Project Report: URL Shortener Web Application

Submitted by: Arshad Murtaza   **Date:** 21-01-2026

---

1. Introduction The objective of this project was to develop a URL Shortener Web Application using Python and Flask. URL shorteners are essential tools in the modern web that convert long, cumbersome URLs into manageable, short links. This application allows users to shorten URLs, automatically saves them to a database, and provides a history of previously shortened links.

2. Technology Stack The project was built using the following technologies:

- **Backend:** Python (Flask Framework) was used for routing and server-side logic.
- **Database:** SQLite was used as the database engine due to its simplicity and zero-configuration requirement. SQLAlchemy (ORM) was used to interact with the database using Python classes instead of raw SQL queries.
- **Frontend:** HTML5 and Bootstrap 5 were used to create a responsive, clean, and user-friendly interface.

3. Database Schema The application uses a single table named `Url`. The schema is designed as follows:

- `id`: Integer, Primary Key. Unique identifier for each entry.
- `original_url`: String. Stores the long URL provided by the user.
- `short_url`: String. Stores the unique 5-character string generated by the app.

4. Implementation Details

4.1. URL Generation Logic One of the core challenges was creating a unique short ID. I implemented a function `generate_short_id()` using Python's `random` and `string` libraries.

- The function selects 5 random characters from ASCII letters and digits.
- *Verification:* Before saving, the application checks the database to ensure the generated code does not already exist, preventing duplicates.

4.2. Routing and Views The Flask application is divided into three main routes:

- **Home Route (`/`):** Accepts both GET and POST requests. On GET, it renders the input form. On POST, it validates the URL , generates a short code, saves it to the database, and returns the shortened link to the user.
- **History Route (`/history`):** Queries the database for all records (`Url.query.all()`) and passes them to the template to be displayed in a table.
- **Redirection Route (`/<short_code>`):** This dynamic route captures the short code from the URL, looks it up in the database, and redirects the user to the original URL using Flask's `redirect` function.

5. Frontend Design I used **Bootstrap** to ensure the application looks professional.

- **Navigation Bar:** A consistent nav bar is included on all pages using Jinja2 template inheritance (`{% extends 'base.html' %}`).
- **Copy Feature:** A JavaScript function was added to the Home Page that allows users to copy the shortened URL to their clipboard with a single click.

[Insert Screenshot of Home Page here] *Figure 1: Home Page showing the input field and a successfully shortened URL.*

[Insert Screenshot of History Page here] *Figure 2: History Page showing the list of all shortened links.*

6. Challenges and Solutions

- **Challenge:** Handling invalid URLs.
  - *Solution:* I added a check to see if the user input starts with `http://` or `https://`. If not, the application automatically prepends `http://` to ensure the redirection works correctly.
- **Challenge:** Data persistence.
  - *Solution:* Setting up SQLite with SQLAlchemy required initializing the database within the `app.app_context()`. This ensures the database file is created automatically when the app is run for the first time.

7. Conclusion This project successfully meets all the requirements specified in the assignment. It provides a functional, user-friendly interface for shortening URLs and tracking link history. The architecture is modular, making it easy to add future features like analytics (tracking click counts) or custom URL aliases.