CS5002NI Software Engineering

MAIN-SIT 35% Coursework

AY 2024-2025

Credit: 30

Student Name: Aadim Busubung Rai

London met ID: 23050298

College ID: np01cp4a230435

Assignment Due Date: 12th May,2025

Assignment Submission Date: 12th May ,2025

Submitted to: Rubin Thapa

Word Count: 5496

I confirm that I understand my coursework needs to be submitted online via MST Classroom under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a marks of zero will be awarded.

Table of Contents

Table of Figures

Table of tables

1. Introduction:

1.1. Introduction (Client Background including Aims and Objectives)

Apple Mart is a locally established supermarket that has steadily grown over the years, earning a strong reputation for its exceptional customer service. Strategically located in the heart of Kathmandu, the store has become a popular shopping destination. Building on its success, Apple Mart has recently expanded by opening a new branch in a different location. With this expansion, the store is committed to maintaining its high service standards and operational efficiency.

To support its growth and uphold its service excellence, Apple Mart has proposed the implementation of a dedicated Inventory Management System (IMS). The new system aims to streamline and automate key inventory-related operations, including tracking sales records, managing supply demands to manufacturers, and providing real-time stock updates for both the administrative team and customers. This initiative is vital to eliminating inefficiencies that plagued the previous system and transitioning towards a more financially and operationally sustainable model.

For this endeavor, Apple Mart has partnered with Global Tech Cooperation, entrusting them with the development and deployment of the IMS solution. After assessing the store's operational challenges and business requirements, the team at Global Tech concluded that a robust IMS is essential for Apple Mart to achieve better warehouse management, cost control, and service consistency across all branches.

To facilitate clear communication among stakeholders and ensure alignment within the development team, various diagrams and modeling tools have been utilized as part of the planning and analysis process. Object-oriented Analysis and Design (OOAD) principles have been applied to create structured and modular software architecture. Additionally, the Unified Modeling Language (UML) has been used to visually represent the system's structure, behaviors, and interactions. UML offers a standardized and widely accepted framework for specifying, visualizing, constructing, and documenting the essential components of both software and business systems.

Through this report, the proposed IMS will be presented using appropriate graphical representations, business cases, and models. The ultimate objective is to deliver a system that

is not only technically sound and user-friendly but also aligned with Apple Mart's operational goals and stakeholder expectations.

2. Work-breakdown Structure:

A Work Breakdown Structure (WBS) is a project management tool that breaks down a project into smaller, more manageable parts. It helps organize and define the total scope of a project by decomposing it into hierarchical levels of work elements. At the first part I used prototype as my work break-down structure.

2.1. Previous WBS

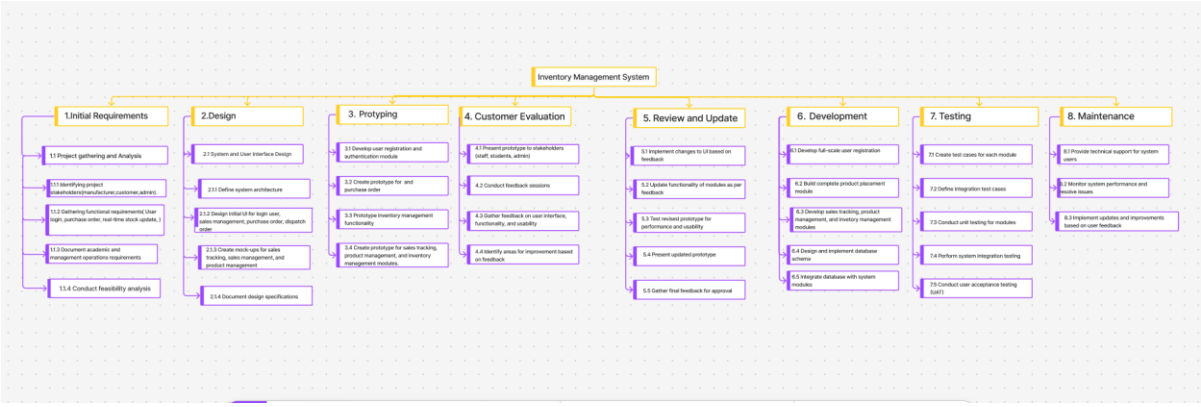

Figure 1: Work-breakdown Structure

2.2. Work breakdown Structure

Then I changed to kanban to make it more agile, allowing flexibility without defining specific roles perfect for a project like ours.

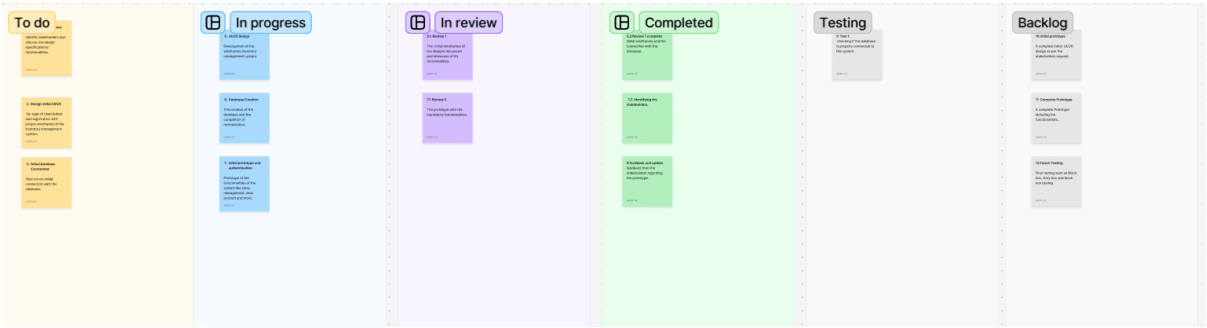

Figure 2 Kanban WBS

3. Gantt chart

A Gantt Chart is a type of bar chart used in project management to visually represent a project schedule. It shows the start and end dates of tasks, their durations, dependencies, and the overall project timeline all in a linear, time-based view. Now with the change in WBS the changes should be made to the Gannt chart too.

