Toll Plaza Finder API - Official Documentation

1. Project Overview

The Toll Plaza Finder API is a Spring Boot-based backend application that determines toll plazas between two Indian pincodes using real-time route calculations and spatial queries. The project leverages PostgreSQL (running via Docker), Redis caching, and an external routing API (OSRM) to provide fast and accurate results.

2. Technology Stack

- Backend: Spring Boot (Java 17)
- Database: PostgreSQL (PostGIS for spatial queries) (Running via Docker Compose)
- Caching: Redis
- External APIs: OpenStreetMap (Nominatim for geocoding), OSRM for route calculation
- Testing: JUnit, Mockito, Jacoco
- Build Tool: Maven
- Code Quality & Coverage: SonarQube
- API Documentation: Swagger UI (Postman can also be used for testing)

3. System Architecture

1. The user provides source and destination pincodes via API.
2. The system fetches latitude/longitude using OpenStreetMap API.
3. The route is calculated using OSRM API.
4. The database queries toll plazas within 50km along the route.
5. The results are cached in Redis to optimize repeated queries.
6. The API returns structured JSON data.

4. API Specification

Endpoint: Get Toll Plazas

URL: POST /api/v1/toll-plazas

Request Body:

{

"sourcePincode": "411001",

"destinationPincode": "400706"

}

Response:

{

"sourcePincode": "411001",

"destinationPincode": "400706",

"distanceInKm": 105.80605,

"tollPlazas": [

{

"name": "Talegaon Toll Plaza",

"latitude": 18.7138719,

"longitude": 73.6458575,

"distanceFromSource": 31.629253764636204

},

]

}

5. Database Design

- The **Toll Plaza table** consists of columns for ID, name, latitude, longitude, and geo state.
- The **Route Cache table** stores source and destination pincodes, route distance, and cached toll plaza data.

6. How to Run the Project (Docker Setup)

Prerequisites:

- Java 17+
- Docker & Docker Compose installed

Step 1: Clone the Repository

git clone https://github.com/your-repo/toll-plaza-finder.git

cd toll-plaza-finder

Step 2: Set Up PostgreSQL & Redis with Docker Compose

Create a docker-compose.yml file (if not already present)

Step 3: Start the Services

docker-compose up -d

Step 4: Configure application.properties

Step 5: Build and Run the Application

mvn clean install

mvn spring-boot:run

7. SonarQube Setup & Test Coverage (59.4%)

Step 1: Configure pom.xml for SonarQube

Step 2: Run SonarQube Analysis

mvn clean verify

mvn clean verify sonar:sonar

🚀 **View Report at: http://localhost:9000**

8. Testing Guide

Run Unit Tests

mvn test

Test API with Swagger UI (or Postman)

1. Open Swagger UI at http://localhost:8080/swagger-ui.html.
2. Select POST /api/v1/toll-plazas.
3. Click Try it out.
4. Enter JSON request:

   {

   "sourcePincode": "834001", // Jharkhand

   "destinationPincode": "400706" // Navi Mumbai

   }
5. Click Execute and verify the response.

9. Deployment Instructions (Docker)

Step 1: Package as JAR

mvn package

Step 2: Run in Production (Dockerized)

docker build -t toll-plaza-api .

docker run --network=tollplaza_network -p 8080:8080 toll-plaza-api