A URL shortener in a REST API context is a service that takes long URLs and converts them into shorter, more manageable URLs. This can be particularly useful for sharing links on social media, in emails, or anywhere space is limited. Here’s an overview of how a URL shortener typically works in a REST API:

**Basic Functionality**

1. **Shortening a URL**:
   * **Request**: A client sends a POST request to the API with a long URL.
   * **Response**: The server processes this request, generates a unique short URL, stores the mapping between the long URL and the short URL, and returns the short URL to the client.
2. **Redirecting to the Long URL:**
   * **Request:** When a user accesses the short URL (e.g., by clicking a link), the client (browser) sends a GET request to the API.
   * **Response:** The server looks up the original long URL associated with the short URL and redirects the user to the long URL using a 301 or 302 HTTP status code.

**Example API Endpoints**

**POST /shorten**: Shortens a given long URL.

**Request Body:**

json

{

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

}

**Response**:

{

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

}

**GET /{shortUrlId}: Redirects to the original long URL.**

* **Request**: Accessing the short URL directly.
* **Response**: A redirect to the long URL.

**Example Implementation in Spring Boot**

* Here is a basic example of how you might implement a URL shortener REST API in Spring Boot:

**Dependencies**

Add the necessary dependencies in your pom.xml:

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

</dependencies>

**URL Entity**

Create an entity to represent the URL mappings:

@Entity

public class UrlMapping {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String longUrl;

private String shortUrl;

// getters and setters

}

**Repository**

Create a repository to handle database operations:

public interface UrlMappingRepository extends JpaRepository<UrlMapping, Long> {

Optional<UrlMapping> findByShortUrl(String shortUrl);

}

**Controller**

Create a controller to handle API requests:

@RestController

@RequestMapping("/api")

public class UrlShortenerController {

@Autowired

private UrlMappingRepository urlMappingRepository;

@PostMapping("/shorten")

public ResponseEntity<?> shortenUrl(@RequestBody Map<String, String> request) {

String longUrl = request.get("longUrl");

String shortUrl = generateShortUrl(); // Implement your logic to generate a short URL

UrlMapping urlMapping = new UrlMapping();

urlMapping.setLongUrl(longUrl);

urlMapping.setShortUrl(shortUrl);

urlMappingRepository.save(urlMapping);

return ResponseEntity.ok(Collections.singletonMap("shortUrl", shortUrl));

}

@GetMapping("/{shortUrl}")

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

Optional<UrlMapping> urlMappingOptional = urlMappingRepository.findByShortUrl(shortUrl);

if (urlMappingOptional.isPresent()) {

return ResponseEntity.status(HttpStatus.MOVED\_PERMANENTLY)

.location(URI.create(urlMappingOptional.get().getLongUrl()))

.build();

} else {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}

}

private String generateShortUrl() {

// Implement your logic to generate a unique short URL

return UUID.randomUUID().toString().substring(0, 6);

}

}

This example demonstrates the basic functionality of a URL shortener service using a REST API in Spring Boot. You can expand upon this by adding features like custom short URLs, expiration dates, and analytics.