POLSKO-JAPOŃSKA
AKADEMIA TECHNIK
KOMPUTEROWYCH

Faculty of Information Technology
MAS

Alisa Kotiun
s20873

BOOKFORM – ONLINE BOOK STORE

DOCUMENTATION

Warsaw, June 2022

Content

User Requirements

BookForm is an IT solution that combines online store and CRM platform. This system connects different departments, from sales to delivery service, as well as organizes notes, activities, and metrics into one system. It is created in order to support Book Store employees with a direct access to the real-time data they need and enable customers to place orders.

1) Book

Book Store wants to save information about all books they sell: ISBN number, Name, Publish Date, Creation Date (may be absent), Genres (there can be many genres for one book, but each book has to have at least one genre), Link to the Picture. There is also a requirement to save details about Cost of the book: Book Value, Discount Value and Final Cost (cost of the book considering a discount). Discount should be in the range from 0 to Max Discount which value is the same for all books.

2) Collection

Collections are used to propose some books to a customer altogether. Collection has a Name and Status (Planned, Active or Withdrawn). Book may belong to only one Collection and Collection can consist of many Books. Only if Book belongs to a Collection its Picture can become a Collection's cover.

3) Author

Each Book is written by at least one Author. For Author there is information about their First Name, Last Name, Date of Birth, and Email. Author can also sign a contract with the Publishing House (Name, Address, Email). Author can sign a contract with the same Publishing House several times (Start Date and End Date of a contract should be saved).

4) Customer

Customer is one of the most important objects in the system. Customers can be divided in two ways:

- By Business Entity: Customer-Company (Name, REGON to save) or Customer-Individual (First Name, Last Name, Date of Birth).
- By Loyalty Status: New, Loyal, Blacklist. For the last one you can add a description (the reason of being added to black list, etc.).

For all the Customers there is also a need to save information about Email, Phone Number, Discount (its max value is different depending on Loyalty Status). Customer can also place many Orders in the store.

5) Order

Order (Creation Date, if it is Delivered, isSubmitted, Final Price) made by Customer consists of many Books. Each Book can be a part of many Orders. For each Book within the Order a Quantity field must be filled.

6) Cart

Cart (Creation Date, Price) is similar to Order, but it has no association with Customer. Cart may contain many Books, as well as Collections.

7) Employee

Book Store desires to monitor information about its workers, as well as give some system permissions depending on their roles. Employee information: First Name, Last Name, Date of Birth, Login, Password, Salary. There are only 3 types of Employees: Courier, Consultant, Manager. Employee may combine different roles at the same time. For each type of the employee we would like to save information about the value of Benefits that may be added to a Salary.

Beside CRUD operations, the BookForm System should also implement the following functionalities:

- Listing all Books published after a certain date;
- Finding Books by ISBN code;
- Calculating a final cost of the Book;
- Listing Authors currently working for a specific Publishing House;
- Calculating a price of a Book for a particular Customer, considering their discounts;
- Listing current Customer's Orders (not yet delivered ones);
- Calculating a final price of the Order;
- Calculate a Salary of the Employee (with benefits considered or not).

Use Case Diagram

Here is a Use Case Diagram that describes main functions of a system. It also identifies the interactions between the system and its actors.

Actors: Anonymous User, Customer, Courier, Consultant, Manager

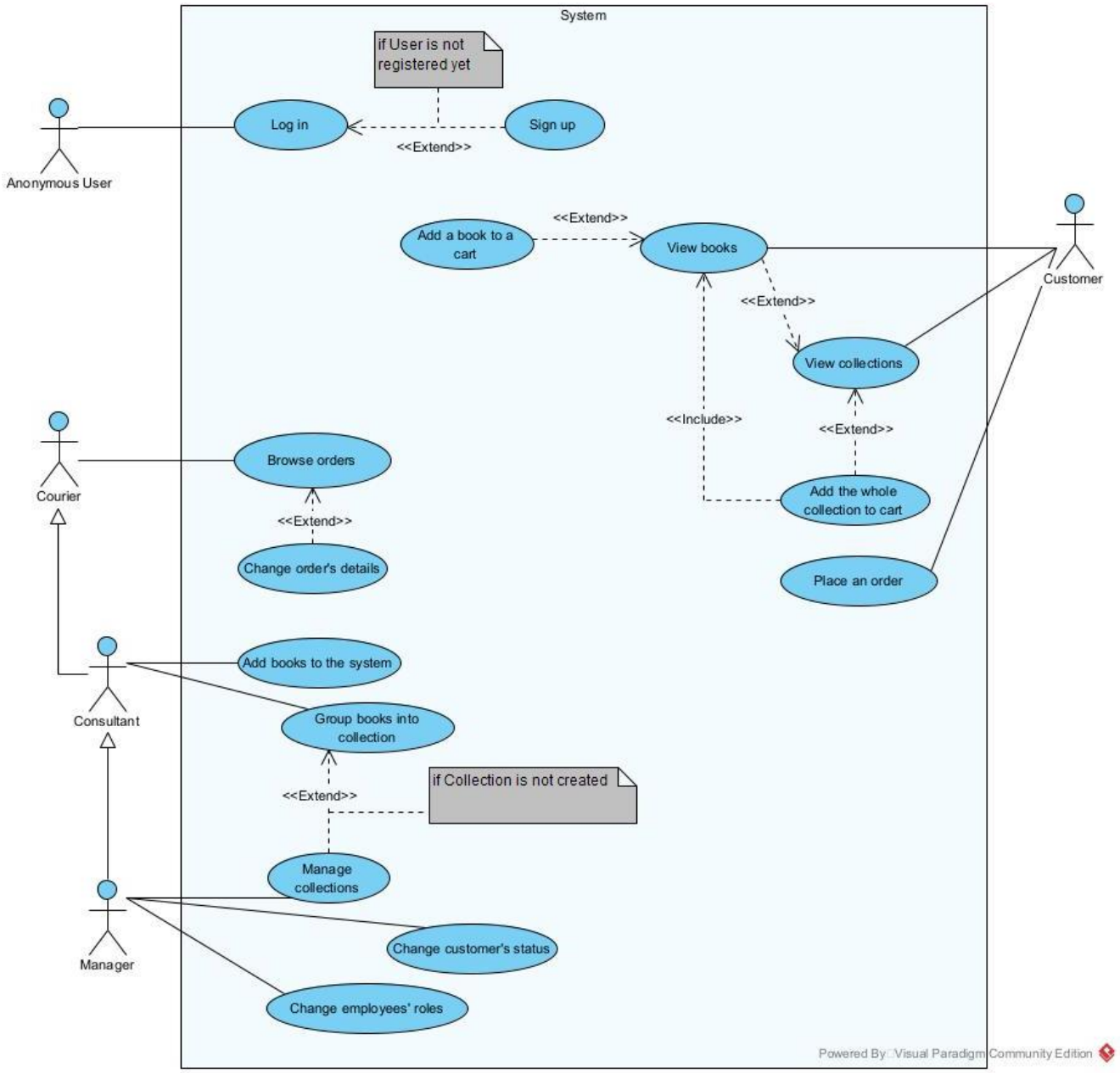

Figure 1 Use Case Diagram

Analytical Class Diagram

Here is an Analytical Class Diagram that significantly impacted the understanding of the requirements of a problem domain as well as identifying its components.

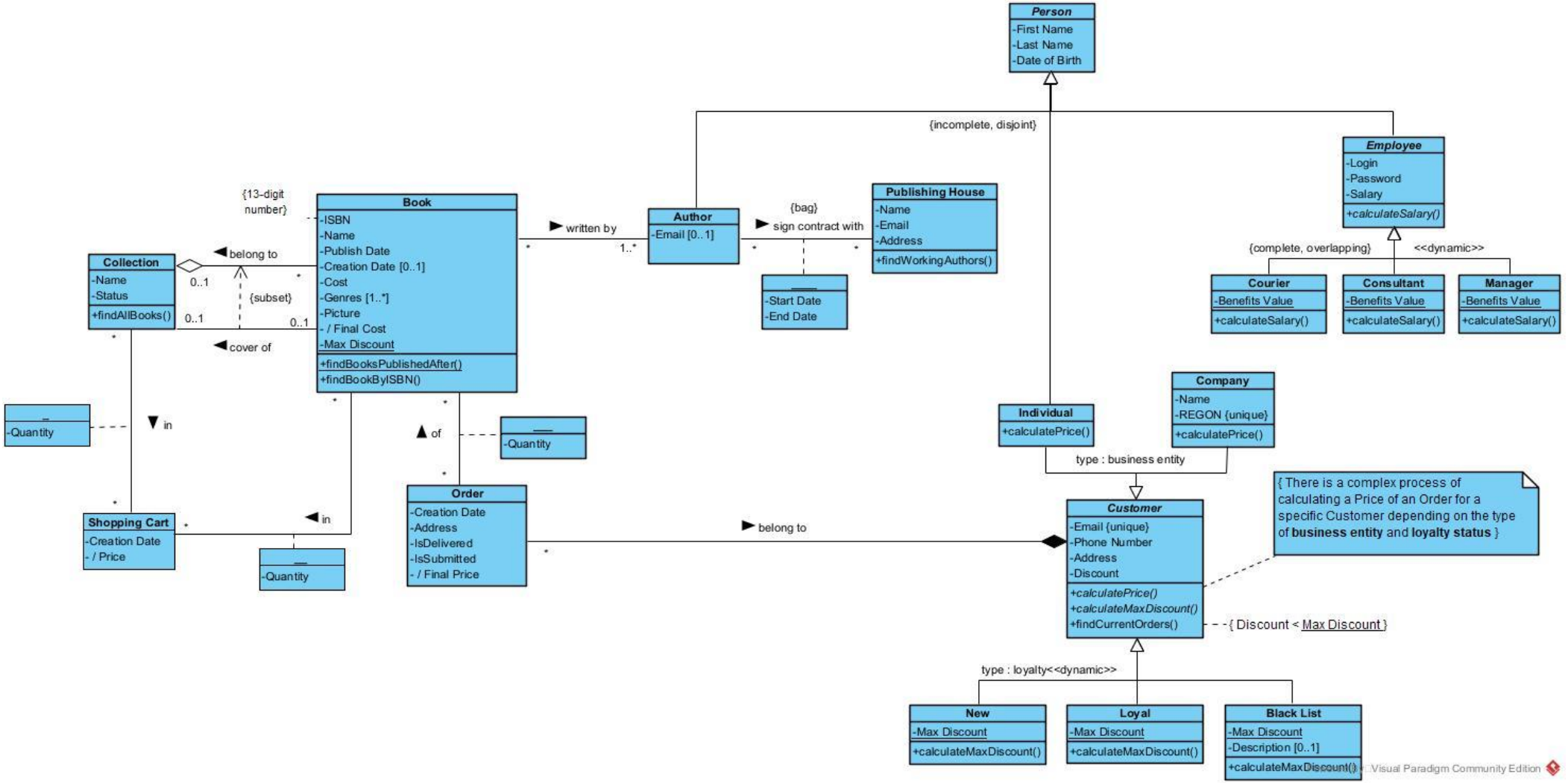

Figure 2 Analytical Class Diagram

Design Class Diagram

Here is a Design Class Diagram that illustartes the specification for the software classes and methods that participate int the software solution.

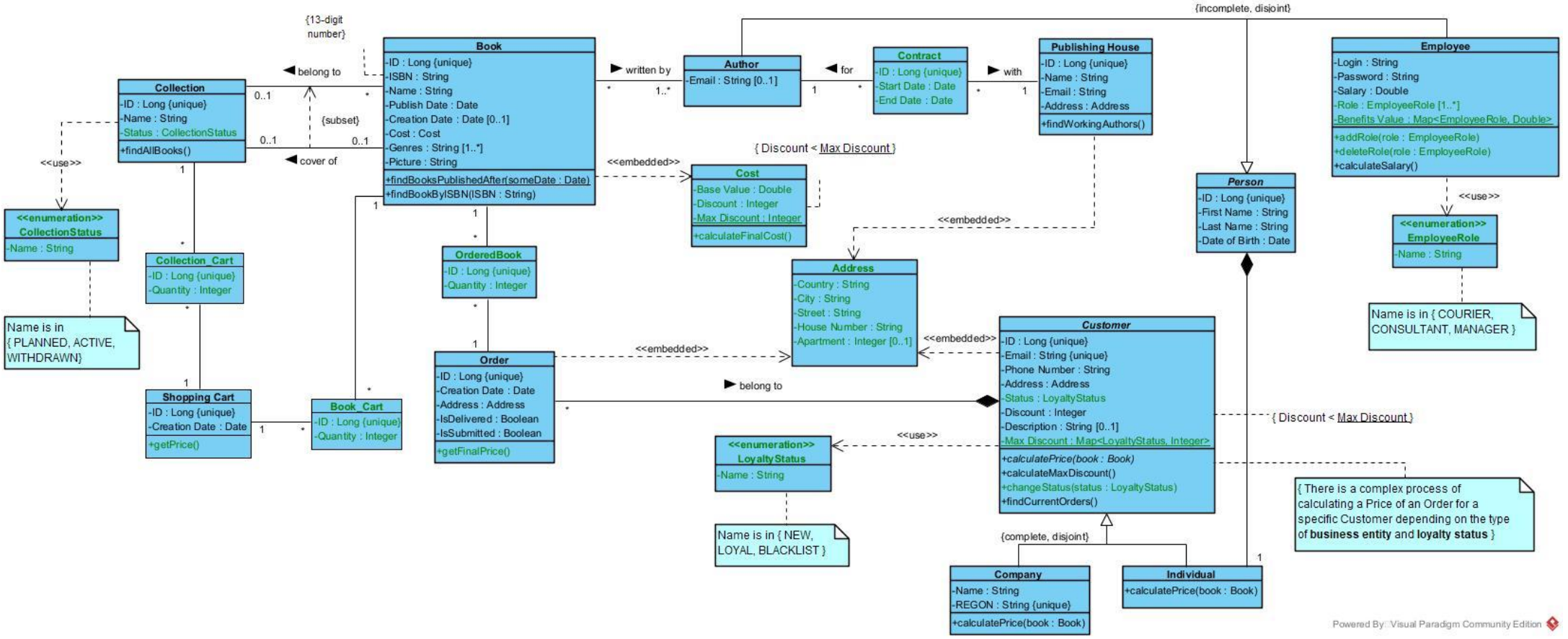

Figure 3 Design Class Diagram

Scenario of a Use Case “Add the whole collection to cart”

- **Name**

“Add the whole collection to cart”

- **Actors**

Customer

- **Purpose and Context**

The use case was created for customers to add the whole collection of books to the shopping cart. Collection consists of many books, so whenever a customer confirms a choice of a collection, all the books are listed and collection is added.

- **Dependencies**
    - **Included use-cases**

“View books” – enabling a customer to view all books in a collection. This use-case is triggered after a customer enters a quantity and before he or she makes a final decision whether to add it or not.

    - **Extended use-cases**

“View collections” – enabling a customer to view collections available in the system. If the customer wants to buy a collection, this use case is extended with “Order the whole collection” use case.

- **Assumptions and Pre-conditions**

Before triggering the following use case, we assume that:

1. Customer accessed an online store web page
2. Customer viewed a list of available collections

Note: in a case of a customer there is no need to be signed up and logged in to the system.

- **Initiating business event**

A trigger for this use case basically consists of two parts:

- a desire of customer to view available collections,
- a desire of customer to buy selected collection.

- **Basic flow of events**

1. The use case starts
2. A customer clicks a button ‘Order a collection’
3. A confirmation window shows up
4. A customer enters a number of collections ordered
5. A “summary” with a list of books shows up
6. A customer confirms a choice
7. A window informing about successful confirmation shows up
8. The use case ends

- ## Alternative flow of events

  a. **Customer refuses to confirm a choice**

  a6. A customer cancels a choice

  a7. A window informing about refusal shows up

  a8. The use case ends

  b. **Customer didn't enter a number of ordered collections**

  b5. A window informing about unsuccessful confirmation shows up

  b6. Continue from point 3 of main scenario

- ## Extension points

None

- ## Post-conditions

  1. A selected collection is added to a shopping cart

Note: in the case of alternative flow A, no collection will be added to a shopping cart.

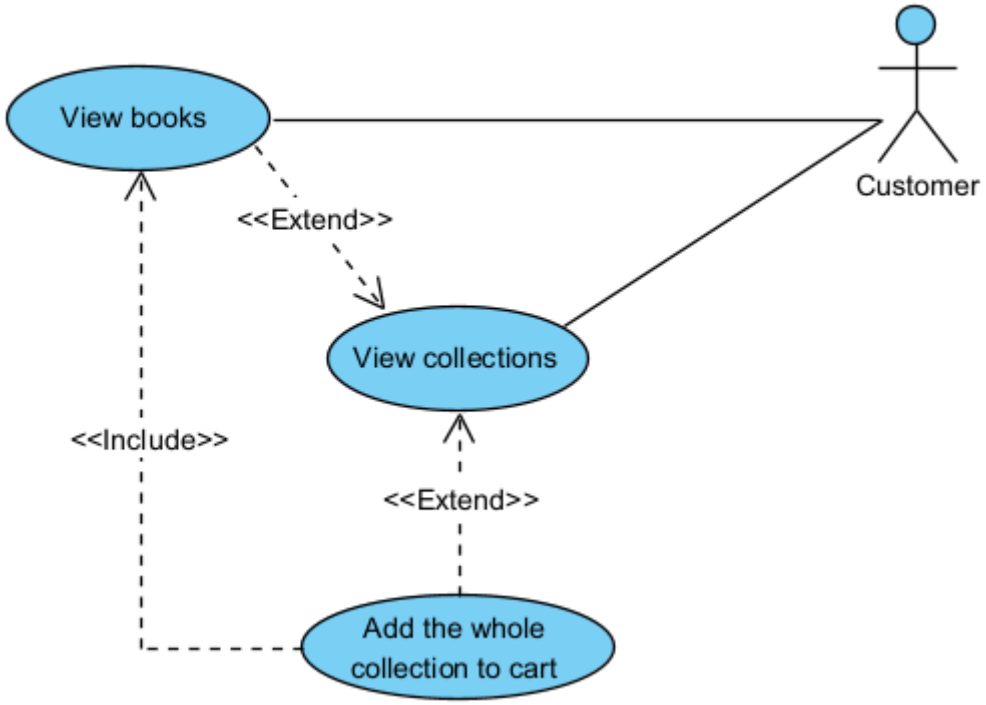

Figure 4 Add the whole collection to cart

Activity Diagram for a Use Case “Add the whole collection to cart”

Here is an Activity Diagram for a Use Case “Add the whole collection to cart” that describes the control flow from a start point to a finish point, as well as shows the various decision paths that exist while the use case is being executed.

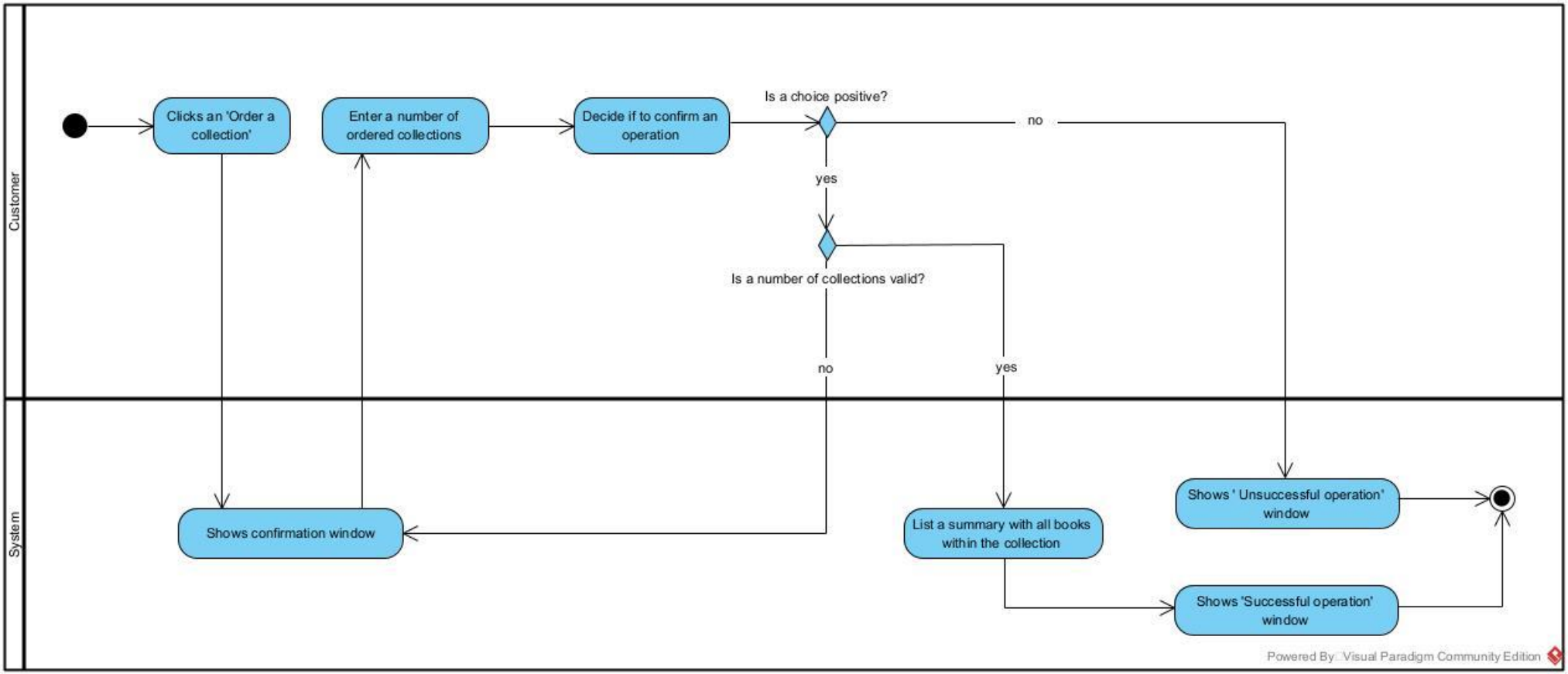

Figure 5 Activity Diagram

State Diagram for “Collection” Class

Here is a State Diagram for “Collection” class that illustrates all the states (statuses) an instance can attain, as well as the transitions between those states.

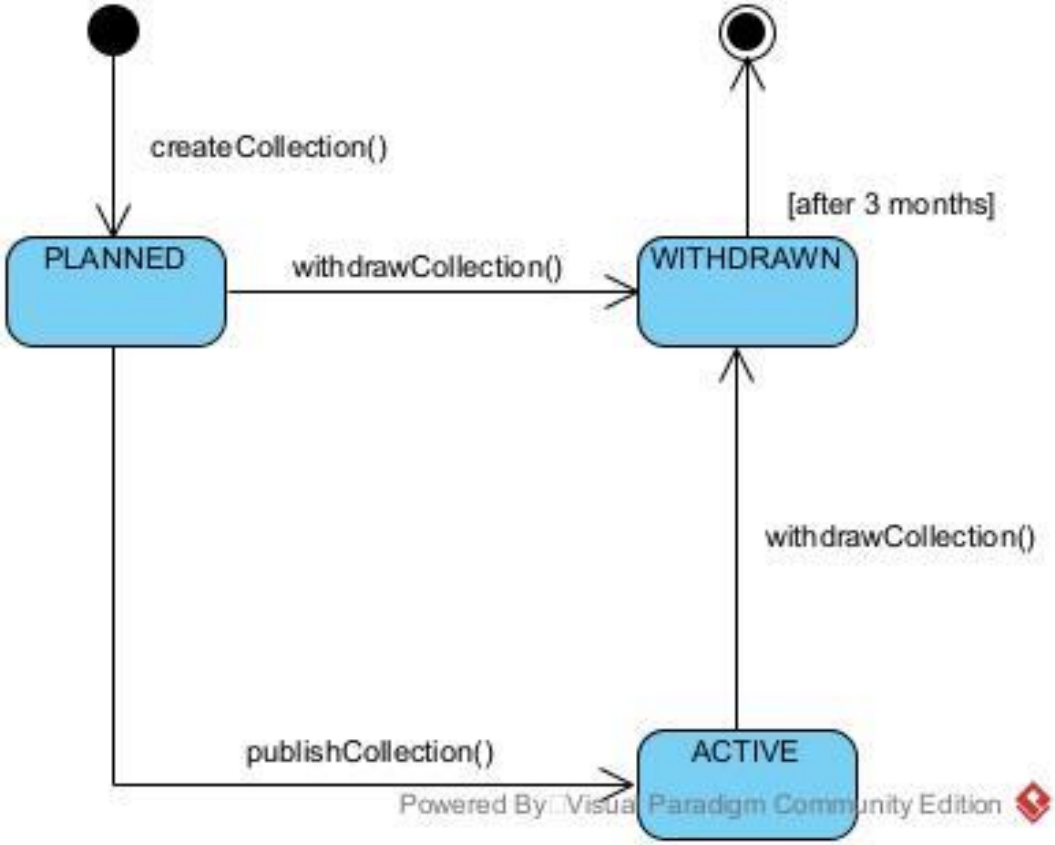

Figure 6 State Diagram

Interaction (Sequence) Diagram for a Use Case “Add the whole collection to cart”

Here is an Interaction (Sequence) Diagram for a Use Case “Add the whole collection to cart” describes the interaction logic between the objects in the system considering the time order.

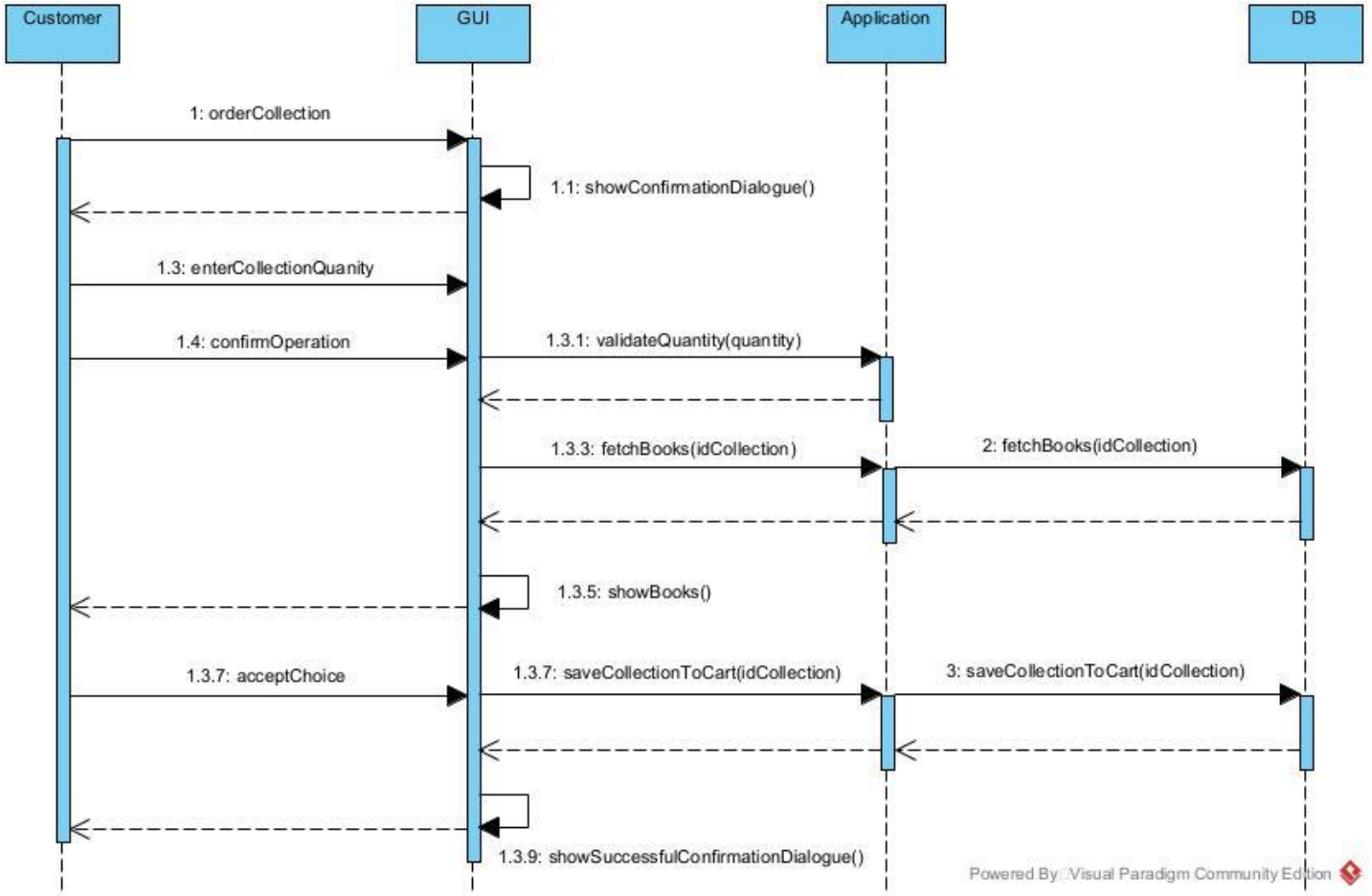

Figure 7 Sequence Diagram

GUI Design

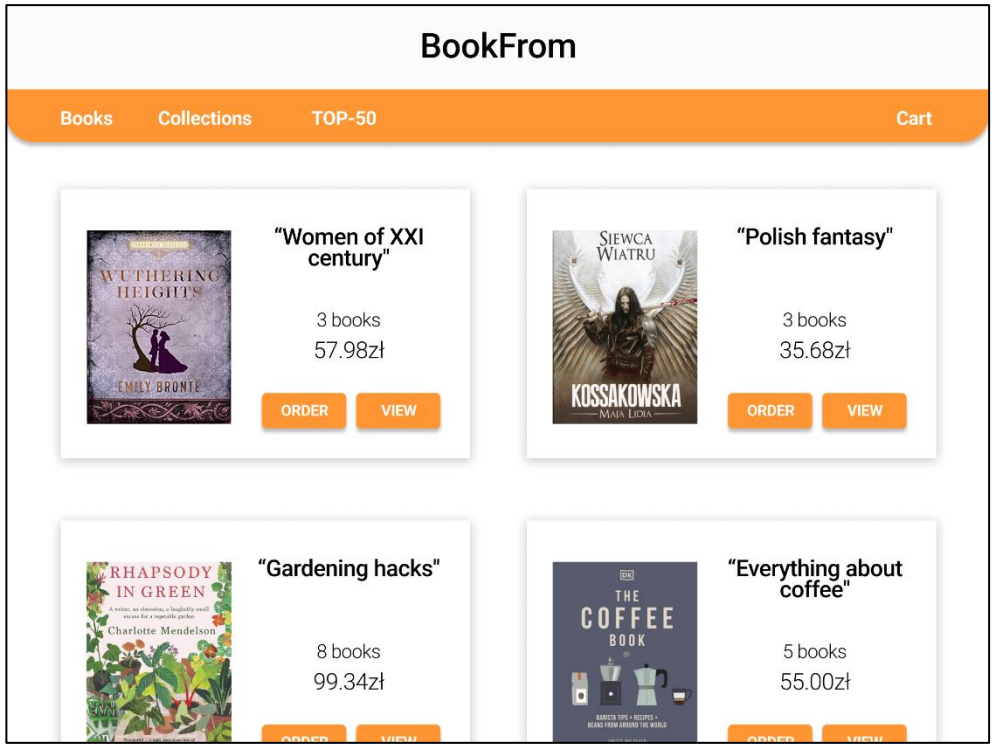

Figure 8 View Collections

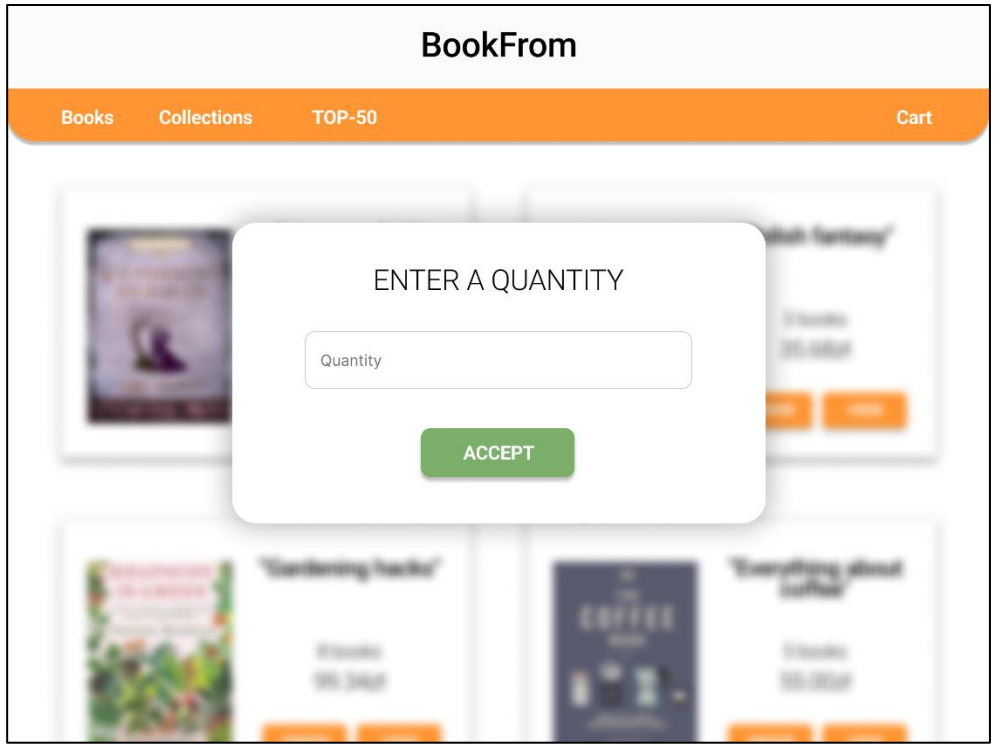

Figure 9 Show Confirmation Window

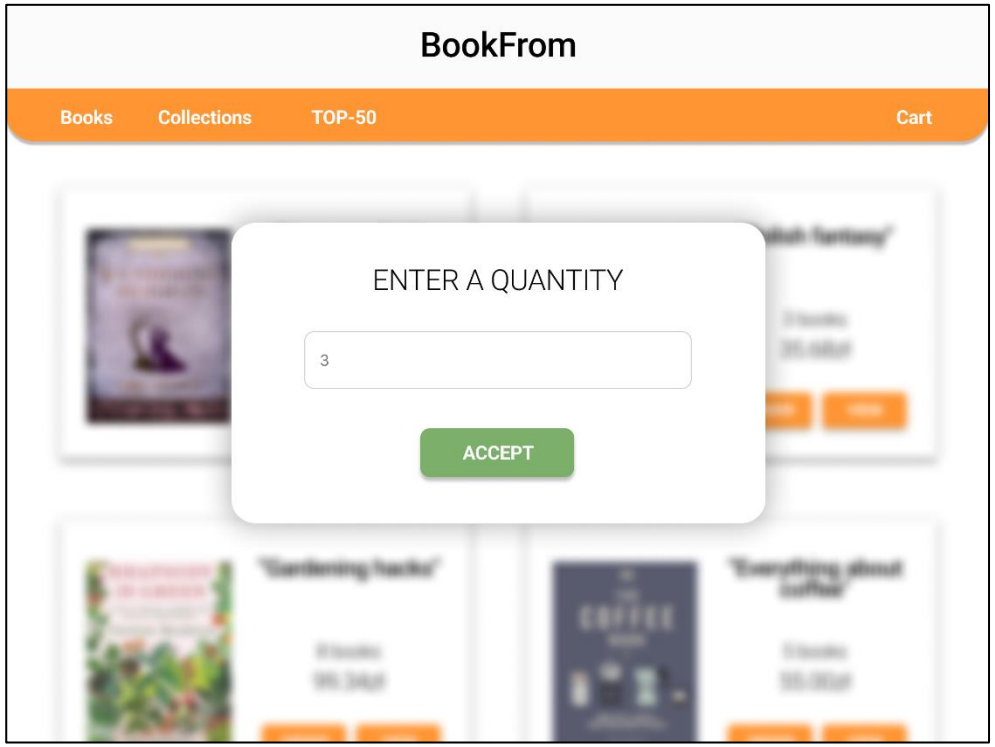

Figure 10 Enter Quantity

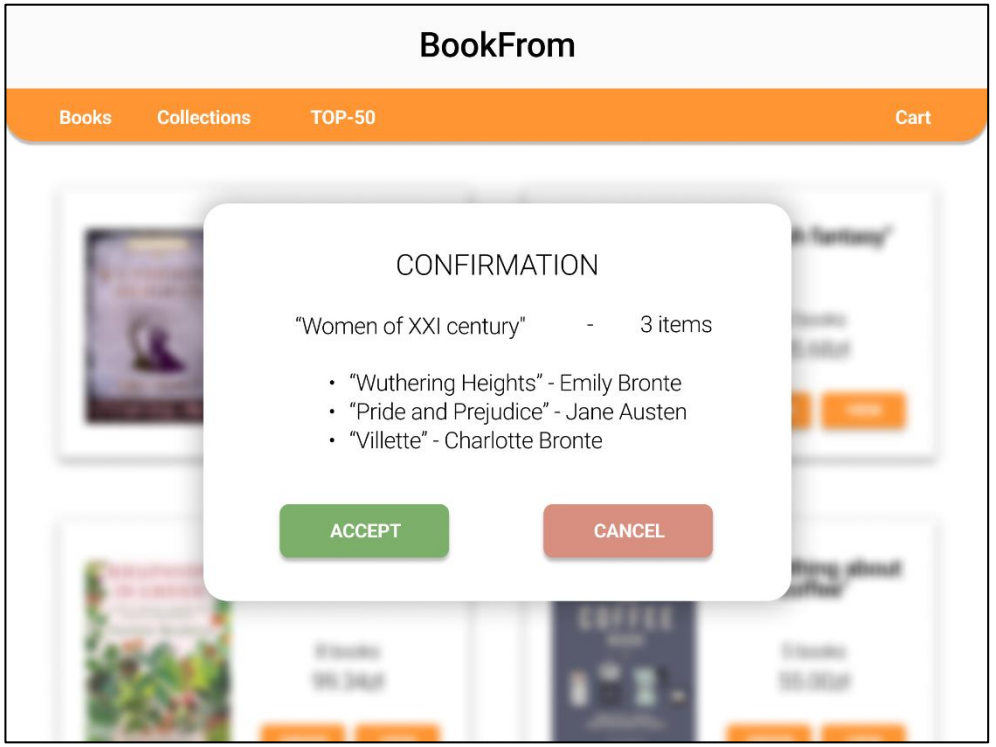

Figure 11 List all books within the collection

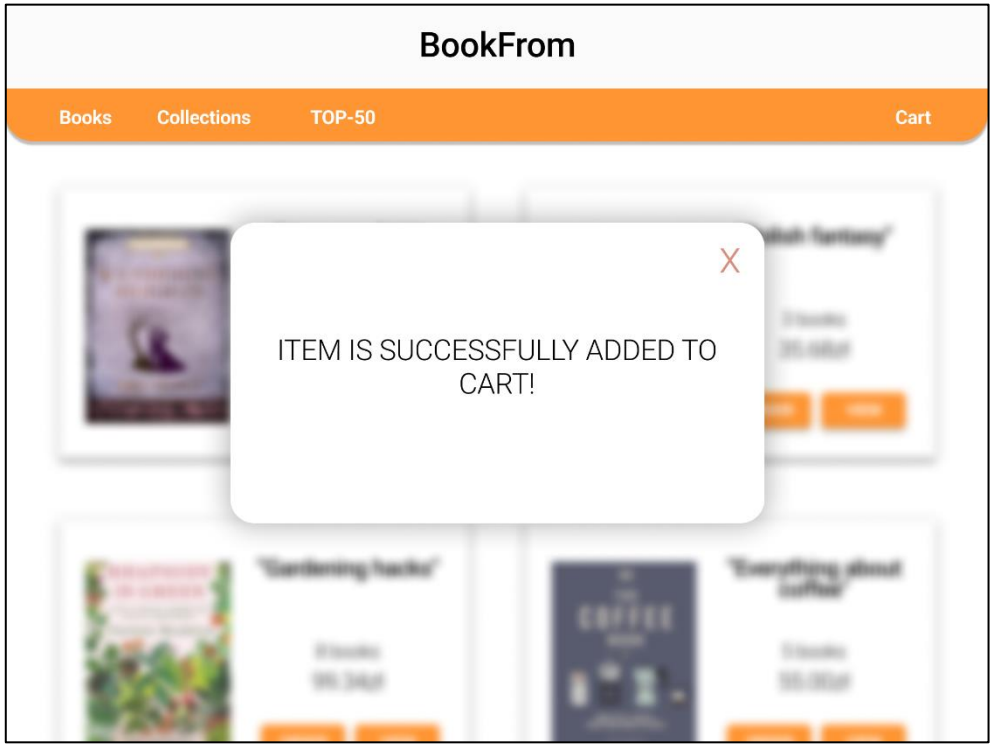

Figure 12 Show 'Successful operation' window

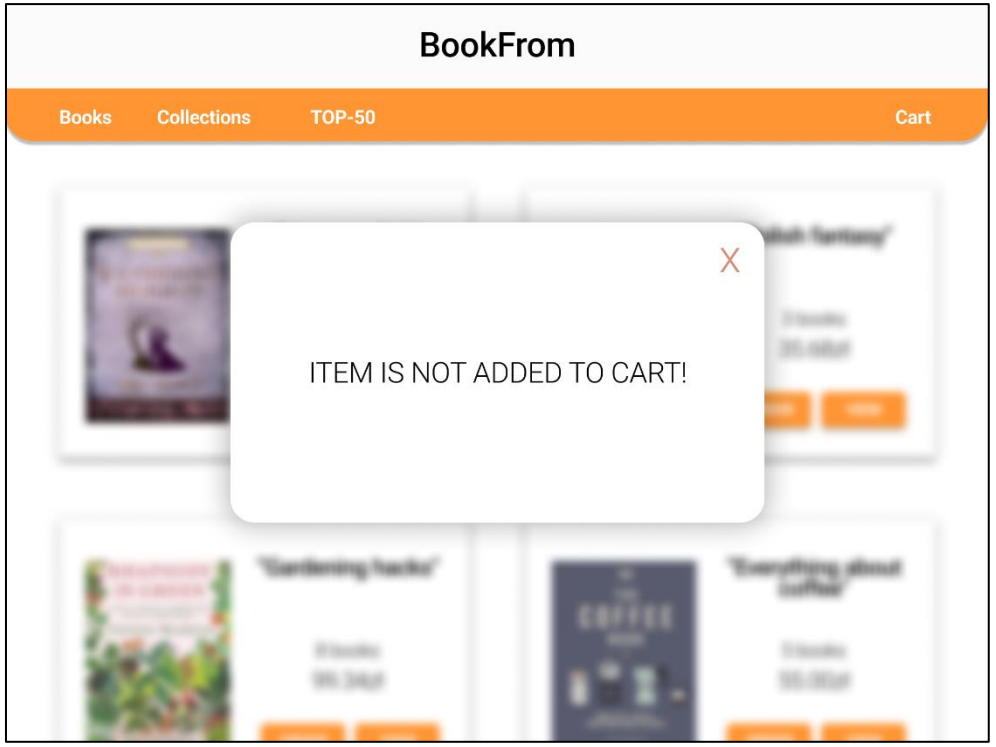

Figure 13 Show 'Unsuccessful operation' window

The Discussion of Design Decisions and the Effect of Dynamic Analysis

During product planning, development and evolution, the set of requirements that were previously established may change. Those changes are essential, because they impact the success of the product. And lower you can find a description how dynamic analysis effected a design decision and vice versa, but firstly it is worth mentioning what technologies were chosen during planning phase.

Here is a list of technologies, tools, architectures that are supposed to be used during a development:

Java: Spring, Spring Boot, JPA, Hibernate
Web development: Spring MVC

The above technologies support different IT decisions and what is more important a developer (me) is familiar with most of them. Spring ecosystem is a great choice for Web application development, while JPA and Hibernate support database maintenance and reduce lines of code for object-table mapping.
