Project Report

URL Shortener Web Application (Advanced Version)

Dharanikota Teja Swaroop

1. Introduction

In the modern digital world, the internet plays an important role in communication and information sharing. Every day, users share web links for articles, videos, online tools, documents, and social media posts. Many of these links are long and complex, containing multiple parameters and tracking codes. Such long URLs are difficult to remember, inconvenient to share, and prone to copying errors.

Sometimes, users face problems when sending long links through messaging applications or emails because they may break or appear confusing. Even a small mistake in copying a link can make it invalid. This creates inconvenience and reduces productivity. To solve this problem, URL shortener applications are commonly used.

A URL shortener converts a long web address into a shorter and easy-to-share link. When the shortened link is accessed, it automatically redirects the user to the original URL. Popular platforms such as Bitly and TinyURL work on the same principle.

The objective of this project is to design and develop an advanced URL Shortener Web Application using Python Flask, SQLAlchemy ORM, and SQLite database. In addition to basic URL shortening, this application also provides user authentication, allowing users to create accounts, log in securely, and manage their own URL history. The project also focuses on building an interactive user interface with copy buttons, animations, and feedback messages.

This project helped in understanding how frontend and backend systems work together, how databases store persistent information, and how security features such as authentication are implemented in real-world applications.

2. Objectives of the Project

The main objectives of this project are:

- To create a web application that shortens long URLs.
- To allow users to register and log in securely.
- To store original URLs and shortened URLs in a database.
- To display user-specific URL history.
- To validate URLs before storing them.
- To provide an interactive and user-friendly interface.
- To understand Flask routing, database integration, and session management.

3. Technologies Used

3.1 Frontend

HTML is used to design the structure of the web pages. CSS is used to enhance the visual appearance by adding layouts, animations, colors, and spacing. JavaScript is used to add interactivity such as copy-to-clipboard functionality and toast notifications. The frontend provides a clean and modern user interface.

3.2 Backend

The backend is developed using Python Flask framework. Flask handles routing, form processing, authentication logic, URL generation, and redirection. Flask-Login is used to manage user sessions and secure routes.

3.3 Database and ORM

SQLite is used as the database because it is lightweight and easy to configure. SQLAlchemy ORM allows interaction with the database using Python objects instead of raw SQL queries, making development simpler and safer.

4. System Overview

The application follows a client-server architecture. The browser acts as the client, sending requests to the Flask server. The server processes the requests, interacts with the database, and returns responses to the client.

The system consists of three major layers:

- Presentation Layer – HTML, CSS, and JavaScript.
- Application Layer – Flask backend logic.
- Data Layer – SQLite database.

Each layer is independent, making the application modular and maintainable.

5. Application Workflow

The workflow of the application is as follows:

1. The user opens the web application.
2. The user signs up or logs in.
3. After authentication, the dashboard is displayed.
4. The user enters a long URL and clicks the shorten button.
5. The system validates the URL.
6. A unique short code is generated.
7. The URL mapping is saved in the database.
8. The shortened URL is displayed with a copy button.
9. The user can view previously shortened URLs in the history page.
10. Clicking a short URL redirects to the original website.

6. Implementation Details

6.1 User Authentication

Users can create accounts using a username and password. Passwords are securely stored using hashing techniques. Flask-Login manages user sessions and restricts access to protected pages.

6.2 URL Shortening Logic

Random alphanumeric characters are generated to create a short code. This code uniquely identifies the original URL in the database.

6.3 URL Validation

The validators library is used to verify whether the entered URL is valid. Invalid URLs are rejected and error messages are displayed.

6.4 Database Storage

Each shortened URL is stored with its corresponding user ID, allowing personalized history.

6.5 Frontend Interactivity

JavaScript enables copy button functionality and toast notifications. CSS animations improve visual experience.

7. Testing and Results

The application was tested using multiple scenarios including valid inputs, invalid inputs, authentication testing, and redirection testing. All test cases passed successfully.

8. Challenges Faced

Challenges included integrating authentication, managing sessions, handling database relationships, and improving UI responsiveness. These challenges improved debugging and design skills.

9. Learning Outcomes

This project enhanced knowledge of Flask development, ORM usage, frontend interactivity, security practices, and full-stack workflows.

10. Future Enhancements

Future improvements include role-based access, analytics tracking, QR code generation, mobile optimization, and cloud deployment.

11. Conclusion

The Advanced URL Shortener Web Application successfully meets its objectives. It provides secure authentication, efficient URL management, and an interactive user experience. The project demonstrates real-world web application development skills.