Previous Gantt Chart

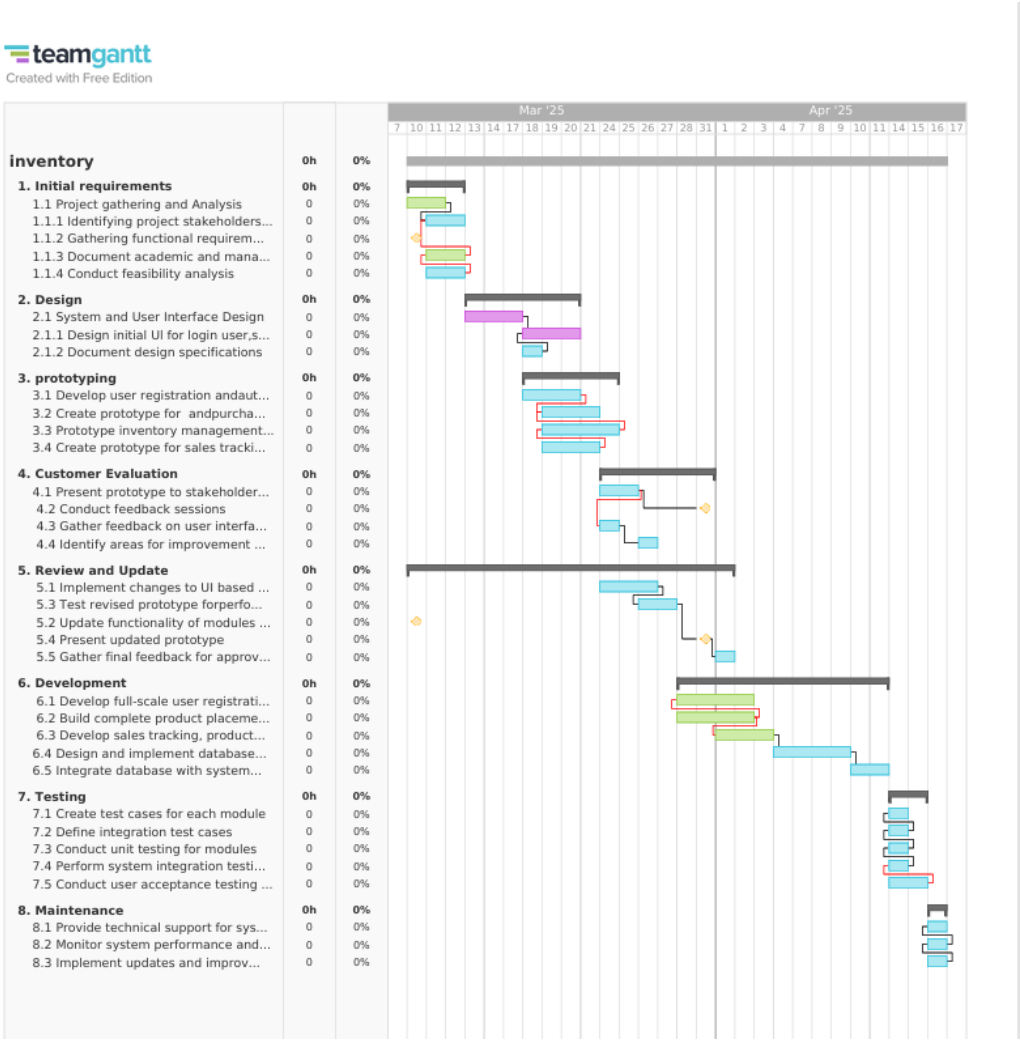

Figure 3: Gantt Chart

Current Gantt Chart

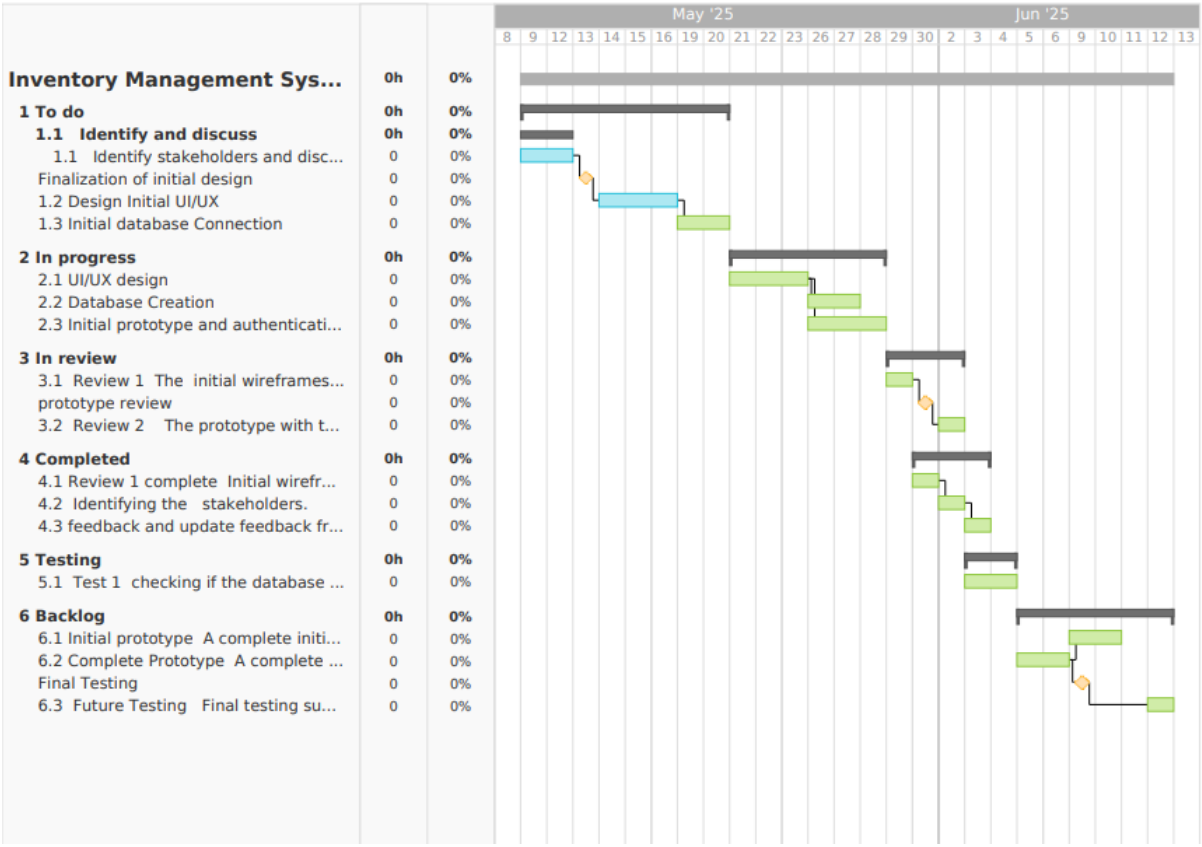

Figure 4: Gannt chart

4. Use Case Diagram

What is a use-case diagram?

The use case diagram allows us to describe the possible usage scenario. Use case diagram is a collection of use cases that a system is developed for. It expresses what a system should do but does not directly address any realization details such as data structures, algorithms, etc (Martina Seidl, 2015). With the help of the use-case diagram we take three things from it:

- What is being described or what the use case is about? – The system
- Who are the major people who interact within the system? - The actors
- And finally, what are the actions the actors can perform? – The use cases

Now to denote the system, actors and the use cases we can help with the notations which are used to represent them.

For representing the use cases which will further be developed into functionalities an oval shape is used and for naming it verb+noun format can be followed. Whereas as for the actors, there are primary actors and secondary actors denoted by a stick man and for the system we put the name of the system at the middle of a container where outside of the boundary we put actors and generally connect them with the appropriate use-cases.

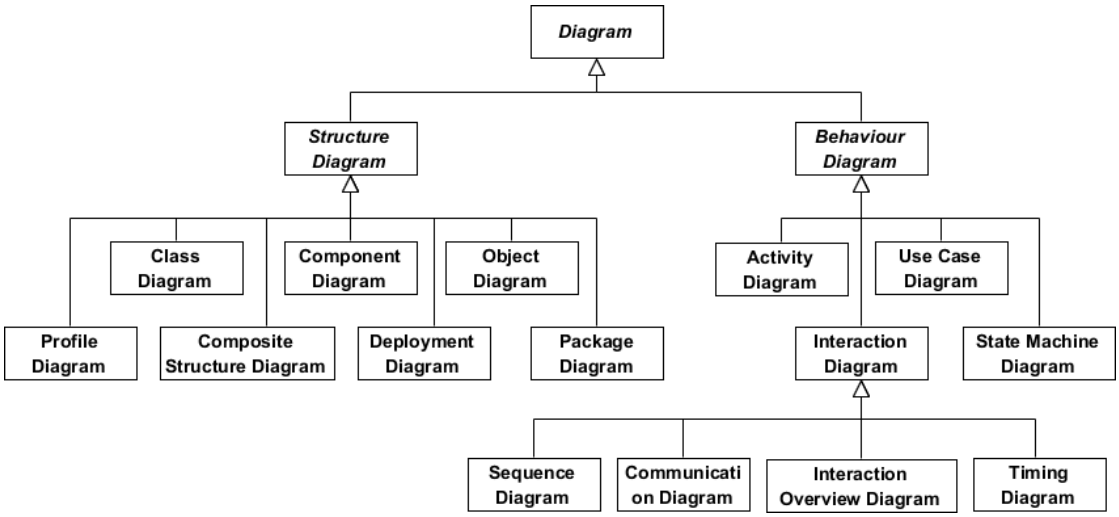

Figure 5: UML Tree

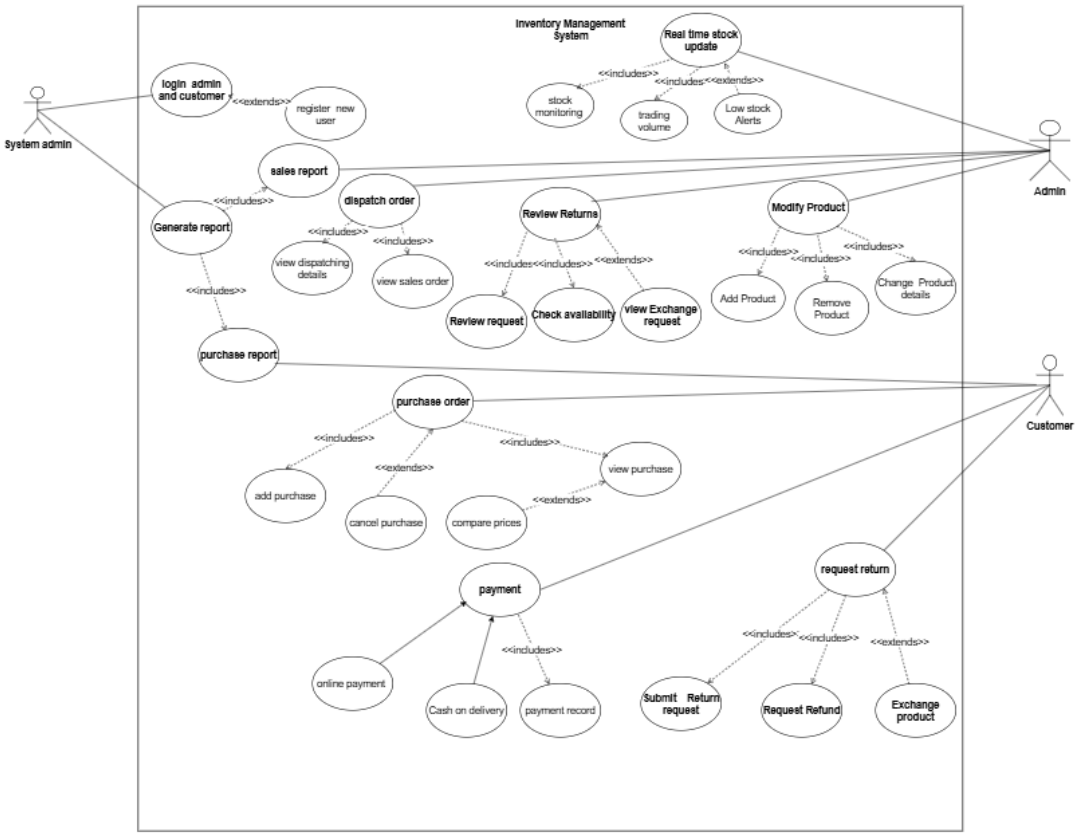

Figure 6 Use case diagram

4.1. High-level Description

In this part of the use-case we define the use-cases and explain it in a more detailed way. In this part, we usually put names, actors where one of them is entitled as an initiator (the initiator is someone from where the use case starts or someone who initiates it) and finally a description about how the use case works.

1) Login admin and customers:
   - Name: logs in admin and customers
   - Actor(s): System Admin (Initiator), Admin, customer
   - Description: This case gives access to the system as admin or customer and if the user is new to the system, it asks you to register and gives access to the system. The system after logging in takes you to respective interfaces.

2) Modify Product:
   - Actor(s): Admin (Initiator)
   - Description: This case gives the admin the option to add a new product, view product details and its history including the supplier details.

3) Generate report
   - Actor(s): System Admin (Initiator), Admin, customer
   - Description: This case gives the admin and the customer the option to generate sales and purchase report respectively.

4) Real time stock updates
   - Actor(s): Admin (Initiator)
   - Description: This case gives the admin the option to view the stock details and monitor it. It also includes facilities like trading volume and an optional automatic alert when the stocks are low.

5) Purchase order
   - Actor(s): customer (Initiator)
   - Description: In this case, customers initiate an order where they can perform tasks such as view purchase, add purchase and cancel purchase.

6) Payment

- Actor(s): customer (Initiator)
- Description: In this use-case, after a customer initiates an order, this use-case comes to play, giving you some options like online payment, payment history details.

4.2. Expanded level description:

Name: Modify product

- Actor(s): admin
- Purpose: managing the addition of new products and viewing product details, comparing prices.
- Overview: This case gives the admin the option to add a new product, view product details and its history including the supplier details.
- Type: Primary

Table 1 Expanded level of add product

Actor Action	System Response
1) After accessing the system, the user checks if any addition of product is needed and performs action accordingly.	2) Add the new product to the system and keep the record.
3)If the user wants to view a product, it allows the user to do so.	3) Displays the product and show its stored details.
5) If the user wants to view its history (which is optional) it can display it too.	6)displays the history of the product if necessary.

Alternative Course:

So, when the actor's actions go off course of the expanded level description, there should be an alternative course of it too.

Line 5: if the user does not want to view its history, then it goes back to line 1 again providing different options for the user to perform.

Name: Generate report

- Actor(s): system admin, customer, admin
- Purpose: managing the generation of new products and sales report.
- Overview: this case gives the admin and the customer the option to generate sales and purchase reports respectively.

- Type: Primary, secondary

Table 2 Expanded level description: Generate report

Actor Action	System Response
1) After performing some actions, the secondary actors (admin and users) demand the primary actor for the report generation. 2) The secondary actors demand the sales report and purchase report separately. 4) Primary actors send the generated reports to the secondary actors separately according to their preference.	3)The system gives all the stored data in its database accordingly.

Alternative Course:

Line 2: if the secondary actors don't demand the sales report or the purchase report then the system just gives it to the system admin.

5. Communication Diagram:

A communication diagram also known as collaboration diagram is an extension of an object diagram that shows the objects along with the messages that travel from one to another. In addition to the associations among objects, communication diagram shows the messages the objects send each other (Paradigm, 2016). It is used to clarify the roles and identify the controller object.

Generally, in communication diagrams, there is a showcase of proper flow of message between the objects and with proper indexing as well. A controlled object is identified and then the messages between the objects are showcased through an arrow. The only difference between a sequence diagram and a communication diagram is that sequence is used or let's say it is made according to time.

Key Elements of a Communication Diagram:

1) **Objects:** it is the collective possible class names of the use case. It is denoted by a rectangular box.
2) **Actors:** the actors are the ones who invokes the interaction and are denoted by a stick man figure.
3) **Links:** the links are the ones that connect actors and objects. they are generally denoted by a straight-line carrying messages as means of interactions.
4) **Messages:** the messages are the means of interactions that have their own unique numbering.

Now for our communication diagram, the use case I've chosen is generate report. For generate the report lets write down the possible classes for it:

1) System Admin- the actor who initiates by entering the type of report they want to get access to.
2) Sales database- If they want to get access to sales report then certain method is called after the approval of the request and the report is accessed by the system admin.
3) Purchase database- If they want to get access to Purchase report then certain method is called after the approval of the request and the report is accessed by the system admin.
4) Generate report- This can be further made into a controller object.

5) Create report- finally after the system admin checks the report, they can create report and send it to the individual via email.
6) Error- In case of any error during the generation of report, the error object shows the error message.

Now let's represent them by putting the domain classes in the object diagram:

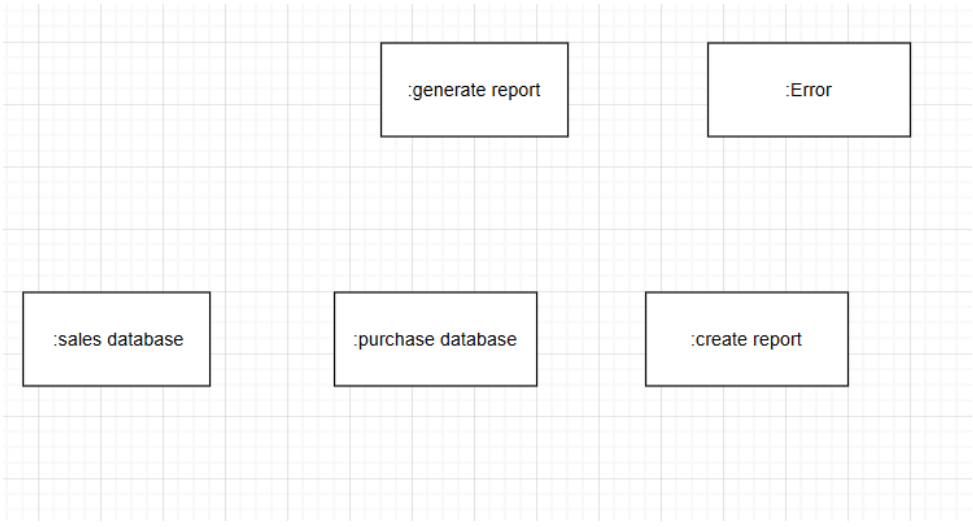

Figure 7: Object representation of domain classes

Next step is to add a boundary object that will have the same name as the use case with addition of UI to its name.

