Case Study Report

By

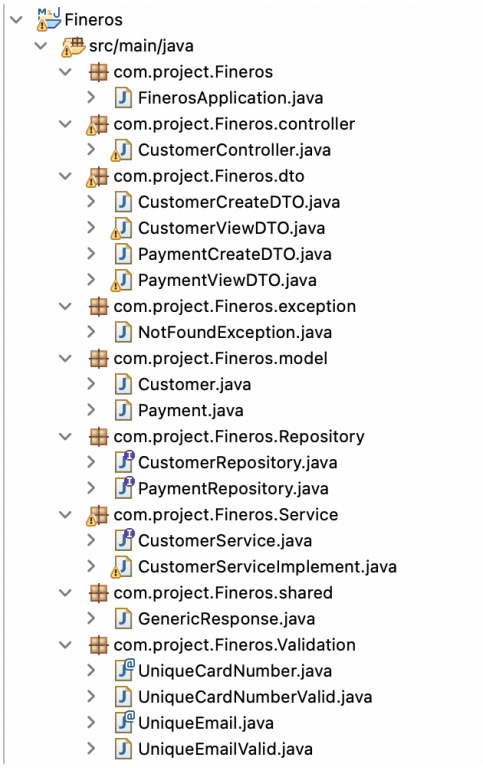
Mert Kaya

Tech Stack

- Spring Initializr
    - The spring boot application project was firstly generated from the website by adding the dependencies that should be user for completing the tasks that were given. Then the project was imported from Eclipse Ide where the project was implemented
    - Dependencies that were chosen;
        - Spring Web
        - H2 database
        - Spring Boot DevTools
        - Spring Data JPA
        - Spring Security(that was not needed but added for future cases)
- Postman
    - Used to test the mapping feature that were implemented in the spring boot application project
- Maven
- Java 17
- Spring Boot -> 2.7.2

Design

13 Classes and 5 interfaces were used for the project in 9 packages for the usage purposes. Furthermore a property was created for validation messages that were needed when the user inputs wrong data.



The idea behind of the design was that the controller class should act like a main class where all the customer actions were organized from one place by reaching out all other instances that were needed to be used for the tasks. This why it is called object-oriented-programming. More clearly the controller class should interact with service level where the functionalities were defined and implemented and the service level should reach to the repositories to handle queries for entity classes for the database. Meanwhile the entity classes that were created for creating tables and fields for the database, should not be reached directly for the security of the data and the system itself. To provide that, the DTO classes in DTO packet were created where the users' data were shown to the users and the transactions for the database should be

done by using these classes. The validation classes were implemented in order to provide the required validations to guide the users when they entered wrong data. The system should block all entry tries by the users that should not be entered. To guide the users to enter the right data, the validationMessages property was created which sends guidance messages to the user when they entered unacceptable data.

1-) Generic Response Class in ".com.project.Fineros.shared" package;

- This class was generated to return string message to the user after the transactions to inform the user that the task was executed successfully

2-) FinerosApplication Class in "com.project.Fineros" package;

- This class was generated automatically by the spring initializr to run the spring boot application.

3-) CustomerController Class in "com.project.Fineros.controller" package;

- This class was created as an api or main class where all the methods from other classes are being called from in order to perform the tasks by simply reaching to the service package that includes an interface and a class that implements the interface.

4-) CustomerService Interface and CustomerServiceImplement in "com.project.Fineros.service" package;

- Methods that were called from the Controller class were added in the interface and were implemented in the class

5-) CustomerRepository and PaymentRepository Interfaces in "com.project.Fineros.Repository" package;

- The CustomerRepository interface was created to extend JpaRepository from spring framework to handle database queries, particularly for the table that the customers were stored, that were needed for the tasks.
- The PaymentRepository interface was created to extend JpaRepository from spring framework to handle database queries, particularly for the table that the payments were stored, that were needed for the tasks.

6-) Customer and Payment classes in "com.project.Fineros.model" package;

- The Customer class was created as an entity class to create a table called , customers, and add fields to the table for the database

- The Payment class was created as an entity class to create a table called , payments, and add fields to the table for the database

7-) NotFoundException class in “com.project.Fineros.exception” package;

- This class extends RunTimeException for the errors that may be occurred during the running state of the program

8-) CustomerCreateDTO, CustomerViewDTO, PaymentCreateDTO, PaymentView classes in “com.project.Fineros.exception” package;

- CustomerCreateDTO class was created to be used while registering a new customer to the database instead of using the table entity, Customer class, itself. And the validations were implemented in this class for the customer entry data.

- CustomerViewDTO class was created to be returned in the methods in controller class as a representation of an entity since entities should never been reached directly for the transactions for the security of the program. It is a final class that has only one method to be a representation for the entity class(Customer)

- PaymenyCreateDTO class was created to be used while making a new payment to the database instead of using the table entity, Payment class, itself. And the validations were implemented in this class for the entry data.

- PaymentViewDTO class was created to be returned in the methods in controller class as a representation of an entity since entities should never been reached directly for the transactions for the security of the program. It is a final class that has only one method to be a representation for the entity class(Payment)

9-) UniqueCardNumber, UniqueEmail Interfaces and UniqueCardNumberValid, UniquesEmailValid classes in “com.project.Fineros.Validation” package;

- UniqueCardNumber interface and UniqueCardNumberValid class were created to be sure that the customer enters a unique credit card number

- UniqueEmail interface and UniqueEmailValid class were created to be sure that the customer cannot register with an existing email.

The Completed Tasks

1-) Register New Customer

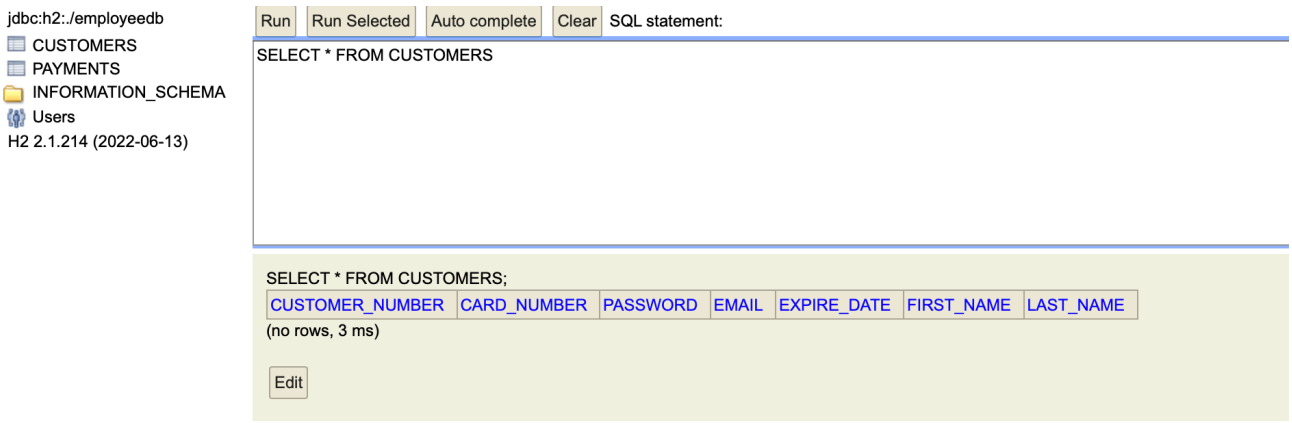
- Code

```java
@RestController
@RequestMapping("/controller")
public class CustomerController {

    private final CustomerService customerService;

    public CustomerController(CustomerService customerService) {
        this.customerService = customerService;
    }

    // Register new Customer
    @PostMapping("v1/customer")
    public ResponseEntity<?> createNewCustomer(@Valid @RequestBody CustomerCreateDTO customerCreateDTO){

        customerService.createNewCustomer(customerCreateDTO);
        return ResponseEntity.ok(new GenericResponse("User Created"));
    }
```

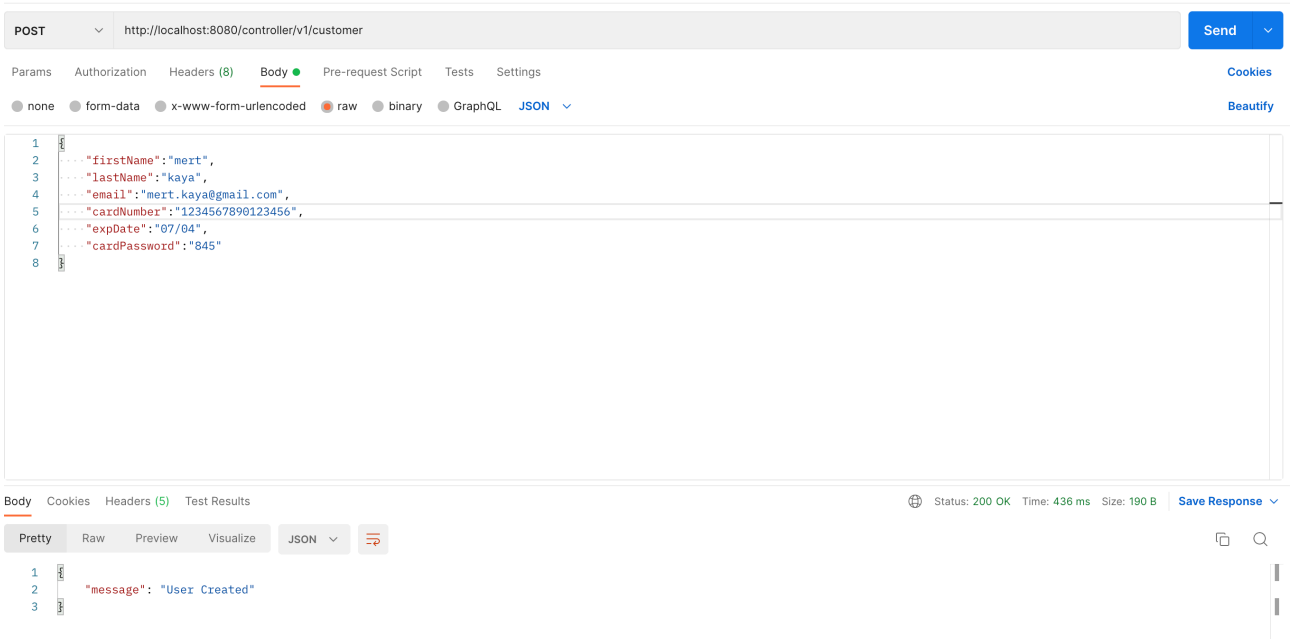
```java
public interface CustomerService {

    CustomerViewDTO createNewCustomer( CustomerCreateDTO customerCreateDTO);

    PaymentViewDTO makeNewPayment(PaymentCreateDTO paymentCreateDTO);

    List<Payment> getPaymentsByCardNumber(String cardNumber);

    List<Payment> getPaymentsByDate(Date startDate, Date endDate);


}
```

```java
@Service
public class CustomerServiceImplement implements CustomerService {

    private final CustomerRepository customerRepository;
    private final PaymentRepository paymentRepository;


    public CustomerServiceImplement(CustomerRepository customerRepository, PaymentRepository paymentRepository) {
        this.customerRepository = customerRepository;
        this.paymentRepository = paymentRepository;

    }

    @Override
    @Transactional
    public CustomerViewDTO createNewCustomer(CustomerCreateDTO customerCreateDTO) {

        final Customer customer = customerRepository.save(new Customer(customerCreateDTO.getFirstName(),
                customerCreateDTO.getLastName(), customerCreateDTO.getEmail(), customerCreateDTO.getCardNumber(), customerCreateDTO.getExpDate(), customerCreateDTO.getCardPassword()));
        return CustomerViewDTO.view(customer);
    }
```

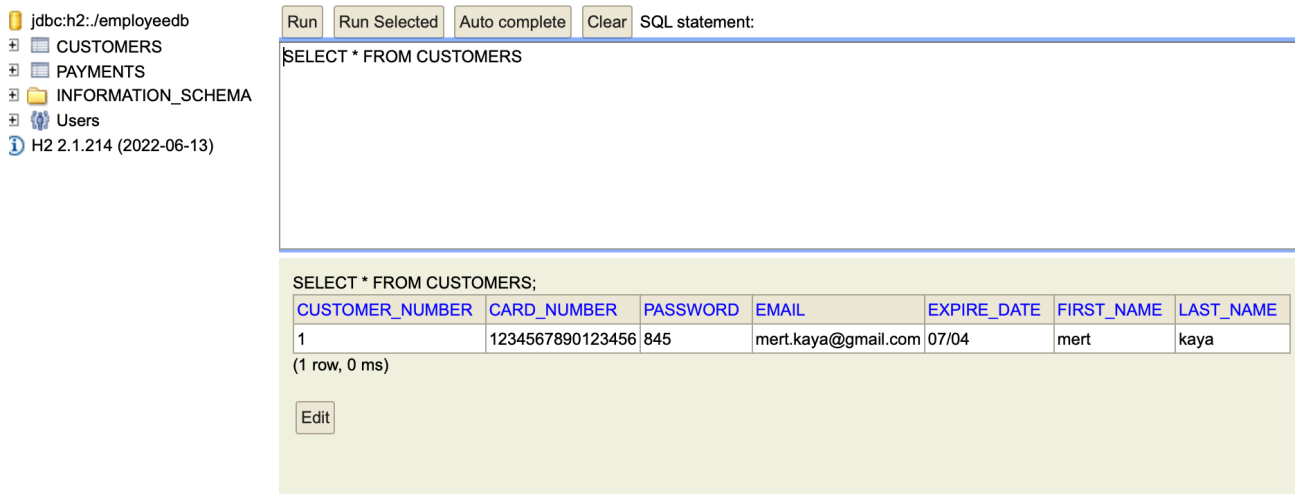
- Testing using postman



Application started successfully and here I see that the customer table is created and there is no value on the table



Here I entered my entries by using the right path that was implemented in the controller class and then I posted the data where I got the response that the user is created.



Now I checked the database and saw that the values I posted was stored in the table succesfully..

2-) Placing a new Payment

- Code

The method in the controller class:

```
// Making a payment
@PostMapping("v1/payment")
public ResponseEntity<?> makeNewPayment(@RequestBody PaymentCreateDTO paymentCreateDTO){
    customerService.makeNewPayment(paymentCreateDTO);

    return ResponseEntity.ok(new GenericResponse("Payment made"));
}
```

```
ublic interface CustomerService {

    CustomerViewDTO createNewCustomer( CustomerCreateDTO customerCreateDTO);

    PaymentViewDTO makeNewPayment(PaymentCreateDTO paymentCreateDTO);

    List<Payment> getPaymentsByCardNumber(String cardNumber);

    List<Payment> getPaymentsByDate(Date startDate, Date endDate);
```

The method in Service implementation class

```
@Override
@Transactional
public PaymentViewDTO makeNewPayment(PaymentCreateDTO paymentCreateDTO) {
    final Payment payment = paymentRepository.save(new Payment(paymentCreateDTO.getCardNumber(), paymentCreateDTO.getAmount()));
    return PaymentViewDTO.view(payment);
}
```

- Testing using postman

Run | Run Selected | Auto complete | Clear | SQL statement:

```
SELECT * FROM PAYMENTS
```

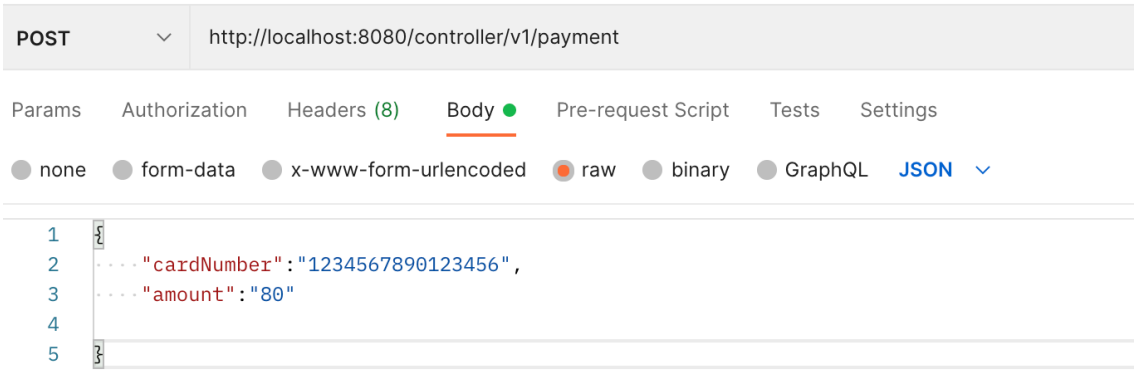
MA

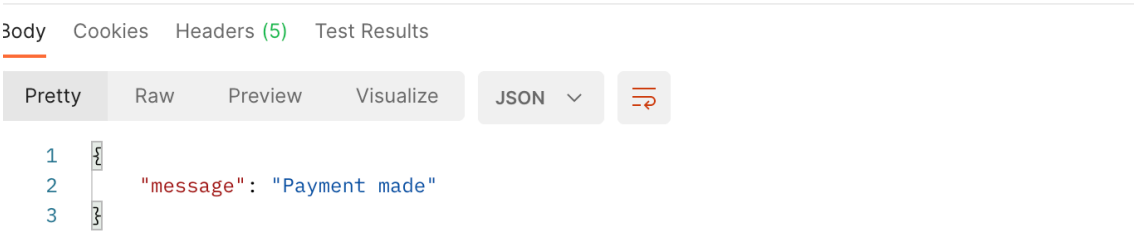
```
SELECT * FROM PAYMENTS;
```

ORDER_ID	AMOUNT	CARD_NUMBER	DATA_CREATED

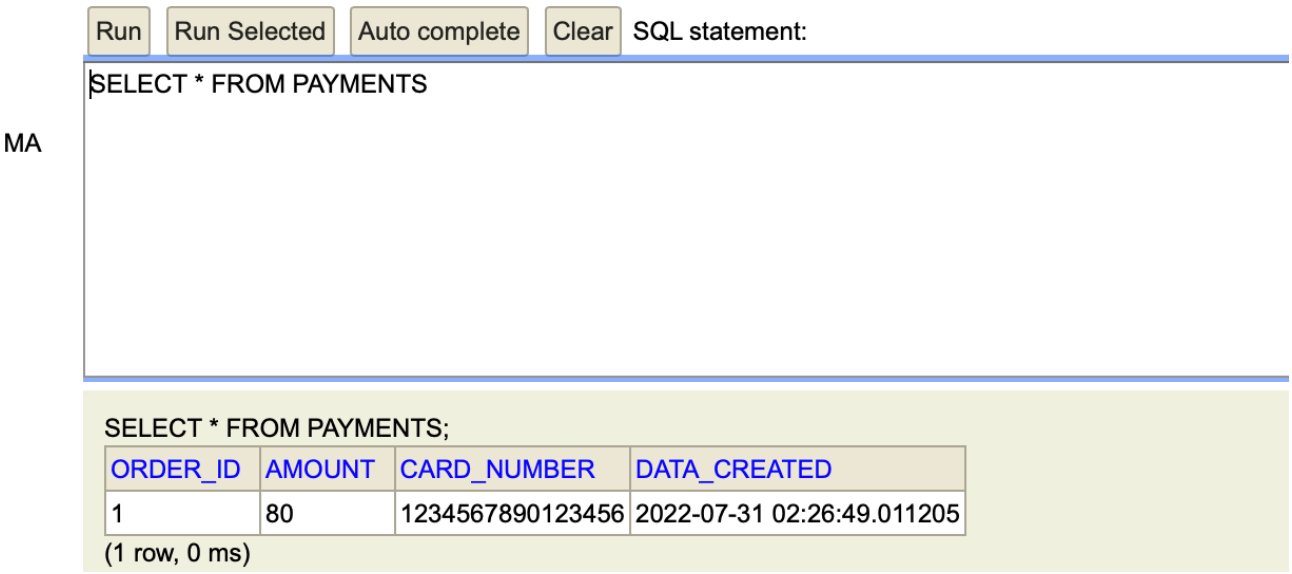
(no rows, 2 ms)

Application started successfully and here I see that the payment table is created and there is no value on the table





Here I entered my entries by using the right path that was implemented in the controller class and then I posted the data where I got the response that the payment made.



MA

Now I checked the database and saw that the values I posted was stored in the table succesfully..

3,4-) List all payments of a customer by search criteria(card number)

The method should take PaymentViewDTO as a parameter into the list but got an error saying that that PaymentViewDTO class does not have Serializable where I actually implemented, thus I believe there is a small error to fix but I could not place more time on it.

- Code

The method in the Controller class

```
//Listing All payments of a customer using the credit card number that the customer has(credit card is unique for customers)
@GetMapping("v1/payments/{cardNumber}")
public  ResponseEntity<List<Payment>> getPaymentsByCardNumber(@PathVariable(name="cardNumber") String cardNumber ){
    final List<Payment> payments = customerService.getPaymentsByCardNumber(cardNumber);
    return ResponseEntity.ok(payments);
}
```

```
@Repository
public interface PaymentRepository extends JpaRepository<Payment, Long> {

    List<Payment> findByCardNumber(String cardNumber);

    List<Payment> findByDataCreatedBetween(Date startDate, Date endDate);

    boolean existsPaymentByCardNumber(String cardNumber);

}
```

The method in Service implementation class

```
@Override
public List<Payment> getPaymentsByCardNumber(String cardNumber) {
    final List<Payment> payments = paymentRepository. findByCardNumber(cardNumber);
    return payments;

}
```

I

Run | Run Selected | Auto complete | Clear | SQL statement:

```
SELECT * FROM CUSTOMERS
```

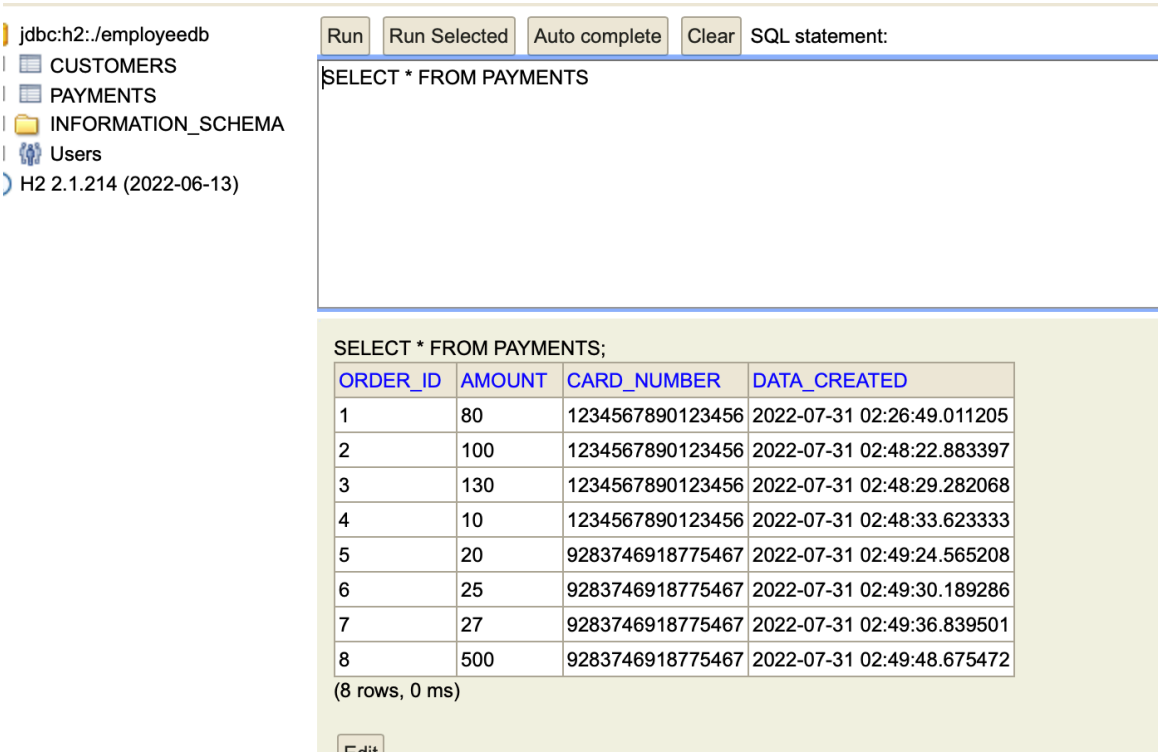
```
SELECT * FROM CUSTOMERS;
```

CUSTOMER_NUMBER	CARD_NUMBER	PASSWORD	EMAIL	EXPIRE_DATE	FIRST_NAME	LAST_NAME
1	1234567890123456	845	mert.kaya@gmail.com	07/04	mert	kaya
2	9283746918775467	345	efe.yildirim@gmail.cam	05/07	efe	yildirim

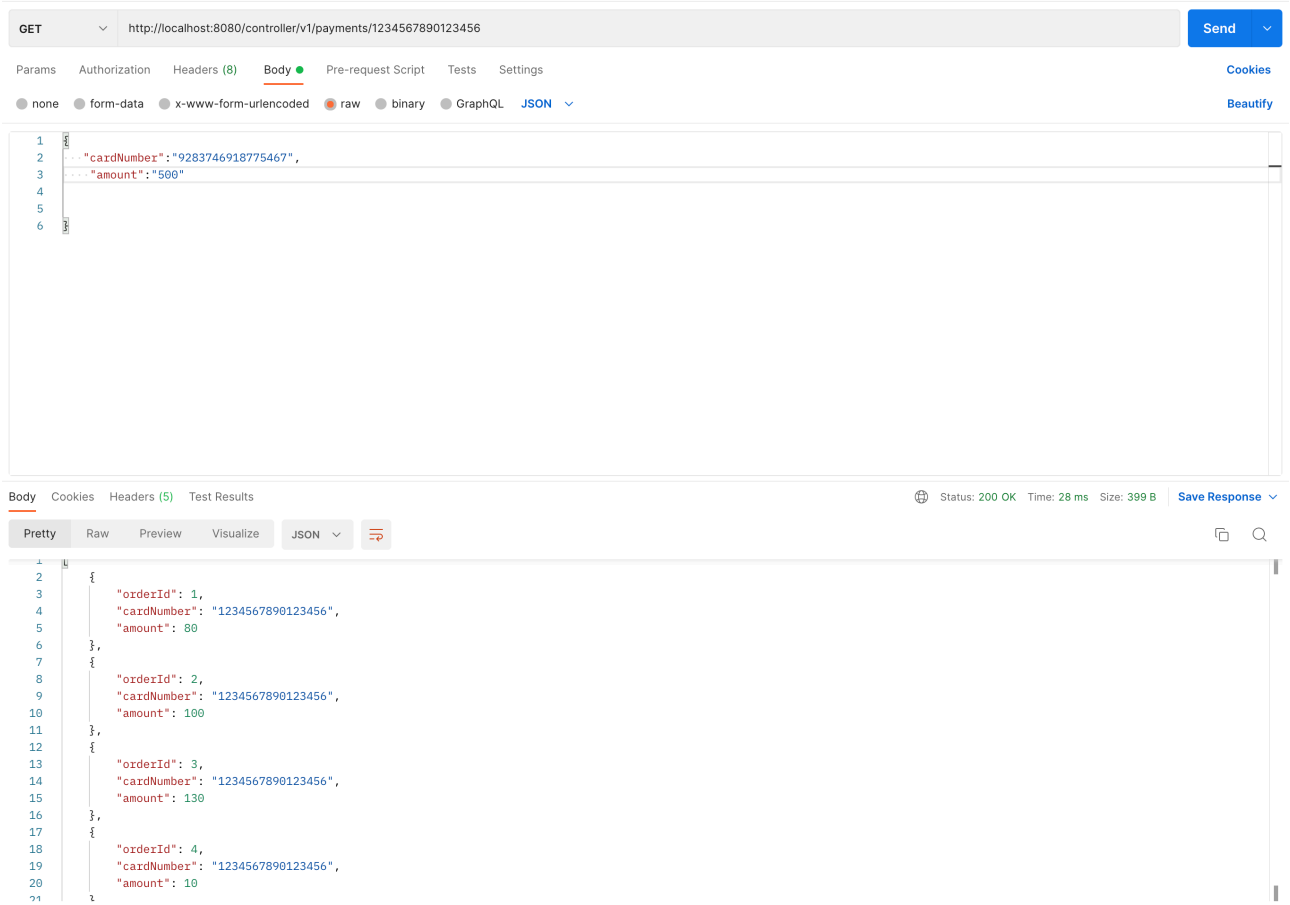
(2 rows, 0 ms)

I started the application and added one more customer, now I see two customers and their credit card numbers as well.

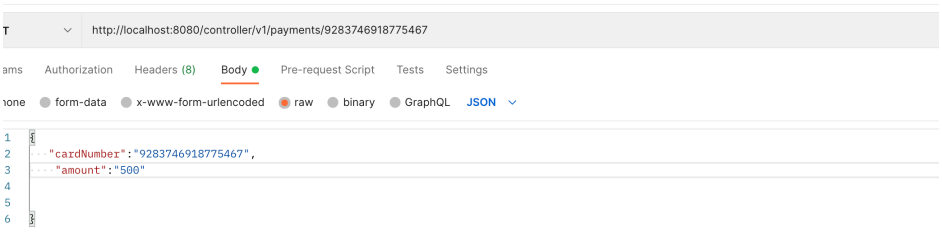
Testing using postman

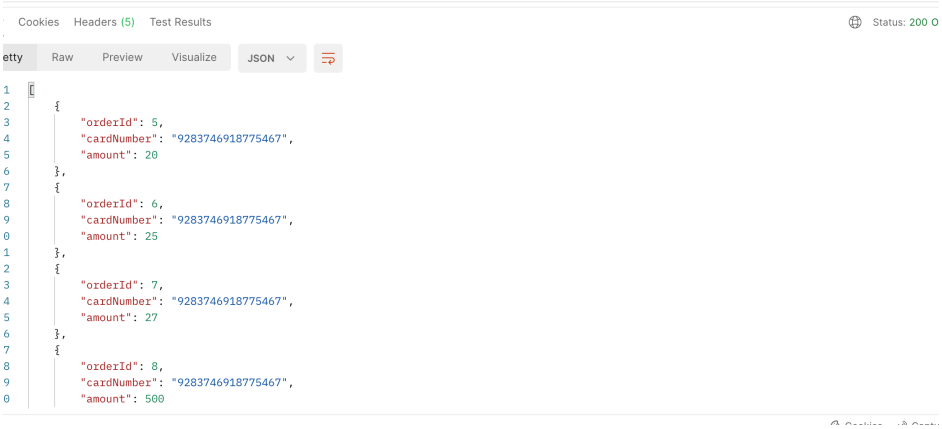


I also made a few more payments for both customers (mert kaya, and efe yildirim).



I used postman to get all the payments that Mert Kaya did by taking his credit card number as a parameter for the query





I now used the postman again to get all the payments that Efe Yildirim made by taking his credit card number as a parameter for the query.

5-) List all payments by date interval(startDate, endDate)

Most of the code part for this task was implemented but could not solve the error that occurred because of my entry through the url as a date is being accepted as a string which is not acceptable for date entry could not be converted.

- Code

The method in the Controller class

```
//Getting the payments between the start date and end date
@GetMapping("v1/payments")
public ResponseEntity<List<Payment>> getPaymentsByDate(@RequestParam("startDate") Date startDate,@RequestParam("endDate") Date endDate){
    final List<Payment> payments = customerService.getPaymentsByDate(startDate, endDate);
    return ResponseEntity.ok(payments);
}
```

```java
public interface CustomerService {

    CustomerViewDTO createNewCustomer( CustomerCreateDTO customerCreateDTO);

    PaymentViewDTO makeNewPayment(PaymentCreateDTO paymentCreateDTO);

    List<Payment> getPaymentsByCardNumber(String cardNumber);

    List<Payment> getPaymentsByDate(Date startDate, Date endDate);


}
```

```java
@Override
public List<Payment> getPaymentsByDate(Date startDate, Date endDate) {
    final List<Payment> payments = paymentRepository.findByDataCreatedBetween(startDate, endDate);
    return payments;
}
```

```java
@Repository
public interface PaymentRepository extends JpaRepository<Payment, Long> {

    List<Payment> findByCardNumber(String cardNumber);

    List<Payment> findByDataCreatedBetween(Date startDate, Date endDate);

    boolean existsPaymentByCardNumber(String cardNumber);

}
```

```
"message": "Failed to convert value of type 'java.lang.String' to required type 'java.sql.Date'; nested exception is org.springframework.core.convert.
    ConversionFailedException: Failed to convert from type [java.lang.String] to type [@org.springframework.web.bind.annotation.RequestParam java.sql.Date] for value
    '2022-07-30\n'; nested exception is java.lang.IllegalArgumentException",
```

The entry error I got after I ran my query from postman as a get request

6-) Customers should not be registered with an existing email.

```
public class CustomerCreateDTO {

    @NotNull(message = "{fineros.constraints.firstname.NotNull.message}")
    @Size(min=2, max=17, message = "{fineros.constraints.firstname.Size.message}")
    private String firstName;

    @NotNull(message = "{fineros.constraints.lastname.NotNull.message}")
    @Size(min=2, max=17, message= "{fineros.constraints.lastname.Size.message}")
    private String lastName;

    @NotNull(message = "{fineros.constraints.email.NotNull.message}")
    @Size(min=12, max=50, message= "{fineros.constraints.email.Size.message}")
    @UniqueEmail
    private String email;

    @NotNull(message = "{fineros.constraints.cardnumber.NotNull.message}")
    @Size(min=16, max=16, message= "{fineros.constraints.cardnumber.Size.message}")
    @UniqueCardNumber
    private String cardNumber;

    @NotNull(message = "{fineros.constraints.expdate.NotNull.message}")
    @Size(min=5, max=5, message= "{fineros.constraints.expdate.Size.message}")
    private String expDate;

    @NotNull(message = "{fineros.constraints.cardpassword.NotNull.message}")
    @Size(min=3, max=3, message= "{fineros.constraints.cardpassword.Size.message}")
    private String cardPassword;
```

```
@Retention(RUNTIME)
@Target(FIELD)
@Constraint(validatedBy = {UniqueEmailValid.class})
public @interface UniqueEmail  {
    String message() default "{Fineros.constraints.UniqueField.message}";

    Class<?>[] groups() default { };

    Class<? extends Payload>[] payload() default { };

}
```

```
public class UniqueEmailValid implements ConstraintValidator<UniqueEmail, String>{

    private final CustomerRepository customerRepository;

    public UniqueEmailValid(CustomerRepository customerRepository) {
        this.customerRepository = customerRepository;
    }

    @Override
    public boolean isValid(String email, ConstraintValidatorContext context) {

        return !customerRepository.existsCustomerByEmail(email) ;
    }

}
```

```
@Repository
public interface CustomerRepository extends JpaRepository<Customer, Long> {

    boolean existsCustomerByEmail(String email);


}
```

Testing Using Postman

jdbc:h2:./employeedb
CUSTOMERS
PAYMENTS
INFORMATION_SCHEMA
Users
H2 2.1.214 (2022-06-13)

Run | Run Selected | Auto complete | Clear | SQL statement:

```
SELECT * FROM CUSTOMERS
```

SELECT * FROM CUSTOMERS;

CUSTOMER_NUMBER	CARD_NUMBER	PASSWORD	EMAIL	EXPIRE_DATE	FIRST_NAME	LAST_NAME
1	1234567890123456	845	mert.kaya@gmail.com	07/04	mert	kaya
2	9283746918775467	345	efe.yildirim@gmail.cam	05/07	efe	yildirim

(2 rows, 2 ms)

POST http://localhost:8080/controller/v1/customer Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings

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

```
{
    "firstName":"mahmut",
    "lastName":"bilecik",
    "email":"mert.kaya@gmail.com",
    "cardNumber":"5394758102984657",
    "expDate":"03/04",
    "cardPassword":"123"

}
```


```
            "code": "email"
        }
    ],
    "defaultMessage": "The email address already exists",
    "objectName": "customerCreateDTO",
    "field": "email",
    "rejectedValue": "mert.kaya@gmail.com",
    "bindingFailure": false,
    "code": "UniqueEmail"
  }
],
"path": "/controller/v1/customer"
}
```

7-) A credit card number should be registered by one customer

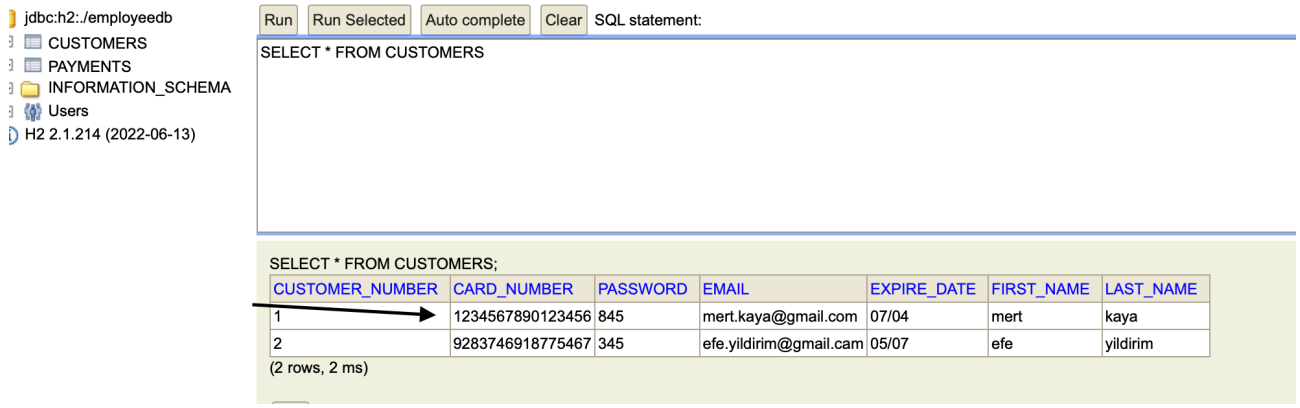
```
public class CustomerCreateDTO {

    @NotNull(message = "{fineros.constraints.firstname.NotNull.message}")
    @Size(min=2, max=17, message = "{fineros.constraints.firstname.Size.message}")
    private String firstName;

    @NotNull(message = "{fineros.constraints.lastname.NotNull.message}")
    @Size(min=2, max=17, message= "{fineros.constraints.lastname.Size.message}")
    private String lastName;

    @NotNull(message = "{fineros.constraints.email.NotNull.message}")
    @Size(min=12, max=50, message= "{fineros.constraints.email.Size.message}")
    @UniqueEmail
    private String email;

    @NotNull(message = "{fineros.constraints.cardnumber.NotNull.message}")
    @Size(min=16, max=16, message= "{fineros.constraints.cardnumber.Size.message}")
    @UniqueCardNumber
    private String cardNumber;

    @NotNull(message = "{fineros.constraints.expdate.NotNull.message}")
    @Size(min=5, max=5, message= "{fineros.constraints.expdate.Size.message}")
    private String expDate;

    @NotNull(message = "{fineros.constraints.cardpassword.NotNull.message}")
    @Size(min=3, max=3, message= "{fineros.constraints.cardpassword.Size.message}")
    private String cardPassword;
```

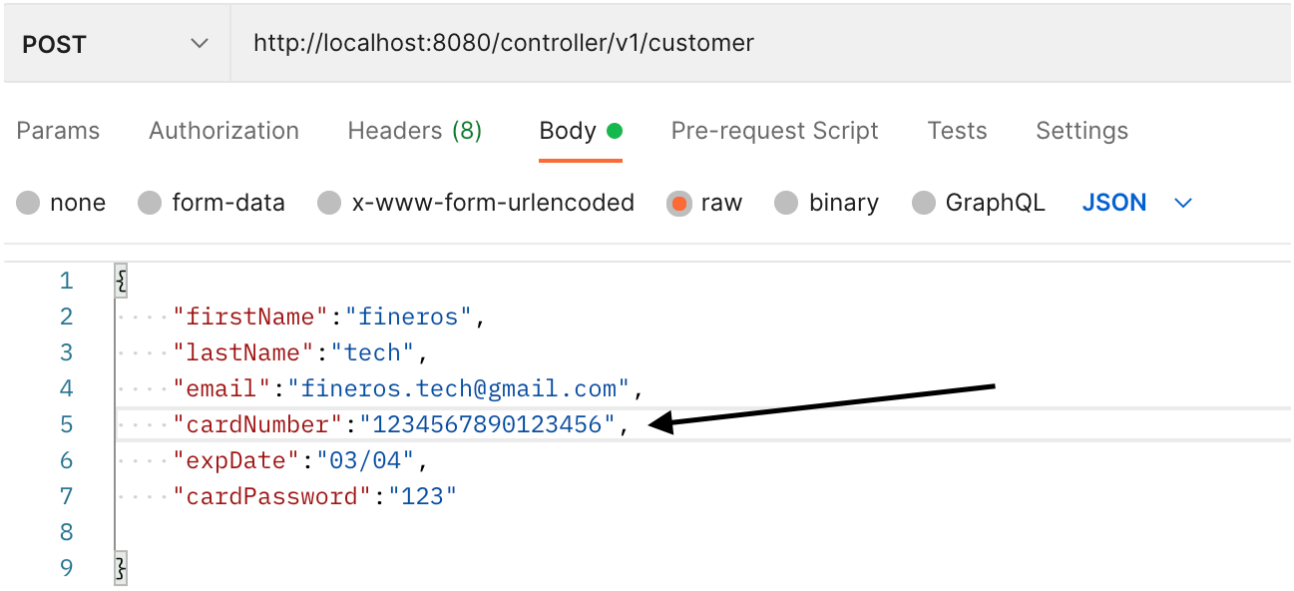
```
@Retention(RUNTIME)
@Target(FIELD)
@Constraint(validatedBy = {UniqueCardNumberValid.class})
public @interface UniqueCardNumber {
    String message() default "{fineros.constraints.UniqueField.message}";

    Class<?>[] groups() default { };

    Class<? extends Payload>[] payload() default { };
}
```

