state of the source using REST APIs.

### **e) Why Do We Need REST APIs?**

REST APIs provide a great deal of flexibility and improved modular architecture, not to mention the ease of integration for a variety of applications and third-party services. Besides, REST APIs are stateless, which means they allow clients to make calls independently for different resources. Here, each call contains all the relevant data for its execution.

Unlike the SOAP protocol, REST is not only limited to XML i.e it can provide the output in XML, JSON, YAML, and many other formats based on the client requests. Similarly, REST is better than the RPC protocol (remote procedure call) as the users don’t need to remember the procedure names or several other parameters. One drawback, however, is that you may not be able to maintain a particular state in REST within different sessions. For this reason, it might be difficult for entry-level developers to get the hang of it.

We have used google translate API from a website provide free API call at the rate of 1 lakh characters per/month.

“https://rapidapi.com/gatzuma/api/deep-translate1/”

Code:

const options = {

      method: 'POST',

      headers: {

          'content-type': 'application/json',

          'X-RapidAPI-Host': 'deep-translate1.p.rapidapi.com',

          'X-RapidAPI-Key': '63ef009d41msh201918119c894d8p1565bbjsnaac4f602f9e3'

      },

      body: `{"q": "${a}","source":"en","target":"en"}`

 };

const s = await fetch('https://deeptranslate1.p.rapidapi.com/language/translate/v2', options);

#### **Translation**

Now we are merging both methods:

1. search and replace words/clauses
2. REST API call

This is how our code looks like-

Code:

async function telugu() {

  document.getElementsByClassName('loader')[0].style.display = "block"

  return Word.run(async (context) => {

    // 1. First function ----> Search in doc

    var searchResults = context.document.body.search('$$\*$$', {matchWildcards: true})

    context.load(searchResults);

    return context

    .sync()

    .then (async function(){

      for (let i = 0; i < searchResults.items.length; i++) {

        let a = searchResults.items[i].text

        // 2. Second function ----> Split

        a = a.split('$$').join('')

        let shan = ''

        // 3. Third Function  -----> REST API Called

        const options = {

          method: 'POST',

          headers: {

            'content-type': 'application/json',

            'X-RapidAPI-Host': 'deep-translate1.p.rapidapi.com',

            'X-RapidAPI-Key': '63ef009d41msh201918119c894d8p1565bbjsnaac4f602f9e3'

          },

          body: `{"q": "${a}","source":"en","target":"te"}`

        };

        const s = await fetch('https://deep-translate1.p.rapidapi.com/language/translate/v2', options)

        const response = await s.json()

        // console.log("Hello");

        // console.log(response);

        shan = response.data.translations.translatedText

        // console.log(shan)

        // 4. Fourth function----> Replace

        searchResults.items[i].insertHtml(shan, "replace")

        searchResults.items[i].font.color = "green";

        searchResults.items[i].font.size = 20

      }

      document.getElementsByClassName('loader')[0].style.display = "none"

    }).then(context.sync)

  })

  .catch(function(e){

    console.log(e.message)

    if (error instanceof OfficeExtension.Error) {

      // console.log("Debug info: " + JSON.stringify(error.debugInfo));

    }

  })

}

#### **HTML Parts**

Locate the <button> element for the insert-paragraph button, and add the following markup after that line.

**Code :**

<button type="submit" class="btn btn-secondary ms-Button1" id="run">Print Document</button>

  <button type="submit" class="btn btn-secondary ms-Button1" id="find">Change Name</button>

Within the Office.onReady function call, locate the line that assigns a click handler to the insert-paragraph button, and add the following code after that line.

**Code :**

document.getElementById("run").onclick = run;

Add the following function to the end of the file.

**Code :**

export async function run() {

  return Word.run(async (context) => {

    const paragraph = context.document.body.insertText(

      `£ ----> English\n¥ ----> Hindi\n$ ----> Telugu\n\n================================ \n

      A research paper is an essay in which you explain what you have learned after exploring your topic in depth.

