UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Euphoria jewelry

Name: Bonțea Carmen-Diana
Group:

Table of Contents

Deliverable 1

Project Specification

The purpose of this project is to monitoring a website of a jewelry store. It focuses on the capabilities of those who manage the store and of customers.

Functional Requirements

The application can be used by 2 types of users: customers and admin.

Customer

The customer has the following capabilities:

- register and log in the application
- add products to the cart
- delete products from the cart
- place order
- add and delete items from my favorite list
- view and modify personal information
- view order history
- review products

Admin

The admin has the following capabilities:

- log in in the application
- add, delete and update product information
- order management

Use Case Model 1

Use Cases Identification

Use case: Add new item
Primary actor: Admin
Main success scenario:

1) user logins into the account
2) user clicks on Add new item button
3) user enters item details
4) user clicks Save item button
5) a confirmation message is displayed that the item has been added successfully

Extensions: In the case of an error when creating an item (ex. Connection drop) an error message is displayed.

Use case: Update personal information
Primary actor: Customer
Main success scenario:

1) user logins into the account
2) user clicks on Info button

3) user clicks on Change Info button
4) a confirmation message is displayed that the information has been updated successfully

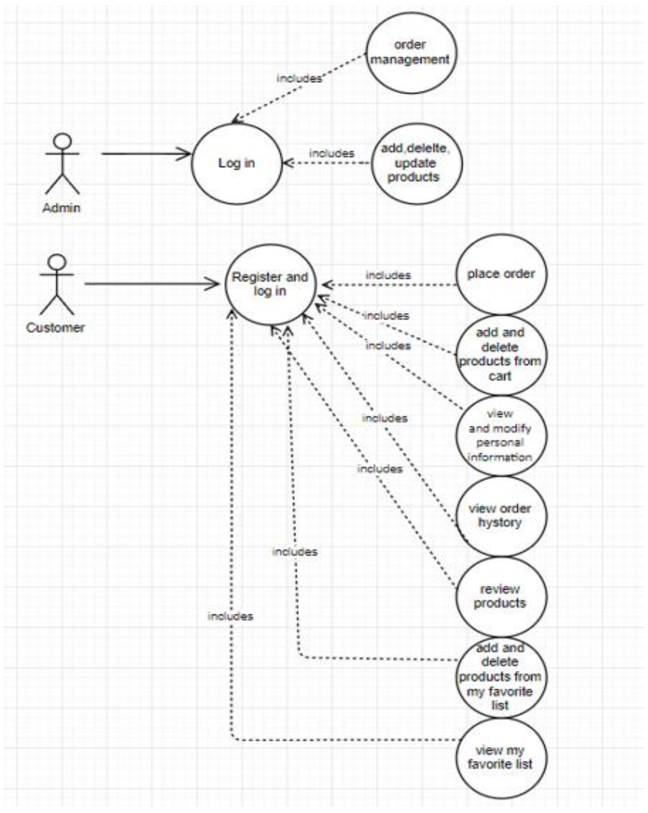
Extensions: In the case of an error when updating the information (ex. Connection drop) an error message is displayed.

Use case: Place an order
Primary actor: Customer
Main success scenario:

1) user logins into the account
2) user clicks on the category (rings, earrings, chains, talismans, bracelets) from which he wants to buy
3) user selects the product he wants to buy
4) user clicks on Add button
5) a confirmation message is displayed that the product has been added successfully
6) user clicks on View cart button
7) user clicks on Finish order button
8) user fills in the fields required to deliver the order
9) user clicks on Place order button
10) a confirmation message is displayed that the order has been placed successful

Extensions: In the case of an error when placing the order (ex. Fields are not all filled) an error message is displayed.

UML Use Case Diagrams



Supplementary Specification

Non-functional Requirements

Availability

Availability describes how likely the system is accessible for a user at a given point in time. While it can be expressed as a probability percentage, it may also be defined as a percentage of time the system is accessible for operation during some time period. The platform should be available 98% of the time, so that customers and the admin can perform operations on it whenever they need.

Security

Security is used to protect sensitive data and access to the database. The application will not allow any action to be taken unless a valid authentication has been performed. A customer cannot place orders or view personal data and the admin cannot add, delete or update product information unless they are logged in. Also, the application does not grant access until the user creates a strong password for registration.

Portability and compatibility

Portability and compatibility means how efficiently one system works in one environment compared to another. In addition to this, it refers to what hardware, operating systems, browsers (and their versions), the software is running. This application should run on any computer which has as operating system, Windows 7 or later versions and on any browser.

Usability

Usability can be described as the capacity of a system to provide a condition for its users to perform the tasks safely and efficiently while enjoying the experience.[1] In software engineering, usability is the degree to which a software can be used by specified consumers to achieve objectives with effectiveness, efficiency, and satisfaction in a context of use. The system should be accessed via a browser, application which is installed by default on any OS. The interface should be simple, intuitive and compact, so that users have a pleasant experience and do not encounter difficulties when using the platform.

Design Constraints

The system must be a web application, developed using Spring Boot. The data is stored in a database and the access is done with JPA. The backed is developed in Java 17. No action on the web app can be done without an initial login. The application must follow the MVC pattern.

Glossary

JPA

Java Persistence API is a Java specification for accessing, persisting, and managing data between Java objects / classes and a relational database.

JDBC API

The Java Database Connectivity (JDBC) API provides universal data access from the Java programming language. Using the JDBC API, you can access virtually any data source, from relational databases to spreadsheets and flat files. JDBC technology also provides a common base on which tools and alternate interfaces can be built.

Lombok

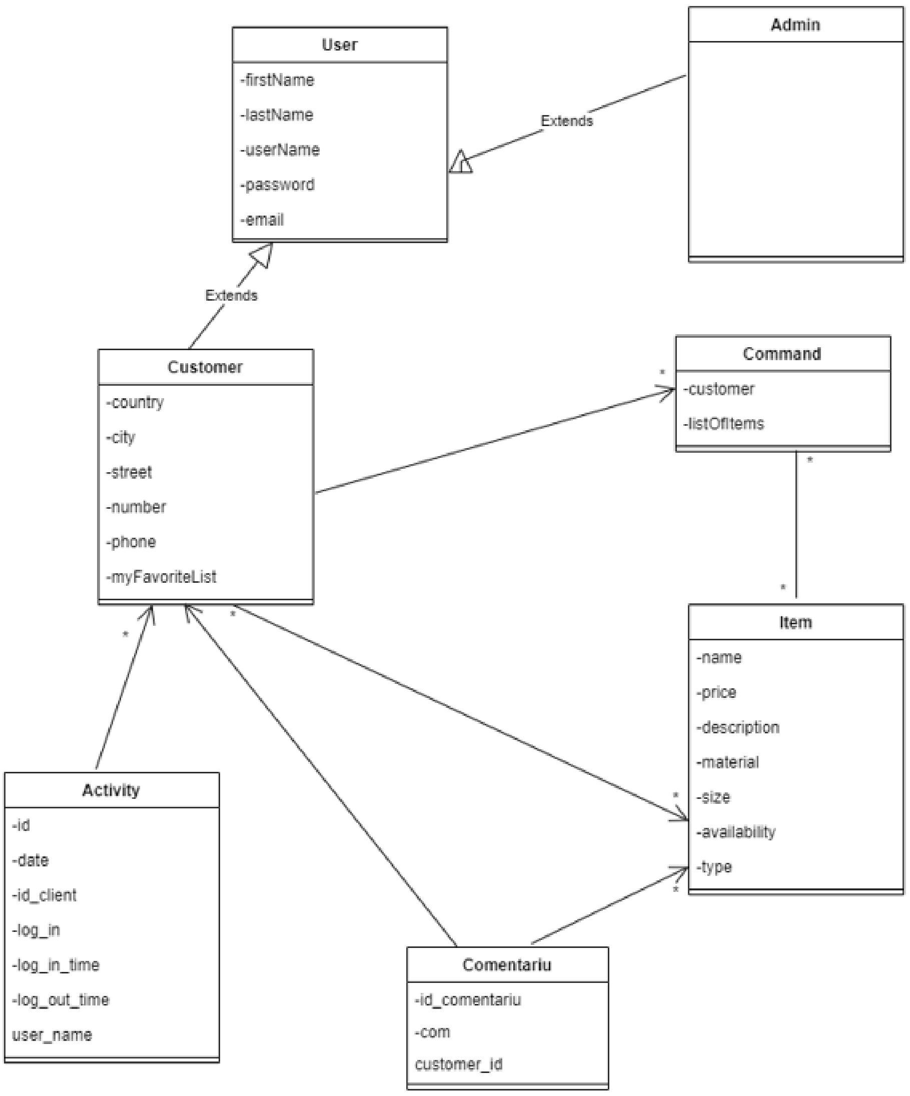
Project Lombok is a Java library tool that generates code for minimizing boilerplate code. The library replaces boilerplate code with easy-to-use annotations.

Spring boot

Java Spring Boot (Spring Boot) is a tool that makes developing web application and microservices with Spring Framework faster and easier through three core capabilities: Autoconfiguration. An opinionated approach to configuration. The ability to create standalone applications.

Deliverable 2

Domain Model



Architectural Design

Conceptual Architecture

Because the purpose of this project is to simulate the behavior of the relationship between a client and a website, I chose as architectural design client server , layered and as pattern Model-View -Controller.

Client-Server

The client-server model, or client-server architecture, is a distributed application framework dividing tasks between servers and clients, which either reside in the same system or communicate through a computer network or the Internet. The client relies on sending a request to another program in order to access a service made available by a server. The server runs one or more programs that share resources with and distribute work among clients.

Why Client-Server?

I chosed this architecture style, because data that is transferred through client-server protocols are platform-agnostic. A single server hosting all the required data in a single place facilitates easy protection of data and management of user authorization and authentication. Resources such as network segments, servers, and computers can be added to a client-server network without any significant interruptions. Data can be accessed efficiently without requiring clients and the server to be in close proximity. All nodes in the client-server system are independent, requesting data only from the server, which facilitates easy upgrades, replacements, and relocation of the nodes.

Layered

Layered architectures are said to be the most common and widely used architectural framework in software development. It is also known as an n-tier architecture and describes an architectural pattern composed of several separate horizontal layers that function together as a single unit of software. A layer is a logical separation of components or code.

Why Layered?

The framework is simple and easy to learn and implement. There is reduced dependency because the function of each layer is separate from the other layers. Testing is easier because of the separated components, each component can be tested individually. Cost overheads are fairly low.

Model-View-Controller(MVC)

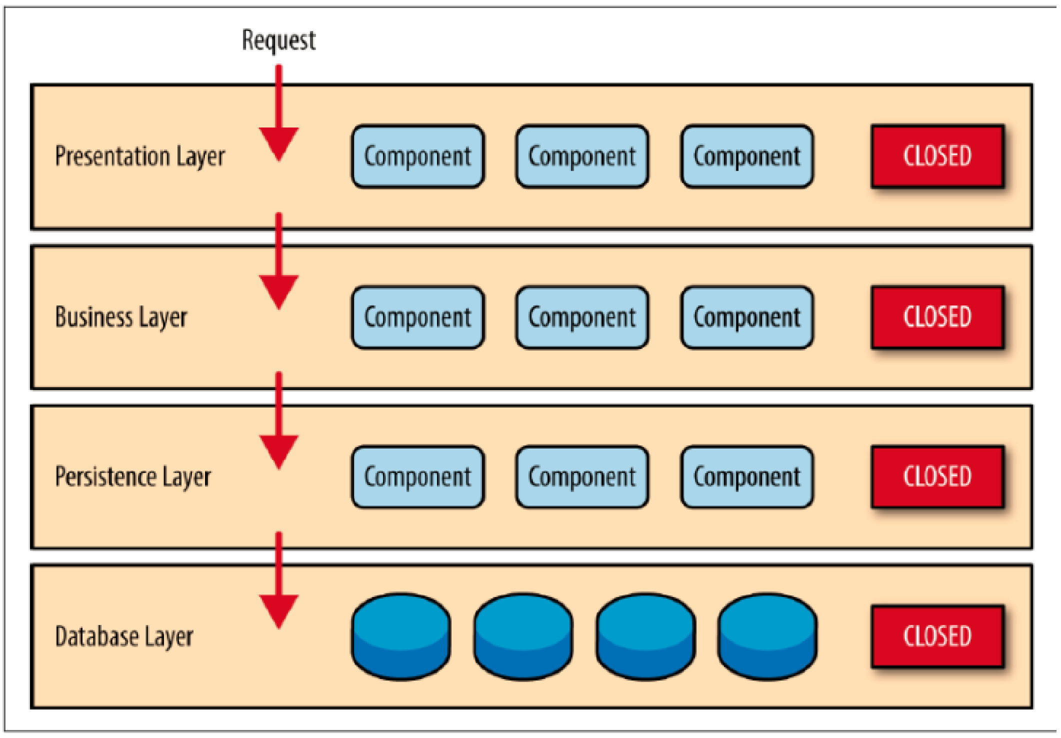
MVC stands for model-view-controller. Model refers to the backend that contains all the data logic, view specifies the frontend or graphical user interface (GUI) and Controller is the brains of the application that controls how data is displayed.

MVC patterns separate the input, processing, and output of an application. This model divided into three interconnected parts called the model, the view, and the controller. All of the three above given components are built to handle some specific development aspects of any web or .net application development.

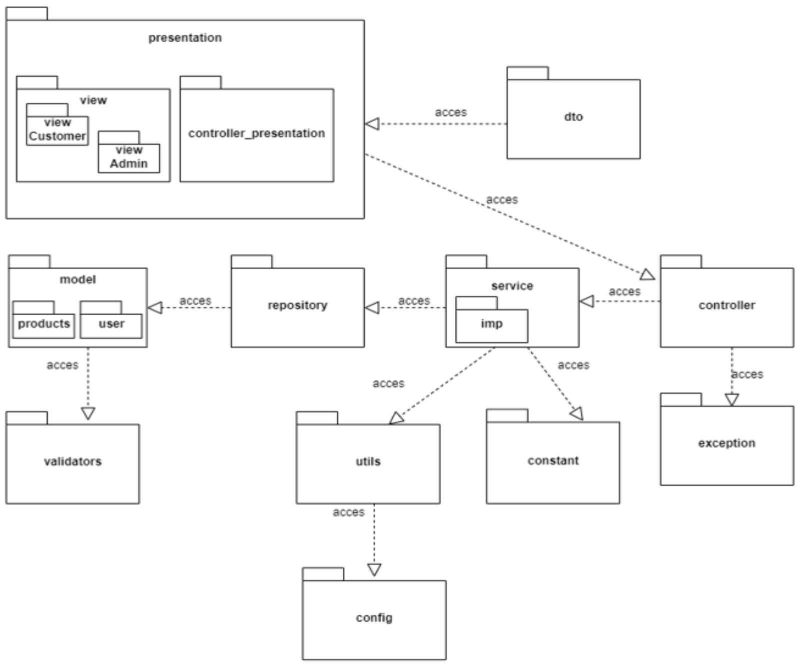
In the MVC application development, the controller receives all requests for the application and then instructs the model to prepare any information required by the view. The view uses that data prepared by the controller to bring the final output.

Why MVC?

The MVC pattern helps you break up the frontend and backend code into separate components. This way, it's much easier to manage and make changes to either side without them interfering with each other. MVC supports rapid and parallel development. If an MVC model is used to develop any particular web application then it is possible that one programmer can work on the view while the other can work on the controller to create the business logic of the web application.

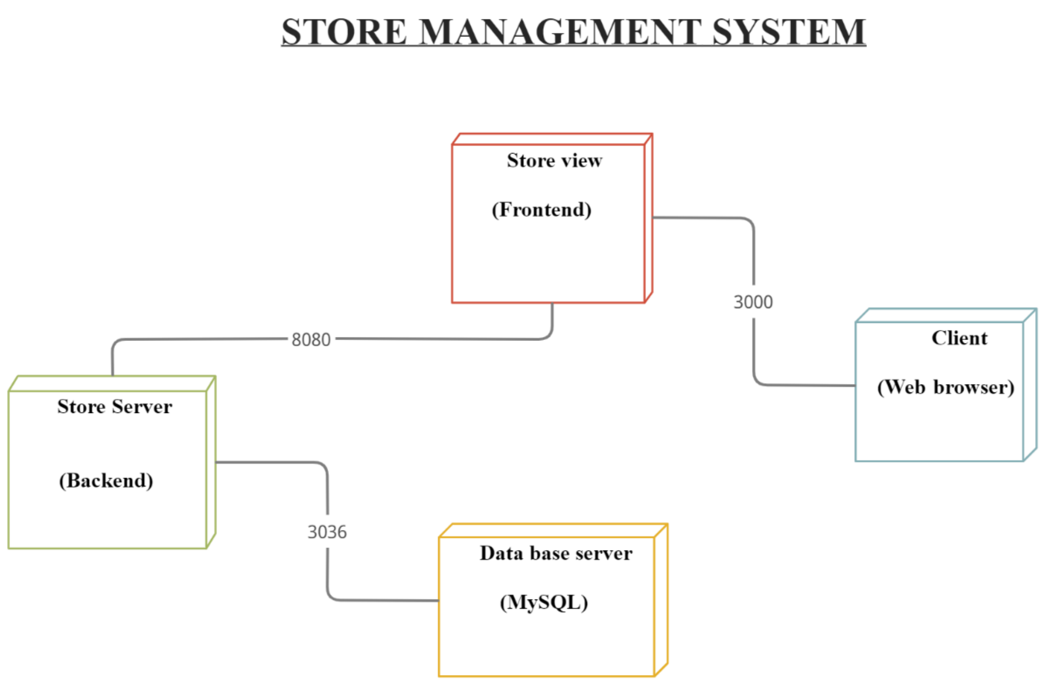


Package Design

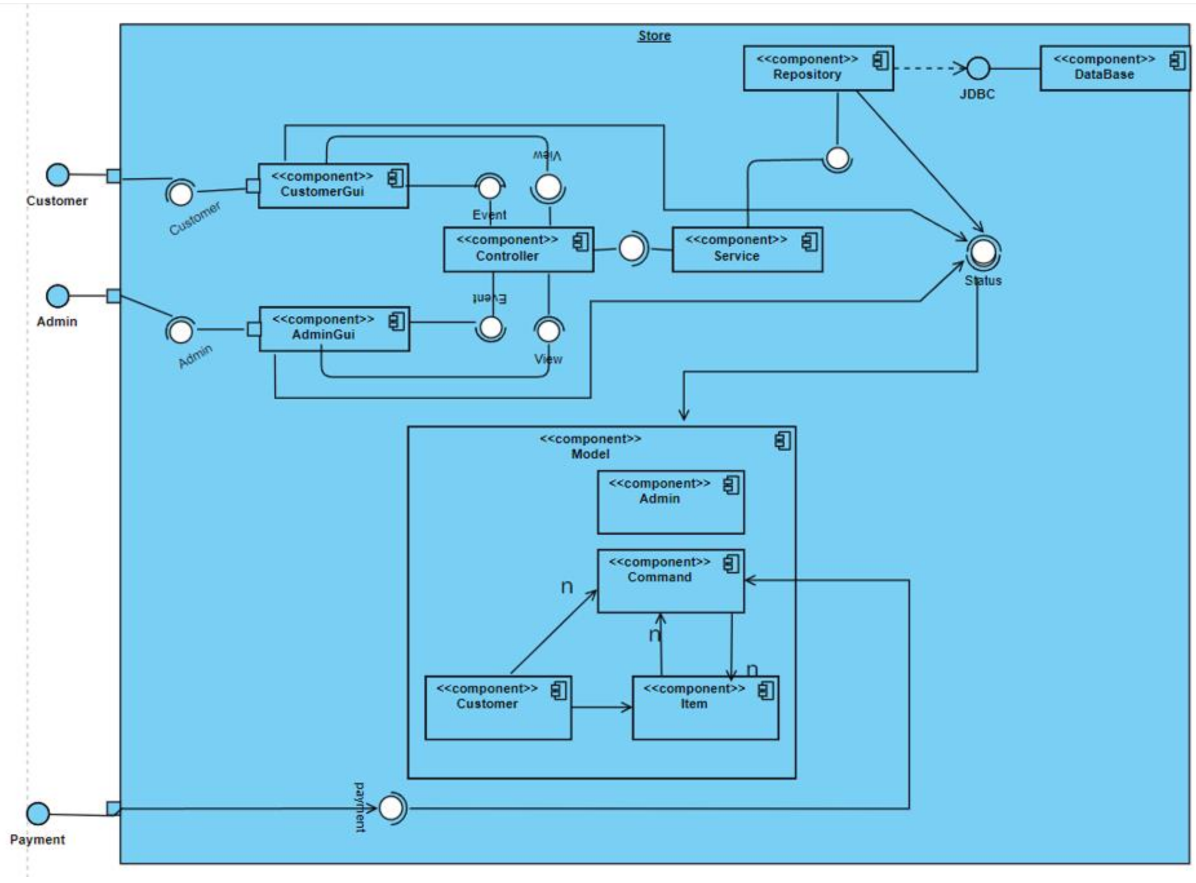


Component and Deployment Diagram

Deployment diagram



Component Diagram

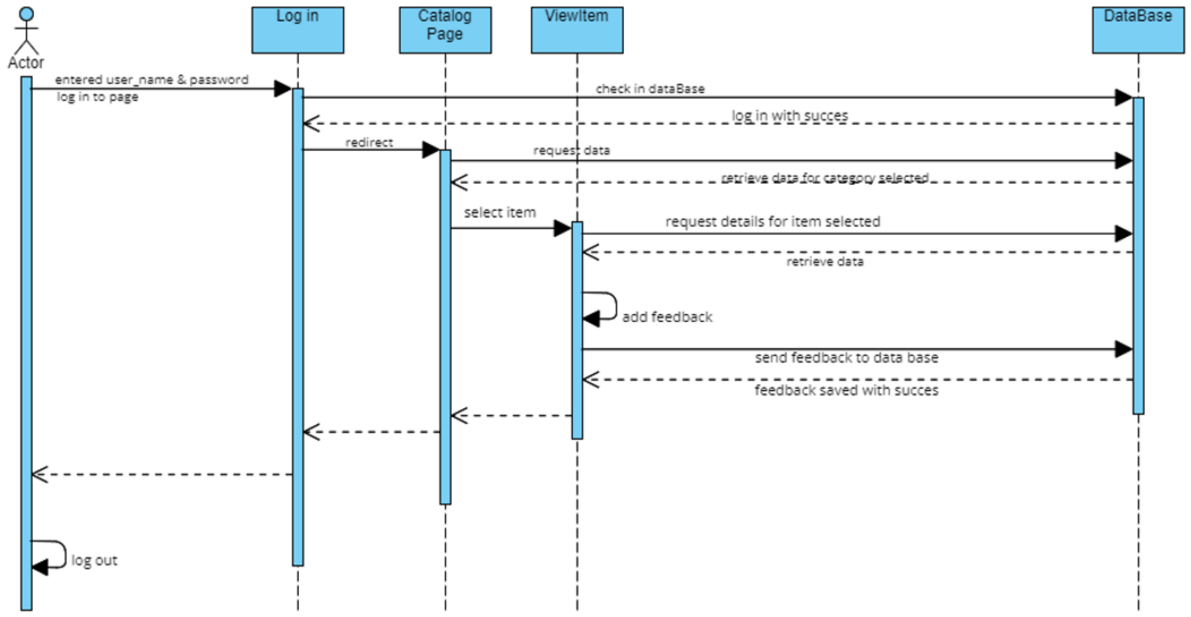


Deliverable 3

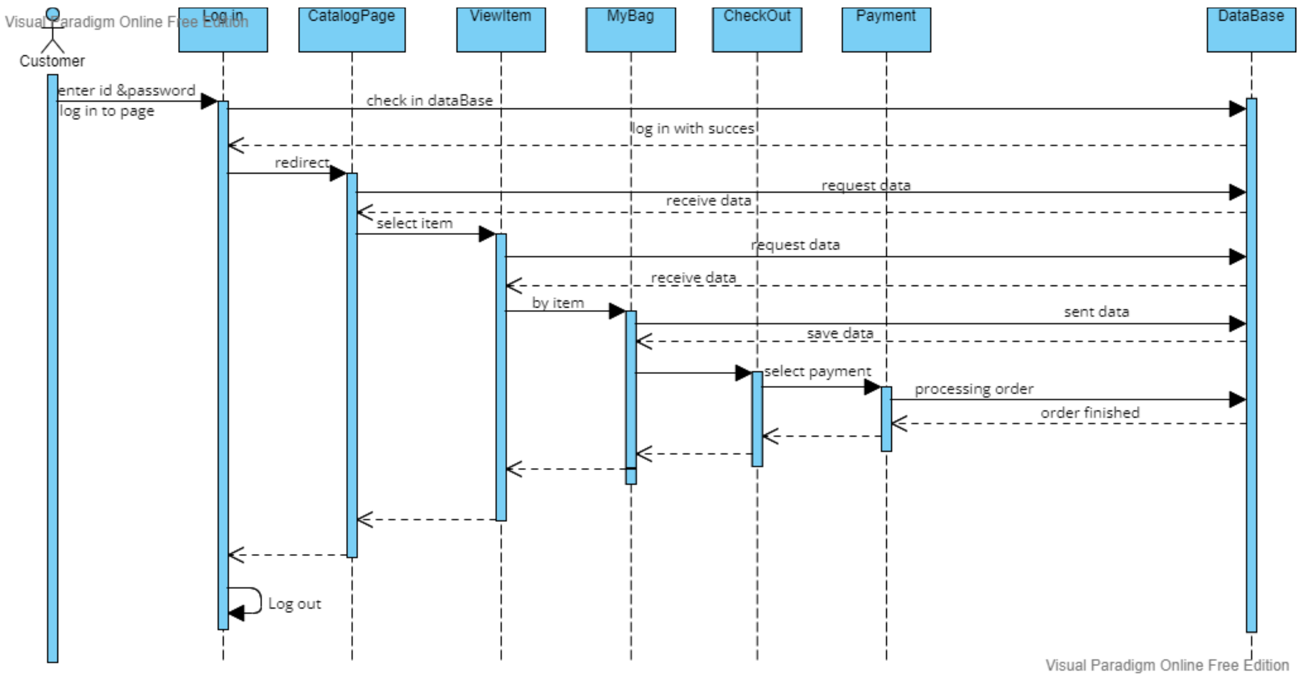
Design Model

Dynamic Behavior

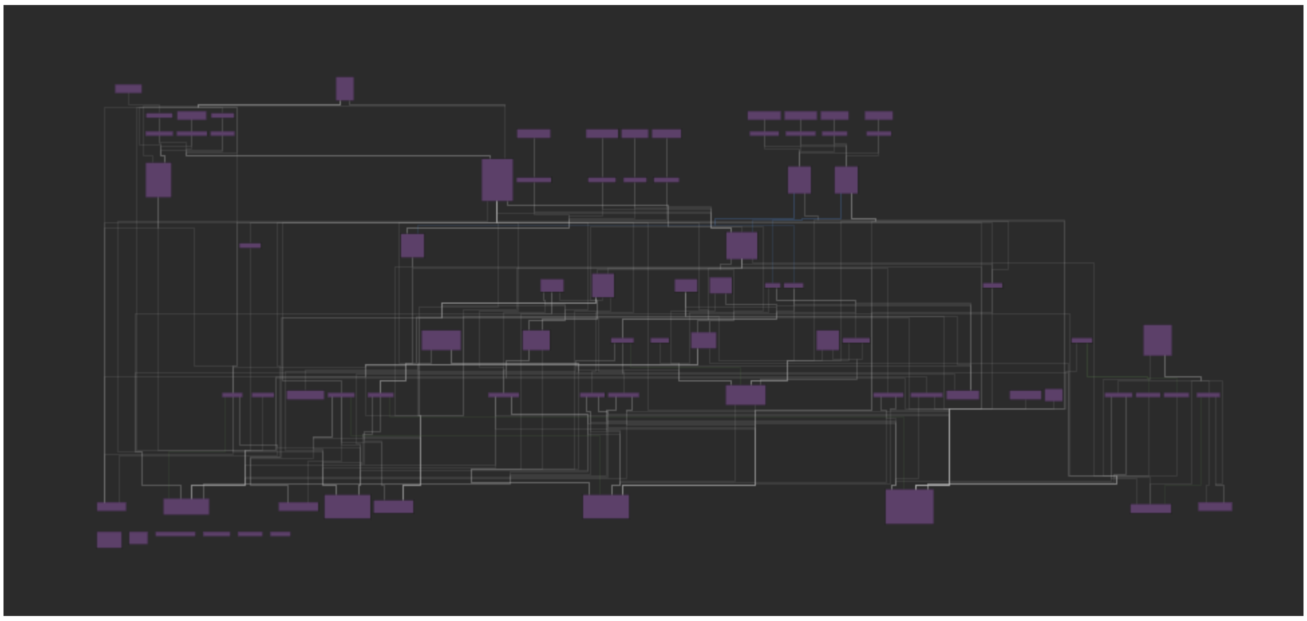
Customer add feedback for a product

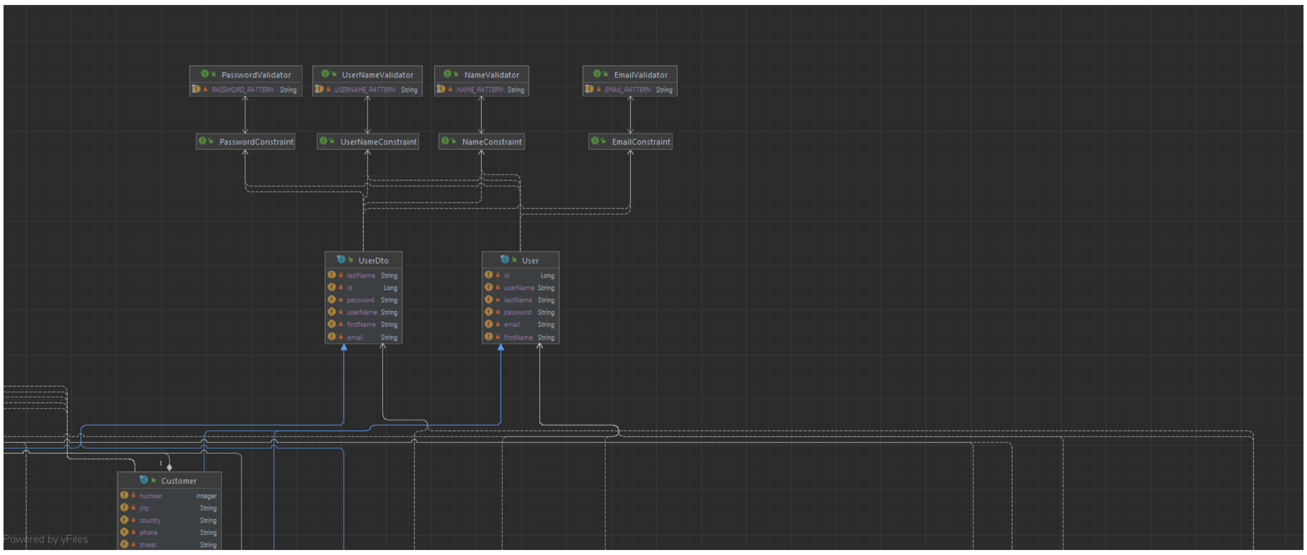


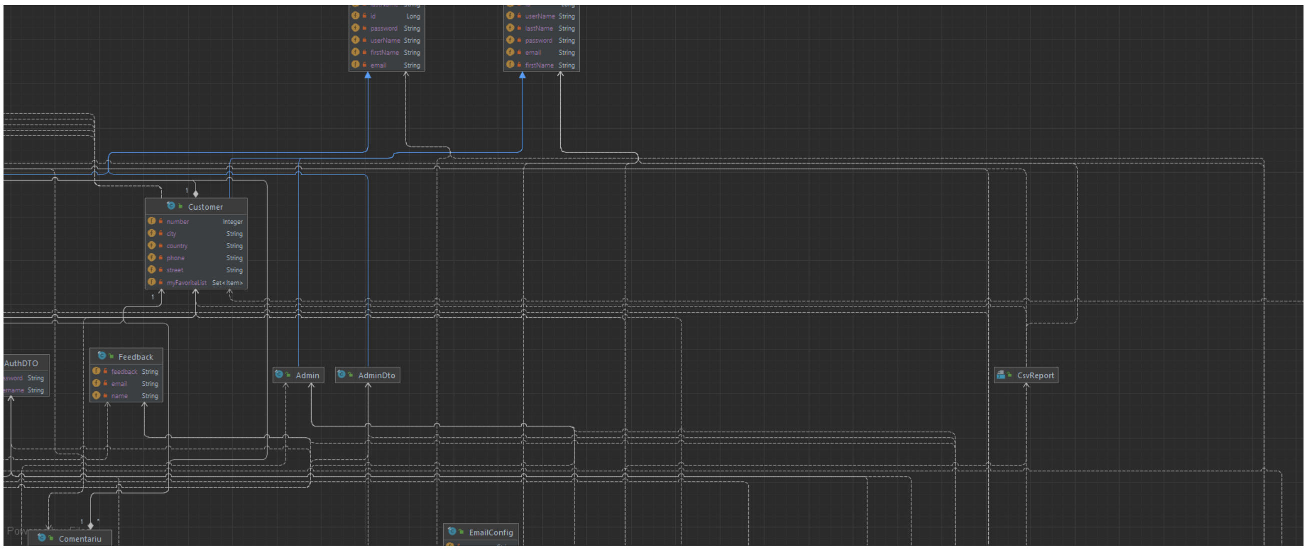
Customer buy an Item

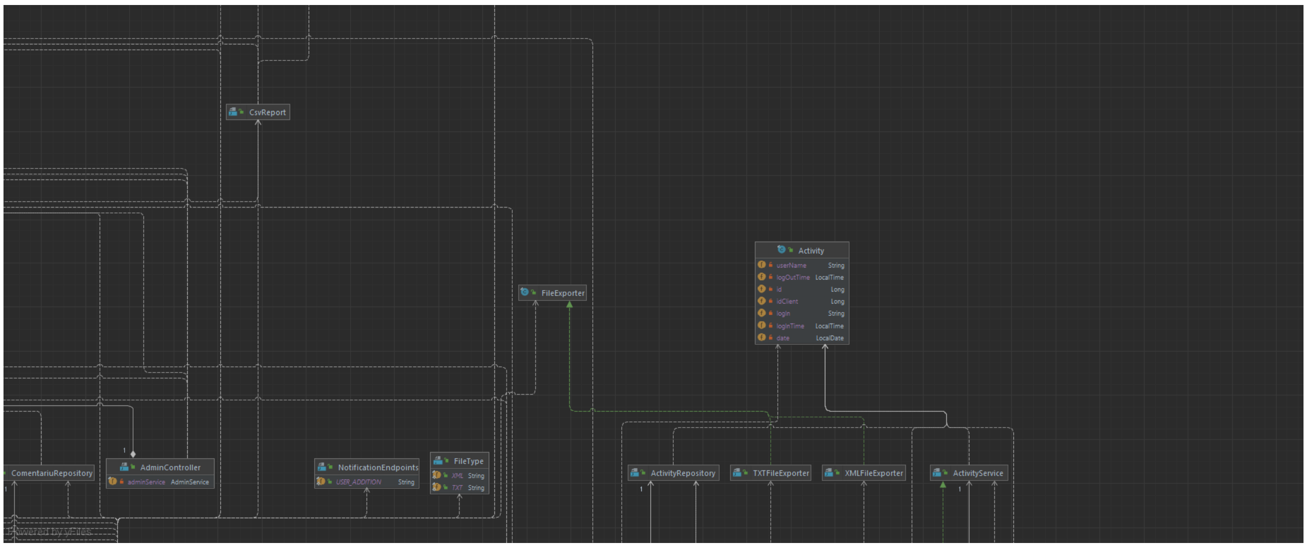


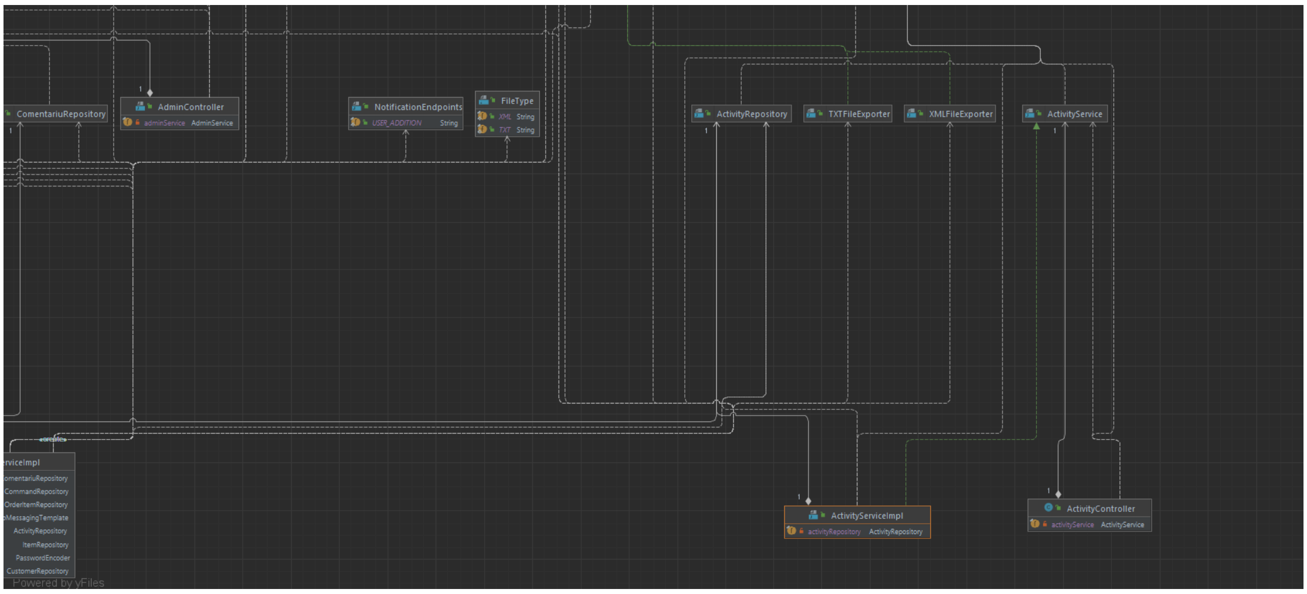
Class Diagram

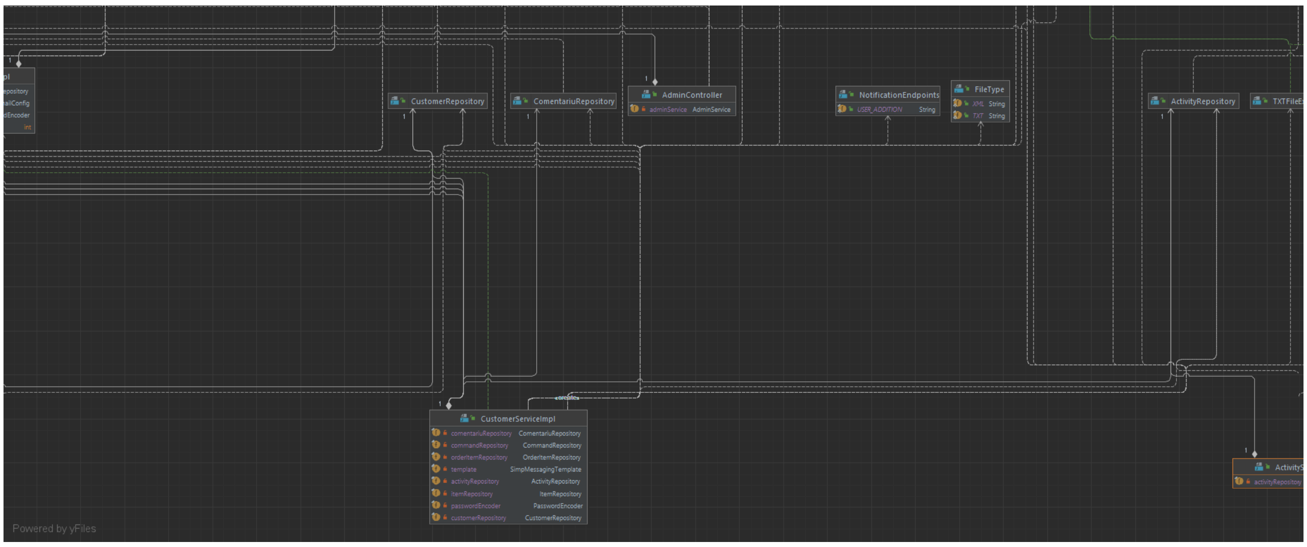


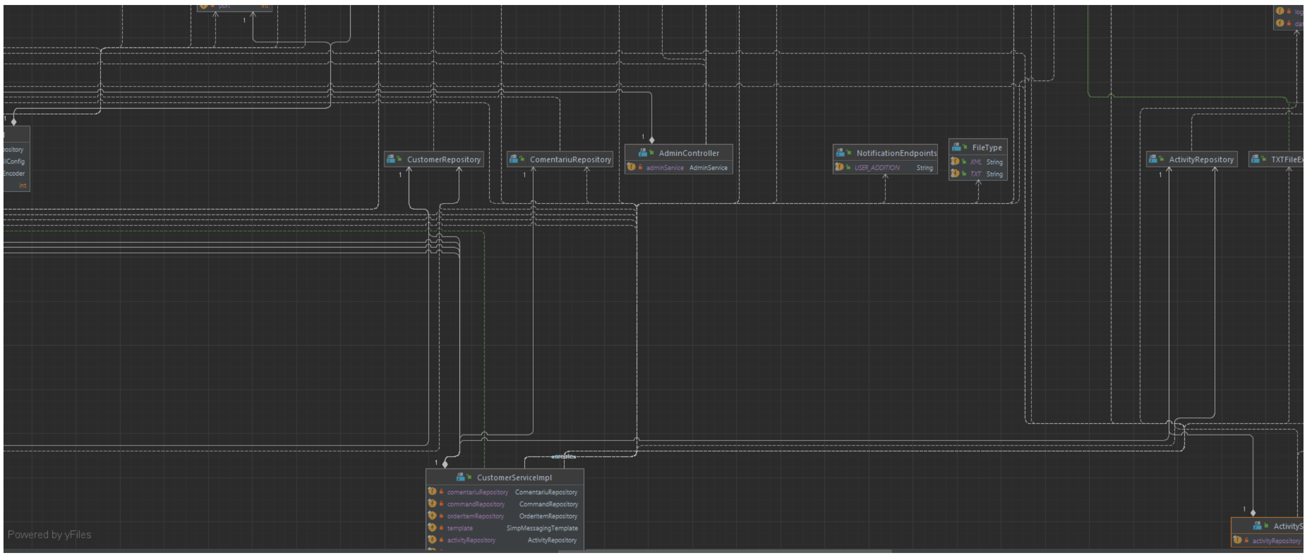

PasswordValidator
PASSWORD_PATTERN String
UserNameValidator
USERNAME_PATTERN String
NameValidator
NAME_PATTERN String
EmailValidator
EMAIL_PATTERN String
PasswordConstraint
UserNameConstraint
NameConstraint
EmailConstraint
UserDto
lastName String
id Long
password String
userName String
firstName String
email String
User
id Long
userName String
lastName String
password String
email String
firstName String
Customer
number Integer
city String
country String
phone String
street String
Powered by yFiles

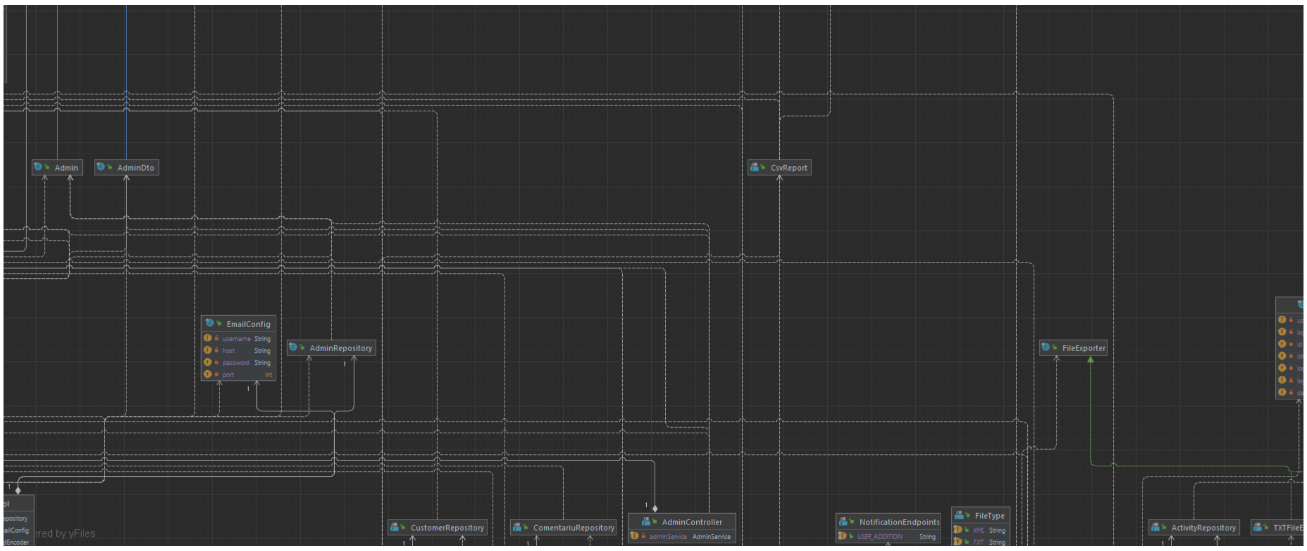

id Long
password String
userName String
firstName String
email String
userName String
lastName String
password String
email String
firstName String
Customer
number Integer
city String
country String
phone String
street String
myFavoriteList Set<Item>
AuthDTO
String
String
Feedback
feedback String
email String
name String
Admin
AdminDto
CsvReport
EmailConfig
Comentariu

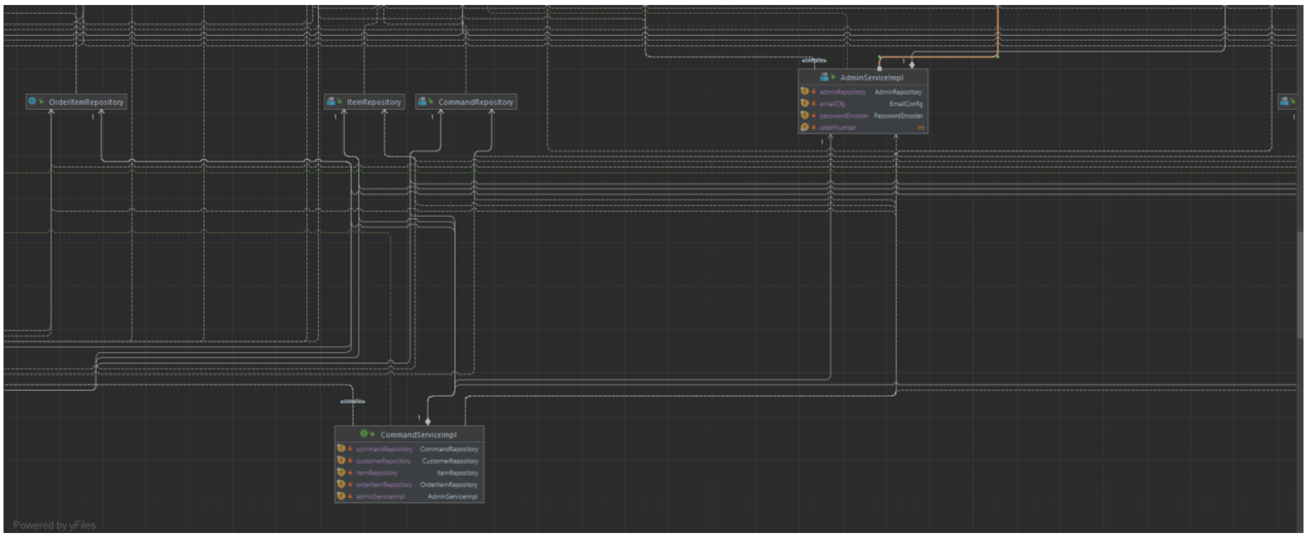

CsvReport
Activity
userName String
logOutTime LocalTime
id Long
idClient Long
login String
loginTime LocalTime
date LocalDate
FileExporter
ComentariuRepository
AdminController
adminService AdminService
NotificationEndpoints
USER_ADDITION String
FileType
XML String
TXT String
ActivityRepository
TXTFileExporter
XMLFileExporter
ActivityService
Powered by yFiles

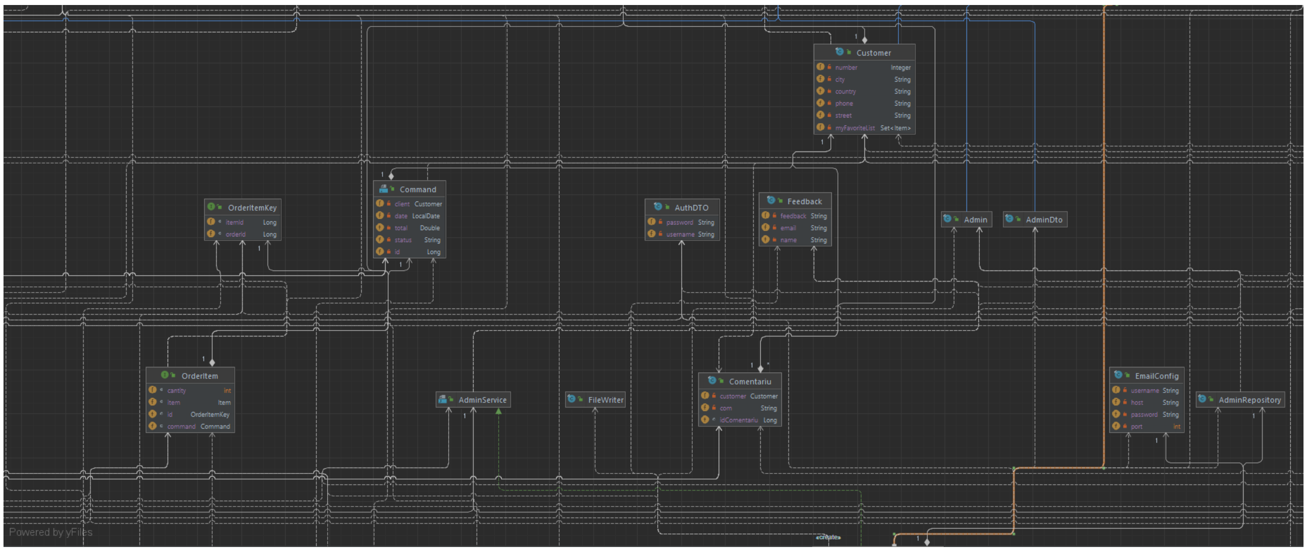

ComentariuRepository
AdminController
adminService AdminService
NotificationEndpoints
USER_ADDITION String
FileType
XML String
TXT String
ActivityRepository
TXTFileExporter
XMLFileExporter
ActivityService
«create»
rviceImpl
ComentariuRepository
CommandRepository
OrderItemRepository
SimpMessagingTemplate
ActivityRepository
ItemRepository
PasswordEncoder
CustomerRepository
ActivityServiceImpl
activityRepository ActivityRepository
ActivityController
activityService ActivityService
Powered by yFiles


CustomerRepository
ComentariuRepository
AdminController
adminService AdminService
NotificationEndpoints
USER_ADDITION String
FileType
XML String
TXT String
ActivityRepository
TXTFileE
«create»
CustomerServiceImpl
comentariuRepository ComentariuRepository
commandRepository CommandRepository
orderItemRepository OrderItemRepository
template SimpMessagingTemplate
activityRepository ActivityRepository
itemRepository ItemRepository
passwordEncoder PasswordEncoder
customerRepository CustomerRepository
ActivityS
activityRepository
Powered by yFiles

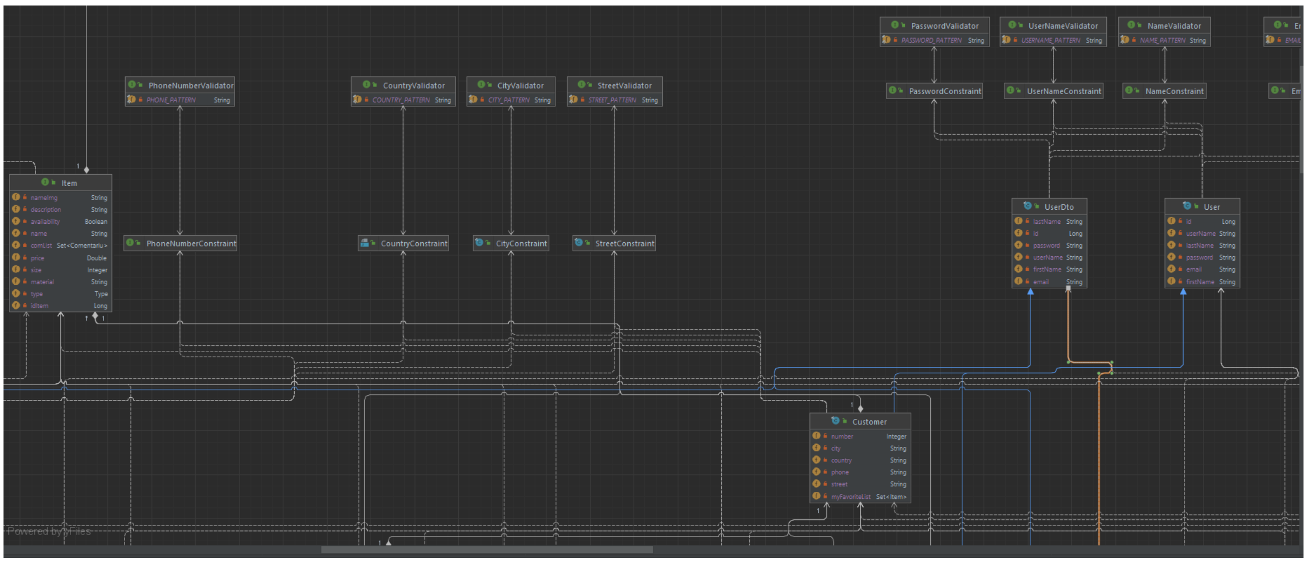

CustomerRepository
ComentariuRepository
AdminController
adminService AdminService
NotificationEndpoints
USER_ADDITION String
FileType
XML String
TXT String
ActivityRepository
CustomerServiceImpl
comentariuRepository ComentariuRepository
commandRepository CommandRepository
orderItemRepository OrderItemRepository
template SimpMessagingTemplate
activityRepository ActivityRepository
activityRepository
Powered by yFiles


Admin
AdminDto
CsvReport
EmailConfig
username String
host String
password String
port int
AdminRepository
FileExporter
CustomerRepository
ComentariuRepository
AdminController
adminService AdminService
NotificationEndpoints
USER_ADDITION String
FileType
XML String
TXT String
ActivityRepository

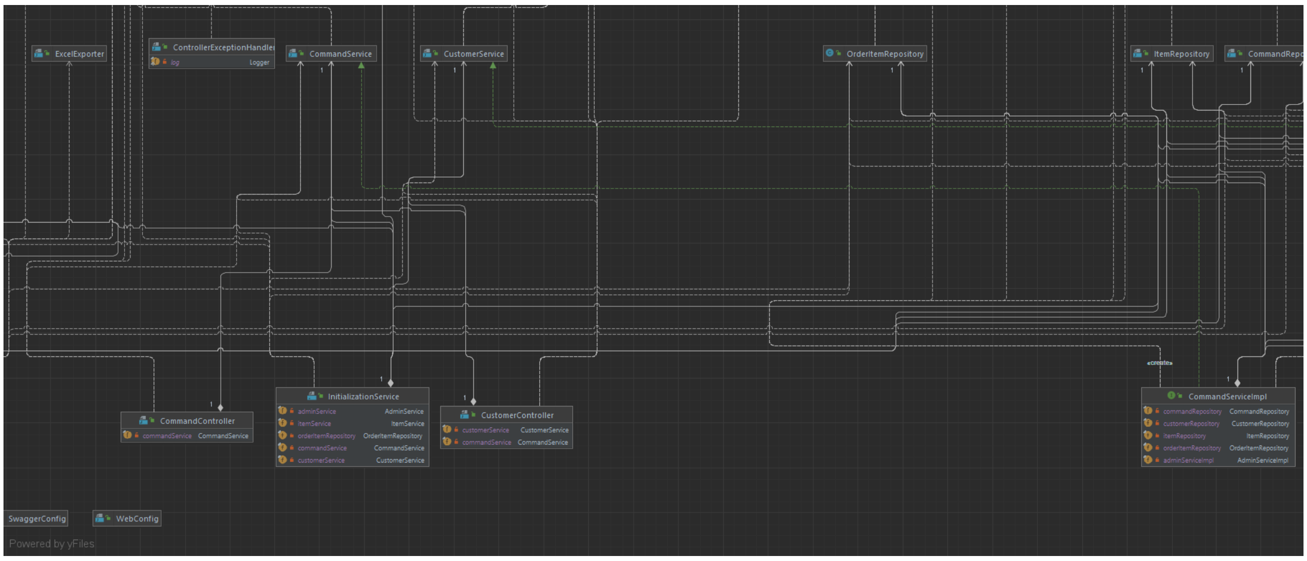

OrderItemRepository
ItemRepository
CommandRepository
AdminServiceImpl
adminRepository AdminRepository
emailCfg EmailConfig
passwordEncoder PasswordEncoder
orderNumber int
CommandServiceImpl
commandRepository CommandRepository
customerRepository CustomerRepository
itemRepository ItemRepository
orderItemRepository OrderItemRepository
adminServiceImpl AdminServiceImpl
Powered by yFiles



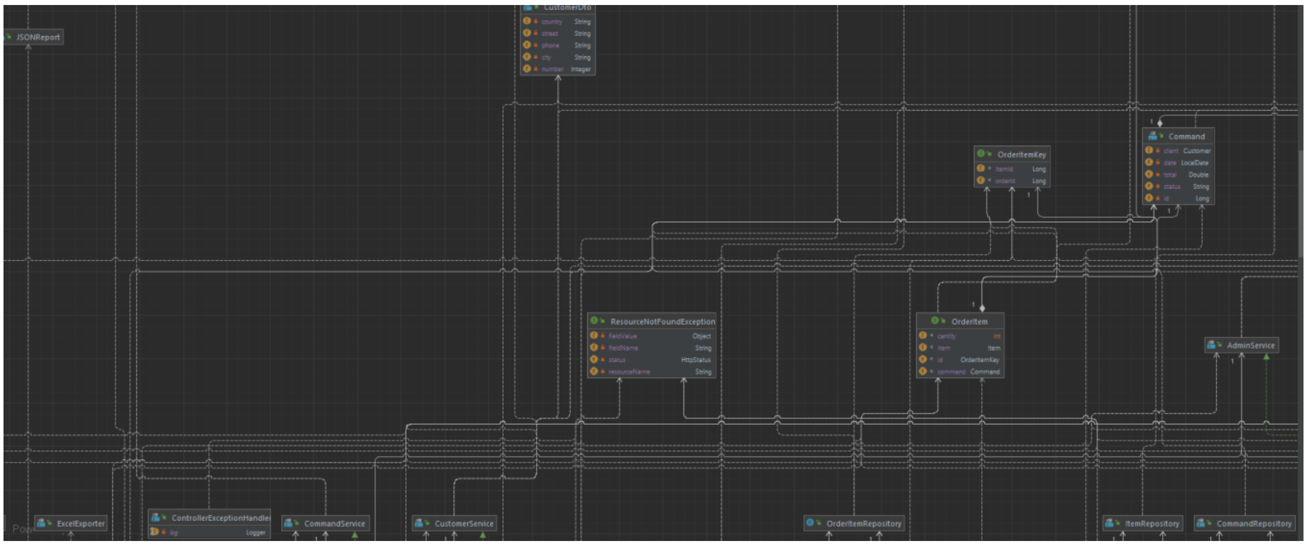
Customer
number Integer
city String
country String
phone String
street String
myFavoriteList Set<Item>
Command
client Customer
date LocalDate
total Double
status String
id Long
OrderItemKey
itemId Long
orderId Long
AuthDTO
password String
username String
Feedback
feedback String
email String
name String
Admin
AdminDto
OrderItem
cantity int
item Item
id OrderItemKey
command Command
AdminService
FileWriter
Comentariu
customer Customer
com String
idComentariu Long
EmailConfig
username String
host String
password String
port int
AdminRepository
«create»
Powered by yFiles



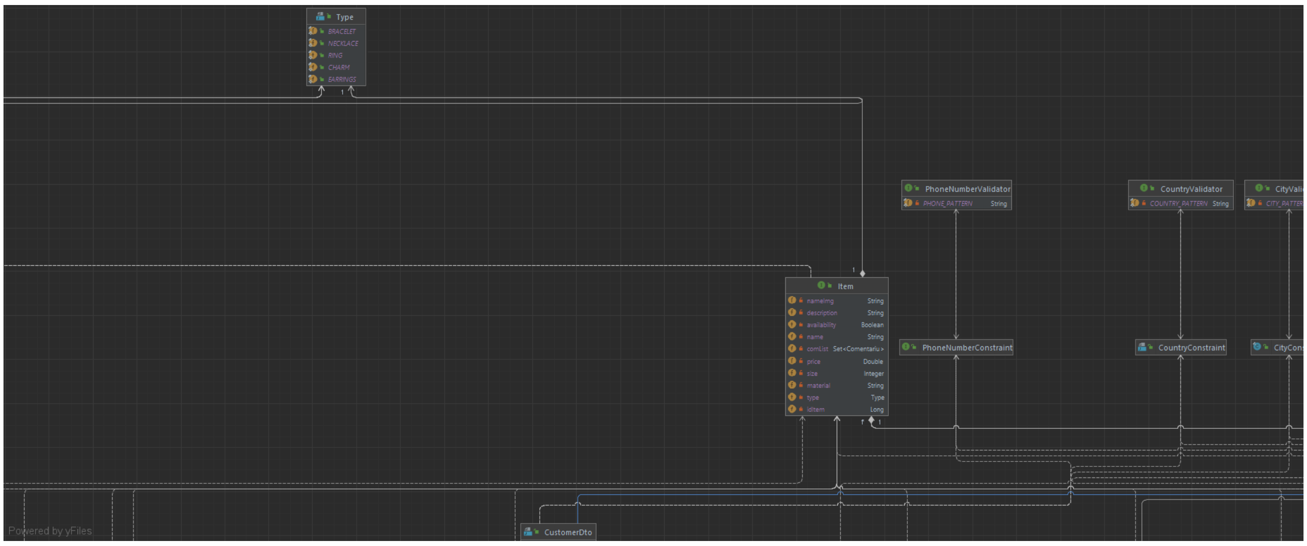
PasswordValidator
PASSWORD_PATTERN String
UserNameValidator
USERNAME_PATTERN String
NameValidator
NAME_PATTERN String
PhoneNumberValidator
PHONE_PATTERN String
CountryValidator
COUNTRY_PATTERN String
CityValidator
CITY_PATTERN String
StreetValidator
STREET_PATTERN String
PasswordConstraint
UserNameConstraint
NameConstraint
Item
nameImg String
description String
availability Boolean
name String
comList Set<Comentariu>
price Double
size Integer
material String
type Type
idItem Long
PhoneNumberConstraint
CountryConstraint
CityConstraint
StreetConstraint
UserDto
lastName String
id Long
password String
userName String
firstName String
email String
User
id Long
userName String
lastName String
password String
email String
firstName String
Customer
number Integer
city String
country String
phone String
street String
myFavoriteList Set<Item>
Powered by yFiles



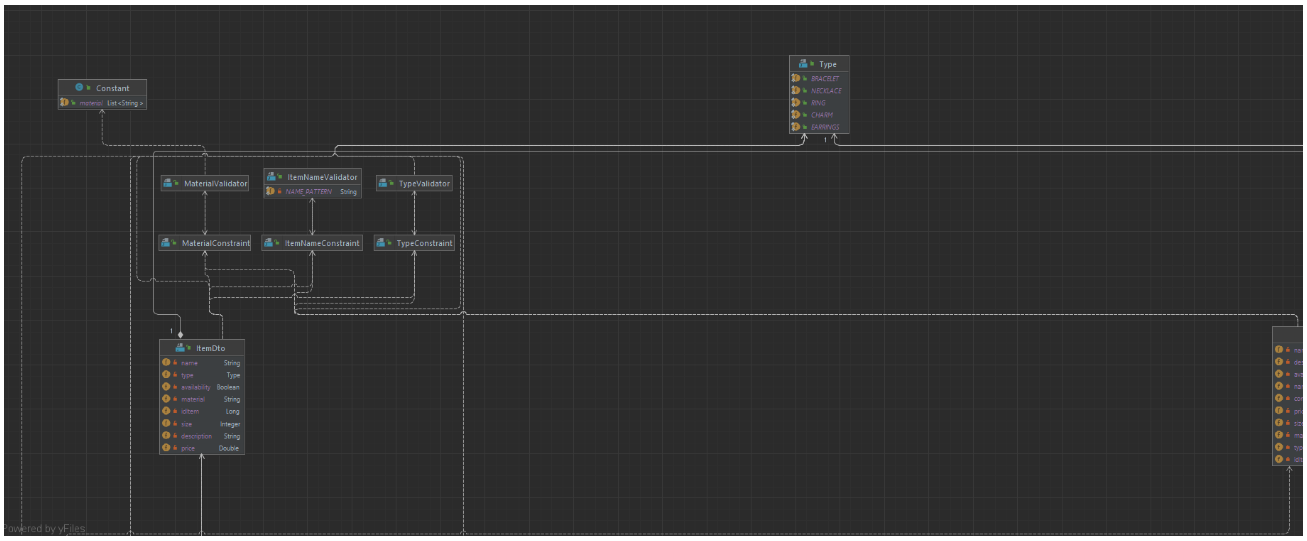
ExcelExporter
ControllerExceptionHandler
log Logger
CommandService
CustomerService
OrderItemRepository
ItemRepository
CommandRepository
«create»
CommandController
commandService CommandService
InitializationService
adminService AdminService
itemService ItemService
orderItemRepository OrderItemRepository
commandService CommandService
customerService CustomerService
CustomerController
customerService CustomerService
commandService CommandService
CommandServiceImpl
commandRepository CommandRepository
customerRepository CustomerRepository
itemRepository ItemRepository
orderItemRepository OrderItemRepository
adminServiceImpl AdminServiceImpl
SwaggerConfig
WebConfig
Powered by yFiles



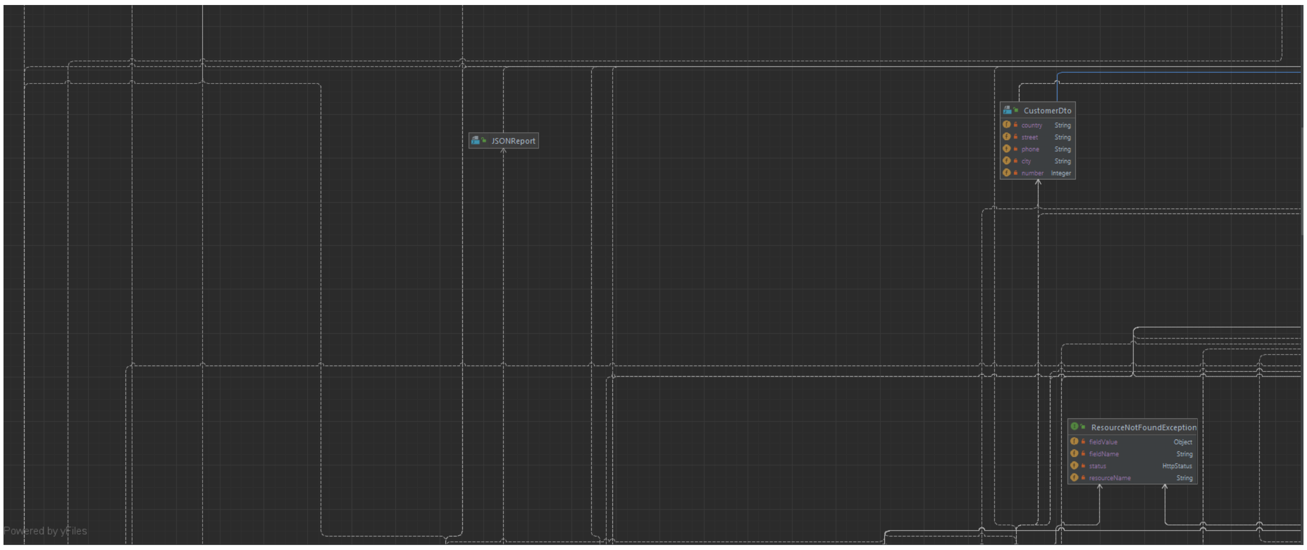
JSONReport
CustomerDto
country String
street String
phone String
city String
number Integer
OrderItemKey
itemId Long
orderId Long
Command
client Customer
date LocalDate
total Double
status String
id Long
ResourceNotFoundException
fieldValue Object
fieldName String
status HttpStatus
resourceName String
OrderItem
cantity int
item Item
id OrderItemKey
command Command
AdminService
ExcelExporter
ControllerExceptionHandler
log Logger
CommandService
CustomerService
OrderItemRepository
ItemRepository
CommandRepository

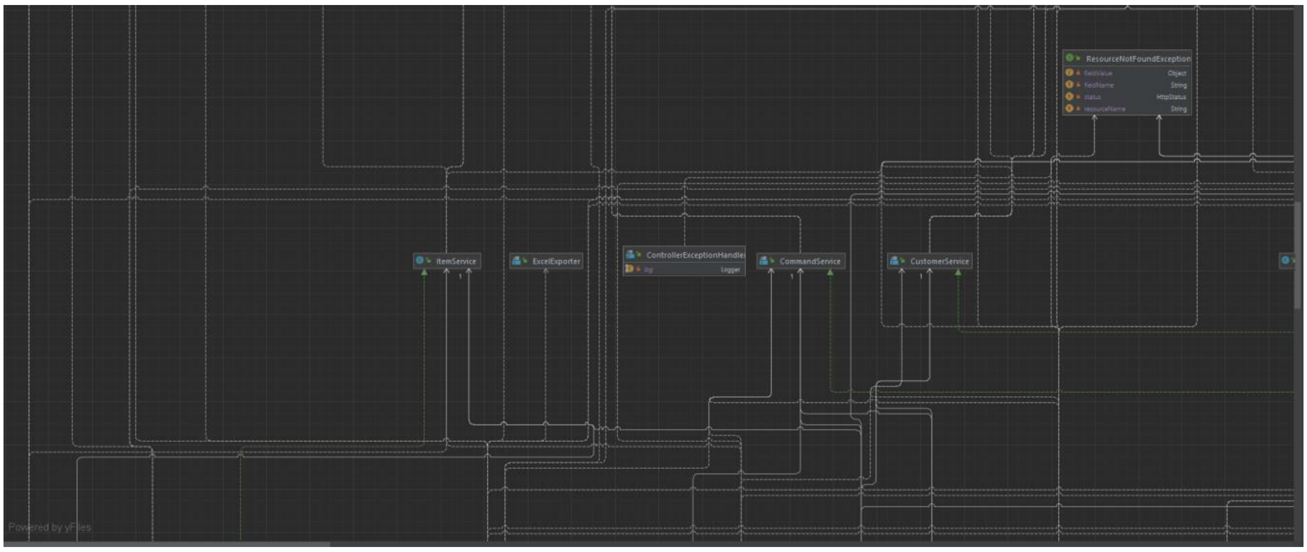


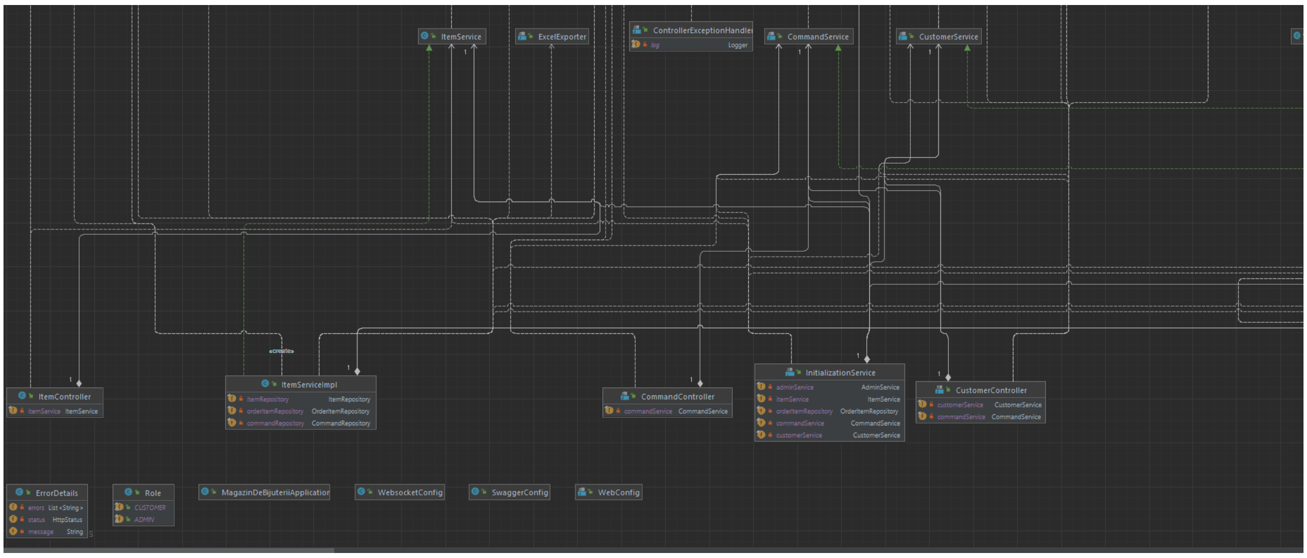
Type
BRACELET
NECKLACE
RING
CHARM
EARRINGS
PhoneNumberValidator
PHONE_PATTERN String
CountryValidator
COUNTRY_PATTERN String
Item
nameImg String
description String
availability Boolean
name String
comList Set<Comentariu>
price Double
size Integer
material String
type Type
idItem Long
PhoneNumberConstraint
CountryConstraint
CustomerDto
Powered by yFiles



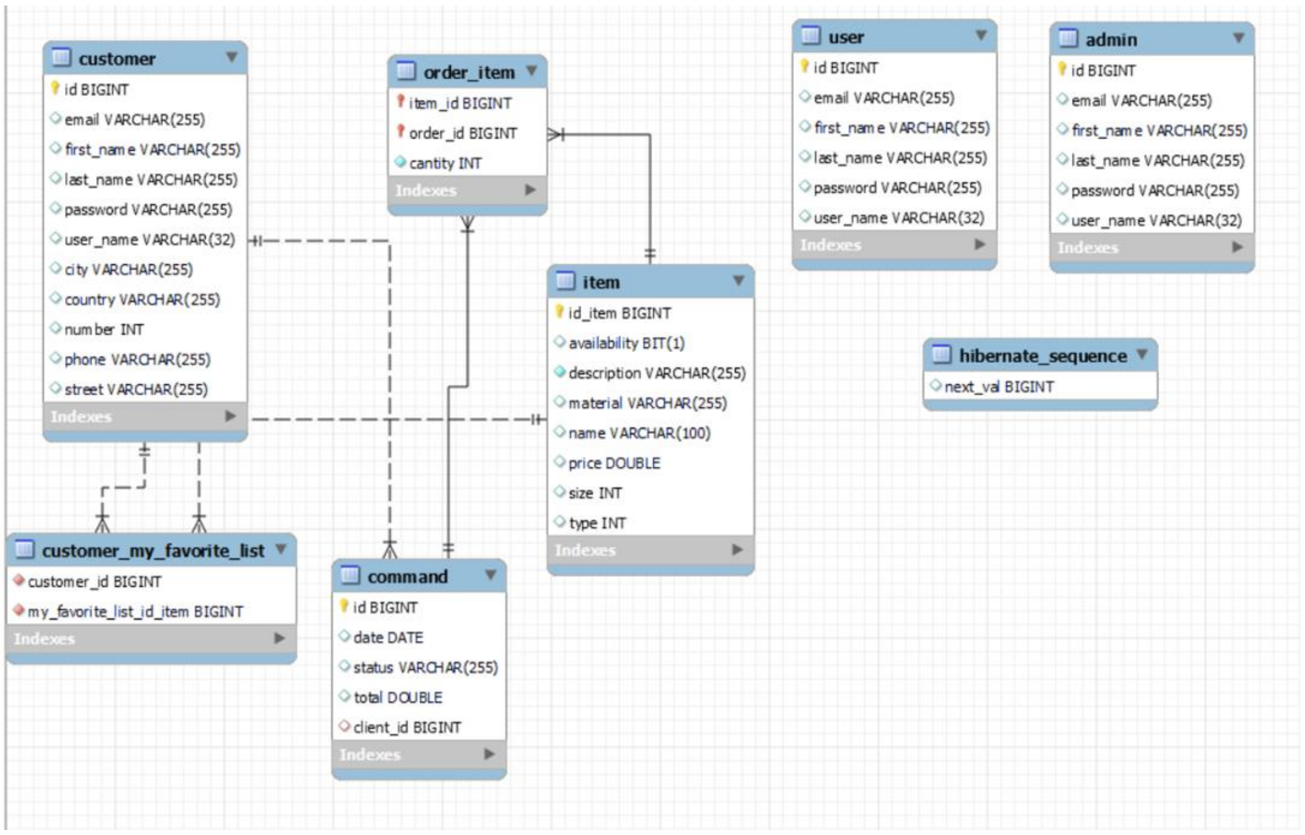
Constant
material List<String>
Type
BRACELET
NECKLACE
RING
CHARM
EARRINGS
MaterialValidator
ItemNameValidator
NAME_PATTERN String
TypeValidator
MaterialConstraint
ItemNameConstraint
TypeConstraint
ItemDto
name String
type Type
availability Boolean
material String
idItem Long
size Integer
description String
price Double
Powered by yFiles


JSONReport
CustomerDto
country String
street String
phone String
city String
number Integer
ResourceNotFoundException
fieldValue Object
fieldName String
status HttpStatus
resourceName String
Powered by yFiles


ResourceNotFoundException
fieldValue Object
fieldName String
status HttpStatus
resourceName String
ItemService
ExcelExporter
ControllerExceptionHandler
log Logger
CommandService
CustomerService
Powered by yFiles


ItemService
ExcelExporter
ControllerExceptionHandler
log Logger
CommandService
CustomerService
«create»
ItemController
itemService ItemService
ItemServiceImpl
itemRepository ItemRepository
orderItemRepository OrderItemRepository
commandRepository CommandRepository
CommandController
commandService CommandService
InitializationService
adminService AdminService
itemService ItemService
orderItemRepository OrderItemRepository
commandService CommandService
customerService CustomerService
CustomerController
customerService CustomerService
commandService CommandService
ErrorDetails
errors List<String>
status HttpStatus
message String
Role
CUSTOMER
ADMIN
MagazinDeBijuteriiApplication
WebsocketConfig
SwaggerConfig
WebConfig

Data Model



System Testing

In software development, testing each part of a program is crucial to assert that all individual parts are correct. A unit is the smallest testable part of the software ans is an important step in the development phase of an application, because:

- ✓ unit tests help to fix bugs early in the development cycle and save costs.
- ✓ understanding the code base is essential and unit tests are some of the best way of enabling developers to learn all they can about the application and make changes quickly.
- ✓ code is more reliable and reusable because in order to make unit testing possible, modular programming is the standard technique used.

In this project, I use JUnit as my testing framework for testing the CRUD API(Add, Delete, Update methods) for the Customer entity. Since I am doing unit tests, I need to isolate dependencies, so I use mocks for that. Mocks are objects that “fake”, or simulate, the real object behavior. This way I can control the behavior of my dependencies and test just the code I want to test.

For each operational endpoint, I ‘will need to test its controller and service by unitary approach, simulating its expected result and comparing with the actual result through a mock standpoint. If you notice in the code, for each operational endpoint I am using the assertThat and verify methods to test different things. By calling the verify method, I am checking that my repository was called and by calling assertThat I am checking that my service answered my call with the correct expected value.

Some notes about the annotations used :
@RunWith (MockitoJUnitRunner. Class) : Invokes the class MockitoJUnitRunner to run the tests instead of running in the standard built in class.
@Mock: Used to simulate the behavior of a real object, in this case, my repository
@InjectMocks: Creates an instance of the class and injects the mock created with the @Mock annotation into this instance
@Test: Tells JUnit that the method to which this annotation is attached can be run as a test case

Save Customer

To verify the method of adding a customer to the database, the first step was to create a Customer Object. Then I call the save method from the repository, which will return a customer. I check if the returned customer object is the same as the one I want to save. I did this check with the isSameAs statement, which I applied to all the fields of the Customer Object.

Delete Customer

As for the method of deleting the customer, this is done based on the id. The delete method will return the customer we want to delete. Using the findById function, I get the customer to be deleted. As in the previous example, I check if the deleted customer is the same as the customer who wanted to be deleted. What is different from the add method is that the customer with the id specified by us may not exist. In this case the repository should return a Nullable Object when the method is called.

Update Client

The update method is the same as the delete method.

Future Improvements

Payment

When the user places an order, he can choose whether to pay for the order with his card or on delivery. If he opts for the first option, he should be able to enter his bank card details. If he opts for the first option, he should be able to enter the details of the bank card, have a refund corresponding to the total order and be notified that the payment has been made successfully.
A solution for implementing this functionality could be to create a Bank Account class, in which to retain customers' bank accounts and cards. When the customer enters his card details we have to check if it exists that card, if so we have to check if he has enough money and then we have to extract the total order.

Discount

Another improvement could be the creation of promotions that would last for a limited period of time. The user will be notified by email when there is a discount and will first receive a code that he will need to apply before finishing the order to benefit from the discount.

Conclusion

To summarize the application is a complex system for managing orders, customers and products. The use of a relational database facilitates their management and outlines a structure that is easy to decipher. It has a beautiful interface that makes it easy to use.

Bibliography

https://www.baeldung.com/swagger-set-example-description

https://www.baeldung.com/spring-boot-bean-validation

https://www.youtube.com/watch?v=JUMbefx67wk