Task1

Technology Used:

Java
Spring Boot
Spring Security
H2 InMemory database

- Task parameters
  (a) Testing
    - Integration testing can be achieved by implementing using **JUnit** and **Mockito**.

  (b) Security
    - Current implementation is confiured with **Basic authentication** that comes with Spring Security framework.
    - Can be easily changed to **JDBC Authentication, LDAP Authentication** or any other Active Directory service.
    - Spring Serurity default user
      - **Username: user**
      - **Password: user**
    - Application Users created for Inmemory authentication
      - **User Role**
        - **Username: test**
        - **Password: test**
      - **Admin Role, User Role**
        - **Username: admin**
        - **Password: admin**

  (c) Scalability
    - Current implementation can be extented to easily integrate with other **relational or non-relational database** that can be distributed by using sharding.
    - If there are **person entities that are visited too frequently**, we can use caching to store the most accessed data, so we can avoid the database hit everytime improving the perfomance.

  (d) Limitations
    - Limited use of Java8 features.
    - Basic Authentication is used which is not something preferable for production ready application.
    - Unit testing is not implemented in this.

  (e) Documentation
    - Haven't used any library in this implementation which can be used to generate documentation, but we can use **Open API** to generate API documentation, and we can configure swagger UI with **Open API**.
    - For this task, I have prepated API documentation manually, and it is in **same document** after this section.

(f) Deployment

- Implemented using spring boot, so it can be easily deployed on any machine.
- No extra configuation is required to make it live, just maven clean install and spring boot run command will host it on any local machine or server.
- **Project cloning and deployement steps are in README.md file in github repository.**

Developed 5 API's to work with person entity.

GET http://localhost:8080/api/v1/persons

- Retrieves all the person that are created
- Request Param/ Request Body: None
- Response: List of person entities

GET http://localhost:8080/api/v1/person/{personId}

- Retrieves the specific person by **personId** parameter
- Request Param/ Request Body: {**personId**}
- Response: Returns Single person entity if exists or else custom not found exception

POST http://localhost:8080/api/v1/persons

- Stores person entities into database
- Can take any number of person entities as input, and this API will insert all entities in database and return the same entities with **unique ID**.
- Request Param/ Request Body: List of person entities
- Response: List of person entities with unique id that are created

PUT http://localhost:8080/api/v1/persons

- Updates person entities into database
- Can take any number of person entities as input, request body is different from POST service wherein **we don't pass ID of entity**, and this API will update all entities in database and returns the updated entities.
- Request Param/ Request Body: List of person entities that needs to be updated, service will search base on **{personId}** that is passed in request body.
- Response: List of updated person entities

DELETE http://localhost:8080/api/v1/person/{personId}

- Deletes the specific person by **personId** parameter
- Request Param/ Request Body: **{personId}**
- Response: Deletes single person entity by **{personId}** if exists or else custom not found exception

Note:

- Changed the person entity to include **{id}** which is not given in the task definition, reason for that is, if we want to update any person we didn't had any unique parameter that can be used to search person, so I have added this **{id}** parameter.

Below is the step by step testing of all the API's.