      In a research paper you include information from sources such as books articles interviews and Internet sites.`

      , Word.InsertLocation.end);

    paragraph.font.color = "black";

    paragraph.font.size = 20;

    await context.sync();

  });

}

Locate the <button> element for the paragraph translator button, and add the following markup after that line.

**Code :**

<button type="submit" class="btn btn-secondary ms-Button1" id="english">English</button><br /><br />

    <button type="submit" class="btn btn-secondary ms-Button1" id="telugu">Telugu</button><br /><br />

    <button type="submit" class="btn btn-secondary ms-Button1" id="hindi">Hindi</button>

#### **Apply Custom Style**

We have added a loader gif which will run between fire of API call and till the translation completion.

**HTML Part :**

<div class="loader">

          <img src="https://media1.giphy.com/media/3oEjI6SIIHBdRxXI40/giphy.webp?cid=ecf05e47ndg4ckzapsr6d4hef3jgkkd1quq5virruxtrhbbu&rid=giphy.webp&ct=g"

           alt="" />

        </div>

**CSS Part :**

.loader{

    display: none;

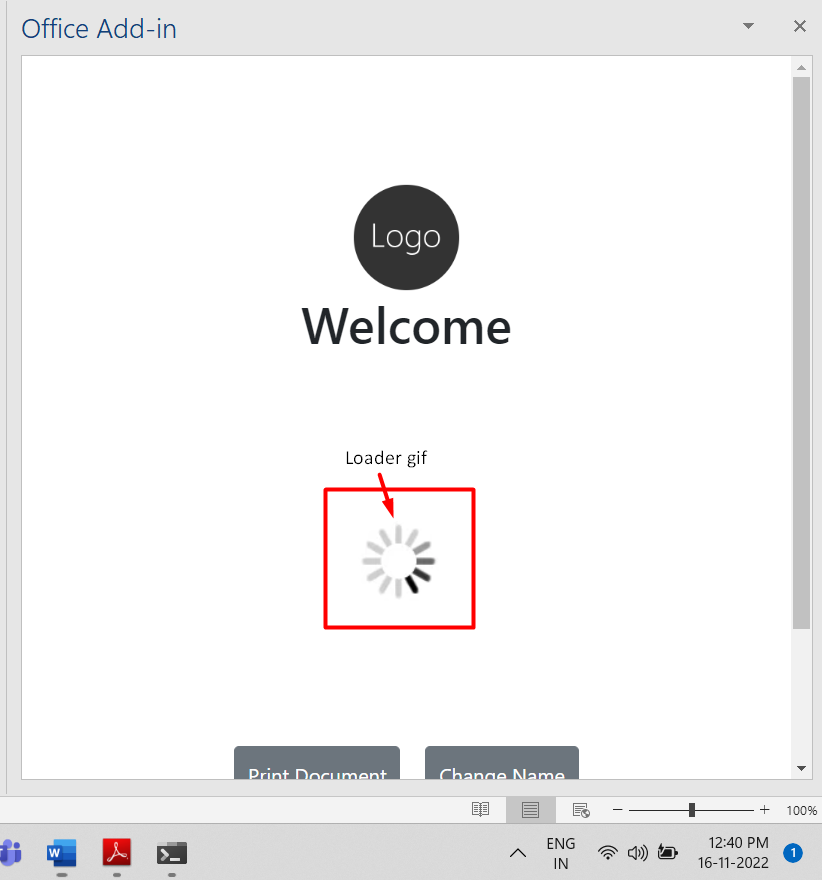
    width: 50%;

    padding-left: 20px;

    margin-left: auto;

