K. J. Somaiya College of Engineering, Mumbai-77

Department of Computer Engineering

Roll No.: 16030720004

Mini Project

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Problem Statement:
Develop a URL Shortner using Python.

Books/ Journals/ Websites referred:

https://docs.python.org/3/library/sqlite3.html
https://flask.palletsprojects.com/en/1.1.x/
https://docs.python.org/3/library/random.html
https://docs.python.org/3/library/string.html

Theory:

Flask (web framework)

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools.

Applications that use the Flask framework include Pinterest and LinkedIn.

Introduction to SQLite in Python

Databases offer numerous functionalities by which one can manage large amounts of information easily over the web and high-volume data input and output over a typical file such as a text file. SQL is a query language and is very popular in databases. Many

websites use MySQL. SQLite is a “light” version that works over syntax very much similar to SQL.

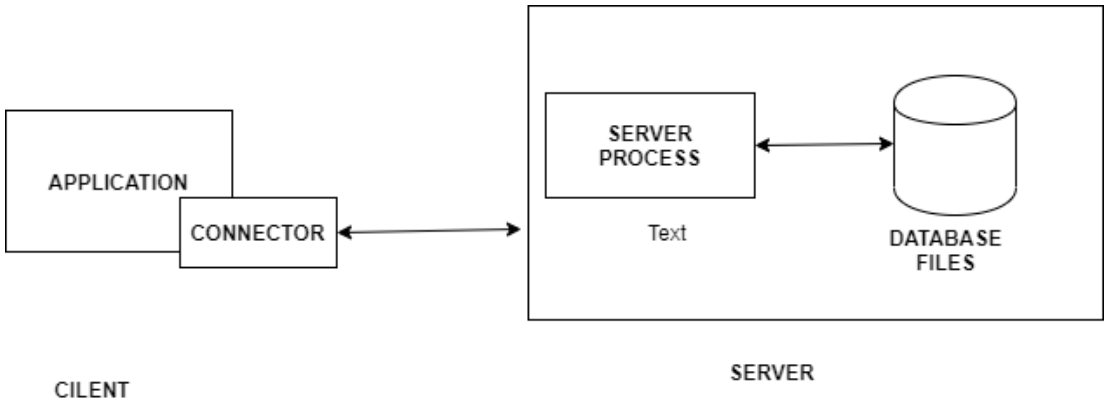
SQLite is a self-contained, high-reliability, embedded, full-featured, public-domain, SQL database engine. It is the most used database engine on the world wide web. Python has a library to access SQLite databases, called sqlite3, intended for working with this database which has been included with Python package since version 2.5.

SQLite has the following features.

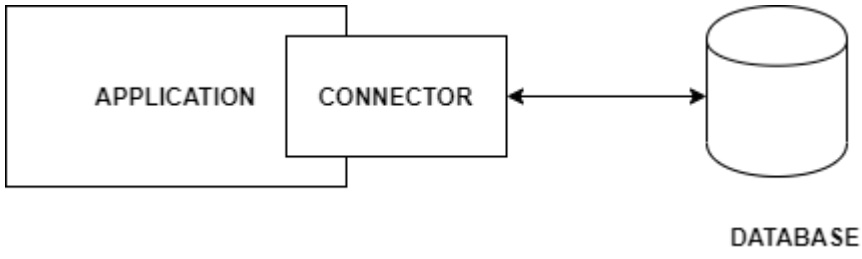
1. Serverless
2. Self-Contained
3. Zero-Configuration
4. Transcational
5. Single-Database

Serverless

Generally, an RDBMS such as MySQL, PostgreSQL, etc., needs a separate server process to operate. The applications that want to access the database server use TCP/IP protocol to send and receive requests and it is called client/server architecture.



SQLite does not require a server to run. SQLite database is joined with the application that accesses the database. SQLite database read and write directly from the database files stored on disk and applications interact with that SQLite database.



Self-Contained

SQLite is self-contained means it does not need any external dependencies like an operating system or external library. This feature of SQLite help especially in embedded devices like iPhones, Android phones, game consoles, handheld media players, etc.

