Impressions

Postman is used in this assignment to see if an API is working as expected - end-to-end.

Since I have used Postman only to execute authenticated requests to test APIs written for personal projects as well as work-related APIs, I did not have a use for the API Testing feature of it, since the end-to-end tests would be handled from inside the applications testing framework. So, I had to familiarize myself with this feature and underlying JavaScript API.

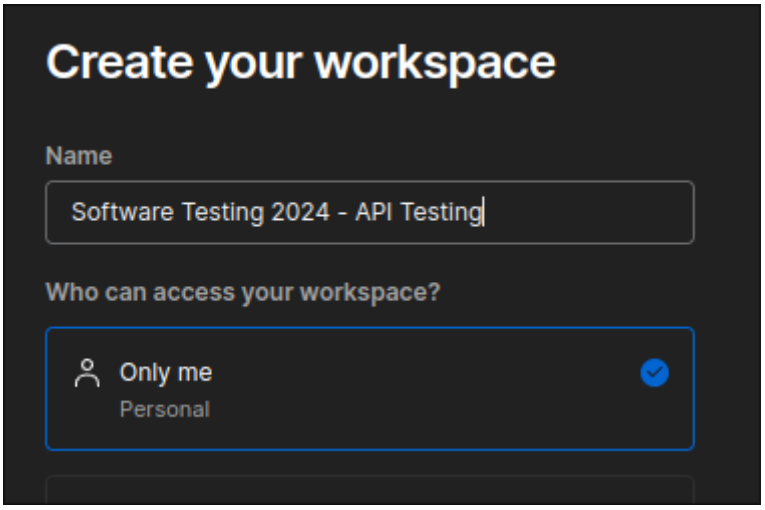
Conveniently, Postman offers pre-made snippets to get started, as shown as on the page 11 from the 08-02 API Testing practical lecture slide. Those are in the current version moved inside the Scripts tab.

Preparing the Project

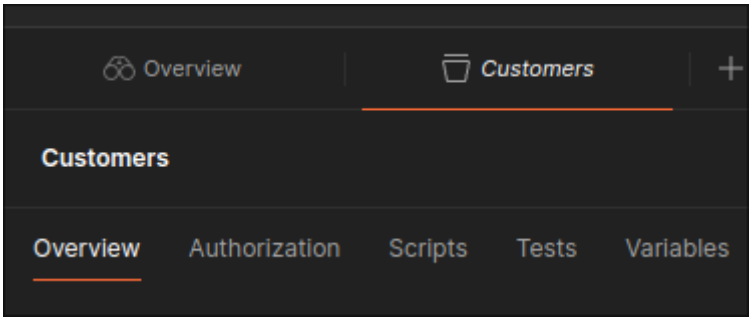
Installing Postman was on macOS, Windows, and Linux no issue.
Postman requires you to login. I already had an account due to previous applications, so it did not bother me.

Since I have used Java 19 on my macOS machine, I could observe some compilation errors in `PhoneNumberValidatorUtils` when switching back to Java 11. Previously, all tests did pass. Inside the fix-commit, the tests pass without issue as well.

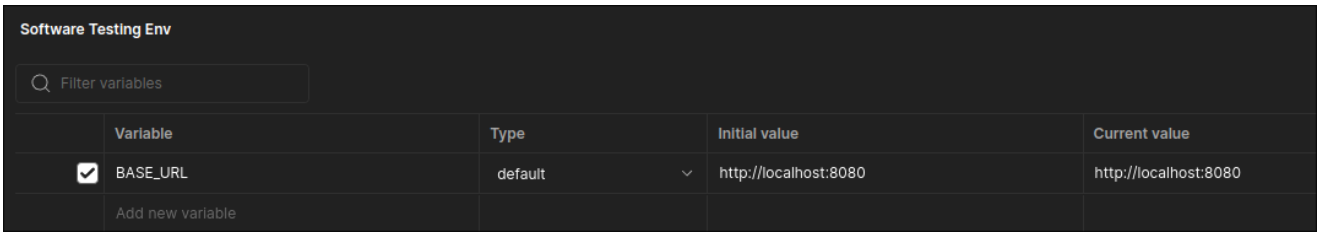
Next, I created a workspace for the project.



After that, a collection has been created.



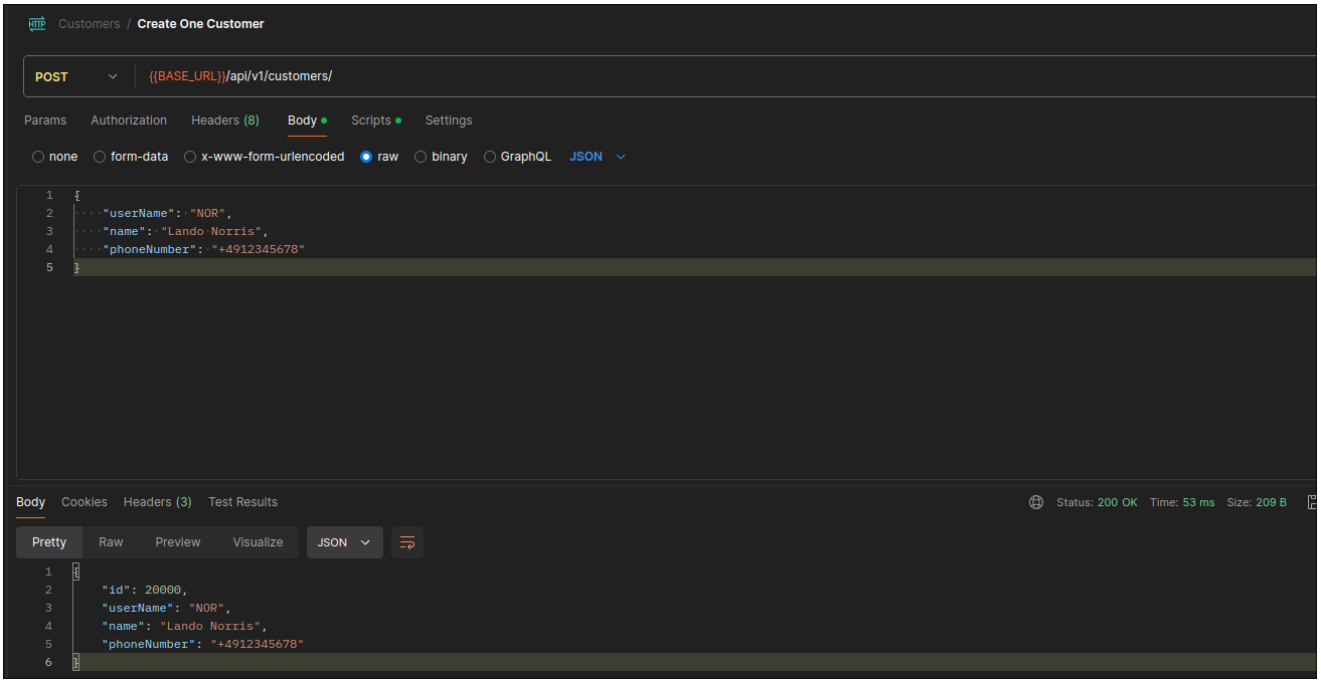
Also, an environment is added to keep the requests uniform.



Getting familiar with the API

Creating a customer

First of all, I have struggled to figure out, which JSON format is used for the keys. Since `CustomerInDTO.java` provides insight which parameters are expected from the POST request. Therefore, I just used the corresponding variable names as keys for the JSON request body.



This is weird, because I thought, that `/api/v1/customer-registration` is responsible for this.
Calling `/api/v1/customer-registration` with the same body results correctly in an error:

POST Customer Registratio ● | GET List all customers | GET Find One Customer ● | D