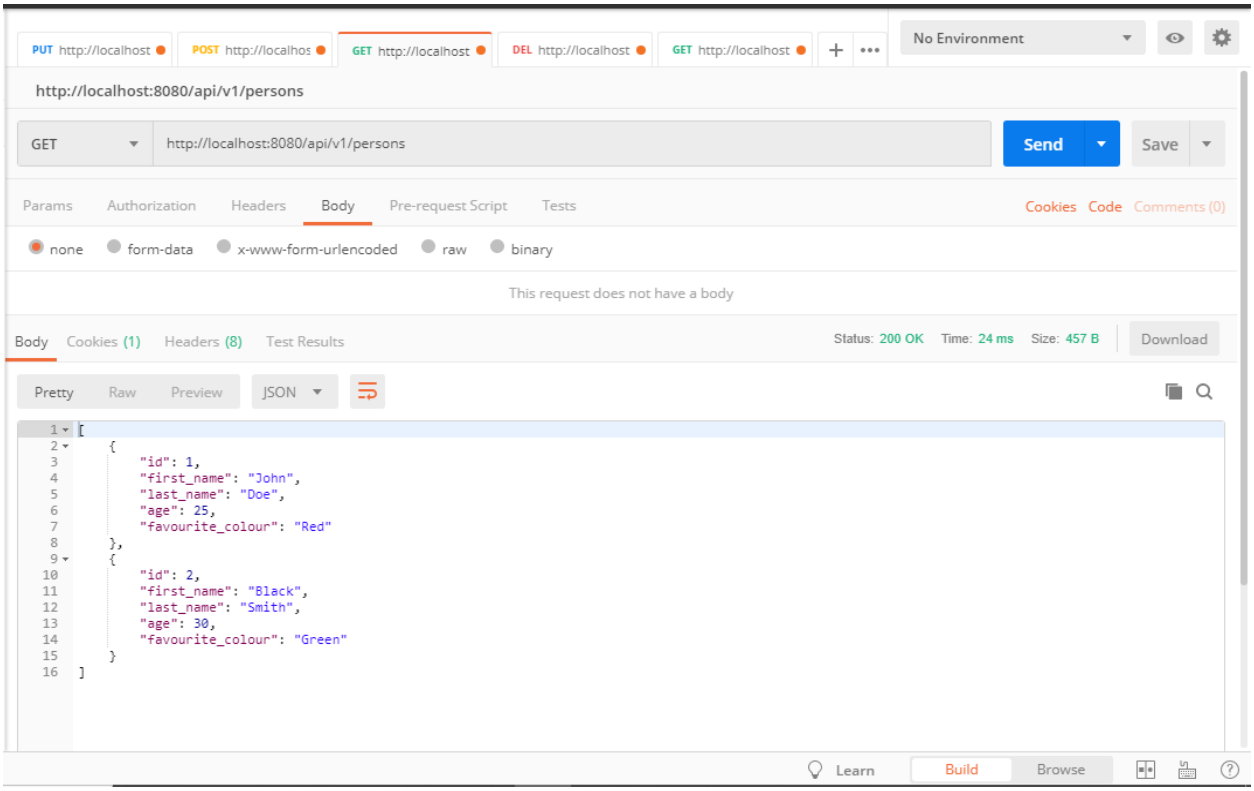

Figure 1 *GET http://localhost:8080/api/v1/persons*

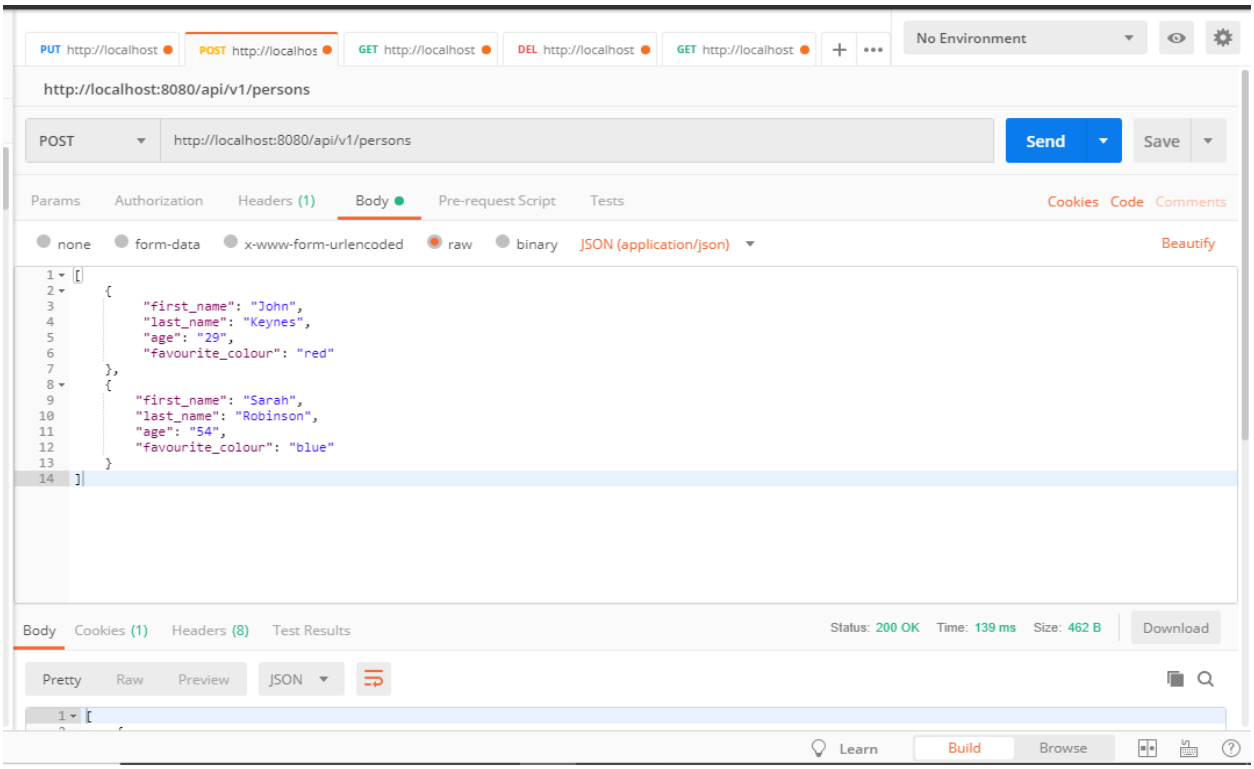

