Name - Gracy Jain

Project - URL Shortener

1.This section imports the necessary modules and libraries. We import Flask from the Flask framework to create our web application. We also import redirect to handle URL redirection, render_template to render HTML templates, request to handle HTTP requests, string to access string-related operations, random to generate random characters, and sqlite3 to interact with the SQLite database.

In [2]:
```
from flask import Flask, redirect, render_template, request
import string
import random
import sqlite3
```

2.Here, we create a Flask application instance by initializing the Flask class with the **name** module. We also establish a connection to the SQLite database file named urls.db and create a cursor object to execute SQL queries.

In [3]:
```
app = Flask(__name__)
db = sqlite3.connect('urls.db')  # Connect to the SQLite database
cursor = db.cursor()
```

3.This code block executes an SQL query using the cursor to create a table named urls if it doesn't already exist in the database. The table has three columns: id (an auto-incrementing integer primary key), long_url (to store the original URL), and short_url (to store the generated short URL). The db.commit() statement is used to commit the changes to the database.

In [4]:
```
cursor.execute('''CREATE TABLE IF NOT EXISTS urls
                  (id INTEGER PRIMARY KEY AUTOINCREMENT,
                  long_url TEXT,
                  short_url TEXT)''')
db.commit()
```

4.This function generates a random short URL. It creates a string of characters by combining uppercase letters, lowercase letters, and digits. It then selects 6 random characters from this string to form the short URL.

In [5]:
```
def generate_short_url():
    characters = string.ascii_letters + string.digits
    short_url = ''.join(random.choice(characters) for _ in range(6))
    return short_url
```

5.This code block defines the home route / for the Flask application. It handles both GET and POST requests. When a user submits a form on the homepage, the function checks if the request method is POST. If it is, it retrieves the original URL from the form data and generates a short URL. It then inserts the long URL and short URL into the database. Finally, it renders the index.html template with the short URL.

In [7]:
```
@app.route('/', methods=['GET', 'POST'])
def home():
    if request.method == 'POST':
        original_url = request.form['url']
        short_url = generate_short_url()

        cursor.execute("INSERT INTO urls (long_url, short_url) VALUES (?, ?)",
                       (original_url, short_url))
        db.commit()

        return render_template('index.html', short_url=request.host_url + short_url)

    return render_template('index.html')
```

6.This code block defines a route with a dynamic parameter . When a user visits a short URL, the function retrieves the corresponding original URL from the database based on the short URL. If the short URL exists in the database, the function redirects the user to the original URL using redirect(). If the short URL is not found, an "Invalid URL" message is displayed.

In [8]:
```
@app.route('/<short_url>')
def redirect_to_url(short_url):
    cursor.execute("SELECT long_url FROM urls WHERE short_url=?", (short_url,))
    result = cursor.fetchone()
    if result:
        original_url = result[0]
        return redirect(original_url)
    else:
        return 'Invalid URL'
```

7.This line of code checks if the script is being executed directly (as opposed to being imported as a module). If it is the main script, it starts the Flask application using app.run() to run the web server.

In [ ]:
```
if __name__ == '__main__':
    app.run()
```

```
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

In [ ]: