Design of URL Shortener

1. Overview

The URL shortener service is a web-based application that accepts long URLs from users and generates shorter, more manageable versions that redirect to the original long URLs. The primary objectives of this service are to make it easier for users to share and remember URLs, and to provide a seamless redirection experience.

2. Component

2.1 FRONTEND

- A simple form for users to input a long URL.
- A button to submit the long URL to the backend service for shortening.
- Display the generated short URL returned by the backend service.
- Display analytics information such as the number of clicks and the creation date of the short URL.

2.2 BACKEND

1. Restful API: The main interface for users to interact with the service. It will expose endpoints for creating short URLs and redirecting to the original URLs.
2. Short Code Generator: Generates unique short codes for each given URL.
3. Data Storage: Stores the mapping between short codes and original URLs, as well as any additional metadata.

2.2.1. Restful API

The backend service or API is responsible for accepting and processing user requests for shortening URLs and redirecting short URLs to the original long URLs. The main API endpoints for a URL shortener service typically include:

Create short URL:

- Method: POST
- Endpoint: /api/shorten , http://127.0.0.1:5000/shorten_url
- Input: JSON payload containing the long URL, e.g., {"long_url": "https://www.example.com/long-url"}
- Output: JSON payload containing the generated short URL, e.g., {"short_url": "https://short.example/abc123"}

This endpoint is responsible for receiving a long URL from the frontend application, generating a unique short URL code, storing the mapping in the database, and returning the generated short URL to the frontend.

Redirect to long URL:

- Method: GET
- Endpoint: /{short_url_code}
- Input: The short URL code as a path parameter, e.g., /abc123
- Output: A 301 (Moved Permanently) or 302 (Found) redirect to the original long URL.

This endpoint is responsible for processing requests to access the short URLs. When a user visits a short URL, the system extracts the short URL code from the path, looks up the corresponding long URL in the database, and redirects the user to the original long URL.

2.2.2. Short Code Generator

The main goal is to generate unique, short, and easily shareable codes that have a low probability of collision. There are two techniques which are feasible to use:

1. **Auto-incrementing ID with base62 encoding:**

Auto-incrementing ID with base62 encoding is a simple and efficient method to generate short codes for a URL shortener service. The idea is to use an auto-incrementing integer ID in your database for each long URL and encode it using base62 encoding, which uses a character set of 62 characters (0-9, a-z, A-Z).

- **Assign an auto-incrementing ID:** When a new long URL is added to your database, assign it a unique auto-incrementing integer ID. Most relational databases such as MySQL support auto-incrementing fields.
- **Encode the ID using base62 encoding:** Convert the unique integer ID to a base62-encoded string. This creates a more compact representation and results in shorter codes compared to using the integer ID directly.
- **Use the encoded ID as the short code:** The base62-encoded ID serves as the short code for the URL shortener service. This short code is unique as long as the auto-incrementing ID is unique.

2. **Hash-based short codes:**

Hash-based short codes involve generating a unique hash of the long URL using a hashing algorithm and then encoding and truncating the hash to create a short code. Here's a more detailed explanation of the process:

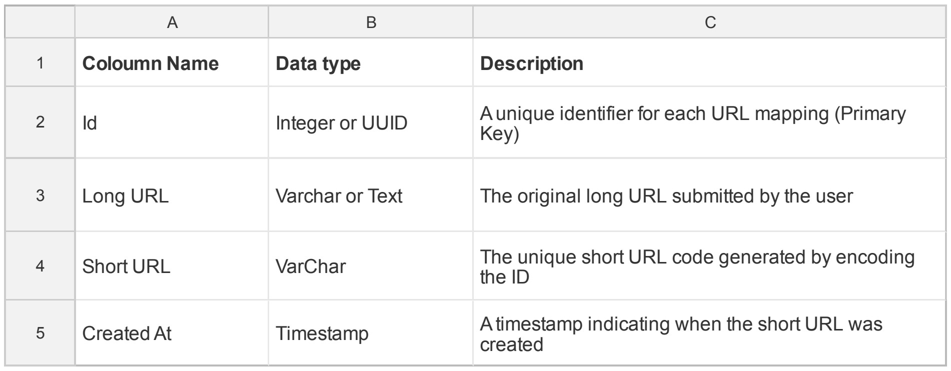
1. **Hash the long URL:** Take the long URL and generate a hash using a suitable hashing algorithm. Popular choices include MD5, SHA-1. These algorithms create a fixed-size output (e.g., 128 bits for MD5) that is unique for each unique input.
2. **Encode the hash:** Encode the hash using a more compact character set like base62 (0-9, a-z, A-Z), which results in a shorter string than hexadecimal representations.
3. **Truncate the encoded hash:** To create an even shorter code, you can truncate the encoded hash to a specific length. Keep in mind that truncating the hash increases the probability of collisions. Choose an appropriate length based on your expected number of URLs and acceptable collision probability.
4. **Handle collisions:** Although hash collisions are rare, they can still occur, especially when truncating the hash. If a collision is detected (i.e., the generated short code is already in use), you can resolve it using various methods.

- Add a salt (a random string or number) to the long URL and rehash it.
- Use a different hashing algorithm or a combination of hashing algorithms.
- Fall back to an alternative short code generation technique, such as an auto-incrementing ID.

2.2.3. Database

The system requires a database to store the mapping between original long URLs and generated short URLs. The database can be a relational database like PostgreSQL or MySQL. Store usage data like click counts, user agent, and timestamps. Schema for the database:

Table name: url_mappings



	A	B	C
1	**Coloumn Name**	**Data type**	**Description**
2	Id	Integer or UUID	A unique identifier for each URL mapping (Primary Key)
3	Long URL	Varchar or Text	The original long URL submitted by the user
4	Short URL	VarChar	The unique short URL code generated by encoding the ID
5	Created At	Timestamp	A timestamp indicating when the short URL was created

3. Conclusion

This design document provides an outline for building a URL shortener service. Two main components includes frontend and backend services. Backend include the Restful API, short code generator, data storage. By focusing on scalability, performance, and security, this service can be built to handle a large number of users and requests.