Figure 2 *POST Request* http://localhost:8080/api/v1/persons

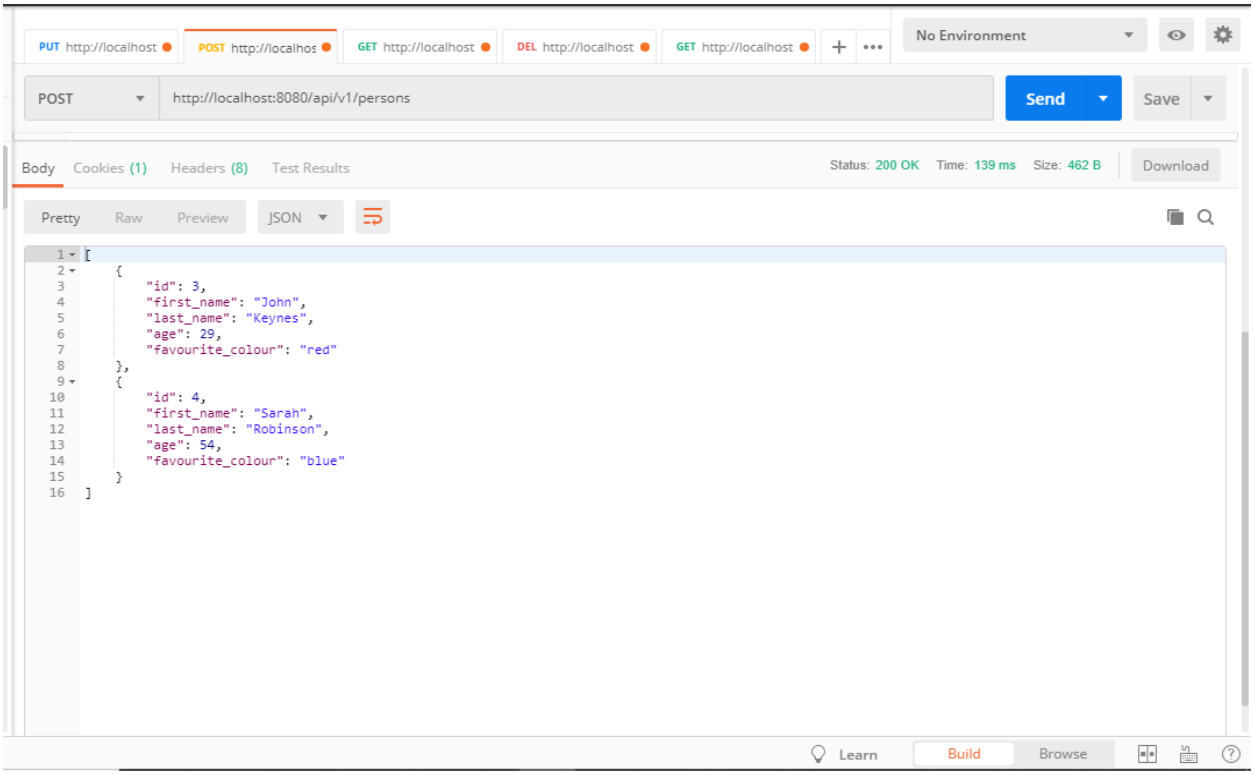

Figure 3 *POST Response* http://localhost:8080/api/v1/persons

