Spring Boot Microservices API Test using Postman

In this lab, Let us test a spring boot microservice that can service the following API requests: GET, POST, PUT DELETE API requests to perform CRUD (Create Read Update Delete) operations.

How to test the API end points with HTTP requests?

Use Postman !

Open a File explorer to navigate to the directory

C:\apitestlabs\customerservice

This directory contains a simple spring boot customer microservice that handles the following requests:

GET request to get all customers
URL: http://localhost:8090/api/customers

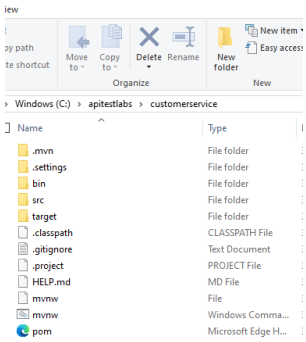
GET request to get a customer using customer id
URL: http://localhost:8090/api/customers/{id}
id is the template variable
For example http://localhost:8090/api/customers/2 will return customer data for a customer id value 2

POST request to create a new customer
URL: http://localhost:8090/api/customers
Post a Json data via request body

PUT request to update an existing customer with an id
URL: http://localhost:8090/api/customers/{id}
id is the template variable
For example http://localhost:8090/api/customers/2

DELETE request to delete an existing customer with an id
URL: http://localhost:8090/api/customers/{id}
id is the template variable
For example http://localhost:8090/api/customers/2

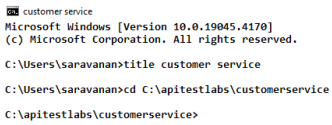
1. Open a File Explorer, Navigate to **C:\apitestlabs\customerservice**



Review the resources in this microservice directory.
You may open this project in Eclipse and review the source code and configuration.

2. Start your eclipse and go to your lab workspace: **C:\apitestlabs\labworkspace**

   cd C:\apitestlabs\customerservice



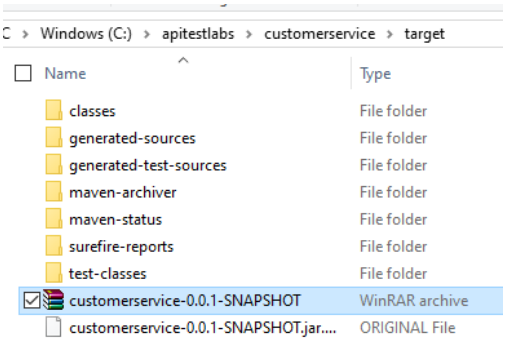
3. Let us build the service using maven. Type the following command in the command window

   mvn install

Make sure you get the Build Successful message.

4. Use the file explorer to navigate to **C:\apitestlabs\customerservice\target**

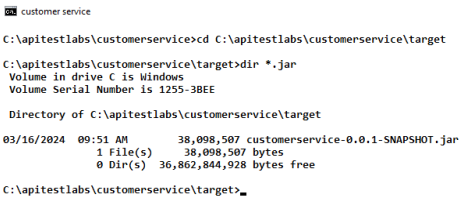
What is in this directory? Can you view the application jar file as shown in the following screen shot?



This is the spring boot application that can be executed using the java command.

5. In the Command window, type the command change directory to **C:\apitestlabs\customerservice\target**

   cd target



6. Run this application using the following command

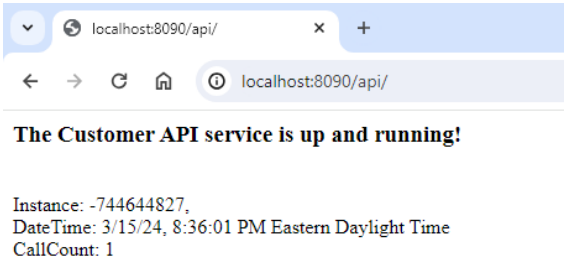
   java -jar customerservice-0.0.1-SNAPSHOT.jar

This application launches a spring boot microservice listening at port 8090.

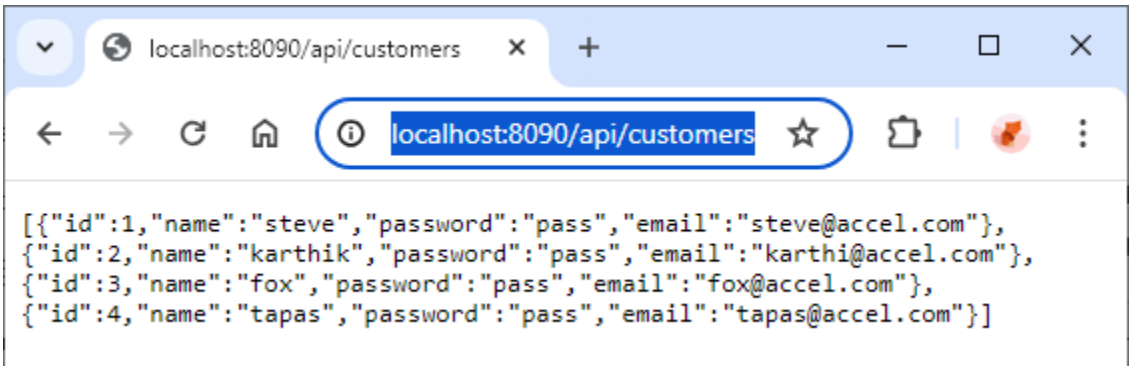
Review the application log messages.

7. Let us use a web browser and test the following end points. Open browser and go to the following URLs one by one and test.

