The Egyptian E-Learning University
Faculty of Computers and Information Technology

Speedy Logistic Website

Graduation Project Documentation

Team members:

Raneem Mohamed Abdelrazek	20-00833
Mariam Hazem Ibrahim Ahmed	20-00346
Sherif Mohammed Hesham	20-00347
Omar Ehab Farouk	20-00313
Malak Moh. Ibrahim Bayoumi Elfatatry	20-01326
Adham Mohamed Ibrahim	20-00615
Adham Ahmed Morsy	20-02061

Supervised by:

PROF. Amira Idrees

Eng. Mohamed Abdelmawgoud Khedr

Acknowledgment

First and foremost, we would like to express our gratitude to PROF. Amira Idrees, the chairman of our committee, for his unwavering support, superb direction, and inspiring words throughout the course of our senior project. We would also like to thank all of our professors for their support and advice throughout our education, as well as their assistants, particularly Eng. Mohamed Khedr for all of the guidance he provided to us. We want to express our gratitude to our family, especially to our parents, for their support, tolerance, and help over the years. We will always be grateful to our parents for remembering us in their prayers. Special thanks and appreciation go out to our coworkers who have supported us, giving us advice, or even just smiled at us once or twice. You are all wonderful to have around.

ABSTRACT

Speedy website is an exciting new effort to make logistics services easier for clients. Water transportation is one of the oldest methods of transfer of goods, people, and services between two points in the world

ABBREVIATIONS

GIS	Geographic Information System
API	Application Programming Interface
SVG	Scalable Vector Graphics
C#	C Sharp
HTML	Hyper Text Markup Language
PDF	Portable Document Format
CSS	Cascading Style Sheets
LINQ	language integrated query

TABLE OF CONTENTS

CHAPTER ONE: INTRODUCTION

1.1 Overview

Logistics companies play a pivotal role in managing the intricate web of processes involved in the movement and storage of goods.

At its core, the logistics field encompasses the planning, implementation, and control of the efficient flow of products, information, and resources from point of origin to point of consumption.

This involves meticulous coordination of various elements such as transportation, warehousing, inventory management, and distribution.

A successful logistics company strives to optimize these processes, ensuring timely delivery, cost-effectiveness, and seamless supply chain operations for its clients.

We redefine logistics excellence by placing a paramount focus on efficient documentation and invoicing services.

As a leading logistics company, we understand that a streamlined supply chain is the backbone of successful businesses.

Our commitment is to simplify and optimize your documentation processes, ensuring that your shipments move seamlessly from origin to destination.

1.2 Problem Background:

Logistics supply chain management systems can encounter several common issues that impact their effectiveness and efficiency. Some of the most common issues in logistics supply chain management systems include:

1. **<u>Information technology and security:</u>**

- Electronic security threats and systems breakthrough
- Challenges related to keeping customer and shipping data safe and secure.

2. **Ship and fleet technology:**

- Continuous technological modernization and fleet modernization cost.
- Challenges related to alternative energy technology in ships.

3. **Environmental challenges:**

- Sustainability trends and environmental issues.
- Challenges related to navigation compliance with emission standards.

4. **Operations and logistics:**

- Effective inventory management and supply chain control.
- Improving the efficiency of operations and improving logistics.

5. **Reduce costs and increase profitability:**

- Increased fuel costs and fuel efficiency.
- Cost pressures under intense competition.

6. **Innovation and technology:**

- Adopting new technologies to improve productivity and efficiency.
- Challenges of integration with AI and big analytics technology.

7. **Changes in demand and supply:**

- Fluctuations in demand for sea freight and their impact on planning and operation.
- Dealing with these challenges depends on the company's ability to adapt, innovate and respond effectively to the changing needs of the shipping and maritime industry.

1.3 Problem Statement:

The shipping industry and shipping companies face many challenges and problems.

Here are some issues that this sector may face:

1-Environmental pollution: Oil spills and environmental pollution can significantly affect the marine and coastal environment.

Cases of illegal dumping of waste from ships.

2-Maritime Safety: High rate of ship accidents and collisions in marine waters. Lack of application of safety and security procedures.

3- Shipping delay: Delays in shipping schedules may lead to supply chain issues.

4- Fuel cost: The high cost of fuel affects the profitability of shipping companies and increases shipping costs.

5- Staff challenges: Lack of skills and qualified labor in this sector.

1.4 Significance of the project:

In a logistics company, documentation and invoices hold significant importance for several reasons:

1. Customs submission: Proper documentation, including customs declarations and invoices, is essential for compliance with international trade regulations. This ensures that shipments move smoothly through customs, avoiding delays and penalties.

2. Shipping Accuracy: Invoices and packing lists provide a detailed account of the shipped goods, aiding in accurate loading, unloading, and

tracking. This minimizes errors and enhances the overall efficiency of logistics operations.

3**. Legal Protection:** Comprehensive documentation, such as bills of lading and contracts, serves as legal protection for the logistics company. It establishes the terms and conditions of the shipment, reducing the risk of disputes and providing a basis for resolution if issues arise.

4. Financial Management: Invoices play a critical role in financial management by documenting the cost of services provided. They serve as a basis for billing clients, tracking revenue, and managing cash flow.

5. Supply Chain Visibility: Proper documentation facilitates supply chain visibility, allowing

logistics companies to track the movement of goods at each stage. This transparency is crucial for effective inventory management and optimizing logistical processes.

6. Client Communication: Invoices serve as a formal communication tool with clients, detailing the services provided, costs incurred, and payment

expectations. Clear and accurate documentation fosters transparency and trust in client relationships.

7. Risk Mitigation: Detailed documentation helps mitigate risks associated with transportation, including loss, damage, or disputes. Having a clear record of the shipment's condition at various checkpoints can aid in resolving issues promptly.

8. Record Keeping: Documentation provides a historical record of logistics transactions. This record-keeping is valuable for internal audits, performance evaluations, and continuous improvement initiatives within the logistics company.

1.5 Aim and Objectives

Aim:

- Streamline document management processes and enhance operational efficiency for a logistics company specializing in documentation and invoicing.

Objectives:

1. Implement a digital document management system to reduce paperwork and enhance accessibility.

2. Ensure accuracy and timeliness in the creation and processing of invoices to improve financial management.

3. Enhance communication channels with clients and partners for seamless document exchange.

4. Implement training programs to enhance the skill set of employees in document handling and invoicing procedures.

5. Integrate advanced technologies, such as optical character recognition (OCR), to automate data extraction from documents.

6. Establish and maintain compliance with industry standards and regulations related to documentation and invoicing.

7. Continuously evaluate and optimize processes to identify areas for improvement and cost-saving opportunities.

8. Foster a culture of accountability and quality control to minimize errors in document preparation and invoicing.

9. Enhance cybersecurity measures to safeguard sensitive information during document exchange.

10. Develop key performance indicators (KPIs) to monitor and measure the effectiveness of document and invoicing processes.

1.6 The Project Will Focus On:

A logistics company specializing in documentation and invoices primarily focuses on efficiently managing the flow of information and financial transactions associated with the movement of goods. Key areas of focus include:

1. Document Management: Prioritizing the organization, storage, retrieval, and sharing of various documents related to shipments, customs, and regulatory compliance.

2**. Invoicing Processes:** Ensuring accurate and timely creation, processing, and delivery of invoices to clients, taking into account diverse billing requirements and international regulations.

3. Regulatory submission: Staying abreast of and adhering to local and international regulations governing documentation, customs, and invoicing in the logistics industry.

4. Communication Channels: Facilitating effective communication with clients, suppliers, and relevant authorities to exchange necessary documents and information seamlessly.

5. Efficiency and Accuracy: Striving for streamlined and error-free processes in document preparation, handling, and invoicing to enhance overall operational efficiency.

6. Technology Integration: Embracing technologies like digital document management systems, OCR, and automation to optimize workflow, reduce manual errors, and improve productivity.

1.7 Project limitations:

A logistic supply chain management system, like any other system, may have certain limitations.

Here are some possible limitations that can arise:

1. Complex Implementation: Implementing a logistic supply chain management system can be complex and time-consuming. It may require significant effort to integrate the system with existing processes, software, and hardware within an organization.

2. Cost: Developing and deploying: a logistic supply chain management system can be expensive. It involves costs for software licenses, hardware infrastructure, training, maintenance, and ongoing system updates. Small businesses or organizations with limited budgets may find it challenging to afford such systems.

3. Data Integration: Integrating data from various sources and systems within an organization can be a challenge. Different departments or divisions may use different software or data formats, making it difficult to achieve seamless data integration. This can result in data inconsistencies and hinder the effectiveness of the logistic supply chain management system.

4. Scalability: Logistic supply chain management systems should be able to handle increasing volumes of data and transactions as a business grows. However, scalability can be a limitation if the system is not designed to handle large amounts of data or if the infrastructure is not robust enough to support increased demand.

5. Technical Dependencies: Logistic supply chain management systems rely on technology infrastructure, such as servers, networks, and software

applications. Any technical issues or failures in these components can disrupt the system's functionality and impact the supply chain operations.

6. Change Management: Implementing a new logistic supply chain management system often requires changes in established processes and workflows. Resistance to change among employees and stakeholders can pose challenges and hinder the successful adoption of the system.

7. External Factors: Logistic supply chain management systems are influenced by external factors that are beyond the organization's control. These factors include natural disasters, political instability, changes in regulations, and disruptions in transportation networks. Such events can impact the system's ability to manage the supply chain effectively.

8. Human Error: While logistic supply chain management systems automate many processes, human error can still occur. Inputting incorrect data, misinterpretation of information, or failure to follow proper procedures can lead to errors that affect the accuracy and efficiency of the system.

1.8 Project Expected Output:

The expected output includes:

1. Efficient Document Handling: Streamlined processes for organizing, storing, and retrieving documents related to shipments, customs, and regulatory compliance.

2. Accurate and Timely Invoices: The creation, processing, and delivery of accurate invoices to clients, reflecting transparent and compliant billing practices.

3. Submission commitment: Strict adherence to local and international regulations governing documentation, customs procedures, and invoicing in the logistics industry.

4. Effective Communication: Seamless communication channels established with clients, suppliers, and relevant authorities for the exchange of necessary documents and information.

6. Skilled Workforce: Employees equipped with the necessary skills and knowledge through training programs to handle document management and invoicing effectively.

7. High Customer Service Standards: Excellent customer service with prompt resolution of inquiries and issues related to documentation and invoices, ensuring client satisfaction.

9. Continuous Improvement: A proactive approach to identifying and implementing process improvements, embracing feedback, and adopting emerging technologies to stay competitive in the logistics sector.

1.9 The main three outcomes that we focus on:

---

1. Efficiency.
2. Visibility.
3. Cost-effectiveness.
4. Accuracy

1.10 Documentation Overview

This documentation is organized in 7 chapters:

Chapter 1: Presenting an Overview and a Problem Background also Project Aim then the objective of the project.

Chapter 2: Presenting related existing systems with problem statement & purposed solution

Chapter 3: System Requirements and Planning

Chapter 4: System Design.

Chapter 5: System Implementation.

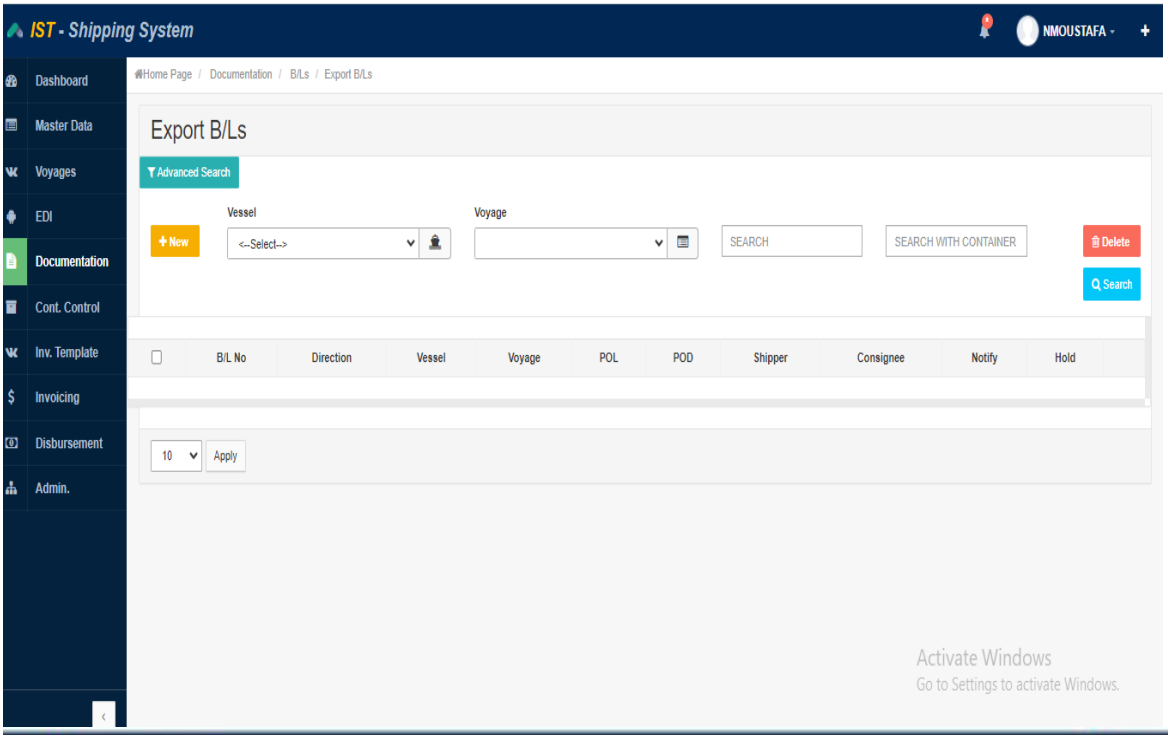
Chapter 6: Presenting the testing scenarios of the system which ensuring that all implemented processes are working correctly.

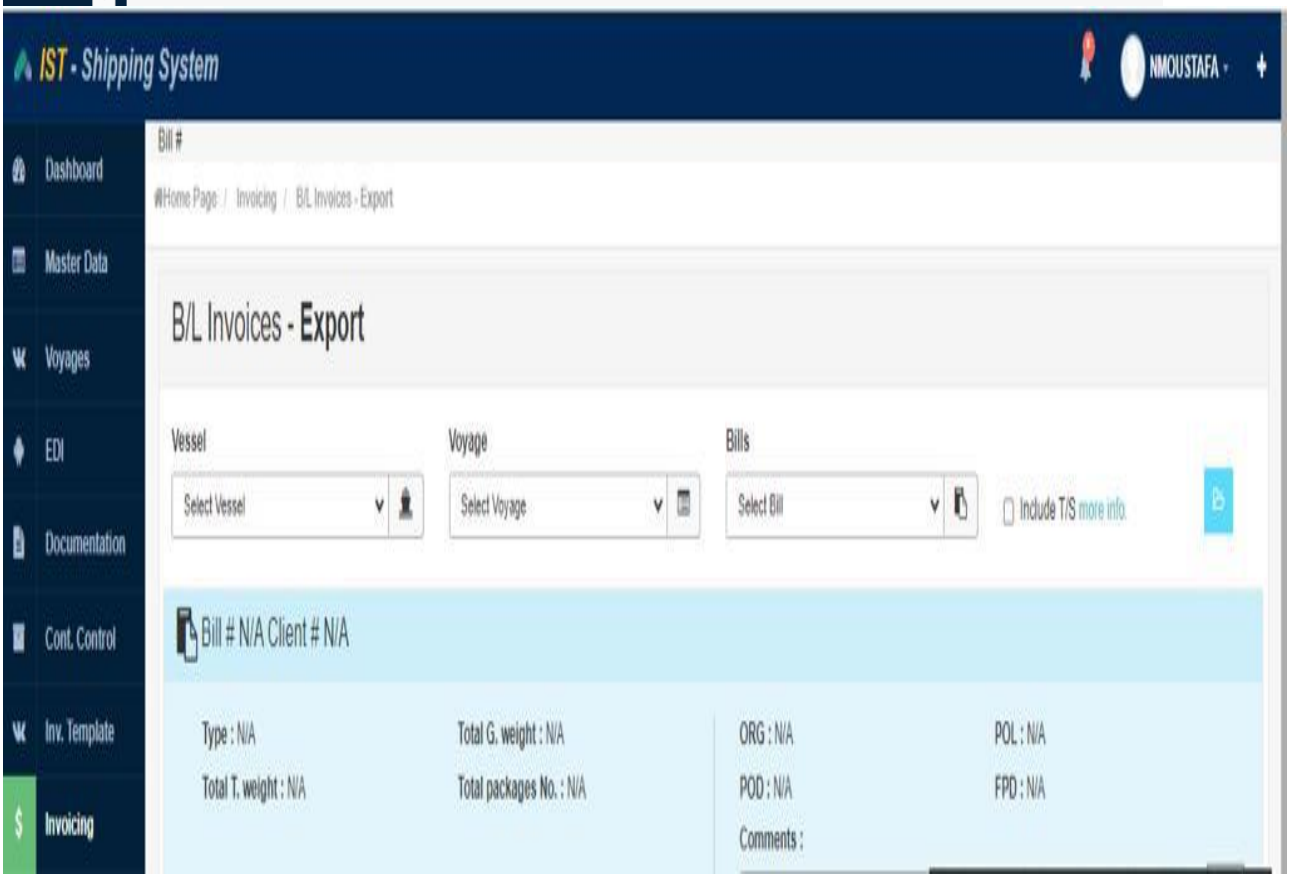
Chapter 7: Presenting conclusion of what is done and the future work of the system.

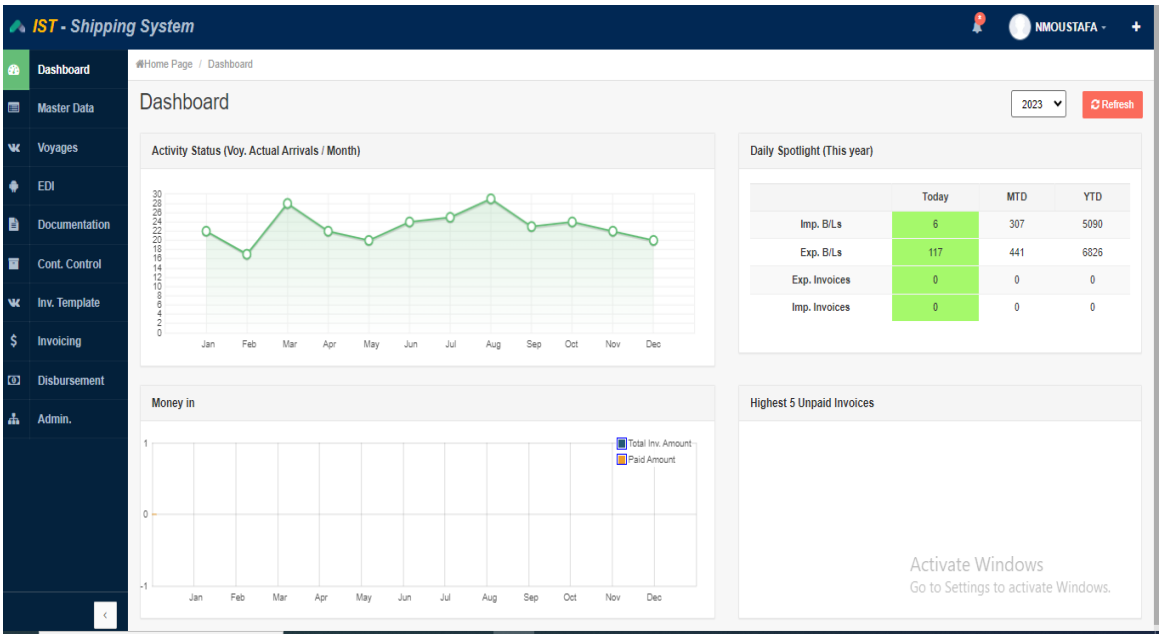
CHAPTER TWO: RELATED EXISTING SYSTEMS

2.1 Existing Systems

Here are some pictures of the IST system:






IST - Shipping System
NMOUSTAFA
Dashboard
Master Data
Voyages
EDI
Documentation
Cont. Control
Inv. Template
Invoicing
Disbursement
Admin.
Home Page / Dashboard
Dashboard
2023
Refresh
Activity Status (Voy. Actual Arrivals / Month)
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
Daily Spotlight (This year)
Today MTD YTD
Imp. B/Ls 6 307 5090
Exp. B/Ls 117 441 6826
Exp. Invoices 0 0 0
Imp. Invoices 0 0 0
Money in
Total Inv. Amount
Paid Amount
Highest 5 Unpaid Invoices
Activate Windows
Go to Settings to activate Windows.

2.2 Overall Problem of Existing System:

1-customer communication: So, some of the problems this existing system face was that it makes it easy on the employees and not on the customer, so that's our first thing to do in our system we are going to make it easy on customers

2- Data entry errors: sometimes people used to enter wrong data for invoices and documentation can result in financial problems and impact customer satisfaction

3-customer expectations: Meeting customer demands for faster shipping, accurate invoices and transparent communication poses a continual challenge

4-Route Optimization: Ensuring optimal routes for shipments to minimize costs and delivery times can be complicated, especially considering dynamic factors like traffic and weather.

2.3 Proposed Solution

- We are going to start with the first problem which is customer communication , and some of the solutions are:

1. User-Friendly Interface: Design a customer interface that is intuitive, easy to navigate, and requires minimal effort for users to access and understand relevant information.

2. Real-Time Updates: Provide real-time updates on shipment status, including estimated delivery times, delays, and any relevant changes, keeping customers informed.

3.Clear Documentation Access: Ensure that customers can easily access and download relevant documentation and invoices from the system without complications.

4. Feedback Mechanism: put a feedback mechanism to gather insights from customers, allowing you to continually improve communication based on their experiences.

5. Educational Resources: Provide resources like FAQs, video tutorials, or guides to help customers understand and navigate the system effectively.

❖ Second problem solutions which is data entry errors:

1. Automation of Data Entry: Introduce automated systems that can extract and input data accurately, reducing reliance on manual entry and minimizing errors.

2. Data Validation Checks: Implement validation checks to ensure that entered data adheres to specific formats, ranges, or criteria, reducing the likelihood of inaccuracies.

3. Error Alerts and Notifications: Set up automated alerts and notifications for potential errors, allowing for quick correction before they impact financial processes or customer satisfaction.

❖ Third problem solutions which is customer expectations:

1. **Accurate Invoicing and Billing:** Implement automated invoicing systems to ensure accuracy and transparency in billing, reducing errors and potential disputes.