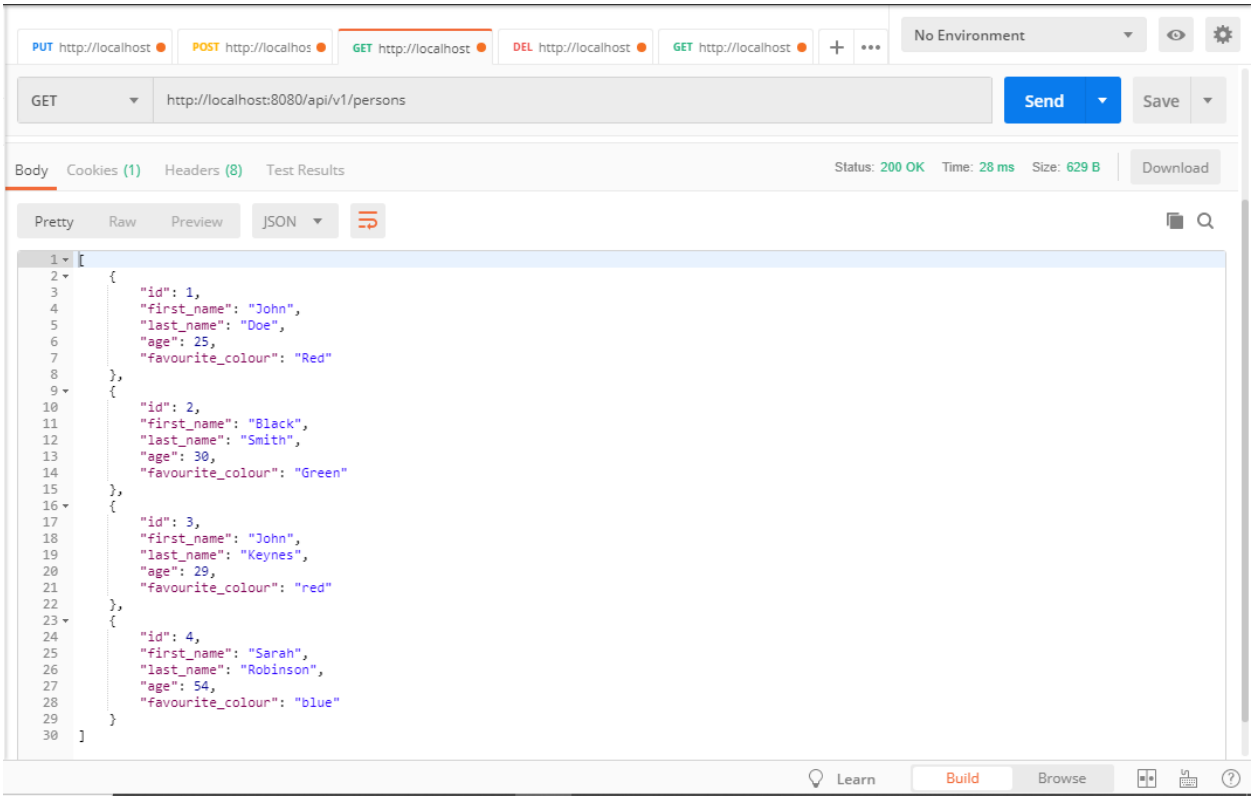

Figure 4 *GET http://localhost:8080/api/v1/persons*

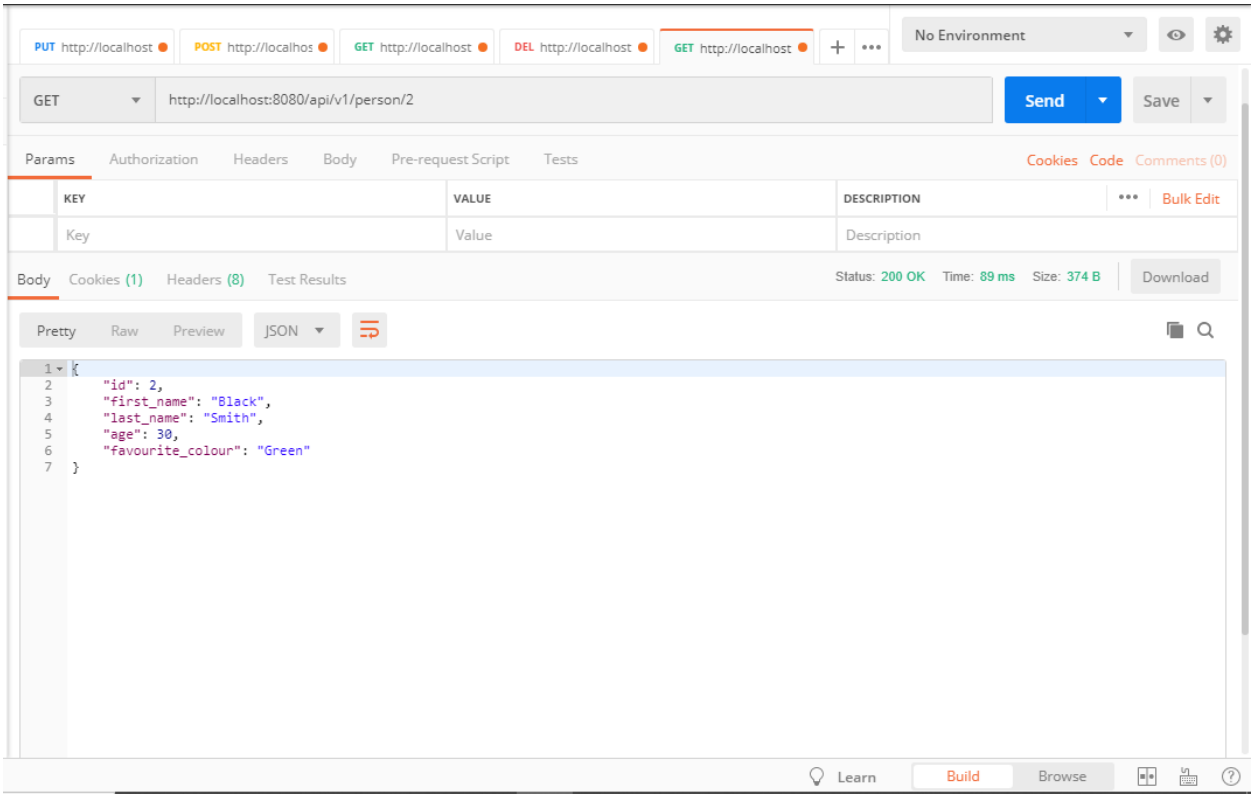

Figure 5 *GET http://localhost:8080/api/v1/person/2*

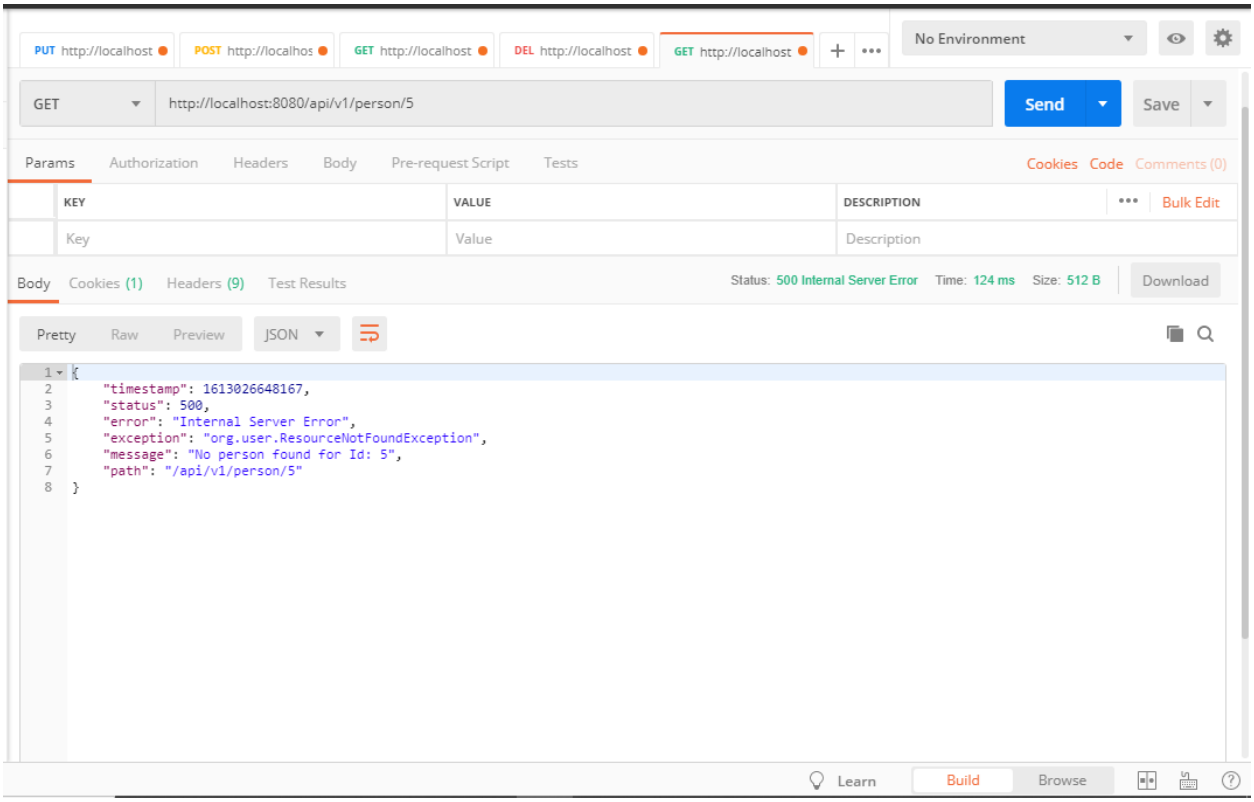

Figure 6 *GET http://localhost:8080/api/v1/person/5*

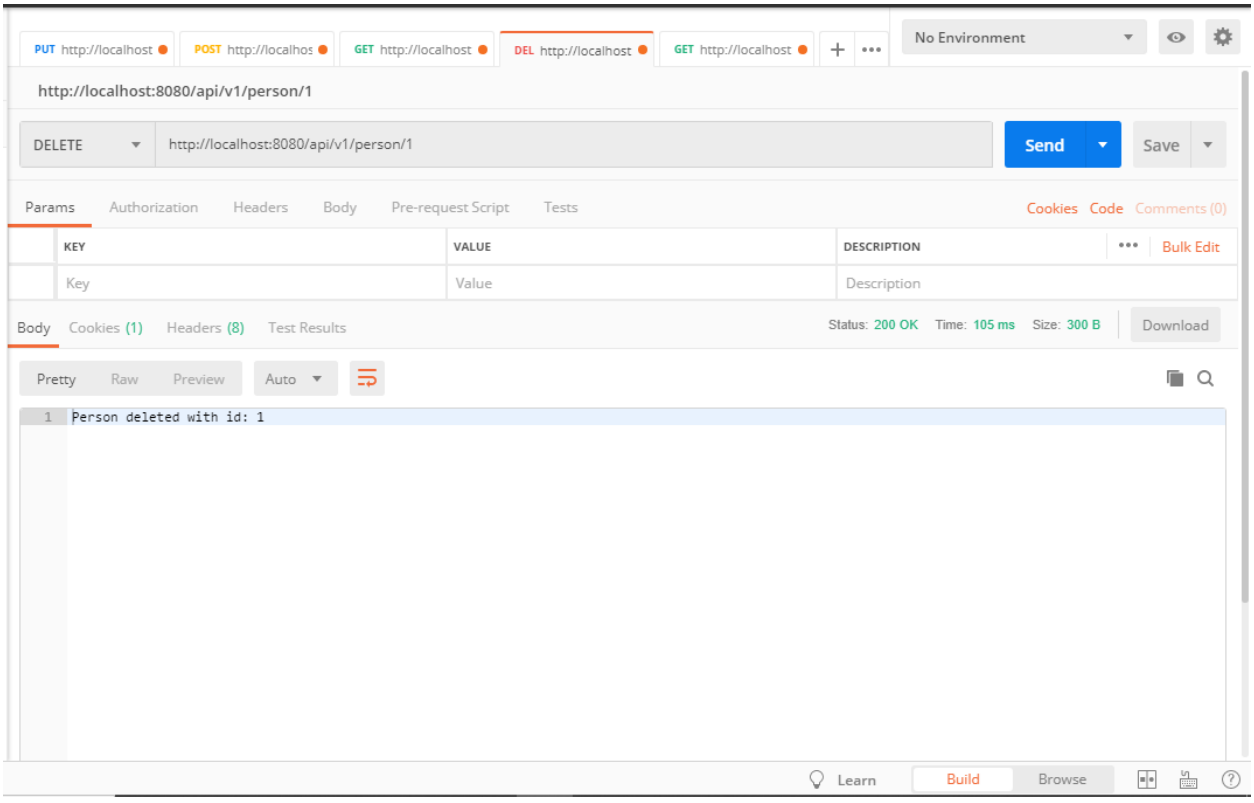