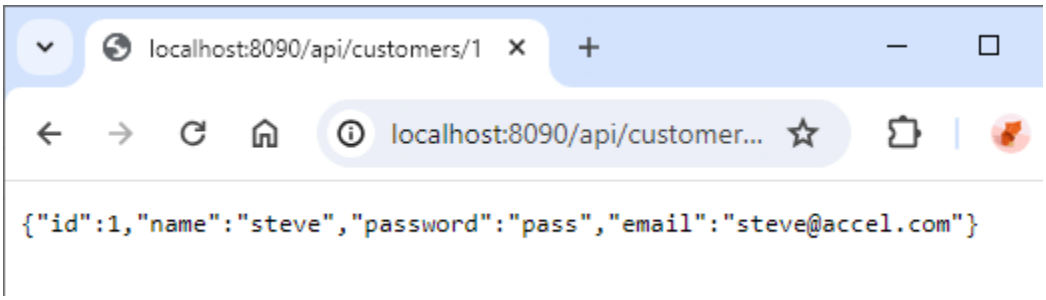
http://localhost:8090/api



http://localhost:8090/api/customers

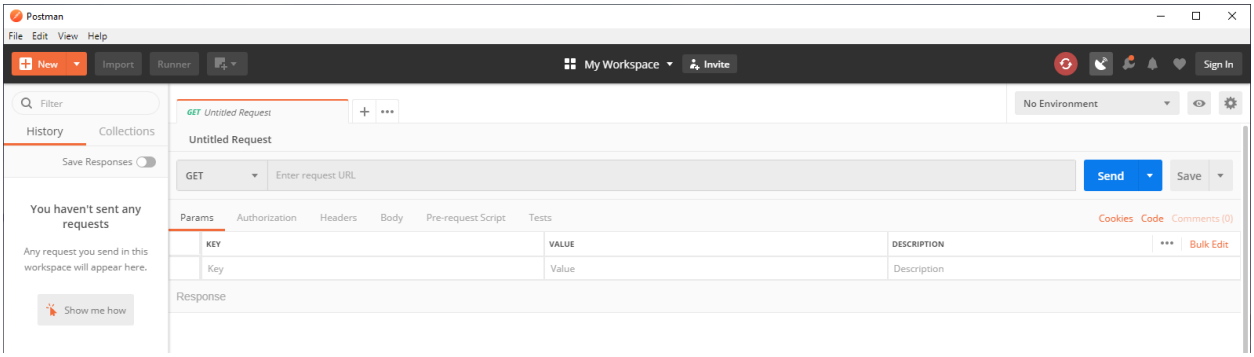


http://localhost:8090/api/customers/1



Part 2 Use Postman to REST end points and CRUD Operations

8. Launch the postman application. Click on the postman in your machine desktop to launch a postman instance.



9. Let us create a new GET request to send a request to this URL

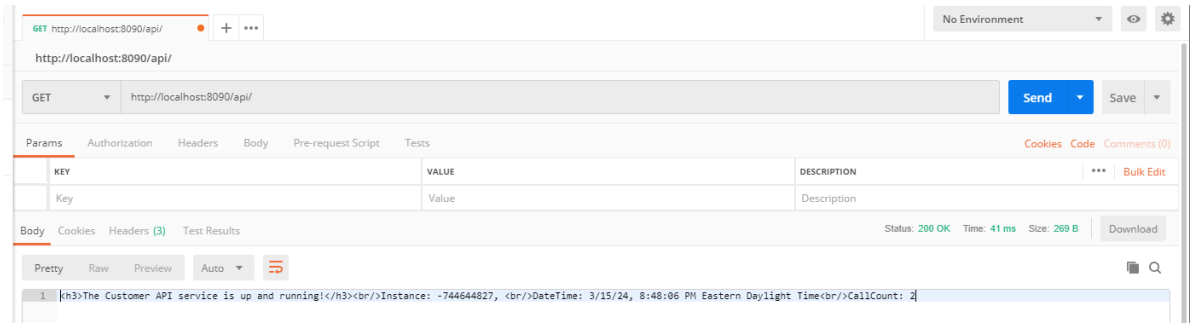
   http://localhost:8090/api/

   In Postman's New request, use the following data values and then click **send.**

   Method: GET

   URL: http://localhost:8090/api/

   Click on **send** a few times.



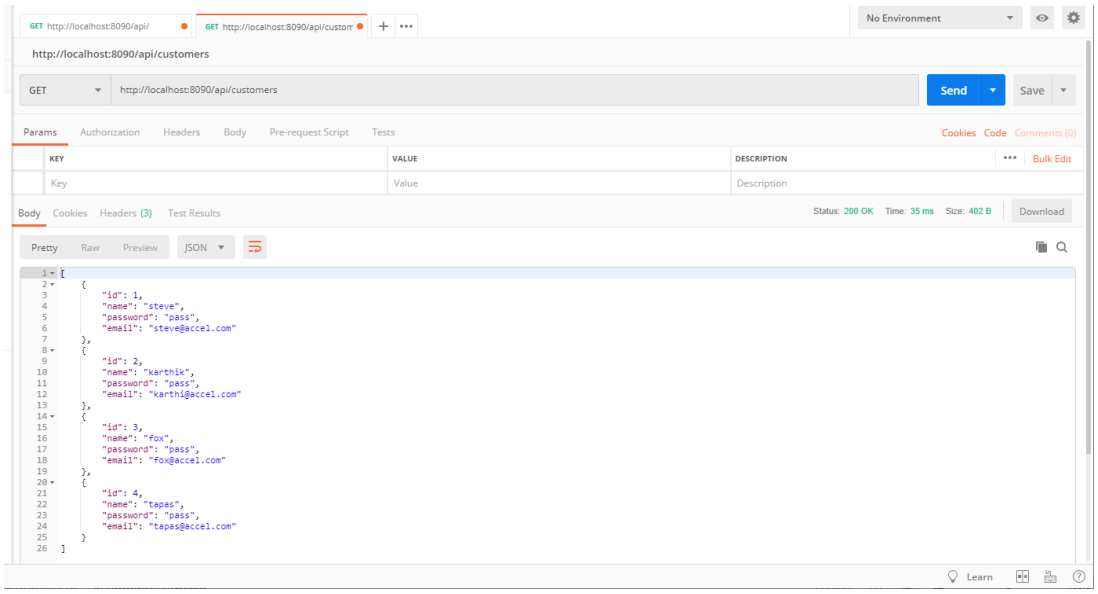
Can you see the API call count in the response body?