2. Clear and Simple Policies: Clearly communicate shipping policies, delivery times, and return procedures to manage customer expectations effectively.

CHAPTER THREE: SYSTEM REQUIREMENTS ENGINEERING AND PLANNING

3.1 Feasibility Study :

A feasibility study for logistics and maritime transportation is critical for evaluating risks, ensuring financial viability, understanding market

dynamics, improving operations, complying with regulations, here is the most important points from the importance of a feasibility study for logistics and maritime transportation:

1-Risk Assessment: Identify and assess potential risks associated with the project.

2-Financial Viability: Evaluate projected costs, revenue streams, and overall financial sustainability.

3-Market Analysis: Analyze market trends, competition, and potential growth areas.

4-Operational Efficiency: Assess and improve operational aspects, including supply chain management.

5-Technical Feasibility: Evaluate the availability of technology, infrastructure, and resources.

6- Decision-Making Support: Provide comprehensive information for strategic decision-making.

3.2 Requirements Elicitation Techniques :

The choice of technique depends on factors such as the project's scope, the availability of stakeholders, the complexity of the system, and the development methodology.

Often, a combination of techniques is used to ensure comprehensive requirements elicitation.

By using these techniques, you can gather a diverse set of requirements, ensuring that the final system meets the needs of all stakeholders and functions effectively within its intended environment

3.3 Targeted Users :

1. **Administrative Staff:** Responsible for creating, organizing, and maintaining documents and invoices within the system.

2. Finance and Accounting Professionals: Involved in processing invoices, managing financial transactions, and ensuring accurate record-keeping.

3. Management and Executives: Require access to high-level reports, financial summaries, and document archives for strategic decision-making.

4. Customer Service Representatives: Access related documentation to address customer inquiries, resolve issues, and provide support.

5. External Auditors: Access documentation for auditing purposes, ensuring financial transparency and compliance.

6. External Clients or Suppliers: Depending on system permissions, external partners may access relevant documentation and invoices for collaborative purposes.

3.4 Functional requirements definition:

Functional requirements definition involves specifying the capabilities and behaviors that a system must exhibit to fulfill its purpose.

These requirements focus on what the system should do, describing its interactions, features, and functions. Below is a structured approach to defining functional requirements:

❖ Identify Stakeholders and Gather Requirements

- **Stakeholders Identification:** Determine all individuals or groups who have an interest in the project, such as users, clients, developers, and project managers.

- **Requirements Gathering:** Use techniques like interviews, questionnaires, user stories, and workshops to collect requirements from stakeholders.

❖ Categorize Requirements

- **Functional Requirements:** Specify what the system should do. These include:
-
    - **User Interfaces:** Describe how users will interact with the system.
    - **System Interfaces:** Detail how the system interacts with other systems.

- **Data Management:** Define how the system will handle data creation, modification, deletion, and retrieval.
- **Business Rules:** Outline the rules the system must follow to ensure correct operation within the business context.

❖ Document Functional Requirements

- **Requirement Statements:** Clearly describe each requirement. A typical format might include:

  - **ID:** A unique identifier for the requirement.
  - **Title:** A brief, descriptive name.
  - **Description:** A detailed explanation of the requirement.
  - **Rationale:** The reason why this requirement is necessary.
  - **Priority:** The importance of the requirement (e.g., must-have, should-have, could-have).
  -

❖ 4. Use Cases and User Stories

- **Use Cases:** Describe the interactions between users (actors) and the system to achieve specific goals. Include:
-
  - **Actors:** Who or what interacts with the system.
  - **Preconditions:** What must be true before the use case starts.
  - **Main Flow:** The standard sequence of events in the interaction.
  - **Alternative Flows:** Variations or exceptions in the sequence.
  - **Postconditions:** The state of the system after the use case ends.

- **User Stories:** Provide a concise way to capture user needs, typically in the format: "As a [user], I want [goal] so that [reason]."

❖ 5. Modeling Functional Requirements

- **Diagrams:** Use visual aids to represent functional requirements, such as:

    - **Use Case Diagrams:** Show actors and their interactions with the system.
    - **Activity Diagrams:** Depict workflows or processes within the system.
    - **Sequence Diagrams:** Illustrate interactions between components over time.
    - **Data Flow Diagrams**: Represent the flow of data within the system.

❖ 6. Validate and Verify Requirements

- **Validation:** Ensure the requirements accurately reflect stakeholder needs.
- **Verification:** Confirm that the requirements are feasible and testable.

❖ Manage Requirements

- **Requirements Traceability:** Maintain a traceability matrix to track requirements throughout the project lifecycle, ensuring each one is addressed.

- **Change Management:** Establish a process for handling changes to requirements, including evaluation, approval, and documentation.

3.5 Functional requirements Specification:

- Client can select the type of container that he wants for his shipment.
- Client can easily get his invoices through our website.
- Client can choose the way he wants to pay for his invoices

3.6 Non-Functional requirements:

- The system should be secure.
- The system should be easy to use.
- The system should be compatible and user friendly.
- The system should be maintainable

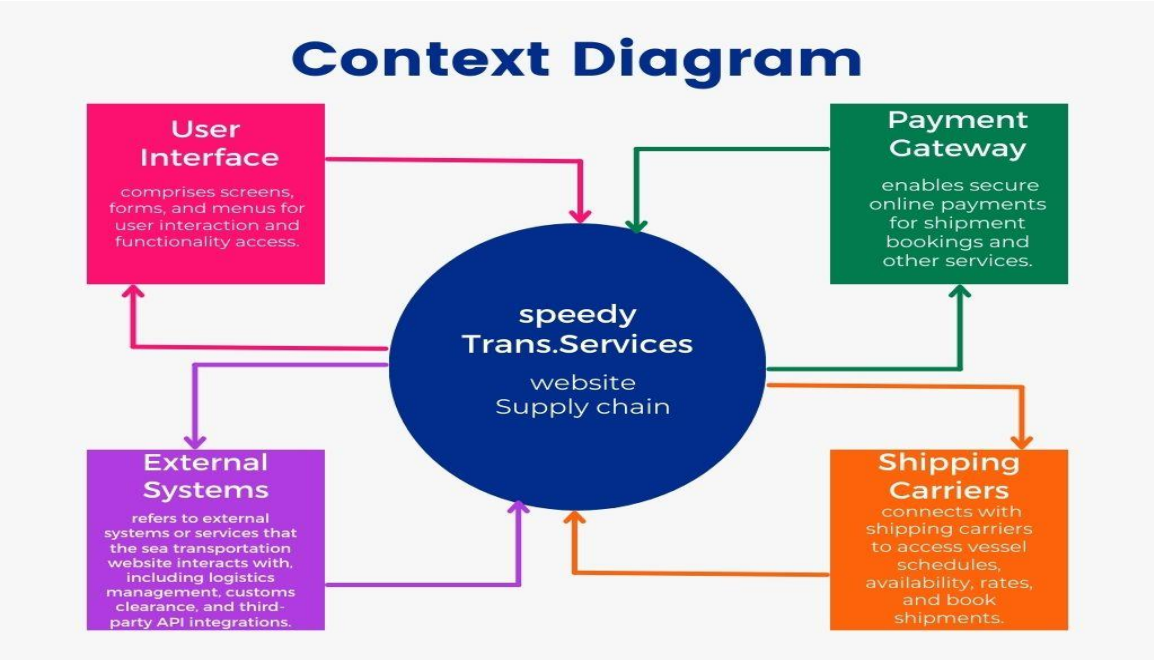
CHAPTER FOUR: SYSTEM DESIGN

4.1 Context Diagram:

- **Context Diagram:** A level 0 DFD (Data Flow Diagram), sometimes referred to as a context diagram, offers a high-level perspective of the system and its connections with outside entities. The limits of the

system are depicted, along with the external entities it interacts with, including users, shipping companies, customs authorities, and other relevant parties.

- Understanding the system's scope and the data flow between it and its external environment is made easier by the context diagram.

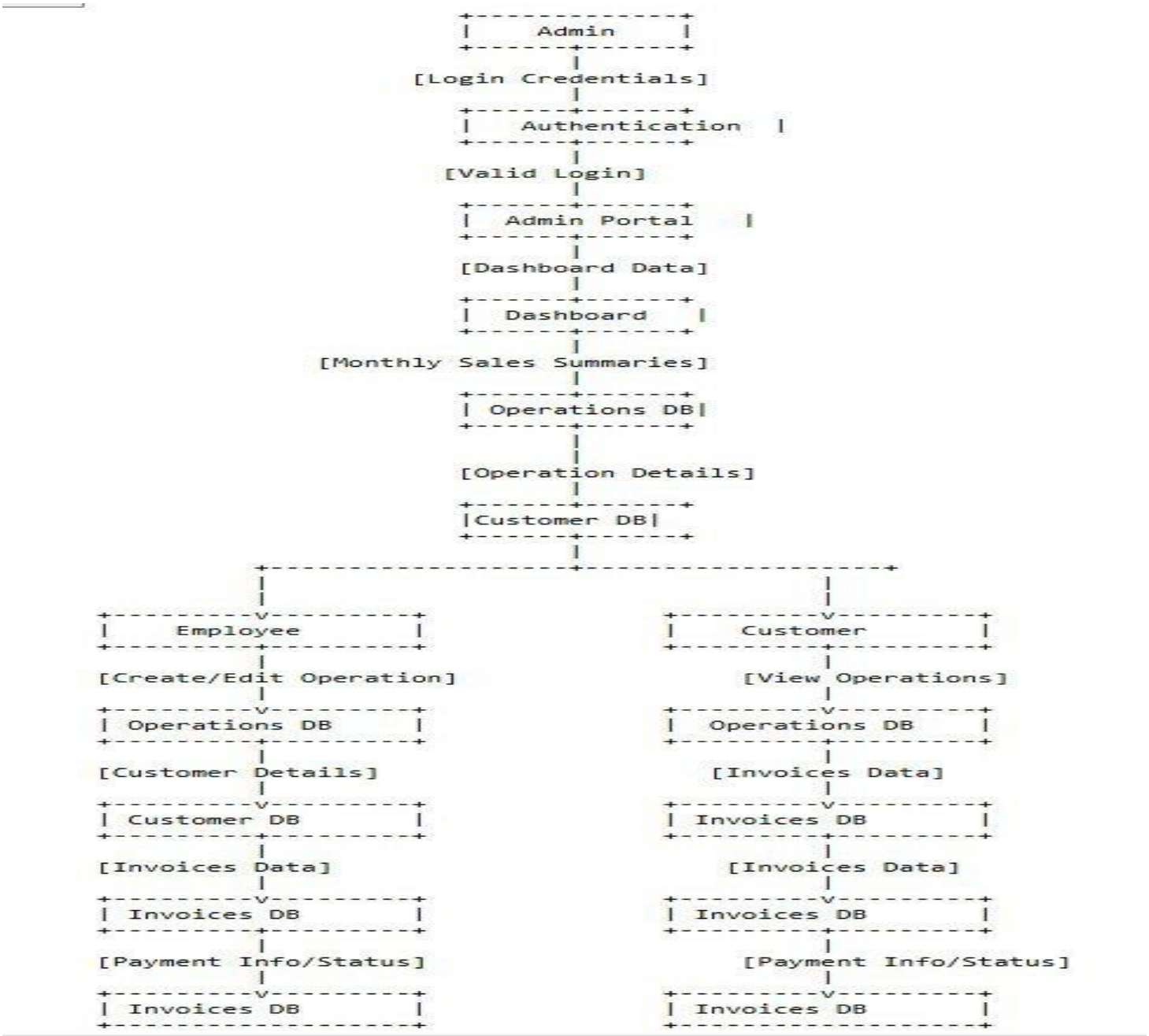


4.2 Data Flow Diagram:

Data flow diagrams, or DFDs, show how data moves through a system. It shows the flow of data through various repositories, procedures, and outside organizations.

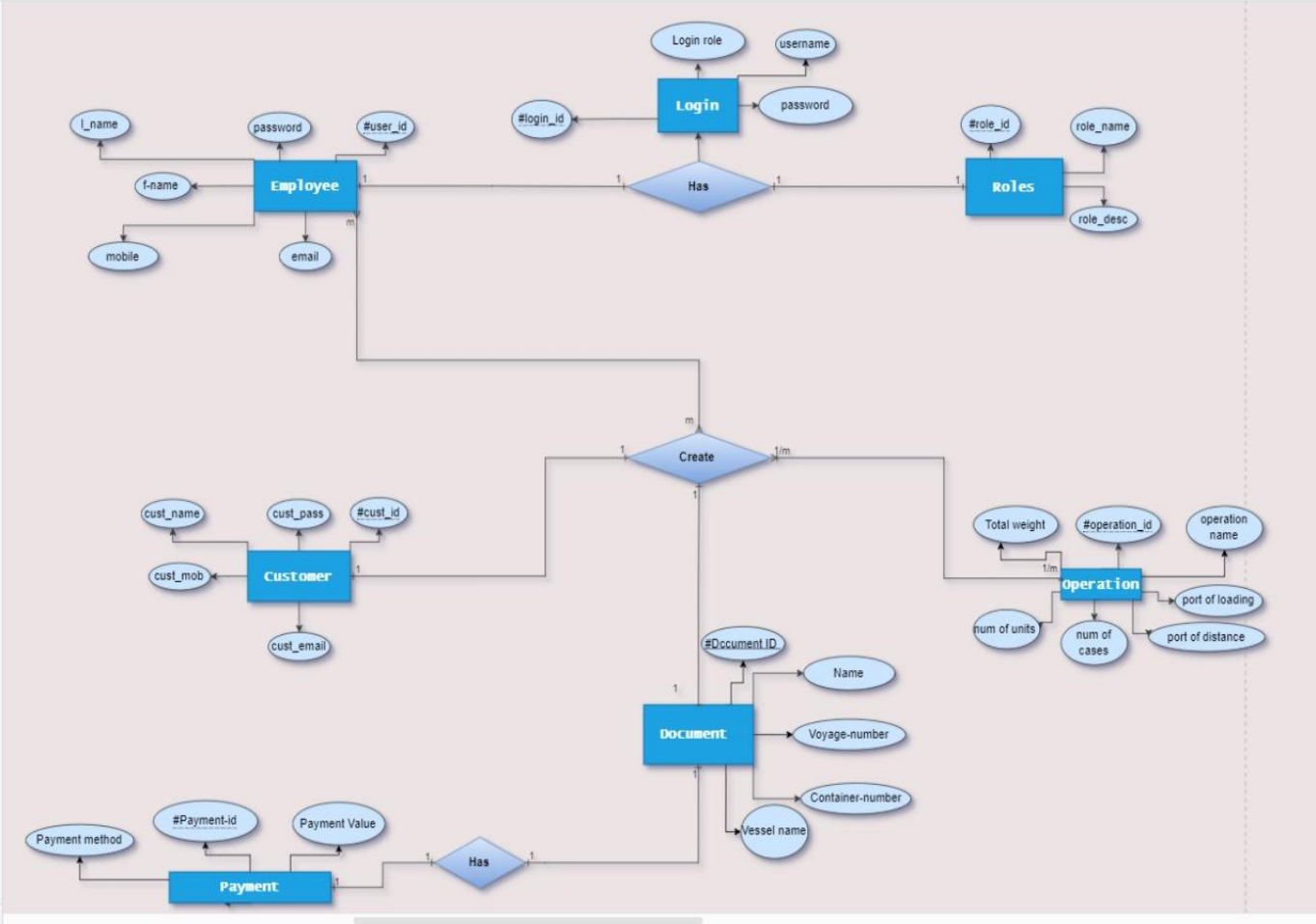
A DFD can show how data flows from user inputs, like shipment details and booking requests, to other processes, including shipment management, documentation, and interaction with external systems, in the context of a marine transportation website.

It facilitates comprehension of the system's internal transformations and data flow

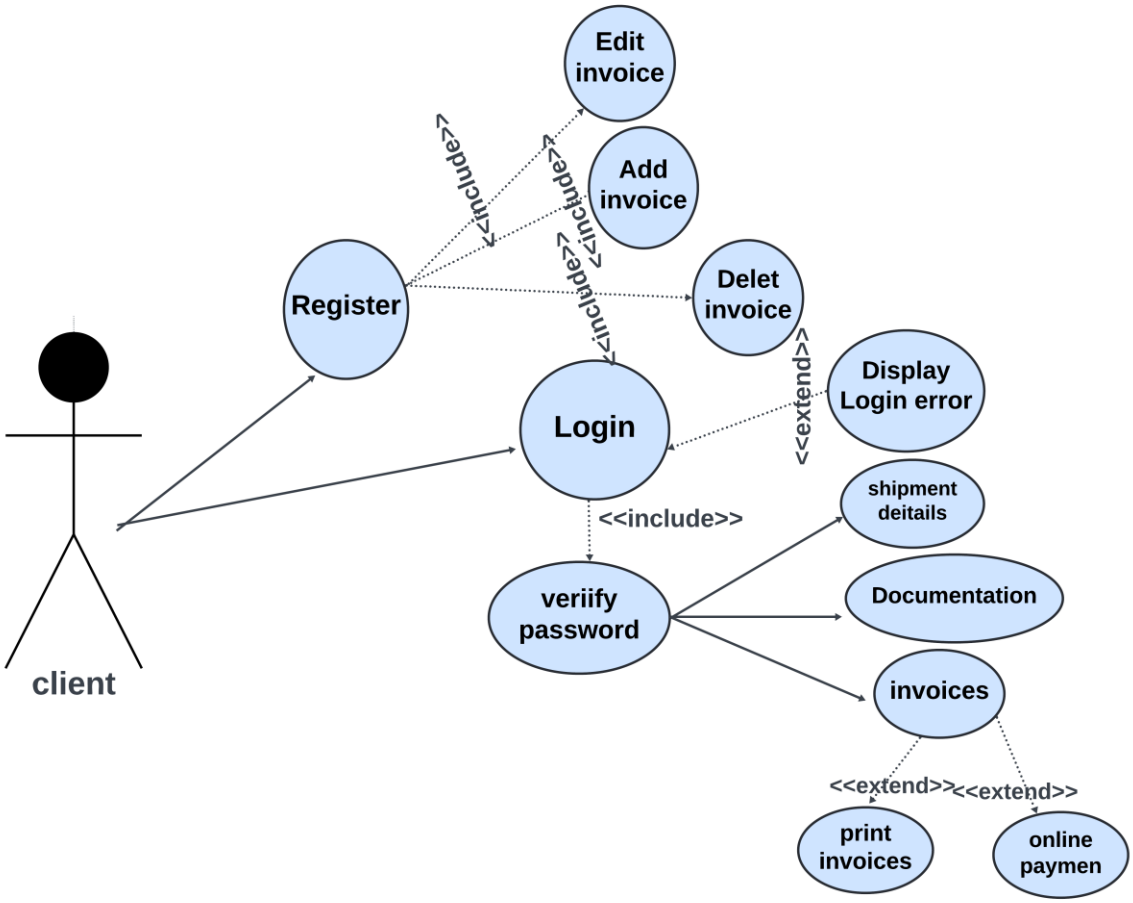


4.3 Entity Relationship Diagram:

Entity-Relationship Diagrams, or ERDs, show the connections between various system entities as well as the characteristics connected to those entities. An Entity Relationship Diagram (ERD) can show the relationships between entities like shipments, customers, carriers, containers, and documents in the context of a website for maritime transportation. It facilitates the creation of the database structure for the system and the comprehension of the connections between various components.

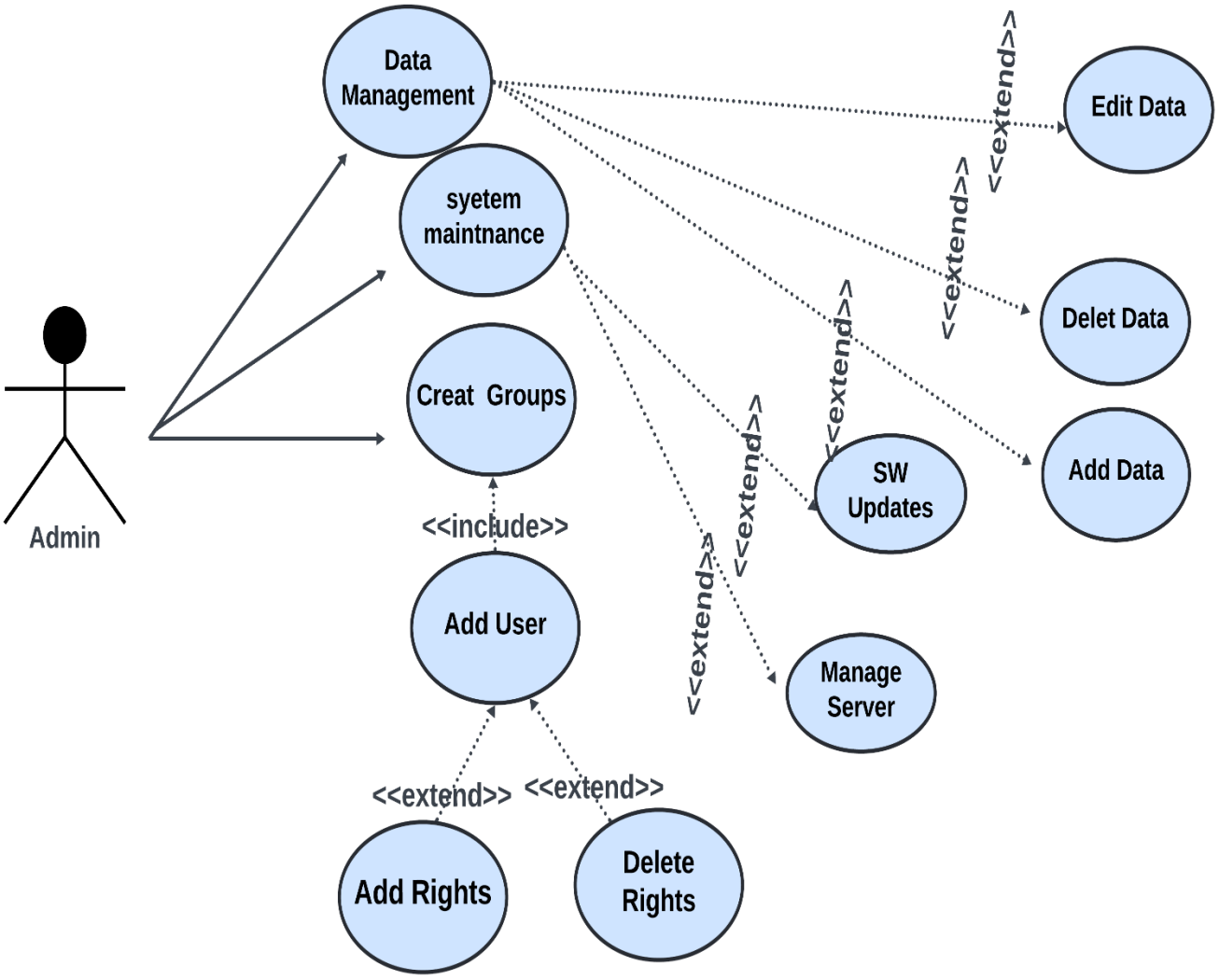


4.4 UML Use Case Diagram:



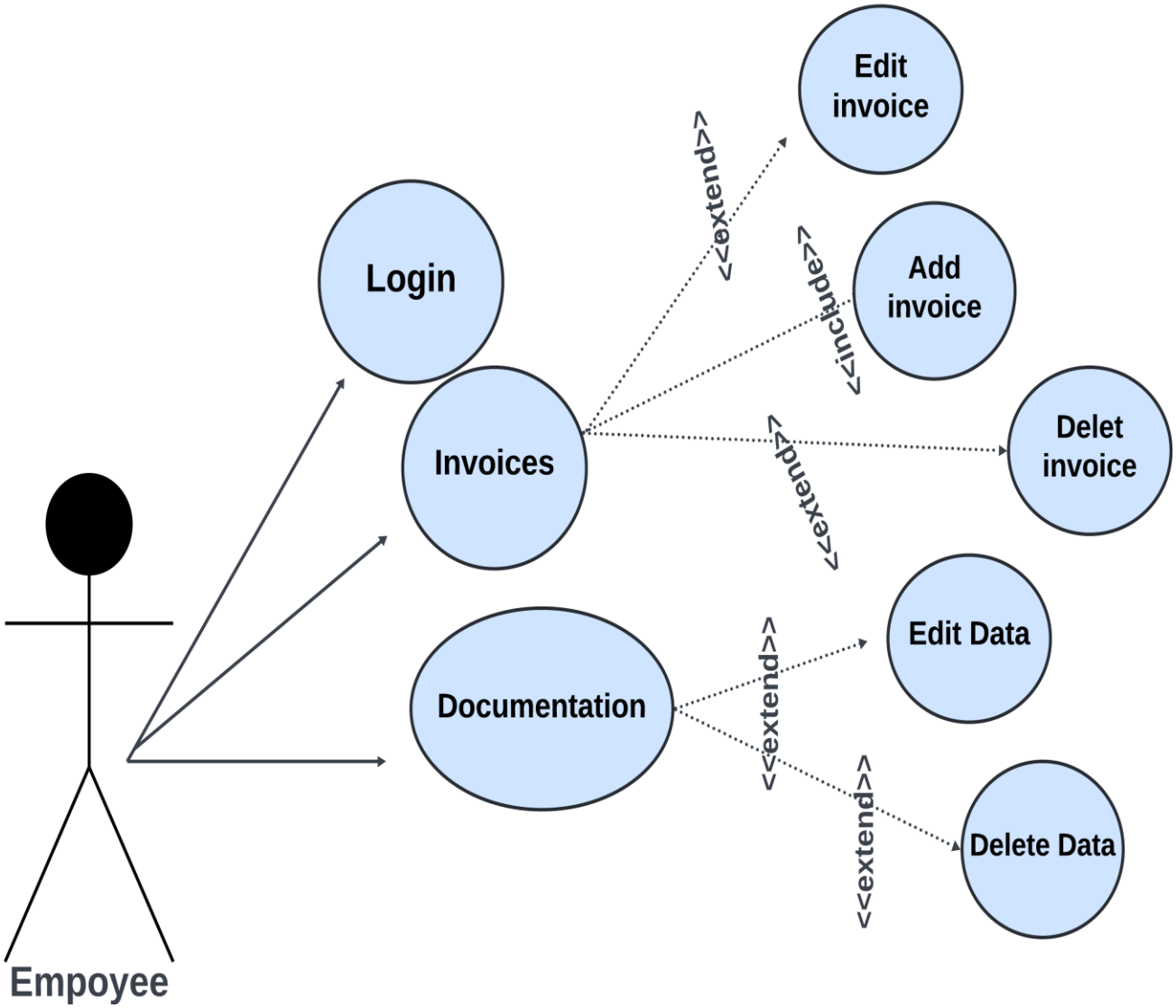
client interact with the system by adding his system (email & password) if use to create new user account click on sign up button, registration

form collects personal information from the user, such as their (name, email address, and a chosen username and password). The collected information is used to create a new user account in the system. A login form, on the other hand, usually only requires the user's username and password to authenticate their identity. After that you will be able to use our website.



Admin interact with the system by Managing data such as (adding, editing and deleting data),

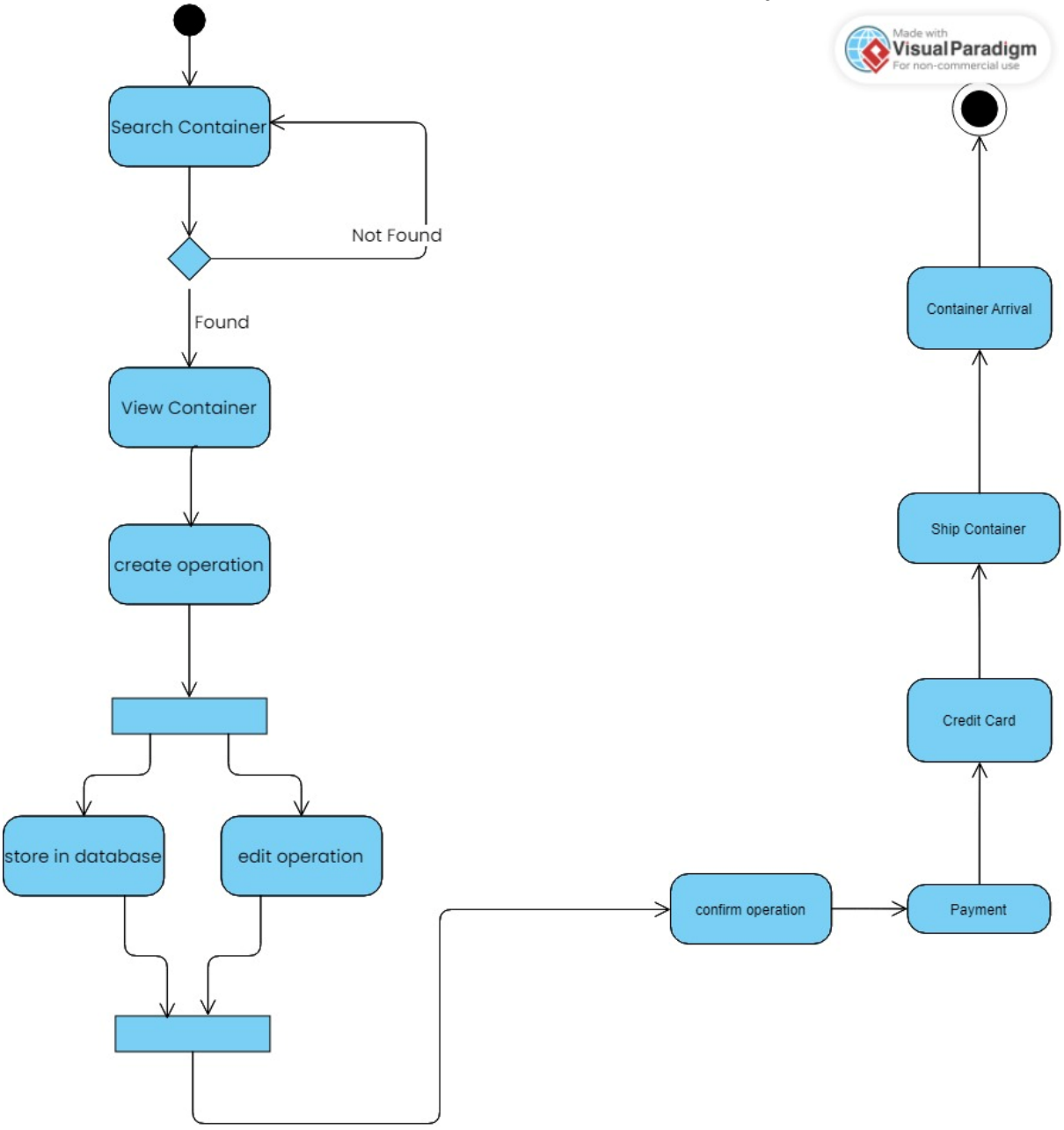
Manage server, updating SW & adding users, (adding or deleting rights).



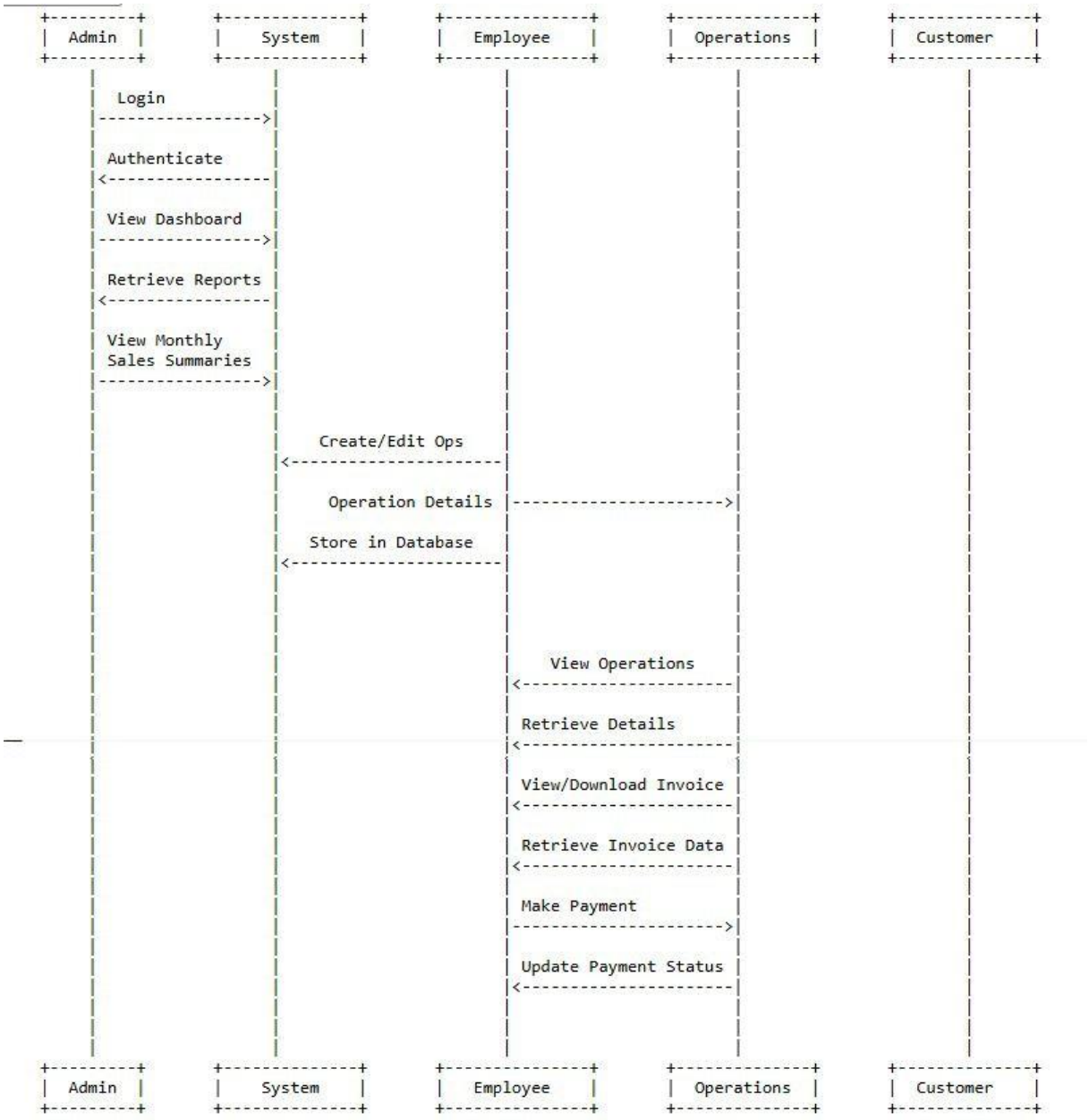
Employee interact with the system by (creating, editing, deleting) invoices .

Also in urgent cases he has the access to edit or delete a document or any data in the document.

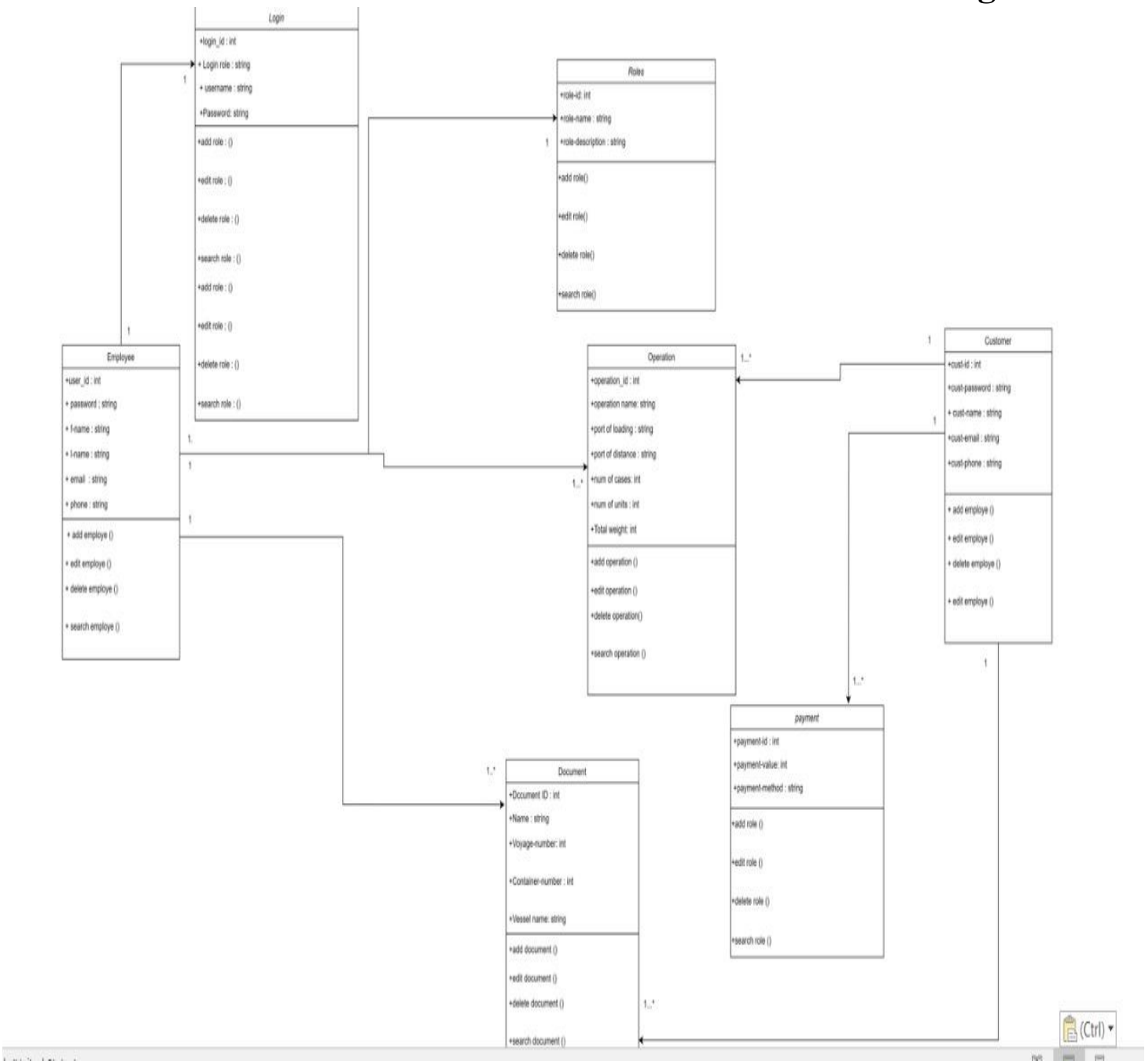
4.5 UML Activity Diagram:



4.6 UML Sequence Diagram:



4.7 UML Class Diagram:



4.8 GUI

A GUI (graphical User Interface) is a system of interactive visual components for computer software. A GUI displays objects that convey information, and represent actions tht can be taken by the uer. The objects change color , size or visibility when the user interacts with them.

GUI overview:

A GUI includes GUI objects, like icons, cursors, and buttons. These graphical elements are sometimes enhanced with sounds, or visual effects like transparency and drop shadows. Using these objects, a user can use the computer without having to know commands.

What are the elements of a GUI?

To make a GUI as user-friendly as possible, there are different elements and objects that the user use to interact with the software. Below is a list of each of these with a brief description.

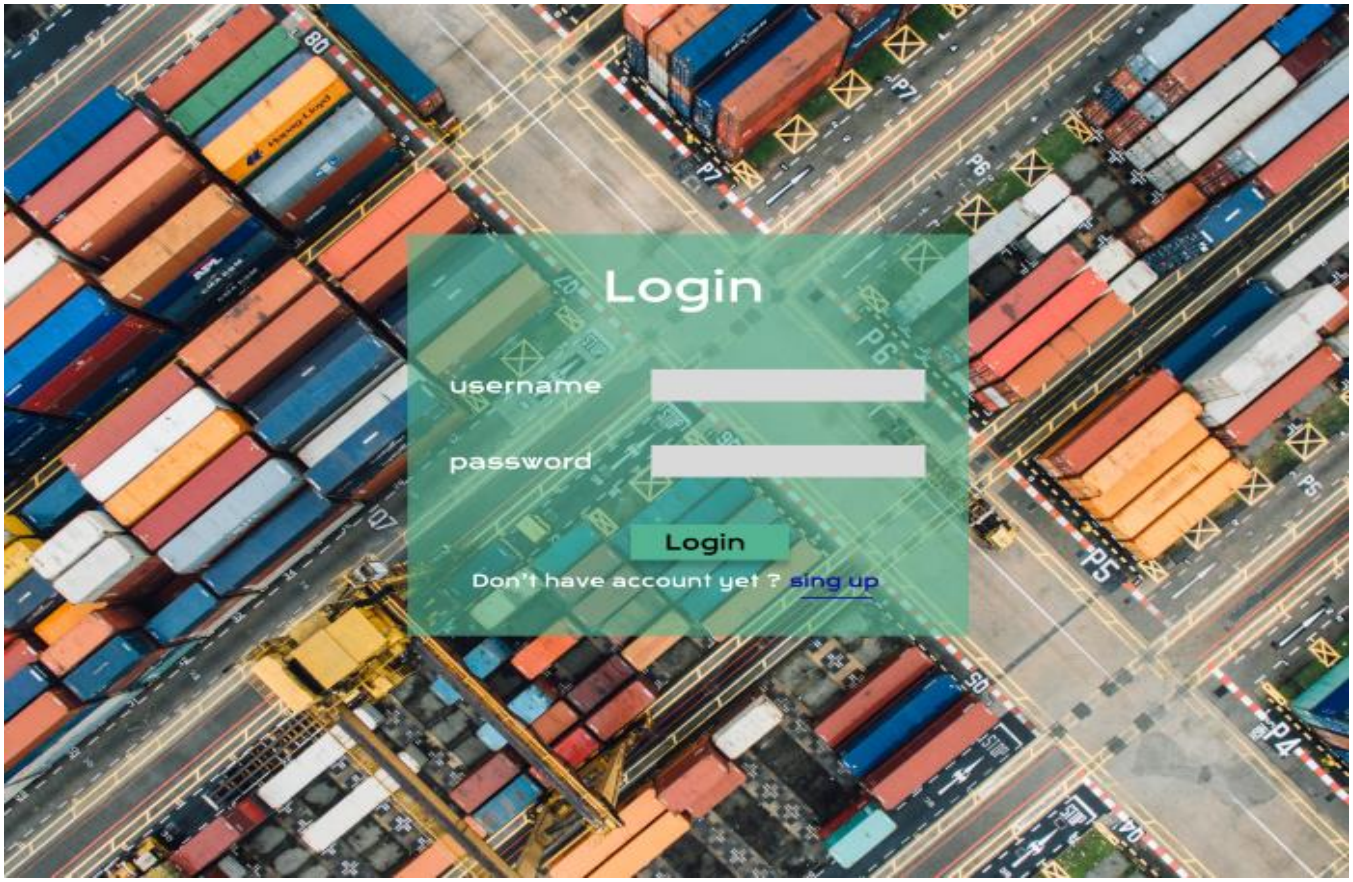
- **Button** - A graphical representation of a button that performs an action in a program when pressed

- **Dialog box** - A type of window that displays additional information, and asks a user for input.
- **Icon** - Small graphical representation of a program, feature, or file.
- **Menu** - List of commands or choices offered to the user through the menu bar.
- **Menu bar** - Thin, horizontal bar containing the labels of menus.
- **Ribbon** - Replacement for the file menu and toolbar that groups programs activities together.
- **Tab** - Clickable area at the top of a window that shows another page or area.
- **Toolbar** - Row of buttons, often near the top of an application window that controls software functions.
- **Window** - Rectangular section of the computer's display that shows the program currently being used.

How does a GUI work?

A GUI uses windows, icons, and menus to carry out commands, such as opening, deleting, and moving files. Although a GUI operating system is primarily navigated using a mouse, a keyboard can also be used via keyboard shortcuts or the arrow keys.

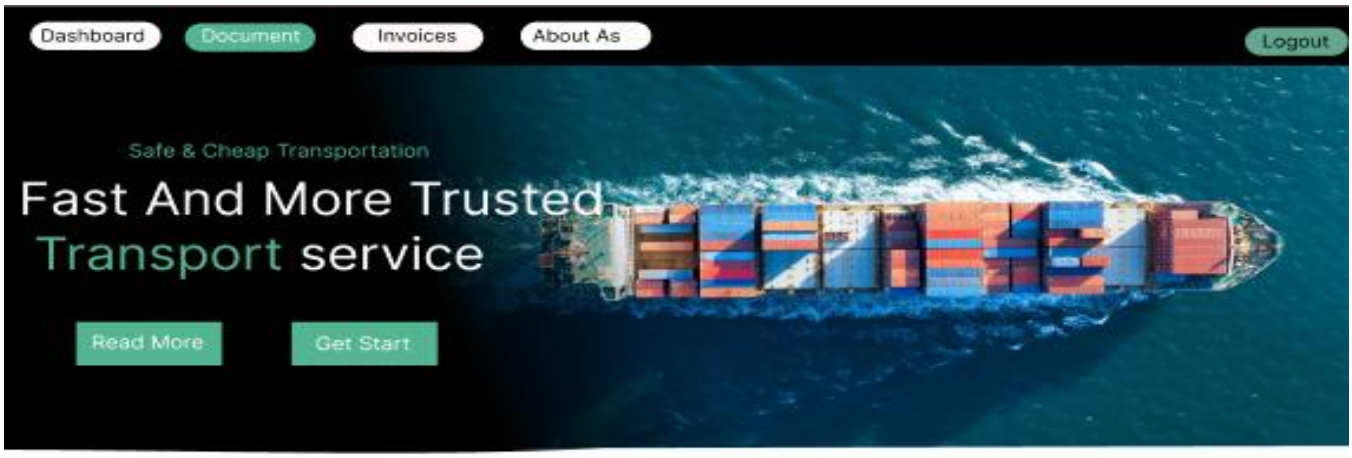
For example, if you want to open a program on a GUI system, you would move the mouse pointer to the program's icon and double-click it. With a command line interface, you need to know the commands to navigate to the directory containing the program, list the files, and then run the file

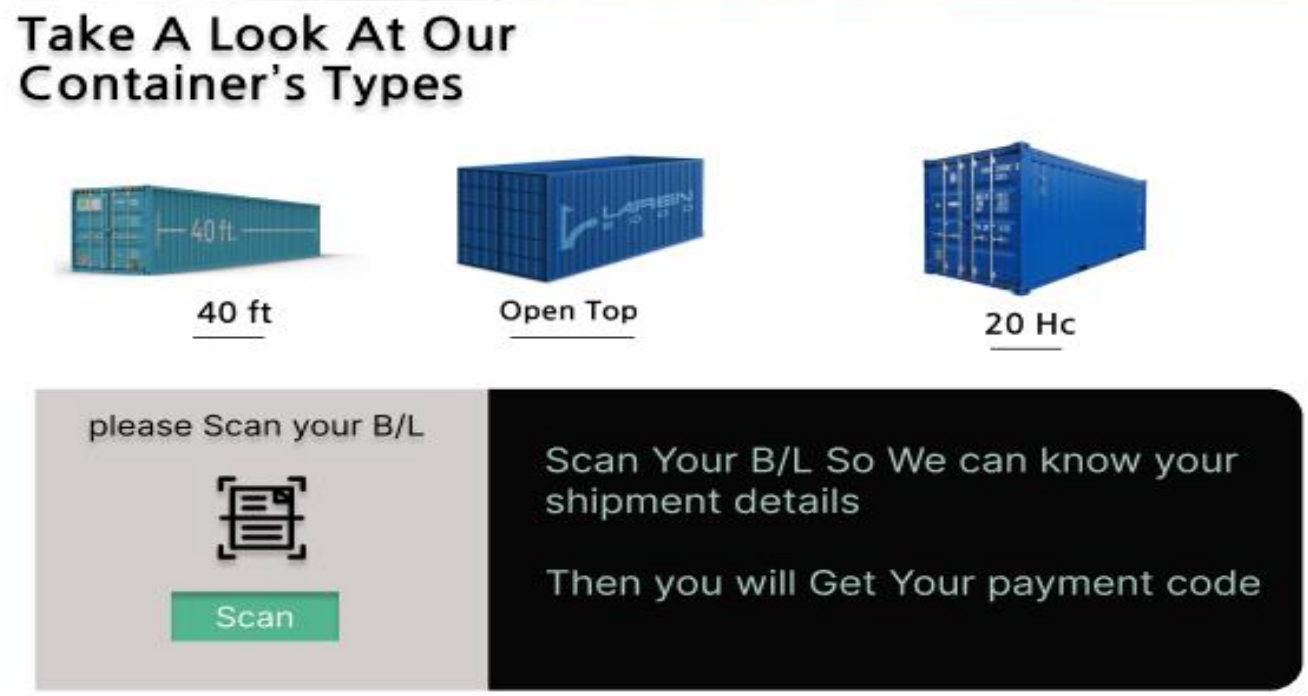


Slide no 1

Welcome to Speedy Logistics Invoices system

In this system we are trying to make the payment process more easier and faster , system will target employees and clients
so both of them must have account on our system if they don't have an account they have to clink on sing up tap otherwise they can put their account username & password

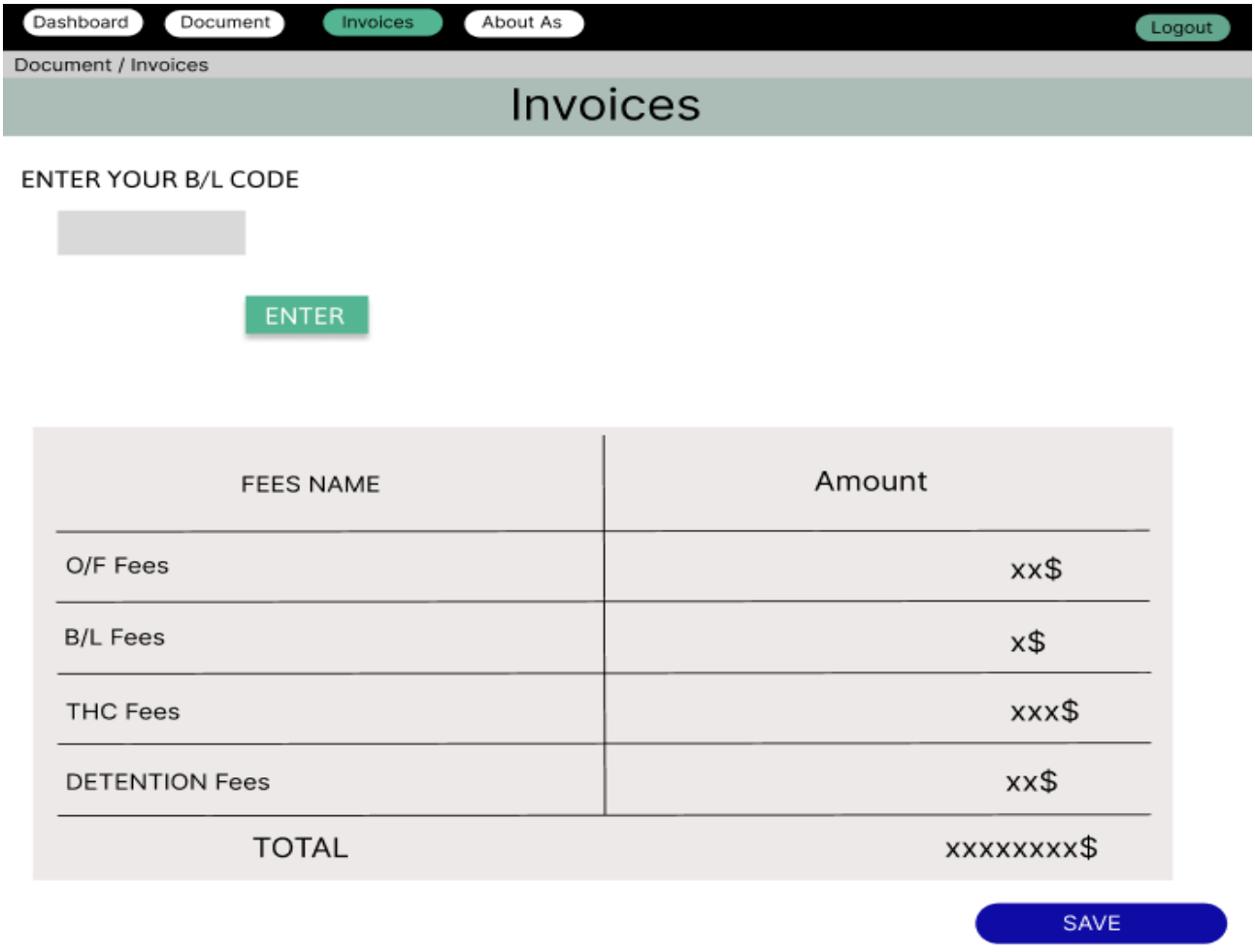




Slide no 2

Document page

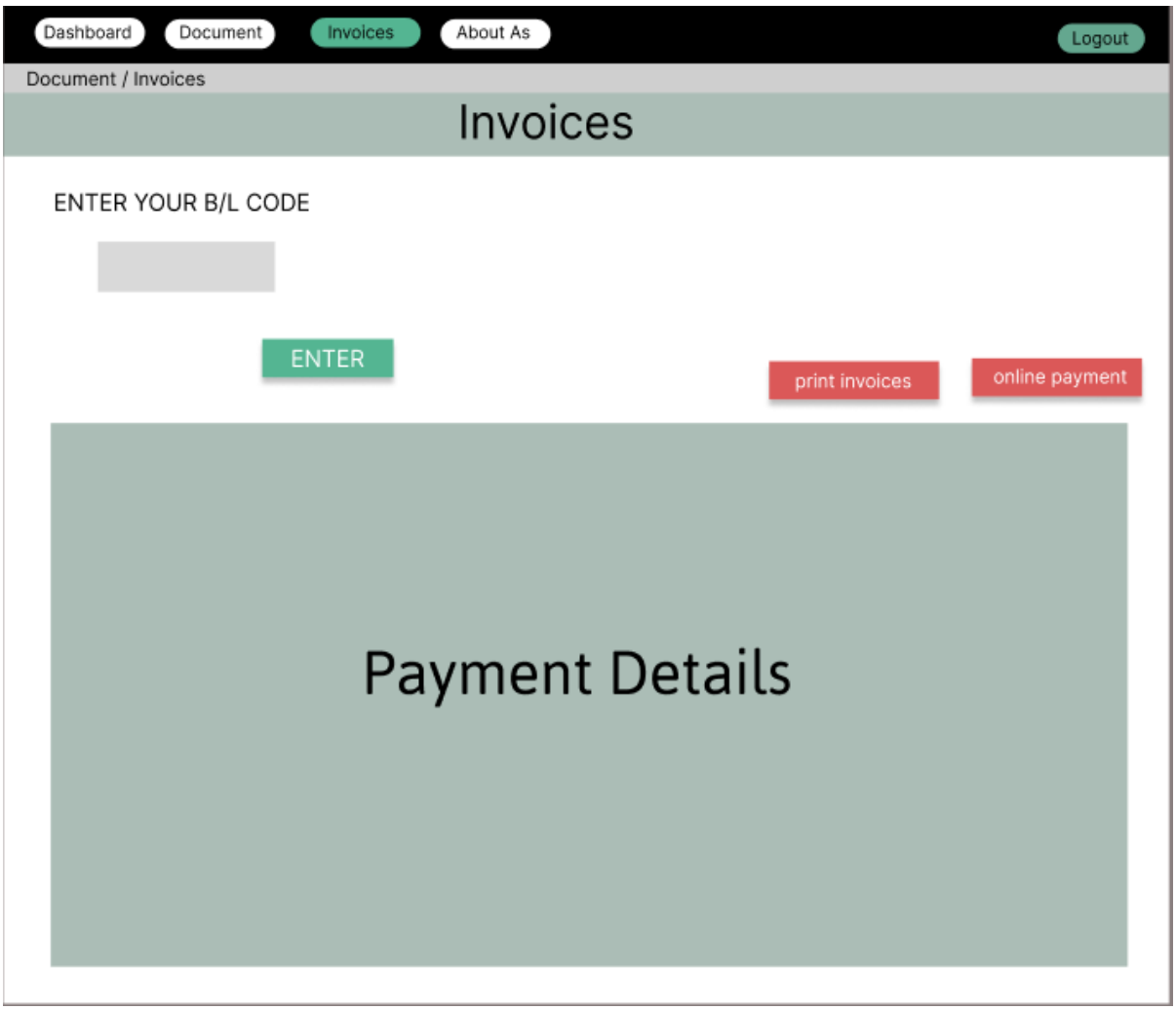
Both users will have access in this page
for employee they will scan B/Ls to create clients invoices , for clients they will scan B/Ls to get their invoices so they can pay it .



Slide no 3

Invoices page

Both users will have access in this page but with different uses.
for employee they will create clients' invoices.



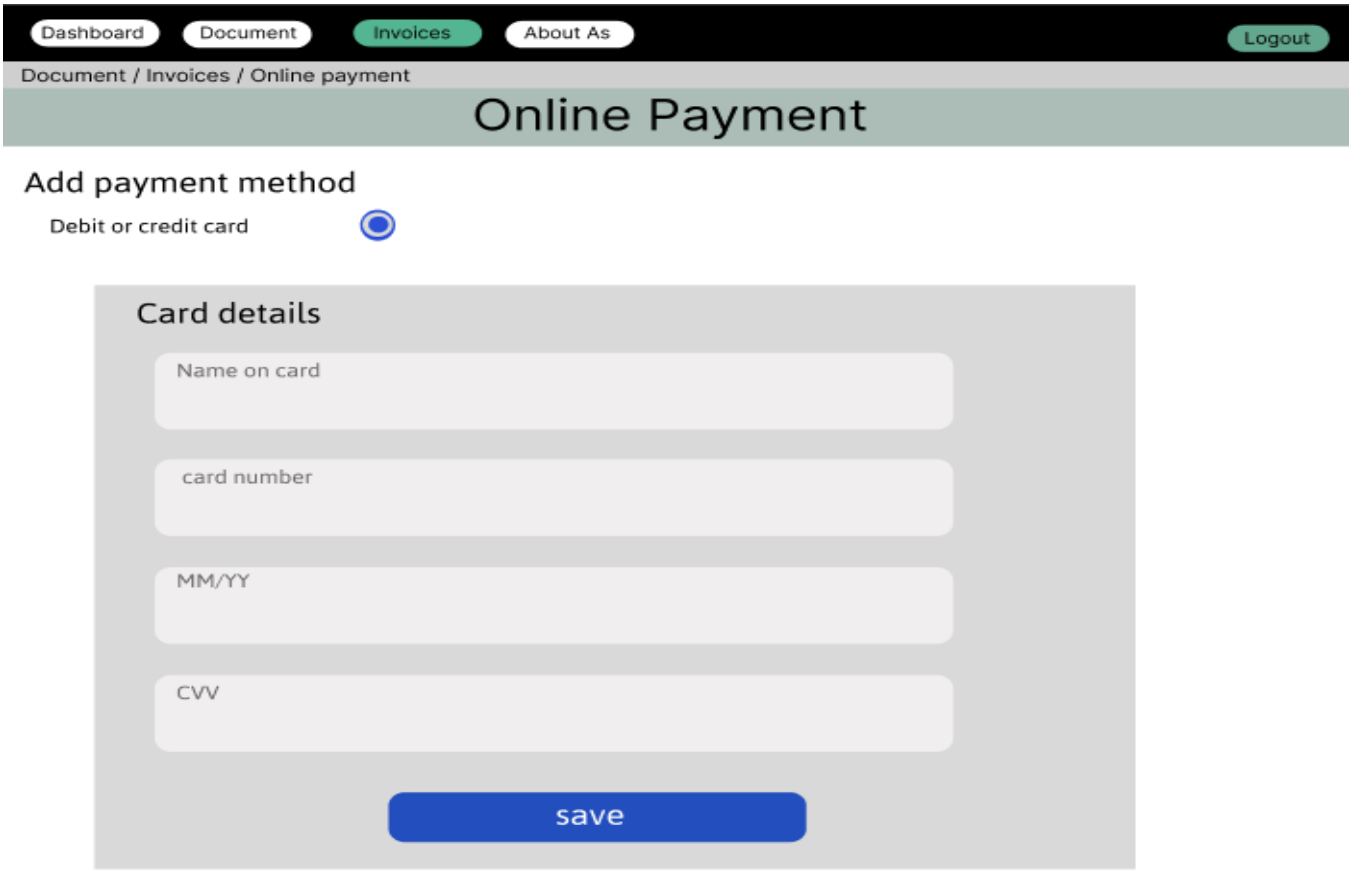
Slide no 4

Invoices page

For clients they will be able to see their payment details and then choose if

they want to pay in the bank so they will only have to print their invoices or online payment

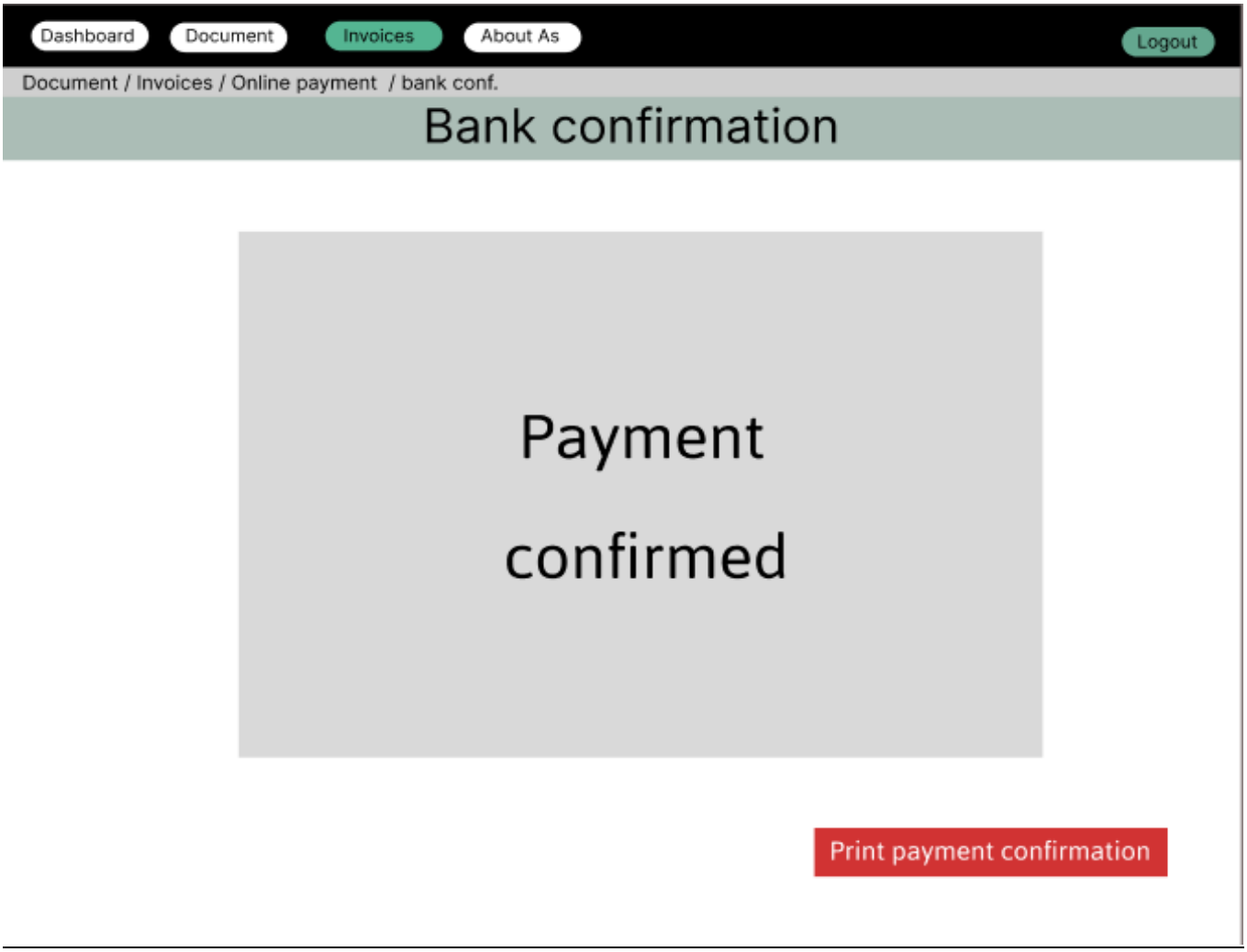
(this button will take them to slide no 5)



Slide no 5

Online payment page

Only clients will have access in this page

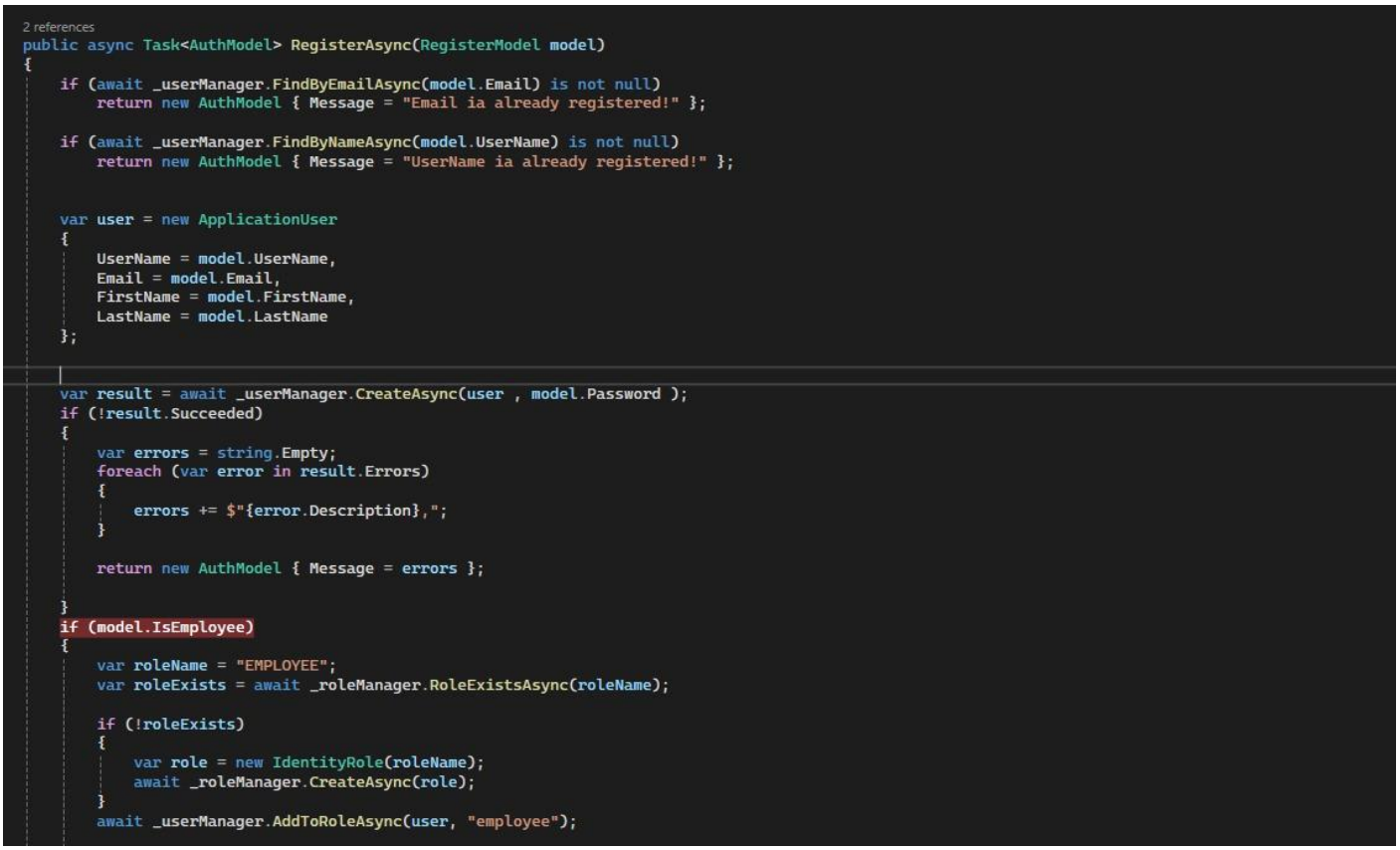


Slide no 6

Bank confirmation page

Both users will have access in this page

CHAPTER FIVE: SYSTEM IMPLEMENTATION :



```
2 references
public async Task<AuthModel> RegisterAsync(RegisterModel model)
{
    if (await _userManager.FindByEmailAsync(model.Email) is not null)
        return new AuthModel { Message = "Email ia already registered!" };

    if (await _userManager.FindByNameAsync(model.UserName) is not null)
        return new AuthModel { Message = "UserName ia already registered!" };


    var user = new ApplicationUser
    {
        UserName = model.UserName,
        Email = model.Email,
        FirstName = model.FirstName,
        LastName = model.LastName
    };

    var result = await _userManager.CreateAsync(user , model.Password );
    if (!result.Succeeded)
    {
        var errors = string.Empty;
        foreach (var error in result.Errors)
        {
            errors += $"{error.Description},";
        }

        return new AuthModel { Message = errors };

    }
    if (model.IsEmployee)
    {
        var roleName = "EMPLOYEE";
        var roleExists = await _roleManager.RoleExistsAsync(roleName);

        if (!roleExists)
        {
            var role = new IdentityRole(roleName);
            await _roleManager.CreateAsync(role);
        }
        await _userManager.AddToRoleAsync(user, "employee");
```

This method handles the registration of a new user, including checking for existing email and username, creating the user, assigning roles, generating a JWT token, and returning the authentication details. It leverages ASP.NET Core Identity for user and role management and JWT for token generation.

```
2 references
public async Task<OperationResult<List<UsersDTO>>> GetUsersInRole()
{
    var usersInRole = new List<UsersDTO>();

    var role = await _roleManager.FindByNameAsync("user");
    if (role == null)
    {
        return usersInRole;
    }

    var userRoles = await _userManager.GetUsersInRoleAsync("user");
    if (userRoles == null)
    {
        return new OperationResult<List<UsersDTO>>
        {
            StatusCode = HttpStatusCode.NotFound,
            ErrorMessageKey = LocalizationKeys.DataNotFound
        };
    }
    var Users = userRoles.ToList();
    var UsersDTO = new List<UsersDTO>();
    foreach (var userRole in userRoles)
    {

        var User = new UsersDTO
        {
            Id = userRole.Id,
            Name = userRole.UserName,

        };

        UsersDTO.Add(User);
    }

    return new OperationResult<List<UsersDTO>>
    {
        Data = UsersDTO,
        StatusCode = HttpStatusCode.OK
    };

}
```

The method GetUsersInRole performs the following steps:

Checks if the role "user" exists.

Retrieves users associated with the "user" role.

Maps each user to a UsersDTO object.

Returns the list of UsersDTO objects within an OperationResult.

```
2 references
public async Task<OperationResult<string>> AddOperation(OperationDTO model)
{
    var result = new OperationResult<string>();

    var OperationsRequest = new Operations();

    var UserId = _userDataProvider.GetUserId();

    if (model.ID != null)
    {

        var OperationDb = await _applicationDbContext.Operation.Where(x => x.ID == model.ID).FirstOrDefaultAsync();



        OperationDb.Name = model.Name;
        OperationDb.NumberOfCases = model.NumberOfCases;
        OperationDb.PortOfDistance = model.PortOfDistance;
        OperationDb.PortOfLoading = model.PortOfLoading;
        OperationDb.NumberOfUnits = model.NumberOfUnits;
        OperationDb.TotalWeight = model.TotalWeight;
        OperationDb.UserId = model.UserId;
        OperationDb.UpdatedAt = DateTime.Now;
        OperationDb.IsDeleted = model.IsDeleted;
        OperationDb.IsPaid = model.IsPaid;
        OperationDb.EmployeeUserId = model.EmployeeId;
        await _applicationDbContext.SaveChangesAsync();

        result.Data = model.ID.ToString();


    }
    else // Add Mode
    {
        OperationsRequest = new Operations
        {
            Name = model.Name,
            NumberOfCases = model.NumberOfCases,
            PortOfLoading = model.PortOfLoading,
            IsDeleted = false,
            PortOfDistance = model.PortOfDistance,
            NumberOfUnits = model.NumberOfUnits,
            TotalWeight = model.TotalWeight,
            UserId = model.UserId,
```

The Add Operation method performs the following steps:

Initializes the result and operation request objects.

Retrieves the current user ID.

Checks if the operation is in update mode or add mode.

Updates an existing operation if model.ID is not null.

Adds a new operation if model.ID is null.

Saves changes to the database.

Sets the status code to OK and returns the result.

```
2 references
public async Task<OperationResult<bool>> DeleteOperation(Guid id)
{
    OperationResult<bool> result = new OperationResult<bool>();

    var updatedRecord = await _applicationDbContext.Operation.Where(x => x.ID == id).FirstOrDefaultAsync();

    if (updatedRecord == null)
    {
        result.StatusCode = HttpStatusCode.NotFound;
        result.ErrorMessageKey = LocalizationKeys.DataNotFound;
        return result;
    }
    updatedRecord.IsDeleted = true;


    await _applicationDbContext.SaveChangesAsync();

    result.StatusCode = HttpStatusCode.OK;
    return result;
}
```

The Delete Operation method performs the following steps:

Initializes the result object.

Retrieves the operation record by its ID.

Checks if the operation exists:

If the operation does not exist, it sets the status code to NotFound and returns the result with an appropriate error message.

Marks the operation as deleted by setting the IsDeleted property to true.

Saves the changes to the database.

Sets the status code to OK, sets the data to true indicating success, and returns the result

```
public async Task<OperationResult<OperationDTO>> GetOperation(Guid id)
{
    var Operation = await _applicationDbContext.Operation
                                            .Where(o => o.ID == id)
                                            .FirstOrDefaultAsync();

    if (Operation == null)
    {
        return new OperationResult<OperationDTO>
        {
            StatusCode = HttpStatusCode.NotFound,
            ErrorMessageKey = LocalizationKeys.DataNotFound
        };
    }

    var document = await _applicationDbContext.Document
                                            .Where(d => d.OperationID == id)
                                            .FirstOrDefaultAsync();


    var operationDTO = new OperationDTO
    {
        ID = Operation.ID,
        Name = Operation.Name,
        NumberOfCases = Operation.NumberOfCases,
        NumberOfUnits = Operation.NumberOfUnits,
        PortOfDistance = Operation.PortOfDistance,
        PortOfLoading = Operation.PortOfLoading,
        UserId = Operation.UserId,
        CreatedBy = Operation.CreatedBy,
        CreatedAt = Operation.CreatedAt,
        TotalWeight = Operation.TotalWeight,
        IsDeleted = Operation.IsDeleted,
        IsPaid = Operation.IsPaid,
        EmployeeId = Operation.EmployeeUserId,
        Documents = document != null ? new Dtos.Documents
        {
            ID = document.ID,
            OperationID = document.OperationID,
            Name = document.Name,
            VoyageNumber = document.VoyageNumber,
            ContainerNumber = document.ContainerNumber,
            ShipID = document.ShipID,
            IsDeleted = document.IsDeleted
```

Retrieve the Operation by ID: Fetches the operation from the database using the provided ID.

Check if the Operation Exists: If the operation is not found, returns a result with NotFound status and an appropriate error message.

Retrieve the Associated Document: Fetches the document associated with the operation using the operation's ID.

Create and Populate OperationDTO: Creates a data transfer object (OperationDTO) with the operation's details.

If a document is found, it is also included in the operationDTO. Return the Result:Returns the OperationDTO wrapped in an OperationResult with an OK status.

```
public async Task<OperationResult<List<OperationDTO>>> GetUserOperations(Guid UserId)
    {
        var operations = await _applicationDbContext.Operation
                                        .Where(o => o.UserId == UserId.ToString() || o.EmployeeUserId == UserId.ToString())
                                        .ToListAsync();

        if (operations == null || !operations.Any())
        {
            return new OperationResult<List<OperationDTO>>
            {
                StatusCode = HttpStatusCode.OK,
                Data = null
            };
        }

        var operationDTOs = new List<OperationDTO>();

        foreach (var operation in operations)
        {
            var document = await _applicationDbContext.Document
                                                    .Where(d => d.OperationID == operation.ID)
                                                    .FirstOrDefaultAsync();

            var operationDTO = new OperationDTO
            {
                ID = operation.ID,
                Name = operation.Name,
                NumberOfCases = operation.NumberOfCases,
                NumberOfUnits = operation.NumberOfUnits,
                PortOfDistance = operation.PortOfDistance,
                PortOfLoading = operation.PortOfLoading,
                UserId = operation.UserId,
                CreatedBy = operation.CreatedBy,
                CreatedAt = operation.CreatedAt,
                TotalWeight = operation.TotalWeight,
                IsDeleted = operation.IsDeleted,
                IsPaid = operation.IsPaid,
                EmployeeId = operation.EmployeeUserId,
                Documents = document != null ? new Dtos.Documents
                {
                    ID = document.ID,
                    OperationID = document.OperationID,
                    Name = document.Name,
                    VoyageNumber = document.VoyageNumber,
                    ContainerNumber = document.ContainerNumber,
```

Retrieve Operations by User ID:

Fetches all operations from the database where the user is either the owner (UserId) or the employee (EmployeeUserId).

Check if Operations Exist:

If no operations are found, returns an OperationResult with an OK status and null data.

Create and Populate OperationDTO List:

Initializes a list of OperationDTO.

For each operation, retrieves the associated document if available.

Creates an OperationDTO object with the operation and document details, then adds it to the list.

Return the Result:

Returns the list of OperationDTO wrapped in an OperationResult with an OK status.

```
0 references
public async Task<IEnumerable<MonthlyProfitDto>> GetMonthlyProfitsAsync()
{

    var monthlyProfits = await _applicationDbContext.Operation_Payment
        .Where(op => !op.IsDeleted)
        .GroupBy(op => new { op.CreatedAt.Year, op.CreatedAt.Month })
        .Select(g => new MonthlyProfitDto
        {
            Year = g.Key.Year,
            Month = g.Key.Month,
            TotalProfit = g.Sum(op => op.PaymentValue),
            OperationCount = g.Count()  // New property to count the operations

        })
        .OrderBy(mp => mp.Year)
        .ThenBy(mp => mp.Month)
        .ToListAsync();

    return monthlyProfits;
}
```

The GetMonthlyProfitsAsync method calculates the total profits and the count of operations for each month. It retrieves this data from the Operation_Payment table, filters out deleted records, groups the data by year and month, and then calculates the total profit and the count of operations for each group. The results are ordered by year and month.

```
2 references
public async Task<IEnumerable<MonthlyProfitDto>> GetMonthlyProfitsByYearAsync(int year)
{
    // Generate a list of all months for the specified year
    var allMonths = GenerateAllMonths(year, year);

    // Get the monthly profits for the specified year from the database
    var monthlyProfits = await _applicationDbContext.Operation_Payment
        .Where(op => !op.IsDeleted && op.CreatedAt.Year == year)
        .GroupBy(op => new { op.CreatedAt.Year, op.CreatedAt.Month })
        .Select(g => new MonthlyProfitDto
        {
            Year = g.Key.Year,
            Month = g.Key.Month,
            TotalProfit = g.Sum(op => op.PaymentValue),
            OperationCount = g.Count()
        })
        .ToListAsync();

    // Perform a left join to include all months
    var result = from m in allMonths
                 join p in monthlyProfits
                 on new { m.Year, m.Month } equals new { p.Year, p.Month } into gj
                 from sub in gj.DefaultIfEmpty()
                 select new MonthlyProfitDto
                 {
                     Year = m.Year,
                     Month = m.Month,
                     TotalProfit = sub?.TotalProfit ?? 0,
                     OperationCount = sub?.OperationCount ?? 0
                 };

    return result.OrderBy(r => r.Year).ThenBy(r => r.Month).ToList();
}
```

The GetMonthlyProfitsByYearAsync method retrieves the total profits and count of operations for each month of a specified year. It ensures that all months of the year are included in the result, even if there were no operations in some months, by performing a left join with a list of all months.

```
2 references
public async Task<OperationResult<StatisticsDTO>> GetStatistics()
{
    var EmployeeRoleId = await _applicationDbContext.Roles.Where(x=>x.Name == "EMPLOYEE").Select(x=>x.Id).FirstOrDefaultAsync();
    var CientRoleId = await _applicationDbContext.Roles.Where(x=>x.Name == "user").Select(x => x.Id).FirstOrDefaultAsync();

    var clientCount = await _applicationDbContext.UserRoles.CountAsync(u => u.RoleId == CientRoleId);
    var EmployeeCount = await _applicationDbContext.UserRoles.CountAsync(u => u.RoleId == EmployeeRoleId);

    var paidOperations = await _applicationDbContext.Operation.Where(x=> x.IsPaid == true).CountAsync();
    var UnpaidOperations = await _applicationDbContext.Operation.Where(x=> x.IsPaid == false).CountAsync();


    var StatisticsDTO = new StatisticsDTO
    {
        ClientsCount = clientCount,
        EmployeeCount  = EmployeeCount,
        UnPaidOperationCount = UnpaidOperations,
        PaidOperationCount = paidOperations,

    };

    return new OperationResult<StatisticsDTO>
    {
        Data = StatisticsDTO,
        StatusCode = HttpStatusCode.OK
    };

}
```

The GetStatistics method retrieves various statistical data from the database, including the count of users with the "EMPLOYEE" and "user" roles, as well as the count of paid and unpaid operations. It returns this data encapsulated in a StatisticsDTO object.

```
public async Task<OperationResult<string>> Adddocument(DocumentsDTO model)
{
    var result = new OperationResult<string>();

    var DocumentRequest = new Models.Documents();

    if (model.ID != null)
    {

        var DocumentDb = await _applicationDbContext.Document.Where(x => x.ID == model.ID).FirstOrDefaultAsync();

        DocumentDb.Name = model.Name;
        DocumentDb.ShipID = model.ShipID;
        DocumentDb.VoyageNumber = model.VoyageNumber;
        DocumentDb.ContainerNumber = model.ContainerNumber;
        DocumentDb.OperationID = model.OperationID;
        await _applicationDbContext.SaveChangesAsync();

        result.Data = model.ID.ToString();


    }
    else // Add Mode
    {

        DocumentRequest = new Models.Documents
        {
            Name = model.Name,
            ShipID = model.ShipID,
            VoyageNumber = model.VoyageNumber,
            ContainerNumber = model.ContainerNumber,
            OperationID = model.OperationID

        };

        _applicationDbContext.Document.Add(DocumentRequest);
        await _applicationDbContext.SaveChangesAsync();

        result.Data = DocumentRequest.ID.ToString();
    }
```

Summary:

The method is called Adddocument and takes a DocumentsDTO model as input.

It creates a new OperationResult<string> to hold the result of the operation.

If the provided DocumentsDTO model has an ID, it updates the existing document with the corresponding ID.

It fetches the document from the database based on the provided ID.

It updates the document properties with the values from the provided model.

It saves the changes to the database.

If the provided DocumentsDTO model doesn't have an ID, it adds a new document to the database.

It creates a new Documents object with the properties from the provided model.

It adds the new document to the database context.

It saves the changes to the database.

Finally, it sets the Data property of the OperationResult to the ID of the added or updated document.

It sets the StatusCode of the OperationResult to HttpStatusCode.OK

```
2 references
public async Task<OperationResult<string>> AddPayment(OperationPayment model)
{

    var result = new OperationResult<string>();

    var PaymentRequest = new Models.OperationPayment();

    var operation = await _applicationDbContext.Operation.Where(x => x.ID == model.OperationID).FirstOrDefaultAsync();

    operation.IsPaid = true;

    PaymentRequest = new Models.OperationPayment
    {
            OperationID = model.OperationID,
            PaymentValue = model.PaymentValue,
            CreatedAt = DateTime.Now,
    };

     _applicationDbContext.Operation_Payment.Add(PaymentRequest);
     await _applicationDbContext.SaveChangesAsync();

    result.Data = PaymentRequest.ID.ToString();

    result.StatusCode = HttpStatusCode.OK;
    return result;

}
```

Input: The method accepts an OperationPayment model representing the payment information.

Initialization: It initializes a new OperationResult<string> to hold the result of the operation.

Retrieve Operation: It fetches the operation associated with the provided OperationID from the database.

Mark Operation as Paid: It updates the IsPaid property of the fetched operation to true, indicating that the operation has been paid for.

Create Payment Request: It creates a new OperationPayment object based on the provided payment model.

Add Payment to Database: It adds the new payment request to the Operation_Payment table in the database.

Save Changes: It saves the changes to the database context.

Set Result Data: It sets the Data property of the OperationResult to the ID of the added payment request.

Set Status Code: It sets the StatusCode of the OperationResult to HttpStatusCode.OK.

Return Result: It returns the OperationResult<string> containing the ID of the added payment request.

```
2 references
public async Task<OperationResult<DocumentsDTO>> GetDocument(Guid id)
{
    var Document = await _applicationDbContext.Document.Where(x=> x.ID == id && x.IsDeleted != true).FirstOrDefaultAsync();



    if (Document == null)
    {
        return new OperationResult<DocumentsDTO>
        {
            StatusCode = HttpStatusCode.NotFound,
            ErrorMessageKey = LocalizationKeys.DataNotFound
        };
    }

    return new OperationResult<DocumentsDTO>
    {
        Data = new DocumentsDTO
        {
                ID = Document.ID,
                Name = Document.Name,
                ContainerNumber = Document.ContainerNumber,
                OperationID = Document.OperationID,
                ShipID = Document.ShipID,
                VoyageNumber = Document.VoyageNumber,
               IsDelete = Document.IsDeleted



        },
        StatusCode = HttpStatusCode.OK
    };
}
```

Input: The method takes a Guid parameter representing the ID of the document to retrieve.

Retrieve Document: It queries the database to fetch the document with the provided ID. It also ensures that the document is not marked as deleted (IsDeleted != true).

Handle Not Found: If no document is found with the provided ID or if the document is marked as deleted, the method returns an OperationResult with a status code of NotFound.

Return Document Information: If the document is found, the method constructs a DocumentsDTO object containing the relevant document information (such as ID, name, container number, operation ID, ship ID, voyage number, and deletion status).

Set Status Code: It sets the status code of the OperationResult to OK.

Return Result: It returns the OperationResult<DocumentsDTO> containing either the document information or an error message if the document is not found.

```
2 references
public async Task<OperationResult<DocumentsDTO>> GetUserDocuments(Guid UserId)
{
    OperationResult<DocumentsDTO> result =
        new OperationResult<DocumentsDTO>();


    var  userId = _userDataProvider.GetUserId().ToString();

    var OperationId = _applicationDbContext.Operation.Where(x=> x.UserId == userId).Select(x=> x.ID).FirstOrDefault();

    var Document = _applicationDbContext.Document.Where(x => x.OperationID == OperationId && x.IsDeleted != true).FirstOrDefault();


    var DocumentObj = new DocumentsDTO();

    DocumentObj.ID = Document.ID;
    DocumentObj.VoyageNumber = Document.VoyageNumber;
    DocumentObj.ContainerNumber = Document.ContainerNumber;
    DocumentObj.OperationID = Document.OperationID;
    DocumentObj.ShipID = Document.ShipID;
    DocumentObj.Name = Document.Name;
  DocumentObj.IsDelete = Document.IsDeleted;

    result = DocumentObj;
    result.StatusCode = HttpStatusCode.OK;
    return result;
}
```

Input: The method takes a Guid parameter representing the user ID.

Retrieve User ID: It fetches the current user's ID from _userDataProvider.

Find Operation ID: It queries the Operation table to find an operation associated with the user ID.

Find Document: It then queries the Document table to find a document associated with the retrieved operation ID, ensuring that the document is not marked as deleted (IsDeleted != true).

Handle Not Found: If no document is found, it should handle this scenario appropriately (the current implementation does not handle this case properly).

Return Document Information: It constructs a DocumentsDTO object with the document details and sets the status code to OK.

Return Result: It returns the OperationResult<DocumentsDTO> containing the document information or an appropriate error message if no document is found.

```
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

namespace CompanyAPIs.Data
{
    21 references
    public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
    {

        0 references
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)

        {

        }

        9 references
        public DbSet<Operations> Operation { get; set; }
        1 reference
        public DbSet<Ships> Ship { get; set; }
        7 references
        public DbSet<Documents> Document { get; set; }
        3 references
        public DbSet<OperationPayment> Operation_Payment { get; set; }


    }
}
```

The ApplicationDbContext class is defined within the CompanyAPIs.Data namespace.

It extends the IdentityDbContext<ApplicationUser> class, which is provided by ASP.NET Core Identity. This class is used for managing user authentication and authorization.

The constructor of ApplicationDbContext takes an instance of DbContextOptions<ApplicationDbContext> as a parameter, which is used to configure the database context options.

The class defines several DbSet properties to represent different entities in the database:

Operation represents a collection of Operations entities.

Ship represents a collection of Ships entities.

Document represents a collection of Documents entities.

Operation_Payment represents a collection of OperationPayment entities.