Figure 7 *DELETE http://localhost:8080/api/v1/person/1*

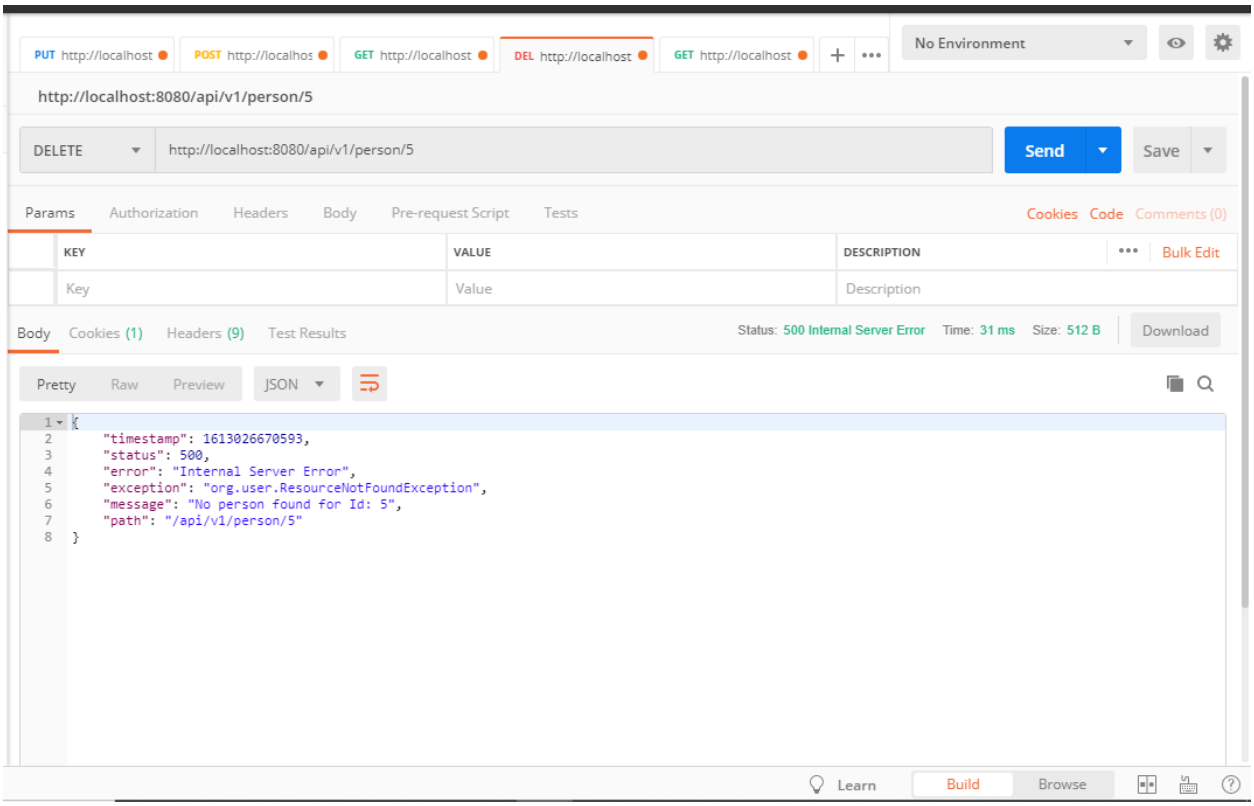

Figure 8 DELETE http://localhost:8080/api/v1/person/5