Customers / **Customer Registration**

POST {{BASE_URL}}/api/v1/customer-registration/

Params | Authorization | Headers (8) | Body ● | Scripts | Settings

○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL

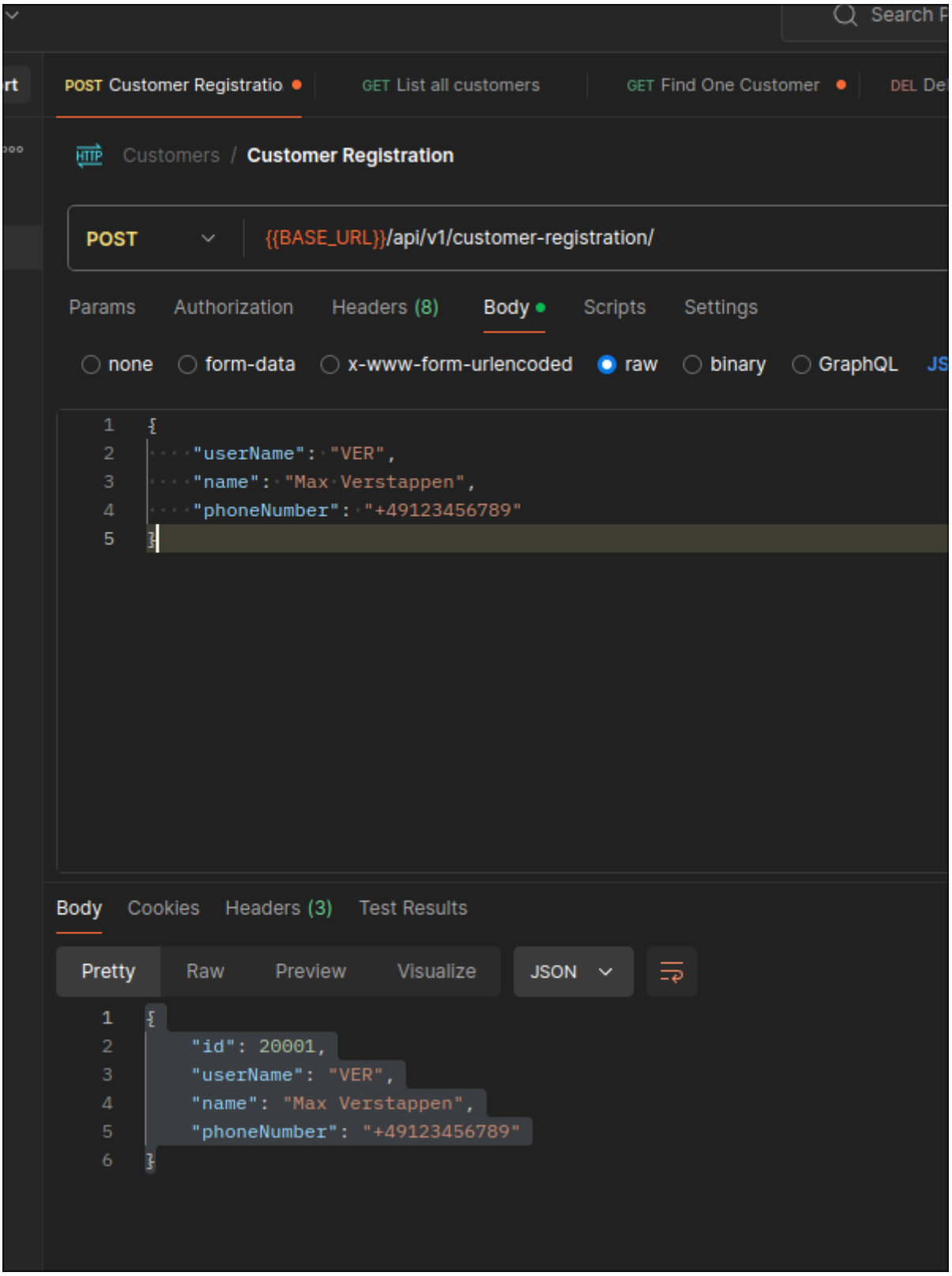
```
1 {
2     "userName": "NOR",
3     "name": "Lando Norris",
4     "phoneNumber": "+4912345678"
5 }
```

Body | Cookies | Headers (4) | Test Results

Pretty | Raw | Preview | Visualize | JSON

```
1 {
2     "timestamp": "2024-06-04T12:21:53.151+0000",
3     "status": 500,
4     "error": "Internal Server Error",
5     "message": "You are already registered",
6     "path": "/api/v1/customer-registration/"
7 }
```

Creating a new user, this endpoint works as well:



Finding one Customer

Using the ID from the previous creation response, I was able to find the customer.

Customers / **Find One Customer**

GET {{BASE_URL}}/api/v1/customers/:id

Params • Authorization Headers (6) Body Scripts Settings

Query Params

	Key	Value
	Key	Value

Path Variables

	Key	Value
	id	20000

Body Cookies Headers (3) Test Results

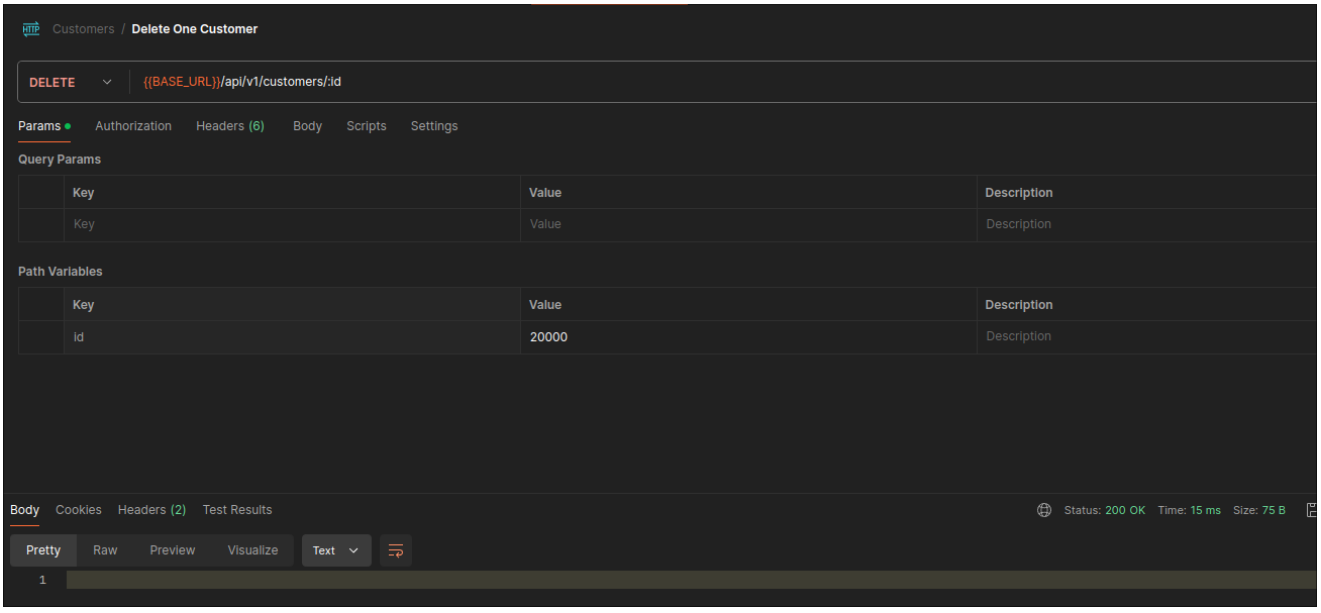
Pretty Raw Preview Visualize JSON

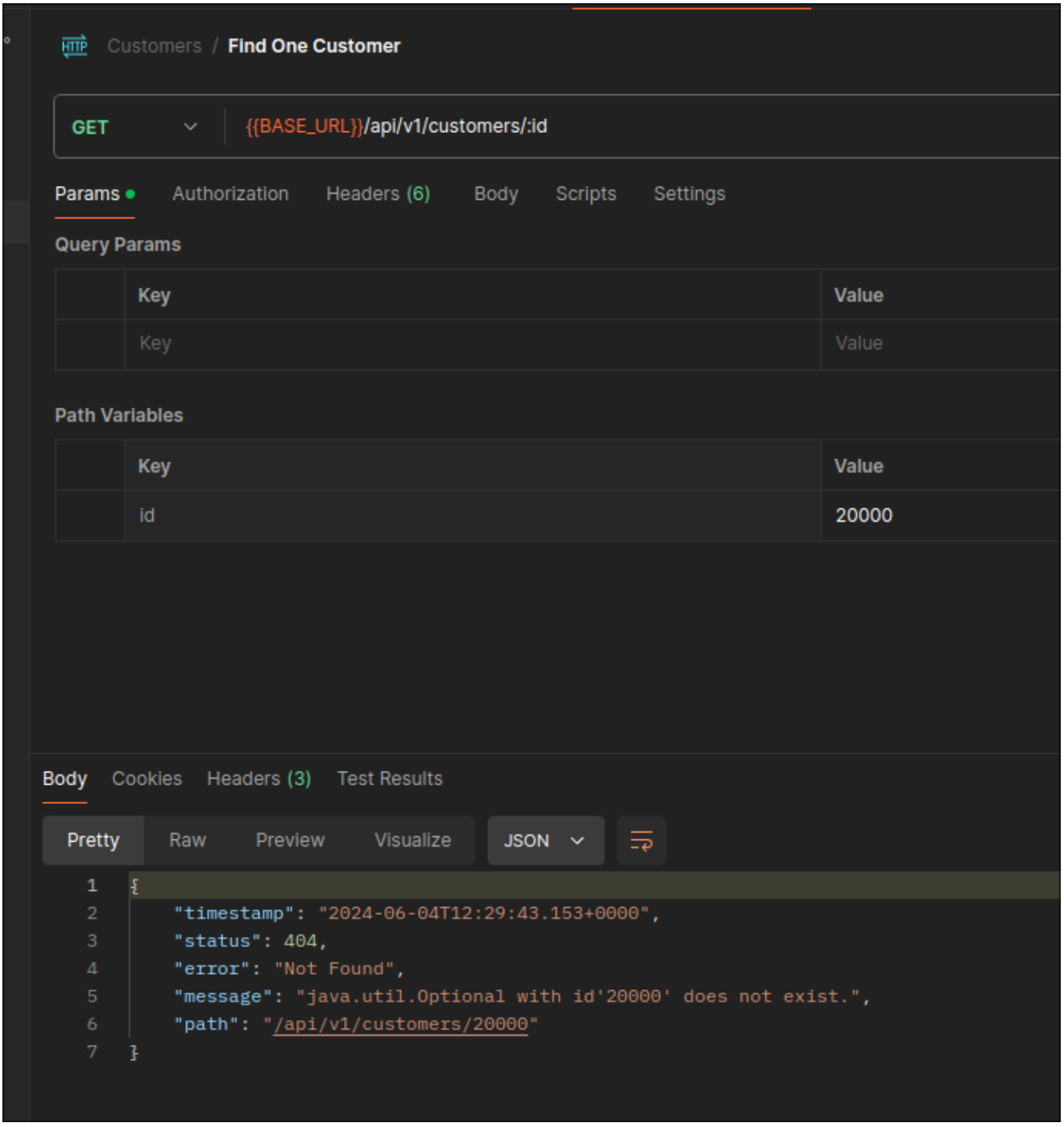
```
1 {
2     "id": 20000,
3     "userName": "NOR",
4     "name": "Lando Norris",
5     "phoneNumber": "+4912345678"
6 }
```

The endpoint also works for `20001`, for user `VER`.