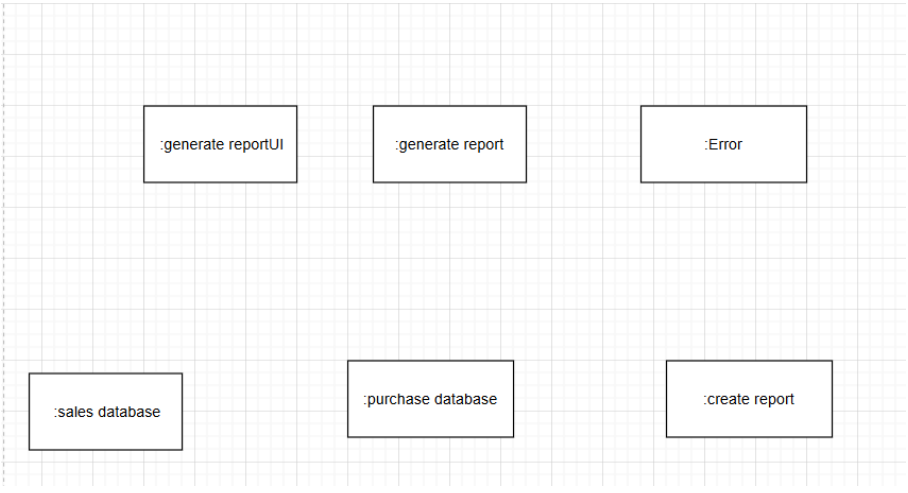

Figure 8 addition of boundary object

Now let's start by adding actors needed for this use case with proper associations to link for upcoming interactions. For the diagram to connect the interactions, I've selected the generate report as the controller object.

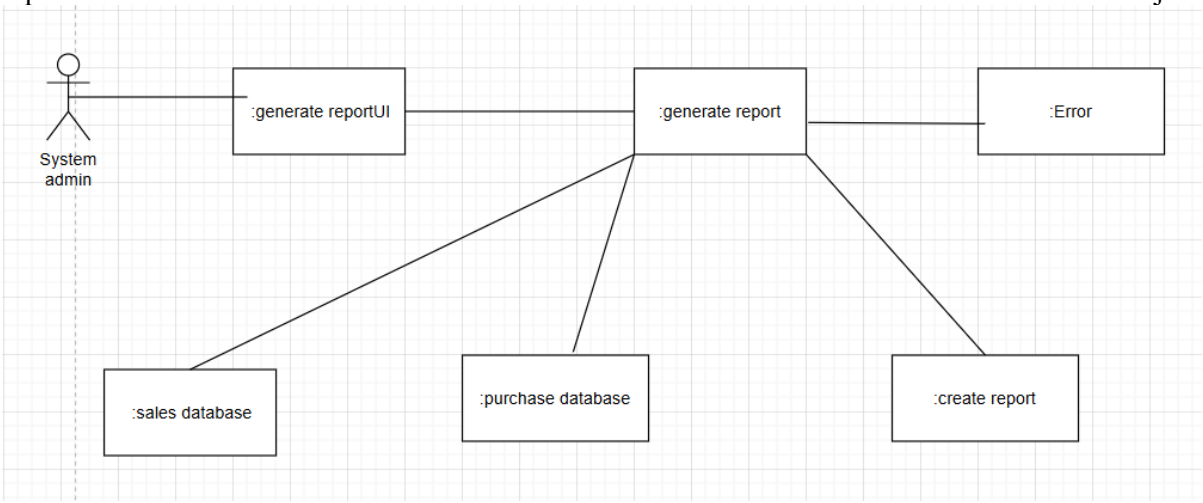

Figure 9

For the next part of the process, we're going to put interactions that are needed for the objects in the diagram.

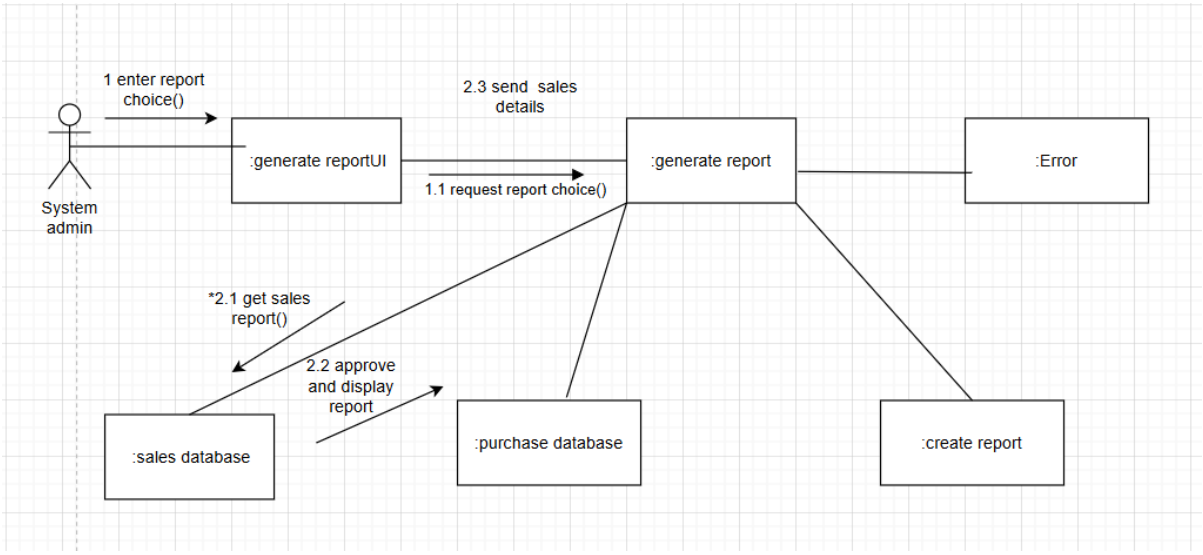

Figure 10 Putting messages

As an initial step, the system admin as per the request of the secondary actors enters the report choice. For this instance, let's take sales report as a choice, when the system admin enters the choice which is the sales report the get sales report method is called. As a reply, the choice request is approved and for now only the details or the report is viewable to the system admin only. Upon reviewing the report, the system admin asks to create a report. The creation method gets called and after the creation send the report to the designated individual via email. The same method is applicable for purchase reports too.

And if any error occurs then the error object shows the error message which occurred during the creation of the report.

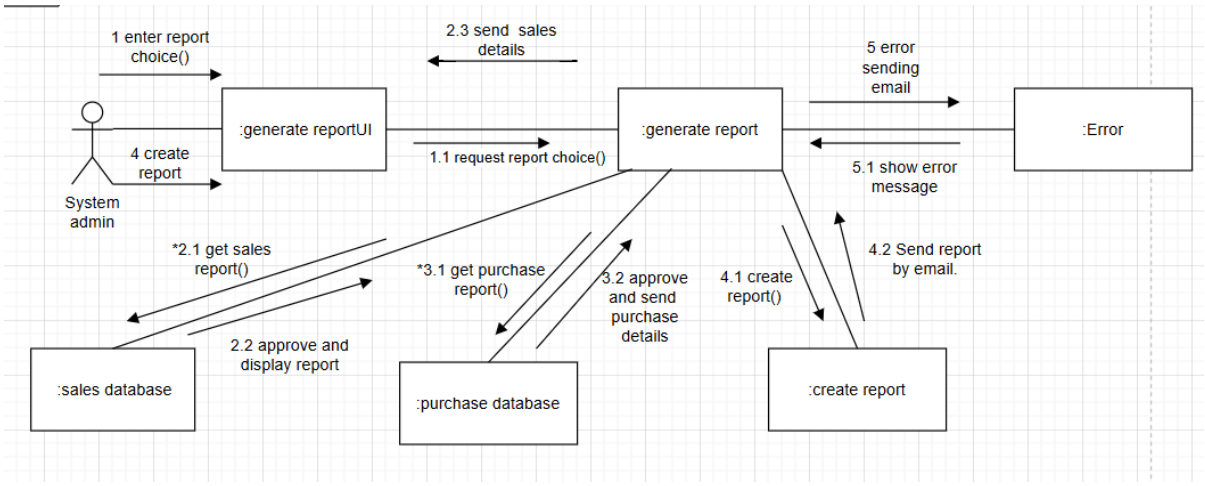

Figure 11 messages

And for the final part we can add a suitable frame for the communication diagram with a generated report written at the top left space.

