Fordham University

Prof. **Zhou Ji**

CISC5550-Cloud Computing

"Cloud Development and Deployment Project"

Sarah ALJamal

Table of Content

Introduction

This project, developed as part of a comprehensive course on cloud computing, aims to enhance a web-based to-do list application to better serve its diverse user base. The enhancements focus on making the application more accessible, visually appealing, and user-friendly. Key features include the integration of the Google Translate API to accommodate users from various linguistic backgrounds, ensuring that language barriers do not hinder the usability of the tool. Additionally, strategic additions such as a distinctive logo and a descriptive image are incorporated to improve visual engagement and intuitive navigation. The application of a soothing background color is intended to create a more pleasant and less distracting user interface. Through these modifications, the project seeks to provide a more inclusive and engaging digital tool that supports the organizational and planning needs of its users, demonstrating the practical application of cloud computing principles learned in the course.

Objective

The main objective of this project is to enhance the usability and accessibility of cloud -based to-do list application. By integrating the Google Translate API, the project aims to break language barriers, allowing users from various linguistic backgrounds to interact seamlessly with the platform. Additionally, the incorporation of visual elements such as the university's logo and a descriptive image seeks to strengthen brand identity and provide intuitive user guidance. The application of a calming background color further aims to create a visually appealing and conducive environment for users, promoting ease of use and engagement. These enhancements are designed to ensure that the application is not only functional and efficient but also welcoming and inclusive for all users.

Project Description

1-ADD Google Translate API

To enhance the accessibility of the To-Do List application, the Google Translate widget was integrated, enabling users to translate the application content into various languages. This feature is particularly valuable in multicultural and diverse user environments.

Implementation

1. **Widget Script Insertion**
   - The Google Translate widget is powered by a script from Google's translation services. The script is dynamically loaded and initializes the translation widget.
   - Below is the code snippet added to the **<head>** section of the HTML to load the Google Translate API:

```
<scriptsrc="//translate.google.com/translate_a/element.js?cb=googleTranslateElementInit"
```

2. **Widget Initialization**
   - A JavaScript function, **googleTranslateElementInit**, is defined to initialize the Google Translate widget with specific configuration options. This function is specified as a callback in the script URL.
   - The widget is configured to use a simple inline layout which integrates discreetly with the application's UI.
   - Code snippet for initializing the widget:

```
<script> function googleTranslateElementInit() { new google.translate.TranslateElement({
pageLanguage: 'en', layout: google.translate.TranslateElement.InlineLayout.SIMPLE },
'google_translate_element');
```

3. **Modify the widget place**

- To align the Google Translate widget with the application's layout and ensure it fits seamlessly within the user interface, CSS modifications were made.

- The widget was initially placed at the top-right corner of the page, which was not ideal for user interaction. It was moved to a more accessible location above to the logo area, making it easier for users to find and use.

- The **#google_translate_element** container was styled to position the widget appropriately.

Challenges and Solutions

- **Problem:** Initially, the styling of the Google Translate widget did not consistently match the application's theme, which affected the overall user interface consistency.
- **Solution:** Custom CSS was applied to the elements within the Google Translate widget. This involved overriding default styles and ensuring that the widget's appearance aligned with the overall design of the application.

- **Problem:** There was a noticeable delay in the loading and initialization of the Google Translate widget, which sometimes led to a blank space being displayed where the widget should be.
- **Solution:** The script loading method was changed to asynchronous loading, and additional JavaScript was added to display a loading placeholder until the widget was fully initialized. This improved the user experience by eliminating the blank space and providing feedback that the widget was in the process of loading.

