System architecture

Requirements

1. REST endpoint for shortening the given original URL
2. REST endpoint to redirect valid short URL to original URL
3. REST endpoint to get metrics of top 3 domains with their count.
4. Request and response in JSON format.
5. System should accept arguments over REST APIs.
6. Docker file to run built binary inside container.
7. For the same original URL it should always return the same short URL.

Components

1. API endpoints

Endpoint	Argument	Response
GET: /	<none>	Welcome to the URL Shortener API!

POST: /shorten	JSON body containing the original URL
	
	{
	"originalUrl": "https://www.example.com/very/long/url"
	}
		JSON response short URL or Error.
		
		{
		"shortUrl": "aaaaaab"
		}

GET: /{shortURL}	shortURL represents valid 7 char long short URL	Lookup the corresponding original URL then redirect to the original URL.

GET: /metrics	<none>	Top 3 domains which are shortened and their count.
		
		{
		"topDomains": [
		{
		"domain": "docs.google.com",
		"count": 7
		},
		{
		"domain": "www.audible.in",
		"count": 1
		}
		]
		}

2. In memory state
   To keep the system simple at this stage we prefer to keep the mapping of URLs in memory. We need to store the following.
   1. Mapping from long URL to short URL, used for shortening API.
   2. Mapping from short URL to long URL, used for the redirection API.
   3. Mapping of domains to their count, used for metrics API.

3. Range counter
   We use a range counter based approach for generating short URLs. The package accepts a range of min and max to use as a counter. The current counter starts with min and increments till max. Package throws error when range is exhausted.

   About short URL
   We prefer to use 62 characters A-Z, a-z, 0-9 as valid for short URLs. We use 7 chars (bytes) length for short URL, as 62 to the power of 7 is 3,521,614,606,208 which is large enough (3 trillion) to be useful for production level and will last for years.

   Ideas about scaling
   Since the range counter is tightly coupled with the web server. We can have multiple instances of url_shortner_main running with different owned ranges and instead of in-memory we can use clustered NO-SQL DB like cassandra to persist state.
   We can use a reverse proxy server with round robin to redirect and load balance the request to different instances of url_shortner_main.