INNOMATICS
RESEARCH LABS

URL Shortener using Flask

PROJECT REPORT

- HRITWIK

SUMMARY:

URL shortening is a technique on the World Wide Web in which a Uniform Resource Locator (URL) may be made substantially shorter and still direct to the required page. This is achieved by using a redirect that links to the web page that has a long URL.
For example, the URL "https://example.com/assets/category_B/subcategory_C/Foo/" can be shortened to "https://example.com/Foo", and the URL "https://en.wikipedia.org/wiki/URL_shortening" can be shortened to "https://w.wiki/U".
Often the redirect domain name is shorter than the original one. A friendly URL may be desired for messaging technologies that limit the number of characters in a message (for example SMS), for reducing the amount of typing required if the reader is copying a URL from a print source, for making it easier for a person to remember, or for the intention of a permalink.

Objective:

This is a beginner project with an objective to allow users to enter a URL that they want to shorten. The URL provided should be a valid URL. If the URL is valid or with a status code of 200, the user will receive a shortened URL which he can use to redirect to the original page. The original URL and the shortened URL will be saved into the database which is displayed on the history page. If the user enters the URL which he has shortened before then they will receive the short URL from the database.

Technology Stack:

1. HTML
2. CSS
3. JavaScript
4. Flask
5. SQLite

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.

SQLite:

SQLite generally follows PostgreSQL syntax. SQLite uses a dynamically and weakly typed SQL syntax that does not guarantee domain integrity. This means that one can, for example, insert a

string into a column defined as an integer. SQLite will attempt to convert data between formats where appropriate, the string "123" into an integer in this case, but does not guarantee such conversions and will store the data as-is if such a conversion is not possible.SQLite is a popular choice as embedded database software for local/client storage in application software such as web browsers. It is arguably the most widely deployed database engine, as it is used today by several widespread browsers, operating systems, and embedded systems (such as mobile phones), among others. SQLite has bindings to many programming languages.

Workflow:

1. Importing libraries and setting up the database

```
from flask import Flask, render_template, request, redirect
import pyperclip as pc
from flask_sqlalchemy import SQLAlchemy
from flask_migrate import Migrate
import os
import string
import random

################################ Setting up SQL database and Flask Object#####################################

app = Flask(__name__)
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
db = SQLAlchemy(app)
migrate = Migrate(app, db)
```

2. Building the Templates
   - Creating the HTML files
   - The Homepage or index.html contains the text field to enter the long URL and display the shortened form of the URL
   - The History page contains all the URLs both long and shortened form

3. Creating the database model

```
class Url(db.Model):
    __tablename__ = 'urls'
    id = db.Column(db.Integer, primary_key = True)
    long = db.Column(db.String())
    short =db.Column(db.String(10))

    def __init__(self, long, short):
        self.long = long
        self.short = "http://127.0.0.1:5000/"+short
```

4. Writing the logic to create the shortened URL:

```
def shorten_url():
    letters = string.ascii_lowercase + string.ascii_uppercase + string.digits
    while True:
        rand_letters = random.choices(letters, k=10)
        rand_letters = "".join(rand_letters)
        short_url = Url.query.filter_by(short=rand_letters).first()
        if not short_url:
            return rand_letters
```

5. Routes

The routes present in the application are:

- Home ‘/’
- Shortened URL ‘/short’
- History ‘/history’

```
@app.route('/')
def home_get():
    return render_template("index.html")

@app.route('/', methods =["POST"])
def home_post():
    long = request.form.get("in_1")
    if long =="":
        return render_template("index.html", empty ="Fill with URL")
    else:
        found_url = Url.query.filter_by(long=long).first()
        if found_url:
            for i in db.session.execute("SELECT short FROM urls WHERE long='{}'".format(long)):
                short =i[0]
                return render_template("shorten.html" , short=short)
        else:
            short = shorten_url()
            URL = Url(long, short)
            db.session.add(URL)
            db.session.commit()
            return render_template("shorten.html" , short="http://127.0.0.1:5000/"+short)
```

```python
@app.route("/short",methods=["POST"])
def copyToClipboard():
    shortenlink=request.form.get("surl")
    pc.copy(shortenlink)
    return render_template("index.html")

@app.route('/history')
def history():
    Hist = db.session.execute("Select * from urls")
    return render_template("history.html", data=Hist)

@app.route("/<short>")
def direct(short):
    for i in db.session.execute("SELECT long FROM urls WHERE short='{}'".format("http://127.0.0.1:5000/"+short)):
        return redirect(i[0])
```

6. Running the application

SUMMARY:

- URL Input: Accept the valid URL from a user and pass it to the URL shortening module to shorten the URL
- URL Shortening: Map the given URL with a maximum of 10 unique alphanumeric characters.
- URL Mapping: Map the short and long URL in the database so that the URL redirection phase can use this information for future use.
- URL Redirection: Concatenate the unique alphanumeric character with the system's domain, to make the shortened URL work.