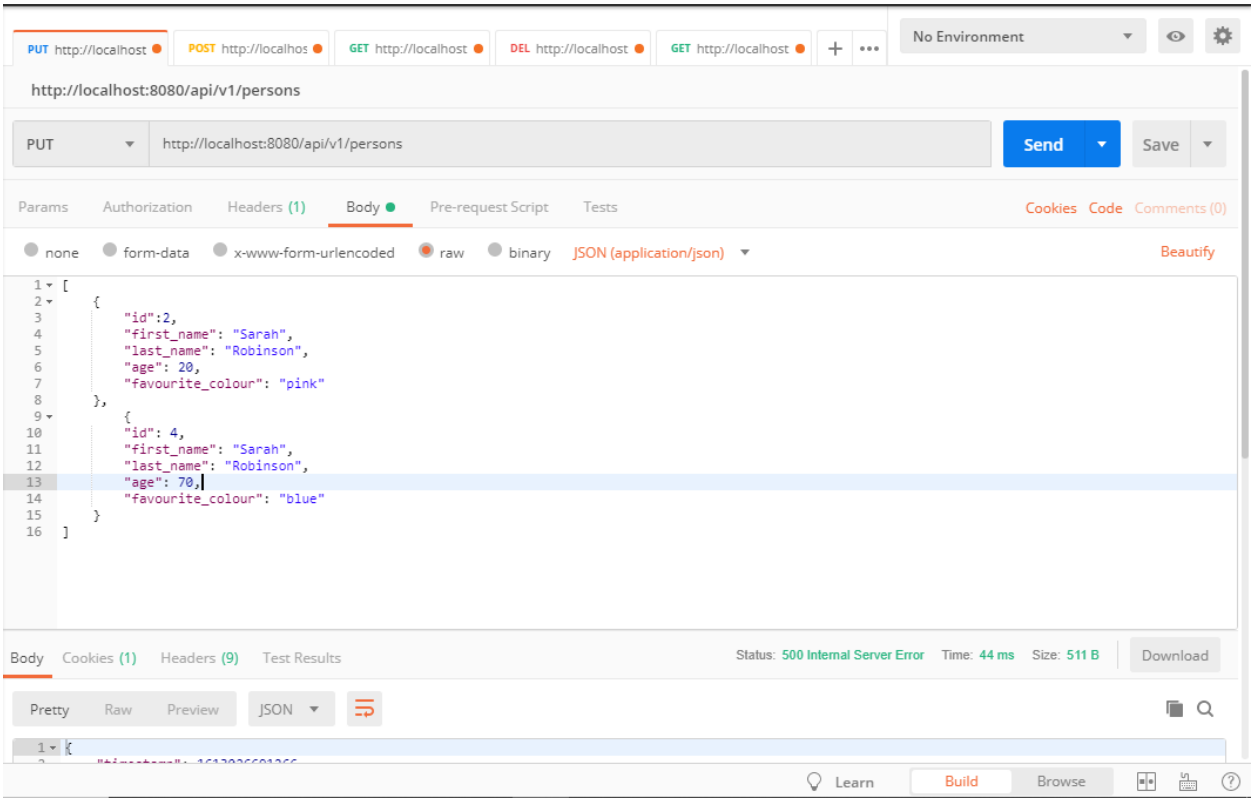

Figure 9 PUT Request http://localhost:8080/api/v1/persons

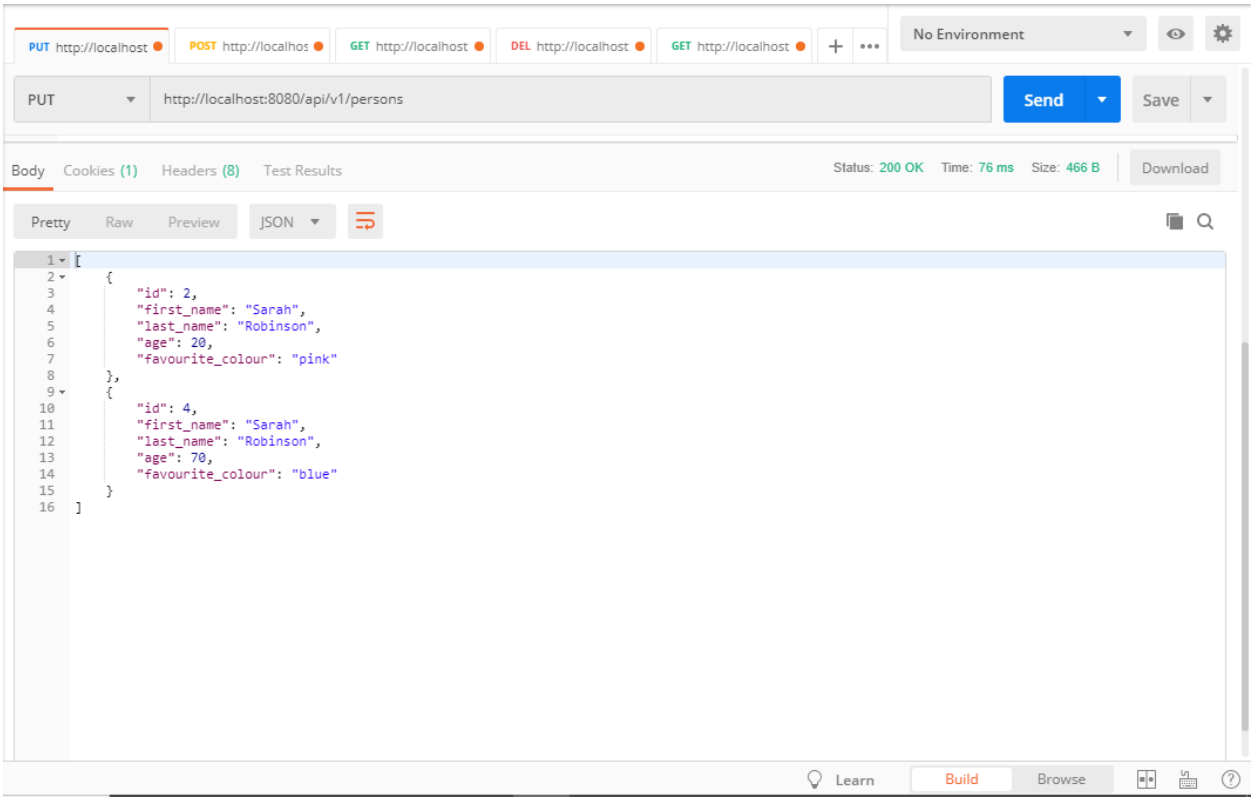