Deleting a customer

Deleting a customer works.





Issues with creating Customers

As shown above, there are two endpoints available for creating customers, namely POST `/api/v1/customers` and POST `/api/v1/customer-registration`.

Taking a look at the corresponding services, we can observe, that the former is only checking if a phone number is already taken, and the latter is providing a more thorough validation logic.

```java
        customerRepository.deleteById(customerId);
    }

    @Transactional(rollbackFor = Exception.class)  4 usages  emmanuelcharleson.dapaah
    @Override
    public Customer addCustomer(Customer customer) {
        Optional<Customer> existsPhoneNumber = customerRepository.selectCustomerByPhoneNumber(customer

        if (existsPhoneNumber.isPresent()) {
            throw new BadRequestException(
                    "Phone Number " + customer.getPhoneNumber() + " taken");
        }
        return customerRepository.save(customer);
    }

    @Transactional(rollbackFor = Exception.class)  3 usages  emmanuelcharleson.dapaah
    @Override
    public Collection<Customer> saveAll(List<Customer> customers) {
        return StreamSupport.stream(customerRepository.saveAll(customers).spliterator(),  parallel: false
```

```java
    @Transactional(rollbackFor = Exception.class)  4 usages  emmanuelcharleson.dapaah
    public Customer registerNewCustomer(Customer customer) {
        Optional<Customer> existsPhoneNumber = customerRepository.selectCustomerByPhoneNumber(cust

        //TODO: Validate customer phone number

        if (existsPhoneNumber.isPresent()) {
            Customer existingCustomer = existsPhoneNumber.get();
            if (existingCustomer.getName().equals(customer.getName())){
                throw new IllegalStateException("You are already registered");
            }
            throw new BadRequestException(
                    "Phone Number " + customer.getPhoneNumber() + " taken");
        }

        return customerRepository.save(customer);
    }
}
```

Therefore it would make sense to eliminate one of those options. I decided to refactor the POST POST `/api/v1/customers` endpoint, to use the same functionality as POST `/api/v1/customer-registration`.

But before we do that, we need to write some tests to validate the theory.

Writing tests in Postman

`/api/v1/customers/list` provides us with a seeded database from which we can test the GET and DELETE endpoints for `/api/v1/customers/<id>`.

To fetch the customer list, we create a pre-request script. Using the provided snippets for accessing the collection variable `BASE_URL`, which works in the URL field of postman, results in `undefined`. [For some reason, the `environment` key has to be used in order to access this variable.]

While Postman Scripts indicates variable types, it does not support Typescript.

```typescript
const customerList = response.json() as any[]
```

Listing all customers

Creating a post response test script, we gain the following results:

Script:

```typescript
pm.test("Status code is 200", function() {
    pm.response.to.have.status(200);
});
```

```
const resp = pm.response.json()

pm.test("Response is an array", function() {
    pm.expect(resp).to.be.an('array');
});

resp.forEach((item, index) => {
    pm.test(`Item ${index + 1} has required fields`, function() {
        pm.expect(item).to.have.property('id');
        pm.expect(item.id).to.be.a('number');
        pm.expect(item).to.have.property('userName');
        pm.expect(item.userName).to.be.a('string');
        pm.expect(item).to.have.property('name');
        pm.expect(item.name).to.be.a('string');
        pm.expect(item).to.have.property('phoneNumber');
        pm.expect(item.phoneNumber).to.be.a('string');
    });
})
```

Customers / **List all customers**

GET {{BASE_URL}}/api/v1/customers/list

Params Authorization Headers (6) Body Scripts Settings

Pre-request

Post-response

```
const resp = pm.response.json()

pm.test("Response is an array", function () {
    pm.expect(resp).to.be.an('array');
});

resp.forEach((item, index) => {
    pm.test(`Item ${index + 1} has required fields`, function () {
        pm.expect(item).to.have.property('id');
        pm.expect(item.id).to.be.a('number');
        pm.expect(item).to.have.property('userName');
        pm.expect(item.userName).to.be.a('string');
        pm.expect(item).to.have.property('name');
        pm.expect(item.name).to.be.a('string');
        pm.expect(item).to.have.property('phoneNumber');
        pm.expect(item.phoneNumber).to.be.a('string');
    });
})
```

(parameter) item: any

Body Cookies Headers (3) Test Results (20001/20001)

Pretty Raw Preview Visualize JSON


```
[
    {
        "id": 9890,
        "userName": "f9890",
        "name": "19890",
        "phoneNumber": "+490009890"
    },
    {
        "id": 17418,
        "userName": "f17418",
        "name": "117418",
        "phoneNumber": "+4900017418"
```


Deleting a customer

Pre Request Tests

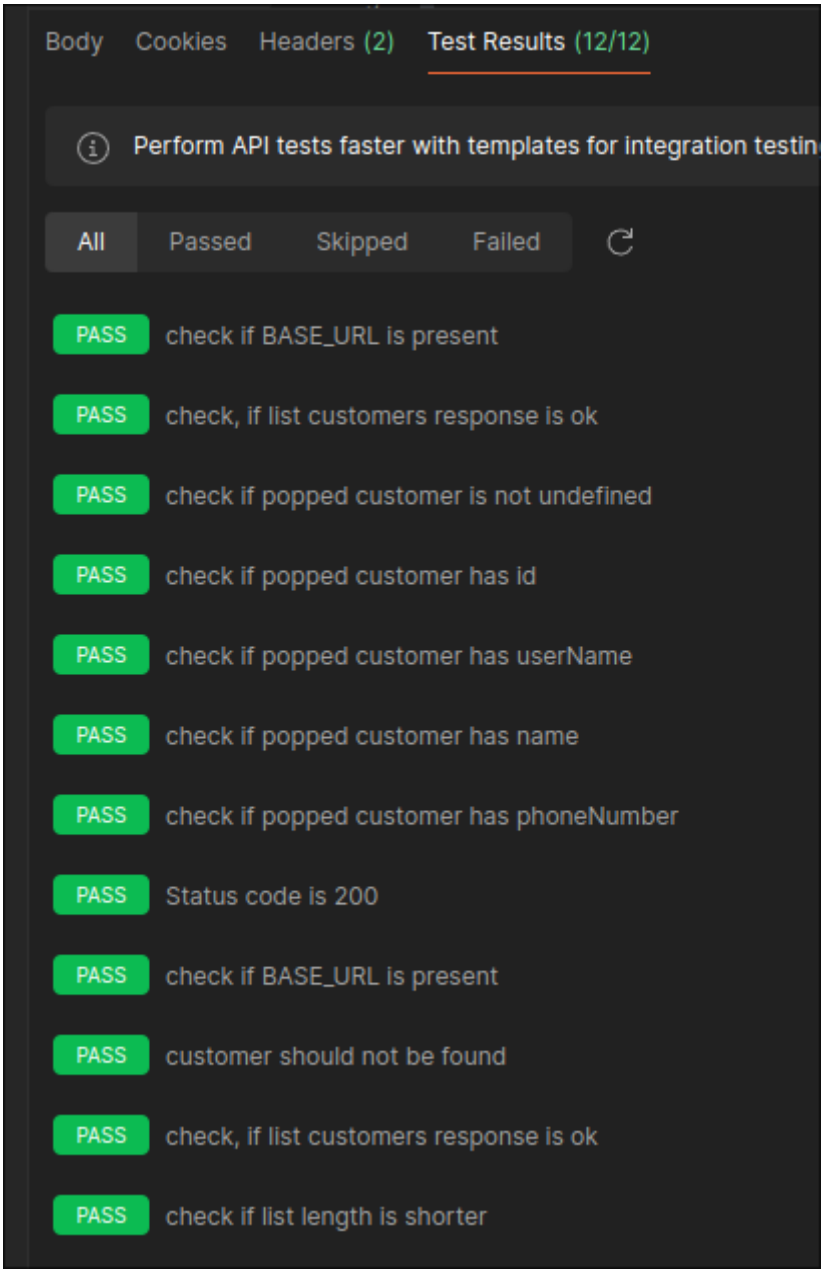
```
const BASE_URL = pm.environment.get("BASE_URL");

pm.test('check if BASE_URL is present', () => {
    pm.expect(BASE_URL).to.not.be.undefined
    pm.expect(BASE_URL).to.be.a('string')
})

pm.sendRequest(`${BASE_URL}/api/v1/customers/list`, function(err,
response) {
    pm.test('check, if list customers response is ok', () => {
        pm.expect(response.status).to.be.eql("OK")
```

```
    })
    const customerList = response.json()
    pm.test('check if list length corresponds to 19999', () => {
        pm.expect(customerList.length).to.be.eql(19999)
    })
    const testCustomer = customerList.pop()
    pm.test('check if popped customer is not undefined', () => {
        pm.expect(testCustomer).to.be.not.undefined
        pm.expect(testCustomer).to.be.a('object')
    })
    pm.test('check if popped customer has id', () => {
        pm.expect(testCustomer.id).to.be.a('number')
    })
    pm.test('check if popped customer has userName', () => {
        pm.expect(testCustomer.userName).to.be.a('string')
    })
    pm.test('check if popped customer has name', () => {
        pm.expect(testCustomer.name).to.be.a('string')
    })
    pm.test('check if popped customer has phoneNumber', () => {
        pm.expect(testCustomer.phoneNumber).to.be.a('string')
    })

    pm.environment.set("testCustomerId", testCustomer.id);
});
```