- **Problem:**To do list application is served over HTTP and I want to keep the Google Translate script compatible with that, so I adjust the protocol of the Google Translate script source URL to explicitly use HTTP. However, it's important to note that serving a website over HTTP instead of HTTPS can make it vulnerable to various security risks, including man-in-the-middle attacks.

```
 script

src="http://translate.google.com/translate_a/element.js?cb=googleTranslateElementInit">

</script>
```

2. Add a title and change its color, size and bold

```
<h1 style="color: rgb(243, 114, 8); text-align: center; font-weight: bold;">To Do List Project</h1>

<p style="color: rgb(243, 114, 8);text-align: center;">CLoud Computing CISC 5550</p>
```

3-Adding Static Images and Background Color

To enhance the visual appeal and user experience of the To-Do List application, static images were added and a background color was applied. These elements are essential in creating an engaging and aesthetically pleasing interface.Static images, such as logos and descriptive icons, were integrated to visually communicate the purpose of the application and reinforce brand identity.

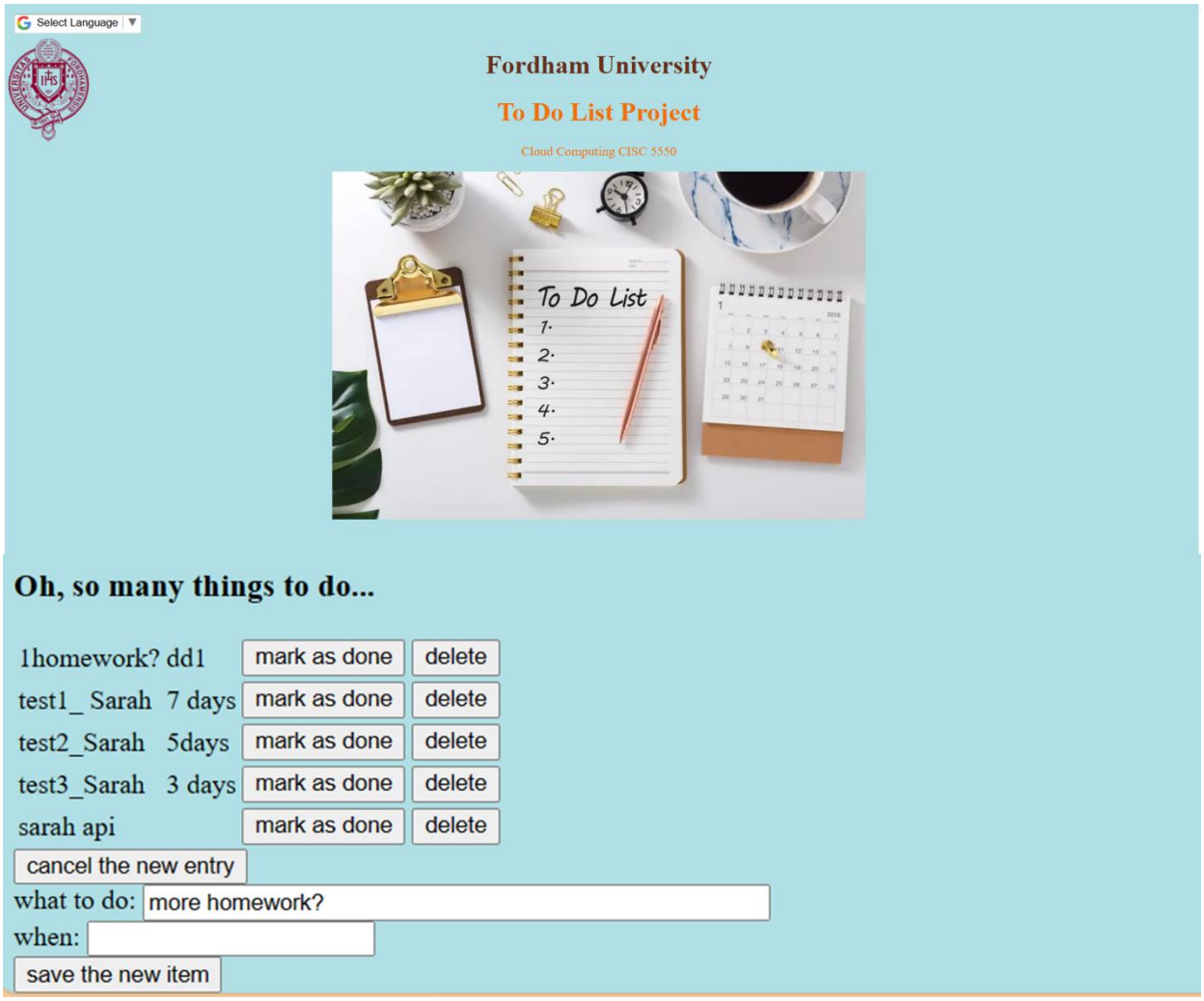


Image Placement

- Images were stored in the **static/images** directory within the application structure. This organizational method simplifies referencing images in the HTML.