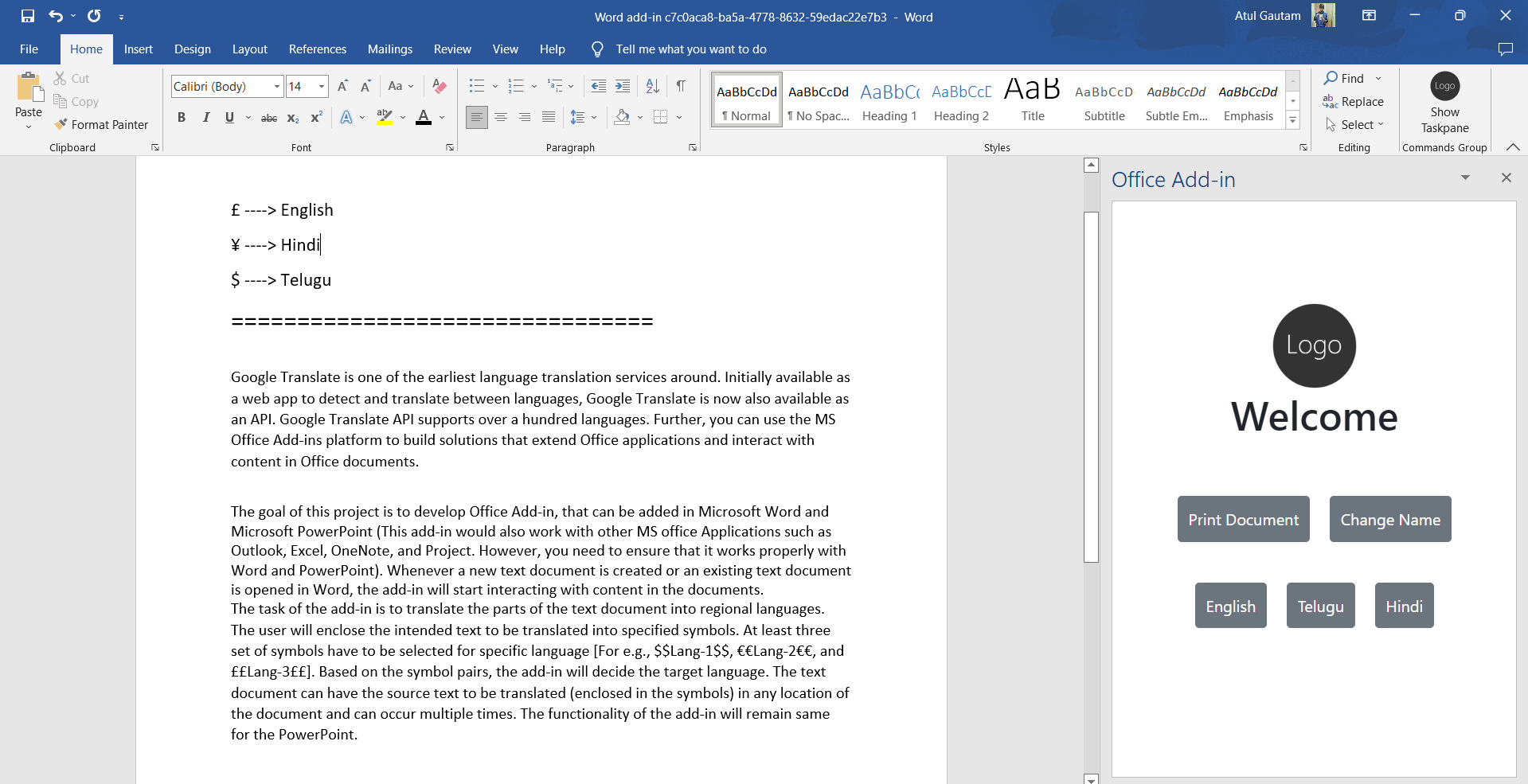
    margin-right: auto;

