Contents

1 SchoolBook

Advanced Software Engineering Project

1.1 Abstract

Today, especially after Covid-19 pandemic, e-commerce are usually used to bought any kind of things. Everyday a new e-commerce site appears out on internet, nevertheless most of them fail due to lack of design using common platforms, that are designed for other use (i.e. Wordpress, Joomla, ect…).

This (didactic) project aims to define a software design for an e-commerce platform for books with specific requirements defined by stakeholders.

The software to be designed is for a book store that wishes to go online. It is to be developed to improve the efficiency for customer using multiple authentication factor in order to improve security.

/newpage

2 Requirement Software Analysis

2.1 Glossary Project

Terms	Definition
Catalog	A listing of all of the products that shop currently offers for sale.
Checkout	An electronic analogue of a real-world checkout in a supermarket. A place where customers can pay for the products in their shopping cart.
Credit card	A card such as VISA or Mastercard that can be used for paying for products.
Customer A party who buys products or services from Clear View Training Limited.	
Order	A document raised by a customer specifying one or more products that have been purchased. The order specifies the quantity of each product. Orders are passed to the Dispatch Department for processing.
Product	A book offered for sale.
Cart	An electronic analogue of a real-world shopping cart. A place where customers can store their items prior to purchase.
Browser	A program which allows users to browse the World Wide Web.
Captcha	A challange that requires to correctly evaluate a distorted image in order to determine that user is human.
OTP	A password that is valid for only one login session.

2.2 MoSCoW Terminology

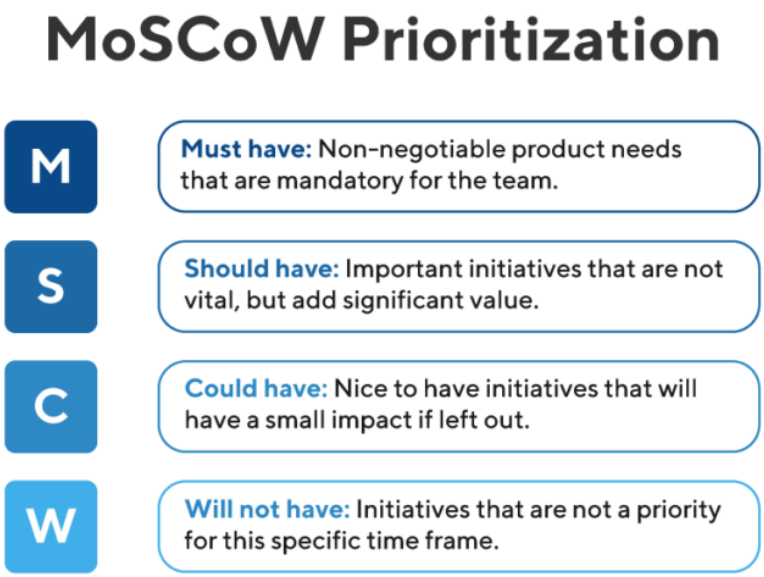


Figure 1: MoSCoW

2.3 Functional Requirements

The basic functional requirements are:

ID	Description	Module
RF_1	System **must** allow the user to register.	Registration/Login
RF_2	System **must** allow the user to login as customer.	Registration/Login
RF_3	System **shall** allow the administrator to delete customers.	Registration/Login
RF_4	System **must** shall collect customer information consisting of name, address, email address, phone number, credit card information.	Registration/Login
RF_5	System **shall** allow the customer to view and edit its customer information.	Registration/Login
RF_6	System **must** display a list of all books of store to users.	Book Details
RF_7	System **must** display detailed book description consisting of ISBN, name, author, publisher, cover image, summary, price to users.	Book Details
RF_8	System **shall** accept all major credit cards.	Payment
RF_9	System **must** validate payment with the credit card processing company.	Payment
RF_10	System **must** allow the customer to choose payment method: credit cards or cash on delivery.	User Interface

ID	Description	Module
RF_11	System **must** allow the customer to place items into cart.	User Interface
RF_12	System **must** allow the customer to remove items from cart.	User Interface
RF_13	System **must** allow the customer to checkcout cart.	User Interface
RF_14	System **must** allow the dispatch department to view all orders.	Orders
RF_15	System **shall** allow customers to view their order history.	Orders
RF_16	System **must** allow the administrator to add books to catalog	Stock Managament
RF_17	System **must** allow the administrator to delete books from catalog	Stock Management
RF_18	System **shall** track information about orders	Delivery&Tracking
RF_19	System **shall** display track information about customer order.	Delivery&Tracking
RF_20	System **must** allow the dispatch department to update state of order.	Delivery&Tracking
RF_21	System **could** save the cart of last customer session.	Registration/Login

2.4 Not-Functional Requirements

The basic not-function requirements are:

ID	Description	Type
NF_1	System **must** use a browser as its user interface.	ComplianceTo-Standards
NF_2	System **must** collect costumer information in according to GPDR	ComplianceTo-Standards
NF_3	System **must** store sales transaction data.	Availability
NF_4	System **shall** use a multifactor authentication.	Security

2.5 Use Case Model

2.5.1 Actors

The table below contains brief semantics for the actors

Actors	Description
Administrator	A special user of the system who can manage shop and other costumers.
Credit Card Issuer	An external company that processes credit card transactions.
Customer	Authenticated user who can buy books from shop.
Dispatcher	Subject or systen who can manage order and shippment.
User	Someone who uses the system but who is not a customer.

2.5.2 Use Case Diagram

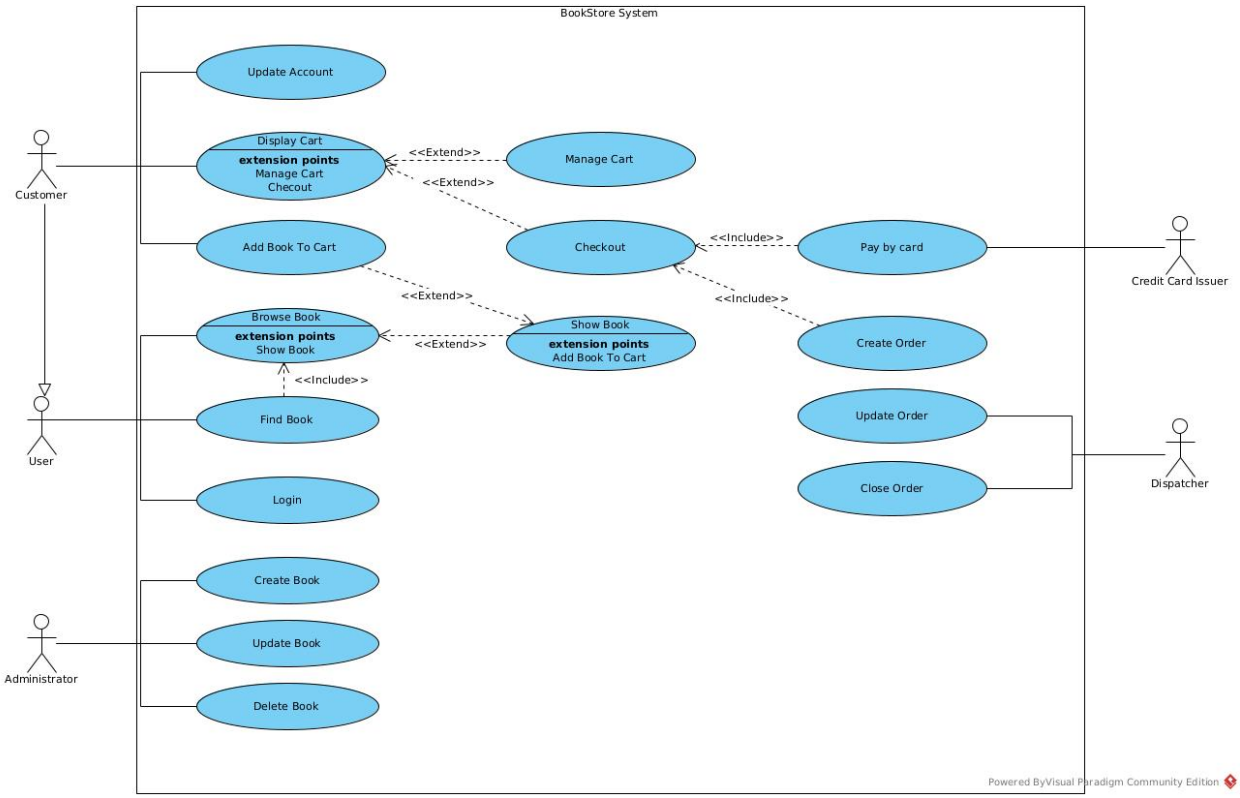


Figure 2: Use Case Model

2.5.2.1 Use Case: Login

ID: **UC1**

Actors:

- User

Preconditions:

- User is not logged on to the system.

Flow of events:

- 1 The scenario begins when the User selects “Login”.
- 2 While the User is not logged on:
  - 2.1 The System asks the User email and password.
  - 2.2 The User enters email and password.
  - 2.3 If email and password are correct:
    - 2.3.1 The System asks the User to solve a captcha
    - 2.3.2 The User enters captcha
    - 2.3.3 If captcha is correct
      - 2.3.3.1 The System sends OTP code to Customer’s phone number:
      - 2.3.3.2 The User enters OTP code
      - 2.3.3.3 It OTP code is correct:
      - 2.3.3.4 The System authenticates the User

Postconditions:

- User is a Customer

Secondary Scenario:

- 1 The scenario begins when the Customer selects “Log On”.
- 2 While the User is not logged on
  - 2.1 The System asks the User email and password.
  - 2.2 The User enters email and password.
  - 2.3 The email and/or password are wrong.
- 3 The System logs a security violation.

Postcondition:

- User is not logged in
- System log a security violation

2.5.2.2 Use Case: Register

ID: **UC2**

Actors:

- User

Preconditions:

- User is not logged on to the system.

Flow of events:

- 1 The use case begins when the User selects "Register".
- 2 The System asks the User to enter a email and password.
- 3 The User enters the requested information.
- 4 The System checks to see if the email is unique and the password is valid.
- 5 While the email is not unique or the password is invalid
  - 5.1 The System asks for a new email and/or password.
- 6 The System asks the Customer for the following information: name and address, phone number, credit card details (optionally).
- 7 The User enters the requested information.
- 8 While information is missing
  - 8.1 The System asks the User for the missing information.
  - 8.2 The User enters the missing information.
- 9 The System confirms that the User information has been accepted

Postcondition:

- 1 The System has saved the Costumer details.
- 2 The Customer is assigned a username and password.
- 3 The Customer is assigned a unique customer identifier.

2.5.2.3 Use Case: Update Account

ID: **UC3**

Actors:

- Costumer

Preconditions:

- None

Flow of events:

- 1 The use case begins when the Costumer selects “Modify Account”.
- 2 The system displays the customer details including name and address, email address, phone number, credit card details.
- 3 The Customer select a field and changes its value.
- 4 The Costumer selects “Save Changes”
- 5 While new informations are invalid
  - 5.1 System asks to insert correct informations
  - 5.2 The Costumer select incorrect informations and changes its value
  - 5.3 The Costumer selects “Save Changes”
- 6 The System confirms the changes

Postcondition:

- 1 The Customer’s details have been updated.

Secondary Scenario: - 1 The use case begins when the Costumer selects “Modify Account”. - 2 The system displays the customer details including name and address, email address, phone number, credit card details. - 3 The Customer select a field and changes its value. - 4 The Costumer exits.

Postcondition:

- 1 The Customer’s details have not changes.

2.5.2.4 Use Case: Delete Account

ID: **UC4**

Actors:

- Administrator

Preconditions:

- None.

Flow of events:

- 1 The use case begins when the Adminstrator selects “Delete Account”.
- 2 The System asks for a email
- 3 The Adminstrator enters the email
- 4 The System displays Customer details related to email
- 5 The Adminstrator confirms the deletion

Postcondition:

- 1 The Customer’s account has been deleted.

2.5.2.5 Use Case: Add Item To Cart

ID: **UC5**

Actors:

- Customer

Preconditions:

- None

Flow of events:

- 1 The Customer selects a product
- 2 The Customer selects “Add Item”.
- 3 The system adds the item to the Customer’s shopping cart.

Postcondition:

- 1 A product has been added to the Customer’s cart.

2.5.2.6 Use Case: Display Cart

ID: **UC6**

Actors:

- Customer

Preconditions:

- None

Flow of events:

- 1 The Customer selects “Display Cart”.
- 2 If there are no items in the cart
  - 2.1 The system tells the Customer that the cart is empty.
  - 2.2 The use case terminates
- 3 For each book in the cart
  - 3.1 The System displays the book detail, quantity, unit price and total price.
- extension point: manageCart
- extension point: checkout

Postcondition:

- None

2.5.2.7 Use Case: Manage Cart

ID: **UC7**

Actors:

- Customer

Preconditions:

- 1 The System is displaying the shopping cart.

Flow of events:

- 1 While the Customer is updating the cart
  - 1.1 The Customer selects an item in the cart.
  - 1.2 If the Customer selects “Remove Item”
    - 1.2.1 The System removes the selected item from the cart.
  - 1.3 If the Customer enters a new quantity for the selected item
    - 1.3.1 The System updates the quantity for the selected item.

Postcondition:

- None

2.5.2.8 Use Case: Checkout

ID: **UC8**

Actors:

- Customer

Preconditions:

- None

Flow of events:

- 1 The use case begins when the Customer selects "Checkout".
- 2 The System presents the final order to the Customer. The order includes an order line for each book that shows the product name, the quantity, the unit price, the total price for that quantity. The order also includes the shipping address of the Customer and the total cost of the order including postage and packing costs.
- 3 The System asks the Customer to accept or decline the order
- 4 The Customer accepts the order.
- 5 The System asks to choose payment method
- 6 If Customer select online payment
  - 6.1 include(AcceptPaymentByCard)
- 7 If Costumer select cash on delivery
  - 7.1 The System add service's cost
  - 7.3 The order status has been set to pending.
  - 7.2 The order has been sent to the Dispatcher.

Postcondition:

- 1 The Customer has accepted the order

2.5.2.9 Use Case: Payment By Card

ID: **UC9**

Actors:

- Customer
- Credit Card Issuer

Preconditions:

- None

Flow of events:

- 1 The use case begins when the Customer accepts the order.
- 2 The System retrieves the Customer's credit card details.
- 3 The System sends a message to the Credit Card Issuer that includes: merchant identifier, merchant authentication, name on card, number of card, expiry date of card, amount of transaction.
- 4 The Credit Card Issuer authorises the transaction.
- 5 The System notifies the Customer that the card transaction has been accepted.
- 6 The System gives the Customer an order reference number for tracking the order.
- 7 The System sends the order to the Dispatcher.
- 8 The System changes the order's state to pending.
- 9 The System displays an order confirmation that the Customer may print out.

Postcondition:

- 1 The order status has been set to pending.
- 2 The Customer's credit card has been debited by the appropriate amount.
- 3 The order has been sent to the Dispatcher.

Secondary Scenario:

- 1 The secondary scenario begins after step 3 of the primary scenario
- 2 The Credit Card Issue doesn't allow transaction
- 3 The system displays a message telling the Customer that their order can't be processed.
- 4 Order is deleted

2.5.2.10 Use Case: Browse Books

ID: **UC10**

Actors:

- Customer

Preconditions:

- 1 A set of books has been identified for browsing

Flow of events:

- 1 The system displays a page containing a maximum of 10 books. This page includes the following summary information for each book: title, author, publisher, price.
- 2 While the Customer is browsing
- 2.1 If there are more products to display
  - 2.1.1 The Customer may select "Next" to view the next page of products.
- 2.2 If the Customer is not on the first page of products
  - 2.2.1 The Customer may select "Previous" to view the previous page of products.

extension point: showbook

Postcondition:

None

2.5.2.11 Use Case: Show Book

ID: **UC11**

Actors:

- User

Preconditions:

- None

Flow of events:

- 1 The use case begins when the Customer select a book.
- 2 The System displays book's information: ISBN, name, author, publisher, price and stock avaibility.

extension point: buybook

Postcondition:

None

2.5.2.12 Use Case: Find Books

ID: **UC12**

Actors:

- User

Preconditions:

- None

Flow of events:

- 1 The Customer selects “Find Book”.
- 2 The System asks the Customer for book search criteria that consist of one or more of the following: title, author, ISBN, category.
- 3 The Customer enters the search criteria.
- 4 The System searches for books that match the Customer’s criteria.
- 5 If the system finds some books
  - 5.1 include(Browse Books).
- 6 Else
  - 6.1 The System tells the Customer that no matching books were found.

Postcondition:

None

2.5.2.13 Use Case: Add Book

ID: **UC13**

Actors:

- Administrator

Preconditions:

- None

Flow of events:

- 1 The use case begins when the Administrator selects “Add Book.
- 2 The System asks the Shopkeeper to enter the following product information: ISBN,title, category, authors, publisher, price, description, image.
- 3 The Administrator enters the requested information..
- 4 The System adds the new book to the catalog.

Postcondition:

- 1 A new book has been added to the catalog.

2.5.2.14 Use Case: Delete Book

ID: **UC14**

Actors:

- Administrator

Preconditions:

- None

Flow of events:

- 1 The use case begins when the Administrator selects “Delete Product”.
- 2 The System asks the Shopkeeper for the book identifier (ISBN) of the book to delete..
- 3 The Administrator enters the requested information..
- 4 The System displays the book details.
- 5 The Administrator confirms the deletion.
- 6 The System deletes the book from the catalog.

Postcondition:

- 1 A book has been deleted to the catalog.

2.6 Class Diagram

This is a first-cut use case diagram

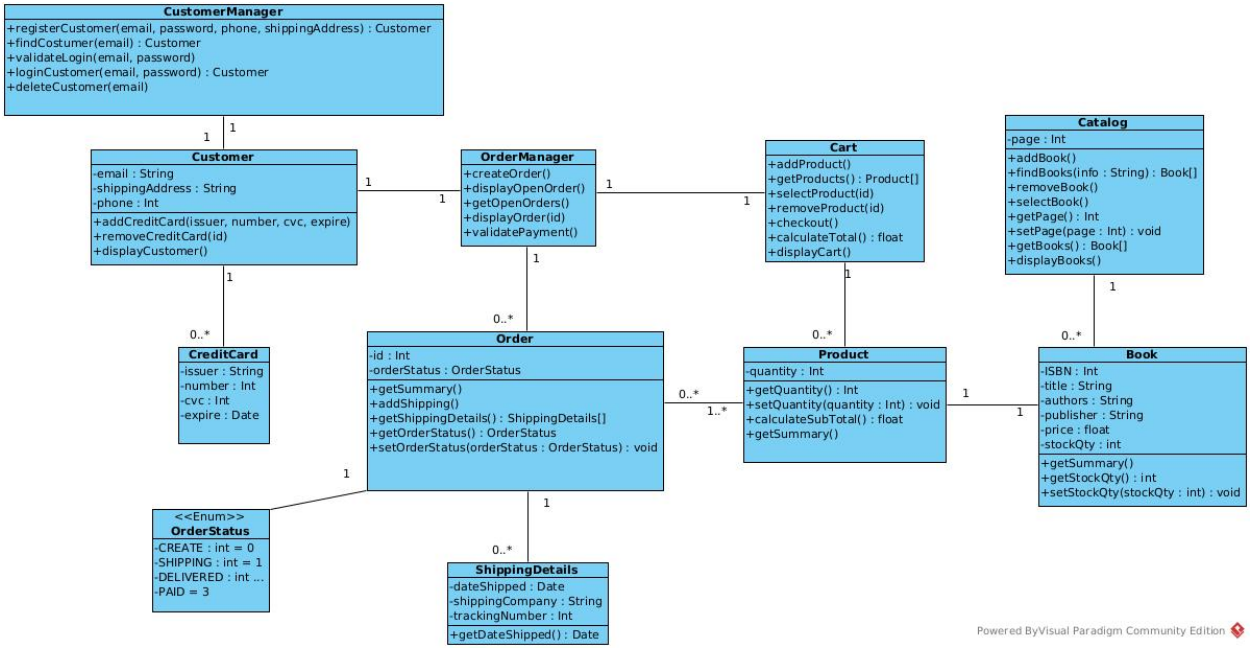


Figure 3: Class Diagram Analysis

2.7 Use Case Realization

The process of use case realization involves demonstrating how the analysis classes that you have identified interact together to realize the behavior specified by the use cases. Use case realizations consist of: - Detailed analysis class diagrams - Sequence and communication diagrams for the use cases.

2.7.1 Sequence diagrams

2.7.1.1 UC1 - Login

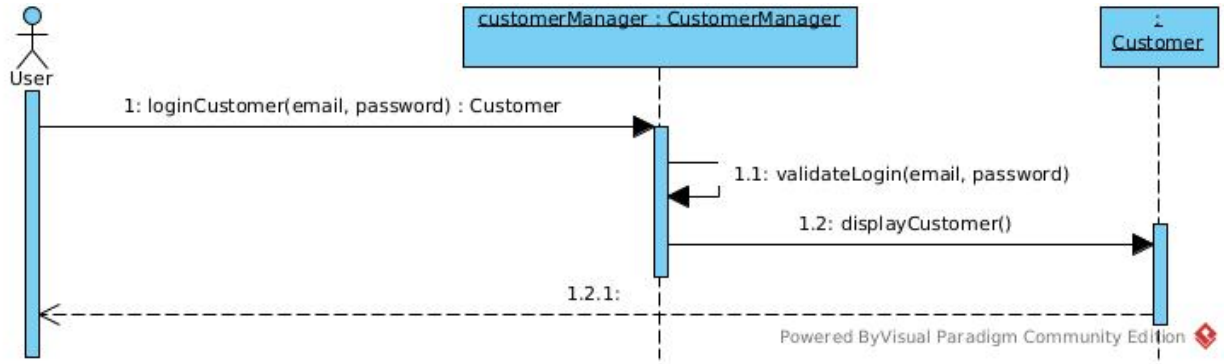


Figure 4: Sequence Diagram Analysis

2.7.1.2 UC2 - Register

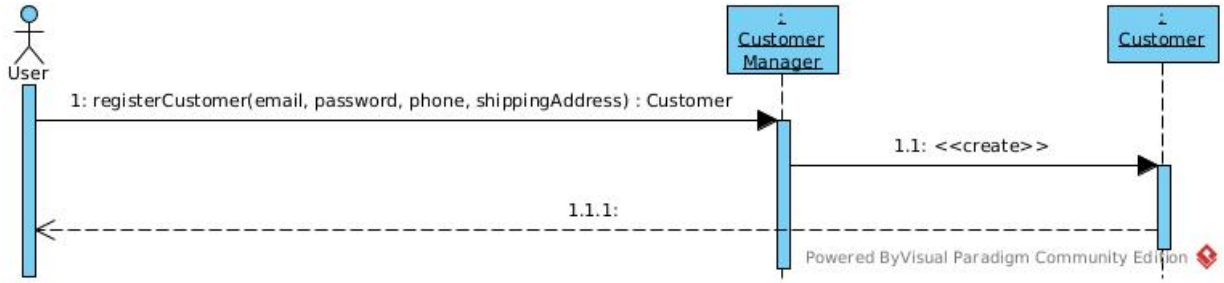


Figure 5: Sequence Diagram Analysis

2.7.1.3 UC3 - UpdateAccount

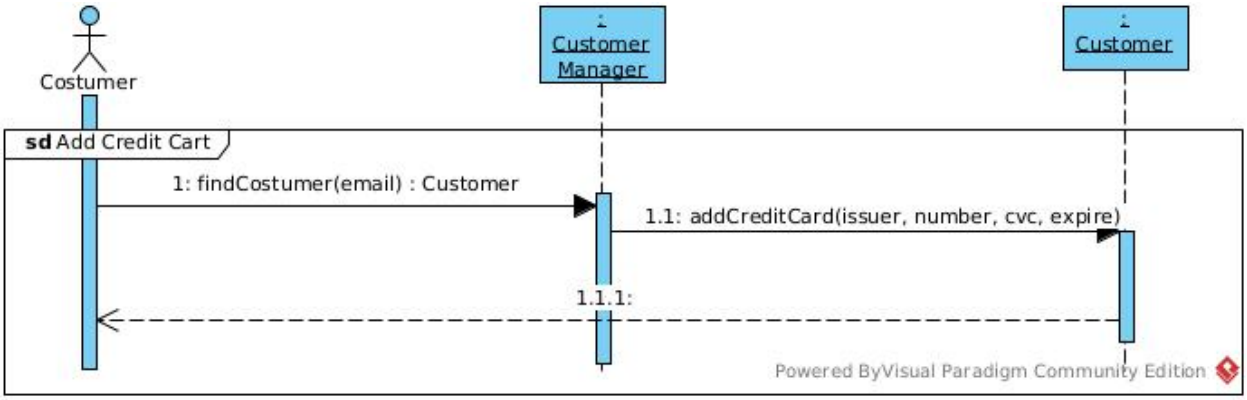


Figure 6: Sequence Diagram Analysis

2.7.1.4 UC4 - DeleteAccount

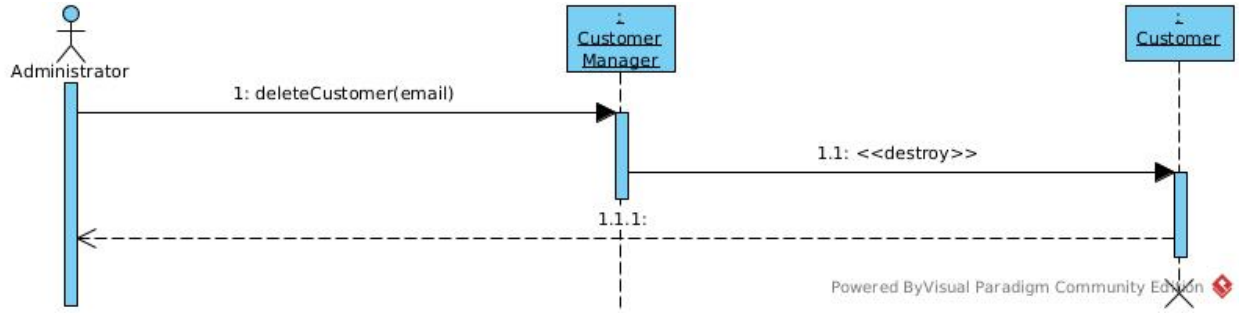


Figure 7: Sequence Diagram Analysis

2.7.1.5 UC5 - AddItemToCart

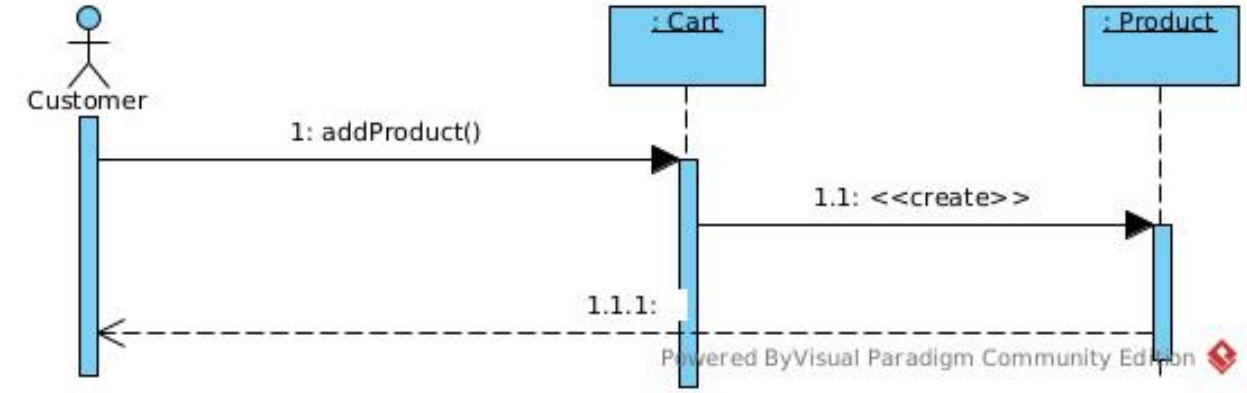


Figure 8: Sequence Diagram Analysis

2.7.1.6 UC6 - DisplayCart

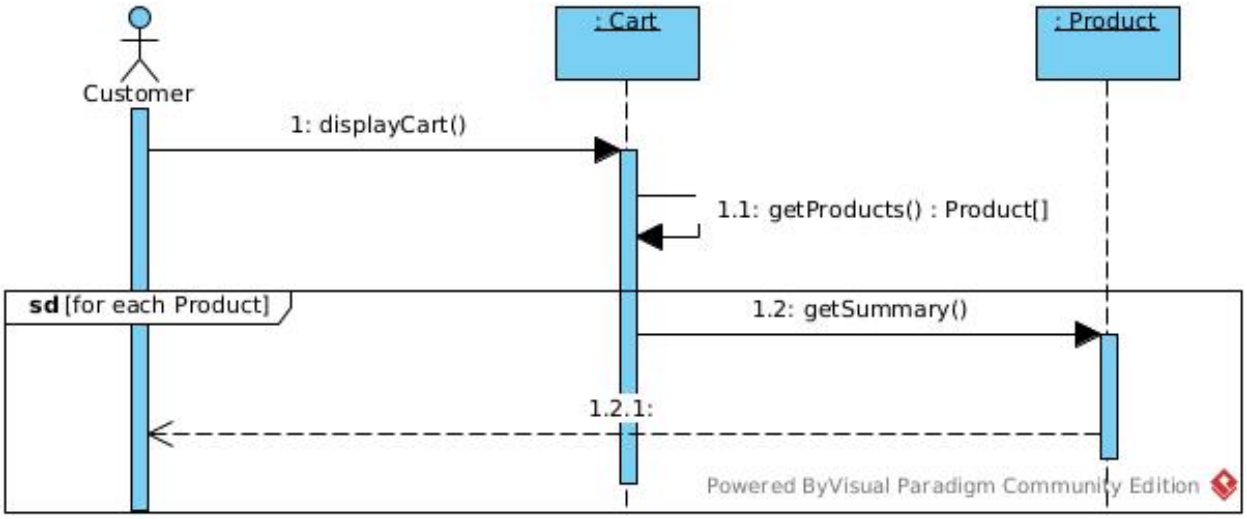


Figure 9: Sequence Diagram Analysis

2.7.1.7 UC7 - ManageCart

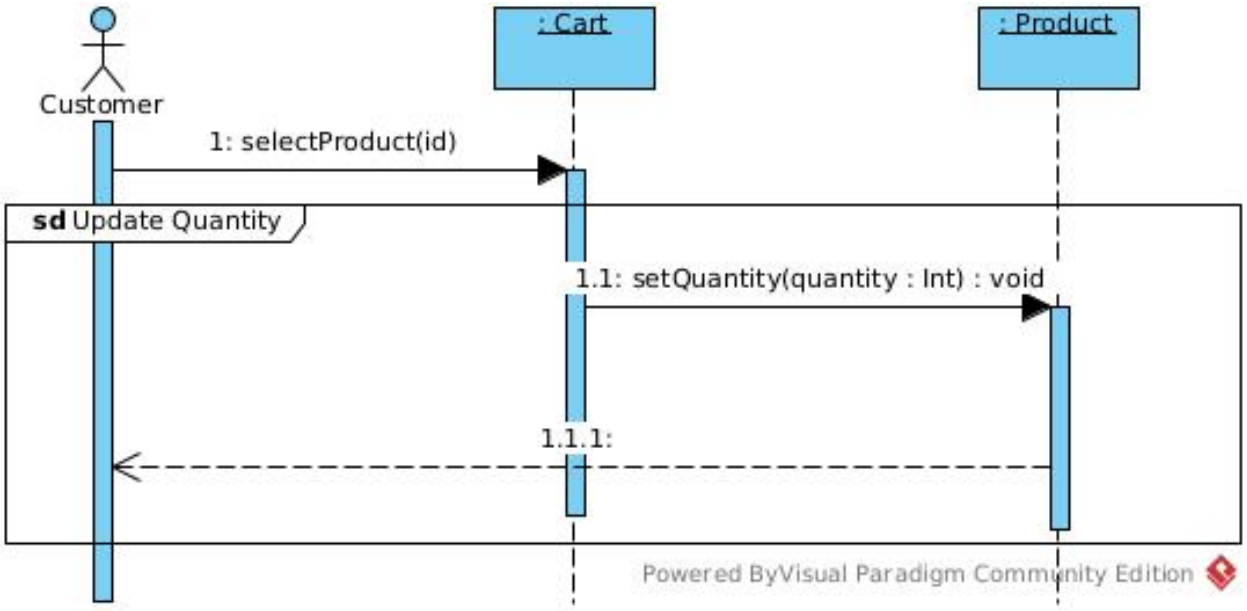


Figure 10: Sequence Diagram Analysis

2.7.1.8 UC8 - Checkout

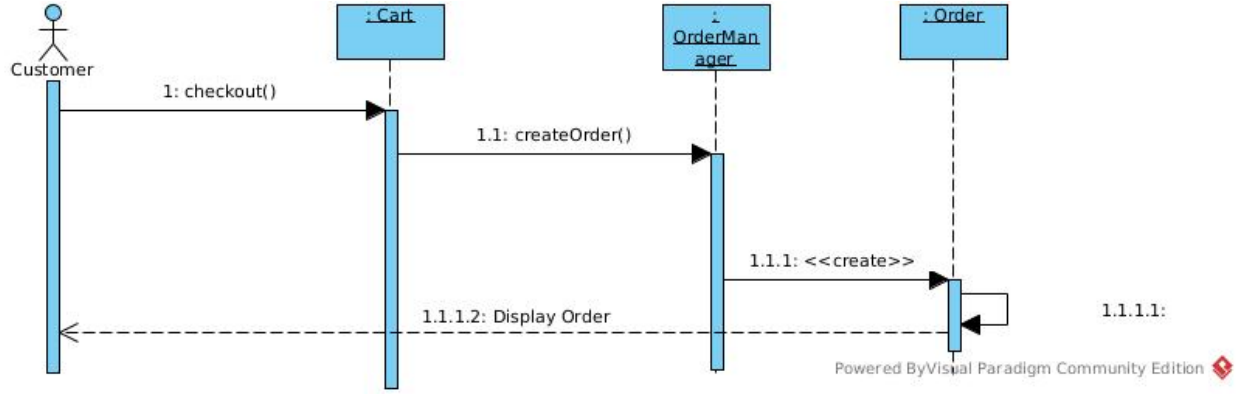


Figure 11: Sequence Diagram Analysis

2.7.1.9 UC9 - PaymentByCard

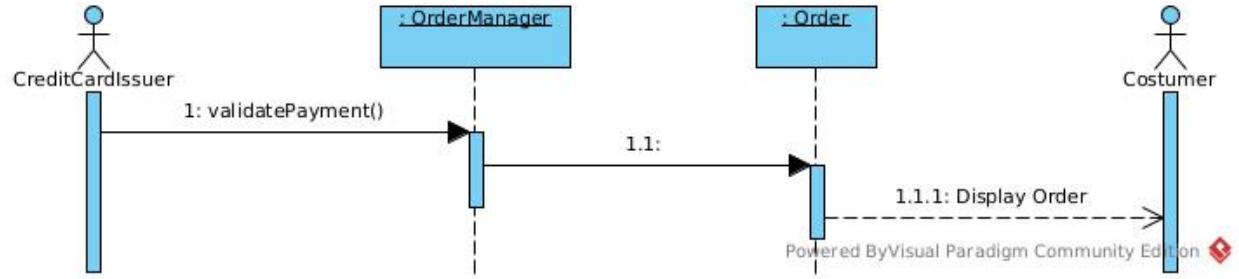


Figure 12: Sequence Diagram Analysis

2.7.1.10 UC10 - BrowseBooks

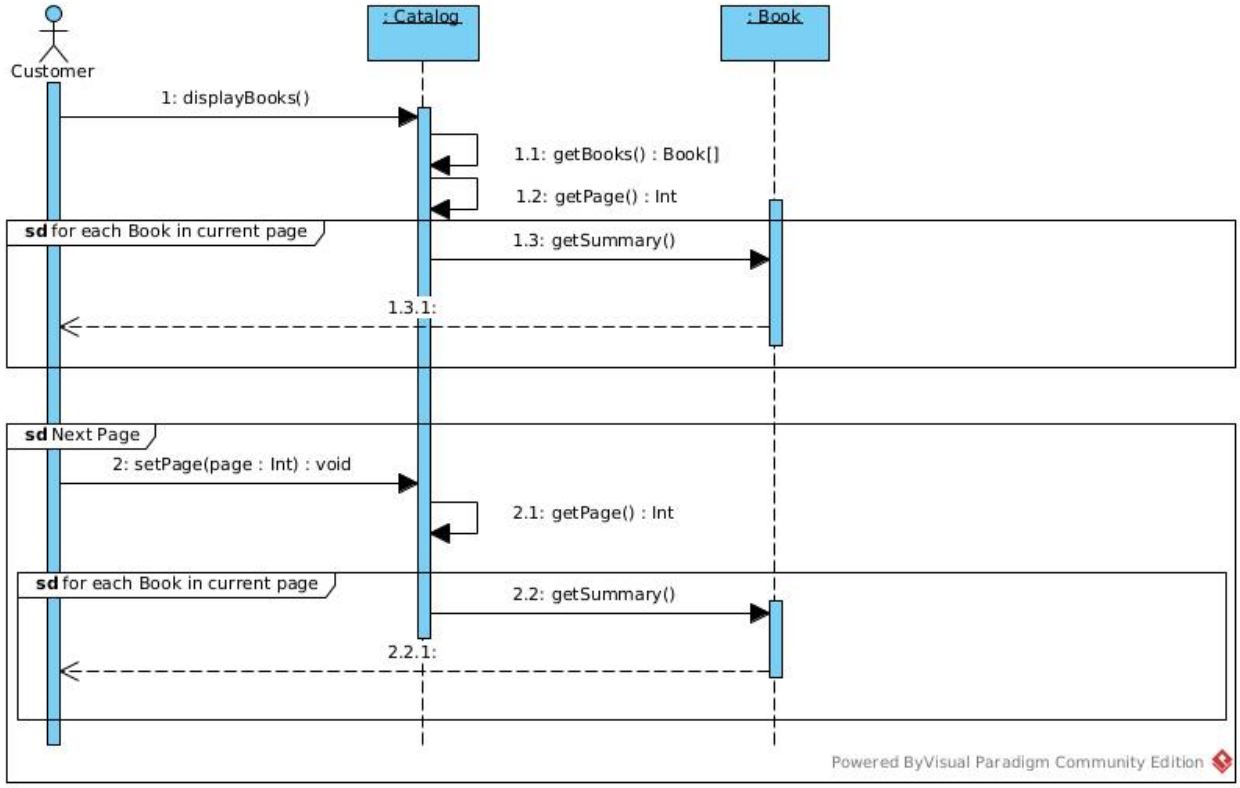


Figure 13: Sequence Diagram Analysis

2.7.1.11 UC10 - BrowseBooks

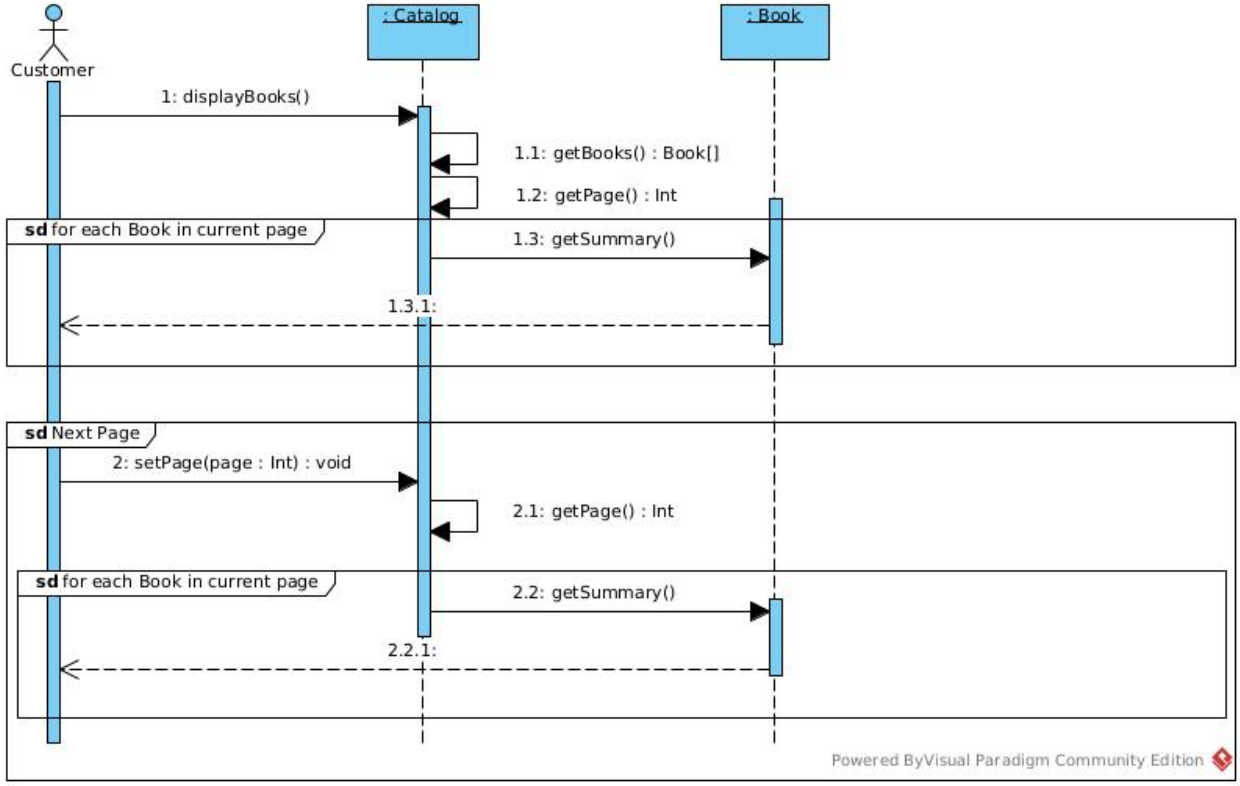


Figure 14: Sequence Diagram Analysis

2.7.1.12 UC11 - ShowBook

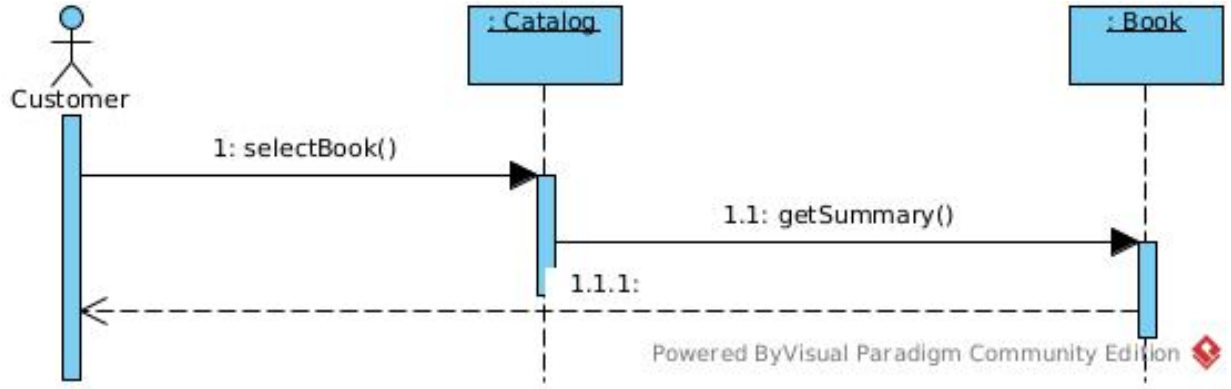


Figure 15: Sequence Diagram Analysis

2.7.1.13 UC12 - FindBooks

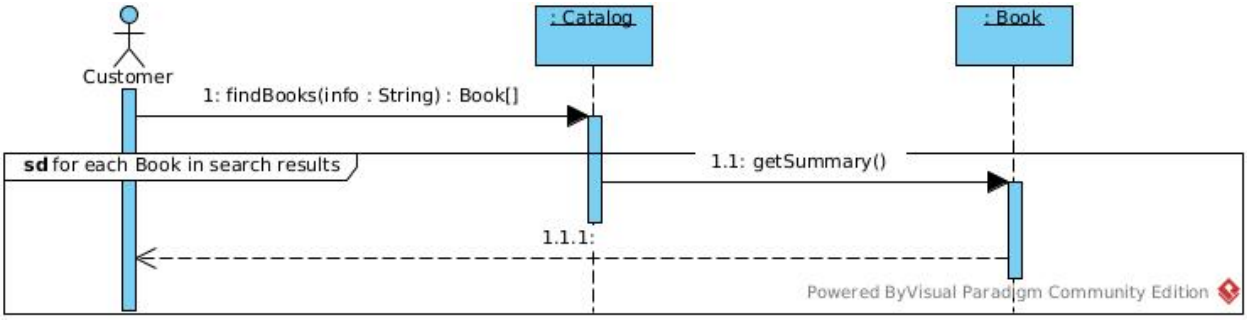


Figure 16: Sequence Diagram Analysis

2.7.1.14 UC13 - AddBook

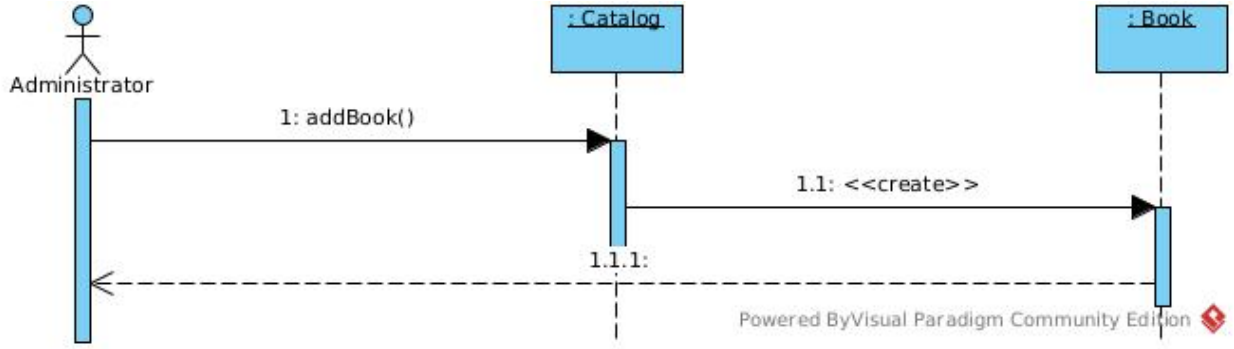


Figure 17: Sequence Diagram Analysis

2.7.1.15 UC14 - DeleteBook

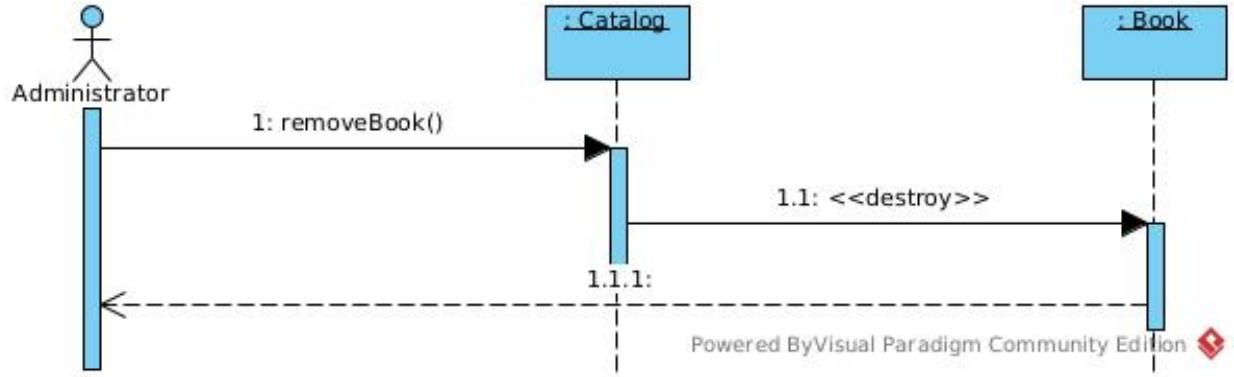


Figure 18: Sequence Diagram Analysis

3 Design

3.1 Class Diagram

In the design phase architect decided to use :

- MVC Pattern as architectural pattern to design design class
- Singleton, Memento and Chain of Responsability pattern in order to satisfy software requirements.

Below there is entire class diagram that will be discussed in next sections.

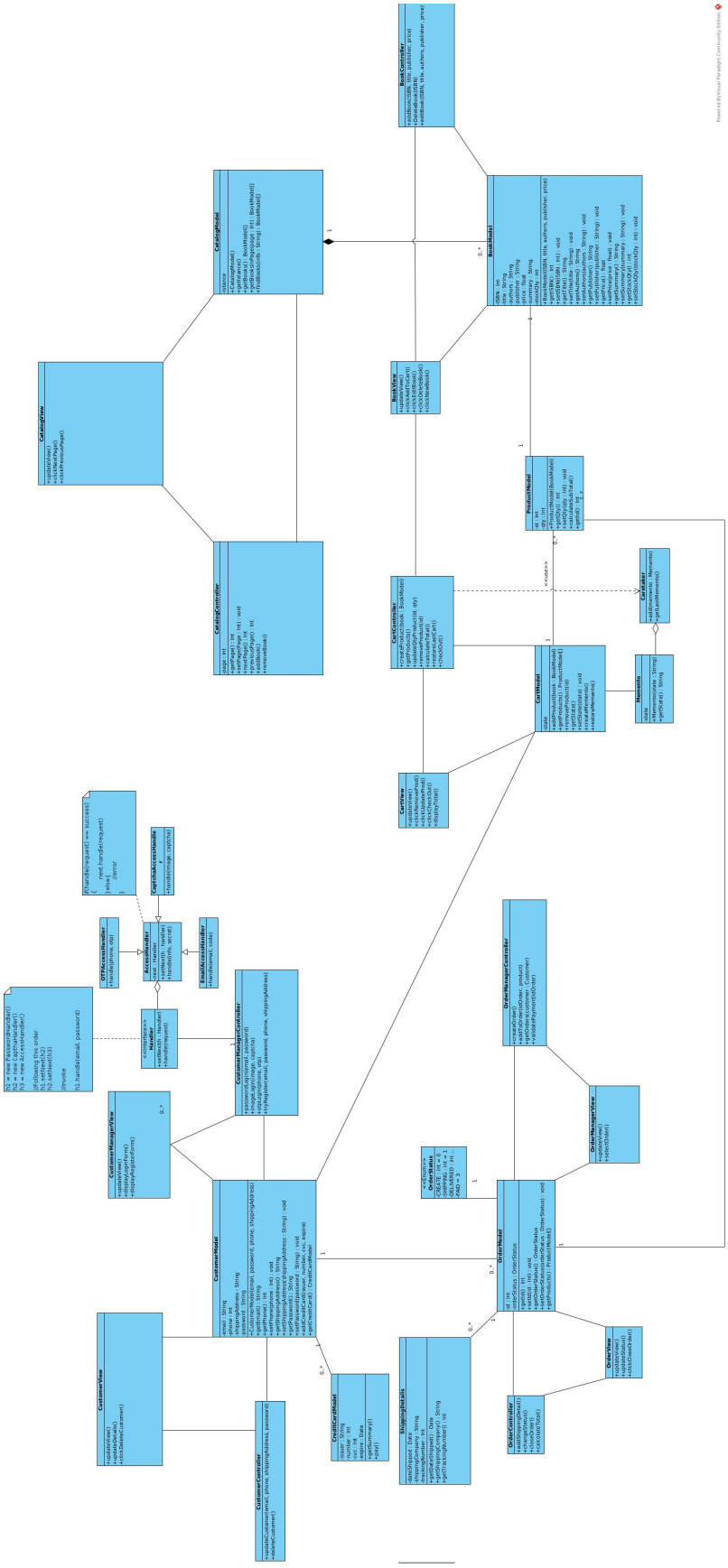


Figure 19: Class Diagram Design

3.2 Patterns

3.2.1 MVC Pattern

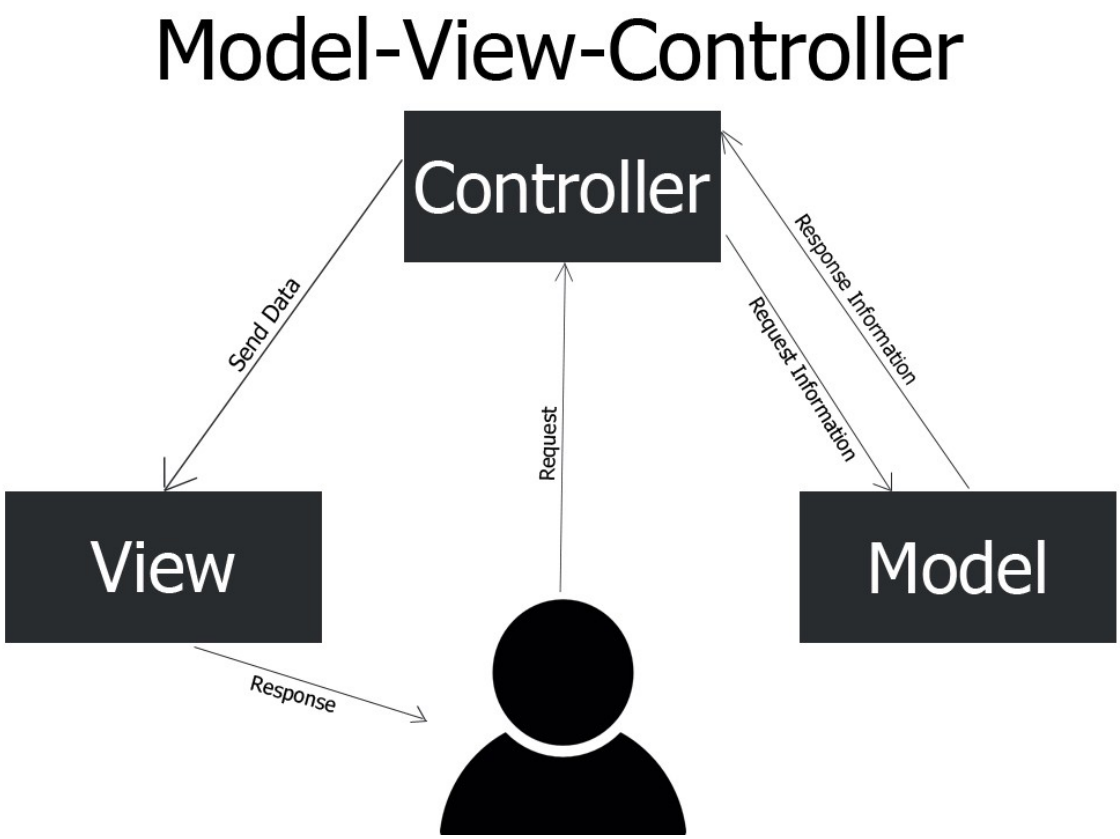


Figure 20: MVC

Model–view–controller (MVC) is a software design pattern commonly used for developing user interfaces that divide the related program logic into three interconnected elements. This is done to separate internal representations of information from the ways information is presented to and accepted from the user.

3.2.1.1 Model

The model represents the data and logic of the app. A model can have a to-one and to-many relationships to other models. This is where the data is manipulated and/or saved.

3.2.1.2 View

The view is the only part of the app the user interacts with directly. It is attached to the model and in turn displays the model's data. It may also update the model by sending appropriate messages to it as long as it matches up with the terminology in the model.

3.2.1.3 Controller

The controller is the go-between for models and views. It provides users with forms and menus for inputing information. The controller receives users input and translates it, and then passes those inputs on to one or more of the views. It interprets all user actions and goes between model and view to connect them.

3.2.2 Singleton

Singleton is a creational design pattern that lets you ensure that a class has only one instance, while providing a global access point to this instance.

The Singleton class declares the static method getInstance that returns the same instance of its own class.

The Singleton's constructor should be hidden from the client code. Calling the getInstance method should be the only way of getting the Singleton object.

In this project shop catalog is oblivious a object with a single instance :

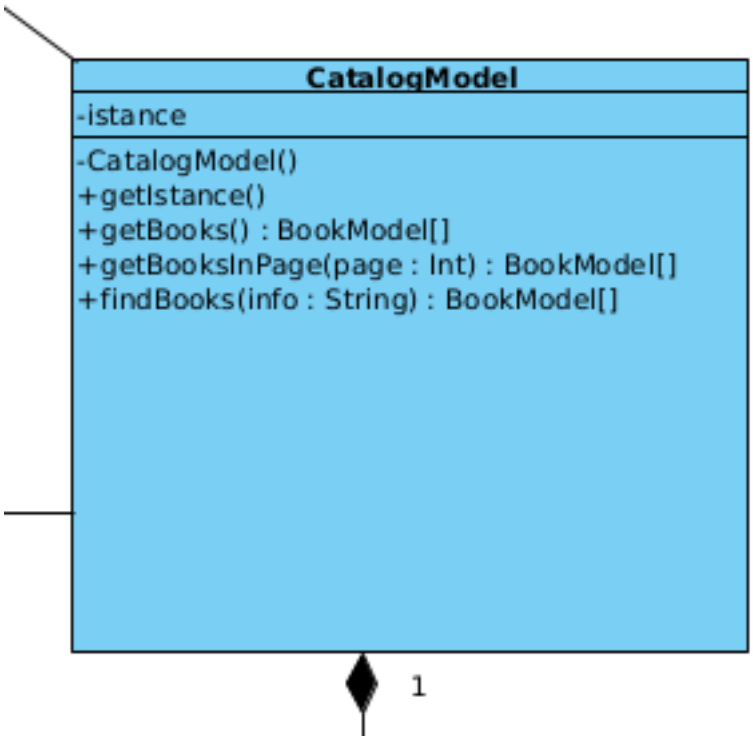


Figure 21: MVC

3.2.3 Memento

Memento is a behavioral design pattern that lets you save and restore the previous state of an object without revealing the details of its implementation.

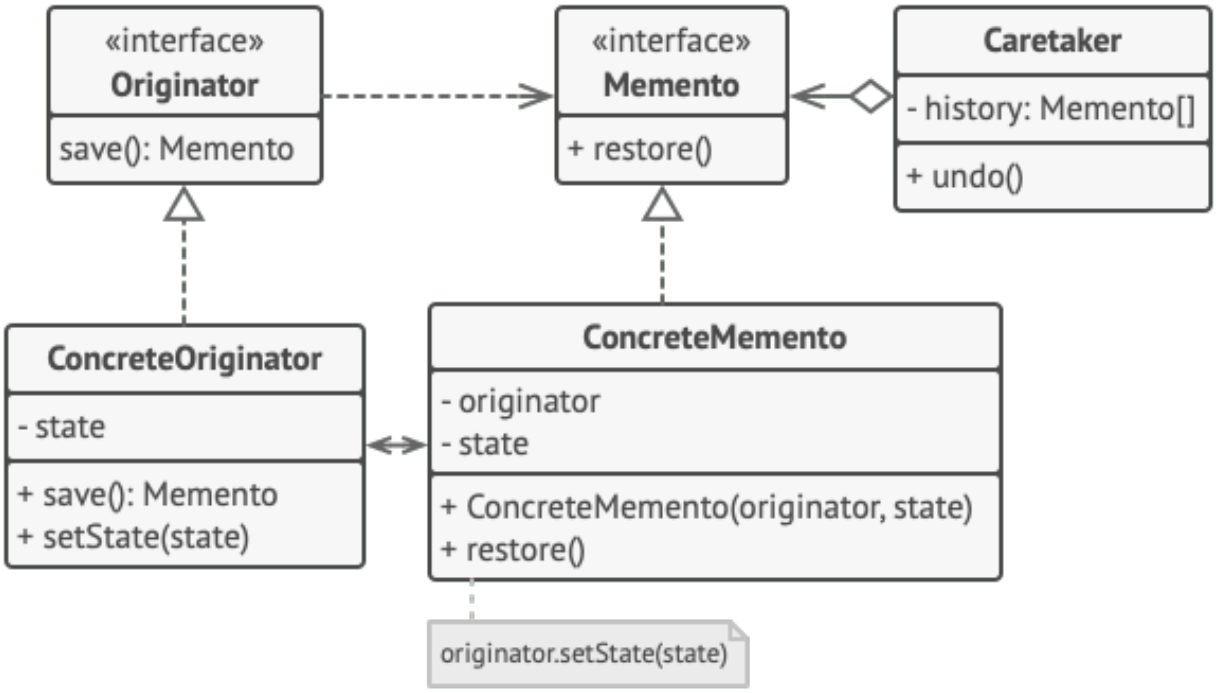


Figure 22: Memento

1. The Originator class can produce snapshots of its own state, as well as restore its state from snapshots when needed.
2. The Memento is a value object that acts as a snapshot of the originator's state.
3. The Caretaker knows not only "when" and "why" to capture the originator's state, but also when the state should be restored.
4. A caretaker can keep track of the originator's history by storing a stack of mementos. When the originator has to travel back in history, the caretaker fetches the topmost memento from the stack and passes it to the originator's restoration method.

In this project, Function requirememente RF_21 aims to save and restore cart of last customer session, momento pattern looks useful for this requirement.

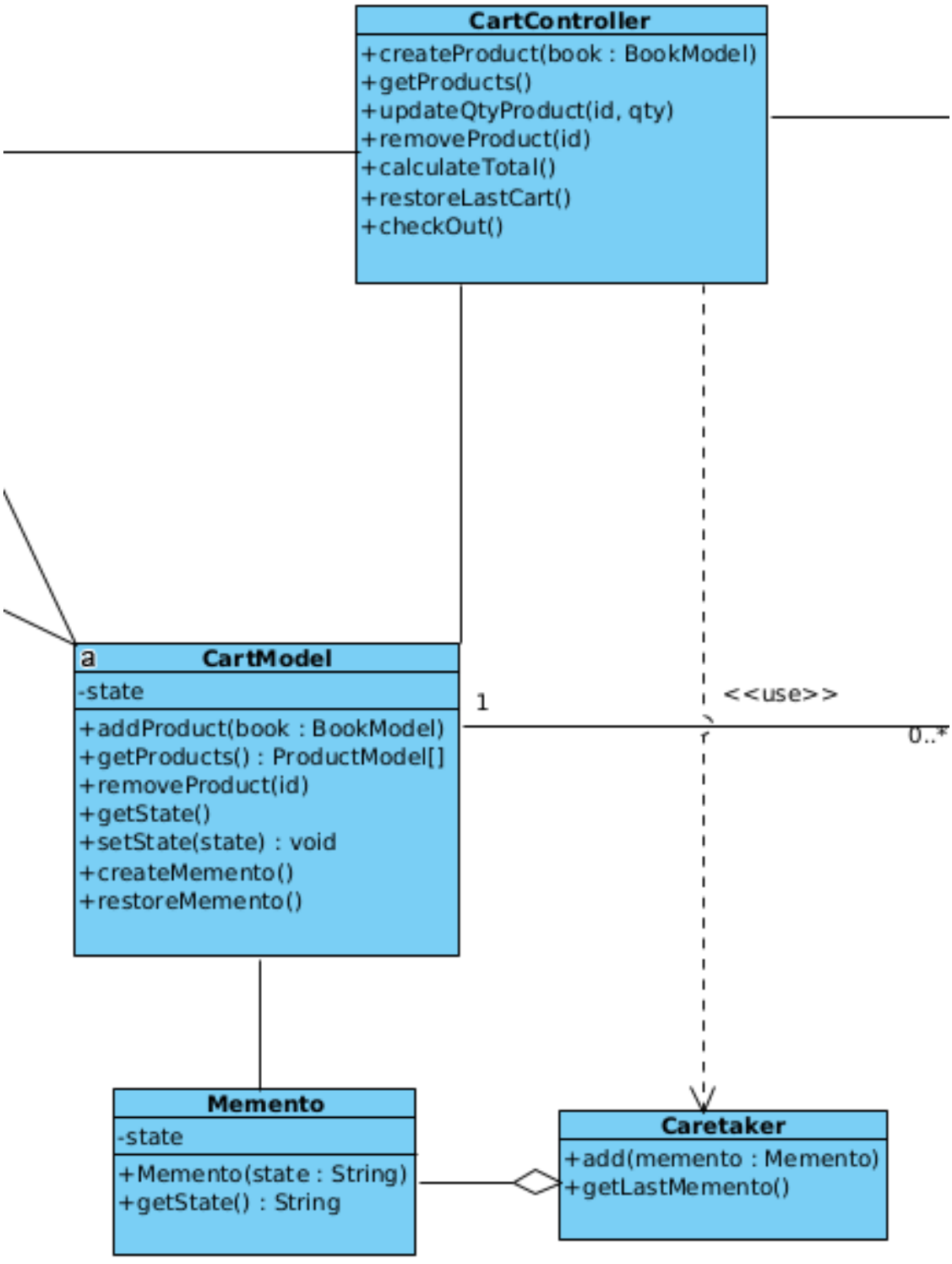


Figure 23: Memento Custom

3.2.4 Chain Of Responsability

Chain of Responsibility is a behavioral design pattern that lets you pass requests along a chain of handlers. Upon receiving a request, each handler decides either to process the request or to pass it to the next handler in the chain.

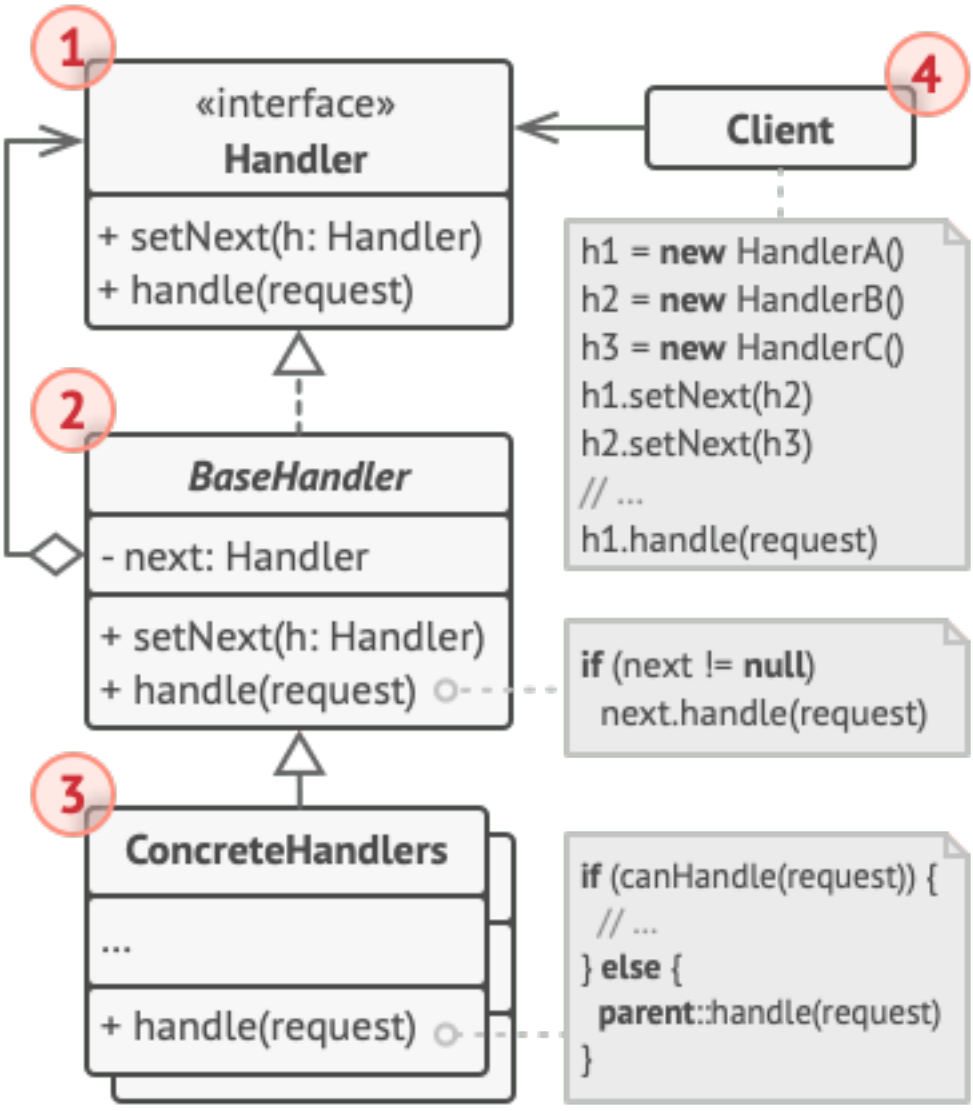


Figure 24: Chain of Responsability

- The Handler declares the interface, common for all concrete handlers. It usually contains just a single method for handling requests, but sometimes it may also have another method for setting the next handler on the chain.
- The Base Handler is an optional class where you can put the boilerplate code that's common to all handler classes. Usually, this class defines a field for storing a reference to the next handler. The clients can build a chain by passing a handler to the constructor or setter of the previous handler. The class may also implement the default handling behavior: it can pass execution to the next handler after checking for its existence.
- Concrete Handlers contain the actual code for processing requests. Upon receiving a request, each

handler must decide whether to process it and, additionally, whether to pass it along the chain.

In this project, not functional requirement NF_4 asks to use a multifactor authentication for access control, using this pattern a sequence of authentication factor can be implemented easily.

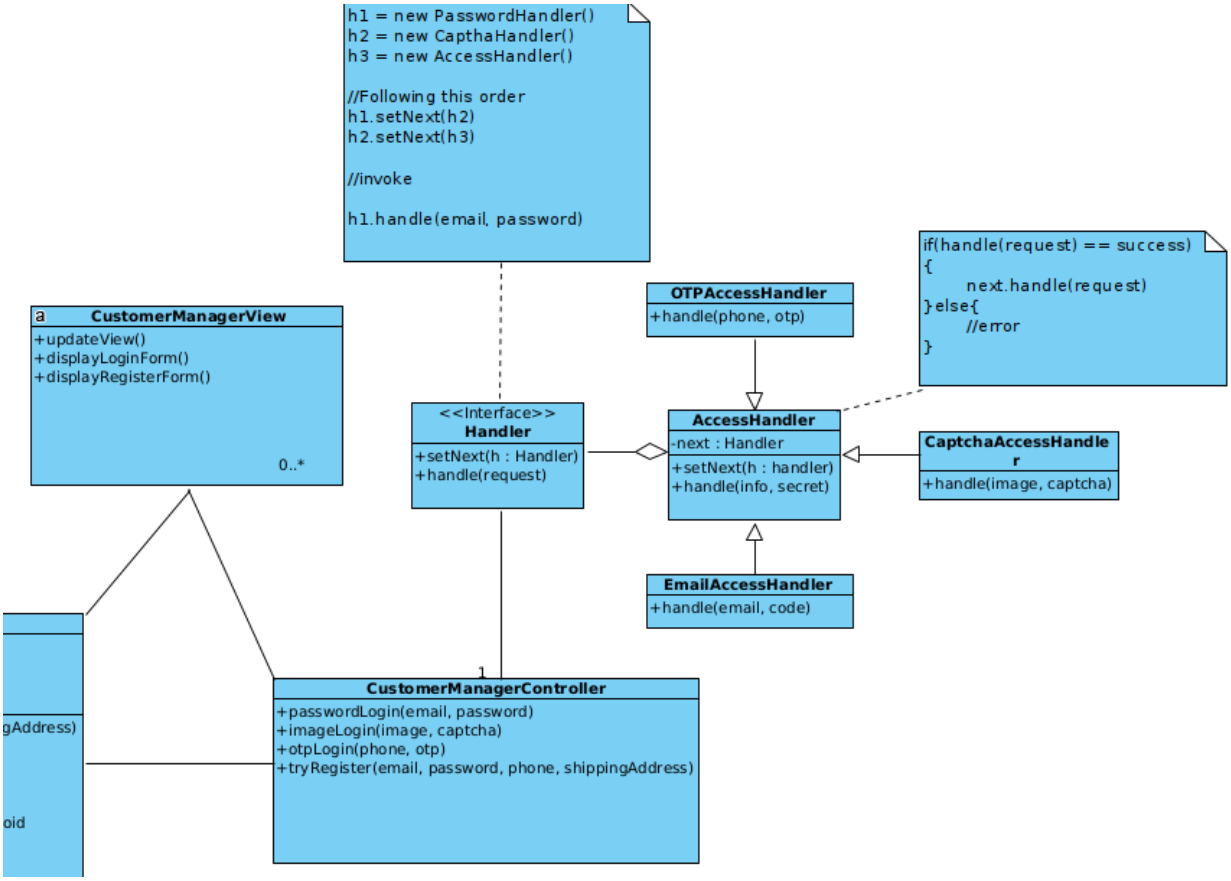


Figure 25: Chain of Responsability

3.3 Sequence Diagram

In order to validate class diagram, some meaningful sequence diagram are showed up

3.3.1 UC1 - Login

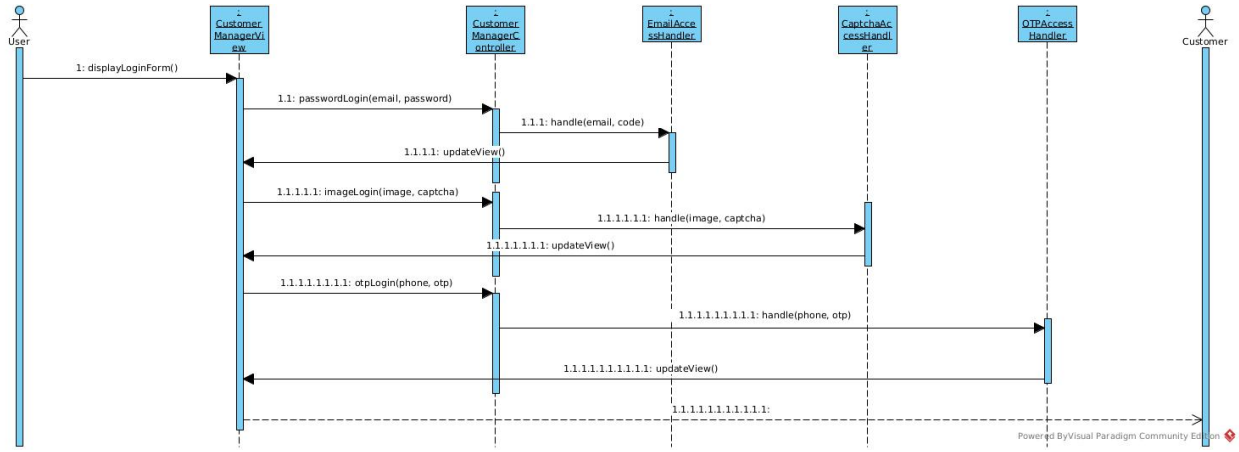


Figure 26: UC1

3.3.2 UC5 - AddItemToCart

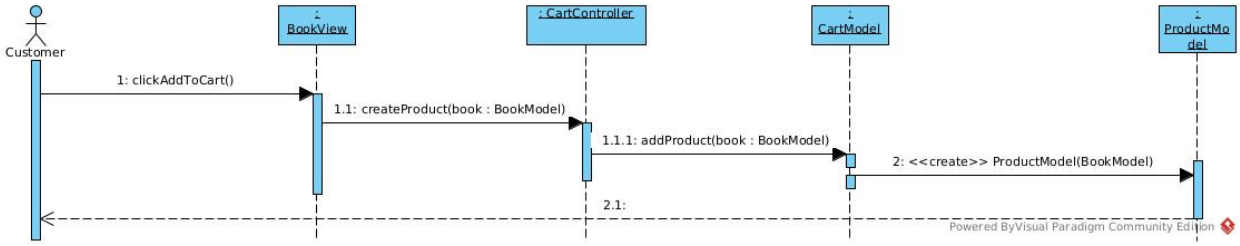


Figure 27: UC5

3.3.3 UC6 - DisplayCart

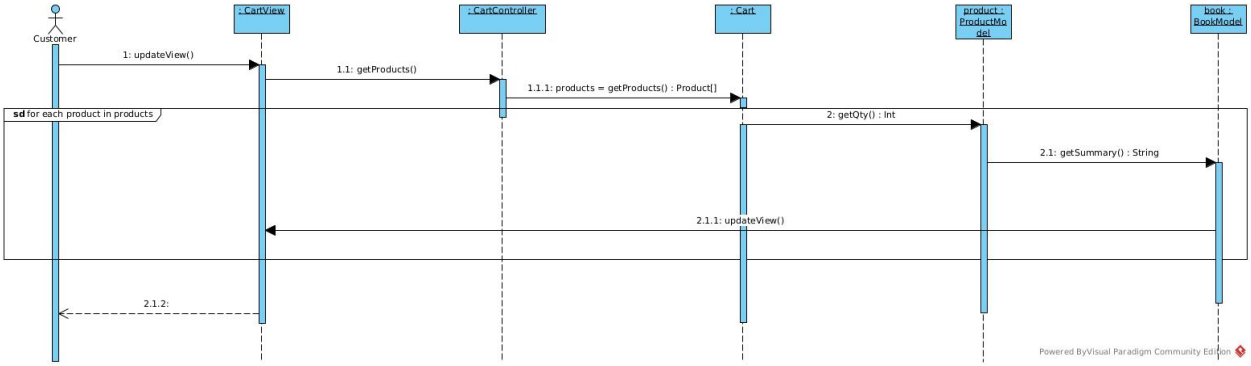


Figure 28: UC6

3.3.4 UC8 - Checkout

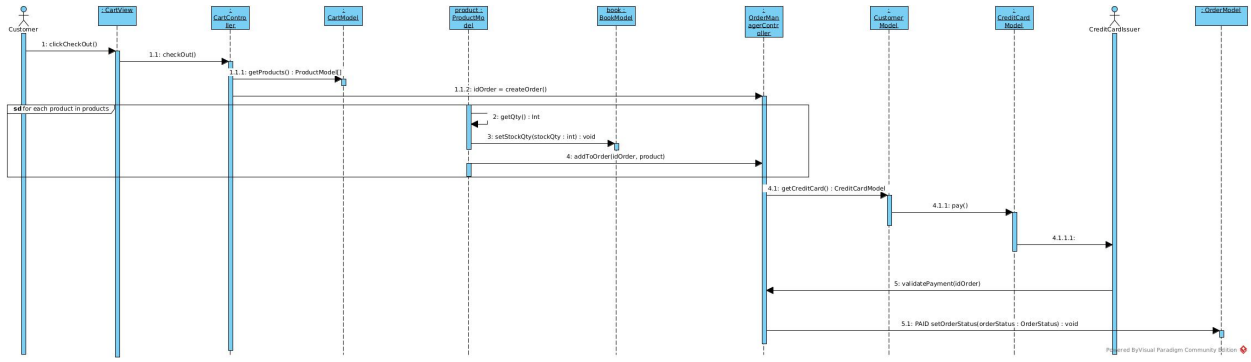


Figure 29: UC8

3.3.5 UC10 - ShowBook

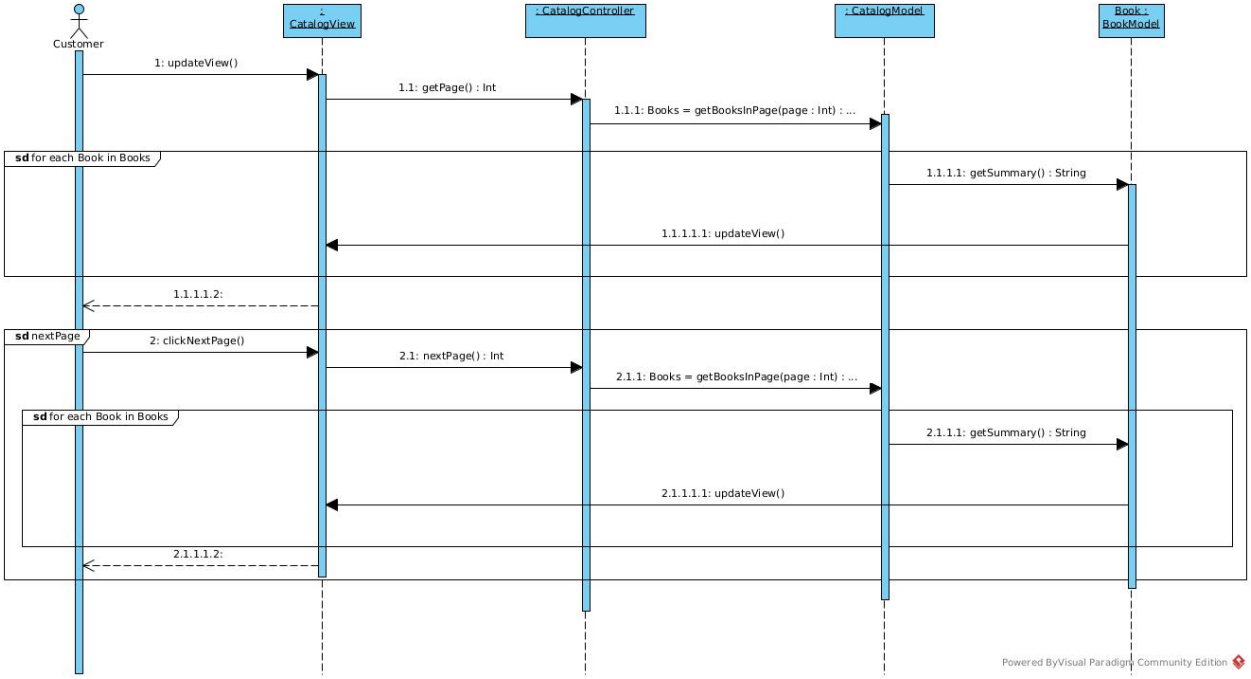


Figure 30: UC10

3.4 Component Diagram

In according to MVC architectural pattern, component diagram shows a logical layered architecture :

- View Package : React Framework Components used as Presentation Layer
- Controller Package : Nodejs script files used as Business Layer
- Model Package : Nodejs script files used as Database Layer

The outer interfaces are used by actor to interface System, instead inner inferfaces declare MVC relationship.

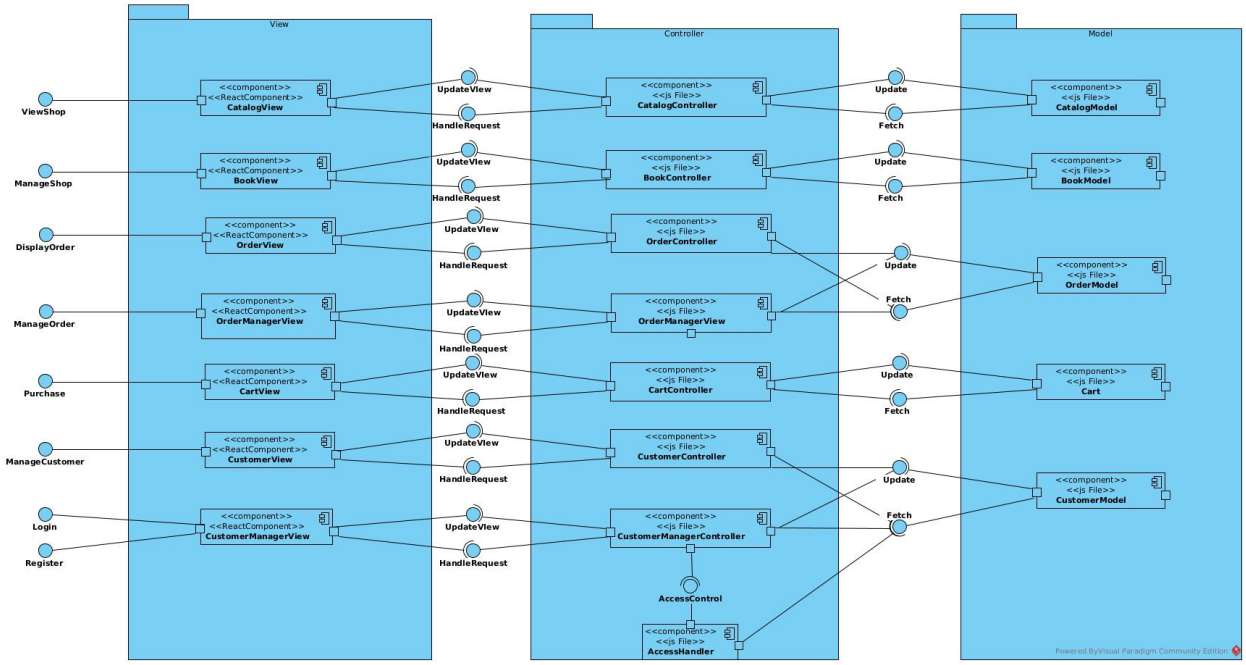


Figure 31: Component Diagram

3.4.1 Deployment Diagram

In order to define physical architecture component diagram is followed by an Deployment Diagram that include previous packages and some artifacts.

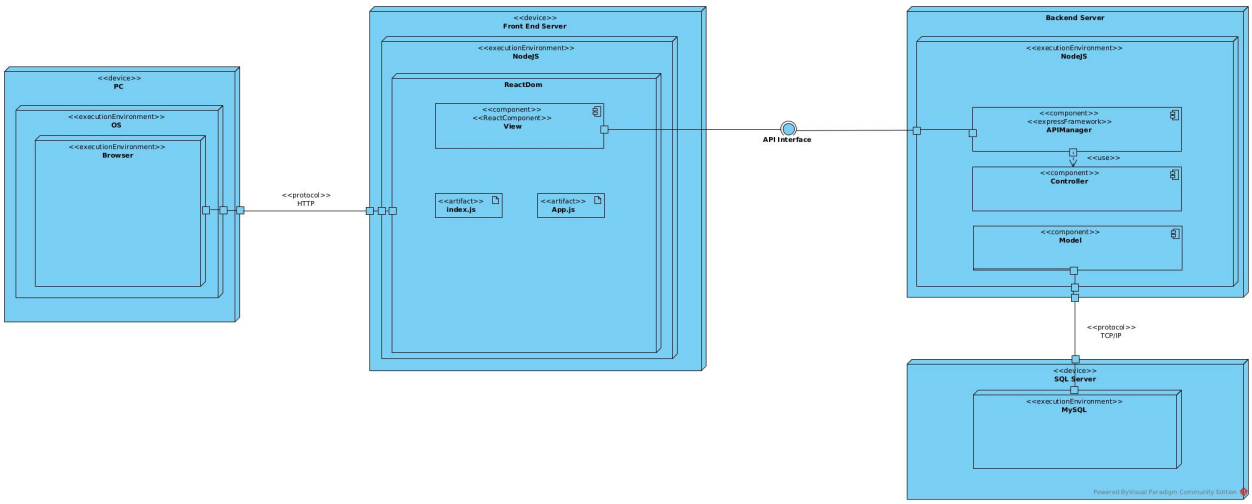


Figure 32: Deployment Diagram

As shown software architecture is defined by:

- Front End Server: implements presentation layer using Nodejs enviroment and React Framework
- Back End Server: implements API Interface (discussed in next chapter) and business layer using Nodejs enviroment and Express Middleware.
- SQL Server: implements Data Access Layer using a MySQL server.

4 Web Services

A Web Service is a system that automatically supports machine-to-machine interactions, facilitating interoperability between heterogeneous systems. The implementation of a system of this type can take place using different architectural styles, including REST, which is based on the use of HTTP methods, such as GET and POST, uniquely identifying resources through a URI. To define the interface of the reference web service, OpenAPI can be used that is a standard to define structure and syntax of REST API.

4.1 REST API Inventory

In this project APICUR.IO is been used in order to define REST API, using OpenAPI standard.

In order to implements differents services of this project, there are 9 endpoints:

- /api/account
- /api/account/card
- /api/book
- /api/cart
- /api/catalog
- /api/checkout
- /api/login
- /api/order
- /api/order/shipping

Let's discuss about them

4.1.1 API /api/login

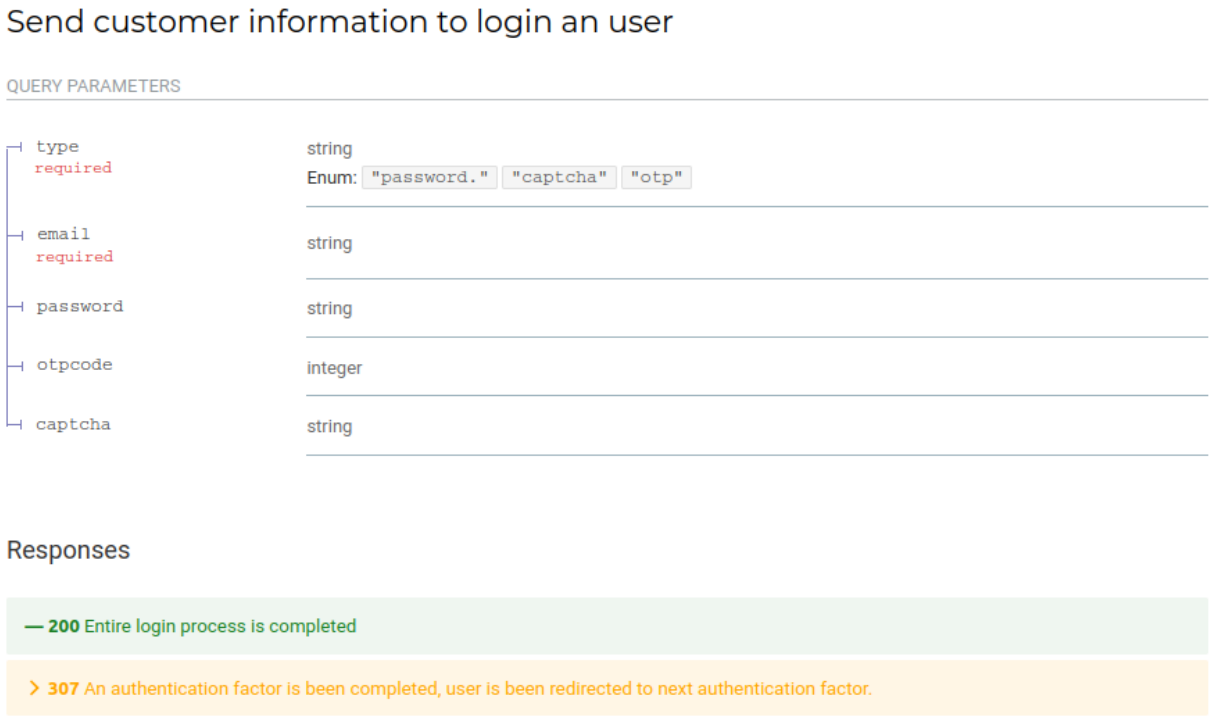


Figure 33: Login

Using this API an user can login using multifactor authentication.

4.1.1.1 200 OK

User have completed all process login and it obtain an JWT token to authenticate itself.

JWT Token is a signed token that contains public customer information and system signature to avoid crafting of illegal tokens.

This token will be required for all process that require an authentication.

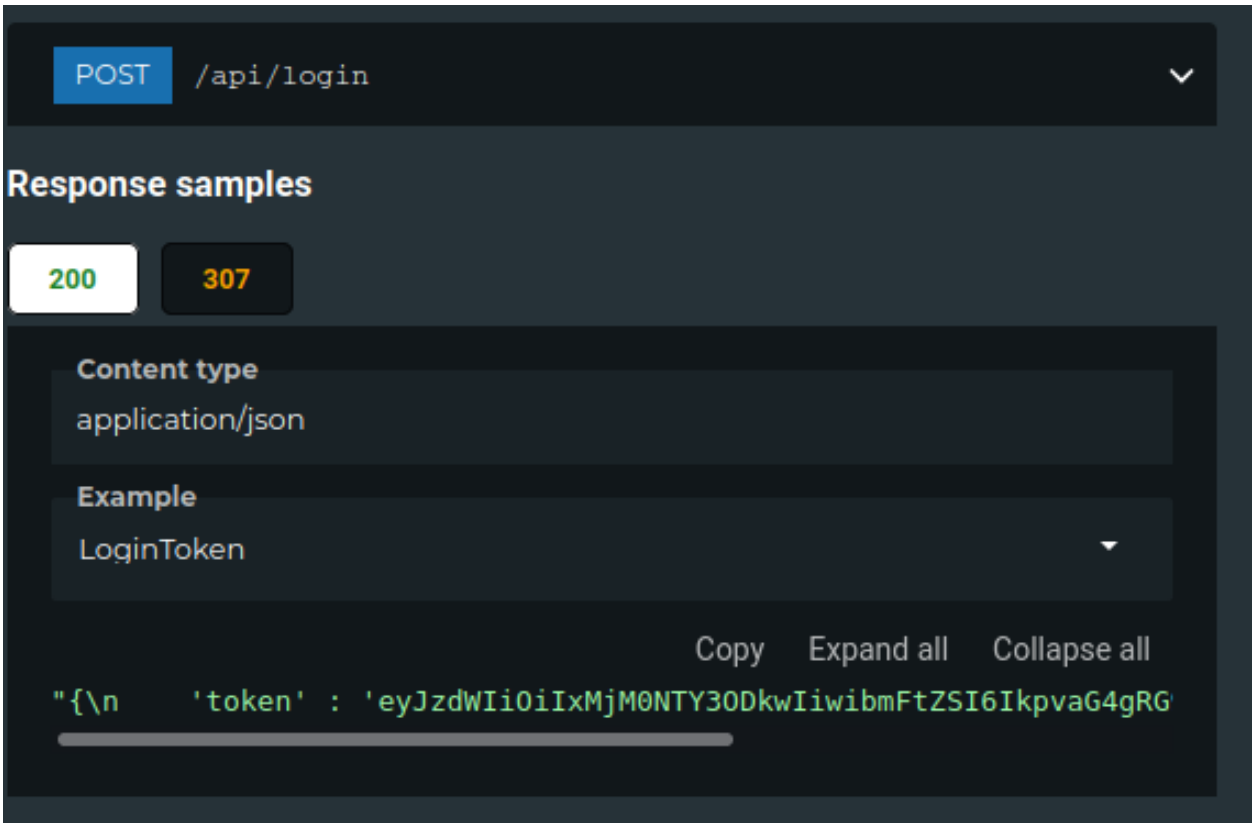


Figure 34: 200

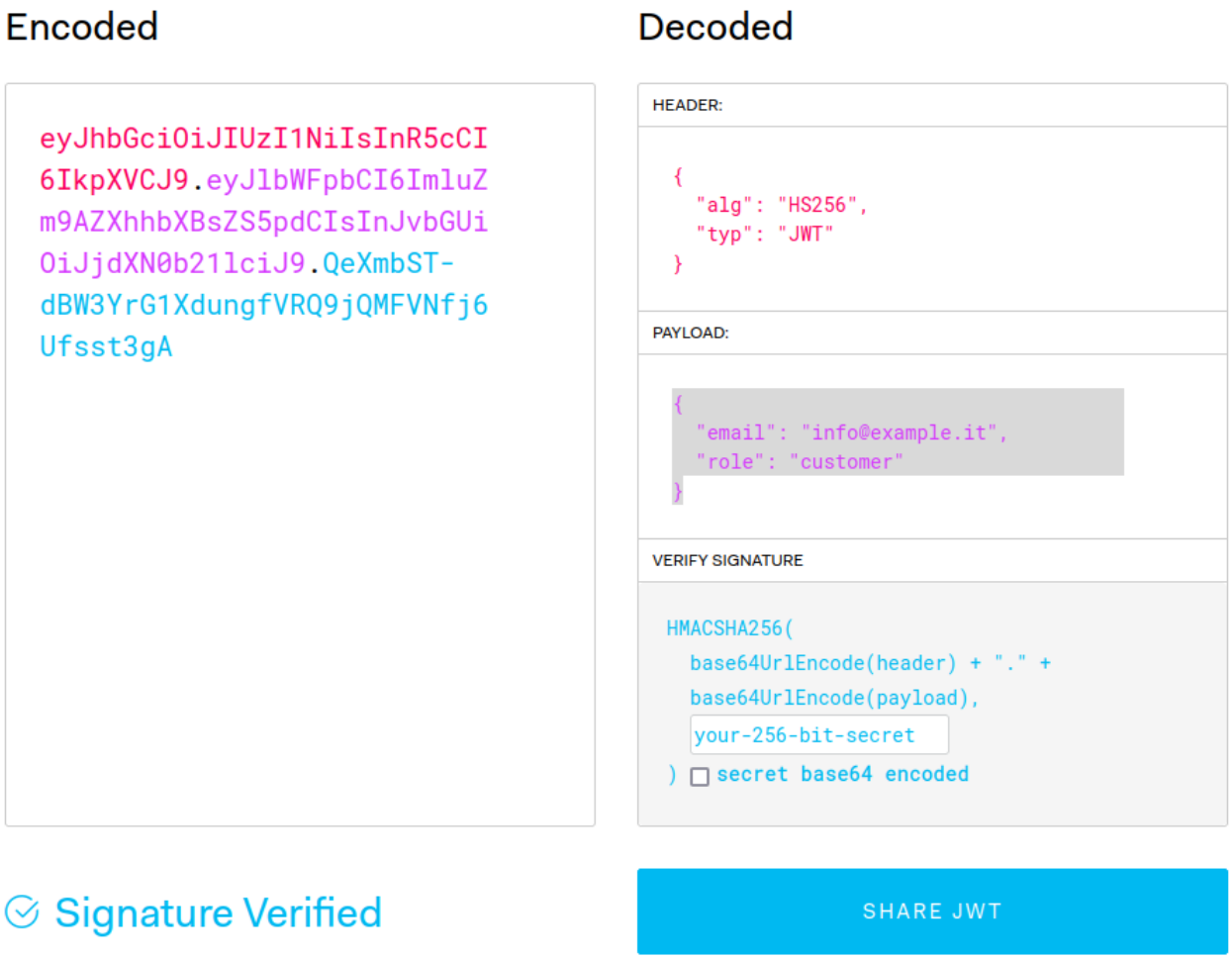


Figure 35: JWT Token

4.1.1.2 307 Temporary Redirect

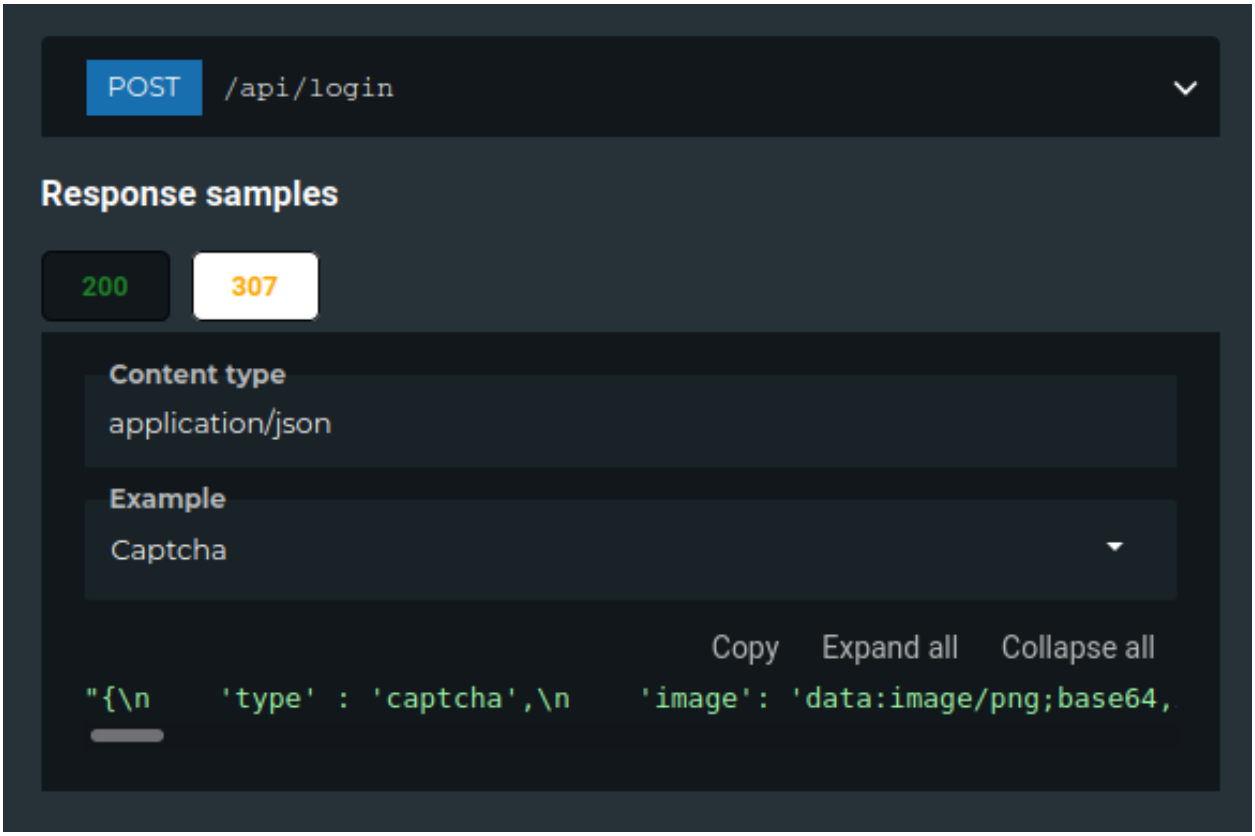


Figure 36: 307

In this example user have input correct password and system replies sending captcha image to solve. This method permits to handle multifactor login.

4.1.2 API /api/account

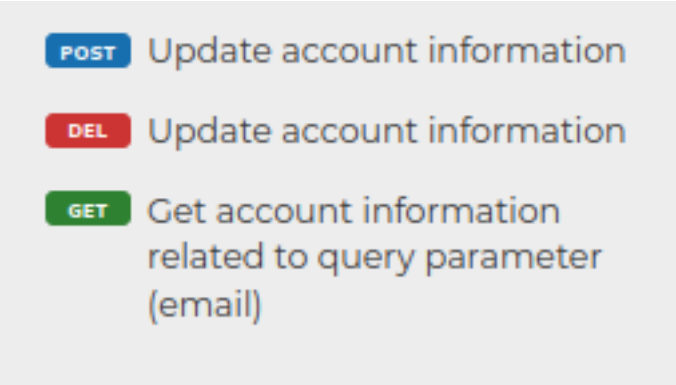


Figure 37: Account REST

This endpoint permit different methods for different actions.

4.1.2.1 GET

Customer or administrator can get costumer details accounts. Note that token obtained in login process is required for customer or administrator identification,

Get account information related to query parameter (email)

QUERY PARAMETERS

email	string

COOKIE PARAMETERS

token required	string

Responses

— **200** Get Costumer information

— **401** Not Administrator attempts to get otherelse costumer information

— **404** Attempt to get no existent costumer info

GET /api/account

Response samples

200

Content type
application/json

Copy Expand all Collapse all

```
"{\n    'email' : 'info@example.it',\n    'phone' : '+39333333333',\
```

4.1.2.2 POST

Customer can update its customer details, note that password is not sended in clear but it is hashed in accord to GPDR (NF_2).

Update account information

QUERY PARAMETERS

email	string
phone	integer
shippingAddress	string
password	string Hashed

COOKIE PARAMETERS

token required	string

Responses

— **200**

— **401** Customer attempts to update otherelse customer.

— **404** Customer attempts to update not existent customer.

POST /api/account

Response samples

200

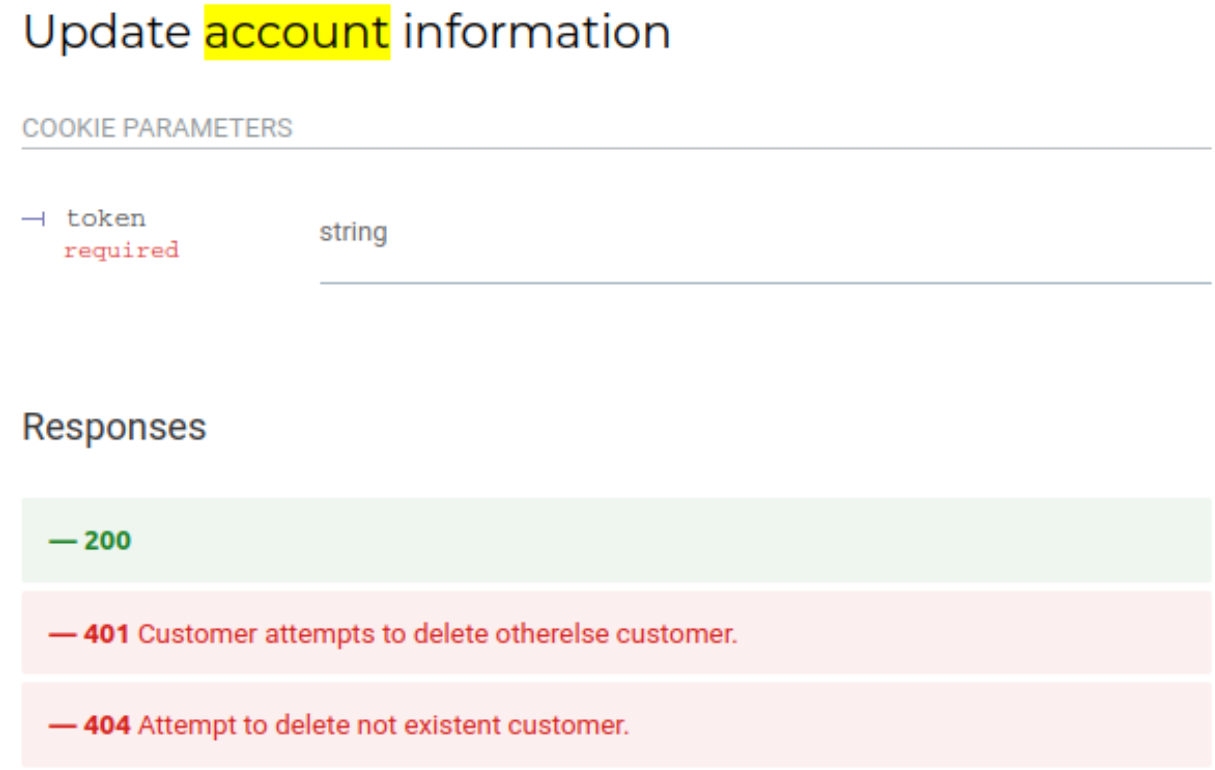
Content type
application/json

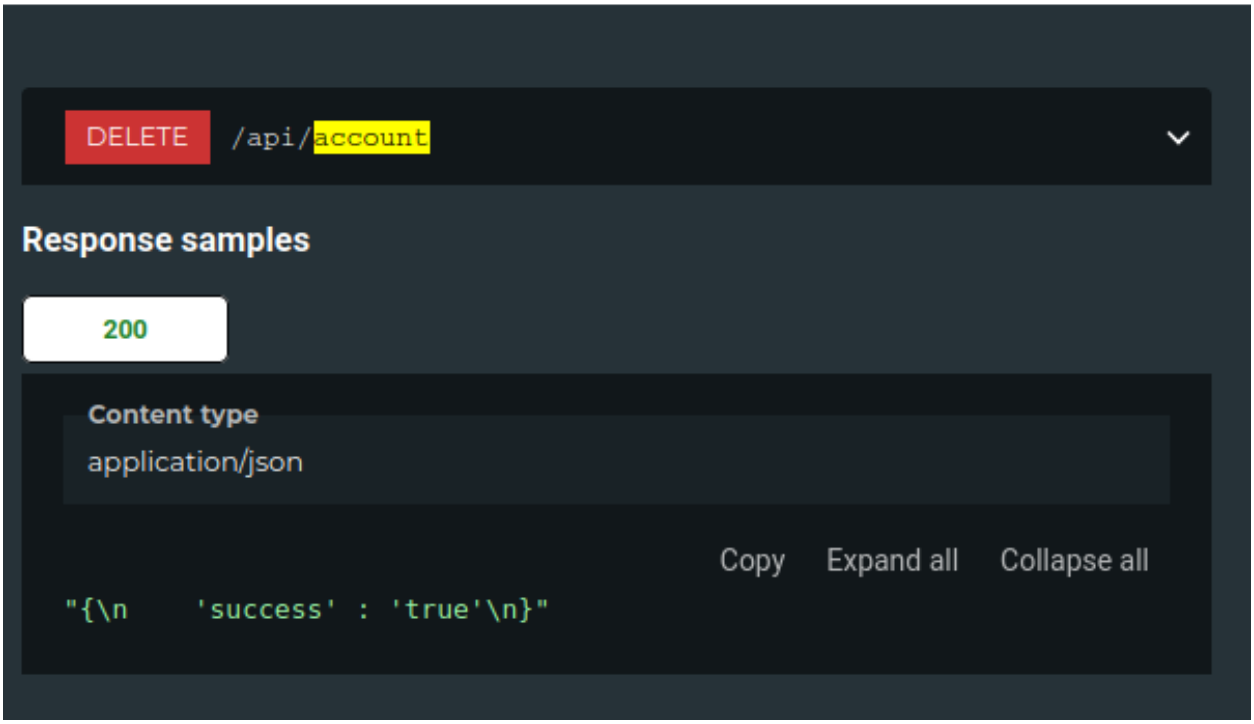
Copy Expand all Collapse all

```
"{\n    'success' : 'true'\n}"
```

4.1.2.3 DELETE

Administrator can delete selected costumer account.





4.1.3 API /api/account/card

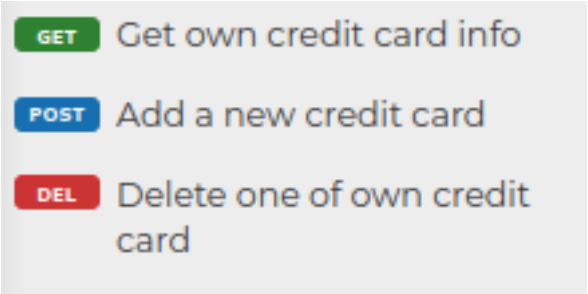
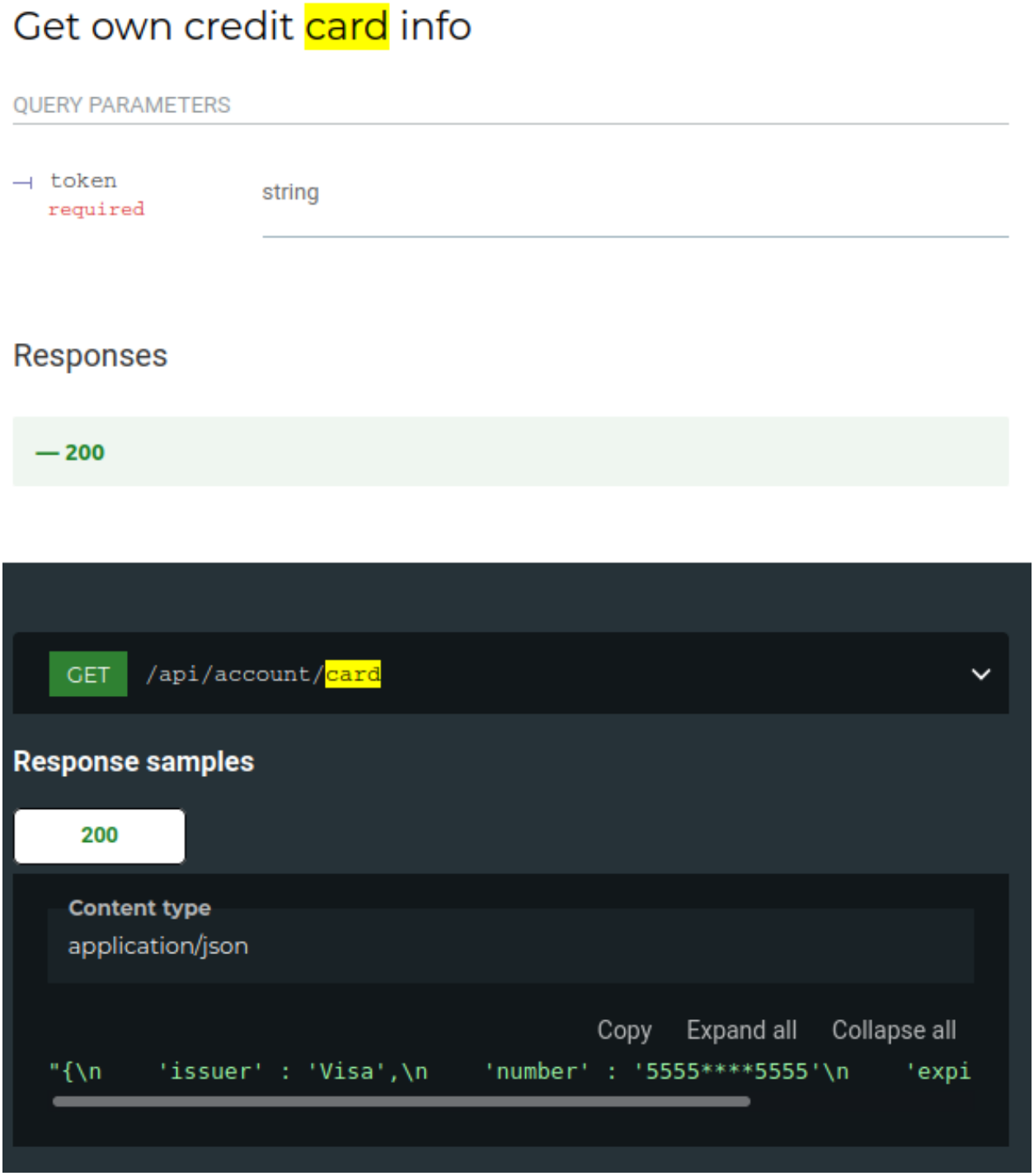


Figure 38: Account Card REST

4.1.3.1 GET

Customer can get its credit card info, note that credit card is hidden to itself in order to improve security.



4.1.3.2 POST

Customer can add a credit card to its account specifing all required parameters.

Add a new credit card

QUERY PARAMETERS

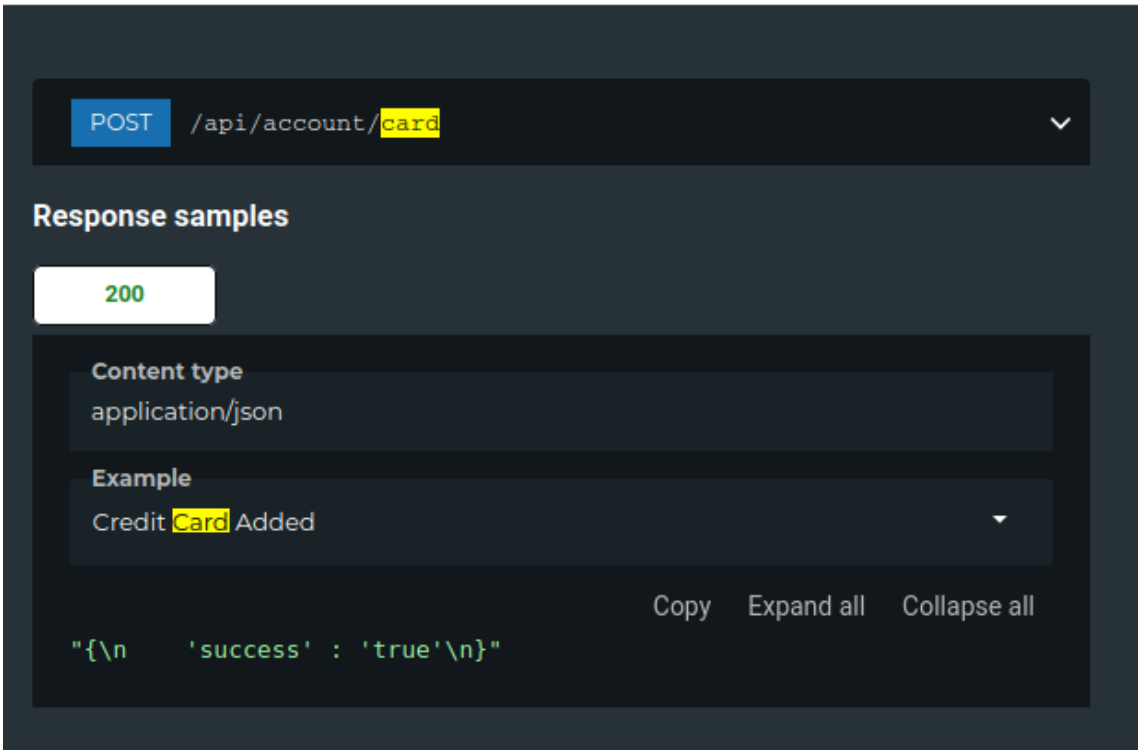
number required	integer	
cvc required	integer	
expire required	string	

COOKIE PARAMETERS

token required	string	

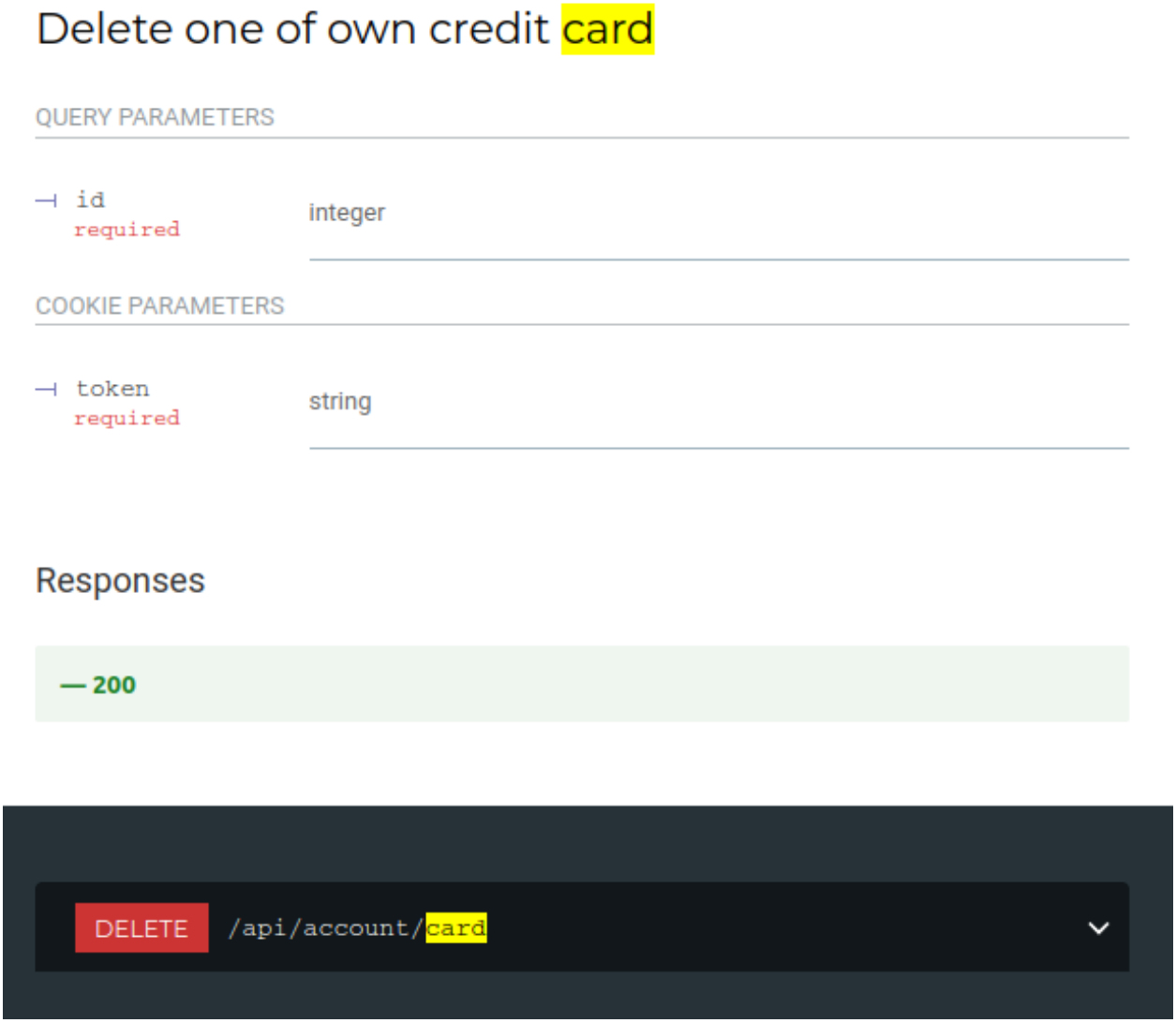
Responses

— 200



4.1.3.3 DELETE

Customer can delete a credit card from its account specifing index.



4.1.4 API /api/book

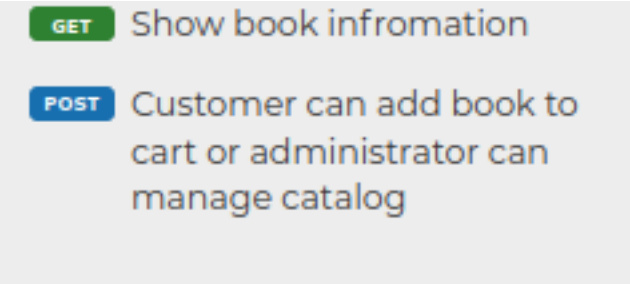
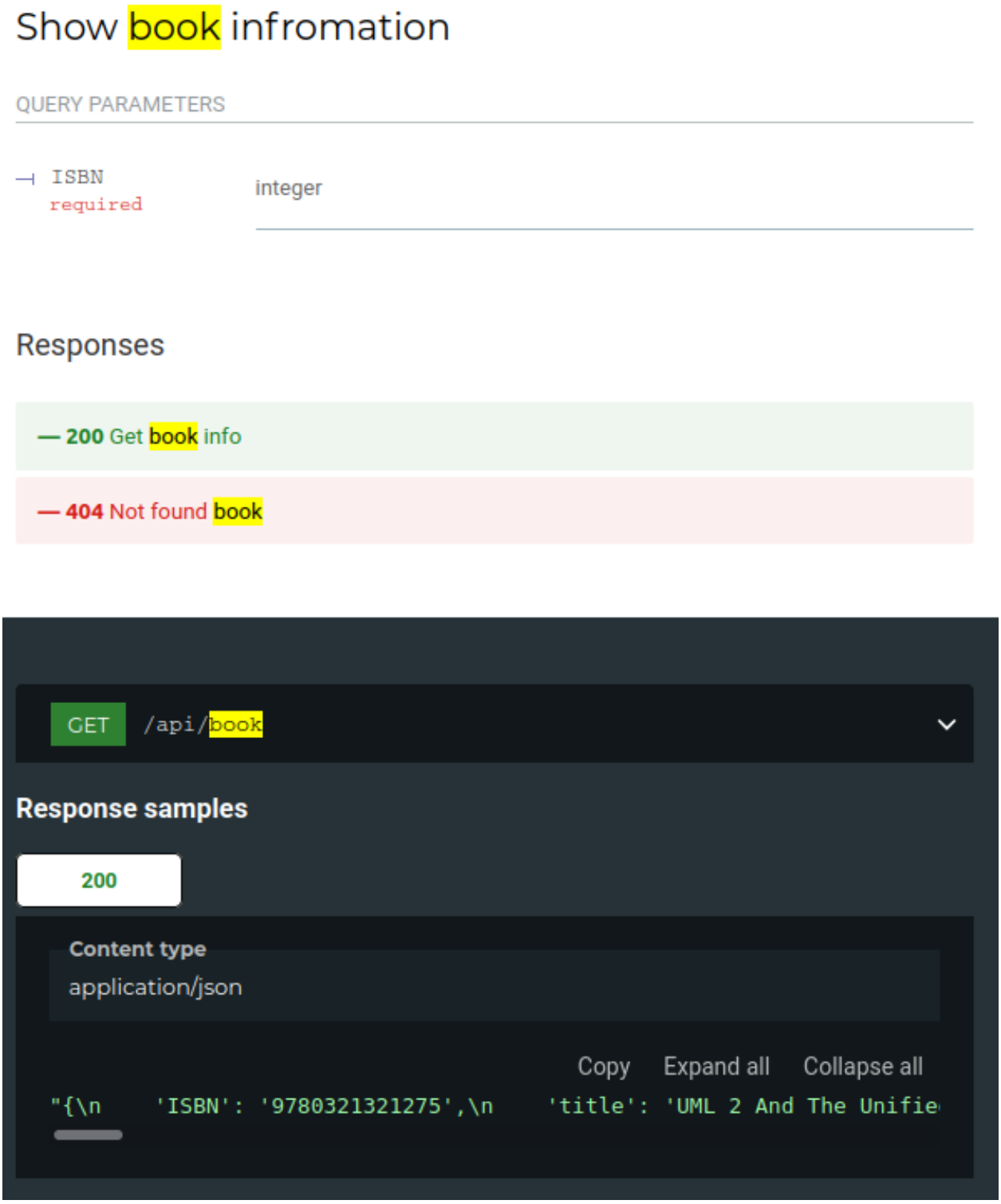


Figure 39: Book REST

4.1.4.1 GET

Any user can watch book information using specified query parameter



4.1.4.2 POST

Note there is a require parameter **action**, it is used by system to apply different action for different roles.

Customer can add book to cart or administrator can manage catalog

QUERY PARAMETERS

action required	string Enum: `"addToCart"` `"editBook"` `"deleteBook"`
ISBN	integer
title	string
authors	Array of strings
publisher	string
price	number <float>
summary	string

Responses

— **200**

— **403** Customer attempts to manage catalog

— **404** Not found book

4.1.5 API /api/catalog

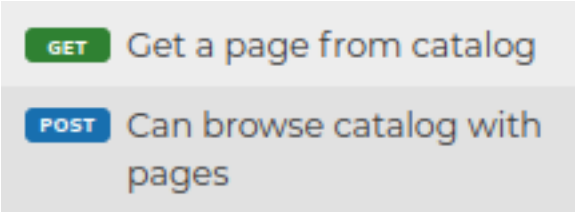


Figure 40: Catalog REST

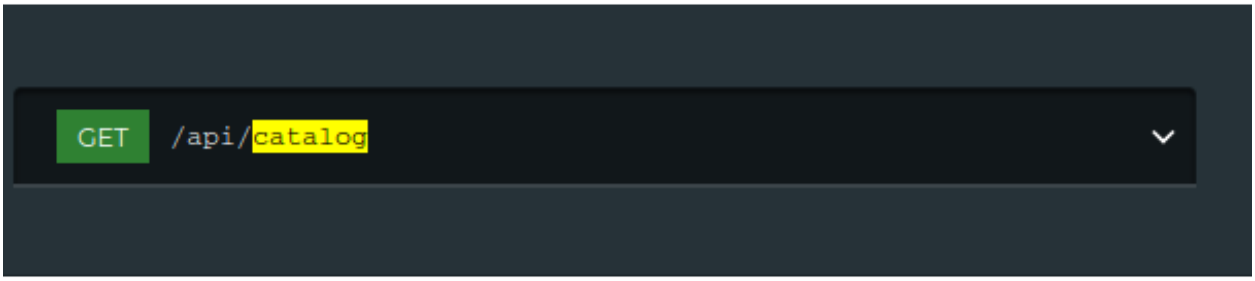
4.1.5.1 GET/POST

Any user can browse catalog using GET e POST requests to surf on catalog

Get a page from catalog

Responses

— 200



Can browse catalog with pages

QUERY PARAMETERS

⊣ action
required

string
Enum: "next" "previuos"

4.1.6 API /api/cart

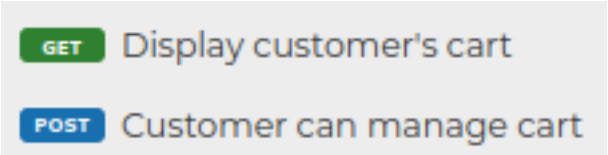


Figure 41: Cart REST

4.1.6.1 GET/POST

Costumer can watch and manage its cart using GET/POST requests and query parameters.

Display customer's cart

COOKIE PARAMETERS

token required	string

GET /api/cart

Customer can manage cart

QUERY PARAMETERS

id required	integer
action required	string Enum: `"updateqty"` `"remove"` `"checkout"`
qty	integer

REQUEST BODY SCHEMA: application/json

Schema not provided

Responses

— 200

— 400

4.1.7 API /api/checkout

Customer try to checkout cart using a card selected by index, in response the System will reply *201 Created* if payment is successful else it will reply *400 Bad Request* showing error information to customer.

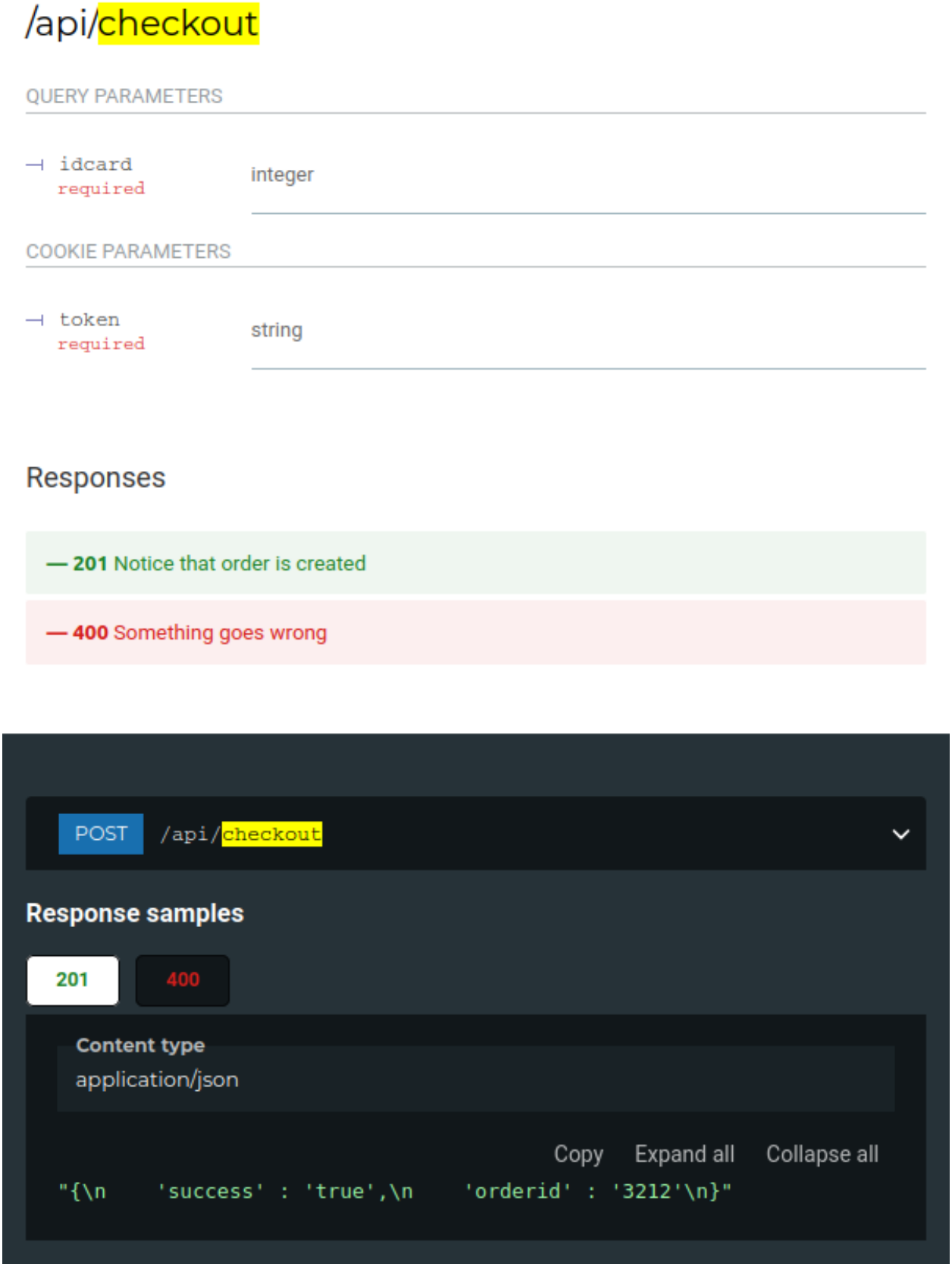


Figure 42: Checkout REST

4.1.8 API /api/order

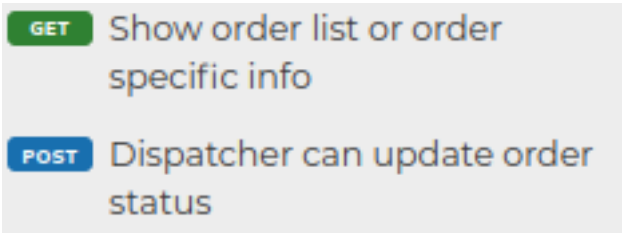
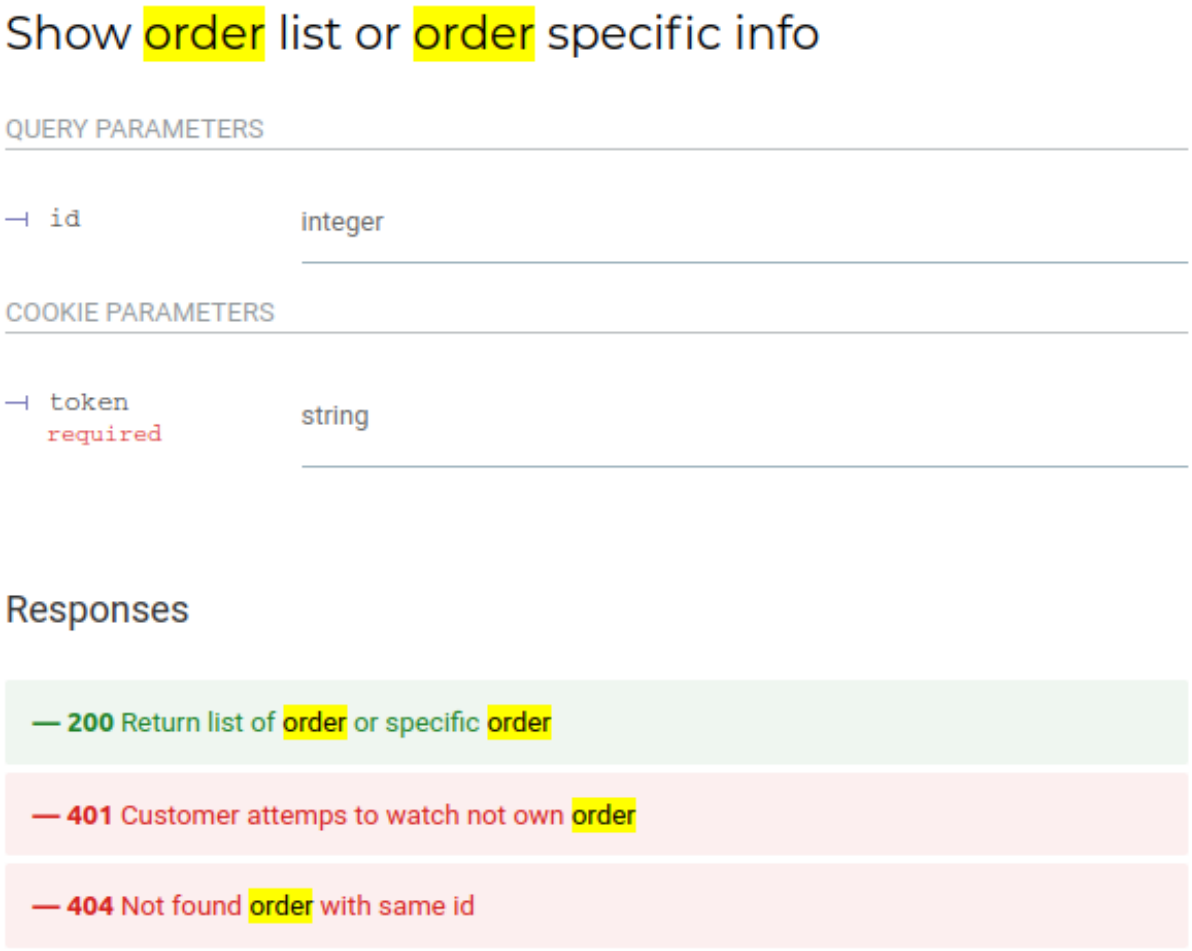


Figure 43: Order REST

4.1.8.1 GET

Customer can get its entire order list if id parameter is not specified else it can get informations for a specific order identified by id



4.1.8.2 POST

Dispatcher can update order status specifing order id.

Dispatcher can update order status

QUERY PARAMETERS

action required	string Enum: `"update"` `"close"`
status required	string
orderid required	integer

COOKIE PARAMETERS

token required	string

Responses

— 200

— 401

— 404

4.1.9 API /api/order/shipping

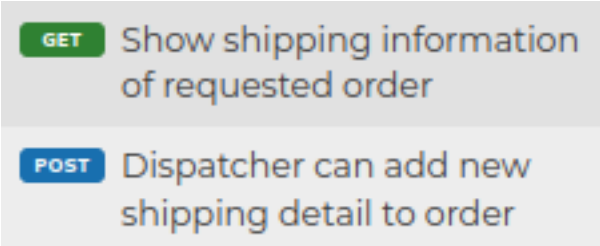
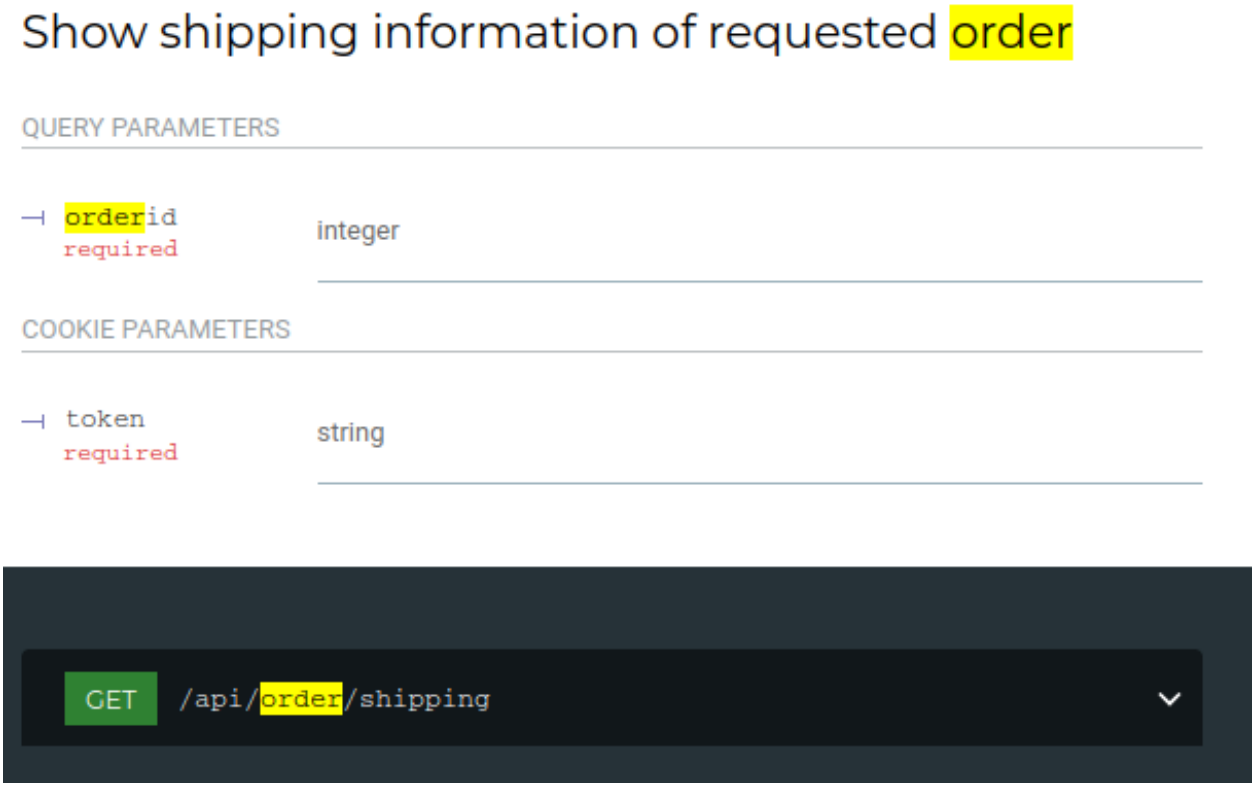


Figure 44: Order Shipping REST

4.1.9.1 GET

Customer can get list of shipping information associated to orderid required parameter.



4.1.9.2 POST

Customer can update list of shipping information associated to orderid required parameter.

Dispatcher can add new shipping detail to order

QUERY PARAMETERS

orderid required	integer	
date required	string	
company required	string	
tracking-number required	integer	

COOKIE PARAMETERS

token required	string	

Responses

— 200

— 401 Customer attempts to change order details

— 404 Order doesn't exists

POST /api/order/shipping

Response samples

200

Content type
application/json

Copy Expand all Collapse all

```
"{\n    'success' : 'true'\n}"
```