SQLite is developed using ANSI-C. The source code is available as a big sqlite3.c and its header file sqlite3.h. If users want to develop an application that uses SQLite, users just need to drop these files into your project and compile it with your code.

Zero-Configuration

SQLite is zero-configuration means no setup or administration needed. Because of the serverless architecture, you don't need to "install" SQLite before using it. There is no server process that needs to be configured, started, and stopped.

Transcational

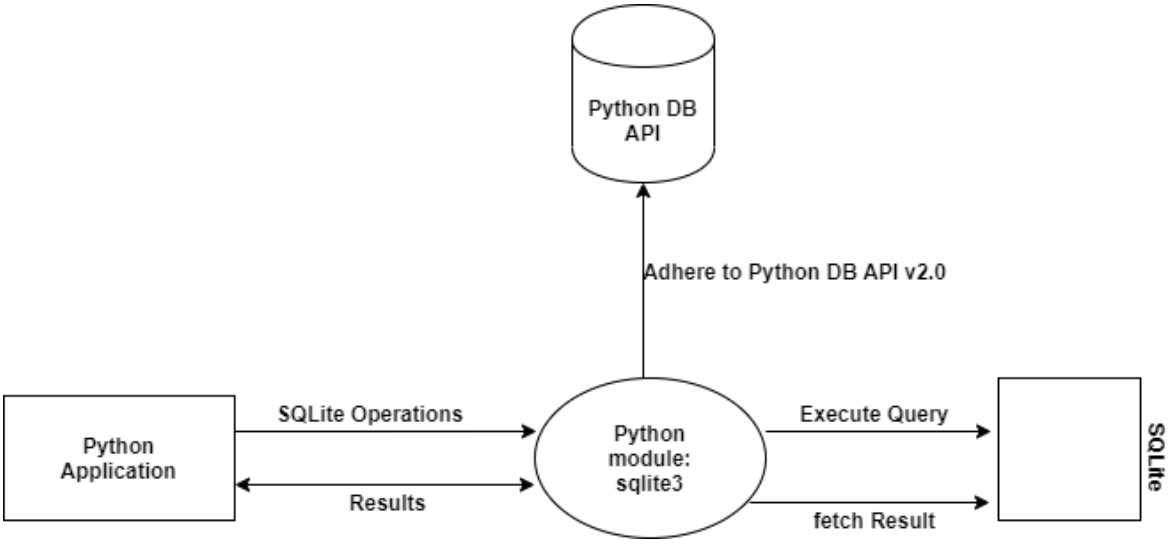
SQLite is Transactional means they are atomic, consistent, isolated, and durable(ACID). All transactions in SQLite are fully ACID-compliant. In other words, all changes within a transaction take place completely or not at all even when an unexpected situation like application crash, power failure, or operating system crash occurs.

Single-Database

SQLite is a single database that means it allows a single database connection to access multiple database files simultaneously. These features bring many nice features like joining tables in different databases or copying data between databases in a single command. SQLite also uses dynamic types for tables. It means you can store any value in any column, regardless of the data type.

Understanding of SQLite Module Working in Python

Python SQLite is used to demonstrate how to develop Python database applications with the SQLite database. You will learn how to perform SQLite database operations from Python. SQLite comes built-in with most of the computers and mobile devices and browsers. Python's official sqlite3 module helps us to work with the SQLite database.



In this diagram, the Python sqlite3 module adheres to Python Database API Specification v2.0 (PEP 249). PEP 249 provides a SQL interface that has been designed to encourage and maintain the similarity between the Python modules that are used to access databases.