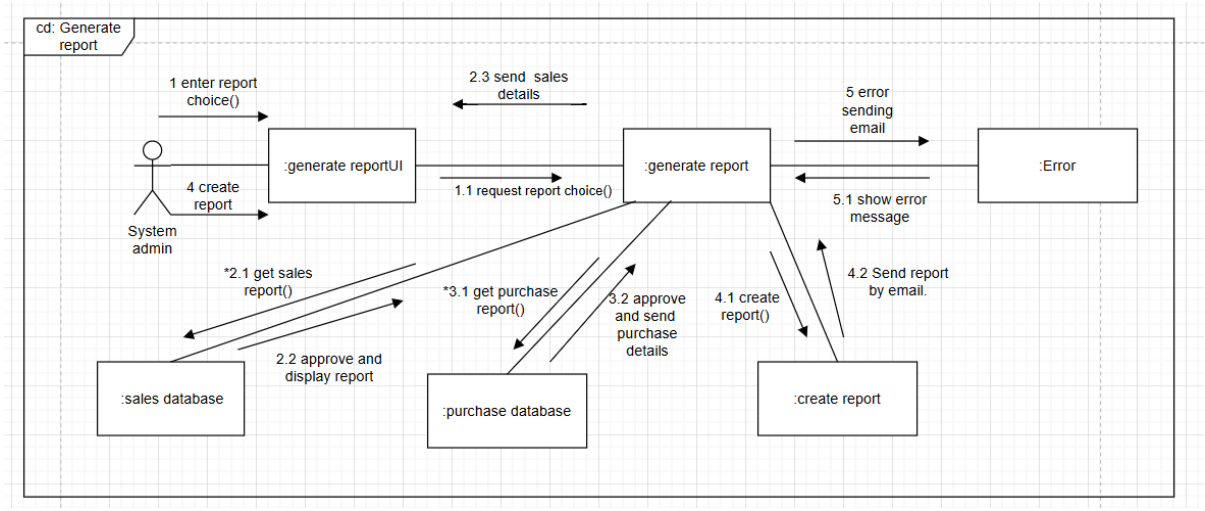

Figure 12 Final communication diagram

6. Activity Diagram:

An Activity Diagram is a type of UML (Unified Modeling Language) diagram used to model the workflow or business process of a system. It visually represents the sequence of activities, decision points, and parallel processes involved in a particular function or use case.

Key Elements of an Activity Diagram:

1. **Start Node** (Black Circle) – Represents the beginning of the workflow.
2. **Activity (Action)** (Rounded Rectangle) – Represents a task or process performed in the system.
3. **Decision Node** (Diamond) – Represents a branching point where a decision must be made (e.g., "Payment Successful?" Yes/No).
4. **Merge Node** – Combines multiple paths back into a single flow.
5. **Fork Node** – Represents parallel execution (e.g., processing payment and updating inventory at the same time).
6. **Join Node** – Synchronizes parallel activities back into one flow.
7. **Arrows (Flowlines)** – Show the sequence of execution.
8. **End Node** (Black Circle with a Border) – Marks the completion of the process.
9. **Actions –** are the processes that are either carried out or kept as option and is generally represented with the help of a rectangular box.

The activity diagram that is in the picture below is of the payment use-case. At first, the diagram is specifically designed according to the swim lane design where on one side customer is kept and admin is kept on the other side. Using the fork nodes, decision nodes and many more is used to make the diagrams more understandable.

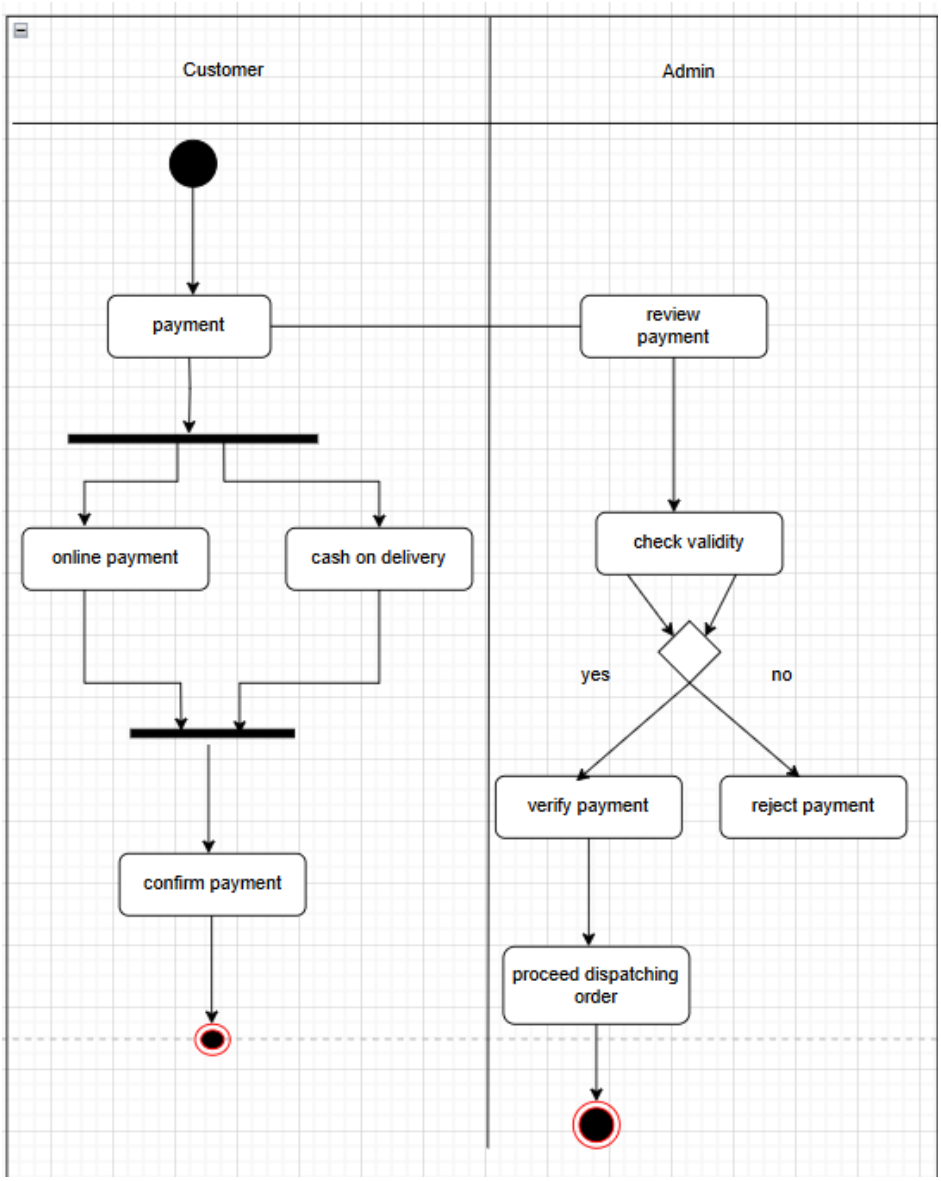

Figure 13

In the diagram above, the activity diagram is of payment feature. So, let's go through how the payment feature works. First the start node is represented as the initiation of the activity diagram of payment method. Then at the admin lane the review payment is processed by putting through decision node to check its validity. And if the payment looks and fills the requirements to be dispatched then the next process is to proceed by dispatching it. While in the customer lane, the option of payment is shown, and the customer lane is concluded by confirming the payment.

7. Agile Methodology:

Agile in simple words, when we generally understand or perceive as something that is quickly able to move with ease. Agile methodology is an iterative and incremental approach to software development that emphasizes flexibility, collaboration, customer feedback, and rapid delivery. It is a system of values and principles which leads to specific niches of project management (Clark, 2019). There are some principles that an agile software development follows which was developed unknowingly or let's say coincidentally at a ski resort called snowbird resort. The manifesto then was called Agile 'Software Development' Manifesto. The agile works on the 4 core values and 12 principles of the manifesto mentioned earlier (beedle, 2001).

4 Values of Agile

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change over following a plan.

12 Principles of Agile

1. Satisfy the customer through early and continuous delivery of valuable software.
2. Welcome and harness changes for the customer's competitive advantage, even late in development.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference for shorter
timescales.
4. Have daily collaboration between businesspeople and developers throughout the project.
5. Build projects around motivated individuals. Create the environment and support developers need and trust them to get the job done.
6. Prioritize face-to-face conversation as the most efficient and effective method of conveying information to and within a development team.
7. Measure progress by the amount of working software completed.
8. Maintain a constant and sustainable pace of development indefinitely.
9. Enhance agility through continuous attention to technical excellence and good design.
10. Keep it simple. Simplicity, the art of maximizing the amount of work not done is essential.

11. Recognize that the best architecture, requirements, and designs emerge from self-organizing teams.
12. Regularly reflect and adapt behavior for continual improvement.