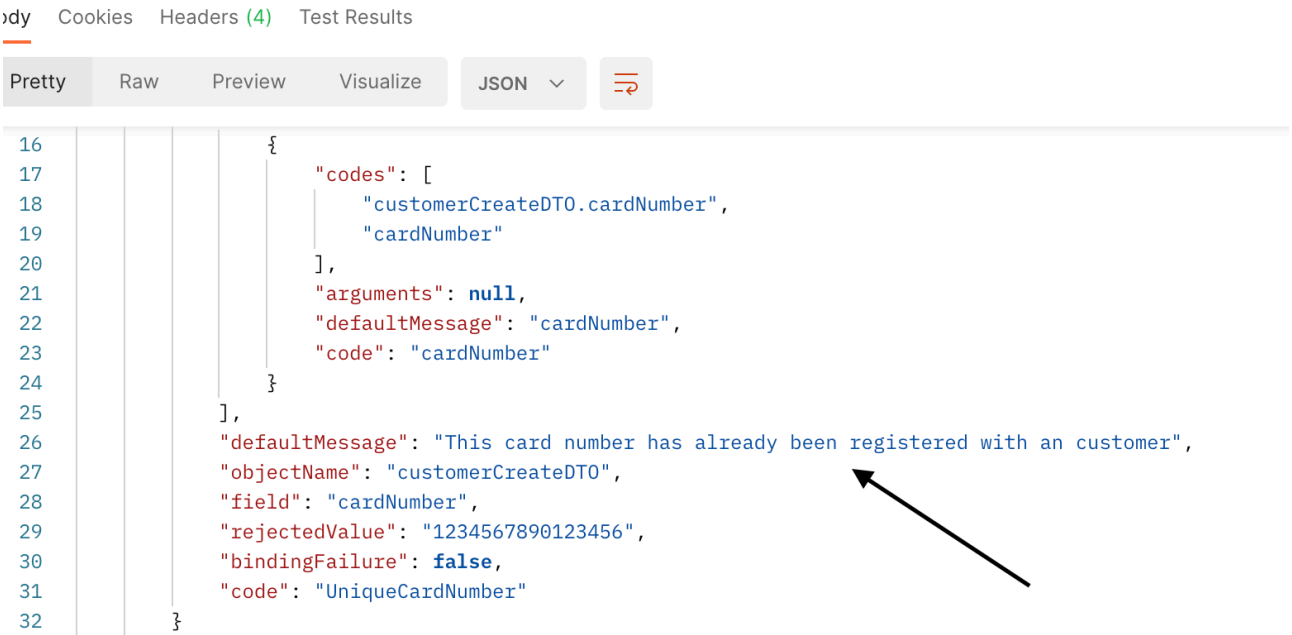
```
public class UniqueCardNumberValid implements ConstraintValidator<UniqueCardNumber, String> {

        private final PaymentRepository paymentRepository;

        public UniqueCardNumberValid(PaymentRepository paymentRepository) {
            this.paymentRepository = paymentRepository;
        }
    @Override
    public boolean isValid(String cardNumber, ConstraintValidatorContext context) {

        return !paymentRepository.existsPaymentByCardNumber(cardNumber);
    }

}
```

```
@Repository
public interface PaymentRepository extends JpaRepository<Payment, Long> {

    List<Payment> findByCardNumber(String cardNumber);

    List<Payment> findByDataCreatedBetween(Date startDate, Date endDate);

    boolean existsPaymentByCardNumber(String cardNumber);

}
```

Testing Using Postman

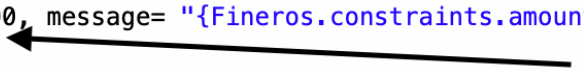






8-) Payment amount must be greater then zero.

```
public class PaymentCreateDTO {

    @NotNull(message = "{Fineros.constraints.cardnumber.NotNull.message}")
    @Size(min=16, max=16, message= "{Fineros.constraints.cardnumber.Size.message}")
    @UniqueCardNumber
    private String cardNumber;

    @NotNull(message = "{Fineros.constraints.amount.NotNull.message}")
    @Size(min=1, max=10000, message= "{Fineros.constraints.amount.Size.message}")
    private Long amount;
```



None of fields for the both tables in the database cannot be null as well and some mix and max with length of the fields were implemented. If a customer entered a value that the system does not expect, the customer will get a message for guidance to enter the right form of the entry value

H2 database setup on the property of the spring boot project

```
spring.datasource.url=jdbc:h2:./employeedb
spring.datasource.username=sa
spring.datasource..password=sa123

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true

spring.jpa.properties.hibernate.format-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect
```

The Not Completed Tasks

It was the first time for me on working with the Spring Boot. I had to make several researches including reading articles, watching videos and deep checking the dependencies, the classes, the interfaces and their methods that the spring boot provides. Since I was not familiar with them, I had several errors on doing the tasks that were completed and were not completed. Each error took much time for me to solve as not all solutions that internet provides worked on my user case since I had different entries, variables and some imports. I did not have time for either unit testing or integration testing. I still did some research and gave a few tries on my own methods and classes but since I was not familiar with the spring boot test features such like @Mock, @InjectMock and more, all of my tries got either error or did not executer what I meant for. Unfortunately that's why I had to skipped that part as I did not have time. The situation is also same with Swagger.

As I described the idea for the design at the beginning of this report, I did not want to use the entities directly from the entity classes but for the method called, getPaymentsByCardNumber I failed to execute the idea. There was a simple error that says the serializable was not found in the class called PaymentViewDTO where I actually implemented the serializable. I am sure that the error I got is pretty easy to figure it out but unfortunately I did not have any more time to spend on this and had to leave the way it works.

For the method called, getPaymentsByDate, I implemented most of the code part and maybe all of the code part. However, I got an error while I tried to enter date parameters and the error is because the page side accepts my date entries as string and it should be converted to the date which I tried a few solutions that did not worked and I had to leave it as it is since I did not have time. This method also takes the entity class directly as a parameter, and it is because first I wanted to test it to be sure it's working and the I would change it in a way that it does not reach with the entry class directly same as I wanted do for the method called, getPaymentsByCardNumber

I created a class that is called Card for the credit card object to have it's own variables and methods but since I was not familiar with database integration in spring boot, I could not store the card object's variables into customer class where the customer table was being created as well as the fields of the table. I tried to find a solution but lost much time and decided to go with an other way which I never prefer.

Make payment method should include more transactions and implementations. Such like checking the amount inside, checking the card number and checking the order as well in where the order class

could be created as well. Since I did not want to check any same work that has been done online for not being affected by them, I wanted to first understand the spring boot features and then did everything on my own. But unfortunately I did not have time to complete this issues.

The Future objectives;

- Test features of the spring boot project should be well known and implemented for both unit testing and integration testing.
- The methods that took directly entity classes as parameters should be changed to the acceptable version.
- Swagger should be searched and well known to integrate it with the project.
- The error for GetPaymentsByDate should be solved by converting the string entry as a date entry.
- Card Class should be created and the properties of a card object like number, names, date, password should be gotten from the card class objects.
- MakeNewPayment method should be implemented with more implementation to include tasks like getting the card entries, order entries and check values from the database and then perform the task
- The order class can be created and a few features like the name, the price and so on can be used.