Program:

1. **app.py**

```
from flask import Flask ,  redirect , render_template ,request , url_for,fla
sh,session
import random as rand
import string
from database.database_operations import DatabaseOperations

#CONSTANTS
DATABASE_NAME = "url_database.db"
app = Flask(__name__)
app.secret_key = "thisismysecretkeyfornow"

def random_string(size=5):
    chars = string.ascii_letters + string.digits
    random_str = ''.join(rand.choices(chars,k=size))
    return random_str

@app.route("/")
def home():
    return render_template("index.html")

@app.route("/random",methods=["GET","POST"])
def random():
    if request.method == "POST":
        actual_url = request.form["random"]
        short_url = random_string()
        DATABASE_OBJECT = DatabaseOperations(DATABASE_NAME)
        run = DATABASE_OBJECT.check_if_exists(short_url)
        while run:
            short_url = random_string()
            run = DATABASE_OBJECT.check_if_exists(short_url)
        db_ops  = DATABASE_OBJECT.insert_into_database(short_url,actual_url)
        DATABASE_OBJECT.close_connection()
        if db_ops==True:
            return render_template("generate.html",short_url=short_url)
        else:
            return render_template("random.html",error=True)
    else:
        return render_template("random.html")


@app.route("/custom",methods=["GET","POST"])
```

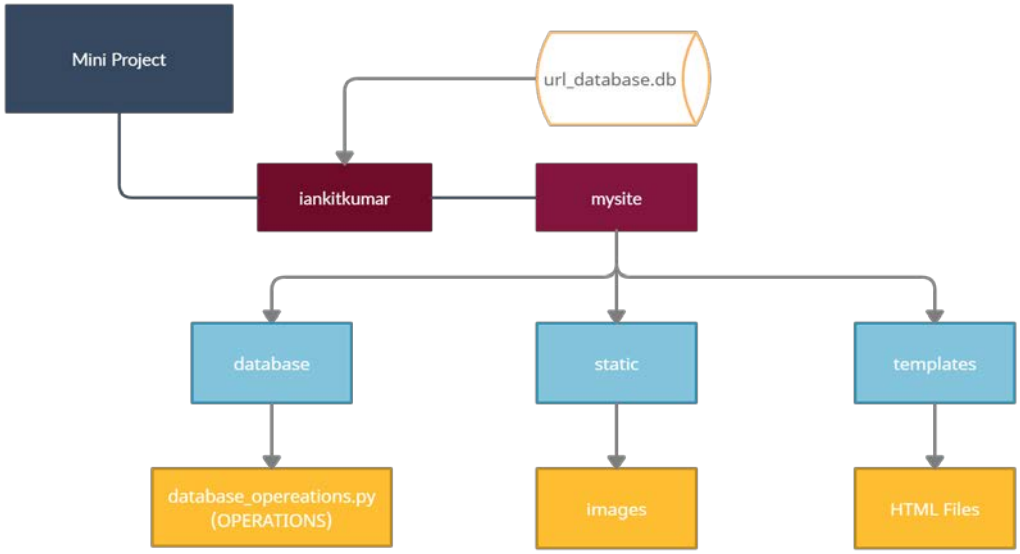
```
def custom():
    if request.method=="POST":
        actual_url = request.form["long"]
        short_url = request.form["short"]
        DATABASE_OBJECT = DatabaseOperations(DATABASE_NAME)
        run = DATABASE_OBJECT.check_if_exists(short_url)
        if run:
            return render_template("custom.html",error=True)
        else:
            DATABASE_OBJECT.insert_into_database(short_url,actual_url)
            DATABASE_OBJECT.close_connection()
            return render_template("generate.html",short_url=short_url)
    else:
        return render_template("custom.html")


@app.route("/<short_url>")
def full_site(short_url):
    # return f"Hello {shorturl}"
    DATABASE_OBJECT = DatabaseOperations(DATABASE_NAME)
    actual_url = DATABASE_OBJECT.get_actual_url(short_url)
    if actual_url:
        actual_url = actual_url[0]
        print("Redirecting...")
        return redirect(actual_url)
    else:
        return redirect(url_for("home"))

if __name__ == "__main__":
    # app = Flask(__main__)
    app.run(debug=True)
```

2. database_operations.py

```
import sqlite3

class DatabaseOperations:
    '''
    Performs Database operation on the given database
    '''
    def __init__(self,database_name):
        '''Initializing the database and its cursors'''
        self.database_name = database_name
        self.connection = sqlite3.connect(self.database_name)
        self.cursor = self.connection.cursor()
```