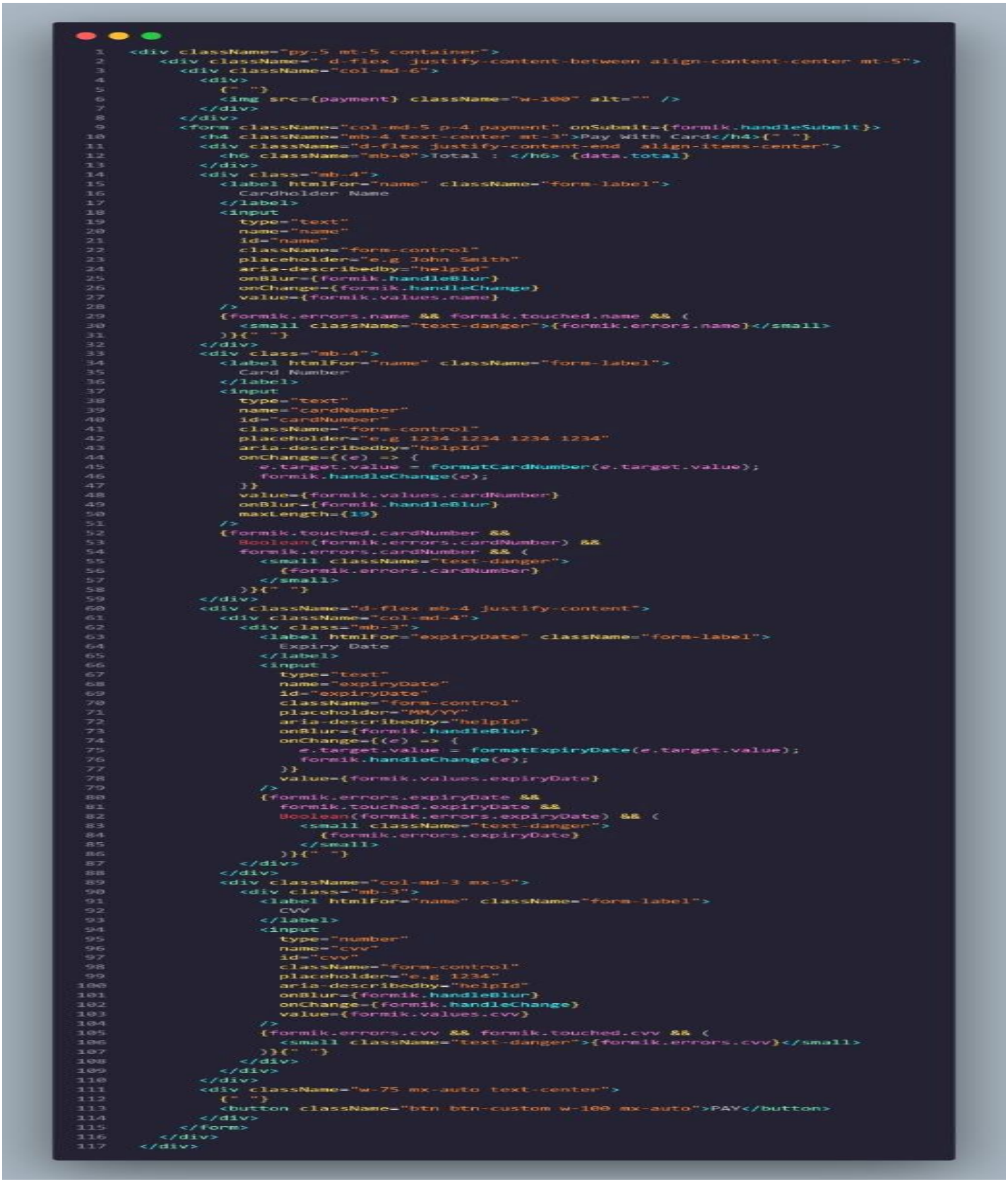
Each DbSet allows querying, inserting, updating, and deleting entities of the corresponding type in the underlying database.

The code does not provide the definitions of the Operations, Ships, Documents, and OperationPayment classes, but they are likely entity classes representing different aspects of the company's operations or data.

FRONT END:

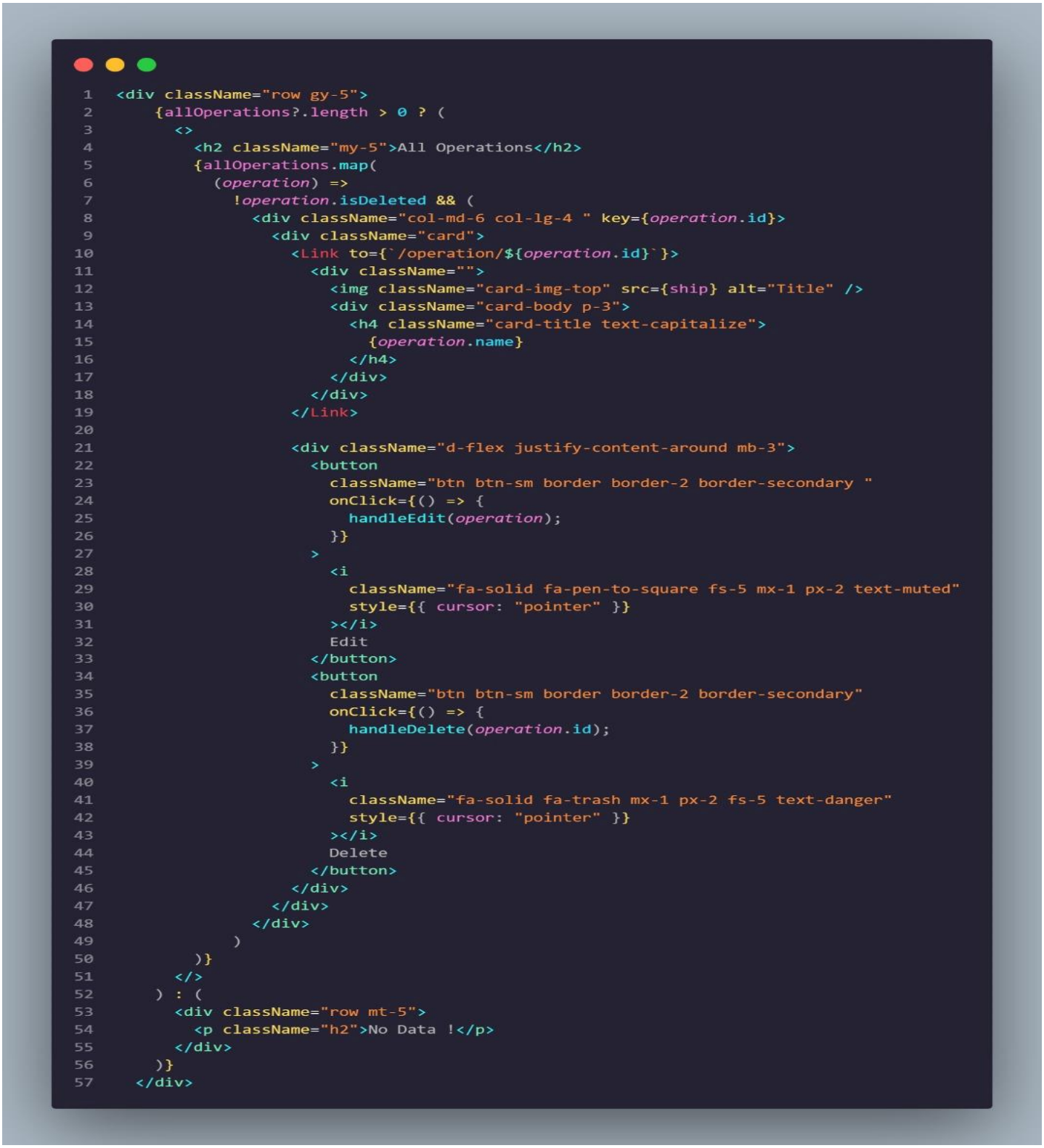
```
<div className="w-75 mx-auto p-3">
  {error && <div className="alert alert-danger">{error}</div>}
  <h2>Login Now :</h2>
  <form onSubmit={formik.handleSubmit}>
    <div className="mb-3">
      <label className="form-label">Email</label>
      <input
        type="email"
        name="email"
        id="email"
        className="form-control"
        placeholder=""
        aria-describedby="helpId"
        value={formik.values.email}
        onChange={formik.handleChange}
        onBlur={formik.handleBlur}
      />
      {formik.errors.email && formik.touched.email && (
        <div className="alert alert-danger">{formik.errors.email} </div>
      )}
    </div>

    <div className="mb-3">
      <label className="form-label">Password</label>
      <input
        type="password"
        name="password"
        id="password"
        className="form-control"
        placeholder=""
        aria-describedby="helpId"
        value={formik.values.password}
        onChange={formik.handleChange}
        onBlur={formik.handleBlur}
      />
      {formik.errors.password && formik.touched.password && (
        <div className="alert alert-danger">{formik.errors.password} </div>
      )}
    </div>
```

This code sets up a login form using React, Formik for form management, and Yup for form validation. It sends a login request to a server endpoint and handles responses accordingly, updating state variables to reflect the status of the login process.



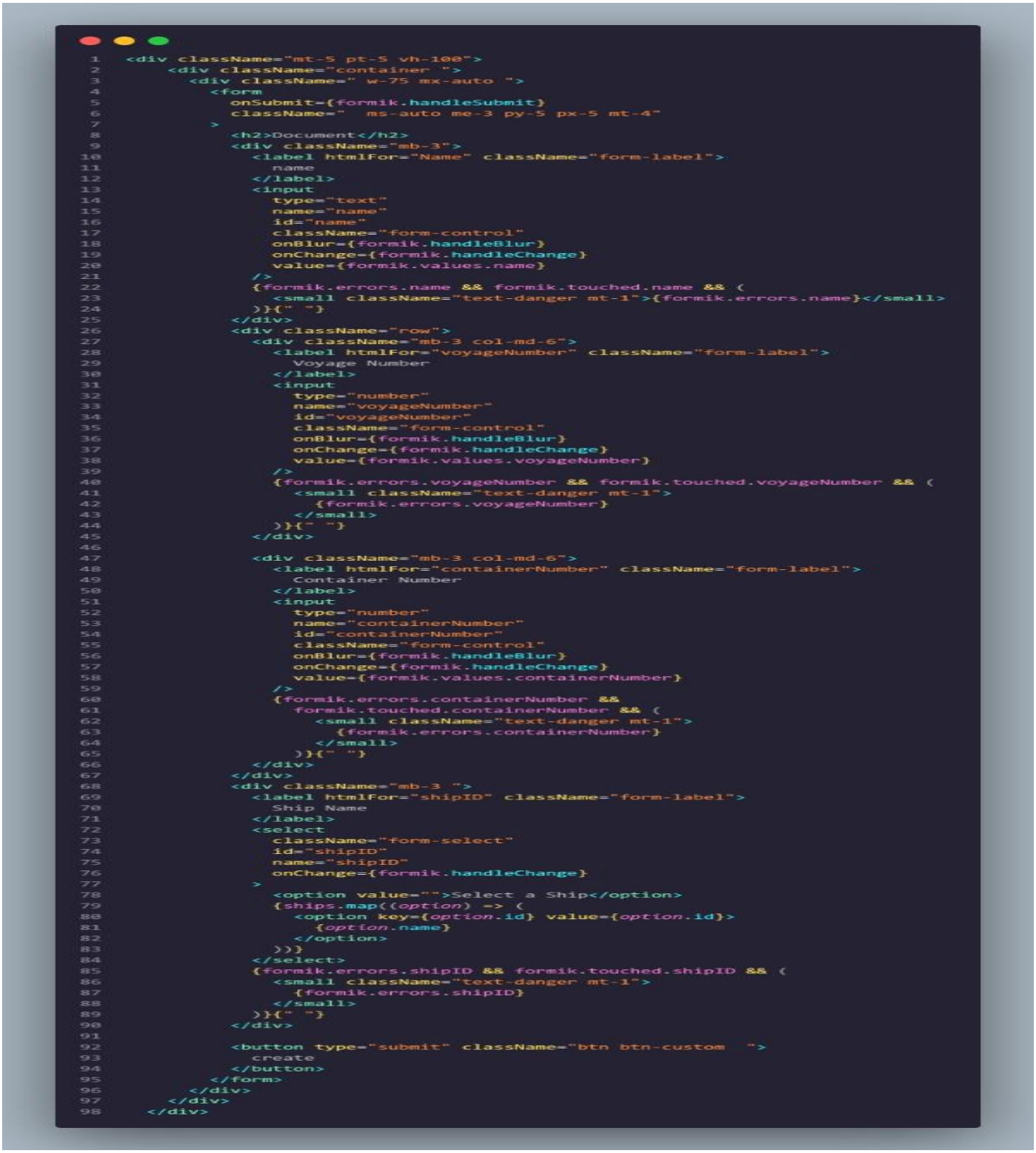
```jsx
<div className="py-5 mt-5 container">
    <div className=" d-flex  justify-content-between align-content-center mt-5">
      <div className="col-md-6">
        <div>
          {" "}
          <img src={payment} className="w-100" alt="" />
        </div>
      </div>
      <form className="col-md-5 p-4 payment" onSubmit={formik.handleSubmit}>
        <h4 className="mb-4 text-center mt-3">Pay With Card</h4>{" "}
        <div className="d-flex justify-content-end  align-items-center">
          <h6 className="mb-0">Total : </h6> {data.total}
        </div>
        <div class="mb-4">
          <label htmlFor="name" className="form-label">
            Cardholder Name
          </label>
          <input
            type="text"
            name="name"
            id="name"
            className="form-control"
            placeholder="e.g John Smith"
            aria-describedby="helpId"
            onBlur={formik.handleBlur}
            onChange={formik.handleChange}
            value={formik.values.name}
          />
          {formik.errors.name && formik.touched.name && (
            <small className="text-danger">{formik.errors.name}</small>
          )}{" "}
        </div>
        <div class="mb-4">
          <label htmlFor="name" className="form-label">
            Card Number
          </label>
          <input
            type="text"
            name="cardNumber"
            id="cardNumber"
            className="form-control"
            placeholder="e.g 1234 1234 1234 1234"
            aria-describedby="helpId"
            onChange={(e) => {
              e.target.value = formatCardNumber(e.target.value);
              formik.handleChange(e);
            }}
            value={formik.values.cardNumber}
            onBlur={formik.handleBlur}
            maxLength={19}
          />
          {formik.touched.cardNumber &&
            Boolean(formik.errors.cardNumber) &&
            formik.errors.cardNumber && (
              <small className="text-danger">
                {formik.errors.cardNumber}
              </small>
            )}{" "}
        </div>
        <div className="d-flex mb-4 justify-content">
          <div className="col-md-4">
            <div class="mb-3">
              <label htmlFor="expiryDate" className="form-label">
                Expiry Date
              </label>
              <input
                type="text"
                name="expiryDate"
                id="expiryDate"
                className="form-control"
                placeholder="MM/YY"
                aria-describedby="helpId"
                onBlur={formik.handleBlur}
                onChange={(e) => {
                  e.target.value = formatExpiryDate(e.target.value);
                  formik.handleChange(e);
                }}
                value={formik.values.expiryDate}
              />
              {formik.errors.expiryDate &&
                formik.touched.expiryDate &&
                Boolean(formik.errors.expiryDate) && (
                  <small className="text-danger">
                    {formik.errors.expiryDate}
                  </small>
                )}{" "}
            </div>
          </div>
          <div className="col-md-3 mx-5">
            <div class="mb-3">
              <label htmlFor="name" className="form-label">
                CVV
              </label>
              <input
                type="number"
                name="cvv"
                id="cvv"
                className="form-control"
                placeholder="e.g 1234"
                aria-describedby="helpId"
                onBlur={formik.handleBlur}
                onChange={formik.handleChange}
                value={formik.values.cvv}
              />
              {formik.errors.cvv && formik.touched.cvv && (
                <small className="text-danger">{formik.errors.cvv}</small>
              )}{" "}
            </div>
          </div>
        </div>
        <div className="w-75 mx-auto text-center">
          {" "}
          <button className="btn btn-custom w-100 mx-auto">PAY</button>
        </div>
      </form>
    </div>
  </div>
```

This code creates a responsive payment form with validation and integration with Formik for form management and validation handling.



```jsx
<div className="row gy-5">
    {allOperations?.length > 0 ? (
      <>
        <h2 className="my-5">All Operations</h2>
        {allOperations.map(
          (operation) =>
            !operation.isDeleted && (
              <div className="col-md-6 col-lg-4 " key={operation.id}>
                <div className="card">
                  <Link to={`/operation/${operation.id}`}>
                    <div classname="">
                      <img className="card-img-top" src={ship} alt="Title" />
                      <div className="card-body p-3">
                        <h4 className="card-title text-capitalize">
                          {operation.name}
                        </h4>
                      </div>
                    </div>
                  </Link>

                  <div className="d-flex justify-content-around mb-3">
                    <button
                      className="btn btn-sm border border-2 border-secondary "
                      onClick={() => {
                        handleEdit(operation);
                      }}
                    >
                      <i
                        className="fa-solid fa-pen-to-square fs-5 mx-1 px-2 text-muted"
                        style={{ cursor: "pointer" }}
                      ></i>
                      Edit
                    </button>
                    <button
                      className="btn btn-sm border border-2 border-secondary"
                      onClick={() => {
                        handleDelete(operation.id);
                      }}
                    >
                      <i
                        className="fa-solid fa-trash mx-1 px-2 fs-5 text-danger"
                        style={{ cursor: "pointer" }}
                      ></i>
                      Delete
                    </button>
                  </div>
                </div>
              </div>
            )
        )}
      </>
    ) : (
      <div className="row mt-5">
        <p className="h2">No Data !</p>
      </div>
    )}
  </div>
```

This code dynamically renders a list of operations with options to edit or delete each operation, and it handles cases where there is no data to display.



```
<div className="mt-5 pt-5 vh-100">
    <div className="container ">
      <div className=" w-75 mx-auto ">
        <form
          onSubmit={formik.handleSubmit}
          className="  ms-auto me-3 py-5 px-5 mt-4"
        >
          <h2>Document</h2>
          <div className="mb-3">
            <label htmlFor="Name" className="form-label">
              name
            </label>
            <input
              type="text"
              name="name"
              id="name"
              className="form-control"
              onBlur={formik.handleBlur}
              onChange={formik.handleChange}
              value={formik.values.name}
            />
            {formik.errors.name && formik.touched.name && (
              <small className="text-danger mt-1">{formik.errors.name}</small>
            )}{" "}
          </div>
          <div className="row">
            <div className="mb-3 col-md-6">
              <label htmlFor="voyageNumber" className="form-label">
                Voyage Number
              </label>
              <input
                type="number"
                name="voyageNumber"
                id="voyageNumber"
                className="form-control"
                onBlur={formik.handleBlur}
                onChange={formik.handleChange}
                value={formik.values.voyageNumber}
              />
              {formik.errors.voyageNumber && formik.touched.voyageNumber && (
                <small className="text-danger mt-1">
                  {formik.errors.voyageNumber}
                </small>
              )}{" "}
            </div>

            <div className="mb-3 col-md-6">
              <label htmlFor="containerNumber" className="form-label">
                Container Number
              </label>
              <input
                type="number"
                name="containerNumber"
                id="containerNumber"
                className="form-control"
                onBlur={formik.handleBlur}
                onChange={formik.handleChange}
                value={formik.values.containerNumber}
              />
              {formik.errors.containerNumber &&
                formik.touched.containerNumber && (
                  <small className="text-danger mt-1">
                    {formik.errors.containerNumber}
                  </small>
                )}{" "}
            </div>
          </div>
          <div className="mb-3 ">
            <label htmlFor="shipID" className="form-label">
              Ship Name
            </label>
            <select
              className="form-select"
              id="shipID"
              name="shipID"
              onChange={formik.handleChange}
            >
              <option value="">Select a Ship</option>
              {ships.map((option) => (
                <option key={option.id} value={option.id}>
                  {option.name}
                </option>
              ))}
            </select>
            {formik.errors.shipID && formik.touched.shipID && (
              <small className="text-danger mt-1">
                {formik.errors.shipID}
              </small>
            )}{" "}
          </div>

          <button type="submit" className="btn btn-custom  ">
            create
          </button>
        </form>
      </div>
    </div>
  </div>
```

This code creates a form for users to input details for creating a document, such as a shipping document, and it handles validation and submission using Formik.

CHAPTER SIX: TESTING AND INSTALLATION

Software testing is a crucial part of software quality assurance and is a review of the coding, design, and specification.

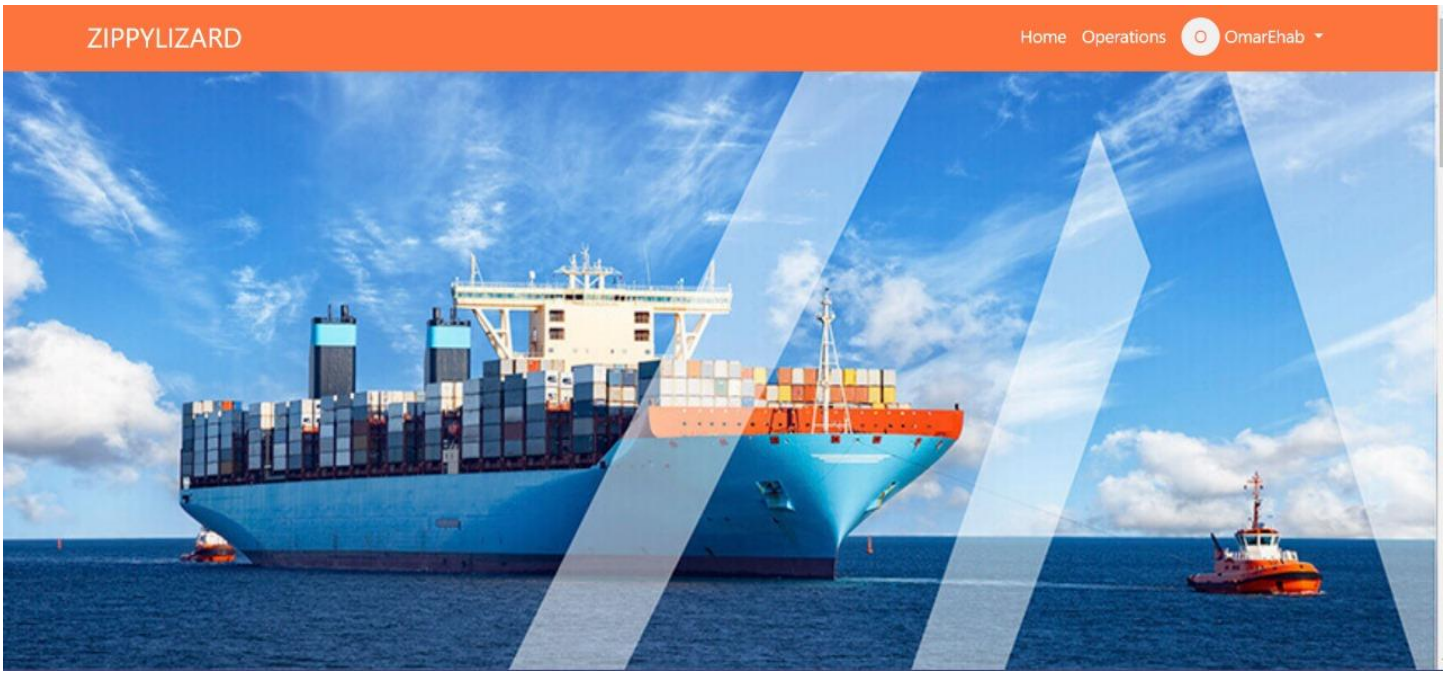
The process of software testing is used to determine the accuracy, completeness and developed quality.

