A URL shortener in a REST API context is a service that takes a long URL and returns a shorter, more manageable version of it. This shortened URL can then be used to redirect users to the original, longer URL. Here’s a basic overview of how it typically works:

**1.POST Request to Shorten URL:** The client sends a POST request to the URL shortener API with the long URL in the request body.

**Example Request:**

**POST /shorten**

Content-Type: application/json

{

"longUrl": "https://www.example.com/some/very/long/url"

}

**2.Generate Short URL**: The server generates a unique identifier for the long URL and creates a shortened URL using this identifier.

**Example Response:**

HTTP/1.1 201 Created

Content-Type: application/json

{

"shortUrl": "https://short.ly/abc123"

}

**3.Redirect on Access**: When a user accesses the short URL, the service redirects the request to the original long URL.

**Example Request:**

GET /abc123

**Example Response:**

HTTP/1.1 301 Moved Permanently

Location: <https://www.example.com/some/very/long/url>

**Key Components of a URL Shortener API**

1. **Endpoint to Shorten URL**: Typically, a POST request where the client sends the long URL and receives the short URL in response.
2. **Endpoint to Redirect**: Typically, a GET request where the server maps the short URL identifier to the long URL and redirects the client.
3. **Data Storage**: A database or in-memory store to map short URL identifiers to long URLs.

**Example Implementation with Spring Boot**

Here’s a simplified version of how you might implement a URL shortener using Spring Boot:

@RestController

@RequestMapping("/api")

public class **UrlShortenerController** {

private final UrlShortenerService urlShortenerService;

public UrlShortenerController(UrlShortenerService urlShortenerService) {

this.urlShortenerService = urlShortenerService;

}

@PostMapping("/shorten")

public ResponseEntity<ShortenedUrl> shortenUrl(@RequestBody LongUrl longUrl) {

String shortUrl = urlShortenerService.shortenUrl(longUrl.getUrl());

return ResponseEntity.status(HttpStatus.CREATED).body(new ShortenedUrl(shortUrl));

}

@GetMapping("/{shortUrl}")

public ResponseEntity<Void> redirect(@PathVariable String shortUrl) {

String longUrl = urlShortenerService.getLongUrl(shortUrl);

return ResponseEntity.status(HttpStatus.MOVED\_PERMANENTLY).location(URI.create(longUrl)).build();

}

}

@Service

public class **UrlShortenerService** {

private final Map<String, String> urlMap = new ConcurrentHashMap<>();

private final String baseUrl = "https://short.ly/";

public String shortenUrl(String longUrl) {

String shortUrl = baseUrl + UUID.randomUUID().toString().substring(0, 6);

urlMap.put(shortUrl, longUrl);

return shortUrl;

}

public String getLongUrl(String shortUrl) {

return urlMap.getOrDefault(shortUrl, "/error");

}

}

class **LongUrl** {

private String url;

// getters and setters

}

class **ShortenedUrl** {

private String shortUrl;

public ShortenedUrl(String shortUrl) {

this.shortUrl = shortUrl;

}

// getters and setters

}

This is a simple implementation that can be extended with features like custom short URLs, expiration times, analytics, and more.