PROJECT REPORT

Design and Development of a URL Shortening Web Application

1. Introduction

The rapid expansion of internet-based services and digital communication platforms has resulted in the frequent use of long and complex Uniform Resource Locators (URLs). Such URLs are commonly generated by search engines, cloud services, and dynamically generated web pages. Although functional, long URLs are difficult to remember, inconvenient to share, and visually unappealing, particularly on platforms such as social media, emails, and instant messaging applications. In some cases, long URLs may even break or become unusable when shared across systems with character limitations.

URL shortening services address these challenges by converting lengthy URLs into short, compact links that redirect users to the original web resource. These shortened links improve usability, enhance readability, and simplify link sharing. Well-known URL shortening platforms demonstrate how such systems can significantly improve user experience while maintaining reliability.

This project focuses on the design and development of a URL Shortening Web Application using the Flask web framework, SQLAlchemy Object Relational Mapper (ORM), and SQLite database. The application allows users to submit valid URLs, generates unique shortened links, stores mappings persistently, and redirects users seamlessly. A dedicated history page further enhances usability by displaying all previously shortened URLs in a structured format.

2. Problem Statement

The objective of this project is to develop a simple, efficient, and user-friendly web-based URL shortening system that does not require user authentication while ensuring correctness and reliability. Many existing systems either lack proper input validation or impose unnecessary complexity for basic usage.

3. Technology Stack

The technology stack was carefully selected to balance simplicity, scalability, and ease of implementation. The frontend of the application was developed using HTML5 for structural markup and CSS3 for styling. Bootstrap 5 was integrated to ensure responsive design across different devices, while JavaScript and the Clipboard API were used to enable seamless copying of shortened URLs.

On the backend, Python was chosen as the primary programming language due to its readability and extensive ecosystem. The Flask micro web framework was used to handle routing, request processing, and server-side logic. SQLite was selected as the database due to its lightweight nature, making it ideal for mini-projects and local deployment. SQLAlchemy ORM was used to abstract database operations, allowing Python objects to represent database records and reducing direct SQL dependency.

4. System Architecture

The application follows a client–server architecture model. The client, typically a web browser, interacts with the server by sending HTTP requests. When a user submits a URL through the input form, the request is processed by the Flask server. The server validates the input, stores the URL mapping in the database, and generates a unique short code.

Upon successful processing, the shortened URL is returned to the client and displayed on the user interface. When a shortened URL is accessed, the server retrieves the corresponding original URL from the database and performs an HTTP redirection. This layered architecture ensures separation of concerns, improved maintainability, and ease of future enhancements.

In addition to its core functionality, the application incorporates security and scalability considerations. Input validation and error handling mechanisms safeguard against malicious or invalid requests, while database indexing ensures efficient retrieval of URL mappings. The modular design allows integration with analytics tools to track usage patterns, enabling administrators to monitor performance and optimize resources.

5. Approach and Methodology

The project was developed using a modular and incremental approach. The application structure clearly separates backend logic, database models, and frontend templates. This separation improves code readability and simplifies debugging and maintenance. Flask routes were designed to distinguish between GET requests for rendering pages and POST requests for handling form submissions.

URL validation was implemented as a critical component of the system. In the first stage, a regular expression was used to verify that the input URL follows a valid format and begins with either http:// or https://. In the second stage, an HTTP HEAD request was sent to check the reachability of the URL, ensuring that the provided link exists and responds correctly.

Short URL generation was achieved using randomly generated alphanumeric strings of fixed length. To avoid collisions, each generated short code was checked against existing database entries. If a collision was detected, a new code was generated. This strategy balances simplicity with reliability and minimizes the probability of duplicate short URLs.

6. Database Design

The database consists of a single table that stores mappings between original URLs and their corresponding short codes. Each record contains a unique identifier, the original URL, and the generated short code. SQLAlchemy ORM was used to define the database schema and perform Create, Read, Insert, and Query operations efficiently.

To enhance reliability and scalability, the database design can be extended with indexing and constraints. Unique constraints on short codes prevent duplication, while indexing on frequently queried fields such as the short code ensures faster lookups during redirection. The schema can also be adapted to include metadata like creation timestamp, expiration date, and access count, enabling features such as analytics, link expiration, and usage monitoring. With SQLAlchemy’s ORM capabilities, these enhancements remain abstracted from raw SQL, making the system easier to maintain, extend, and integrate with other services.

7. History Page Implementation

A dedicated history page was implemented to display all stored URLs in a structured and readable format. The page uses a Bootstrap-styled table to list original URLs alongside their shortened versions. Each shortened URL is clickable, allowing users to verify the redirection functionality. This feature enhances transparency and helps users track previously generated links.

8. User Interface Design

The user interface was designed with a focus on simplicity, clarity, and modern aesthetics. Bootstrap 5 was used extensively to ensure responsiveness and consistency across devices. A glassmorphism design approach was adopted, incorporating translucent cards, subtle blur effects, and gradient backgrounds to create a professional and visually appealing experience.

9. Error Handling and Testing

The application provides clear and meaningful error messages for various failure scenarios, including empty submissions, invalid URL formats, unreachable URLs, and invalid short links. Bootstrap alerts were used to display feedback without disrupting user experience. Comprehensive testing was performed using valid URLs, invalid inputs, non-existent domains, and duplicate submissions to verify system robustness.

10. Limitations and Future Enhancements

Although the application meets its objectives, it has certain limitations. Data is stored locally using SQLite, and features such as user authentication, URL expiration, analytics, and click tracking are not implemented. Future enhancements may include a login system, custom short URLs, click count tracking, administrative dashboards, and deployment on cloud platforms.

11. Conclusion

This project successfully demonstrates the design and development of a full-stack web application using Flask and SQLAlchemy. By following a structured development methodology, the application achieves reliability, usability, and maintainability. The project provides valuable insights into web application architecture, input validation, database integration, and responsive user interface design, making it well suited for academic evaluation and practical learning.