URL Shortener Web Application

Project Report

1. Introduction

In today's digital era, sharing information quickly and efficiently has become extremely important. URLs (Uniform Resource Locators) play a crucial role in accessing online resources, but many URLs are long, complex, and inconvenient to share. Long URLs are difficult to remember, prone to errors when copied manually, and often look unprofessional when shared through messages, emails, or social media platforms.

A URL Shortener addresses this problem by converting a long URL into a short, unique, and easy-to-share link. When a user accesses the shortened link, they are automatically redirected to the original URL. Popular platforms like Bitly and TinyURL provide such services, which inspired the development of this project.

The goal of this project is to design and implement a **basic URL Shortener Web Application** using **Flask for backend development**, **HTML/CSS/Bootstrap for frontend**, and **SQLAlchemy ORM for database management**. This project focuses on understanding backend workflows, database interaction using ORM, URL validation, and integration of frontend with backend logic.

2. Problem Statement

The main problem addressed in this project is the inconvenience of handling long URLs. Users often face challenges when sharing lengthy URLs, including formatting issues, broken links, and poor readability.

The objectives of the project are:

1. To allow users to input a long URL and generate a shortened version.
2. To store both original and shortened URLs in a database.
3. To provide a history page where users can view previously shortened URLs.
4. To validate user input to ensure only valid URLs are shortened.
5. To implement redirection functionality from short URL to original URL.
6. To use SQLAlchemy ORM for clean and efficient database interaction.

3. System Architecture and Design

The application follows a **three-layer architecture**:

3.1 Frontend Layer

The frontend is built using HTML and CSS, with optional Bootstrap styling. It contains two main pages:

- **Home Page**: Allows users to enter a URL and generate a shortened link.

- **History Page**: Displays all previously shortened URLs along with their original URLs.

The frontend communicates with the backend using HTTP requests (GET and POST).

3.2 Backend Layer

The backend is implemented using Flask, a lightweight Python web framework. Flask handles:

- Routing
- Request processing
- URL validation
- Short URL generation
- Redirection logic

3.3 Database Layer

The database layer uses **SQLite** with **SQLAlchemy ORM**. ORM abstracts raw SQL queries and allows interaction with the database using Python objects, making the code more readable and maintainable.

4. Technology Stack Used

- **Programming Language**: Python
- **Backend Framework**: Flask
- **Frontend**: HTML, CSS (Bootstrap-ready)
- **Database**: SQLite
- **ORM**: SQLAlchemy
- **Development Environment**: VS Code
- **ORM Demonstration**: Jupyter Notebook (.ipynb)

5. Database Design

The database consists of a single table named URL.

Table: URL

Column Name	Data Type	Description
id	Integer	Primary Key
original_url	String	Stores the original long URL
short_code	String	Stores the generated short code

Each entry in the table represents a mapping between an original URL and its shortened version.

6. Approach to Solution

6.1 URL Validation

Before shortening a URL, it is important to validate user input to avoid storing invalid or malformed URLs. For this purpose, the urlparse module from Python's standard library was used.

The validation checks:

- Whether the URL has a valid scheme (http or https)
- Whether the URL has a network location (domain name)

This ensures that only valid URLs are processed and stored.

6.2 Short URL Generation

A short code is generated using:

- Random selection of alphanumeric characters
- Fixed-length short codes (6 characters)

This approach provides:

- Uniqueness
- Readability
- Scalability for small to medium datasets

Each generated short code is stored in the database along with the original URL.

6.3 Backend Logic Flow

1. User submits a URL via the Home page.
2. Flask receives the POST request.
3. URL validation is performed.
4. If valid:
    - A short code is generated.
    - Data is stored in the database.
    - The shortened URL is displayed.
5. If invalid:
    - An error message is shown.
6. When a short URL is accessed:

- Flask looks up the short code in the database.
- Redirects the user to the original URL.

6.4 History Page Implementation

The History page retrieves all records from the database and displays:

- Original URL
- Corresponding shortened URL

This feature allows users to track and reuse previously shortened links.

7. ORM Implementation Using SQLAlchemy

SQLAlchemy ORM was used to:

- Define database models
- Insert new records
- Query existing records

The ORM implementation was also demonstrated using a Jupyter Notebook (orm_sqlalchemy.ipynb), where database operations such as adding and retrieving URLs were executed interactively. This helped in understanding ORM workflows independently of the Flask application.

8. Testing Strategy

The application was tested using multiple test cases:

- Valid URLs
- Invalid URLs
- Duplicate URLs
- Long URLs
- Redirection functionality
- Database persistence after server restart

Testing ensured that the application behaves correctly under different scenarios and handles errors gracefully.

9. Challenges Faced

9.1 URL Validation

Handling different URL formats was challenging. Using Python's urlparse helped simplify this problem.

9.2 Database Integration

Ensuring that SQLAlchemy ORM was properly initialized with Flask required careful configuration of the application context.

9.3 Unique Short Codes

Although random generation works for small-scale applications, ensuring uniqueness is critical. For this basic project, collisions are unlikely but can be handled in future enhancements.

10. Future Enhancements

The following improvements can be made:

- User authentication
- Click count tracking
- Expiration time for short URLs
- Improved UI using Bootstrap
- REST API support
- Deployment on cloud platforms

11. Conclusion

This project successfully demonstrates the development of a basic URL Shortener Web Application using Flask and SQLAlchemy ORM. It covers essential backend concepts such as routing, validation, database interaction, and redirection. The project also emphasizes clean code structure, ORM usage, and user-friendly design.

Through this project, a strong foundation in Flask backend development and ORM-based database management was established. The application meets all the stated objectives and serves as a solid base for further enhancements and real-world deployment.