10. Let us get all customers. Create a new postman request to this URL

    http://localhost:8090/api/customers

    Method: **GET**
    URL: **http://localhost:8090/api/customers**

    Click on **send. Can you get the list of customers from the API?**



11. We want to create a postman collection with all requests. Let is save every request under test to this collection and call the collection as "**Customers API Tests**." Click on the **Save** button (next to Send) and **save as**

Use the following details and save this request to a collection:

Customers API Tests

Request name

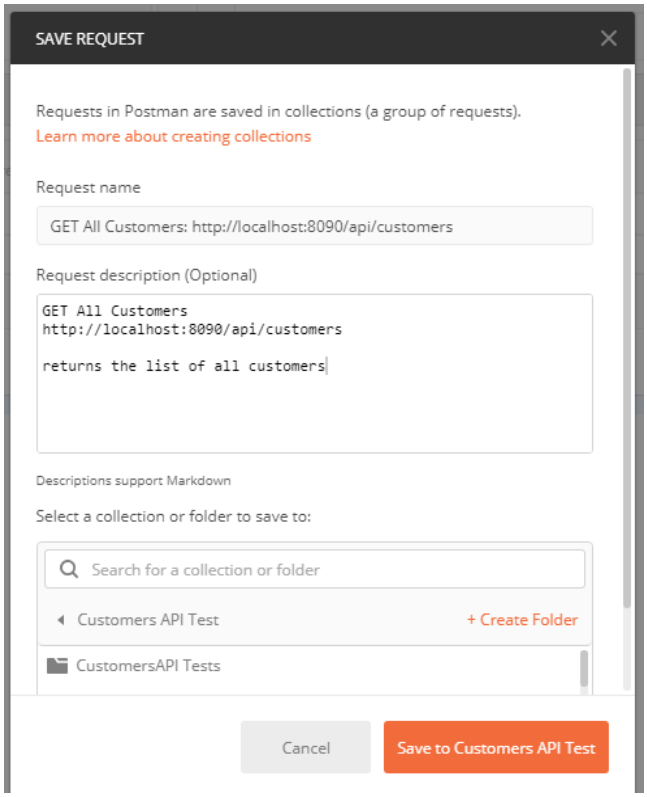
GET All Customers: http://localhost:8090/api/customers

Request description (Optional)

GET All Customers
http://localhost:8090/api/customers
returns the list of all customers

Select a collection or folder to save to:

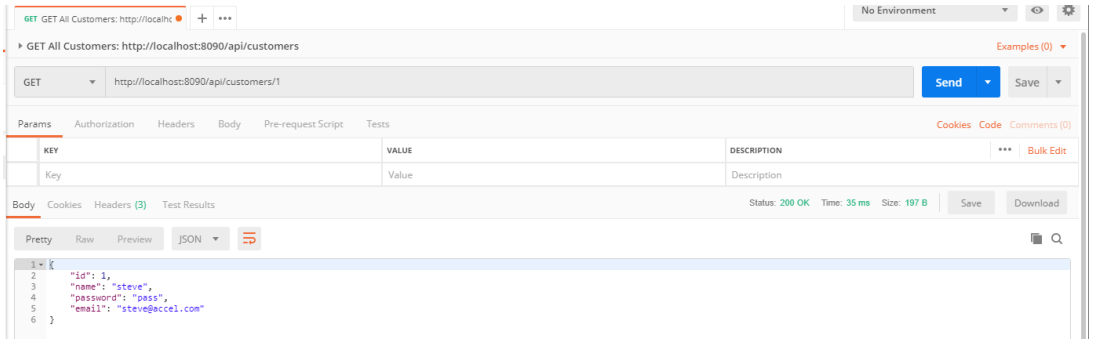
Customers API Tests



12. Let us get a customer data using customer id. Create a new postman request with the following request data:

    Method: **GET**
    URL: **http://localhost:8090/api/customers/1**

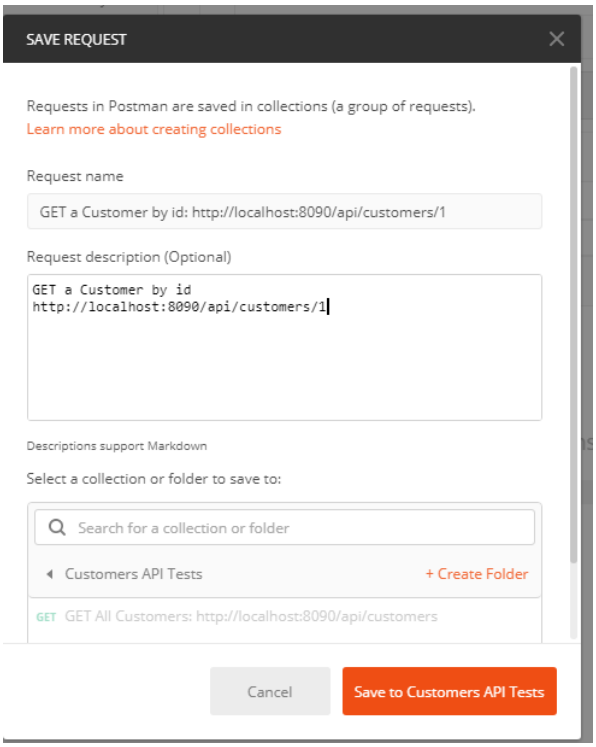
    Click on **send**.



Inspect the response Body, Headers, Status, Size, and any other details.

What is the HTTP status code?

13. Save this request to **Customers API Tests** collection.

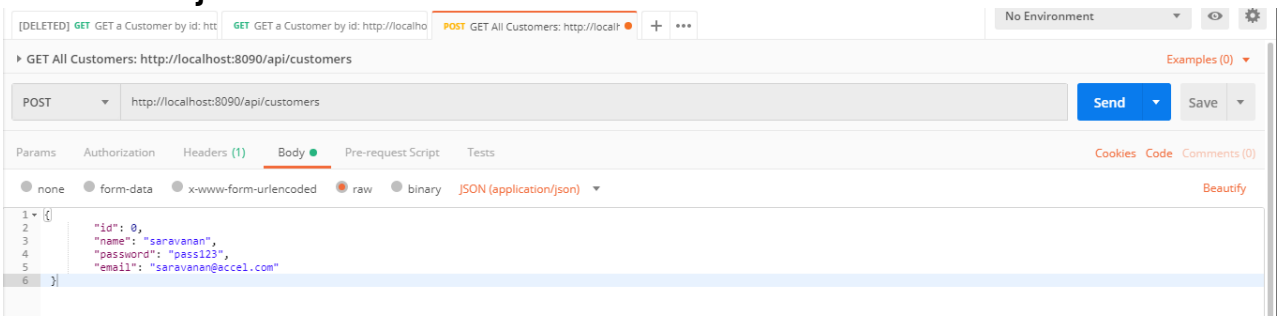


14. Let us create a new customer and post the data to API. Create a new postman request with following details.

Method: **POST**
URL: **http://localhost:8090/api/customers**
Request Body:
raw **JSON (application/json)**

Pay load data:

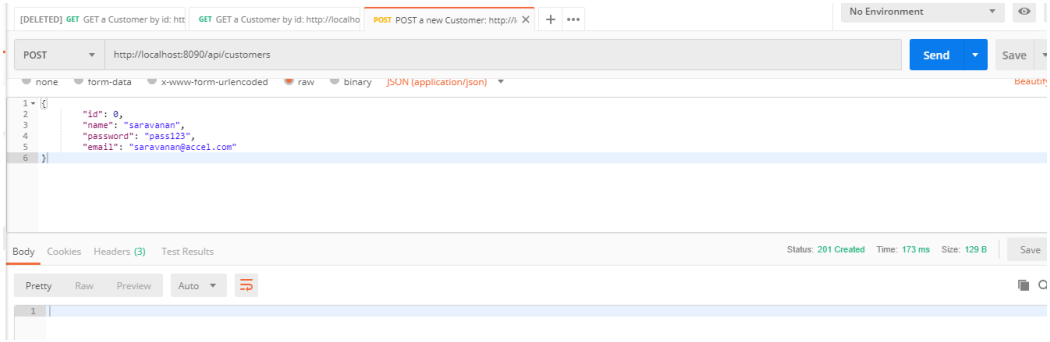
```
{
    "id": 0,
   "name": "saravanan",
    "password": "pass123",
    "email": "saravanan@accel.com"
}
```



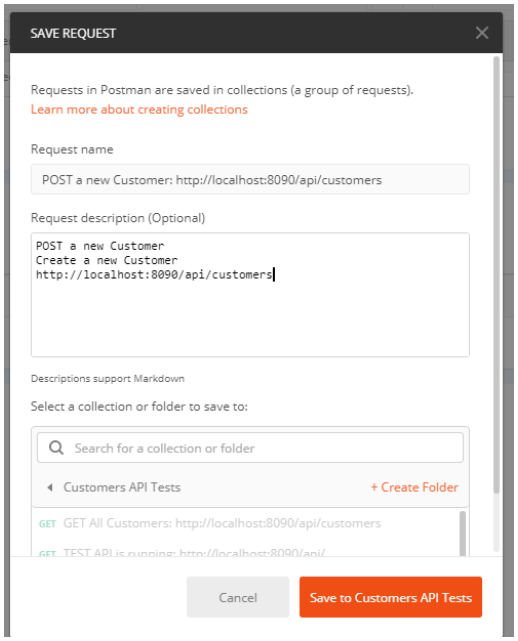
Click on **send.** What is the response status code? **201 Created?**

Where is the new Customer created?

How to find the location of this customer? Click on response Headers to see this.



15. Save the previous POST request to **Customers API Tests** collection.

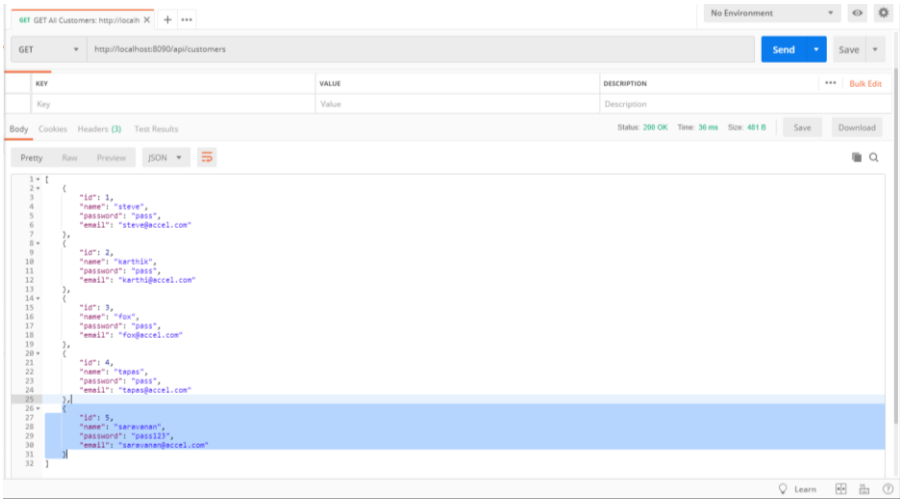


16. Let us test the newly created Customer data using GET requests. Create new postman GET requests to the following URL and send.

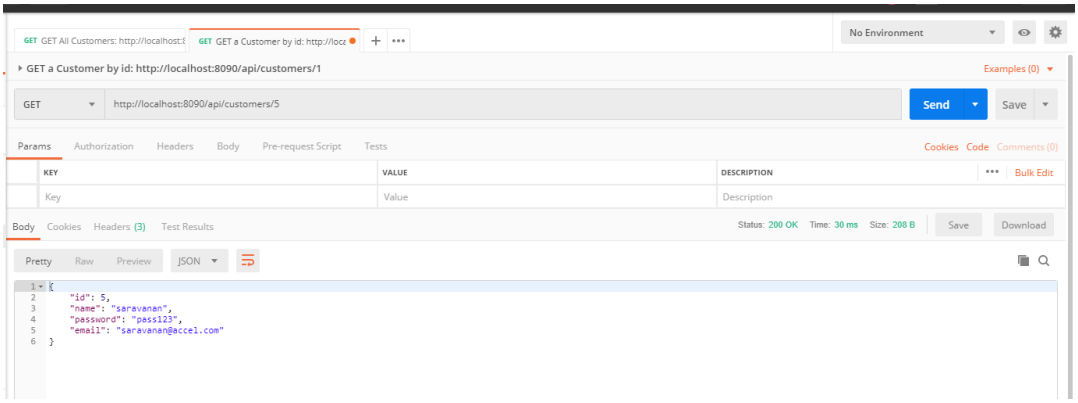
    Method: **GET**
    URL: http://localhost:8090/api/customers

    Method: **GET**
    URL: http://localhost:8090/api/customers/5

    Can you get the new Customer data?
    http://localhost:8090/api/customers



    http://localhost:8090/api/customers/5

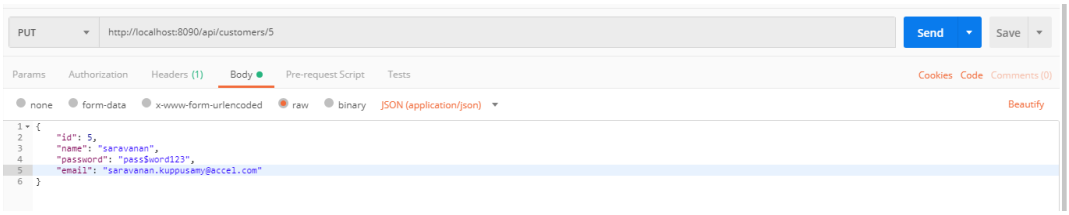


17. Let us update the new Customer data. We will use PUT request with a request body data with modified password and email data. Create a new postman PUT request with the following data.

Method: **PUT**
URL: **http://localhost:8090/api/customers/5**

Request Body:
raw **JSON (application/json)**
Pay load data:

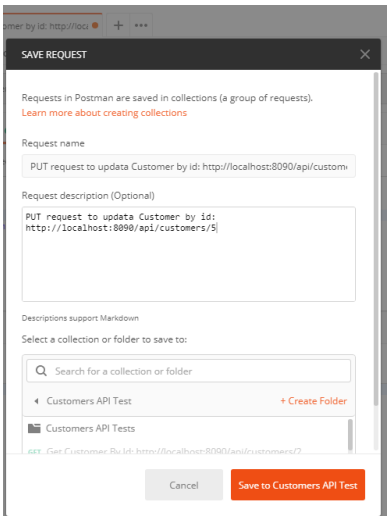
```
{
    "id": 5,
    "name": "saravanan",
    "password": "pass$word123",
    "email": "saravanan.kuppusmy@accel.com"
}
```



