URL Shortener Web Application

Project Report

1. Introduction

In the modern digital world, sharing long URLs can be inconvenient and error-prone. URL Shortener applications solve this problem by converting long URLs into short, easy-to-share links. This project focuses on building a basic URL Shortener Web Application using Flask, Bootstrap, and a database ORM.

The main objective of this project is to design a user-friendly web application that allows users to shorten URLs, store them in a database, and view previously shortened URLs. The project also emphasizes backend logic, frontend integration, data storage, and validation of user input.

2. Project Objectives

The primary objectives of this project are:

- To create a web application that shortens long URLs.
- To store original and shortened URLs in a database.
- To allow users to copy shortened URLs easily.
- To display previously shortened URLs on a history page.
- To validate URLs before processing them.
- To design a clean and responsive user interface using Bootstrap.

3. Technologies Used

The following technologies were used in this project:

Frontend Technologies

- HTML: Used for structuring web pages.
- CSS: Used for styling the interface.

- Bootstrap: Used for responsive design and UI components.

Backend Technologies

- Python: Programming language used for backend logic.
- Flask: Lightweight web framework used for routing and request handling.

Database and ORM

- SQLite: Used as the database for storing URL data.
- SQLAlchemy ORM: Used to interact with the database using Python objects instead of SQL queries.

4. System Architecture

The application follows a simple client-server architecture:

- The frontend interacts with users and sends requests to the backend.
- Flask processes the requests and performs URL shortening logic.
- SQLAlchemy stores and retrieves data from the SQLite database.
- The server sends processed data back to the frontend for display.

This architecture ensures separation of concerns between the user interface, business logic, and data storage.

5. Application Workflow

The working of the application can be explained in the following steps:

1. The user opens the home page of the application.
2. The user enters a long URL and selects a custom short URL length.
3. When the user clicks the “Shorten URL” button, the form data is sent to the Flask backend.
4. The backend validates whether the entered URL is valid.
5. If the URL is valid, a random short code is generated.
6. The original URL and the generated short URL are stored in the database.
7. The shortened URL is displayed on the web page.
8. The user can copy the shortened URL using the copy button.

9. When the user opens the shortened URL, the application redirects to the original website.
10. The history page displays all previously shortened URLs.

6. URL Validation

URL validation is an important feature of this project. It ensures that only valid URLs are processed and stored in the database. This was implemented using a validation library that checks the format of the entered URL.

If the user enters an invalid URL, the system displays an error message and prevents the shortening process. This improves the reliability and security of the application.

7. Database Design

The application uses a single database table to store URL information. The table contains the following fields:

- ID: Unique identifier for each record.
- Original URL: Stores the long URL entered by the user.
- Short URL: Stores the generated short code.

The use of SQLAlchemy ORM makes database operations easier by allowing interaction through Python classes and objects instead of raw SQL queries.

8. Frontend Design

The frontend of the application is designed using Bootstrap to ensure responsiveness and better user experience. The home page contains:

- Input field for entering the original URL.
- Input field for selecting custom short URL length.
- Shorten URL button to generate the short link.
- Output field to display the shortened URL.
- Copy button to copy the shortened URL.

- Navigation link to access the history page.

The history page displays all stored URLs in a tabular format, making it easy for users to view their previously shortened links.

9. Copy Button Functionality

A JavaScript-based copy button is implemented to allow users to copy the shortened URL with a single click. This feature improves usability and reduces manual effort. The Clipboard API is used to copy the text directly from the input field.

10. Custom Short URL Length Feature

The application allows users to choose the length of the short URL. This feature provides flexibility and control to users. The backend dynamically generates random short codes based on the selected length using alphanumeric characters.

11. Challenges Faced

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

- Integrate frontend and backend components properly.
- Handling database connections and ORM configurations.
- Validating user input effectively.
- Preventing errors related to duplicate and invalid URLs.
- Managing routing and redirection logic.

These challenges were resolved through debugging, testing, and proper implementation of Flask and SQLAlchemy features.

12. Testing and Results

The application was tested using different types of URLs including valid and invalid inputs. The following results were observed:

- Valid URLs were shortened successfully.
- Invalid URLs were rejected with proper error messages.
- All shortened URLs were stored in the database.
- The history page displayed all saved links correctly.
- Redirection from short URL to original URL worked without errors.

This confirms that the application meets the functional requirements.