

# Chapter 3 – Design

## Introduction

Between requirements analysis and implementation, software design is a crucial stage in the software development lifecycle. Creating a blueprint for a software system's structure, behavior, and interactions is necessary to make sure it fits user needs and is also effective, manageable, and scalable. This overview delves into the fundamental ideas, tenets, and practices that guide efficient software development.

The foundation of successful software development is good software design. It affects the ultimate product's quality, the development process, and the project's success. An effective software system reduces the possibility of errors and ensures long-term viability since it is simpler to comprehend, adapt, and maintain.

Making a number of crucial judgments is necessary for software design. Which programming language is ideal for this project? How should data be efficiently stored and retrieved? Should the system be modular or a single piece? Trade-offs are frequently included in these choices. While putting security first may slow down processing performance, choosing a user-friendly design may result in more complexity. To produce a well-rounded system, designers must strike a balance between the opposing objectives of functionality, performance, security, scalability, and usability.

The prototyping model is a method for developing systems that involves creating a model, testing it, and then making any necessary revisions until, at long last, a reliable model is created from which the complete system or item may now be created. This methodology performs effectively in situations where not all of the project needs are initially fully known. There is an iterative, experimental process between the designers and the clients. The key steps of this methodology include defining the core needs, creating a working prototype, testing the working prototype, and modifying or expanding the requirements.

## **Related design strategies**

Due to the distributed nature of the web, the requirement for adaptable user interfaces, and the difficulties associated with controlling data flows over the network, designing web-based systems calls for unique considerations. The following software design tactics were specifically chosen for web-based systems because the advantages are very high

- Divide the system into a client-side (the browser) and a server-side (the backend) component using a client-server architecture. Scalability is made possible by this division since you may separately scale and optimize each component.
- In order to provide a consistent user experience on computers, tablets, and smartphones, responsive web design involves creating user interfaces that can adjust to various screen sizes and devices.
- Maintenance and installation are easy.
- Development is cost-effective.

## **Object Oriented analysis and design (OOAD)**

A process called object-oriented analysis and design (OOAD) is used to create software systems that adhere to object-oriented concepts. It entails decomposing a challenging issue into smaller, more manageable pieces, then designing and arranging these pieces with object-oriented principles. An overview of the main procedures and ideas in object-oriented analysis and design is given below

- Recognize and collect from stakeholders the functional and non-functional requirements of the software system.
- To depict the interactions between system actors and the system itself, create use case diagrams.
- Determine the relationships between the main system components.
- Dividing the system into more compact, coherent, and loosely connected components will help.

Modular, maintainable, and adaptive software systems are the goals of object-oriented analysis and design. In order to model and represent real-world items and their interactions in the software

domain, it places a strong emphasis on the use of objects, encapsulation, inheritance, and polymorphism. Developers can build strong and adaptable software systems that closely match the problem domain and effectively address user needs by adhering to these guidelines.

## **MVC Architecture**

A design pattern called Model-View-Controller (MVC) architecture is used in software development to divide an application's responsibilities into three primary parts: Model, View, and Controller. This division aids in managing complexity, enhancing maintainability, and fostering reuse in software systems. Here is a brief description of each MVC component:

The separation of functions is the primary principle of MVC. It is possible to modify one aspect of the system without impacting the others by keeping the Model, View, and Controller distinct. The codebase becomes more modular and is simpler to maintain as a result. For instance, you can modify the View (data presentation) without affecting the Model (data processing logic) or the Controller (user input handling logic).

The Model maintains data management (for example, database queries), the View manages UI presentation (for example, HTML templates), and the Controller coordinates the interaction between the Model and the View depending on user requests (for example, URL routing). MVC is frequently used in web development.

## System Architecture

Any software project's foundation is its system architecture, which offers a tactical plan for how all the parts fit together. A well-architected software system necessitates careful thinking to accomplish usefulness, performance, and maintainability, much as a well-designed structure necessitates meticulous planning to assure stability, usability, and beauty. The fundamental components, interactions, and technologies that make up the software solution are defined by the system architecture. It is comparable to generating a roadmap to direct developers in building a stable and unified digital environment. System architecture lays the foundation for the successful design, implementation, and evolution of a software system by defining the links between distinct modules, defining communication protocols, choosing appropriate technologies, and handling scalability, security, and other issues. It's an important first step that establishes the framework.

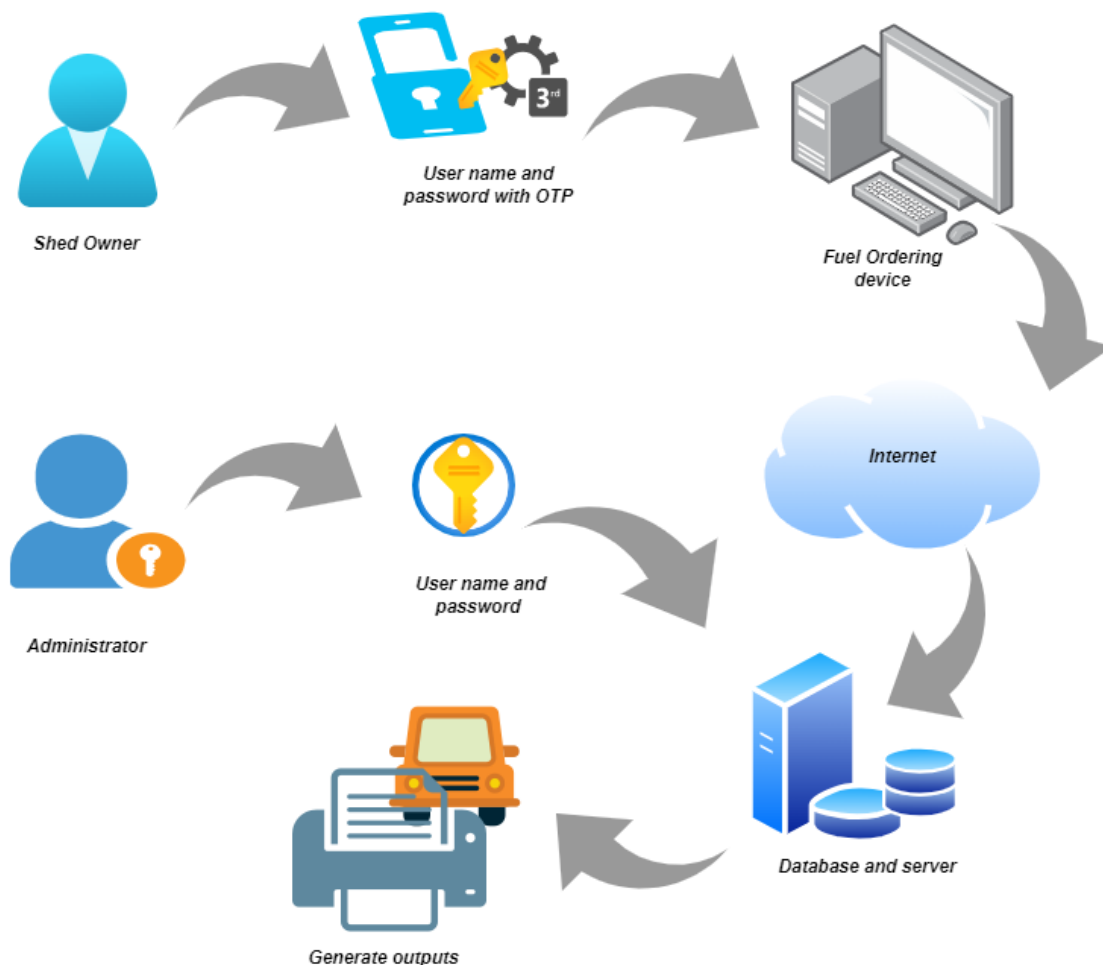


Figure 1 : System Architecture Diagram

## **UML Diagrams**

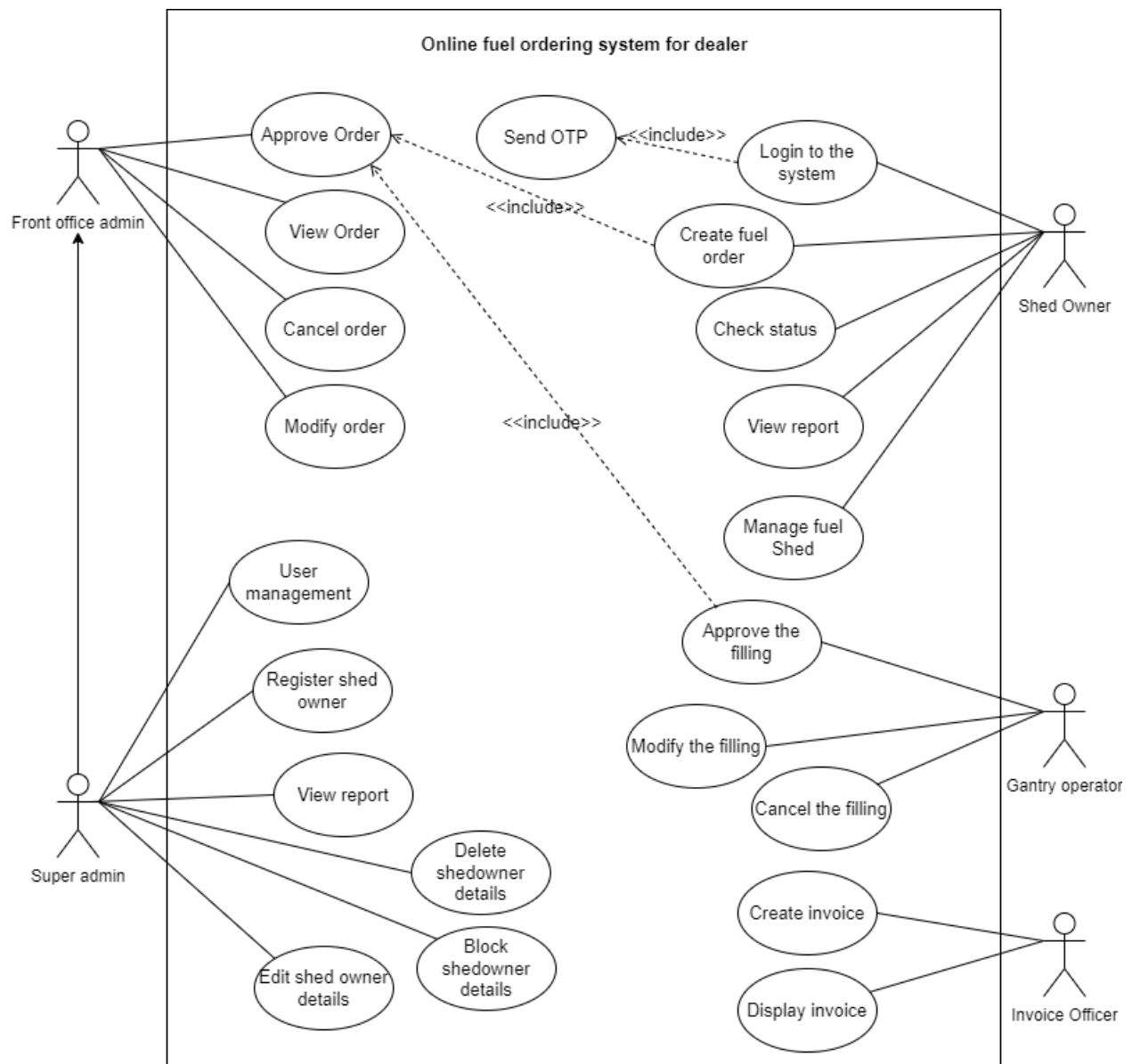
A graphical representation that is widely used in the fields of software engineering and system design is the Unified Modeling Language (UML) diagram. It acts as a uniform notation system, providing a visual framework to express the complex structure, dynamic behavior, and relationships present in various systems and processes. Classes, objects, interactions, and component relationships can all be represented using a variety of symbols, shapes, and relationships in UML diagrams. Both technical and non-technical stakeholders can benefit from this visual language's ability to clarify complicated ideas and help them understand the basic construction and operation of a system. Throughout the whole software development lifecycle, UML diagrams are essential because they help with requirements analysis, design structuring, code implementation, and documentation. UML diagrams essentially go above the limitations.

### **Use Case Diagrams**

An illustration that provides a high-level view of the functional requirements of a system from the perspective of its users or external entities is a use case diagram, a key tool in the Unified Modeling Language (UML). It functions as an effective communication tool, allowing stakeholders to understand and see the interactions between different actors (users, other systems, or entities), as well as the features offered by the system.

Actors and use cases are shown as stick figures and ovals, respectively, in a use case diagram. The interactions or functionalities that each actor can start or take part in are represented by the lines connecting actors to use cases. The diagram's clarity and simplicity make it especially helpful for requirement analysis since it assists in defining and identifying the essential features that the system must provide to meet user needs.

By aiding in the identification of system boundaries, the assessment of critical user interactions, and the discovery of viable use case situations, use case diagrams serve as a foundation for the development process. This kind of model makes it easier to match user expectations with development efforts, ensuring that the finished system effectively satisfies those demands.



**Figure 2: Use Case Diagram**

## Use Case Scenarios

Table 1: Use case scenario 1

Use case	Description
Name	Place a fuel order
Description	Shed owner place an order
Actors	Shed Owner
Scenario	<ol style="list-style-type: none"><li>1. The Shed owner logs into the system</li><li>2. The user selects material type and quantity</li><li>3. The User selects the Create Order button.</li></ol>
Alternative flow	<ol style="list-style-type: none"><li>1. User can add another fuel order by click on add another button</li></ol>

Table 2: Use case scenario 2

Use case	Description
Name	Approve fuel order
Description	Admin and super admin can approve the fuel order which is requested by the shed owner.
Actors	Super admin, front office admin
Scenario	<ol style="list-style-type: none"><li>1. Admin log into the system</li><li>2. Admin view the new order list</li><li>3. Select the order from the list</li><li>4. Admin approves the order to proceed</li></ol>
Alternative flow	<ol style="list-style-type: none"><li>1. Admin can reject the order as well as cancel the order</li></ol>

## Activity Diagram

An essential component of the Unified Modeling Language (UML), an activity diagram is a graphic representation of how decisions, actions, and systems flow inside a system or process. This diagram type offers a simple and understandable approach to representing the order of events and the reasoning underlying them, making it particularly useful for representing dynamic parts of software systems, business processes, and workflows.