HTML Code for Images

- Images were added using the **<img>** tag within HTML. Here's an example snippet for inserting the logo and a descriptive image:

```
<div id="logo-container"> <img src="{{ url_for('static', filename='images/fordham.png') }}" alt="Logo"> </div> <div id="descriptive-image-container"> <img src="{{ url_for('static', filename='images/todolist.jpg') }}" alt="Task Icon"> </div>
```

- The **url_for** function dynamically generates URLs for static files, which is particularly useful in web frameworks like Flask.

4-Adding background Color:
A background color was chosen to create a friendly and calm environment for users interacting with the application.

CSS for Background Color

- The background color was set directly in the **<body>** tag's style attribute to ensure it applies to the entire page.

```
<body style="background-color: powderblue;">
```

- **powderblue** was selected for its soft and neutral tone, which minimizes visual strain and enhances readability.

Challenges and solutions

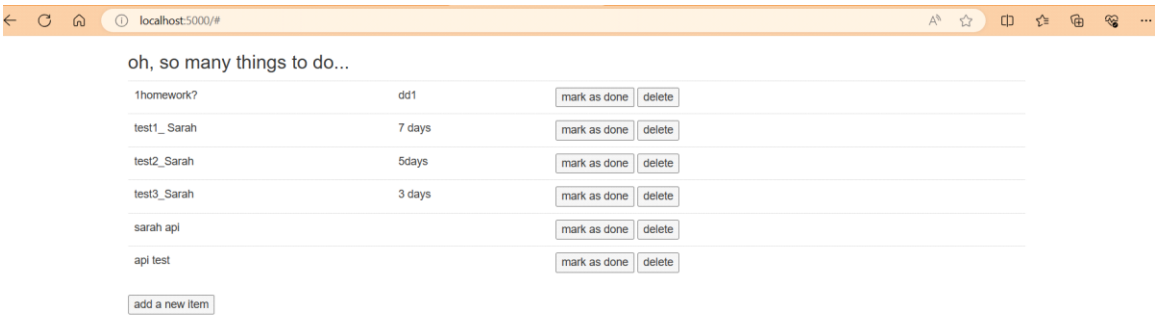
- **Problem:** Initially, the images did not scale properly on different devices, leading to layout disruptions, especially on smaller screens.
- **Solution:** Responsive CSS properties were added to the images. The width was set to a percentage and height was set to auto to maintain aspect ratio.

- **Problem:** The background color appeared inconsistently across different web browsers, affecting the uniformity of the user experience.
- **Solution:** Added browser-specific CSS hacks and ensured that the HTML and body tags were both styled to cover cases where the default margin/padding in browsers affected the background's rendering:

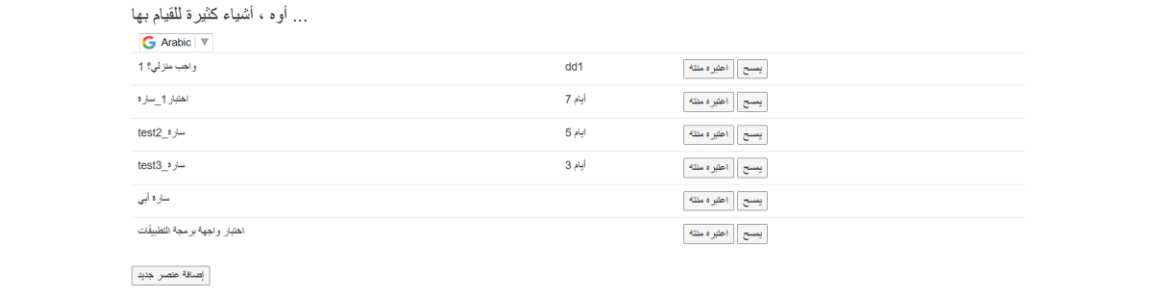
This adjustment guaranteed that the background color appeared consistently across all browsers, providing a uniform user experience. These enhancements—introducing static images for visual aid and applying a soothing background color—have significantly improved the functional aspects of the application, while the challenges encountered were effectively resolved to ensure a smooth and cohesive user experience.