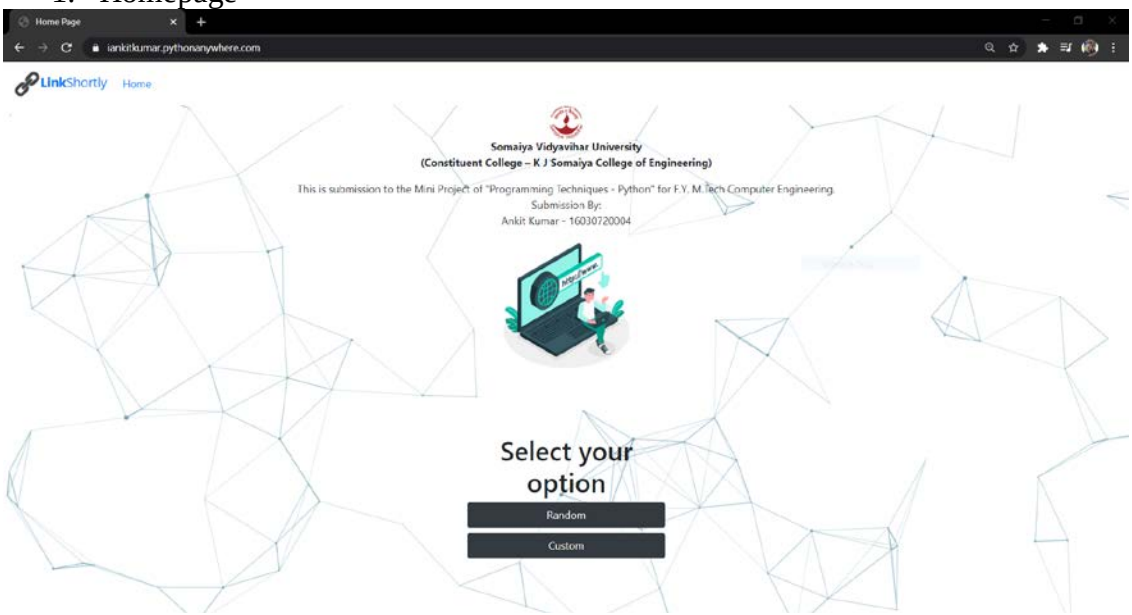
```
        self.cursor.execute("CREATE TABLE IF NOT EXISTS urls(short_url ch
ar(5) PRIMARY KEY NOT NULL,actual_url varchar(500) NOT NULL)")
        self.connection.commit()

    def close_connection(self):
        '''Closes the connection to Database'''
        self.connection.close()

    def check_if_exists(self,short_url):
        '''Returns True if a shortened url exists in DB else returns Fals
e'''
        sql_query='''SELECT * FROM urls where short_url=?'''
        check_tuple = (short_url,)
        self.cursor.execute(sql_query,check_tuple)
        output = self.cursor.fetchone()
        if not output:
            return False
        else:
            return True

    def insert_into_database(self,short_url,actual_url):
        '''Inserts a new row into the table.
        Return True if the insert is successfull else returns False.'''
        try:
            sql_query = '''INSERT INTO urls VALUES(?,?)'''
            input_tuple = (short_url,actual_url)
            self.cursor.execute(sql_query,input_tuple)
            self.connection.commit()
            return True
        except Exception as e:
            print(f"[INSERTION FAILED]: e")
            return False

    def get_actual_url(self,short_url):
        '''Get the actual_url from the short_url from DB.
        Returns False if no URL is found '''
        sql_query = '''SELECT actual_url FROM urls where short_url=?'''
        check_tuple = (short_url,)
        self.cursor.execute(sql_query,check_tuple)
        output = self.cursor.fetchone()
        if not output:
            return False
        else:
            return output
```

Folder Structure:

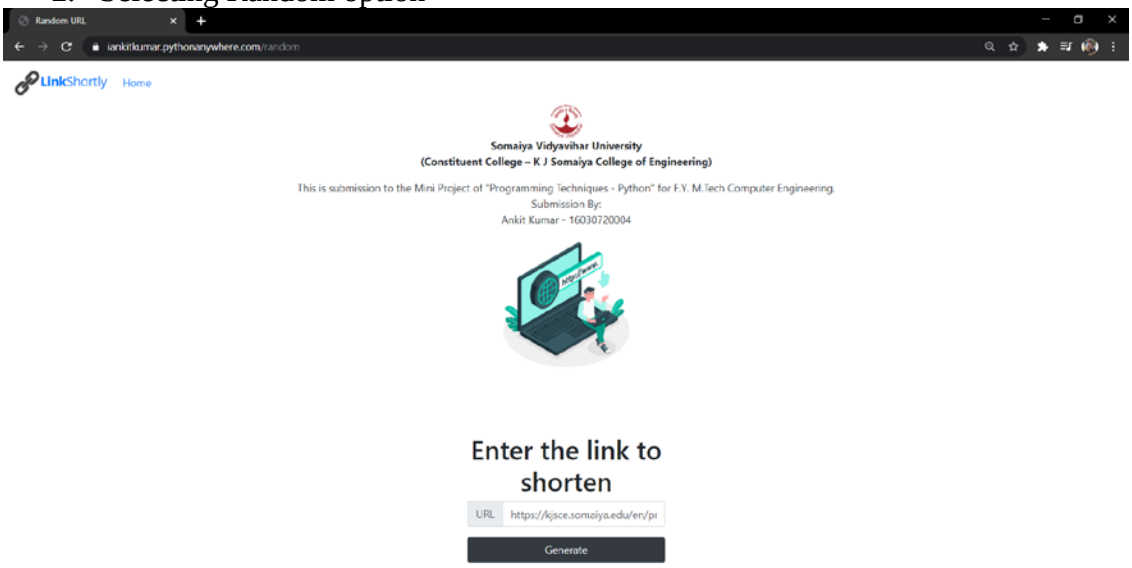


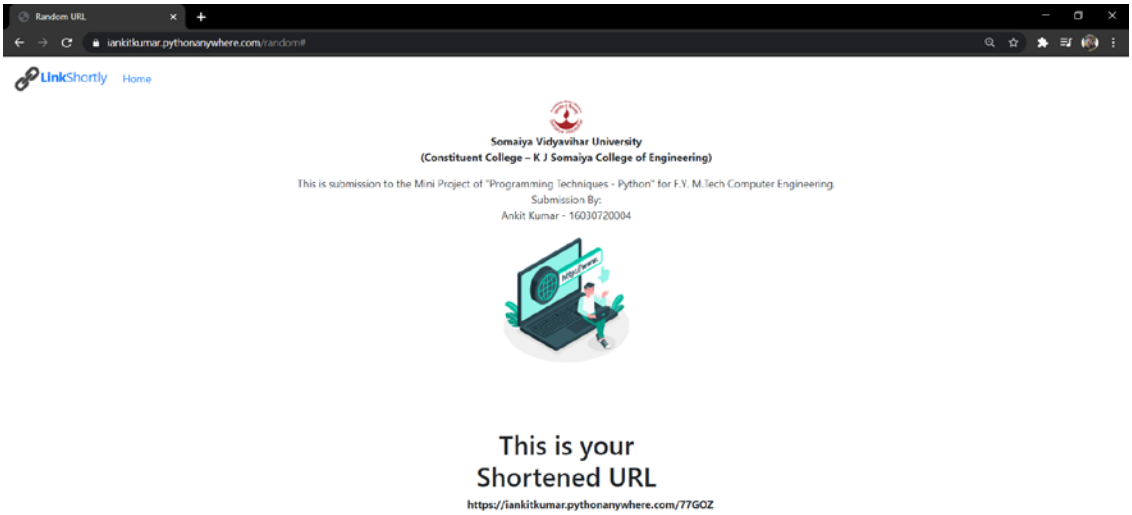
Output:

