URL Shortener Implementation

Objective

Develop a URL Shortener service that converts long URLs into short, unique, and easily shareable links. The service should handle redirection, storage, scalability, and other essential features as outlined below.

Requirements

1. **Short URL Generation**
    a. Generate a unique short identifier for each long URL.
    b. Utilize Base62 encoding (characters `a-z`, `A-Z`, `0-9`) to keep URLs concise.
    c. Ensure that the same long URL always maps to the same short URL to prevent duplicates.
2. **Redirection**
    a. When a user accesses the short URL, they should be redirected to the original long URL.
    b. Handle cases where the short URL does not exist gracefully, providing appropriate error messages.
3. **Storage**
    a. Implement a storage mechanism to maintain the mapping between short URLs and long URLs.
    b. Ensure data persistence in case of system restarts or failures.
4. **Scalability & Performance**
    a. Design the system to handle high traffic with minimal latency.
    b. Consider caching frequently accessed URLs to improve performance.

Optional Challenges

For candidates looking to showcase their skills further, consider implementing one or more of the following:

1. **RESTful API**
    a. Develop a RESTful API allowing users to create, retrieve, and manage short URLs.
2. **User Interface**
    a. Build a simple web interface where users can input long URLs and receive shortened versions.
    b. Display analytics and manage existing short URLs.
3. **Tests**
    a. Include unit and integration tests to verify the functionality of your URL Shortener.
    b. Provide instructions on how to run these tests.

4. **Dockerization**
    a. Containerize the application using Docker for easier deployment.
5. **Expiration**
    a. Allow short URLs to expire after a specified duration.
    b. Implement mechanisms to clean up expired URLs to free up resources.
6. **Analytics**
    a. Track the number of times each short URL is accessed.
    b. Provide basic analytics, such as access timestamps or geographic data.

Deliverables

1. **Source Code**
    - documented and organized codebase.
    - Include any necessary configuration files.
    - Provide access to the source code via a public repository (e.g., GitHub, GitLab).
2. **README**
    - Instructions on how to set up, run, and test the application.
    - Describe any prerequisites or dependencies.
3. **Design Document**
    - Outline your architecture and design choices.