Testing and Results

Initially testing google translate without changing Background or adding any pictures



Use google translate to Arabic



Use google translate to French

Testing google translate with changing Background and adding logo and pictures.

Translate from English to Arabic

Translate from English to French

Conclusion

In conclusion, the enhancements made to the To-Do List application, including the addition of static images such as a logo, the application of a soothing background color, and the integration of the Google Translate API, have significantly enhanced both its appeal and functionality. These improvements not only make the application more visually engaging but also more accessible to a diverse, global user base. By effectively combining visual cues with multilingual support, the application now offers a more intuitive and inclusive user experience, demonstrating how thoughtful design can transform a simple tool into a versatile and user-friendly platform.

References

(Google Translate API documentation).

https://flask.palletsprojects.com/en/3.0.x/tutorial/static/

DOCKERIZATION AND CLUSTERIZATION

1. Create a docker image of the frontend app

    a- Create a docker file that is a text document that contains all the commands a user could call on the command line to assemble an image. Essentially, it automates the process of creating Docker images. Dockerfiles adhere to a specific format and set of instructions you can use to create the image.

```
C:\Users\Sara\Downloads\assignment 4>type Dockerfile
# Use an official Python runtime as a parent image
FROM python:3.8-slim

# Update and install system packages
RUN apt-get update && apt-get install -y --no-install-recommends \
    && rm -rf /var/lib/apt/lists/*

# Install Python packages
RUN pip install --no-cache-dir flask requests

# Set the working directory in the container
WORKDIR /myflask-sarah

# Copy the application files and the templates folder into the container
COPY frontend.py .
COPY backend.py .
COPY todolist.db .
COPY templates ./templates
 # Copy the templates folder into the working directory

# Make port 5000 available to the world outside this container
EXPOSE 5000

# Run frontend.py when the container launches
CMD ["python", "frontend.py"]
```

b-Build and run docker image

```
C:\Users\Sara\Downloads\assignment 4>docker build --no-cache -t finalproject:Latest .
[+] Building 7.8s (5/13)                                                                    docker:default
 => [internal] load build definition from Dockerfile                                                 0.0s
 => => transferring dockerfile: 759B                                                                 0.0s
 => [internal] load metadata for docker.io/library/python:3.8-slim                                   1.8s
 => [auth] library/python:pull token for registry-1.docker.io                                        0.0s
 => [internal] load .dockerignore                                                                    0.0s
 => => transferring context: 2B                                                                      0.0s
 => [1/8] FROM docker.io/library/python:3.8-slim@sha256:2f911e2866173a52104dc16b5e42b7069c2eba05eb78556d18b1ca665  6.0s
 => => resolve docker.io/library/python:3.8-slim@sha256:2f911e2866173a52104dc16b5e42b7069c2eba05eb78556d18b1ca665  0.0s
 => => sha256:11c49f621bb96ca802f21cb54a4940088f0bf85e5948db600433a11da779c774 1.37kB / 1.37kB       0.0s
 => => sha256:5b29acce18dd86b702bc6a011e25b3dd631090f7ffe35fe2a4977abf53eb3954 6.95kB / 6.95kB       0.0s
 => => sha256:b0a0cf830b12453b7e15359a804215a7bcccd3788e2bcecff2a03af64bbd4df7 14.68MB / 29.15MB     6.0s
 => => sha256:72914424168c8ebb0dbb3d0e08eb1d3b5b2a64cc51745bd65caf29c335b31dc7 3.51MB / 3.51MB       3.3s
 => => sha256:545ebfaa75064d83d4862d6b0ca34e531e2e90d431f42a8e59968cc372099695 11.68MB / 11.68MB     3.7s
 => => sha256:2f911e2866173a52104dc16b5e42b7069c2eba05eb78556d18b1ca665d0dc445 1.86kB / 1.86kB       0.0s
 => => sha256:80ee918b20840648abeedaab21c9dd7a6b03105ec8f362d48d55195e0402ebfb 243B / 243B           3.5s
 => => sha256:d361726ad66f2bc2e1928c9b5ddaf7f33b10226d544315ca2408b7d9e2dacd16 3.14MB / 3.14MB       4.9s
 => [internal] load build context                                                                    0.0s
 => => transferring context: 18.42kB                                                                 0.0s

C:\Users\Sara\Downloads\assignment 4>docker run -p 5000:5000 finalproject:Latest
 * Serving Flask app 'frontend'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
```

