TAXI BOOKING SYSTEM

A backend service based on REST APIs

Siddhant Tanpure

Introduction

This document contains implementation details of a simple taxi booking system Restful web service APIs. The document will help to understand how to use different APIs to book a taxi (reset, book) and to check status of the taxis (tick, check).

The Create, Read, Update and Delete operations (CRUD) in Restful web services are performed using HTTP verbs. The most commonly used verbs are POST, GET, PUT, PATCH and DELECT.

Design Decisions

1) The reset (API/reset) service will be called before calling any other service (book, check, tick) to create resources in this case taxis. Then only all other services will be available to the user.
2) If tick is called at any time the request will return status 200 OK, even if there are not taxis booked or available.
3) The data will not be stored in any permanent memory, there when services are stopped all the data will be lost.

Technologies Used

1) Spring boot framework - http://spring.io/
2) Junit Test framework - https://spring.io/guides/gs/testing-web/
3) Manual testing and development is done using Postman - https://www.getpostman.com/
4) IntelliJ IDE - https://www.jetbrains.com/idea/
5) Maven - https://maven.apache.org/

API Documentation

There are four main REST APIs:

1) Reset – api/reset
2) Book – api/book
3) Tick – api/tick
4) Check – api/check

The following sections describes the procedure of using each of the four aforementioned APIs.

1. Reset – api/reset

TITLE	Reset car information
REQUIREMENTS	When called will reset all cars data back to the initial state regardless of cars that are currently booked.
URL	/api/reset
METHOD	PUT
URL PARAMS	None
DATA PARAMS	None
SUCCESS RESPONSE	200 OK
ERROR RESPONSE	400 Bad Request
SAMPLE CALL	http://localhost:8080/api/reset
NOTES	Method PUT is used here because the service either creates or update resources.

2. Book – api/book

TITLE	Customer request for cars
REQUIREMENTS	1) System should pick the nearest available car to the customer location and return the total time taken to travel from the current car location to customer location then to customer destination. 2) All cars are available initially and become booked once it is assigned to a customer. It will remain booked until it reaches its destination, and immediately become available again. 3) If there is more than one car near the customer location, your service should return the car with the smallest id. 4) Cars can occupy the same spot 5) If there is no available car that can satisfy the request, your service should return an empty response, not an error 6) only one customer to a car
URL	/api/book

METHOD	PUT
URL PARAMS	None
DATA PARAMS	{ source: { x: x1, y: y1 }, destination: { x: x2, y: y2 }}
SUCCESS RESPONSE	1) If all cars are booked 200 OK Response body: None 2) If a car is available Code: 200 Response body: { car_id: id, total_time: t }
ERROR RESPONSE	3) If service is called before calling the reset 400 Bad Request
SAMPLE CALL	http://localhost:8080/api/book Body: { "source" :{ "x" :-3, "y" :-5}, "destination": { "x":-3, "y":-6} }
NOTES	"Only one car be assigned to a customer." This requirement is not handled in the system as there is no way to identify if same customer is trying to book again. Method PUT is used here because the service update resources.

3. Tick – api/tick

TITLE	Time stamp increment
REQUIREMENTS	When called should advance your service time stamp by 1 time unit
URL	/api/tick
METHOD	PUT
URL PARAMS	None
DATA PARAMS	None
SUCCESS RESPONSE	200
ERROR RESPONSE	400 Bad Request
SAMPLE CALL	http://localhost:8080/api/tick
NOTES	Method PUT is used here because the service update resources.

4. Check – api/check

TITLE	Check status of the taxis
REQUIREMENTS	When called returns the details of all the taxis
URL	/api/check
METHOD	GET
URL PARAMS	None
DATA PARAMS	None
SUCCESS RESPONSE	200 OK
ERROR RESPONSE	400 Bad Request
SAMPLE CALL	http://localhost:8080/api/tick
NOTES	Method GET is used here because the API retrieves the information of the resource here.

Frequently asked questions: -

1) How to configure the project?

i. Install any IDE to import the project (example IntelliJ IDEA, Eclipse)
ii. Use open project option to import the project

2) How to run the project in Linux?

i. Run the executable jar provided in the project folder using command: Java -jar taxibookingsystem-api-1.0-SNAPSHOT.jar
ii. Use commands provided below to execute from the terminal.

Reset API	curl -i -X PUT -H "Content-Type:application/json" http://localhost:8080/api/reset
Book API	curl -i -X PUT -H "Content-Type:application/json" http://localhost:8080/api/book/ -d '{ "source" :{ "x" :-3, "y" :-5}, "destination": { "x":-3, "y":-6} }'

Check API	curl http://localhost:8080/api/check
Tick API	curl -i -X PUT -H "Content-Type:application/json" http://localhost:8080/api/tick

3) How to run the project in windows?

i. Once the successful build is completed click on run.
ii. Go to the Postman run below URLs.

Reset API	http://localhost:8080/api/reset
Book API	http://localhost:8080/api/book/ Body: - { "source" :{ "x" :-3, "y" :-5}, "destination": { "x":-3, "y":-6} }
Check API	http://localhost:8080/api/check
Tick API	http://localhost:8080/api/tick

Help and Support

Feel free to reach out to Siddhant Tanpure via email on *sbtanpur@asu.edu* for any help and support with respect to the usage of this software.
