National College of Ireland

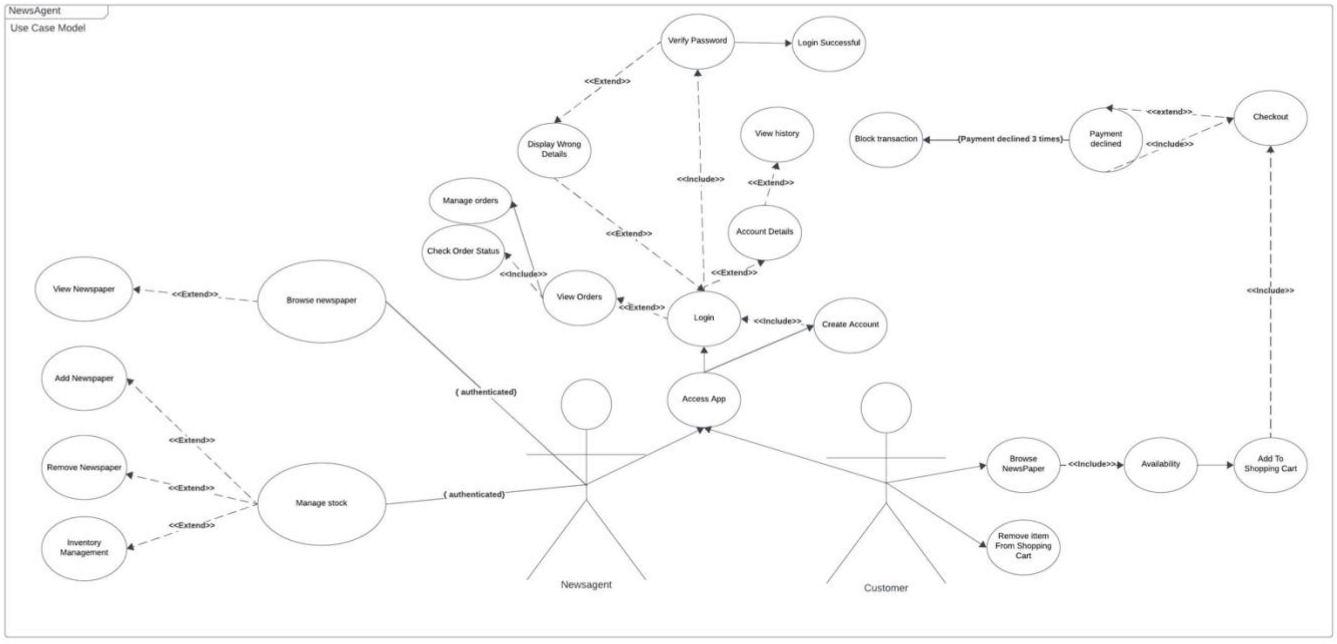
Object Oriented Software Engineering Project

Part A

Luis Martins & George Carp

Table of Contents

1. As a team, identify and describe the actors and construct a use case diagram. Describe what your use case diagram is showing (approx. 100 words)



(Please follow the following link to access the use case diagram in full-scale)

https://lucid.app/lucidchart/cbdc597d-2abd-4804-8a87-aee4753c65f3/edit?viewport_loc=2060%2C-1271%2C3111%2C1792%2C0_0&invitationId=inv_4ee0dda4-7aa2-42c2-8572-4706b3b8aff6

As a team, we drew a use case diagram for the domain assigned to us: “Newsagent”.

In this use case diagram, we have two actors: “Newsagent” & “Customer”.

This case diagram shows the relationship between different elements in a system. The actions/functions are represented using UML standard oval-shaped boxes, and relationships are represented as lines. The diagram is arranged in a tree-like structure with a central element and other elements branching off from it. The diagram represents the functionalities of an allegorical app, where the actor newsagent, which functionality consists of browsing newspapers; buying newspapers; managing stock; and managing orders; And the second actor, that represents the functionalities of a customer, which functionality includes: browsing newspapers; adding to shopping cart; removing from shopping cart; checkout; checking availability and managing their orders;

Both actors require authentication for most actions, the authentication flow includes <u>creating an account,</u> <u>logging-in;</u> and <u>verifying password.</u>

These diagrams are useful to understand the features of a newsagent and a customer, respectively, as part of a “Newsagent” domain.