c-Verify the existence of the image using the command docker image

```
C:\Users\Sara\Downloads\assignment 4>docker images
REPOSITORY                TAG       IMAGE ID       CREATED         SIZE
finalproject              Latest    76d62003aa08   3 minutes ago   139MB
sarahsue82/newtodolist1   latest    085740be313a   3 weeks ago     139MB
newtodolist               latest    71d4671b64c9   3 weeks ago     139MB
sarahsue82/newtodolist    latest    71d4671b64c9   3 weeks ago     139MB
```

3-Push the image to Docker Hub

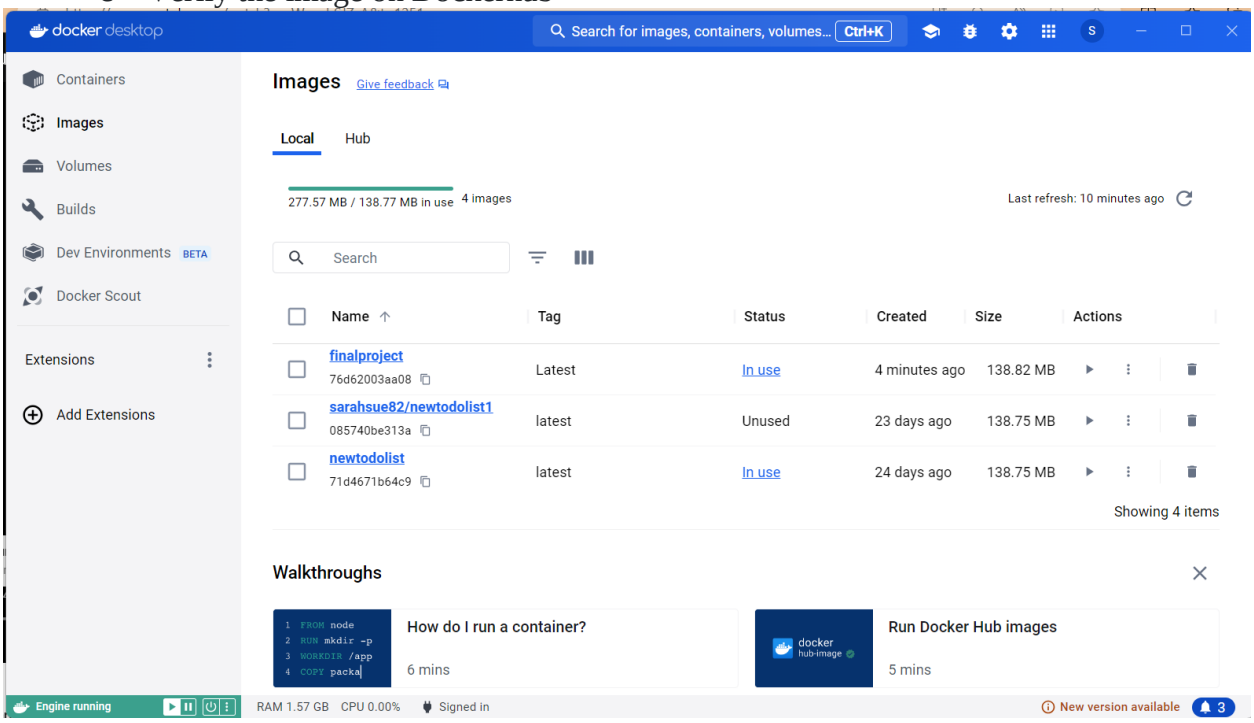
a- Create a new account for Dockerhub or login if you have already an account

```
C:\Users\Sara\Downloads\assignment 4>docker login
Authenticating with existing credentials...
Login Succeeded
```