Some of the Popular Agile Frameworks are mentioned below:

- **Scrum:** Time-boxed sprints with roles (Scrum Master, Product Owner, Team), ceremonies (Daily Stand-up, Sprint Planning, Review, Retrospective)
- **Kanban:** Visual workflow management, continuous delivery without fixed-length iterations
- **XP** (Extreme Programming), Safe, Lean, etc.
- **DDSM** (Dynamic System Development Method)

For the agile methodology I've chosen kanban methodology. Kanban is a Japanese term that translates to "signboard" or "billboard." It refers to a visual management system that helps teams visualize their work, manage their workflow, and optimize efficiency (Martins, 2025). Kanban is a So now let's talk about the benefits of the usage of kanban:

1) Flexibility: In kanban, we can re-order tasks while the scrum limits this. This makes kanban easier to use and adapts with ease.
2) Efficiency and continuous delivery- It allows us to work in a flow state manner continuously instead of waiting for certain tasks to end.
3) Visualization- From a project manager aspect and worker's perspective the visualization of the workflow makes it easier and more flexible as mentioned before (University, 2020).

Use Kanban if:

- Your team needs to handle ongoing, interrupt-driven work
- You value workflow visibility and flexible prioritization
- You want to improve lead time and throughput
- You prefer continuous improvement without rigid roles or ceremonies.

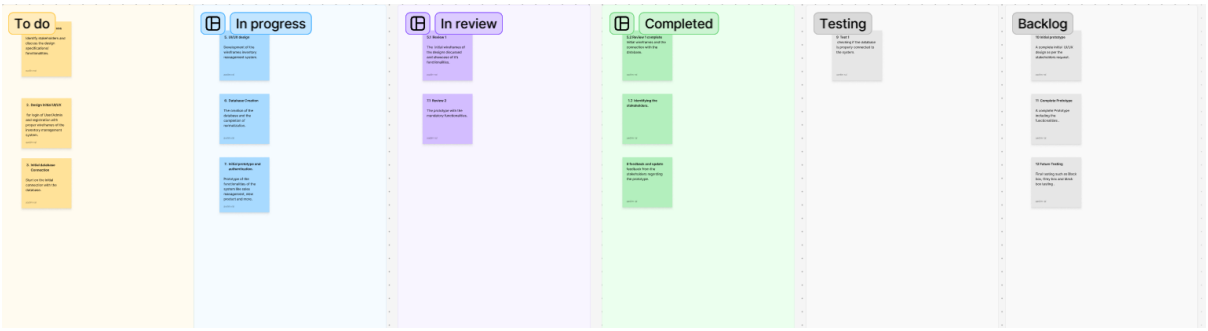

Figure 14 Agile Kanban

8. Class diagram

A Class Diagram is a type of UML (Unified Modeling Language) diagram that visually represents the static structure of a system by showing its classes, attributes, methods (operations), and the relationships between the classes.

Table 3 Use cases and Domain classes

Use Case	**Domain Classes**
Login Admin and Customer	User, AuthenticationService, Admin, Customer
Register New User	RegistrationService, User, Profile
Generate Report	ReportGenerator, Report, Admin, SystemData
Purchase Report	Report, PurchaseOrder, PurchaseDetail
Sales Report	Report, SalesOrder, SalesRecord
Dispatch Order	Order, Inventory, DispatchService, SalesOrder
View Dispatching Details	DispatchDetail, Inventory
View Sales Order	SalesOrder, Product
Review Request	Request, RequestService, Admin
Check Availability	Inventory, Product, StockChecker
Review Returns	ReturnRequest, ReturnService, Admin
View Exchange Request	ExchangeRequest, ReturnService
Modify Product	Product, ProductService, Admin
Add Product	Product, ProductService
Remove Product	Product, ProductService
Change Product Details	Product, ProductService
Purchase Order	PurchaseOrder, Customer, Product, OrderItem, Inventory
Add Purchase	PurchaseOrder, Product, Inventory
Cancel Purchase	PurchaseOrder, CancellationService
Compare Prices	Product, PricingService
View Purchase	PurchaseOrder, Product, Customer
Payment	Payment, PaymentGateway, Order, Customer
Online Payment	OnlinePayment, PaymentGateway

Cash on Delivery	CashPayment, Delivery
Payment Record	PaymentHistory, TransactionRecord
Request Return	ReturnRequest, ReturnService, Customer
Submit Return Request	ReturnRequest, ReturnService
Request Refund	Refund, ReturnRequest, Payment
Exchange Product	ExchangeRequest, Product, ReturnService
Real-Time Stock Update	Inventory, StockUpdateService, Product
Stock Monitoring	Inventory, StockAlert
Trading Volume	Product, SalesRecord
Low Stock Alerts	Inventory, AlertService

Here is the full Class diagram:

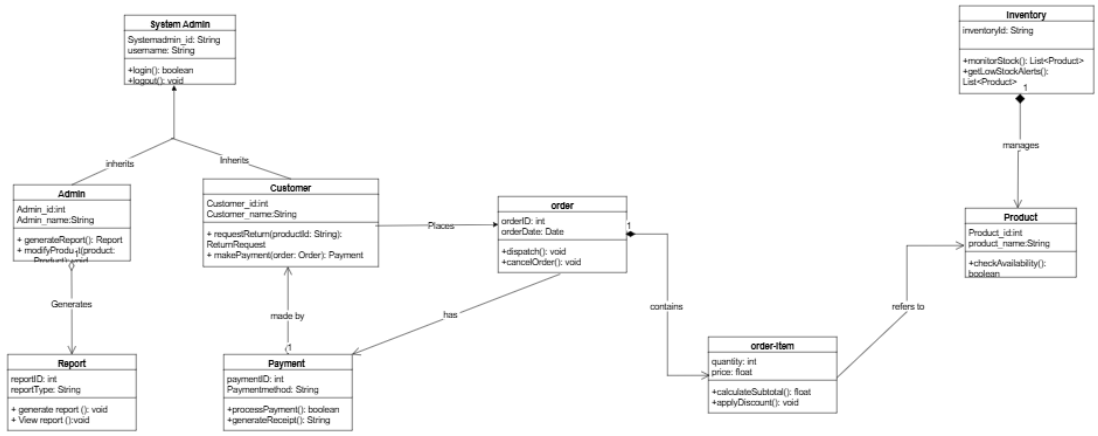

Figure 15 class diagram

9. Further development:

As for the further development of the inventory management system, we must be precognitive and state things such as the architectural choice, what type of architectural design, design pattern, development plans including things like the languages used in the development and finally testing to be carried out and our maintenance plan.

9.1. Architecture choice

Architectural design in software engineering is the process of defining a structured solution that meets all the technical and operational requirements of a project while optimizing common quality attributes such as performance, security, and manageability (Engineering, n.d.). Architecture design is basically the blueprint or the road map for the developers to get a better grasp of the technical functionalities to be met in the making of the construction of the software.

There are some architectural choices that plays a pivotal role in making the decisions for the overall behavior and the structure of the system. Some of them are:

1) Scalability
2) Architectural style/ pattern
3) Security Measures
4) Data management
5) Performance optimization and error handling
6) Maintainability and compliance with today's industry.

Architecture design pattern types:

1) Layered architecture: This type of architecture organizes into layers for some roles or functionalities.
2) Client-Server architecture: It divides Client and server.
3) Microservices Architecture: It Structures the application as a collection of loosely coupled, independently deployable services.
4) Event-Driven Architecture: Based on the production, detection, and consumption of events.

Event-Driven Architecture

What is Event-Driven Architecture?