Click on **send.**

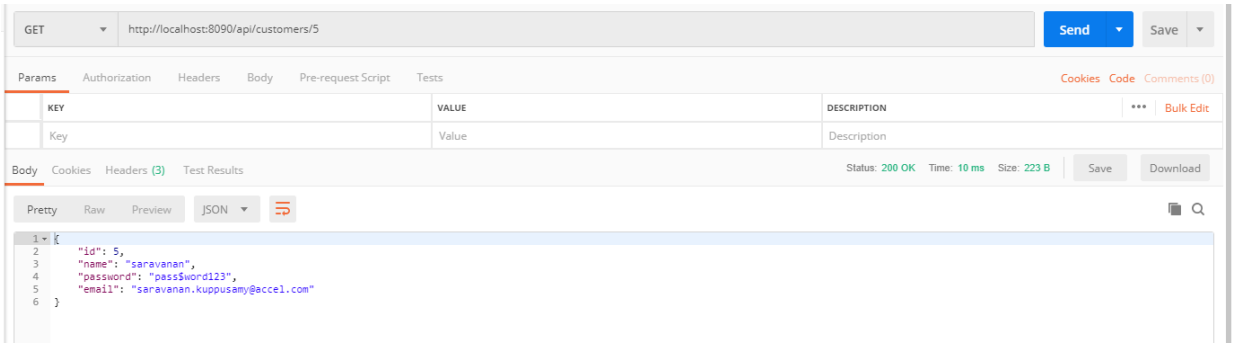
What is the response code?

18. Save the previous PUT request to **Customers API Tests** collection



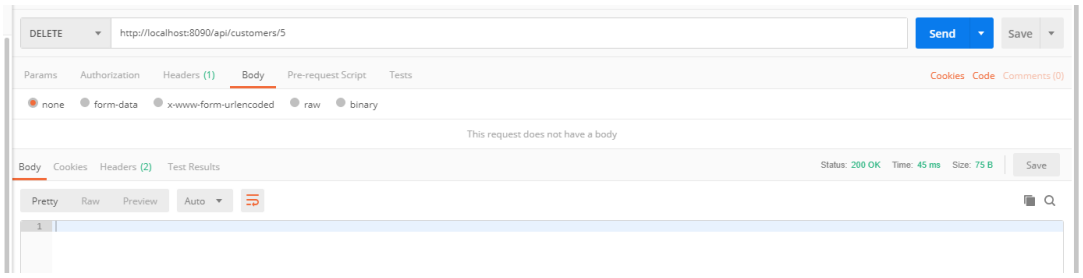
19. Let us verify the updated Customer data using GET requests. Use the following end points and GET method invocations to verify the modified customer data.

    http://localhost:8090/api/customers
    http://localhost:8090/api//customers/5



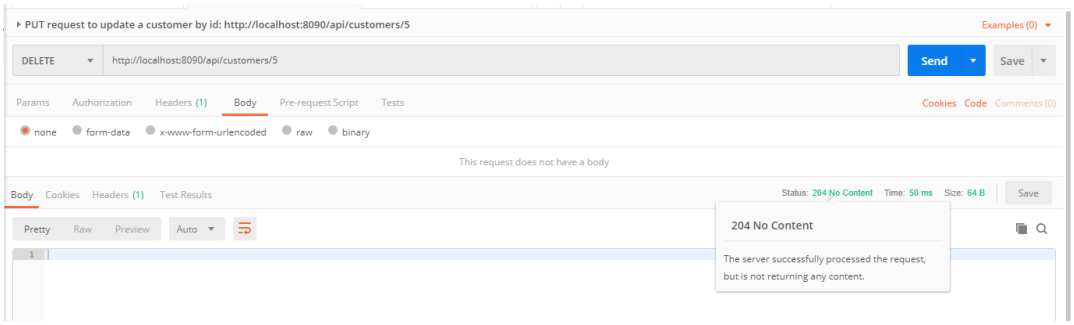
20. Let us DELETE an existing Customer using the customer id. Create a new postman DELETE request with the following data.

    Method: **DELETE**
    URL: http://localhost:8090/api//customers/5
    Body: none

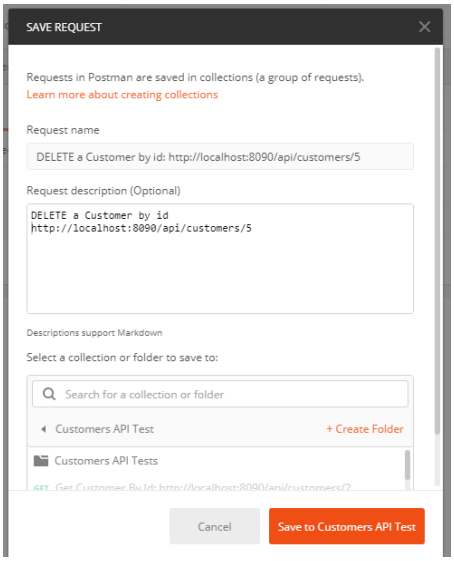


Click on **send.**

What is the response?



21. Save the previous DELETE request to **Customers API Tests** collection.

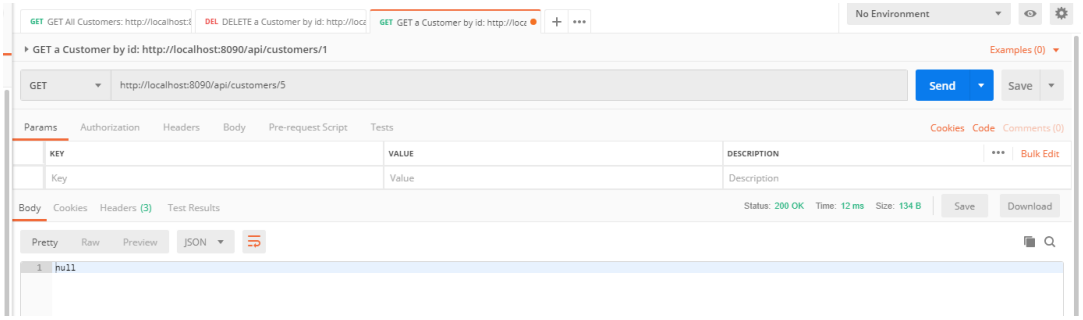


22. Now, let us use the GET requests to verify that customer data is no longer available. Use the following URLs to see postman responses