2. Each team member should select one distinct use case from the use case model and describe it in detail. The use case must contain an alternate flow or exceptional flow.

1) The first use case of the app is “accessing the app”. When the user first accesses the app, they are presented with two options, either “login” or “create an account”, this being an **alternate flow.**

When the user finishes the registration, they have then to authenticate using their freshly created account, and only then they’ll have access to the application.

2) When a “Customer” tries to checkout, if the payment is unsuccessful, they are redirected back to the “Checkout” page.

If after three tries, the payment remains unsuccessful. The transaction is blocked.

This being an exceptional flow.

3. As a team, create a glossary in which all project-related terminology that requires clarification is both listed and fully defined.

Actor: A role or entity interacting with the system in use case diagrams.

Use Case: A specific function or interaction of the system with an actor.

Alternate Flow: A deviation from the main flow, representing exceptional or alternative scenarios.

Exception Flow: A sequence of steps for handling unexpected or error situations.

Association Line: A solid line connecting an actor to a use case, indicating an association.

Include Relationship: A dashed line with an arrow indicating that one use case includes another.

Extend Relationship: A dashed line with an arrow indicating that one use case extends another.

{Authenticated}: Found in certain use-case relationships. Indicates that the user must be authenticated to proceed. In other words, depicts functionality that is only available once the user logs-in.

4. As a team, create a conceptual class diagram modelling the architecture of the proposed system.
 The conceptual class diagram should demonstrate the use of three or more of the following:

attributes;

relationships;

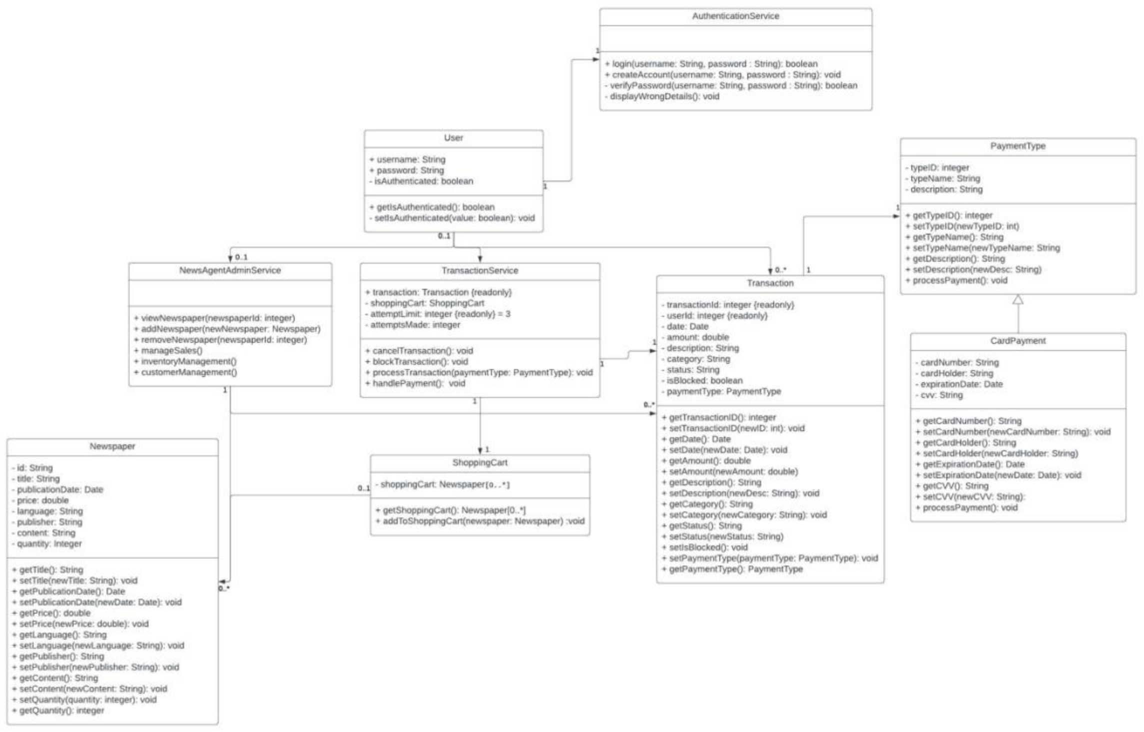
navigability;

association;

multiplicity; and

composition.

Describe what the conceptual class diagram is showing (approx. 100 words)


AuthenticationService
+ login(username: String, password : String): boolean
+ createAccount(username: String, password : String): void
- verifyPassword(username: String, password : String): boolean
- displayWrongDetails(): void
User
+ username: String
+ password: String
- isAuthenticated: boolean
+ getIsAuthenticated(): boolean
- setIsAuthenticated(value: boolean): void
PaymentType
- typeID: integer
- typeName: String
- description: String
+ getTypeID(): integer
+ setTypeID(newTypeID: int)
+ getTypeName(): String
+ setTypeName(newTypeName: String
+ getDescription(): String
+ setDescription(newDesc: String)
+ processPayment(): void
NewsAgentAdminService
+ viewNewspaper(newspaperId: integer)
+ addNewspaper(newNewspaper: Newspaper)
+ removeNewspaper(newspaperId: integer)
+ manageSales()
+ inventoryManagement()
+ customerManagement()
TransactionService
+ transaction: Transaction {readonly}
- shoppingCart: ShoppingCart
- attemptLimit: integer {readonly} = 3
- attemptsMade: integer
+ cancelTransaction(): void
+ blockTransaction(): void
+ processTransaction(paymentType: PaymentType): void
+ handlePayment(): void
Transaction
- transactionId: integer {readonly}
- userId: integer {readonly}
- date: Date
- amount: double
- description: String
- category: String
- status: String
- isBlocked: boolean
- paymentType: PaymentType
+ getTransactionID(): integer
+ setTransactionID(newID: int): void
+ getDate(): Date
+ setDate(newDate: Date): void
+ getAmount(): double
+ setAmount(newAmount: double)
+ getDescription(): String
+ setDescription(newDesc: String): void
+ getCategory(): String
+ setCategory(newCategory: String): void
+ getStatus(): String
+ setStatus(newStatus: String)
+ setIsBlocked(): void
+ setPaymentType(paymentType: PaymentType): void
+ getPaymentType(): PaymentType
CardPayment
- cardNumber: String
- cardHolder: String
- expirationDate: Date
- cvv: String
+ getCardNumber(): String
+ setCardNumber(newCardNumber: String): void
+ getCardHolder(): String
+ setCardHolder(newCardHolder: String)
+ getExpirationDate(): Date
+ setExpirationDate(newDate: Date): void
+ getCVV(): String
+ setCVV(newCVV: String):
+ processPayment(): void
Newspaper
- id: String
- title: String
- publicationDate: Date
- price: double
- language: String
- publisher: String
- content: String
- quantity: Integer
+ getTitle(): String
+ setTitle(newTitle: String): void
+ getPublicationDate(): Date
+ setPublicationDate(newDate: Date): void
+ getPrice(): double
+ setPrice(newPrice: double): void
+ getLanguage(): String
+ setLanguage(newLanguage: String): void
+ getPublisher(): String
+ setPublisher(newPublisher: String): void
+ getContent(): String
+ setContent(newContent: String): void
+ setQuantity(quantity: integer): void
+ getQuantity(): integer
ShoppingCart
- shoppingCart: Newspaper[0..*]
+ getShoppingCart(): Newspaper[0..*]
+ addToShoppingCart(newspaper: Newspaper) :void
1
1
0..1
0..1
0..*
1
1
1
1
1
1
0..*
1
1
0..1
0..*

(Please follow the following link to access the class diagram in full-scale)

https://lucid.app/lucidchart/00f93a8d-5fb5-4030-a7f5-e51461a56768/edit?view_items=rK7KRsC-Bbyz&invitationId=inv_e034d119-2f5a-4a04-bb06-7a8d085932e0

This conceptual class diagram illustrates the key classes and their relationships within our domain system.

Starting with the "User" class that contains attributes like "username," "password," and "isAuthenticated." The "User" class interacts with the "AuthenticationService" class, which handles user authentication related operations like login and account creation.

The "TransactionService" class handles all transactions, each transactionService is associated with a user, and each instance has a shoppingCart, represented by the "ShoppingCart" class, and handles one transaction, represented by the “Transaction” class, each transaction has a paymentType associated, represented by the PaymentType class.

Each TransactionService intance can only handle one transaction, per user, at a time.

For administration, there is a class called "NewsAgentAdmin", which equips newsagents with capabilities to view newspapers, manage stock (add or remove newspapers), and oversee sales, including inventory management.

There is a class called Newspaper to represent newspapers in the system, used by all classes that deal directly with newspapers.

This diagram provides a view of how the different classes that are part of our system collaborate to support user interactions and manage essential functionalities.

5. Each team member should select one of the system operations and should develop a contract for it.

Operation Name 1: addToShoppingCart(newspaper: Newspaper)

- “newspaper”: Parameter representing the newspaper the user wants to add to their cart.

Responsibilities:

- “Check User Authentication”: Ensuring that the user is authenticated. If not, call "displayWrongDetails" in the authenticationService.

- “Check Item Availability”: Verify if the selected newspaper item is available in the inventory, in practical terms, if the newspaper quantity field is bigger than 0.

- “Add Item to Cart”: If all preconditions are met, add the selected item to the user's shopping cart.

Then Update the user's shopping cart to include the new item.

Preconditions:

-The user must be logged in and have an active session.

-The "item" parameter must represent a valid newspaper available in the system's inventory.

-The item's availability in the inventory should be checked before adding it to the cart.

Postconditions:

-If the preconditions are met, the selected newspaper is successfully added to the user's shopping cart.

-The user's shopping cart is updated to include the new item.

Operation Name 2: processPayment(paymentMethod: PaymentMethod)

Responsibilities:

- “Check Shopping Cart”: Ensure that the user has items in their shopping cart. If the cart is empty, exit the method.

- “Process Payment”: Initiate the payment process using the specified payment method. This may involve interactions with external payment methods.

- “Update Order Status”: Update the status of the user's orders based on the payment result, for example in as stated on our diagram “Payment Declined”.

Preconditions:

- The user must be logged in and have an active session.

- The user's account must have sufficient funds to complete the purchase.

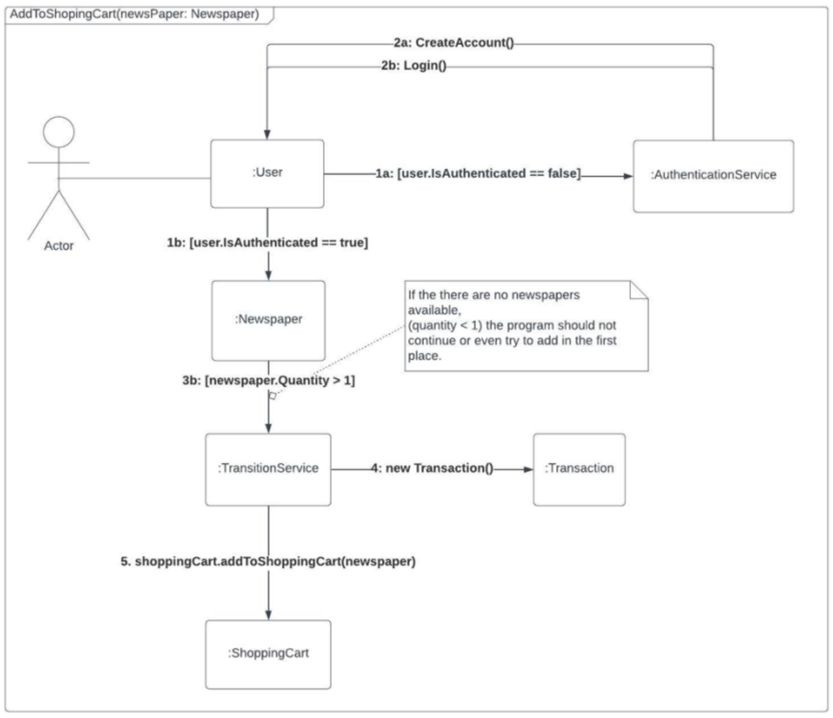
Postconditions:

- If the user has items in the shopping cart, a valid payment method is provided, the order status is updated accordingly.

- If the payment is successful the purchased items are added to the user's order history.

6. Using appropriate design patterns, each team member should create a communication diagram based on the contract they developed in task 5, including a description of what the diagram is showing (approx. 100 words).

1) "addToShoppingCart()" Operation Communication Diagram

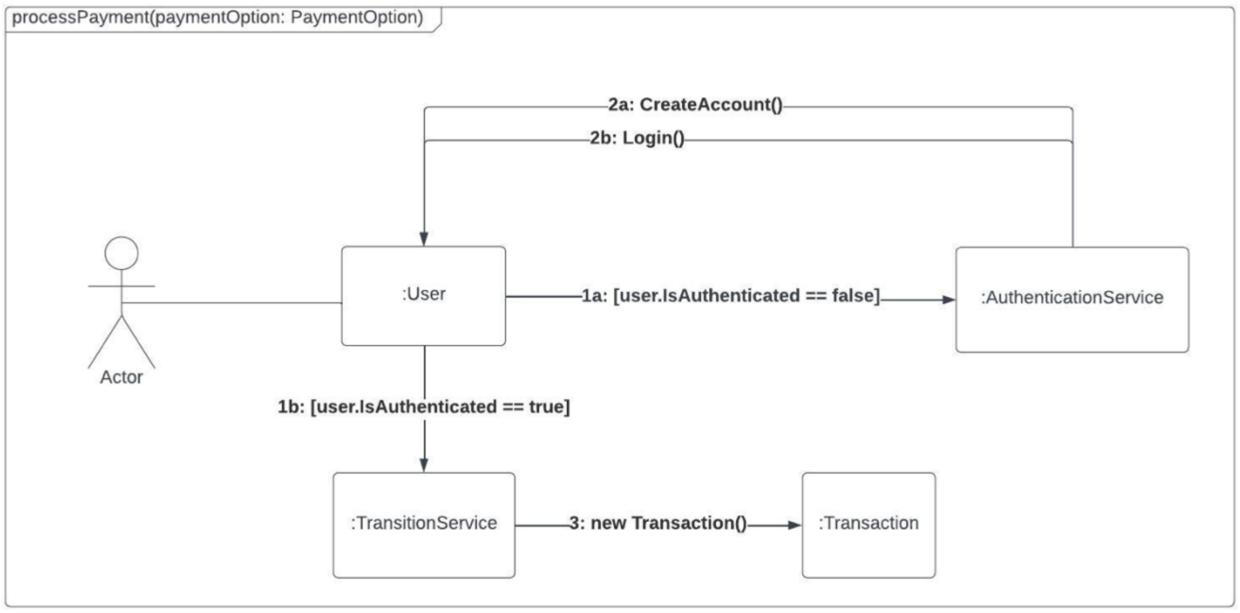

AddToShopingCart(newsPaper: Newspaper)
2a: CreateAccount()
2b: Login()
Actor
:User
1a: [user.IsAuthenticated == false]
:AuthenticationService
1b: [user.IsAuthenticated == true]
:Newspaper
If the there are no newspapers available, (quantity < 1) the program should not continue or even try to add in the first place.
3b: [newspaper.Quantity > 1]
:TransitionService
4: new Transaction()
:Transaction
5. shoppingCart.addToShoppingCart(newspaper)
:ShoppingCart

This diagram illustrates the process when a user wants to add an item to their shopping cart. It involves interactions between the user (Actor), and the following classes:

- User,
- AuthenticationService
- Newspaper
- TransactionService
- Transaction
- ShoppingCart

The user initiates the action through the user interface, leading to a series of checks and confirmations before the item is successfully added to the cart.

2) "processPayment (paymentOption: PaymentOption) " Operation Communication Diagram


processPayment(paymentOption: PaymentOption)
2a: CreateAccount()
2b: Login()
Actor
:User
1a: [user.IsAuthenticated == false]
:AuthenticationService
1b: [user.IsAuthenticated == true]
:TransitionService
3: new Transaction()
:Transaction

This diagram depicts the steps involved when a user attempts to process a payment for the items in their shopping cart. It showcases interactions between the user (Actor), and the following classes:

- User,
- AuthenticationService
- TransactionService
- Transaction

(Please follow the following link to access the diagram in full-scale)

https://lucid.app/lucidchart/5fc51065-b400-4d21-b4b2-fafb7889511f/edit?view_items=Xr.KJ2wIVYUo%2CWs.KN~Qfw2nC%2CBp.K4eCCR2WP%2CzR.KcDRPgXSu%2C_q.KuhROJpSw%2CTL9K01MiKz5-%2CPN9KUxMydSPw%2C03_KxCusYUxz%2CP09Kq9z995Gr%2C-q.KfTJiKTXI%2CWr.KryHVSQ0~%2CzR.K5E63WRfm%2CVs.KS3d.~YoD%2CP6_KWiI5W20Z%2Cq6_KZnF8l0ch%2Czp.KjKqvJm1V%2CW3_KZOGhF8ec%2CUW9KKEDzui77&invitationId=inv_a04dc492-86dd-4cf2-b9a8-ad8d7b4af5cb

7. Discuss how risk, quality and communication will be managed in your project. Provide justifications for your choices.

While specific risks for our OOP project may not be clear at the moment, risk management was an essential practice. As we progressed, we considered potential risks such as technical challenges, scope changes, or resource constraints. With a small team, risks might differ but are equally important to address. The key to risk management is to identify, assess, and develop mitigation strategies as they become apparent. Regularly monitoring the project helped us to adapt to emerging risks and evaluate the effectiveness of mitigation strategies. Effective risk management ensures that potential issues are addressed proactively, reducing the likelihood of project delays or failures.

Quality management in the project began with a comprehensive understanding of the project's requirements. As a team, this involved determining what diagrams to draw, defining classes and attributes, and starting with a small, evolving diagram.

Over time, our diagrams got bigger, reflecting our project's growth and complexity.

We’ve been diligent in maintaining and exploring up-to-date project management documentation and tooling, using resources like Google Docs, UML’s official specification, tutorials, design comments and brainstorming. This commitment to quality ensured reliability, maintainability, and alignment of our project with its intended goals.

Effective communication is vital for collaboration in a small project team. We initiated this by holding our first meeting to plan how to proceed. This initial meeting allowed us to share our opinions and explore ideas and concepts to complete the plan. As the project progressed, communication involved frequently group communication, using various tools such as in-app comments and messages on Microsoft Teams. Utilizing these tools, we had meetings to discuss our vision, share previous experiences, set goals and organize tasks, and build the project collaboratively.

In a small team, clear, succinct, and open communication is even more critical for success.

8. Describe and justify the development methodology you will follow.

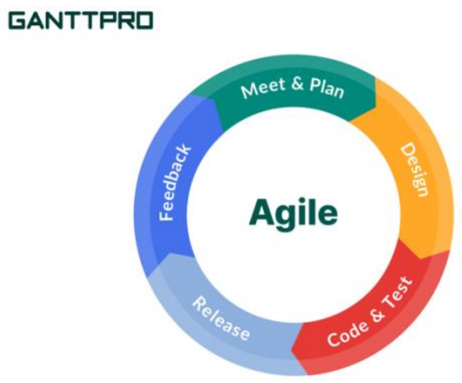


Image credits to: blog.ganttpro.com

In our project, we have implemented a customized Agile/Scrum-inspired methodology that is specifically designed to meet the requirements of our small team.

Although we do not have designated roles such as a Product Owner or Scrum Master, we have fully embraced the fundamental principles of Agile and Scrum to ensure effective and collaborative project management. The cornerstone of our work lies in our weekly sprints and daily standups.

Each day, we gather together to establish clear objectives, select tasks, and monitor our progress. This daily routine not only helps us stay focused but also fosters continuous collaboration.

During the sprint planning phase, that occurs at the start of each sprint, we collectively plan our tasks to ensure they align with the project goals. This collaborative approach allows every team member to actively contribute to the decision-making process.

At the end of each sprint (usually a week), we conduct a sprint review to assess the completed work and showcase our accomplishments with the team, where we reflect on our work and achievements, and identify areas for improvement. This enables us to verify that the project is progressing in the right direction and allows us to make any necessary adjustments.
This continuous feedback loop is crucial in refining our processes and maintaining efficiency.

Our approach, an adaptation of scrum, is specifically tailored to accommodate the dynamic nature of our small team and the daily demands of our project. By fully embracing the principles of Agile/Scrum, we ensure transparency, adaptability, and a strong focus on iterative development, decreasing the time-to-production while maintaining product quality.