Event-driven architecture is a method of developing enterprise IT systems that allows information to flow in real time between applications, microservices, and connected devices as events occur throughout the business. The event is the flow of a data point, and the collection of events is called stream. It is the continuous delivery of data points. (Seetharamugn, 2023)

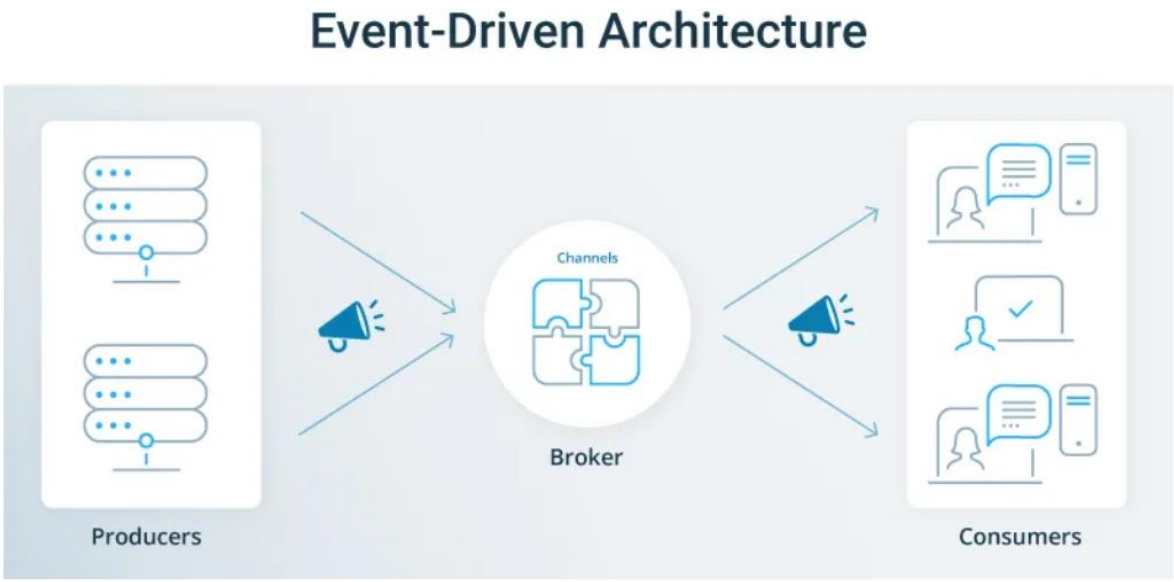

Figure 16 (Seetharamugn, 2023)

Advantages of using Event-driven Architecture:

The number one advantage is that when an event is not being tracked down or let's say that an event is changed and if the team is not able to receive proper information about it then it may cause some havoc. It can cause a hassle for the system and the people. It is highly decoupled and flexible, making it suitable for the users. It is also scalable and suitable for real-time applications. To add another service, you can just have it subscribe to an event and have it generate new events of its own; there's no impact on existing services. Everything happens in a split second and waits for none.

This type of architecture is perfectly aligned with our requirements such as the distributions, excellent services and especially since we have a real-time stock update feature businesses like us can thrive while using this design as a backbone of it.

9.2. Design Pattern

Christopher Alexander's architectural concepts laid the foundation for design patterns, which were later adapted for software by the "Gang of Four." Their seminal book, "Design Patterns: Elements of Usable Objects," introduced 23 design patterns to the software development community, marking a turning point in how developers approached design challenges. According to it, design patterns are the efficient design patterns that help with recurring problems with software designs.

Now let's look at the different types of design patterns:

1.Creation Pattern

2.Behavioral pattern

3. Structure Pattern

MVC pattern

Using an organized code structure, the MVC design pattern divides an application into 3 interconnected components. The Model manages the application's data and business logic, including information retrieval, storage, and processing. The View is in control of giving the user a legible representation of the data. By processing user input, updating the Model in response to actions, and refreshing the View appropriately, the Controller serves as a link between the Model and the View. Projects may be developed, maintained, and scaled more easily because of this division of responsibilities, which also enables team members to work independently on data and logic. Frameworks like Django, Ruby on Rails, and Spring make extensive use of MVC, to help developers in creating reliable and modular applications quickly (Seetharamugn, 2023).

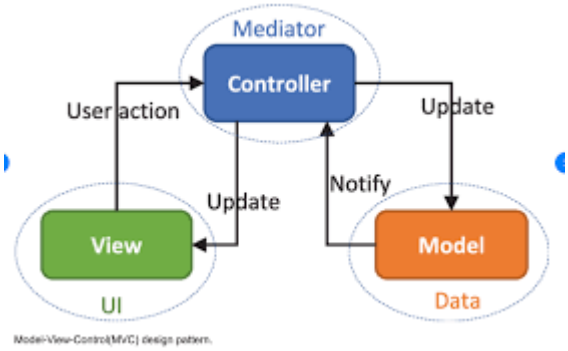

Figure 17

9.3. Development

Now for the development part of the system let's start by stating the languages and the tools we're going to use for the system. For this part we must work according to the SRS. Things stated such as functional requirements and non-functional requirements are made possible in this part. Let's look at some of the tools that will help us achieve this goal.

Tools to be used:

1) Trello

Trello is a website which is used to make kanban boards easily with the already existing modules available on the site. It is used for the visualization part of the system activities.

Figure 18 Trello

2) Zoom

For the regular reviewing and updating part with the team for easy convenience, zoom an online meeting holder where meetings can be held freely, is used in this.

Figure 19 zoom

3) Draw.io

A free website-based software that is an intriguing place for creating diagrams such as diagrams related to the UML tree.

Figure 20 draw.io

4) Figma

Figma is a web based collaborative site used for designing part especially during UI/UX which would be helpful for the front-end team.

Figure 21 Figma

5) Visual Studio Code

VS Code is a code editor to build and debug apps. With the Flutter extension installed, you can compile, deploy, and debug Flutter apps.

Figure 22 vs code

6)Eclipse

Eclipse is one of the most popular IDE which is an integrated development environment used in computer programming. It contains a base workspace and an extensible plug-in system for customizing the environment.

Figure 23 Eclipse

7) GitHub

GitHub is a proprietary developer platform which provides access control, bug tracking, software feature requests and sharing the code they have created.

Figure 24 GitHub

Languages:

For the languages, we must choose something that is compatible and aligns with the skills of the development team. JavaScript, HTML5, CSS3, PHP, and Java will be used for the development ensuring familiarity among the developers.

For the front-end side Html, CSS would be used where java script / java can be used to give functionality to the buttons and the software. For backend, PHP can be used for doing things like creating, read, writing and delete.

9.4. Priority order of the features:

For the priority order let's use MoSCoW prioritization.

Must have

- Users must be able to log in and a different interface should appear after it (customer and admin)
- Customers must be able to make payments with view of the product, buy and place order.
- The admin must be able to make changes to the system such as modifying pages, adding products, sales tracking and so on.

Should have

- The customer should be able to request a generating report.
- The admin should be able to review the return requests and report requests.
- Notifications for both admin and customer.

Could have

- A live chat could have made the system easier to use for individuals who are not familiar with using technology.
- A video containing how the system works and what they provide could be interactive.

Won't have

- The system won't have pop-up ads for now.
- Automatically pausing of the videos
- High level navigation bar and slides.

9.5. Testing Plan

Testing is one of the most important steps where we can identify the errors and the fulfillments will be checked.

Let’s look at some of the testing types:

1) Unit testing
2) Integration testing
3) System testing
4) Regression testing
5) Smoke testing

After careful consideration, System Testing appears to be the most suitable approach for our methodology. System testing is a high-level software testing technique in which the entire application is tested as a complete system. It ensures that the integrated software meets both the functional and non-functional requirements as specified in the project documentation.

This type of testing is typically performed after unit testing and integration testing. While unit tests validate individual components and integration tests check interactions between modules, system testing simulates real-world user scenarios, examining the software's end-to-end behavior in a production-like environment. This makes it crucial for validating the overall functionality and stability of the application before it goes live.

The primary objective of system testing is to determine whether all parts of the system function together seamlessly and deliver the expected output. It helps detect issues related to:

- System architecture and design flaws,
- Configuration errors, and
- Inconsistencies across integrated modules.

System testing encompasses both:

- Functional testing, which verifies features such as data input, processing logic, outputs, and user interactions, and
- Non-functional testing, which evaluates aspects like performance, scalability, security, compatibility, and usability.

Some common types of system tests include:

- End-to-End Testing – Ensuring complete user workflows behave as intended.
- Load and Stress Testing – Measuring how the system performs under normal and extreme conditions.
- Security Testing – Identifying vulnerabilities that could be exploited by malicious users.
- Regression Testing – Making sure that recent code changes haven't adversely affected existing functionality.

By conducting thorough system testing, we can significantly reduce the risk of defects reaching production, enhance user satisfaction, and ensure that the software system aligns with business and user expectations.

9.6. Maintenance plan:

1. Daily Maintenance

- Check system uptime and performance.
- Monitor server health (CPU, memory, disk usage).
- Verify successful backups.
- Scan for unauthorized login attempts or suspicious activity.

2. Weekly Maintenance

- Review and apply software/OS patches.
- Check web applications for broken links or errors.
- Test core features (login, forms, APIs).
- Review analytics and log files for anomalies.

3. Monthly Maintenance

- Update dependencies and libraries.
- Test application on multiple browsers/devices.
- Clean up unused files or temporary data.
- Optimize database performance (indexing, cleanup).

4. Quarterly Maintenance

- Perform full backup and test recovery process.
- Review security policies and access controls.
- Conduct codebase and UX/UI audits.
- Plan upcoming feature updates or deprecations.