Figure 10 *PUT Response* http://localhost:8080/api/v1/persons

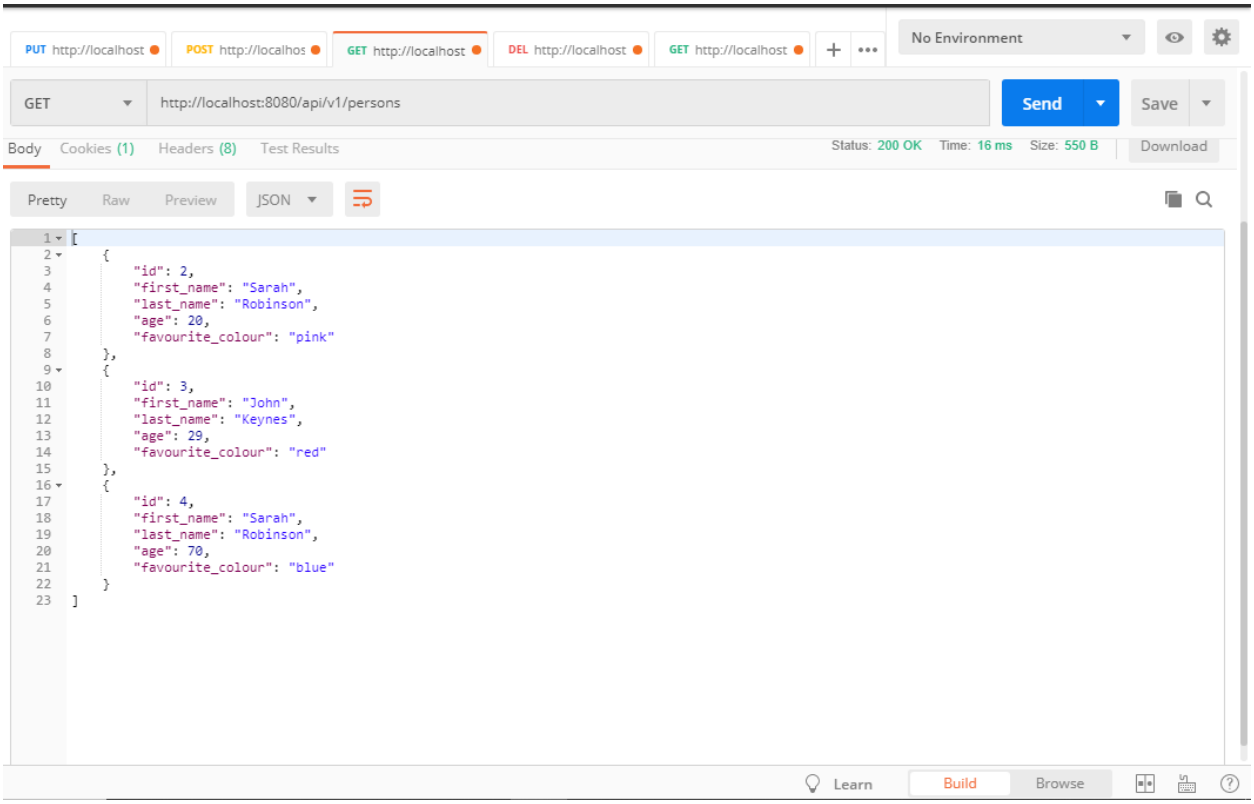

Figure 11 *GET* http://localhost:8080/api/v1/persons