Technical investigation known as testing is carried out on behalf of stakeholders with the goal of revealing information about the product's quality in relation to the environment in which it will be used.

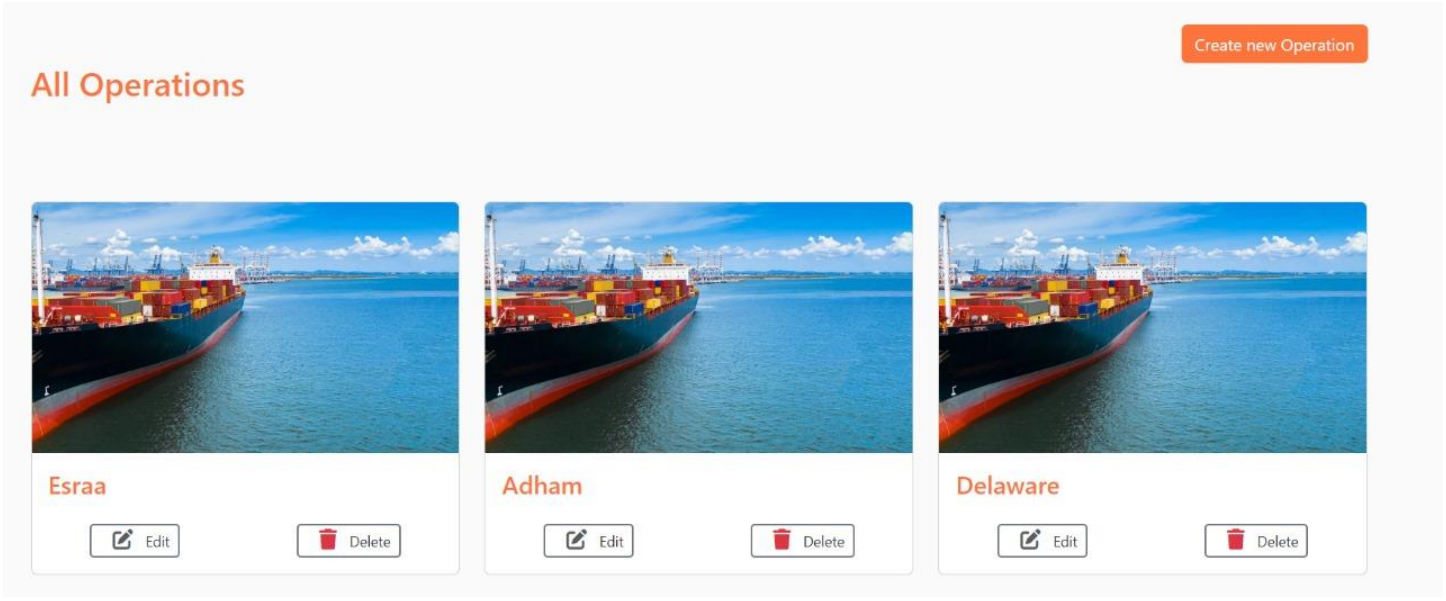
Software testing, put simply, is the process of determining whether the results obtained actually match those anticipated and that there are no flaws in the software system.

The types of testing we have already used to test our application, such as functional testing, unit testing, will be covered in the following sections, then state test cases we already used to test how our application will behave in the running environment

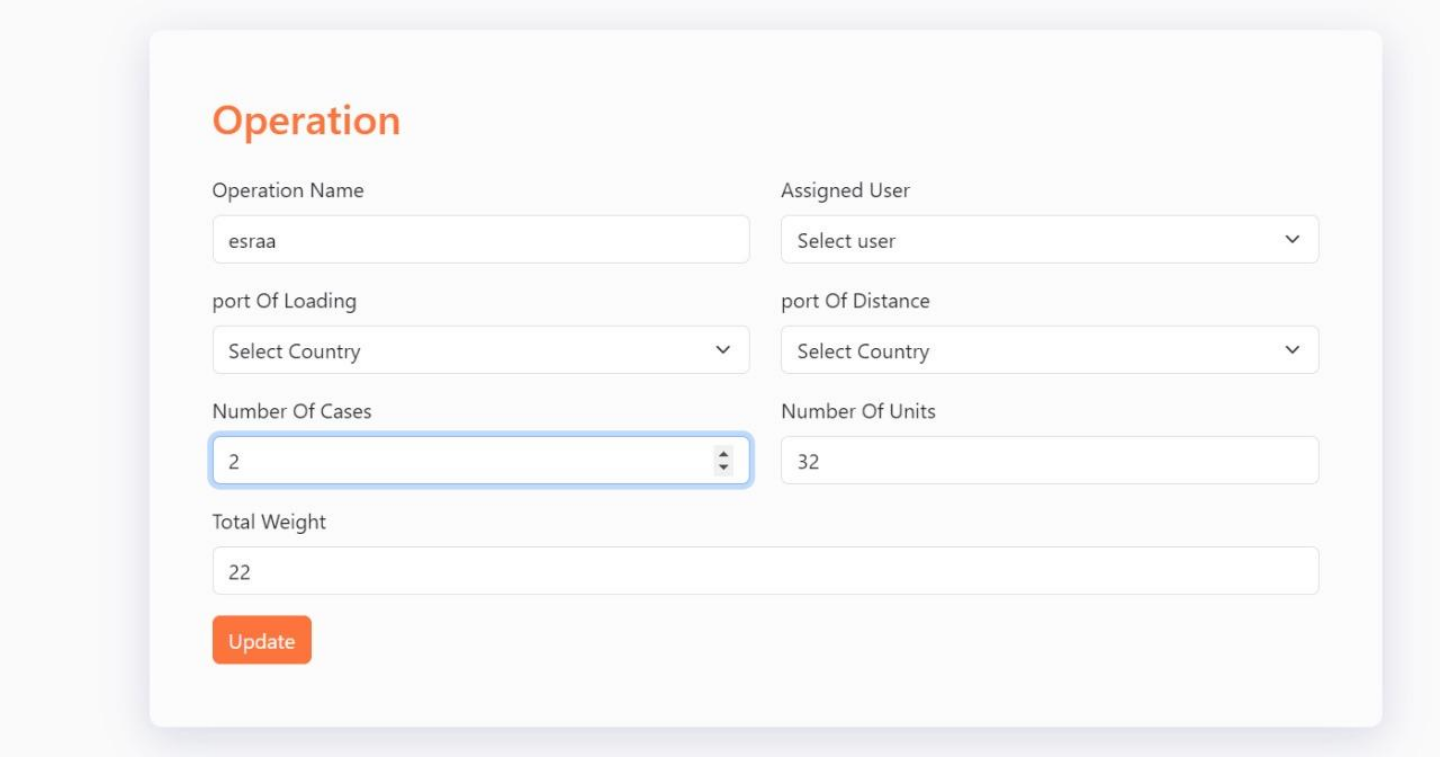
Home Page : in the home page we talked about our website and how we can help the costumers and also talk about maritime logistics and transportation and the main purpose of our website is online payment by our website to make it easier for the customer we will explain our system in the following photos.



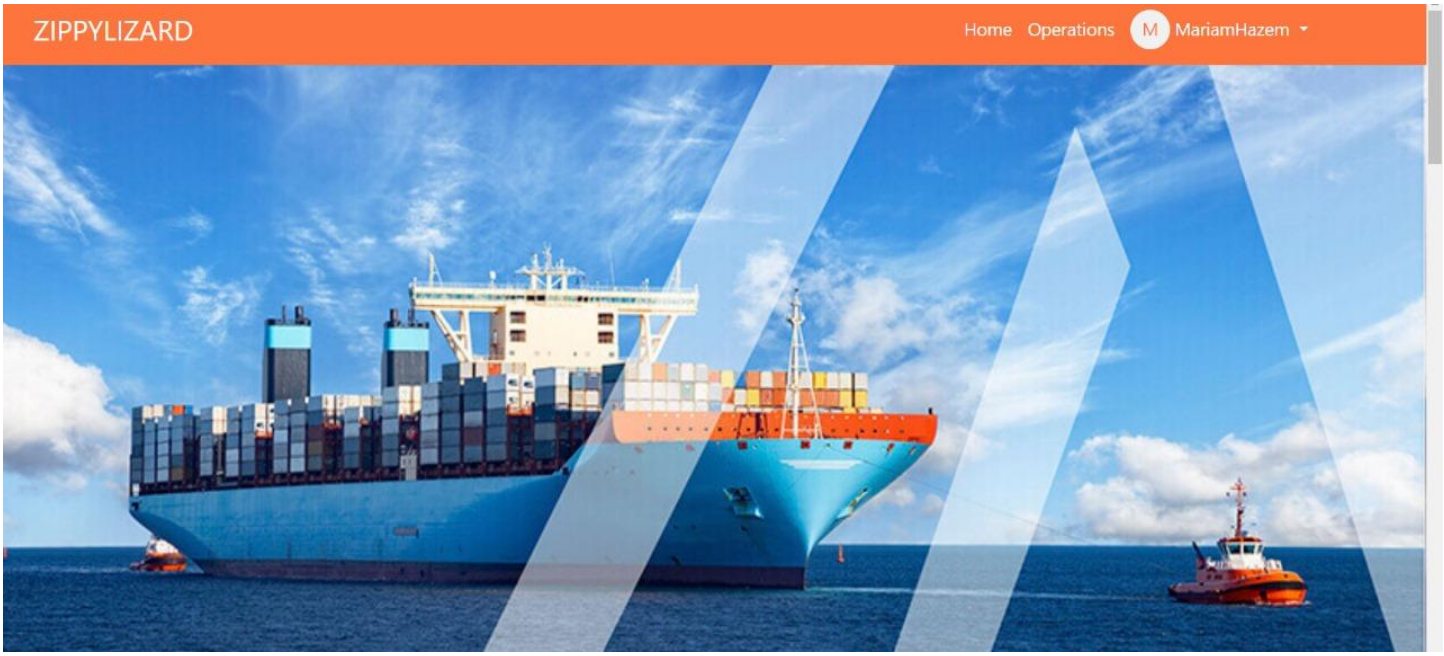
In this page we have three main characters (User , Employee and Admin) , each of them have different account from each other to make login in the system and in this page the employee make login in the website.



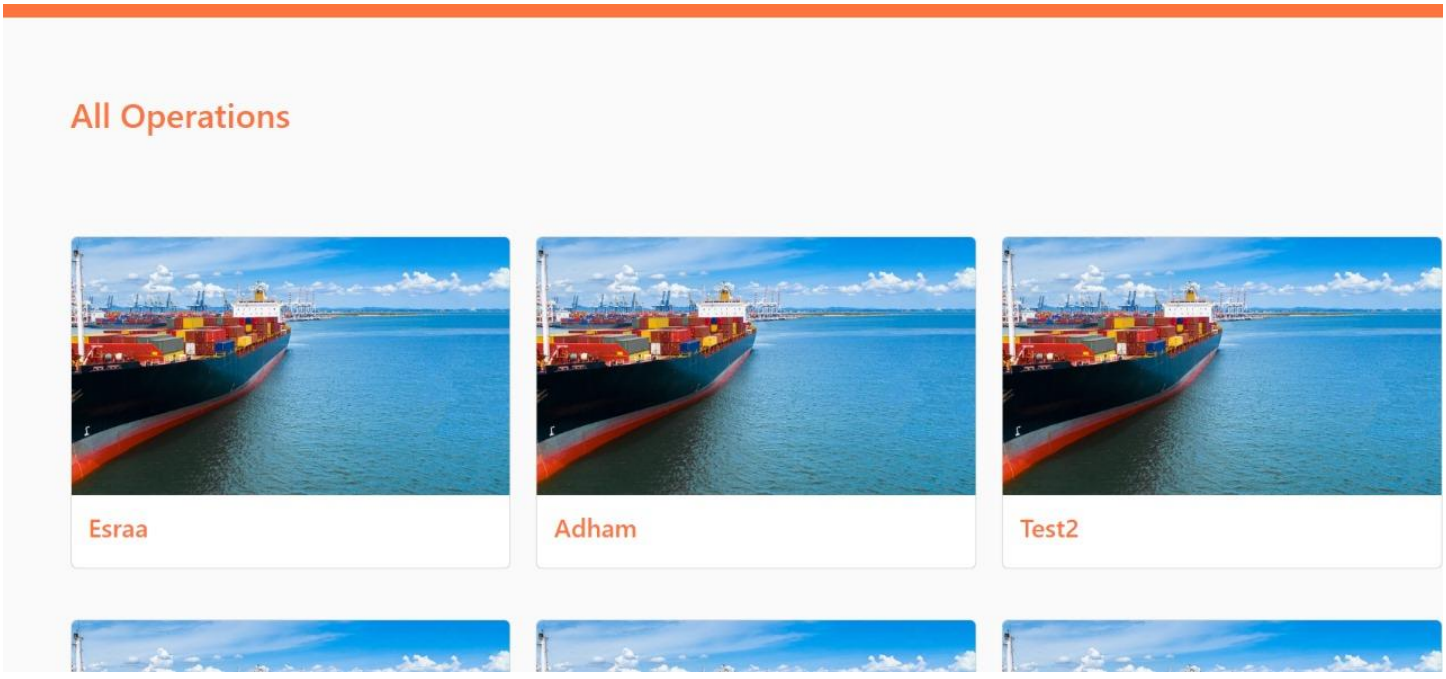
Here the employee make the operations for the costumers and we have previous operations that has been done before and he can make (update or delete) for it and also can make new operation for a new costumer



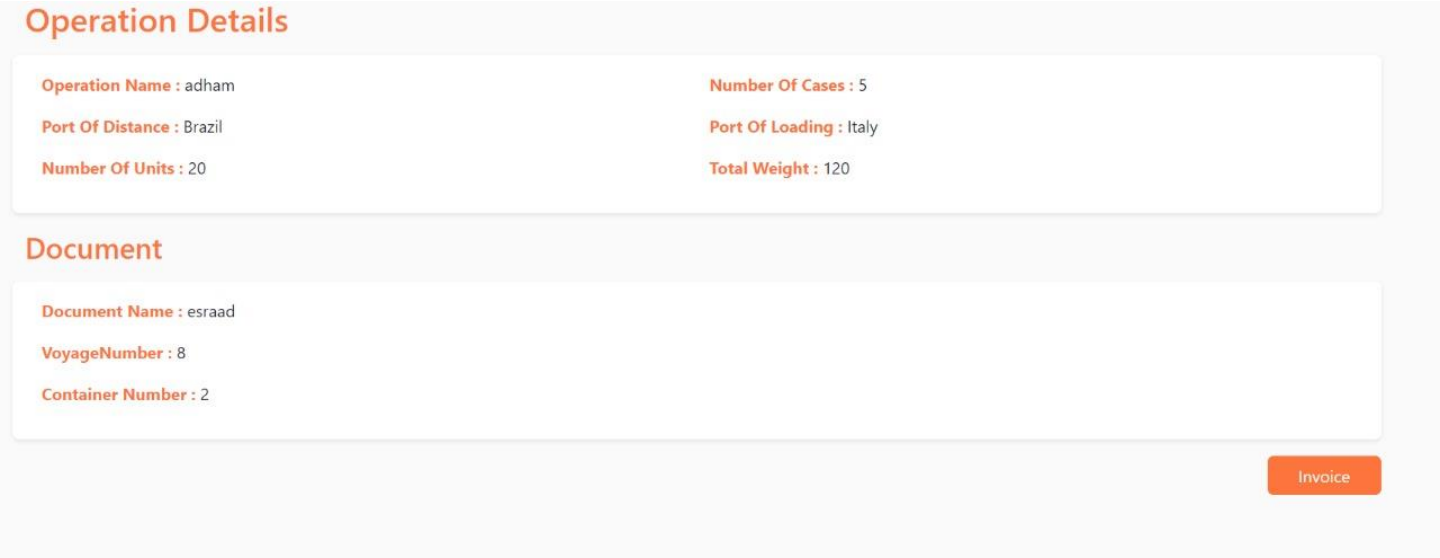
The explain of the operation: we have operation name , the name of the user , select the port of loading and select the port of distance , number of cases and number of units and the total weigh the customer can choose what he want or can write after finish he will click on the button of update .



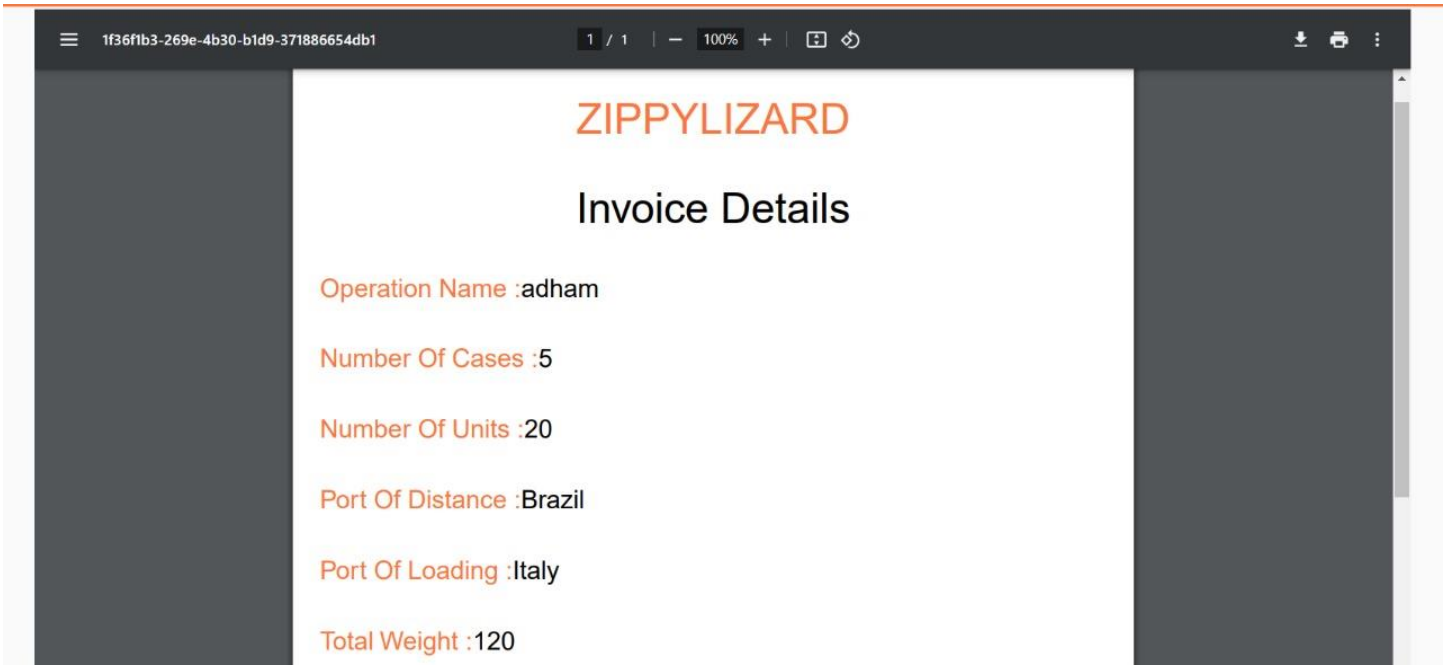
Here the customer make login and we will talk about the role of the customer in the following pages.



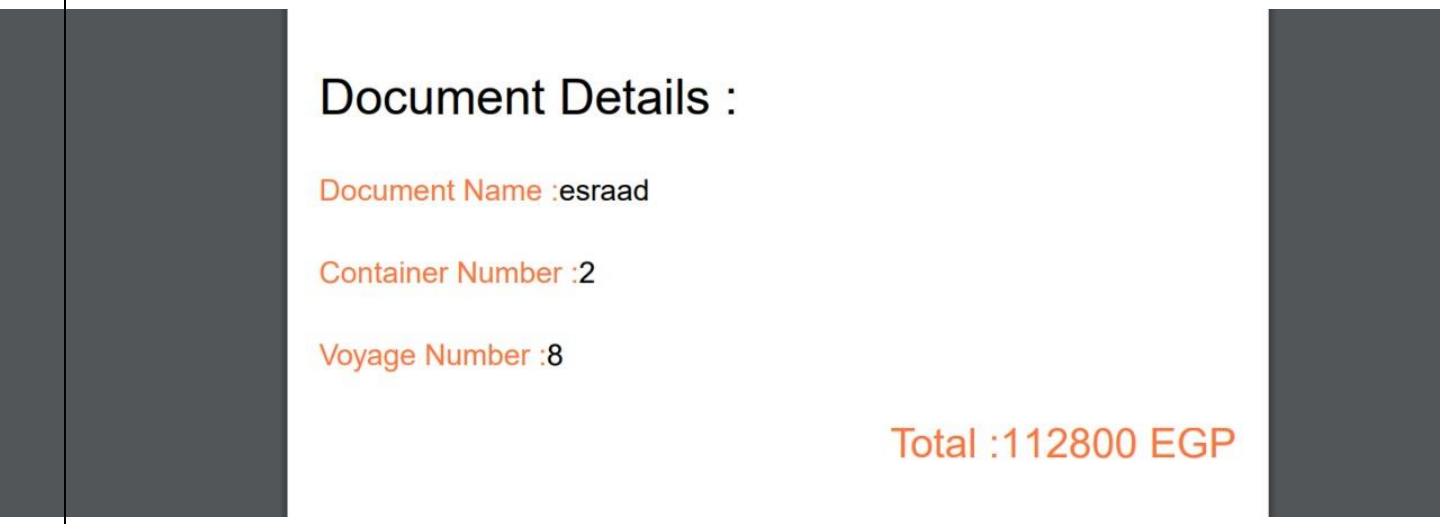
The customer can see that all the operations that has done and the new operation that the employee create it for the customer.