Post Request Tests

```
pm.test("Status code is 200", function() {
    pm.response.to.have.status(200);
});
const BASE_URL = pm.environment.get("BASE_URL");

pm.test('check if BASE_URL is present', () => {
    pm.expect(BASE_URL).to.not.be.undefined
    pm.expect(BASE_URL).to.be.a('string')
})

pm.sendRequest(`${BASE_URL}/api/v1/customers/list`, function(err,
response) {
    pm.test('check, if list customers response is ok', () => {
        pm.expect(response.status).to.be.eql("OK")
    })
    const customerList = response.json()
    pm.test('check if list length is shorter', () => {

pm.expect(customerList.length).to.be.eql(pm.environment.get('previousLengt
h') - 1)
    })
```

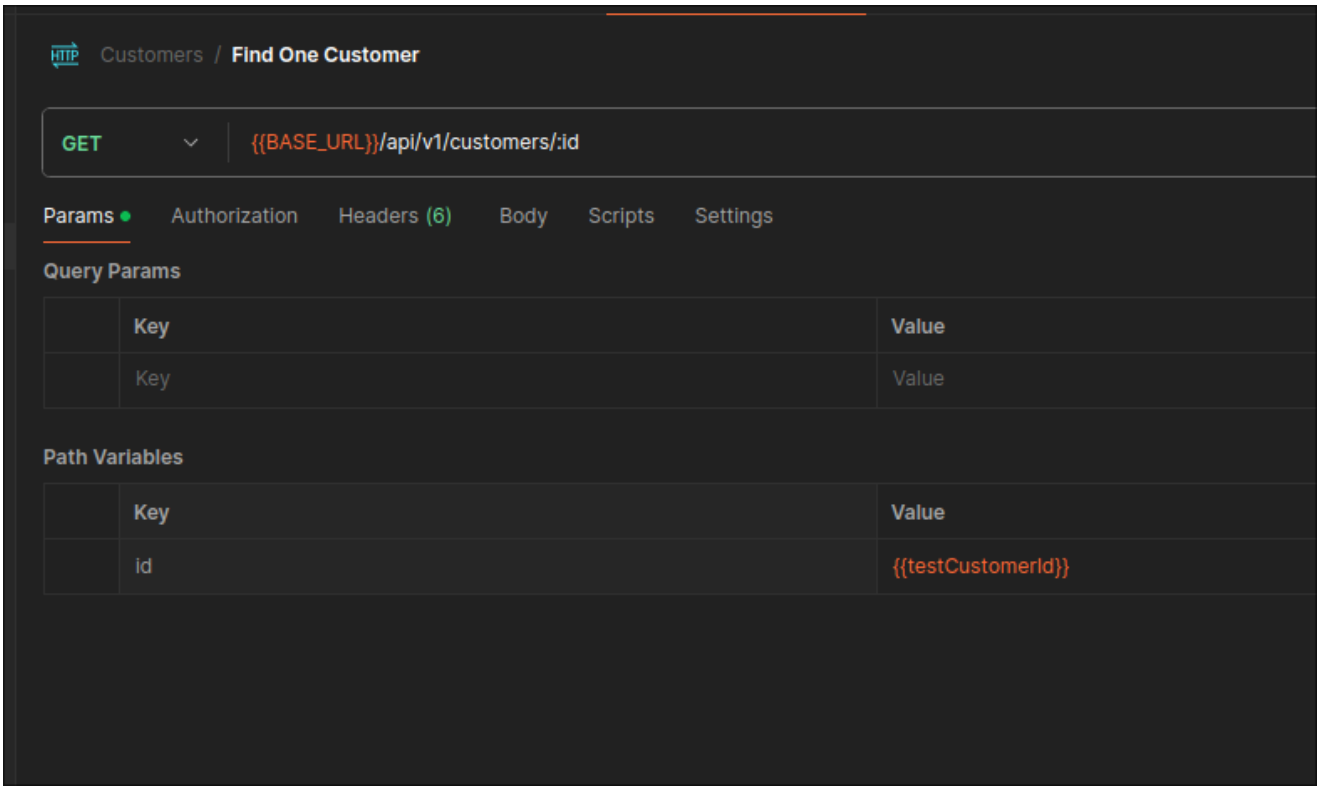
```
});

pm.test('customer should not be found', () => {

pm.sendRequest(`${BASE_URL}/api/v1/customers/${pm.environment.get('testCus
tomerId')}`, function(err, response) {
        pm.expect(response.code).to.be.eql(404);
    });
})
```



The scripts test for non-existence of a customer ID implicitly.

Finding a Customer

Initial Setup:



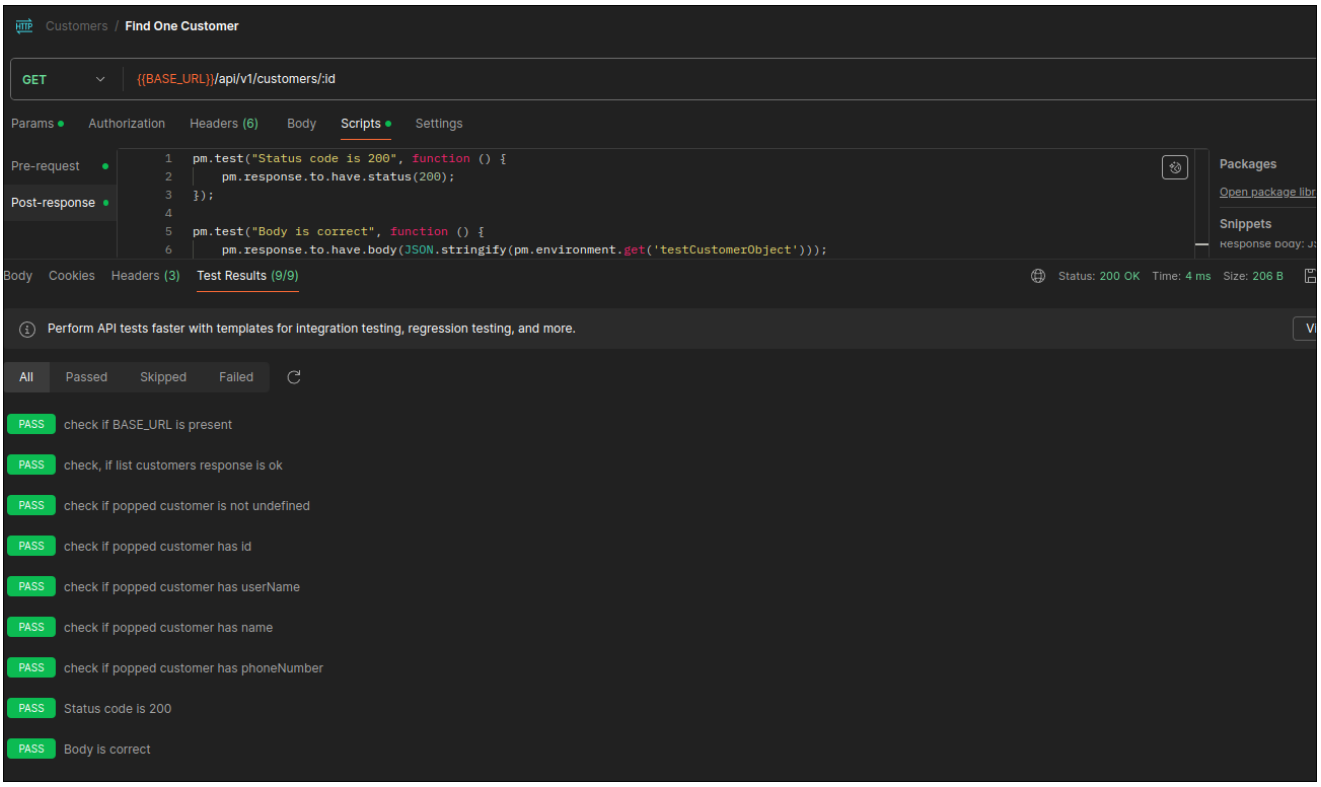
Pre-request script is identical to the corresponding deletion test, with the following exception:

```
pm.environment.set('testCustomerObject', testCustomerObject);
```

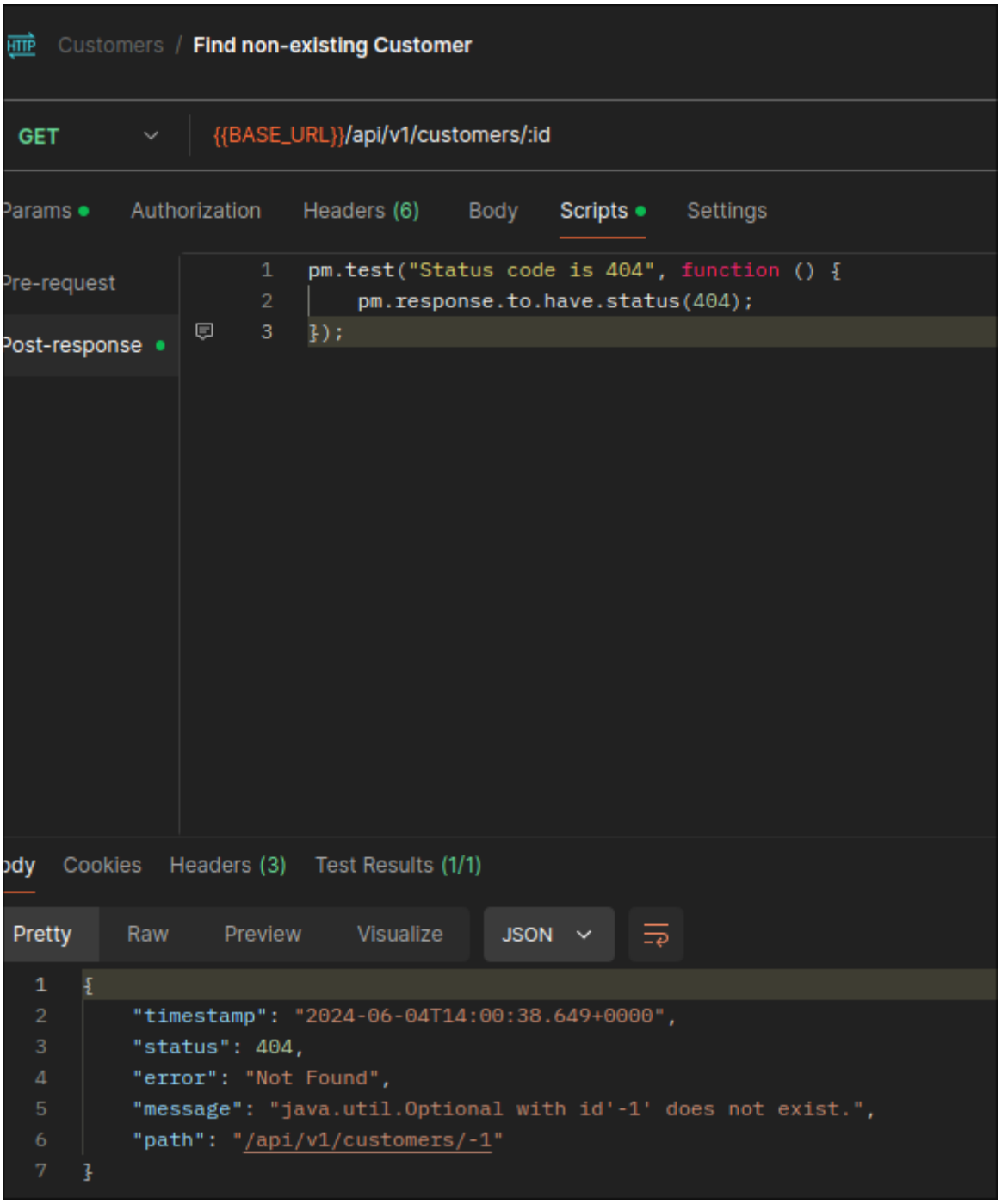
The post request script executes the following tests:

```
pm.test("Status code is 200", function() {
    pm.response.to.have.status(200);
});

pm.test("Body is correct", function() {

pm.response.to.have.body(JSON.stringify(pm.environment.get('testCustomerOb
ject')));
});
```

Resulting in:



Also, a negative test case has been added to the requests, to request a non-existing customer. The ID parameter is set to `-1`.



Creating Customers

The management controller has a create endpoint, which essentially writes to the customer repository through its service as the customer registration controller does.

```
@PostMapping
public CustomerOutDTO addCustomer(@RequestBody CustomerInDTO dto) {
    Customer customer = dto.toEntity();
    customerManagementService.addCustomer(customer);
    return new CustomerOutDTO(customer);
}
```

With its service function:

```
@Transactional(rollbackFor = Exception.class)
@Override
public Customer addCustomer(Customer customer) {
    Optional<Customer> existsPhoneNumber =
customerRepository.selectCustomerByPhoneNumber(customer.getPhoneNumber());

    if (existsPhoneNumber.isPresent()) {
        throw new BadRequestException(
                "Phone Number " + customer.getPhoneNumber() + " taken");
    }
    return customerRepository.save(customer);
}
```

If we take a look at the customer registration controller service, we can observe that a similar function is implemented:

```
@Transactional(rollbackFor = Exception.class)
public Customer registerNewCustomer(Customer customer) {
    Optional<Customer> existsPhoneNumber =
customerRepository.selectCustomerByPhoneNumber(customer.getPhoneNumber());

    //TODO: Validate customer phone number

    if (existsPhoneNumber.isPresent()) {
        Customer existingCustomer = existsPhoneNumber.get();
        if (existingCustomer.getName().equals(customer.getName())){
            throw new IllegalStateException("You are already registered");
        }
        throw new BadRequestException(
                "Phone Number " + customer.getPhoneNumber() + " taken");
    }

    return customerRepository.save(customer);
}
```

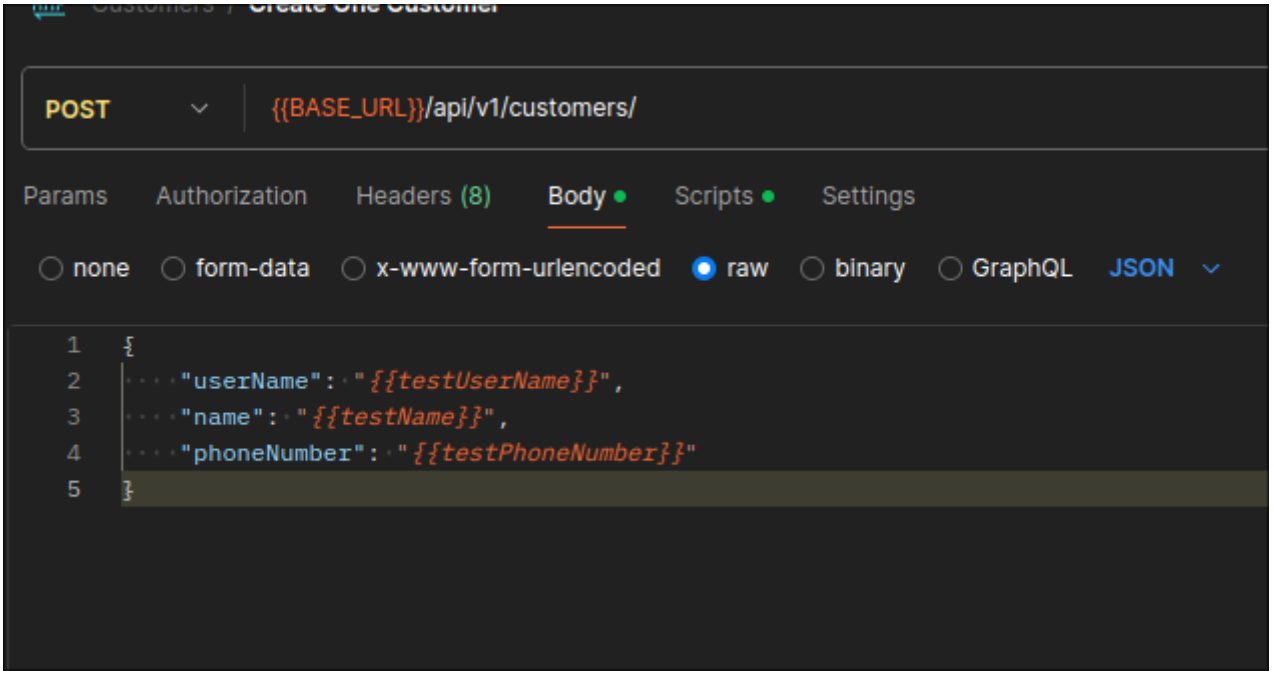
The latter of which being more strict, test cases for both endpoints will differ.

Removing the former controller for the customer management controller, leaving only the customer registration endpoint for adding new customers to the database would isolate the "Create" part for a customer model of corresponding CRUD operations to the registration service.

However, since it is unclear, how both functions would fit into a specific architecture given by the project, I'll leave both endpoints "as is" and implement tests for them.

