URL Shortener Web Application (Basic)

1. Introduction

This project is about building a basic URL Shortener Web Application using the Flask framework in Python. The main purpose of this application is to convert long and complex URLs into short and easy-to-share links.

In real-life situations, long URLs are difficult to share and remember, especially on platforms like social media, emails, messaging applications, and online documents. A URL shortener helps to solve this problem by providing a shorter version of the original URL.

This project was developed as part of the backend development module under the Innomatics internship program. The main intention of this project is not only to build a working application but also to understand how backend technologies work together. Through this project, practical experience was gained in Flask routing, handling HTTP requests, using HTML templates, and storing data in a database.

2. Project Objectives

The main objectives of this project are listed below:

- To design and implement a basic URL shortening system using Flask.
- To allow users to enter a long URL and generate a shortened version of that URL.
- To store both the original URL and the shortened URL in a database.
- To provide a history page where users can view all previously shortened URLs.
- To redirect users to the original URL when a shortened URL is accessed.

These objectives ensure that the application is simple, functional, and user-friendly while covering important backend concepts.

3. Technologies Used

The following technologies and tools were used in this project:

- Python: Used as the core programming language for backend logic.
- Flask: A lightweight web framework used for routing, request handling & application structure.
- HTML: Used to create the frontend user interface.
- Jinja2: Flask's template engine used to dynamically display data on HTML pages.
- SQLite: A lightweight database used to store original URLs and shortened URLs.
- Werkzeug: Used internally by Flask for request handling and redirection.

These technologies were chosen because they are simple, beginner-friendly, and suitable for small to medium-scale web applications.

4. Application Architecture

The application follows a simple and organized folder structure, which helps in maintaining clean code and separation of concerns.

app.py: This file contains all the Flask routes, business logic, URL generation logic, database connections, and redirection functionality.

templates folder: This folder contains HTML files such as index.html (home page) and history.html (history page). These files handle the frontend display.

instance folder: This folder contains the SQLite database file (urls.db) where all original and shortened URLs are stored.

By separating backend logic, frontend templates, and database storage, the application becomes easier to understand and manage.

5. Application Workflow (Step-by-Step)

Step 1: The user opens the home page of the URL Shortener application.
Step 2: The home page displays an input field where the user can enter a long URL.
Step 3: The user enters the URL and clicks on the Shorten button.
Step 4: Flask receives the entered URL through a POST request.
Step 5: The application checks whether the entered URL is in a valid format.
Step 6: A unique short code is generated for the URL.
Step 7: The short code is appended to the base URL to create the shortened URL.
Step 8: Both the original URL and the shortened URL are saved into the SQLite database.
Step 9: The shortened URL is displayed on the home page so the user can copy it.
Step 10: The user can click on View History to see all previously shortened URLs.
Step 11: Clicking on any shortened URL redirects the user to the original URL.

This step-by-step flow ensures smooth functioning of the application.

6. Database Design

The application uses SQLite as the database because it is lightweight, easy to configure, and well-suited for small projects.

A single table is used in the database to store the following details:

- Original URL
- Shortened URL

SQLite was selected because it does not require complex setup and integrates easily with Flask applications.

7. URL Validation and Redirection Logic

Before generating a shortened URL, the application verifies whether the entered URL follows a valid format. This helps prevent incorrect or malformed URLs from being stored in the database.

When a user accesses a shortened URL, Flask captures the short code from the URL path. The application then searches the database for a matching record. If a match is found, Flask redirects the user to the corresponding original URL using Flask's redirection functionality.

This logic ensures correct URL mapping and smooth redirection.

8. Challenges Faced and Debugging

During the development of this project, several challenges were encountered:

- Understanding Flask routing and request handling.
- Managing form submissions using POST requests.
- Ensuring that each shortened URL is unique.
- Implementing correct redirection logic.
- Connecting the database and displaying stored data on the history page.

These challenges were resolved through documentation study, debugging, testing, and hands-on experimentation.

9. Learning Outcomes

Through this project, the following learning outcomes were achieved:

- Gained practical experience with Flask backend development.
- Understood how web applications process user input.
- Learned how to integrate a database with a Flask application.
- Improved debugging and problem-solving skills.
- Gained confidence in building complete backend-driven web applications.

10. Conclusion

This project helped in understanding how a simple web application works from start to end using Flask. By working on this URL Shortener application, practical knowledge was gained about handling user input, generating dynamic responses, storing data in a database, and redirecting users based on stored information.

Building this application also improved confidence in backend development, especially in working with Flask routes, HTML templates, and SQLite integration. The project provided hands-on experience in solving real-world problems such as managing long URLs and making them easier to share. Overall, this project was a useful learning experience and created a strong foundation for developing more advanced web applications in the future.