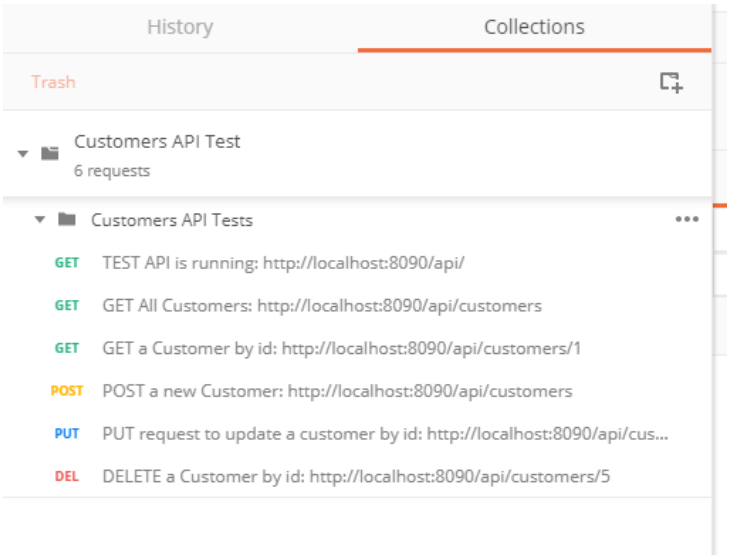
    Method: **GET**
    URL: http://localhost:8090/api/customers

    Method: **GET**

    URL: http://localhost:8090/api/customers/5

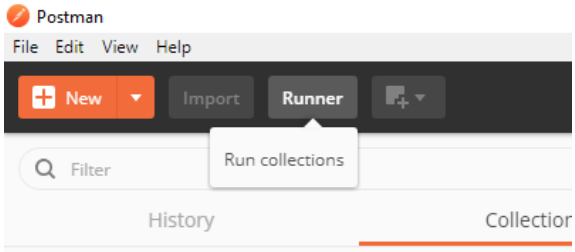


23. Review the postman collection: **Customers API Tests**. Run the requests with new data, modify the data and delete and test API.