}

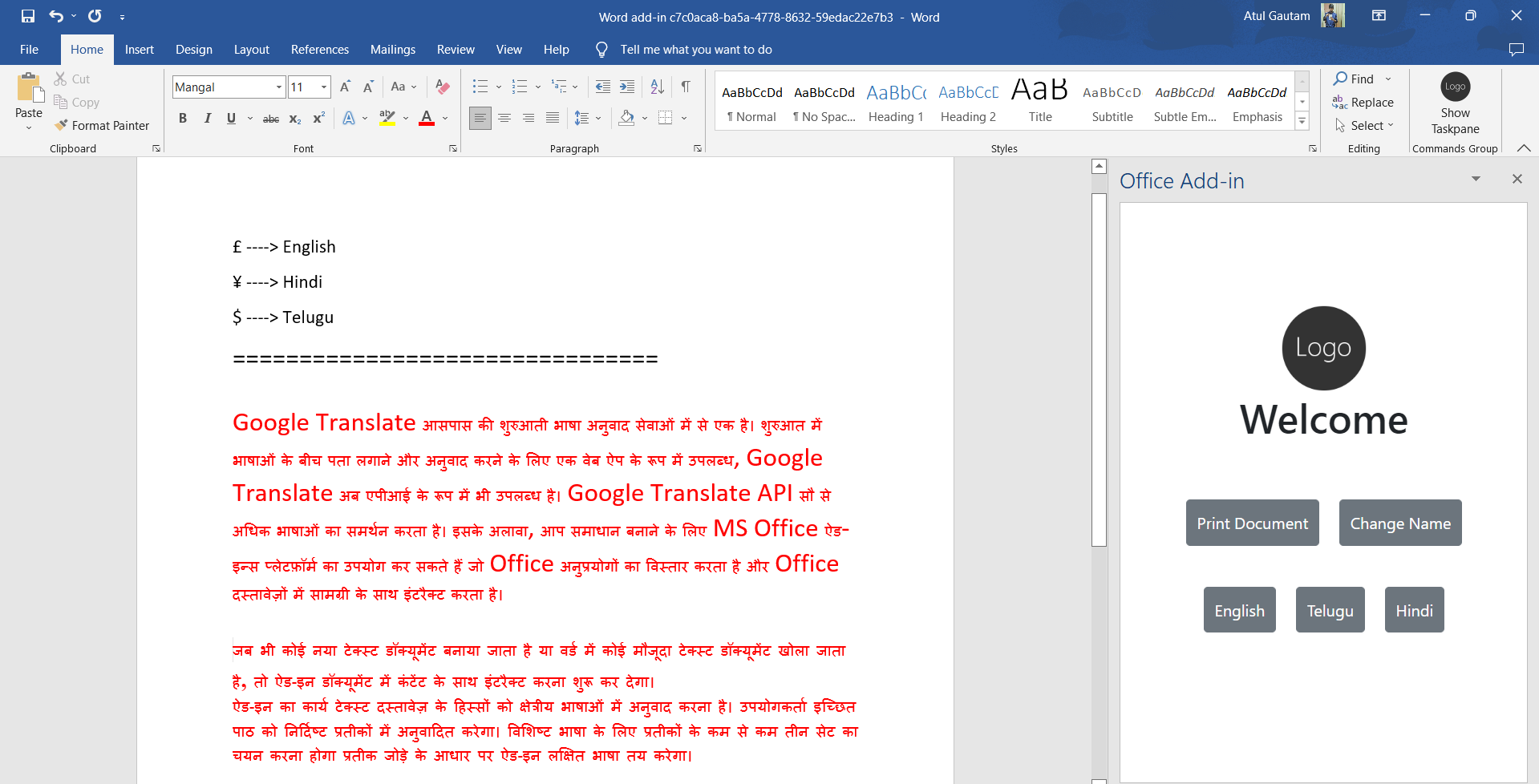


### **Text translator Output**

**Before translation :**

****

**After translation :**

****

1. **Complete code:**

**6. Use cases in the enterprise**

These are some use cases where automatic summarization can be used across the enterprise: