ActiveTO REST API Project Overview

1. Project Summary

Project Name: ActiveTO REST API

Technologies Used: Spring Boot, JUnit, MockMVC, Python, MySQL, Docker

Description
This project is a REST API built with Spring Boot that powers ActiveTO Web application. The API provides endpoints for searching and sorting weekly updated drop-in activities at City of Toronto's recreation facilities by their category, type, activity name and facility. It also provides location and contact info for facilities. This API is designed for publicly access and can easily be integrated into a front-end web or mobile application.

Tech Stack

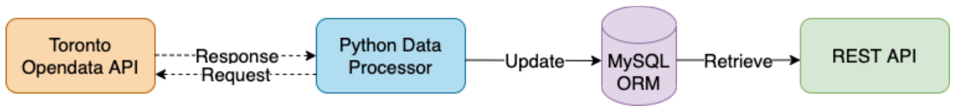
- Backend Framework: Spring Boot
- Data processor: Python
- Database: MySQL with JPA/Hibernate for ORM
- Build Tool: Maven
- API Documentation: Swagger/OpenAPI for API documentation

Key Features

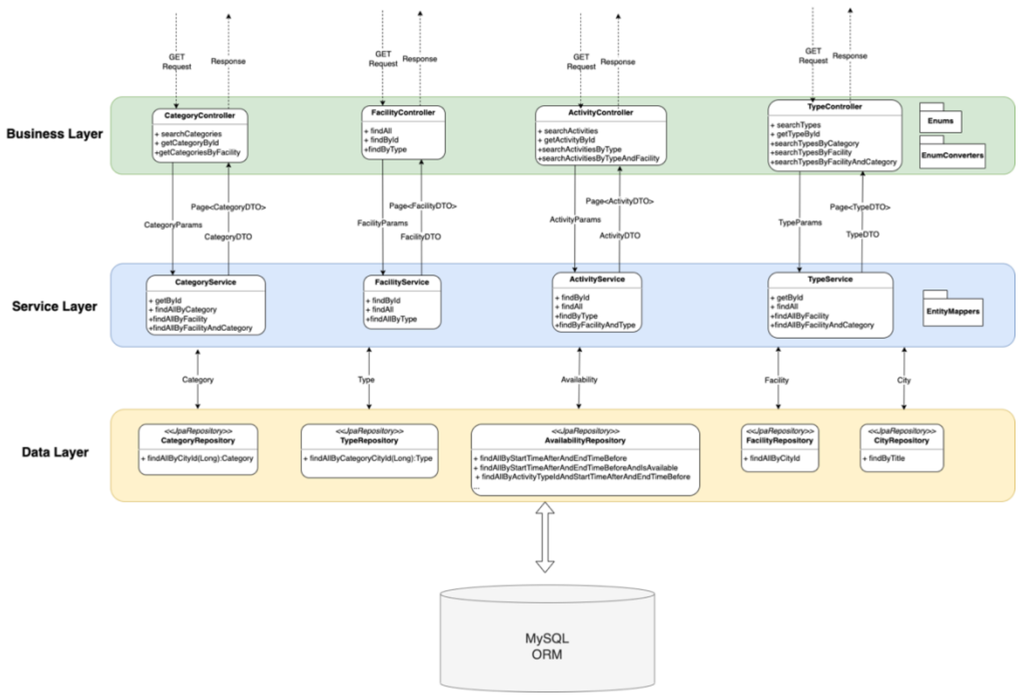
- Data Retrieval: The API allows clients to retrieve resource data (e.g., categories, types, availabilities, facilities) via GET requests.
- Weekly Update: A python script is run once a week for fetching newly updated activity data from Toronto Opendata API and updates the database after processing accordingly.
- No Authentication Required: Publicly accessible; no login or authentication is needed to use the API.
- Stateless: Each request from the client contains all the necessary information to service the request, ensuring simplicity and scalability.