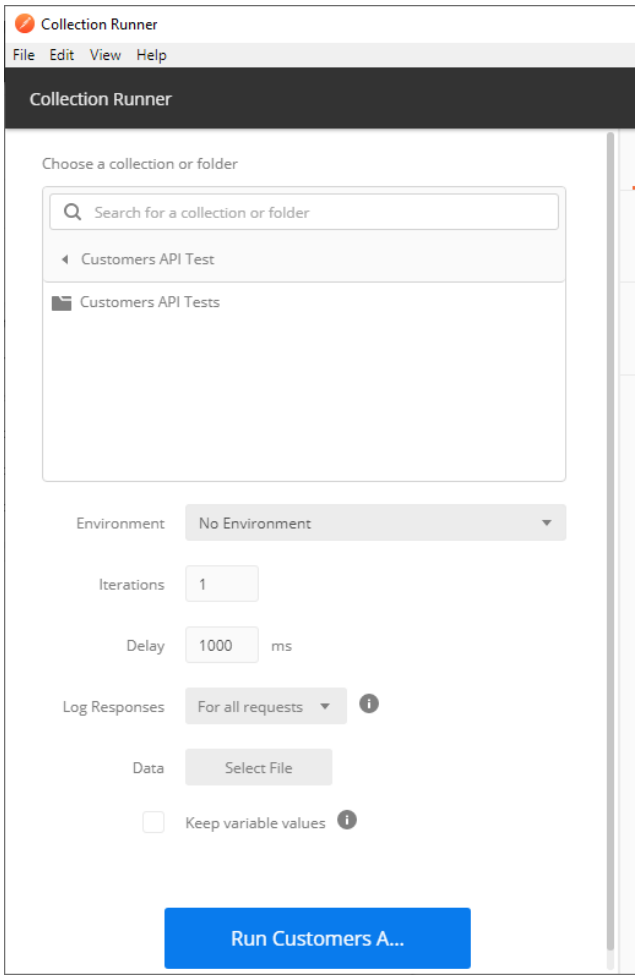


Part 3 Run the postman collection

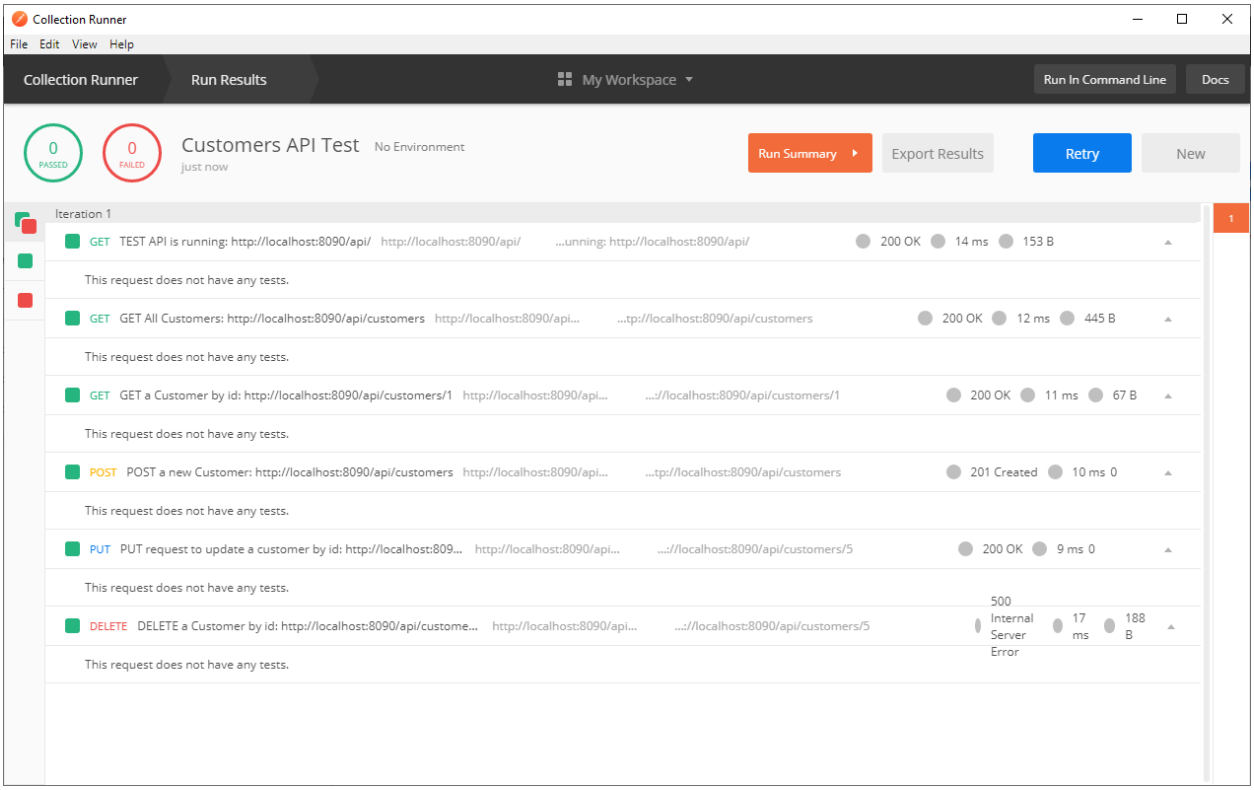
24. Let us run the postman collection: **Customers API Tests**. In postman, click on **Runner → Run Collections**



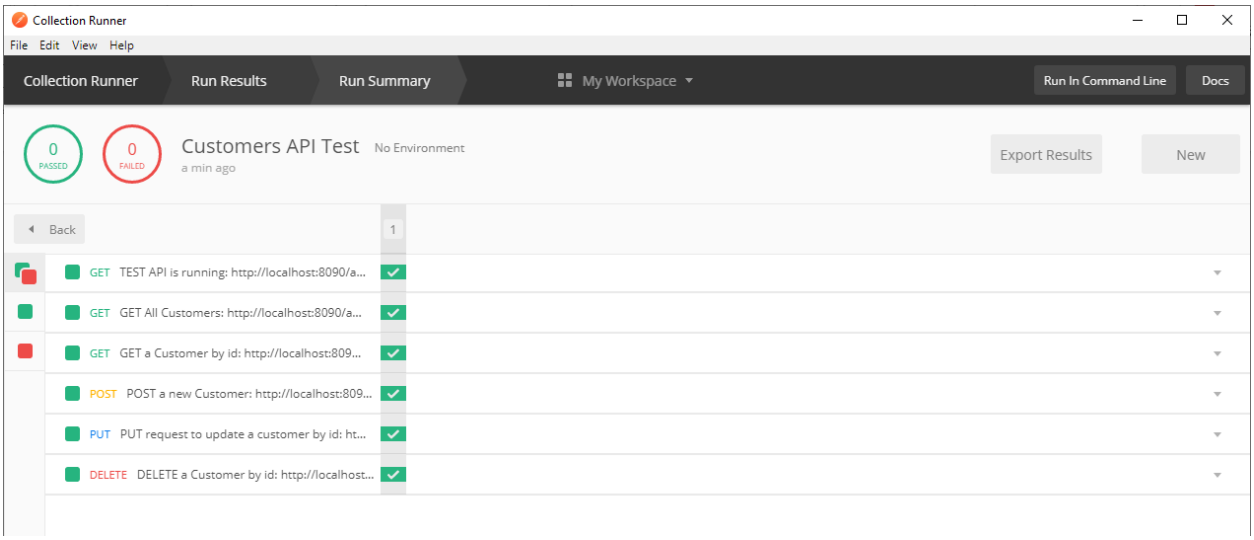
25. In the collection runner dialog, use the following configuration with a delay of **1000 ms**, all requests log responses and **Run Customers API Tests collection**.



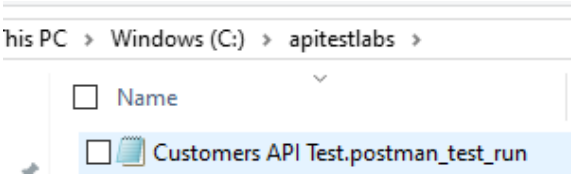
26. See the results and review the results.



27. Click on **Run Summary** and review.



28. Click on **Export Results** and export the results to **C:\apitestlabs** directory.



By default it will export the results to a file called **Customers API Test.postman_test_run.json**

29. Use a text editor to open the file and review the file contents.