Design Documentation for Congestion Tax Services

Table of Contents

Introduction

This document describes the design and implementation of congestion tax service which helps to calculate the congestion tax for vehicles.

Design Considerations

Given details

- Interval and amounts, max charge, single charge rule, tax free days and tax exempted vehicles
- Date time details
- Congestion tax calculation can be called over HTTP
- There are bugs in the calculation
- There is no structure for the code

Assumptions

- The services will be used by vehicle owners to get the congestion taxes for the time entries and the amount can be paid online.
- Considering that the number of users can be high congestion tax service will be implemented with REST API architecture style.
- Considering tax calculation service as utility service. Tax data is not stored in a database as it can be added later to the service when sufficient vehicle details are available.
- Time intervals, amounts, holidays etc are not changing but can be different for different cities. These details can be stored in a file and can configure according to the city on runtime.

Defining requirements

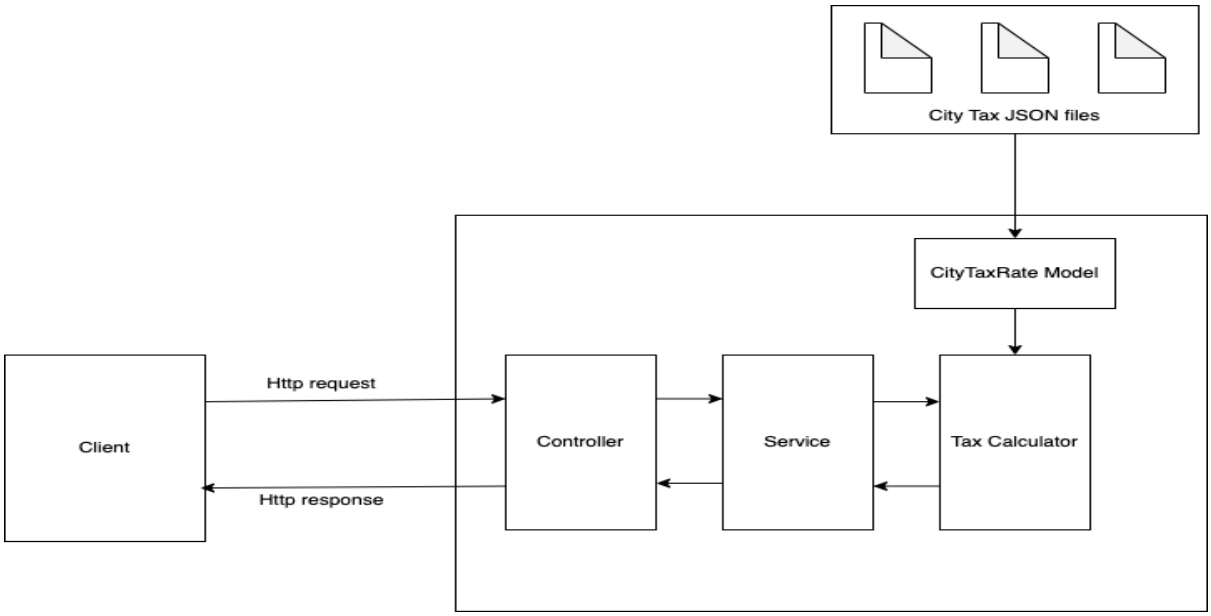
Functional requirements

1. Add application entry point convert the solution to a RESTful web service, so that it can be called over HTTP.
2. Tax calculator can be applied to multiple cities
3. Consider edge cases and testability - Create tests cases with different inputs and test
4. Find and fix bugs in tax calculation

Non-functional requirements

1. Maintainability - Code refactoring and simplify the calculation logic
2. Code quality – Check using SonarQube
3. Reliability - Exception handling
4. Portability - containerization

System Design



The configured JSON file for a city will be read on application startup and the time and tax rate data, currency, max charge per day and holidays for the city will be populated on the data model.

The Http request for calculating the congestion tax will be handled by controller. The service layer will use the tax calculator utility class to get the calculated tax.

API Documentation

Generate API documentation using swagger and open api.

