URL Shortener Web Application (Basic) - Project Report

Tech Stack: HTML, CSS, Bootstrap, Flask (Python), SQLAlchemy ORM, SQLite

This report describes the complete approach followed to design and implement a basic URL Shortener web application. The application shortens long URLs, stores them in a database, and displays previously shortened URLs on a History page.

1. Project Overview

A URL Shortener converts a long web address into a short, shareable link. When a user opens the short link, they are redirected to the original URL. This project implements a basic URL shortener using Flask as the backend framework, SQLAlchemy as the ORM layer, and SQLite as the database. A Bootstrap-based frontend provides a clean and responsive interface.

2. Problem Statement

Long URLs can be difficult to share and may look cluttered in messages, documents, or social media posts. Manually copying long links can lead to errors. The goal of this project is to create an application that generates a short URL for any valid long URL and preserves a history of previously shortened links.

3. Objectives and Requirements

1. Accept a long URL input from the user on the Home page.
2. Validate whether the entered URL is correct before processing.
3. Generate a unique short code and create a shortened URL.
4. Save the original URL and shortened URL details into a SQLite database.
5. Display all previously saved URLs on a History page.
6. Provide a copy option so users can copy the shortened URL easily.
7. Redirect users from the shortened URL to the original URL.

4. Technology Stack

Frontend: HTML, CSS, Bootstrap for responsive layout and styling.

Backend: Flask (Python) for routing, request handling, and server-side logic.

ORM: SQLAlchemy for defining models and interacting with the database using Python objects.

Database: SQLite for lightweight local storage of URL records.

Validation: validators library for verifying URL format before saving.

5. Approach and Development Process

The project was implemented in stages to ensure correctness and clarity. The workflow started with creating the user interface, followed by database modeling, then backend routing, and finally validation and testing.

5.1 UI Design (Frontend)

Two pages were created using Bootstrap components:

- **Home Page:** A form to enter the long URL, a button to shorten it, and an output field that shows the shortened URL along with a copy button.
- **History Page:** A table view showing all original URLs along with their shortened URLs and created timestamps.

5.2 Database Setup (SQLite + SQLAlchemy)

SQLite was selected because it is lightweight and does not require external server setup. SQLAlchemy ORM was used to define a URL model and perform database operations such as insert and fetch.

5.3 Short Code Generation

A short code is generated using a random combination of letters and digits. The system ensures uniqueness by checking the database before saving. This prevents duplicate short codes and ensures each shortened URL remains valid and reliable.

5.4 URL Validation and User Input Handling

To avoid broken links and invalid database entries, the application validates input URLs. If the user enters a URL without a scheme (http or https), the system automatically prefixes https://. If validation fails, an error message is displayed and the URL is not saved.

6. System Design and Routes

6.1 Database Model

The URL table stores the following fields:

- **id** (Primary Key): Unique row identifier.
- **original_url**: The full long URL provided by the user.
- **short_code**: The generated short identifier stored for redirection.
- **created_at**: Timestamp for when the URL was shortened.

6.2 Flask Routes

- **/** (GET, POST): Displays the Home page and processes URL shortening requests.
- **/history** (GET): Displays previously saved URLs from the database.
- ***/short_code*** (GET): Redirects to the corresponding original URL if the short code exists.

7. Detailed Workflow

1 User opens the Home page and enters a long URL.

2 Application validates the input URL and auto-adds https:// if required.

3 A unique short code is generated.

4 The original URL and short code are saved in SQLite.

5 The shortened URL is displayed to the user with a copy button.

6 User can visit the History page to see all saved URLs.

7 Opening the shortened URL redirects to the original URL.

8. Examples and Test Cases (6+)

The following examples were used to test the application. These cases confirm URL validation, shortening, database saving, and redirection behavior.

8.1 Valid URL Inputs (Should Shorten Successfully)

- https://google.com
- https://github.com
- https://www.youtube.com/results?search_query=flask+url+shortener
- https://www.amazon.in/s?k=wireless+mouse&ref;=nb_sb_noss
- https://www.wikipedia.org/wiki/URL_shortening
- https://in.linkedin.com/

Expected Result:

A short URL is generated and shown on the Home page. The entry is stored in the database and appears on the History page. Opening the short URL redirects to the original URL.

8.2 Inputs Without Scheme (Auto-fix Feature)

- google.com
- github.com
- www.youtube.com

Expected Result: The application prefixes https:// automatically and proceeds normally.

8.3 Invalid Inputs (Should Show Error and Not Save)

- hello123
- htp://google.com
- 12345
- www..com

Expected Result: A validation error is displayed and no record is inserted into the database.

8.4 Duplicate URL (Should Reuse Existing Short Link)

Example: Submit https://google.com multiple times.

Expected Result: The application returns the existing short URL instead of creating duplicate database entries.

9. Challenges and Solutions

- **Redirect failures for inputs without scheme:** Fixed by auto-adding https:// when missing.
- **Duplicate records for repeated URLs:** Fixed by checking the database for an existing original URL before inserting.

- **Invalid URL strings entering the database:** Fixed by applying URL validation and blocking invalid submissions.

10. Results

The completed application successfully shortens URLs, stores them using SQLite, provides a History page for stored links, supports one-click copying, and redirects short links to the correct original URLs. The project meets all basic requirements including URL validation and database persistence.

11. Future Enhancements

- User accounts (login/signup) to store history per user.
- Click analytics (track number of visits per short URL).
- Custom short codes chosen by the user.
- Expiry date for short URLs.
- Admin dashboard for URL management.

12. Submission Contents

The final submission.zip includes the complete working project code (Flask application, templates, and static files) along with this report.pdf.