Creating customers through the CustomerManagment Controller

POST to `/api/v1/customers` with the following body:



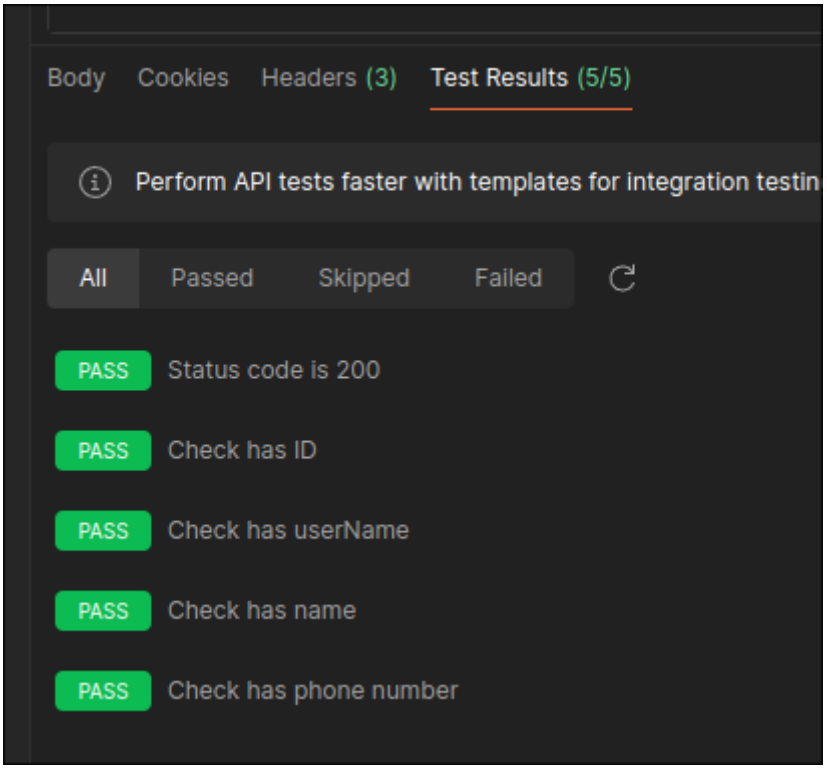
☑	testCustomerObject	default		[object Object]
☑	testName	default	Lando Norris	Lando Norris
☑	testUserName	default	NOR	NOR
☑	testPhoneNumber	default	+49123456	+49123456
☑	testInvalidPhoneNumber	default	nononophone1234	nononophone1234

Positive Tests

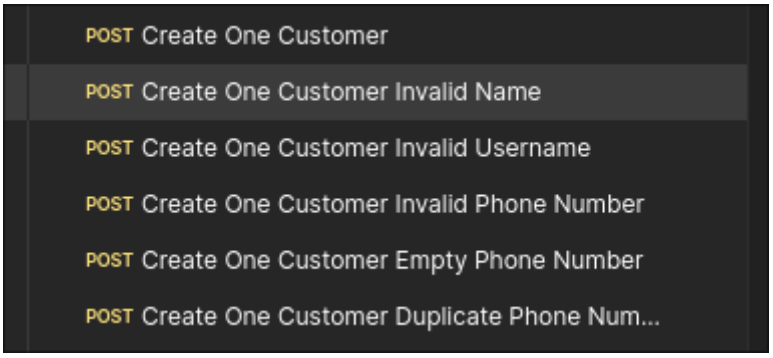
```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

var jsonData = pm.response.json();

pm.test("Check has ID", function () {
    pm.expect(jsonData.id).to.be.a('number')
});


pm.test("Check has userName", function () {

pm.expect(jsonData.userName).to.be.eq(pm.environment.get('testUserName'))
});

pm.test('Check has name', () => {
    pm.expect(jsonData.name).to.be.eq(pm.environment.get('testName'))
})

pm.test('Check has phone number', () => {

pm.expect(jsonData.phoneNumber).to.be.eq(pm.environment.get('testPhoneNumb
```

```
er'))
})
```

Response:



The following pictures contain the results of the following requests



aiming for a 400, BadRequest status code with one value of the JSON key-value pair being an empty string.

POST {{BASE_URL}}/api/v1/customers/

Params Authorization Headers (8) Body Scripts Settings

Pre-request

Post-response

```
pm.test("Status code is 400", function () {
    pm.response.to.have.status(400);
});
```

Body Cookies Headers (4) Test Results (1/1)

Pretty Raw Preview Visualize JSON

```
{
    "timestamp": "2024-06-04T19:55:10.215+0000",
    "status": 400,
    "error": "Bad Request",
    "message": "Phone Number +49123456 taken",
    "path": "/api/v1/customers/"
}
```

Customers / **Create One Customer Invalid Username**

POST {{BASE_URL}}/api/v1/customers/

Params Authorization Headers (8) Body Scripts Settings

Pre-request

Post-response

```
pm.test("Status code is 400", function () {
    pm.response.to.have.status(400);
});
```

Body Cookies Headers (4) Test Results (1/1)

Pretty Raw Preview Visualize JSON

```
{
    "timestamp": "2024-06-04T19:56:52.556+0000",
    "status": 400,
    "error": "Bad Request",
    "message": "Phone Number +49123456 taken",
    "path": "/api/v1/customers/"
}
```

Surprisingly, sending an empty phone number to the `/api/v1/customers` endpoint results in a `500` HTTP status code, instead of a `400`.

Customers / **Create One Customer Empty Phone Number**

POST {{BASE_URL}}/api/v1/customers/

Params Authorization Headers (8) Body Scripts Settings

Pre-request

Post-response

```
pm.test("Status code is 400", function () {
    pm.response.to.have.status(400);
});
```

Body Cookies Headers (4) Test Results (0/1) Status: 500 Internal Server Error Time: 14 ms

Pretty Raw Preview Visualize JSON


```
{
    "timestamp": "2024-06-04T19:58:07.484+0000",
    "status": 500,
    "error": "Internal Server Error",
    "message": "Could not commit JPA transaction; nested exception is javax.persistence.RollbackException: Error while committing the transaction",
    "path": "/api/v1/customers/"
}
```


This indicates a server-side error, which must be addressed. The desired result is a 400 BadRequest HTTP status code, since its a client side issue.

The following addition helps to create the desired effect:

```
@Transactional(rollbackFor = Exception.class)  4 usages  emmanuelcharleson.dapaah *
@Override
public Customer addCustomer(Customer customer) {
    if(customer.getPhoneNumber().isEmpty()) {
        throw new BadRequestException("Phone Number is empty");
    }
```

An invalid phone number should also result in a 400 HTTP status code.
It fails though, due to the missing validation inside the service.

Customers / **Create One Customer Invalid Phone Number**

POST {{BASE_URL}}/api/v1/customers/

Params Authorization Headers (8) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary Gr

```
1 {
2     "userName": "{{testUserName}}",
3     "name": "{{testName}}",
4     "phoneNumber": "{{testInvalidPhoneNumber}}"
5 }
```

Body Cookies Headers (3) Test Results (0/1)

Pretty Raw Preview Visualize JSON

```
1 {
2     "id": 20000,
3     "userName": "NOR",
4     "name": "Lando Norris",
5     "phoneNumber": "nononophone1234"
6 }
```

Adding the following lines takes care of that:

```
84
85      PhoneNumberValidator phoneNumberValidator = new PhoneNumberValidator();
86      boolean isValid = phoneNumberValidator.validate(customer.getPhoneNumber());
87
88      if (!isValid) {
89          throw new BadRequestException("Invalid Phone Number");
90      }
```

Customers / **Create One Customer Invalid Phone Number**

POST {{BASE_URL}}/api/v1/customers/

Params Authorization Headers (8) Body Scripts Setting

none form-data x-www-form-urlencoded raw binary

```
1 {
2     "userName": "{{testUserName}}",
3     "name": "{{testName}}",
4     "phoneNumber": "{{testInvalidPhoneNumber}}"
5 }
```

Body Cookies Headers (4) Test Results (1/1)

Pretty Raw Preview Visualize JSON

```
1 {
2     "timestamp": "2024-06-04T20:08:51.160+0000",
3     "status": 400,
4     "error": "Bad Request",
5     "message": "Invalid Phone Number",
6     "path": "/api/v1/customers/"
7 }
```

Checking for duplicate phone numbers resulting in 400s:

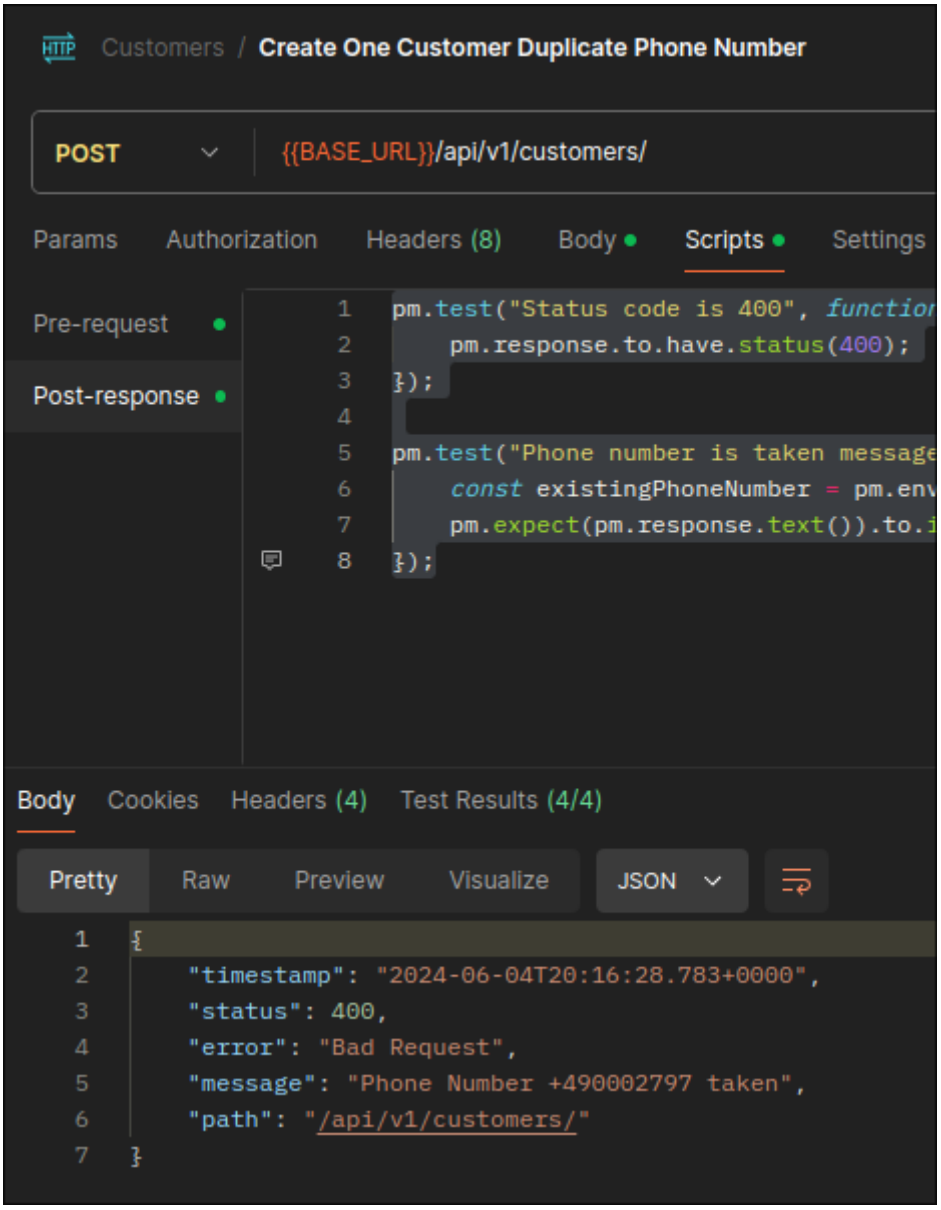
Pre Request script:

```
const BASE_URL = pm.environment.get('BASE_URL')
pm.test('base url present', () => pm.expect(BASE_URL).to.be.a('string'))

// get a phone number from the database
pm.sendRequest(`${BASE_URL}/api/v1/customers/list`, function (err,
response) {
    const resp = response.json();
    const testCustomer = resp.pop()
    pm.test('check if testCustomer has phone number', () =>
pm.expect(testCustomer.phoneNumber).to.be.a('string'))
    pm.environment.set('existingPhoneNumber', testCustomer.phoneNumber)
});
```

