URL Shortener Web Application (Basic)

Introduction:

The problem statement was to build a URL shortener web application using Flask and SQLAlchemy. The web application was supposed to have two pages, the homepage and the history page. The homepage was supposed to have a form where the user could enter a URL, and the application would return a shortened URL , The history page was supposed to show all the URLs that the user had shortened in the past.

Approach:

1.Identify the required functionalities of the application

The Functionalities we need to implement in this application are:

- Accepting a user-provided URL and generating a short URL for it
- Redirecting the user to the original URL when they access the short URL
- Displaying a history of all URLs that have been shortened

2.Install required modules

We will be using several modules for this Flask application , including Flask itself , SQLALchemy , Flask-Migrate and validators.

Here is how to install these modules:

```
pip install Flask
pip install Flask_SQLAlchemy
pip install Flask_migrate
pip install validators
```

Once these modules are installed , We need to import them in our Flask application using ‘import’ statement

Here is how to import the modules

```
from flask import Flask, render_template, request, redirect, url_for
from flask_sqlalchemy import SQLAlchemy
from flask_migrate import Migrate
import os
import validators
import string
import random
```

3.Set up the configuration for the SQLALchemy database

We need to set up a configuration for the SQLALchemy database we will be using to store the URLs . We will be using SQLite for this application. We can add the following configuration to our Flask application:

```
basedir = os.path.abspath(os.path.dirname(__file__))
app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///" + os.path.join(basedir, "data.sqlite")
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
```

These configurations tell Flask where to find the database file , and specify that SQLalchemy should track modifications to the database.

Then we need to create the following

- Instance of the SQLAlchemy class
- Instance of the Migrate class
- A model for the URLs table

4.We define the Url model for our database

We defined the name of the table to be “url_table” using the ‘__tablename__’ attribute.

We will define the Url model with three attributes

- ‘Id’: An integer field that serves as the primary key for the table
- ‘original_url’: A string field that stores the original URL entered by the user.
- ‘short_url’ : A string field that stores the generated short URL for the original URL

The '__repr__()' method was also defined to return a string representation of the object for debugging purposes.

After defining the 'Url' model , we can use Flask-Migrate to create the database table for this model. We will need to run the following commands in the terminal:

```
flask db init
flask db migrate -m "create urls table"
flask db upgrade
```

5.Implementing the URL shortening functionality

This function generates a short URL using a combination of lowercase and uppercase alphabets and digits. It generates a random string of length 3 and checks if that string already exists in the database . if the string is already in the database, it will continue generating new random strings until it find a unique one . Finally, it returns the unique short URL.

```
def generate_short_url():
    characters = string.ascii_letters + string.digits
    while True:
        short_url = "".join(random.choice(characters) for i in range(3))
        if not Url.query.filter_by(short_url=short_url).first():
            return short_url
```

6.Defining the Flask routes

Now, we define the Flask routes for our application. The first route is the home_page route , where the user can enter a URL to be shortened

Here we also need a route to handle the form submission when the user submits a URL to be shortened . We can create this route by adding the 'methods' parameter to the decorator and specifying that it accepts 'POST' requests.

The next route is the redirect route, where the user is redirected to the original URl when they enter the shortened URL

7.Create HTML templates

To display the content of our application , we need to create HTML templates. Flask uses the Jinja templating to render templates. We will create thress templates : ‘layout.html’ , ‘home.html’ , ‘history.html’.

The ‘layout.html’ template will serve as a base template for all other templates.It will contain the basic HTML structure , including the header , footer , and common elements .

The ‘{% block %}’ and ‘{% endblock %}’ tags are placeholders for the content that will be inseted into the the template when it is rendered.

The ‘home.html’ template will be rendered for the home page of the application , where the user will enter the URL to be shortened.

The ‘history.html’ template will be rendered for the history page of the application , where the user can view all the URLs they have previously shortened.

Copy Button Feature

We added js code to handle the copy to clipboard functionality when a user clicks the “Copy ” button next to a shortened URL on the History page.

The code uses the ‘navigator.clipboard.writeText()’ method to write thr shortened URL to the clipboard.

BY -

Suryateja Kalapala