b- Tag and push the image to Dockerhub

```
C:\Users\Sara\Downloads\assignment 4>docker push finalproject:Latest
The push refers to repository [docker.io/library/finalproject]
e0c17e9a8274: Preparing
eb9b105013eb: Preparing
d32185895c95: Preparing
4fdd425128c4: Preparing
c7a39adf7185: Preparing
269420c335ac: Waiting
950d1c0d353c: Waiting
b60a1f471434: Waiting
8d8e7f754ef8: Waiting
cbb0bec46633: Waiting
7a75d57a5024: Waiting
52ec5a4316fa: Waiting
denied: requested access to the resource is denied
```

c- Verify the image on Dockerhub



4- Use glcoud commands to deploy that Docker image to a cluster on Google Cloud (using Kubernetes)

a-Initialize the SDK

```
C:\Users\Sara\Downloads\assignment 4>gcloud init
Welcome! This command will take you through the configuration of gcloud.

Settings from your current configuration [default] are:
accessibility:
  screen_reader: 'False'
compute:
  zone: us-central1-a
core:
  account: sarahsuegirl@gmail.com
  disable_usage_reporting: 'False'
  project: myproject-681982

Pick configuration to use:
 [1] Re-initialize this configuration [default] with new settings
 [2] Create a new configuration
Please enter your numeric choice:
```

b- Create a new project name finalproject-771944

```
* Commands that require authentication will use sarahsuegirl@gmail.com by default
* Commands will reference project `finalproject-771944` by default
Run `gcloud help config` to learn how to change individual settings

This gcloud configuration is called [default]. You can create additional configurations if you work with multiple accoun
ts and/or projects.
Run `gcloud topic configurations` to learn more.
```

c- Set the time zone configurations and validate it

d- Create a cluster with 3 nodes

```
C:\Users\Sara\Downloads\assignment 4>gcloud container clusters create sarah-final-cluster --num-nodes=3  --zone us-centr
al1-a  --disk-type="pd-ssd"      --disk-size "50"
Default change: VPC-native is the default mode during cluster creation for versions greater than 1.21.0-gke.1500. To create a
dvanced routes based clusters, please pass the `--no-enable-ip-alias` flag
Creating cluster sarah-final-cluster in us-central1-a...|
Creating cluster sarah-final-cluster in us-central1-a... Cluster is being health-checked (master is healthy)...done.
Created [https://container.googleapis.com/v1/projects/finalproject-771944/zones/us-central1-a/clusters/sarah-final-cluster].
To inspect the contents of your cluster, go to: https://console.cloud.google.com/kubernetes/workload_/gcloud/us-central1-a/sa
rah-final-cluster?project=finalproject-771944
kubeconfig entry generated for sarah-final-cluster.
NAME                LOCATION       MASTER_VERSION      MASTER_IP     MACHINE_TYPE  NODE_VERSION        NUM_NODES  STATUS
sarah-final-cluster  us-central1-a  1.28.7-gke.1026000  34.122.58.55  e2-medium     1.28.7-gke.1026000  3          RUNNING

C:\Users\Sara\Downloads\assignment 4>
```

5-Use the image we created and pushed into dockerhub .Pull the image from dockerhub and then deploy into cluster.

```
C:\Users\Sara\Downloads\assignment 4>kubectl create deployment  final-deployment --image finalproject:Latest
deployment.apps/final-deployment created
```

6- create a deployment directly using `kubectl` command without writing a YAML file

And get deployments

```
C:\Users\Sara\Downloads\assignment 4>kubectl get deployments
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
final-deployment   0/1     1            0           44s
```