Post Response:

```
pm.test("Status code is 400", function () {
    pm.response.to.have.status(400);
});

pm.test("Phone number is taken message", function () {
    const existingPhoneNumber = pm.environment.get('existingPhoneNumber')
    pm.expect(pm.response.text()).to.include(existingPhoneNumber);
});
```

Result



Creating customers through the CustomerRegistration Controller

The previous steps are repeated, by duplicating the test cases and adjusting the endpoint URL.

All fail, with the exception of `Create One Customer Duplicate Phone Number Registration` and `Create One Customer Registration`.

The following result in a 500:

Customers / **Create One Customer Empty Phone Number Registration**

Save

POST {{BASE_URL}}/api/v1/customer-registration/ Send

Params Authorization Headers (8) Body Scripts Settings Cookies

Query Params

	Key	Value	Description	••• Bulk Edit
	Key	Value	Description	

Body Cookies Headers (4) Test Results (0/1) 500 Internal Server Error 15 ms 439 B Save as example

Pretty Raw Preview Visualize JSON


```
{
    "timestamp": "2024-06-04T20:58:58.309+0000",
    "status": 500,
    "error": "Internal Server Error",
    "message": "Could not commit JPA transaction; nested exception is javax.persistence.
        RollbackException: Error while committing the transaction",
    "path": "/api/v1/customer-registration/"
}
```

Customers / **Create One Customer Invalid Username Registration**

Save

POST {{BASE_URL}}/api/v1/customer-registration/ **Send**

Params | Authorization | Headers (8) | Body | Scripts | Settings | Cookies

Query Params

	Key	Value	Description	••• Bulk Edit
	Key	Value	Description	

Body | Cookies | Headers (4) | Test Results (0/1) | 500 Internal Server Error | 20 ms | 439 B | Save as example

Pretty | Raw | Preview | Visualize | JSON


```
{
    "timestamp": "2024-06-04T20:58:40.868+0000",
    "status": 500,
    "error": "Internal Server Error",
    "message": "Could not commit JPA transaction; nested exception is javax.persistence.
        RollbackException: Error while committing the transaction",
    "path": "/api/v1/customer-registration/"
}
```

Customers / **Create One Customer Invalid Name Registration**

POST {{BASE_URL}}/api/v1/customer-registration/

Params | Authorization | Headers (8) | Body | Scripts | Settings

Query Params

	Key	Value	Description
	Key	Value	Description

ody | Cookies | Headers (4) | Test Results (0/1) | 500 Internal Server Error | 106 ms | 439 B

Pretty | Raw | Preview | Visualize | JSON


```
{
    "timestamp": "2024-06-04T20:58:24.297+0000",
    "status": 500,
    "error": "Internal Server Error",
    "message": "Could not commit JPA transaction; nested exception is javax.persiste
        RollbackException: Error while committing the transaction",
    "path": "/api/v1/customer-registration/"
}
```

This request accepts an invalid phone number as well:

Customers / **Create One Customer Invalid Phone Number Registration**

POST {{BASE_URL}}/api/v1/customer-registration/

Params Authorization Headers (8) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
    "userName": "{{testUserName}}",
    "name": "{{testName}}",
    "phoneNumber": "{{testInvalidPhoneNumber}}"
}
```

Body Cookies Headers (3) Test Results (0/1) Status: 200 OK Time: 88

Pretty Raw Preview Visualize JSON

```
{
    "id": 20000,
    "userName": "NOR",
    "name": "Lando Norris",
    "phoneNumber": "nononophone1234"
}
```

Fixing the Issues

To fix the latest error, we implement the following lines:

```
@Transactional(rollbackFor = Exception.class)  4 usages  emmanuelcharleson.dapaah *
public Customer registerNewCustomer(Customer customer) {
    PhoneNumberValidator phoneNumberValidator = new PhoneNumberValidator();
    boolean isValid = phoneNumberValidator.validate(customer.getPhoneNumber());

    if (!isValid) {
        throw new BadRequestException("Invalid Phone Number");
    }

```

Next, we are tackling the `Create One Customer Invalid Name Registration` error. It seems like the Validator is failing, as it should. With using the debugger, we found the error (`IllegalStateException`) to reside when `customerRepository.save(customer)` is called.

Adding validation and annotations to the `CustomerInDTO` helped to pass all the tests.

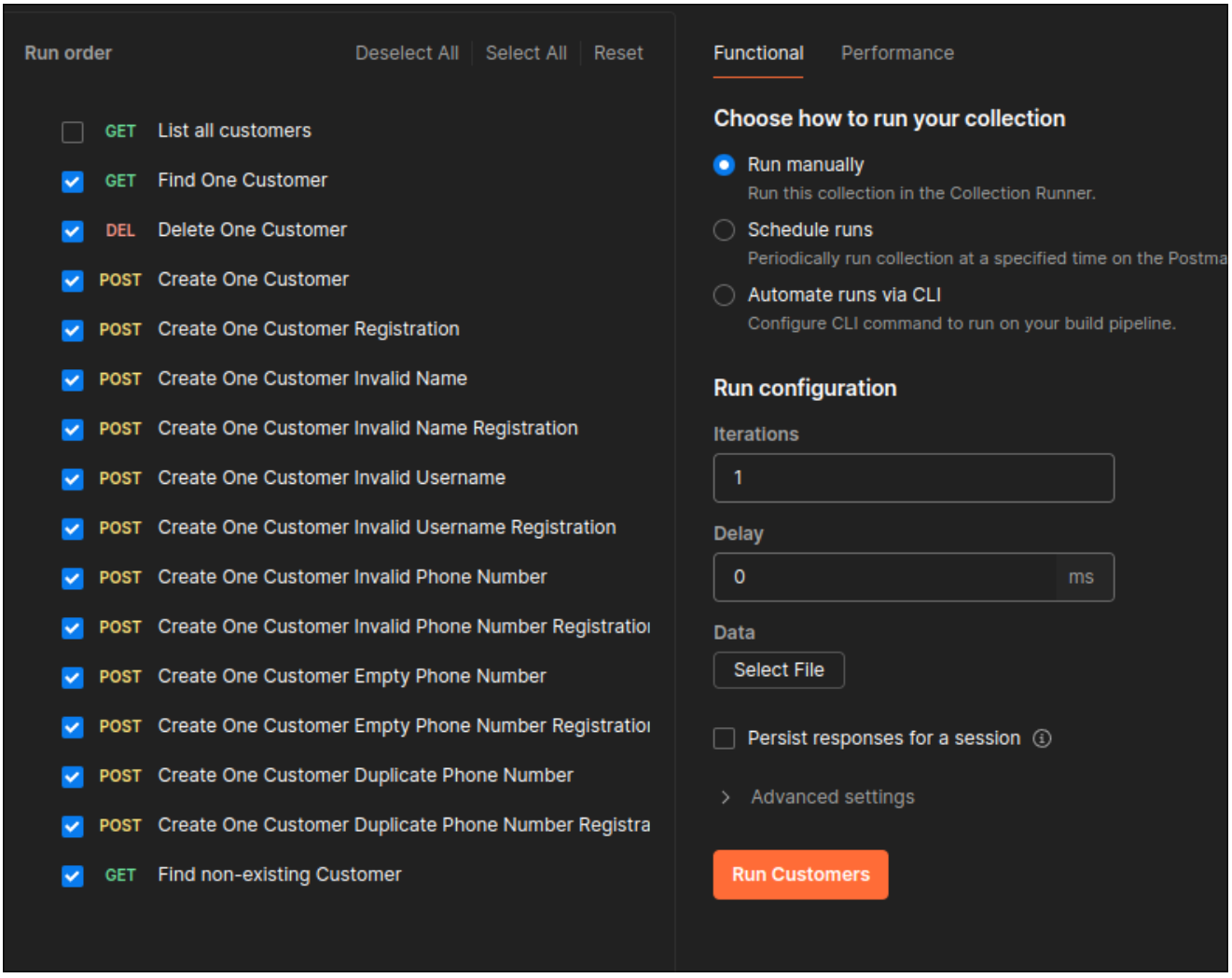
```
diff --git
a/src/main/java/com/softwaretesting/testing/customerRegistration/controlle
r/CustomerRegistrationController.java
b/src/main/java/com/softwaretesting/testing/customerRegistration/controlle
r/CustomerRegistrationController.java
index 54c3a82..bfb0ac2 100644
---
a/src/main/java/com/softwaretesting/testing/customerRegistration/controlle
r/CustomerRegistrationController.java
+++
b/src/main/java/com/softwaretesting/testing/customerRegistration/controlle
r/CustomerRegistrationController.java
@@ -4,11 +4,16 @@ import
com.softwaretesting.testing.dto.inbound.CustomerInDTO;
 import com.softwaretesting.testing.dto.outbound.CustomerOutDTO;
 import com.softwaretesting.testing.model.Customer;
 import
com.softwaretesting.testing.customerRegistration.service.CustomerRegistrat
ionService;
+import org.springframework.validation.annotation.Validated;
 import org.springframework.web.bind.annotation.*;