*Since during development project requirements may change, the technologies and tools may change as well. That's why the existing list of tools may change in the future

Previously you could find results of the performed Dynamic Analysis: Use Case diagram, Activity diagram, Sequence diagram, Class Diagram etc. This phase of development significantly impacted some design decisions:

Inheritance implementation: In the project there is a wide range of inheritances used which needed a lot of consideration on how to implement them. After performing Dynamic Analysis, we got:

- Overlapping dynamic inheritance for Employee class was flattened and the modifiable list of Roles was created instead;
- Multi-aspect inheritance for Customer (type: loyalty, business entity) was also simplified to the regular one (Company and Individual extend from Customer). In addition, a Status field was added to Customer class to represent the loyalty type;
- Multi-inheritance for Customer-Individual wasn't adapted usually – it was replaced with a composition between Person and Individual.

Constrains impact: After analyzing existing constraints for a business process, it turned out they should be handled gracefully as well:

- For some classes (Collection, Customer, Employee) there are some fields that can have only predefined values. Such a constraint was implemented by introducing Enum types for the above classes;
- For overlapping or dynamic inheritances some static constraints exist (e.g. Max Discount). They were implemented as a Map with a "type" or "status" as a key.

Associations implementation: During a creation of Use Case Diagram we got a huge amount of use cases that include/extend each other. This fact seriously changed the way the associations were created and implemented between classes.

- For most 'many to many' associations the auxiliary (joint) class was created. So, in the case of Author – Publishing House or Book – Order such a class saves not only joint objects' references, but an additional information (e.g. Quantity);
- For complex attributes (Cost and Address) there was an idea to create separate classes and add a composition to classes that contain them. However, the Dynamic Analysis showed we don't use this information anywhere else, so there is not point to create separate tables. We decided to embed Cost and Address instead.

'Supportive' classes: Dynamic Analysis (Sequence and Activity diagram especially) helped to understand that few business processes are impossible without adding some classes:

- Cost and Address were already mentioned previously;
- Shopping Cart was added in the last moment of Class Diagram creation. When the scenario of main use case was prototyped, it turned out that a product is not added to an order directly. Then the Cart class was created. It is similar to an Order, but has no information about Address and Customer yet. Only when a Customer decides to place an order all the Cart's items will be duplicated to an Order instance.

Conclusion: well-performed Dynamic Analysis can help to identify the design patterns and the design architectural decisions that implement the requirements, increase design quality, and reduce the amount of work and cost.