5. Annual Maintenance

- Renew domain, SSL certificates, and licenses.
- Conduct penetration testing and security audits.
- Review hosting plan and resource usage.
- Evaluate performance trends and scalability needs

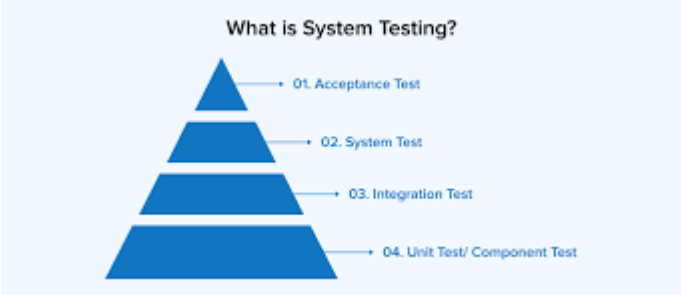

Figure 25 System testing (TavtaSoft)

10. Prototype:

A prototype is an early, simplified version of a system or product that is built to:

- Visualize ideas
- Test functionality
- Gather feedback from users and stakeholders
- Reduce risk and clarify requirements before full-scale development

Here are some of the prototypes of the system itself:

Figure 26 Login

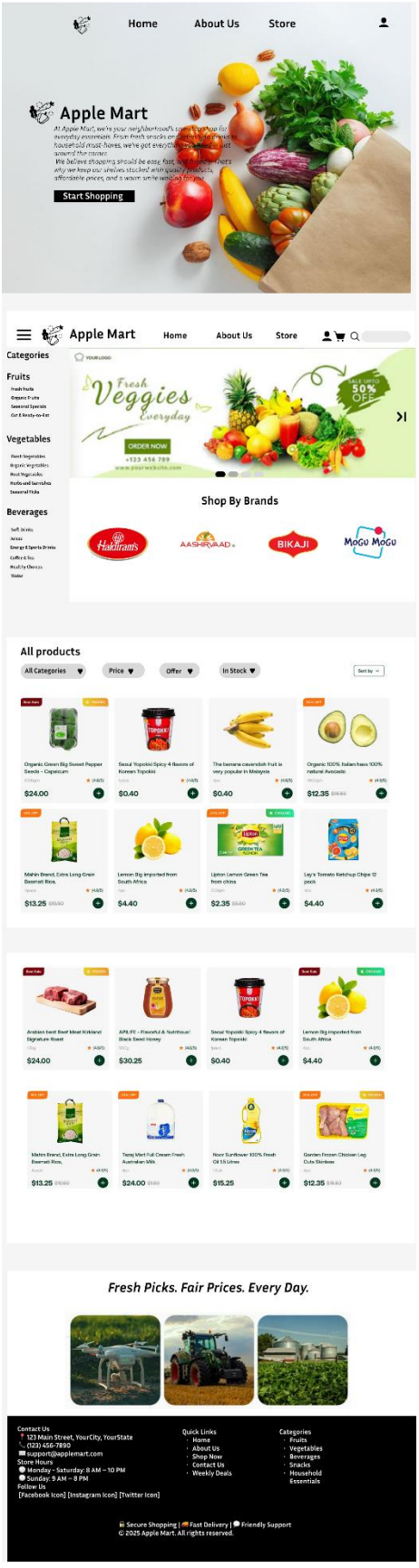

Figure 27 User interface

Figure 28: protoype 2

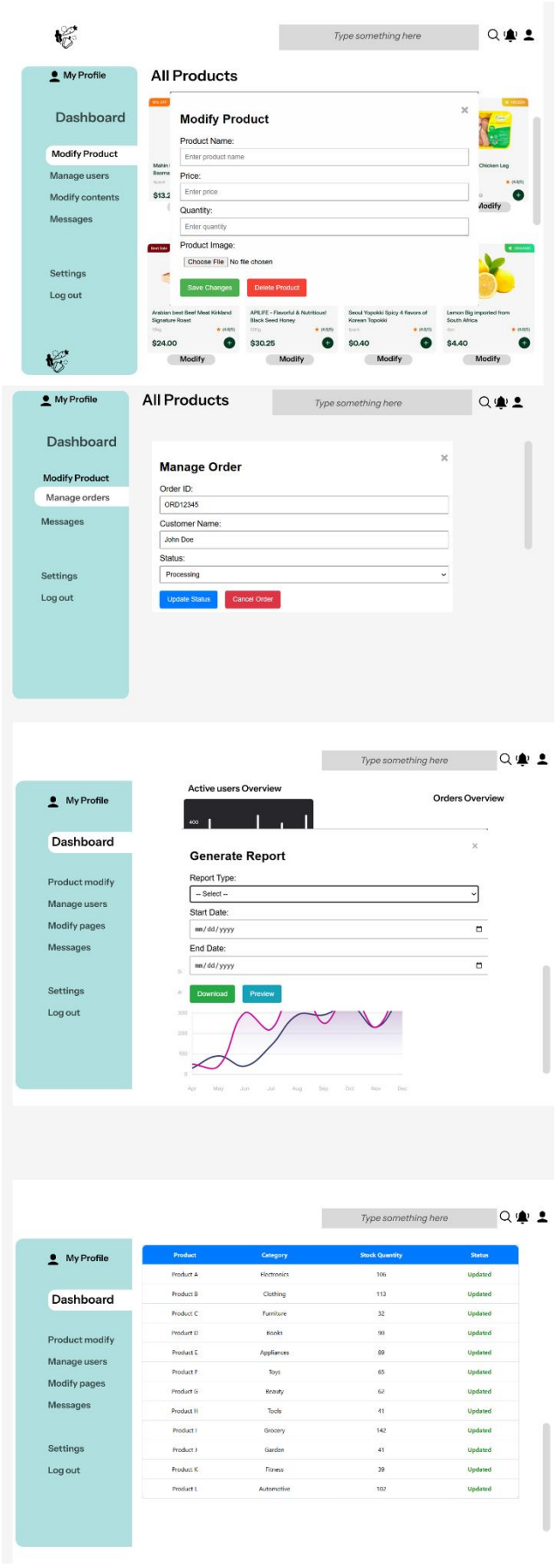

Figure 29: prototype 3

Figure 30: payment prototype

11.Conclusion

In conclusion, effective system modeling plays a crucial role in the successful development of modern software solutions. Tools like UML diagrams, including class, sequence, and activity diagrams provide clear, visual representations of system architecture and dynamic behaviors. These models ensure that developers, designers, and stakeholders maintain a shared understanding throughout the project lifecycle. Furthermore, incorporating prototyping allows teams to validate concepts early, refine user interfaces, and minimize costly revisions by identifying issues before full-scale development begins.

When integrated with Agile methodologies, these modeling and design practices enhance adaptability, support iterative development, and foster continuous improvement. This approach enables teams to deliver robust, user-centric systems that are both scalable and aligned with real-world requirements.

During the completion of this coursework, several challenges arose, particularly in understanding and constructing the various UML diagrams and the Work Breakdown Structure (WBS). The WBS posed initial confusion regarding its format and purpose. However, through the guidance and support of our respected tutors and module leaders, these uncertainties were clarified. Their timely feedback and encouragement were instrumental in building our confidence and helping us apply the theoretical concepts in a practical context.

The regular weekly reviews were especially valuable, providing structured opportunities for reflection, correction, and refinement. They ensured that progress remained aligned with learning objectives and encouraged collaboration between students and instructors. Overall, this experience has deepened our appreciation for systematic design in software engineering and highlighted the importance of communication, iteration, and mentorship in achieving academic and professional success.

12. References

beedle, M. (2001). agile manifesto. In *https://agilemanifesto.org/authors.html.*

Clark, W. (2019). *Agile Methodology.* Independently published.

Engineering, F. (n.d.). Retrieved from Fynd engineering: https://www.fynd.academy/blog/architectural-design-in-software-engineering#what-is-architectural-design-in-software-engineering

Martina Seidl. (2015). *UML @ Classroom.* Springer International Publishing.

Martins, J. (2025, january 25). Retrieved from asana: https://asana.com/resources/what-is-kanban

Paradigm, V. (2016). *Visual Paradigm*. Retrieved from https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-communication-diagram/

Seetharamugn. (2023, August 30). Retrieved from Medium.com: https://medium.com/@seetharamugn/the-complete-guide-to-event-driven-architecture-b25226594227

University, K. (2020). *Kanban University*. Retrieved from Kanban University: https://kanban.university/kanban-guide/