+import javax.validation.Valid;
+import javax.validation.constraints.NotBlank;
+

 @RestController
 @RequestMapping("api/v1/customer-registration")
+@Validated
 public class CustomerRegistrationController {

     private final CustomerRegistrationService
customerRegistrationService;
@@ -19,7 +24,7 @@ public class CustomerRegistrationController {


     @PostMapping
-    public CustomerOutDTO registerNewCustomer(@RequestBody CustomerInDTO
dto) {
+    public CustomerOutDTO registerNewCustomer(@Valid @RequestBody
CustomerInDTO dto) {
         Customer customer = dto.toEntity();
         customerRegistrationService.registerNewCustomer(customer);
         return new CustomerOutDTO(customer);
diff --git
a/src/main/java/com/softwaretesting/testing/customerRegistration/service/C
ustomerRegistrationService.java
```

```
b/src/main/java/com/softwaretesting/testing/customerRegistration/service/C
ustomerRegistrationService.java
index fdba85a..e032def 100644
---
a/src/main/java/com/softwaretesting/testing/customerRegistration/service/C
ustomerRegistrationService.java
+++
b/src/main/java/com/softwaretesting/testing/customerRegistration/service/C
ustomerRegistrationService.java
@@ -3,6 +3,7 @@ package
com.softwaretesting.testing.customerRegistration.service;
 import com.softwaretesting.testing.exception.BadRequestException;
 import com.softwaretesting.testing.dao.CustomerRepository;
 import com.softwaretesting.testing.model.Customer;
+import com.softwaretesting.testing.validator.PhoneNumberValidator;
 import org.springframework.beans.factory.annotation.Autowired;
 import org.springframework.stereotype.Service;
 import org.springframework.transaction.annotation.Transactional;
@@ -22,13 +23,18 @@ public class CustomerRegistrationService {

     @Transactional(rollbackFor = Exception.class)
     public Customer registerNewCustomer(Customer customer) {
-        Optional<Customer> existsPhoneNumber =
customerRepository.selectCustomerByPhoneNumber(customer.getPhoneNumber());
+        PhoneNumberValidator phoneNumberValidator = new
PhoneNumberValidator();
+        boolean isValid =
phoneNumberValidator.validate(customer.getPhoneNumber());
+
+        if (!isValid) {
+            throw new BadRequestException("Invalid Phone Number");
+        }

-        //TODO: Validate customer phone number
+        Optional<Customer> existsPhoneNumber =
customerRepository.selectCustomerByPhoneNumber(customer.getPhoneNumber());

         if (existsPhoneNumber.isPresent()) {
             Customer existingCustomer = existsPhoneNumber.get();
-            if (existingCustomer.getName().equals(customer.getName())){
+            if (existingCustomer.getName().equals(customer.getName())) {
                 throw new IllegalStateException("You are already
registered");
             }
             throw new BadRequestException(
diff --git
a/src/main/java/com/softwaretesting/testing/dto/inbound/CustomerInDTO.java
b/src/main/java/com/softwaretesting/testing/dto/inbound/CustomerInDTO.java
index 950c3d6..99fd8a6 100644
---
```

```
a/src/main/java/com/softwaretesting/testing/dto/inbound/CustomerInDTO.java
+++
b/src/main/java/com/softwaretesting/testing/dto/inbound/CustomerInDTO.java
@@ -8,9 +8,14 @@ package com.softwaretesting.testing.dto.inbound;

 import com.softwaretesting.testing.model.Customer;

+import javax.validation.constraints.NotBlank;
+
 public class CustomerInDTO {
+       @NotBlank
        private String userName;
+       @NotBlank
        private String name;
+       @NotBlank
        private String phoneNumber;
```

Running all tests on a fresh instance

The tests will be run in the following order:



List all customers is skipped, since it has 20.000 tests. The results can be found above.

Customers - Run results

Ran today at 23:39:55 · View all runs

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	Software Testing Env	1	7s 250ms	52	14 ms

RUN SUMMARY

		1
GET Find One Customer	9 | 0	
DELETE Delete One Customer	12 | 0	
POST Create One Customer	7 | 0	
POST Create One Customer Registration	4 | 3	✕
POST Create One Customer Invalid Name	0 | 1	✕
POST Create One Customer Invalid Name Regist...	1 | 0	
POST Create One Customer Invalid Username	0 | 1	✕
POST Create One Customer Invalid Username R...	1 | 0	
POST Create One Customer Invalid Phone Numb...	1 | 0	
POST Create One Customer Invalid Phone Numb...	1 | 0	
POST Create One Customer Empty Phone Numb...	1 | 0	
POST Create One Customer Empty Phone Numb...	1 | 0	
POST Create One Customer Duplicate Phone Nu...	4 | 0	
POST Create One Customer Duplicate Phone Nu...	4 | 0	
GET Find non-existing Customer	1 | 0	

Turns out, that implementing the `CustomerRegistrationController` Logic is breaking the `CustomerManagementController` Implementation.

Adding the following annotation eliminates the issue

```
@PostMapping   emmanuelcharleson.dapaah *
public CustomerOutDTO addCustomer(@Valid @RequestBody CustomerInDTO dto) {
    Customer customer = dto.toEntity();
    customerManagementService.addCustomer(customer);
    return new CustomerOutDTO(customer);
}
```

Customers - Run results

Ran today at 23:51:43 · View all runs

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	Software Testing Env	1	7s 204ms	52	11 ms

RUN SUMMARY

			1
GET	Find One Customer	9 | 0	
DELETE	Delete One Customer	12 | 0	
POST	Create One Customer	7 | 0	
POST	Create One Customer Registration	7 | 0	
POST	Create One Customer Invalid Name	1 | 0	
POST	Create One Customer Invalid Name Regist...	1 | 0	
POST	Create One Customer Invalid Username	1 | 0	
POST	Create One Customer Invalid Username R...	1 | 0	
POST	Create One Customer Invalid Phone Numb...	1 | 0	
POST	Create One Customer Invalid Phone Numb...	1 | 0	
POST	Create One Customer Empty Phone Numb...	1 | 0	
POST	Create One Customer Empty Phone Numb...	1 | 0	
POST	Create One Customer Duplicate Phone Nu...	4 | 0	
POST	Create One Customer Duplicate Phone Nu...	4 | 0	
GET	Find non-existing Customer	1 | 0	

Spring MVC

Having already worked out the way of testing the repository with Postman, writing the test was actually straightforward.
Some 500 HTTP codes were hard to test or left out, because its inconclusive if those codes were there by design.

Coverage

The Spring MVC tests were able to cover 100% of lines and branches.

testing

Element	Missed Instructions	Cov.	Missed Branches	Cov.	M
com.softwaretesting.testing.dto.outbound		50%		n/a	
com.softwaretesting.testing.dto.inbound		78%		n/a	
com.softwaretesting.testing		37%		n/a	
com.softwaretesting.testing.validator		100%		100%	
com.softwaretesting.testing.util		100%		100%	
com.softwaretesting.testing.model		100%		100%	
com.softwaretesting.testing.customerManagement.service		100%		100%	
com.softwaretesting.testing.customerRegistration.service		100%		100%	
com.softwaretesting.testing.customerManagement.controller		100%		n/a	
com.softwaretesting.testing.config		100%		n/a	
com.softwaretesting.testing.customerRegistration.controller		100%		n/a	
com.softwaretesting.testing.exception		100%		n/a	
Total	45 of 1,630	97%	0 of 64	100%	

Conclusion

While Postman is compatible with every REST API, its testing capabilities are sufficient in the extend of the Chai.js framework.

While it helps to create integration tests, its developer experience is cumbersome. I did not find a way to re-use code and state is usually handled by environment variables. Also its shareable and team feature seem to offer capabilities for collaborative team work.
When comparing this experience with the Spring MVC integration testing, Postman testing seems less time efficient.
When residing inside one framework the testing tools, programming language and environment stays mostly the same.

Furthermore, if MVC tests are omitted for the sake of Postman tests, the code coverage for controllers need to be implemented otherwise. This could result in writing code twice, once for the framework and one suite for Postman.