7-get pods add photo

```
C:\Users\Sara\Downloads\assignment 4>kubectl get pods
NAME                                READY   STATUS             RESTARTS   AGE
final-deployment-66f9ff9f9b-mjmjf   0/1     ImagePullBackOff   0          96s
```

8- expose the deployment If you haven't already exposed your deployment to receive external traffic.

```
C:\Users\Sara>kubectl expose deployment my-deployment --type=LoadBalancer --name=my-service --port=5000
service/my-service exposed
```

9-external ip

```
C:\Users\Sara\Downloads\assignment 4>kubectl get svc
NAME         TYPE           CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
kubernetes   ClusterIP      10.38.32.1    <none>        443/TCP          9m7s
my-service   LoadBalancer   10.38.42.77   <pending>     5000:32618/TCP   30s
```

Source Code:

1-Source code for Frontend

```
#Frontend application

from flask import Flask, request, redirect, url_for, render_template
import requests # Import the requests module

app = Flask(__name__)

@app.route("/")
def show_list():
   resp = requests.get("http://localhost:5001/api/items")  # Use requests.get
   resp = resp.json()
   return render_template('index.html', todolist=resp)

@app.route("/add", methods=['POST'])
def add_entry():
   data = {
      'what_to_do': request.form['what_to_do'],
      'due_date': request.form['due_date']
   }
   resp = requests.post("http://localhost:5001/api/items", json=data)  # Use requests.post
   return redirect(url_for('show_list'))


@app.route("/delete/<x>", methods=['GET'])
def delete_entry(x):
   data = {'what_to_do': x} #request.form['what_to_do']}
   resp = requests.delete("http://localhost:5001/api/items", json=data)  # Use requests.delete with JSON
payload
   return redirect(url_for('show_list'))
   #return render_template('index.html', todolist=resp)


@app.route("/mark/<what_to_do>", methods=['POST'])
def mark_item(what_to_do):
   response = requests.post("http://localhost:5001/api/items/mark", json={'what_to_do': what_to_do})
   return redirect(url_for('show_list'))


if __name__ == '__main__':
   app.run(host='0.0.0.0',port=5000)
```

2-Source code for Backend

```
#API Backend "DATABASE_HANDLING"
from flask import Flask, render_template, redirect, g, request, url_for,jsonify
import sqlite3

DATABASE = 'todolist.db'

app = Flask(__name__)
app.config['DATABASE'] = DATABASE  # Set the DATABASE configuration


@app.route("/api/items")
def get_items():
   print("it is an error")
   db = get_db()
   print("it is a get_db_error")

   cur = db.execute('SELECT what_to_do, due_date, status FROM entries')
   print("it is a query_error")

   entries = cur.fetchall()
   print("it is a fetchall error")
   tdlist = [dict(what_to_do=row[0], due_date=row[1], status=row[2]) for row in entries]
   return jsonify(tdlist)
   print("it is a return error")


@app.route("/api/items", methods=['POST'])
def add_item():
   data = request.json
   db = get_db()
   db.execute('INSERT INTO entries (what_to_do, due_date) VALUES (?, ?)',
         [data['what_to_do'], data['due_date']])
   db.commit()
   return jsonify({'success': True})


@app.route("/api/items", methods=['DELETE'])
def delete_item():
   data = request.json
   what_to_do = data['what_to_do']
```

```
    db = get_db()
    db.execute("DELETE FROM entries WHERE what_to_do = ?", (what_to_do,))
    db.commit()
    return jsonify({'success': True})


@app.route("/api/items/mark", methods=['POST'])
def mark_as_done():
    what_to_do = request.form['what_to_do']
    db = get_db()
    db.execute("UPDATE entries SET status='done' WHERE what_to_do = ?", (what_to_do,))
    db.commit()
    return jsonify({'success': True})




def get_db():
    """Opens a new database connection if there is none yet for the
    current application context.
    """
    if not hasattr(g, 'sqlite_db'):
        g.sqlite_db = sqlite3.connect(app.config['DATABASE'])
    return g.sqlite_db


@app.teardown_appcontext
def close_db(error):
    """Closes the database again at the end of the request."""
    if hasattr(g, 'sqlite_db'):
        g.sqlite_db.close()


if __name__ == '__main__':
    app.run(host='0.0.0.0',port=5001)
```