Documentation can be accessed using : *http://localhost:8080/swagger-ui/index.html#*

API Details

End point: /congestion-tax to calculate tax (POST)

Content-Type : application/json

Request Body :

```
{
   "vehicle":{
      "VehicleType":"Car"
   },
   "dates":[
 "2013-01-14 21:00:00",
"2013-02-07 06:23:27",
   ]
```

}

Response status : 200 OK

Response : *{*

"statusCode": "Tax calculated successfully",

"statusMessage": "Congestion Tax : 10 SEK"

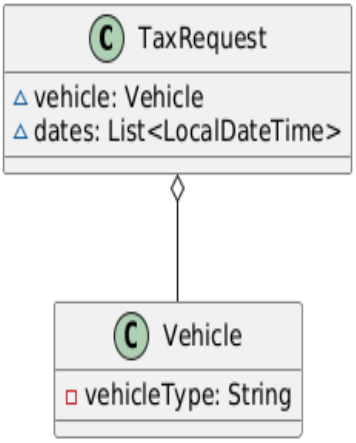
}

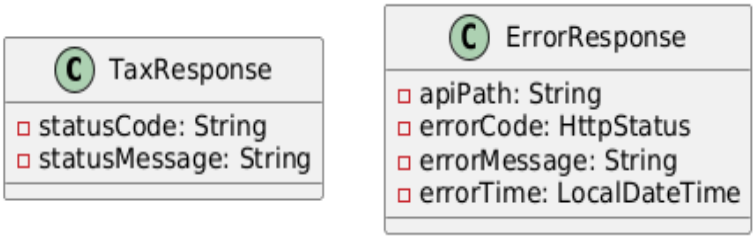
Curl statement for HTTP call :

```
curl --location 'http://localhost:5000/congestion-tax' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--data '{
   "vehicle":{
      "VehicleType":"Car"
   },
   "dates":[
      "2013-01-14 21:00:00", "2013-02-07 06:23:27" ] }'
```

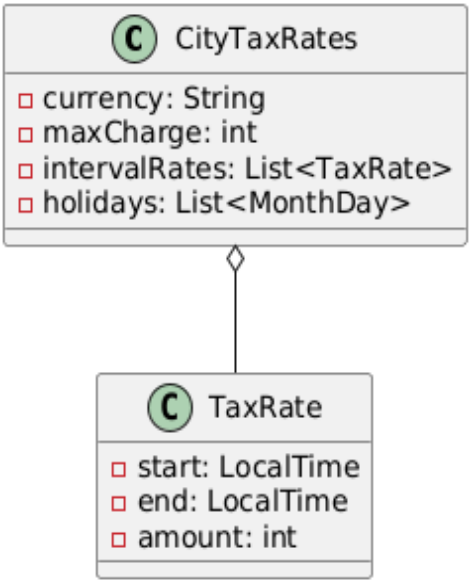
Data Models

Requests and responses

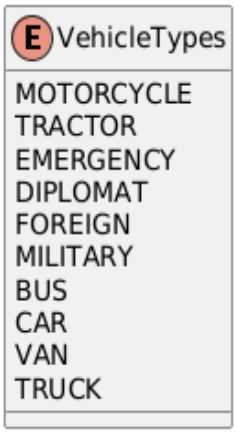




City tax rates – read from JSON



Vehicle Types



Implementation

Development of REST API with the help of Spring Boot.

Layered structure implementation with controller, service and utility classes.

1. Controller – Handling requests and responses
2. Service interface – classes can implement service interface to provide different implementations for tax calculations.

3. Service – Implement service methods.
4. Utility classes – Calculation, conversion and data reading.
5. Configuration –Configure object mapper to read dates in java.time format.
6. Exception handler and Validation – Global exception handler class contains handlers for exceptions and custom validation for checking white spaces in string.
7. Logging – Use Spring AOP to add logging.

Testing

A test project with test cases for functional testing with the help of Junit testing framework for the following cases

1. Few datetime entries
2. Maximum tax perday
3. On weekends, holidays and July month
4. Tax exempted vehicles
5. Single charge rule
6. For no tax time i.e. between 18.30-05.59
7. Invalid vehicle type test
8. Invalid date time test

Code Quality

Configure SonarQube in project to find code issues, bugs, vulnerabilities and code complexity and resolve.

Deployment

Dockerfile for building project and docker image and to run container which will help to run the service on other systems also will help for the deployment on cloud platforms.