2. Architecture Overview

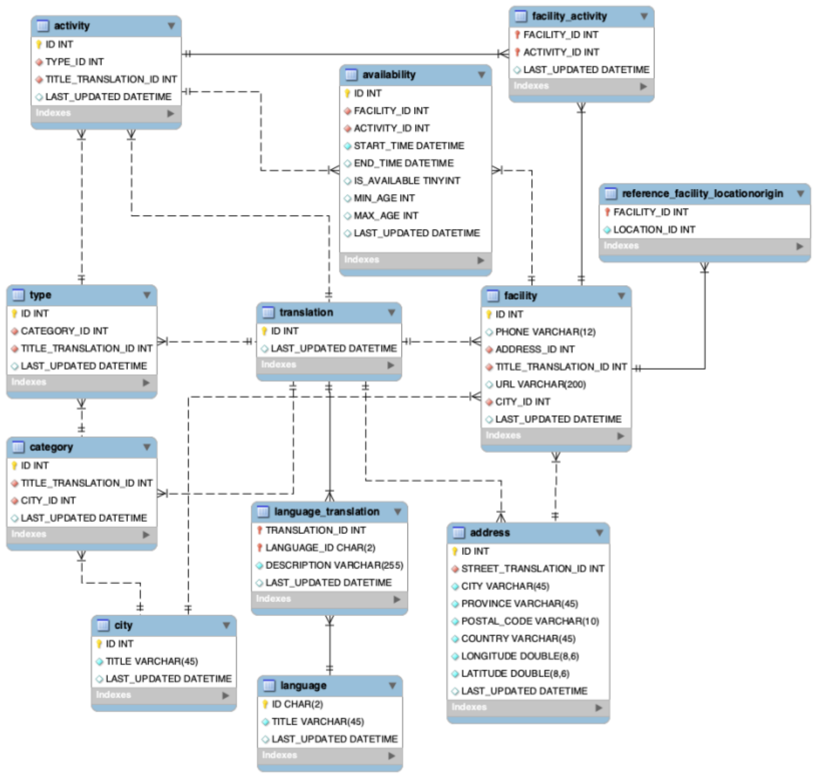
Stack Overview



REST API Architecture



Database Schema



Endpoints Overview

1. Facility
   - GET /toronto/types/{typeId}/facilities: Retrieves facilities that have a specific type of activities by type ID.
   - GET /toronto/facilities: Retrieves all facilities.
   - GET /toronto/facilities/{id}: Retrieves a specific facility by ID.

2. Activity
    - GET /toronto/types/{typeId}/activities: Retrieves a specific type of activities by type ID.
    - GET /toronto/facilities/{facilityId}/types/{typeId}/activities: Retrieves a specific type of activities in a specific facility by facility ID and type ID.
    - GET /toronto/activities: Retrieves all activities.
    - GET /toronto/activities/{id}: Retrieve a specific activity by ID.

3. Type
    - GET /toronto/types: Retrieves all types.
    - GET /toronto/type/{id}: Retrieves a specific type by ID.
    - GET /toronto/facilities/{facilities}/types: Retrieves all types in a specific facility by facility ID.
    - GET /toronto/facilities/{facilityId}/categories/{categoryId}/types: Retrieves all types in a certain category in a certain facility.
    - GET /toronto/categories/{categoryId}/types: Retrieves all types in a specific category.

4. Category
    - GET /toronto/facilities/{facilityId}/categories: Retrieves all categories that a specific facility provides.
    - GET /toronto/categories: Retrieves all categories.
    - GET /toronto/categories/{id}: Retrieves a specific category by ID.

For more detailed information, visit https://www.mollyzhang.dev/apps/activeto/api/swagger-ui

3. Deployment and CI/CD Pipeline

Tools & Technologies

- CI/CD Platform: GitHub Actions
- Version Control: Git/GitHub
- Build Tool: Maven (for building the Spring Boot project)
- Testing Frameworks: JUnit for unit testing, MockMvc for integration testing
- Deployment: Docker
- Proxy server: Nginx
- Hosting platform: DigitalOcean

Pipeline Workflow

1. Source Code Checkout
    - The pipeline is triggered when a commit is pushed to the main branch or when a pull request is created.
    - The CI/CD system checks out the latest version of the source code from the Git repository.

2. Build
    - The project is built using Maven, which compiles the source code, runs code quality checks, and packages the application as a JAR file.
    - Dependency management is handled during the build process, ensuring all required libraries are included.

3. Testing
    - Unit Tests: JUnit tests are automatically executed to verify the correctness of individual components.
    - Integration Tests: MockMvc is used to run integration tests on API endpoints, ensuring that all services communicate properly.

4. Packaging
    - The application is packaged into a Docker image, which encapsulates the API and its dependencies.

    - The Docker image is pushed to a Docker registry for easy deployment.

5. Deployment
    - After successful testing in the staging environment, the application is automatically deployed to the DigitalOcean instance.

4. Security Considerations

- Environment Variables: Sensitive information such as API key is stored in environment variables in a .env file and never exposed in the client-side code.

5. Live Demo and Source Code

- Live Demo: https://www.mollyzhang.dev/apps/activeto/api/swagger-ui
- Source Code: https://github.com/dongyue-zhang/Active_Toronto_Docker