Activities are shown as rounded rectangles in an activity diagram, and arrows linking these rectangles show the direction of control moving between activities. Diamond-shaped decision points symbolize decision points that use conditional logic to direct the flow leading to various branches of activities. As parallel or concurrent execution paths are indicated by forks and joins, complicated scenarios involving numerous concurrent operations can be modeled.

Activity diagrams are frequently used for system comprehension, process design, and requirement analysis. They assist in determining the series of actions necessary to achieve a particular objective, recording the order of execution and any decision points that might affect the result. This level of specificity makes it easier for stakeholders to work together since it gives them a shared visual language to talk about and improve the process logic.



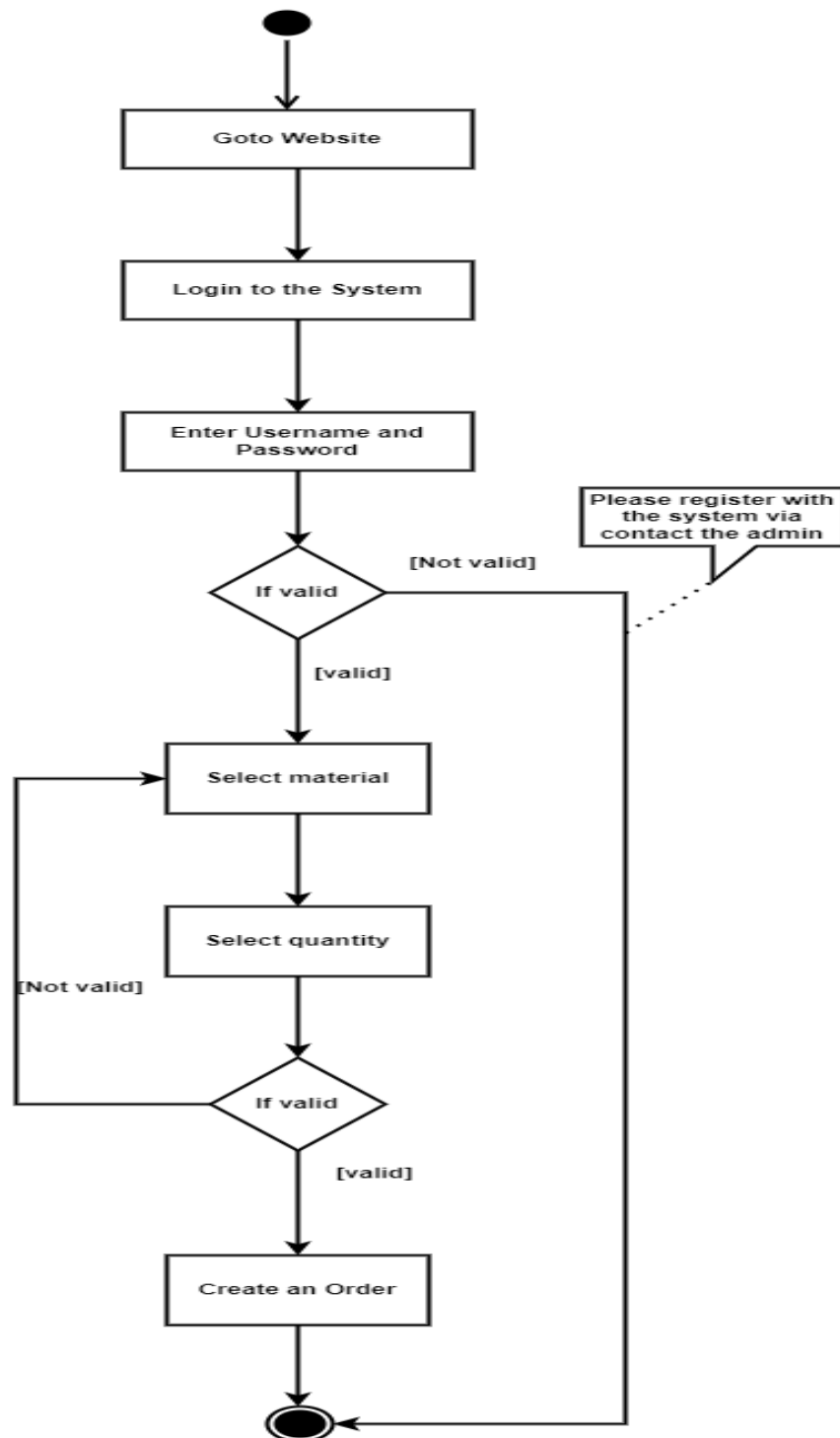


Figure 3: Place order by Shed Owner activity diagram

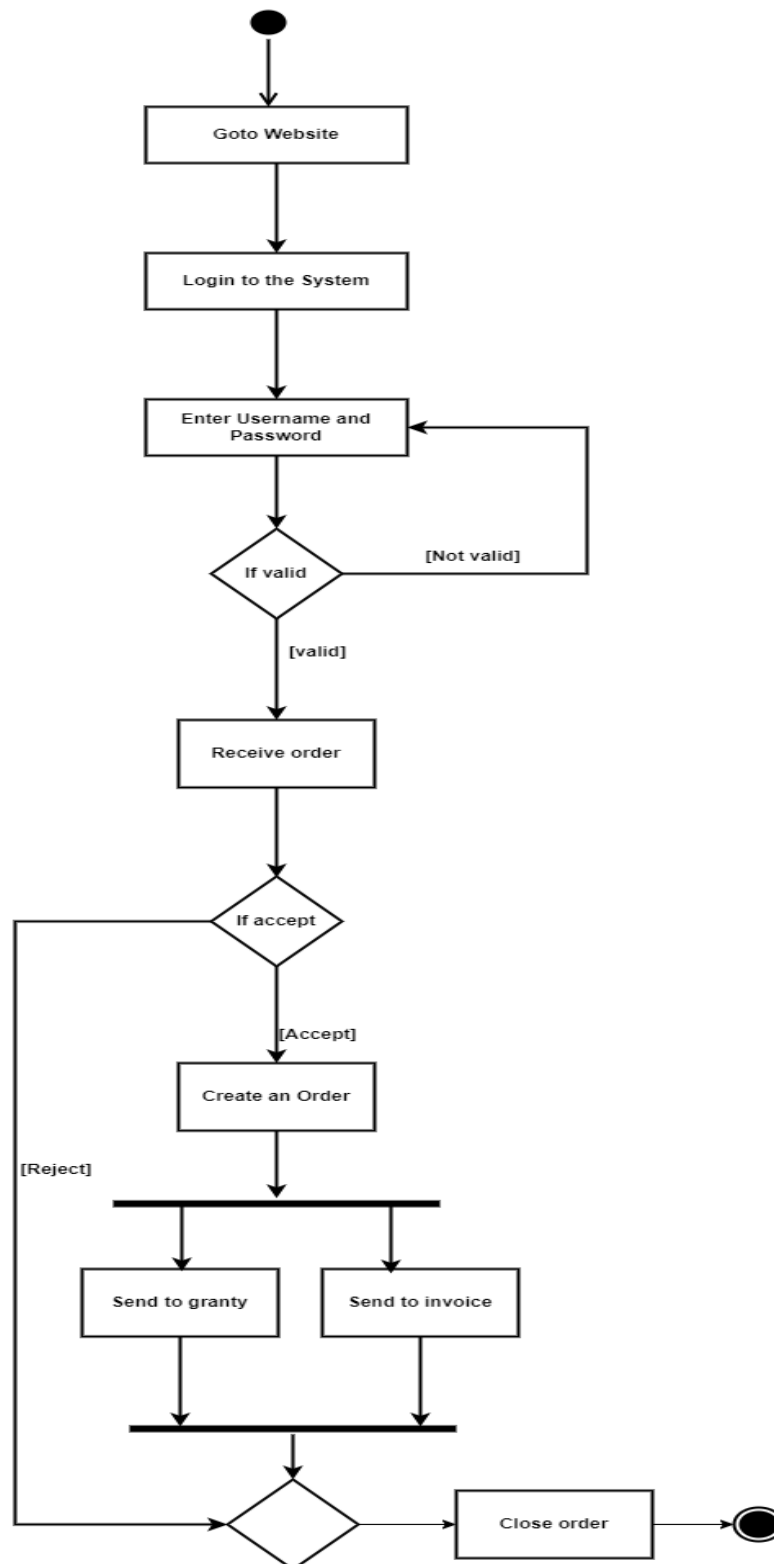


Figure 4: Employee Create an order activity diagram

## Sequence Diagram

A visual representation that captures the interactions and chronological order of messages sent between distinct objects or components inside a system is a sequence diagram, which is a key component of the Unified Modeling Language (UML). This particular form of diagram is particularly adept at capturing the dynamic behavior of software systems and processes, offering a thorough explanation of how objects interact to carry out particular functionalities.

In a sequence diagram, messages are represented by arrows that go horizontally between lifelines, which stand in for objects or actors as lifelines. Stakeholders can comprehend how objects communicate, what activities they take, and how they react to one another's demands thanks to the messages' orderly transmission.

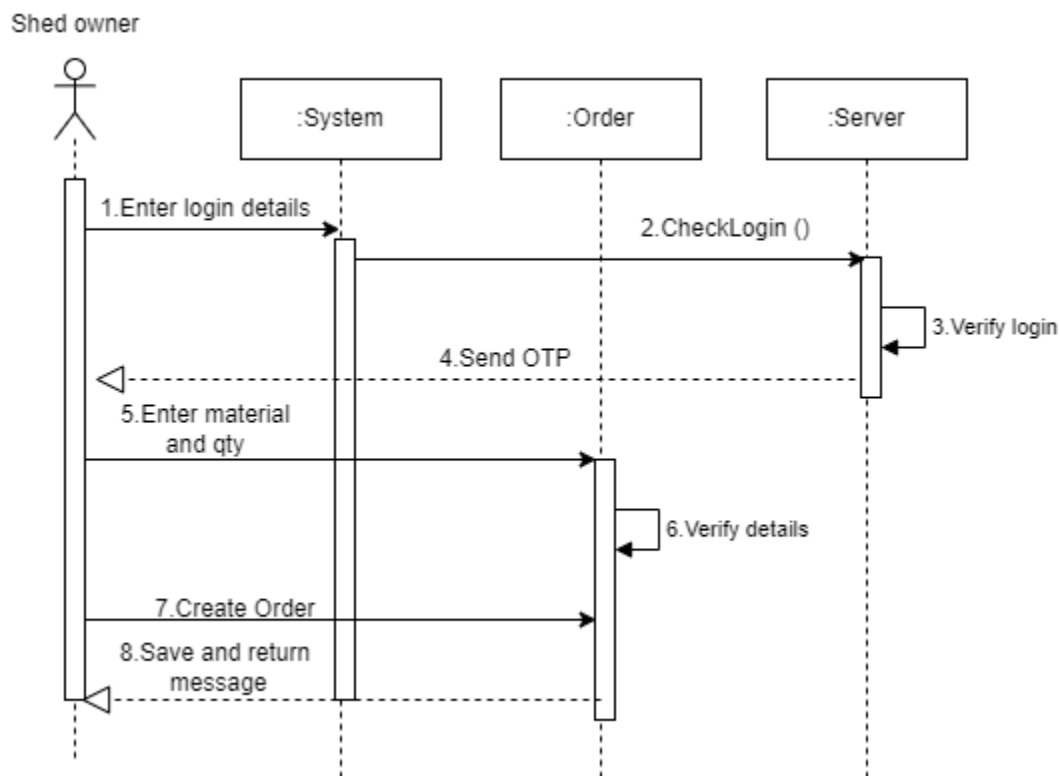


Figure 5: Place order by Shed Owner sequence diagram

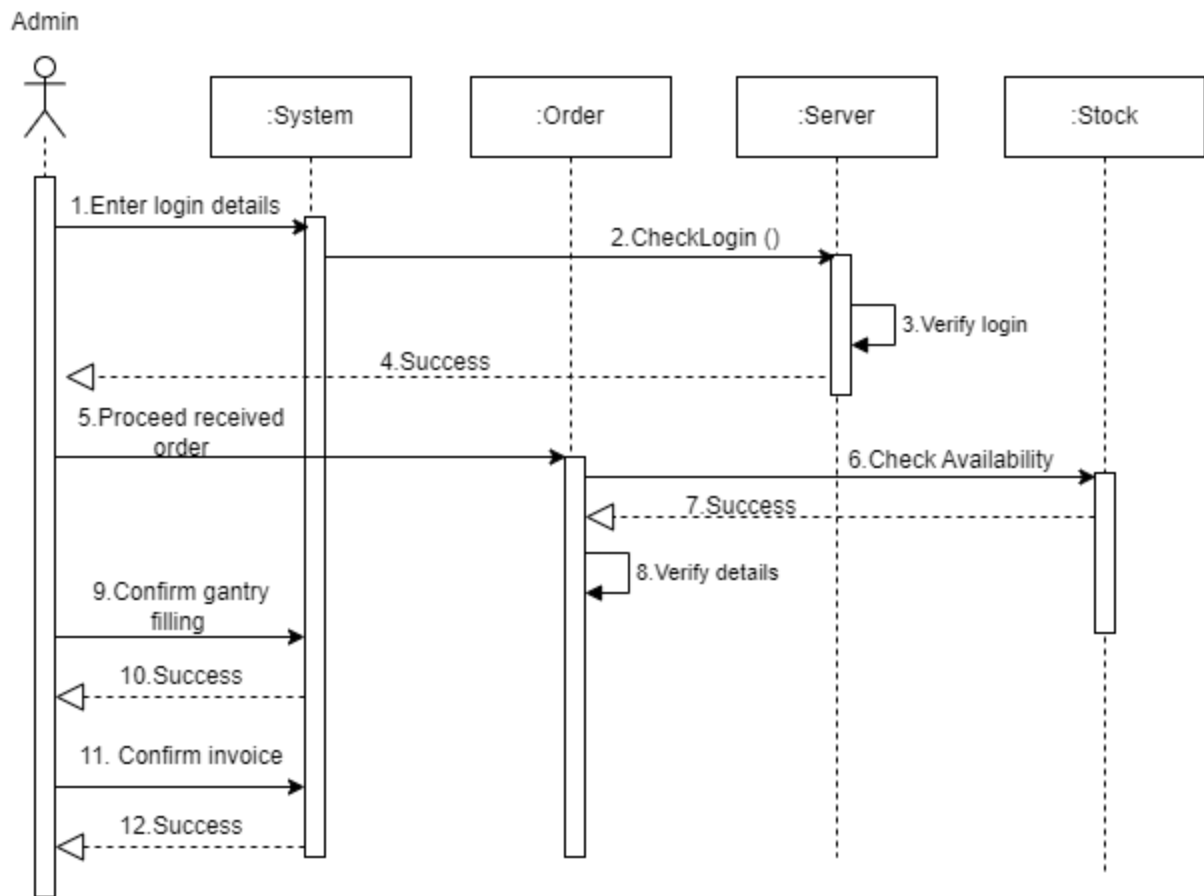


Figure 6: Employee Create an order sequence diagram

## Database Design

A key component of software development is database design, which entails developing a structured and effective data storage solution to satisfy the information management requirements of an application or system. It includes a methodical procedure for developing the data model, classifying data components, establishing linkages, and improving data manipulation and retrieval.

Integrity, accuracy, and accessibility of data are the main goals of database design. It starts with carefully examining the demands of the users, the application's requirements, and the many kinds of data that need to be saved and managed. Then, this data is transformed into a logical data model, which is frequently represented by Entity-Relationship (ER) diagrams that list the entities (data objects), attributes (properties), and relationships among them.

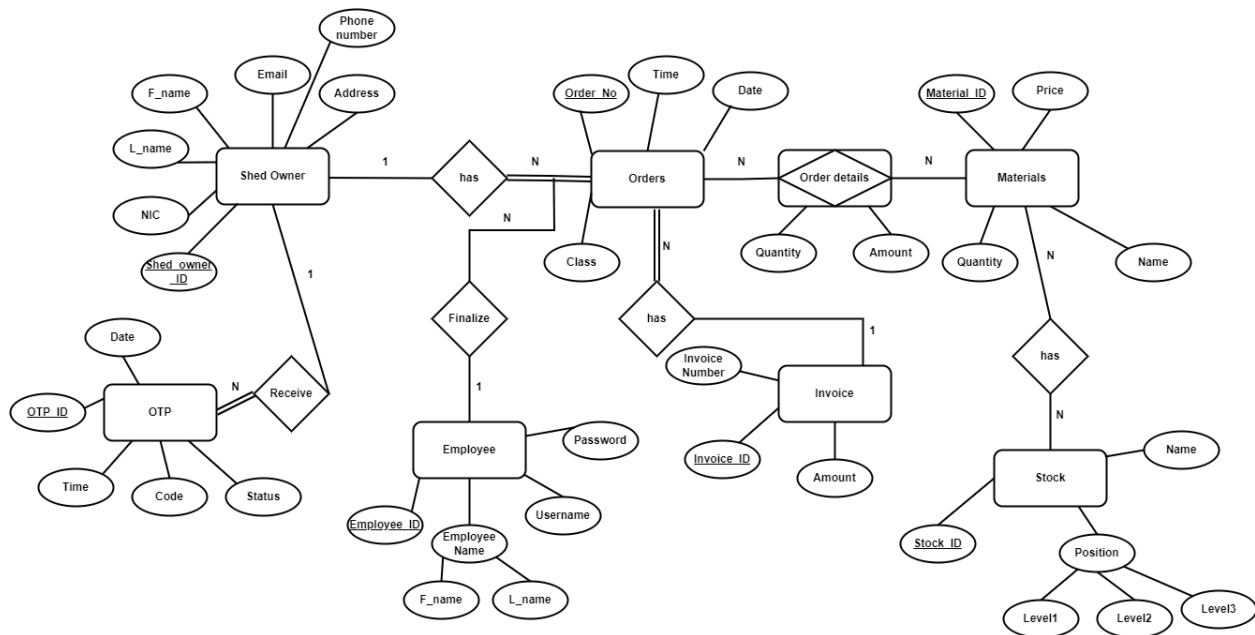



Figure 7: ER Diagram

## User Interface Design


The goal of user interface design is to make things simple for users to use on computers, mobile devices, and websites. When done correctly, it assists consumers in learning and using things without feeling confused or frustrated. Additionally, it improves the appearance of objects and promotes user confidence. By avoiding issues and ensuring user satisfaction, good user interface design helps users save time and money. Additionally, it is aware of new technologies and consumer trends, so it's always attempting to improve. In order to have things go smoothly and seem good for everyone. Here, I am using draw.io to create some major interfaces for the project.





Navinna Shed  
test@gmail.com

Dashboard

Order

 Search

## Order Fuel



Fuel Type  

select

Fuel Quantity  

select

City  

select

Date of delivery  

select

Add another Fuel


Amount

Diesel	Petrol	Super Diesel
quantity	quantity	quantity
amount	amount	amount
		Total

Submit

Cancel


Figure 8: Place an Order by Shed owner



Navinna Shed  
test@gmail.com



Dashboard

Order

 Search

Report

## View Orders



Search filter

Order ID

Dropdown button

Order Status

Dropdown button

From Date

Dropdown button

To Date

Dropdown button

Search

Show

50

entries

Search

Order ID	Customer Name	Material	Quantity	Status	Date
Value 1	Value 2	Value 3	Value 1	Value 2	Value 3
Value 4	Value 5	Value 6	Value 4	Value 5	Value 6
Value 7	Value 8	Value 9	Value 7	Value 8	Value 9
Value 10	Value 11	Value 12	Value 10	Value 11	Value 12

Showing 4 to 10 entries

Figure 9: Create an order by Employee

# Progress

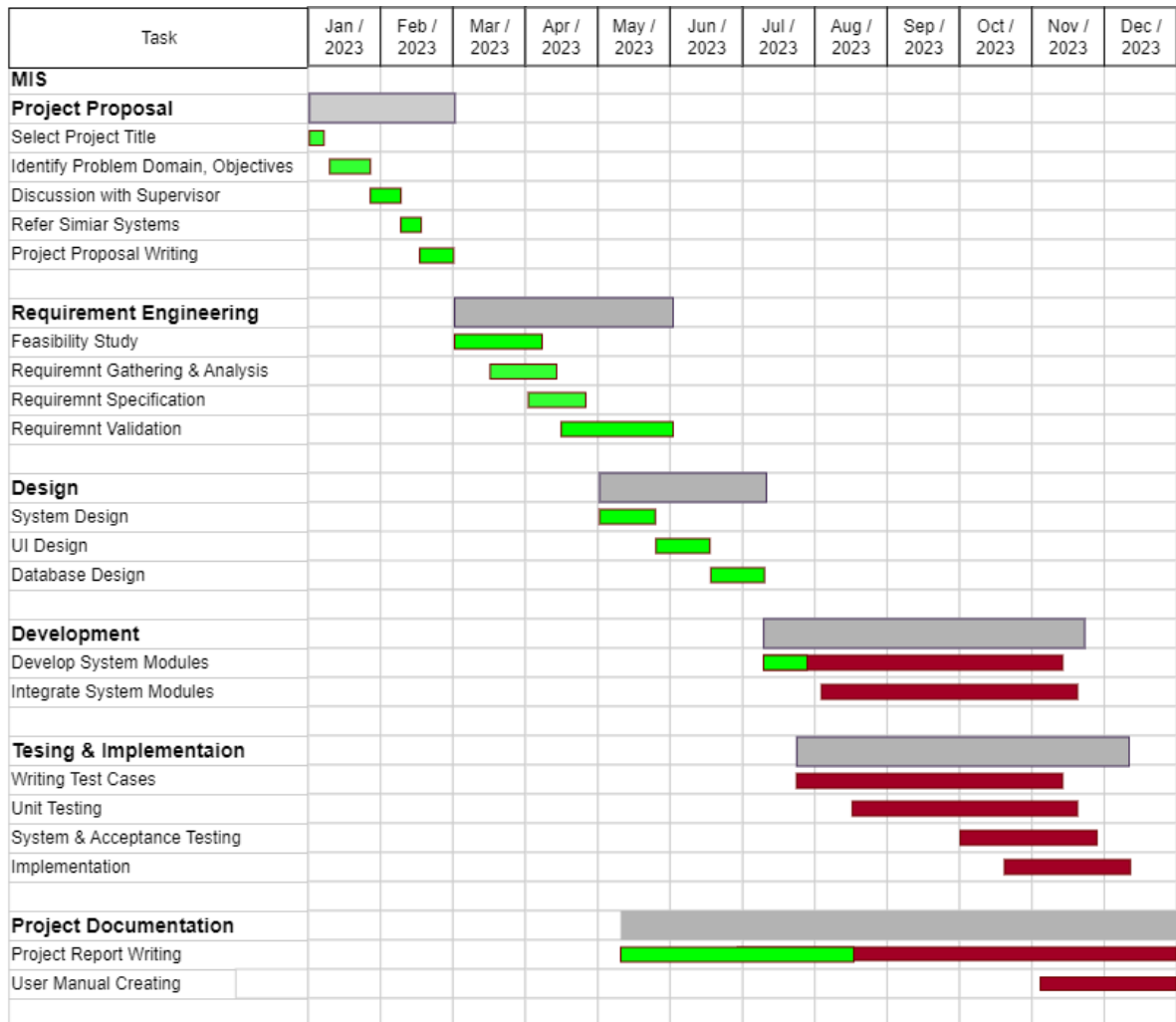


Figure 10: Gantt Chart



Supervisor *			
Supervisor's Comments			
Name	Prof. K P Hewagamage		
Designation	Prof	Signature	
		Date	

\* - Supervisor must be a UCSC academic staff member

Advisor +	
Advisor's Comments	
Name	

Designation		Signature	
		Date	

+ - Advisor can be selected as candidate wish