Source code for HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Todo List Example</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device.width, initial-scale=1">

    <link rel="stylesheet" href="{{ url_for('static',
filename='css/bootstrap.min.css') }}">
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<!-- Include jQuery -->
    <script src="{{ url_for('static', filename='js/bootstrap.min.js')
}}"></script>

    <style>
        #logo-container {
            position: fixed;
            top: 50px;
            left: 10px;
        }

        #logo-container img {
            width: 100px;
            height: auto;
        }

        #google_translate_element {
            margin-top: 20px;
            margin-left: 10px;
        }

        .header-area {
            display: flex;
            align-items: center;
            justify-content: space-between;
```

```
        }

        #descriptive-image-container {
            text-align: center;
            margin: 1rem 0;
        }

        #descriptive-image-container img {
            width: 45%;
            height: auto;
        }
    </style>
</head>

<body style="background-color: powderblue;">
    <div id="logo-container">
        <img src="{{ url_for('static', filename='images/fordham.png') }}"
alt="Logo">
    </div>
    <div id="google_translate_element"></div>

    <h1 style="color: rgba(99, 30, 5, 0.907); text-align: center; font-weight:
bold;">Fordham University</h1>

    <h1 style="color: rgb(243, 114, 8); text-align: center; font-weight:
bold;">To Do List Project</h1>
    <p style="color: rgb(243, 114, 8); text-align: center;">Cloud Computing CISC
5550</p>

    <div class="container">
        <div id="descriptive-image-container">
            <img src="{{ url_for('static', filename='images/todolist.jpg') }}"
alt="Task Icon">
        </div>

        <div class="header-area">
            <h3>Oh, so many things to do...</h3>
        </div>

        <table class='table'>
            {% for entry in todolist %}
            <tr>
                <td {% if entry.status=='done' %} class='done' {% endif %}>{{
entry.what_to_do|safe }}</td>
                <td>{{ entry.due_date|safe }}</td>
```

```
                <td>
                    <button
onclick="location.href='/mark/{{entry.what_to_do|urlencode}}'">mark as done
</button>
                    <button
onclick="location.href='/delete/{{entry.what_to_do|urlencode}}'">delete</button>
                </td>
            </tr>
            {% else %}
            <tr>
                <td>
                    <em>Unbelievable. Nothing to do for now.</em>
                </td>
            </tr>
            {% endfor %}
        </table>
        <button onclick="toggle_entry_form();" id='toggle_button'>add a new
item</button>

        <div class="container">
            <form action="/add" method="POST" id="add-form" style="display:none">
                <div class="row">
                    <div class="col-sm-6">
                        what to do:
                        <input type="text" size="50" name="what_to_do"
value="more homework?" />
                    </div>
                    <div class="col-sm-3">
                        when:
                        <input type="text" name="due_date" value="" />
                    </div>
                    <div class="col-sm-3">
                        <input type="submit" value="save the new item" />
                    </div>
                </div>
            </form>
        </div>
    </div>

    <script>
        // Ensure the document is fully loaded before executing any JavaScript
        $(document).ready(function() {
            // Define the toggle_entry_form function in the global scope
            window.toggle_entry_form = function() {
                // Toggle the visibility of the add-form
```

```
                $('#add-form').toggle();
                // Change the text of the toggle button
                $('#toggle_button').text(function(_, text) {
                    return text === 'add a new item' ? 'cancel the new entry' :
'add a new item';
                });
            };
        });
    </script>

    <!-- Google Translate script -->
    <script>
        function googleTranslateElementInit() {
            new google.translate.TranslateElement({ pageLanguage: 'en', layout:
google.translate.TranslateElement.InlineLayout.SIMPLE },
'google_translate_element');
        }
    </script>
    <script
src="http://translate.google.com/translate_a/element.js?cb=googleTranslateElement
Init"></script>
</body>
</html>
```