1. Homepage



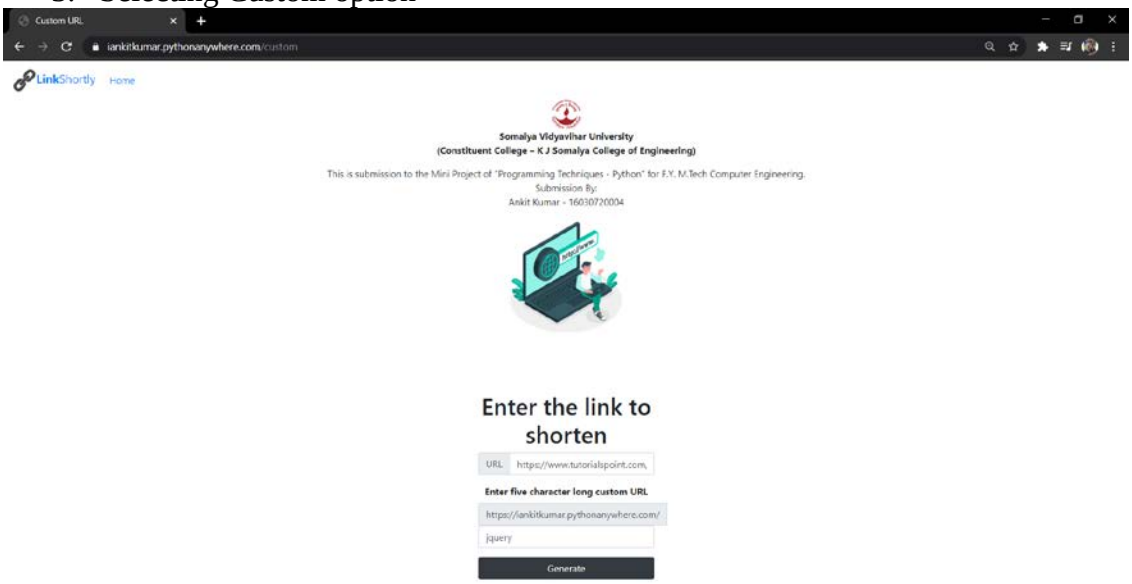
2. Selecting Random option

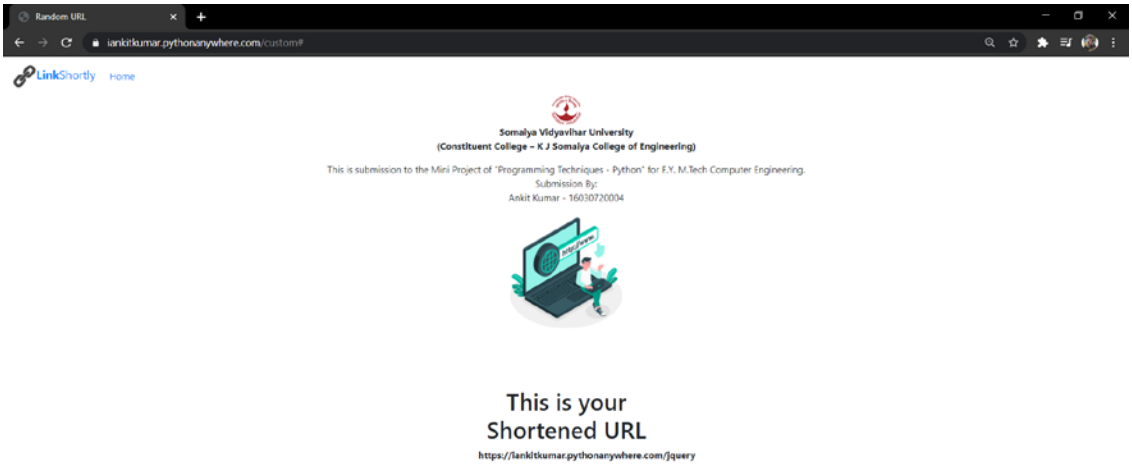




Actual URL: https://kjsce.somaiya.edu/en/programme/master-of-technology-computer-engineering
Short URL: https://iankitkumar.pythonanywhere.com/77GOZ

3. Selecting Custom option





Actual URL: https://www.tutorialspoint.com/jquery/index.htm
Short URL: https://iankitkumar.pythonanywhere.com/jquery

Conclusion: Thus, the mini project of URL Shortner is implemented in python using flask and sqlite3.

Date: 27 February 2021 **Signature of faculty in-charge**