Here the customer see all the details of the operation and the document part that shown in this page is we have document name , voyage number and container number , and we have the invoice button.

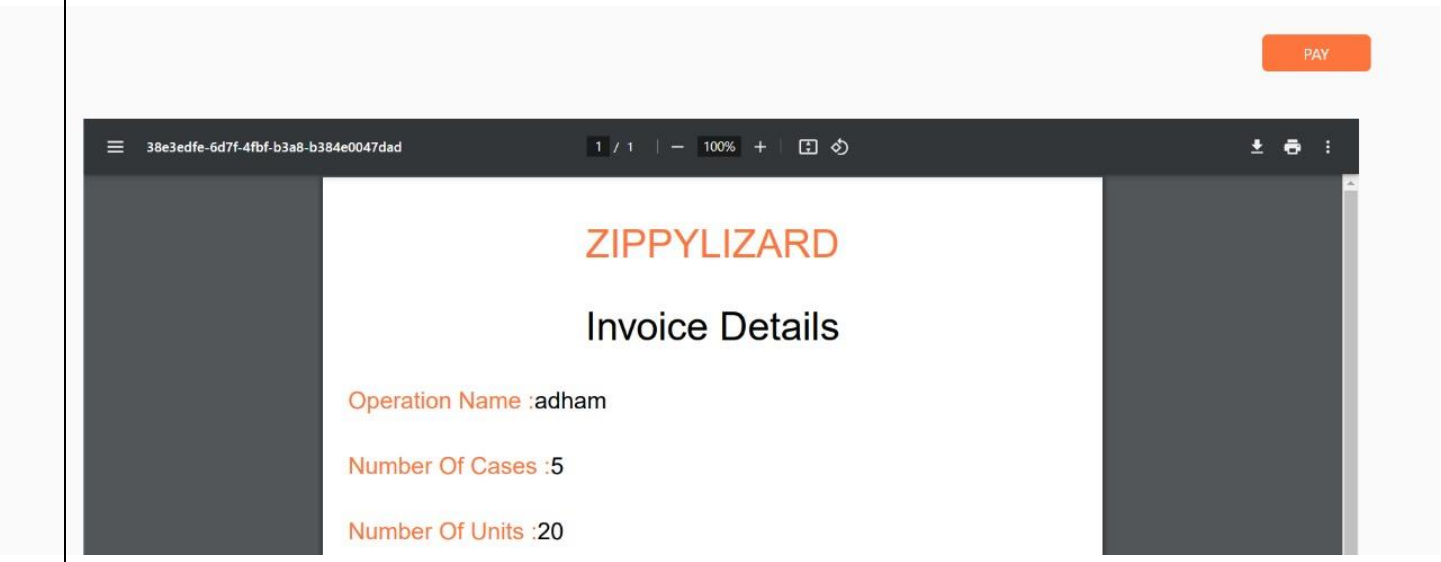


After the client click on the invoice button , the invoice details will shown for the customer as shown and thigs that shown is operation name and number of cases and number of units and port of distance and port of loading and the total weight.

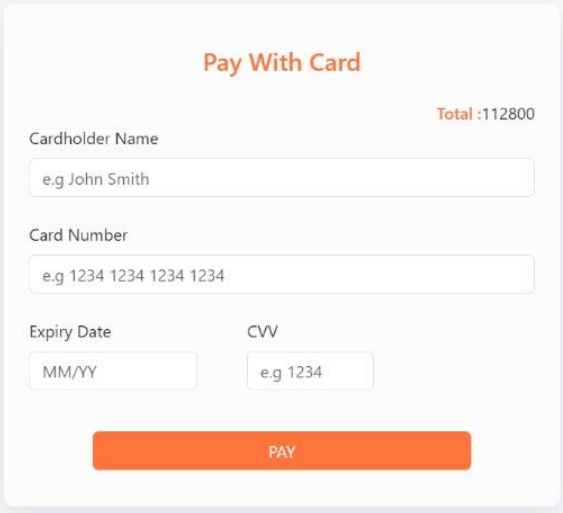


Here is the document details and it also appears in the invoice for the customer and the details is ; document name , container number and voyage number and the total amount that the customer will pay

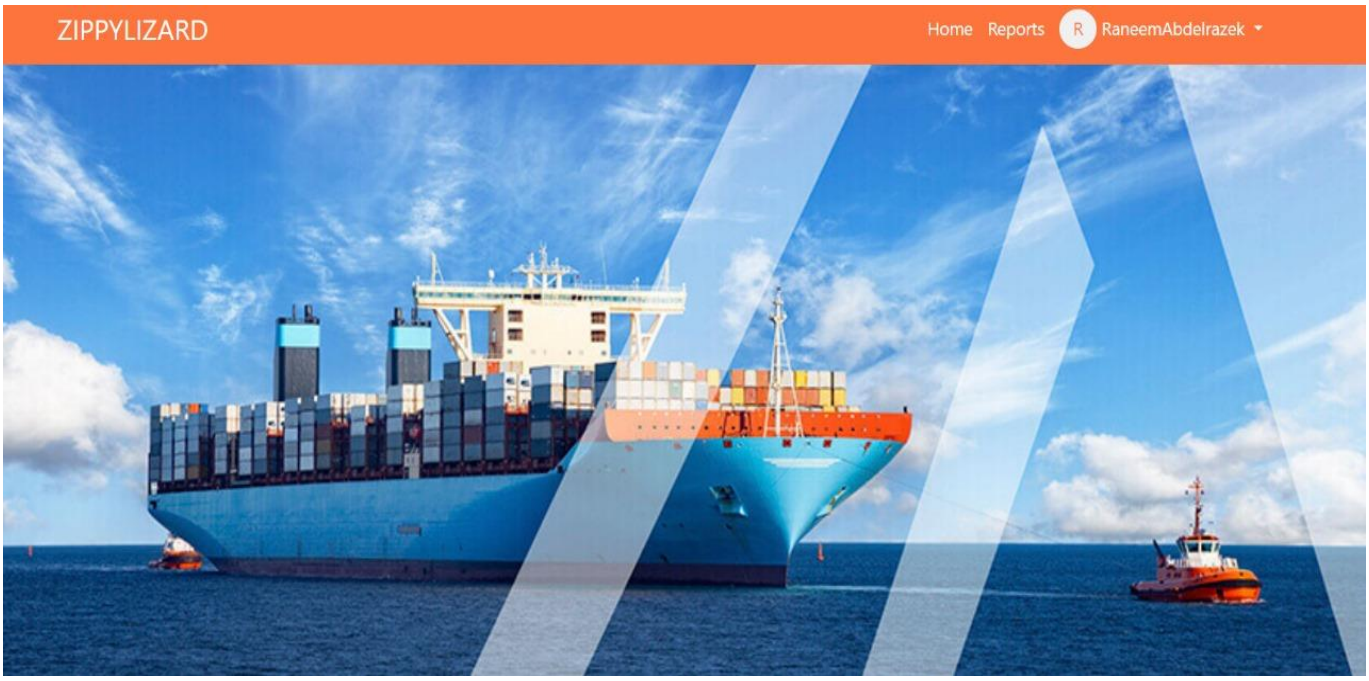
.



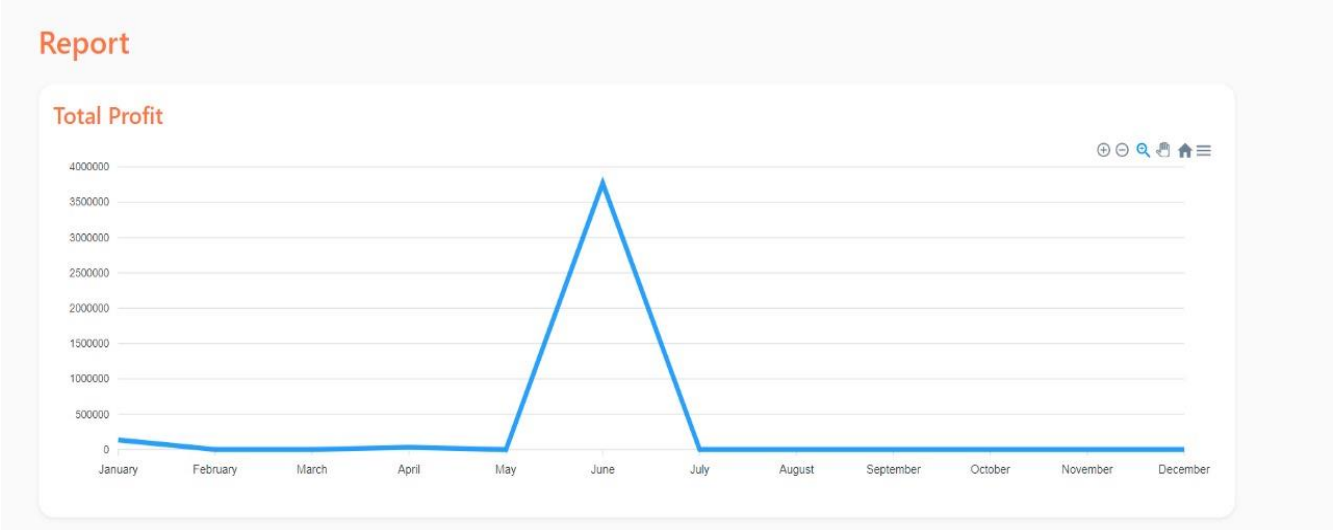
After the customer chick his invoice he can download this invoice and pay in the bank or he can click on the button (pay) to pay online from our system instead of the bank.



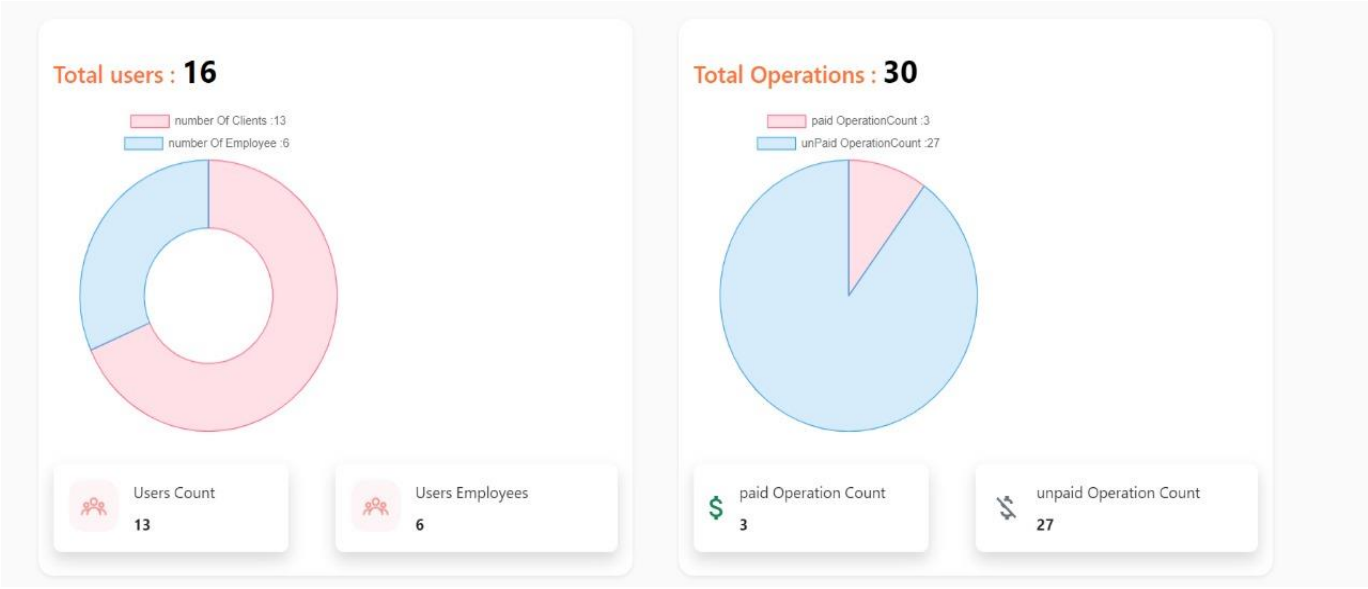
After the customer choose to pay online from our website this page will appear to him and he should enter the card holder name and card number and the date and cvv and after finishing a message will appear as payment done successfully .



After the previous process, the role of the user has finished and now we will talk about the role of the admin in the website.

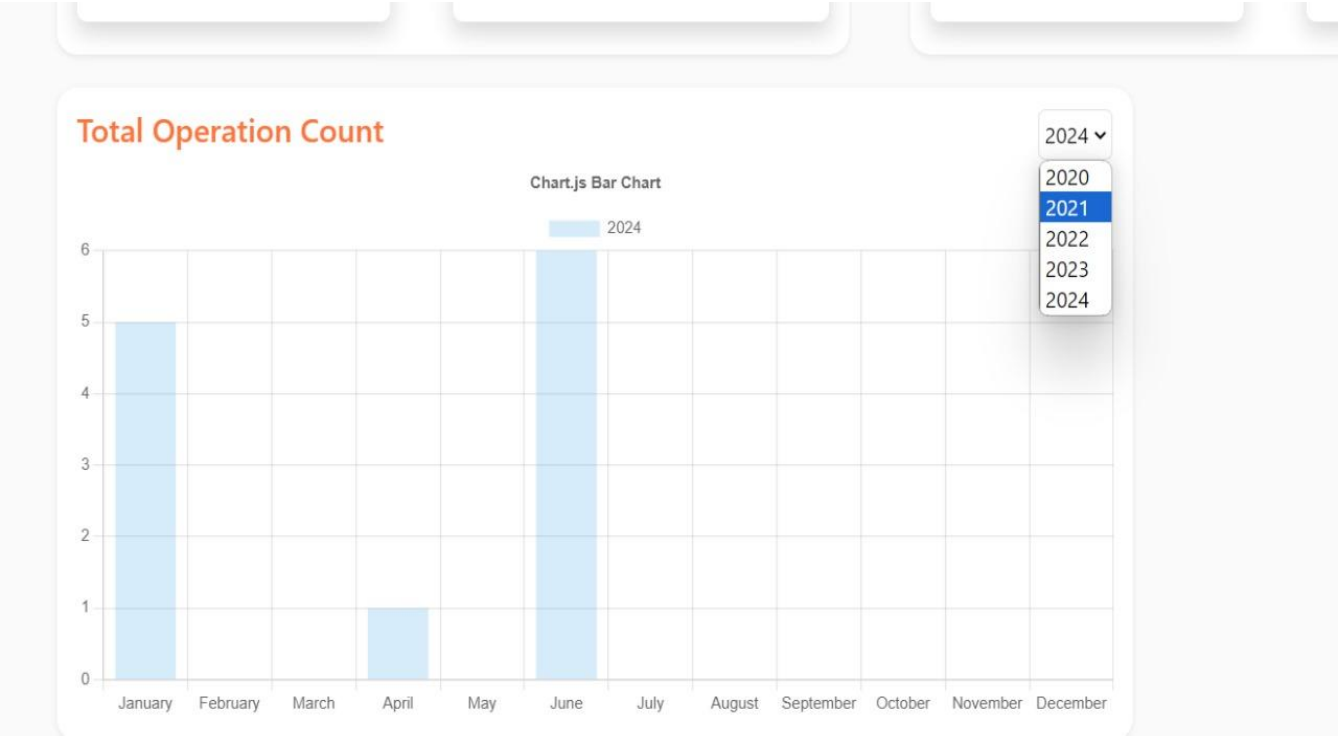


Here the admin only who have access to see the dashboard when he click on report . this graph show for the admin the total profit in each month and the page of report only admin have access on it.



The first circle it shows for the admin the total users , and here as given they are 16 user where the pink part is the number of clients and the blue part is number of employee and the second circle is number of operations and the total operations here is 30 , the pink part covers the paid operation which is 3 and the blue part covers the unpaid operations which are 27

.



Here we have the bar chart that show the total number of operations has done in the all years , as shown in the box like 2021, 2021, and 2024 and the admin only who can see

reports.

CHAPTER SEVEN: PROJECT CONCLUSION AND FUTURE WORK

❖ A Web Page to Scan Documents:

Implementing web-based scanning is a simple and cost-effective way to reduce paperwork and ensure accuracy and timeliness in the creation of clients B/Ls, as it also save the results automatically, which can eliminate human intervention, but there were some difficulties such as: software license, Document scanners cannot be directly accessed by a web page like a web camera. Hence, it requires a bridge to communicate with scanners.

❖ Tracking vessel:

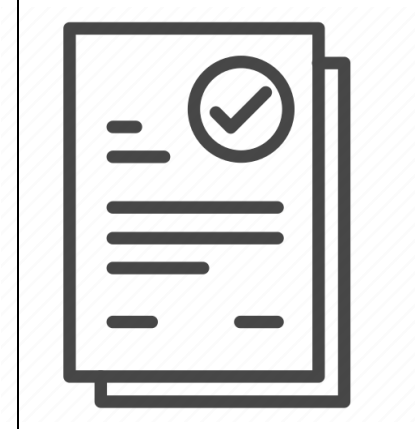
System needs to use the ArcGIS program in drawing maps, but there were some difficulties such as: software license, GPS usage was difficult as it covers small map scale and gives no exact coordination.

Our future planning:

1. Scan Document.

2. Container Tracker.

1. Is to Build a Web Page to Scan Documents:



To build a web page to scan documents in a web browser, you'll either have to develop a module from scratch or choose a document scanning SDK.

The latter is a better option as it eliminates the need to perform complex coding and saves time and effort.

Choosing the Right Document Scanner SDK

Document scanners cannot be directly accessed by a web page like a web camera. Hence, it requires a bridge to communicate with scanners. The TWAIN scanning protocol is the most preferred and recommended when working with document scanners.

Choosing the correct document scanner SDK may be challenging for you, with so many available options. The below parameters will help make your selection easier.

- **Easy Integration:** The document scanner SDK you choose must be easy to integrate. Building a document scanning module for your

website should not take more than a few lines of JavaScript code. Hence, you must choose the one that is easy to integrate.

- **Supported OS, Browsers, and Technology:** Are you using a Windows system or a MacBook? Which browser are you using? Is it Safari, Chrome, or Firefox? Select a document scanner SDK that supports a wide range of OS and browsers.

- **Fast Speed:** Speed is another crucial aspect you may want to consider while looking for a document scanner SDK for the web. The top ones can scan thousands of documents in a single session. Choosing such options will save you time and effort.

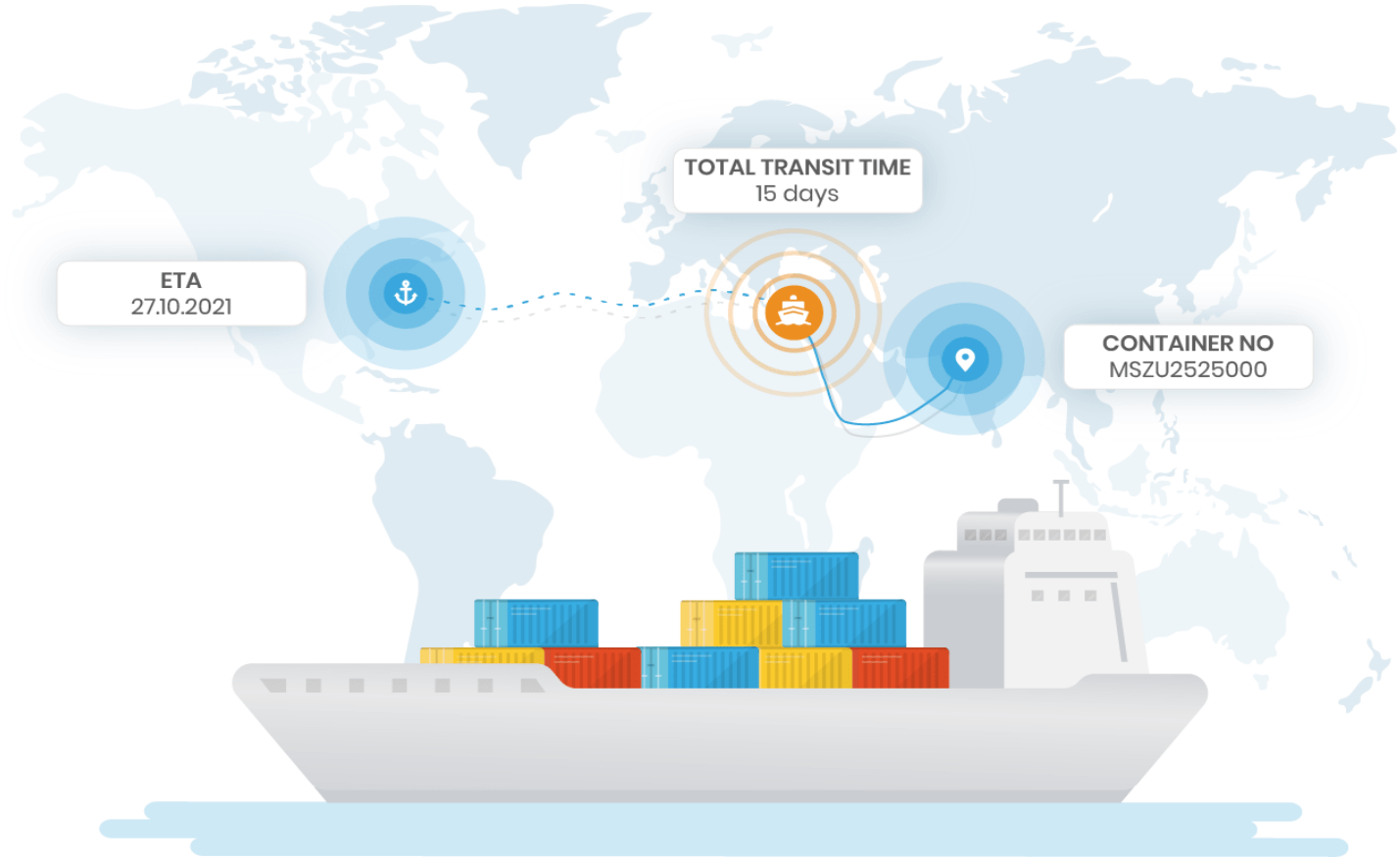
- **Sharing and Downloading Options:** For easy sharing and downloading, ensure the document scanner SDK that you choose supports formats such as JPEG, BMP, PNG, PDF, and TIFF files.

- **Basic Editing Features:** You wouldn't want to use a separate document editor to make fundamental changes such as crop, mirror, flip, erase, rotate, etc. Hence, look for a document scanner SDK with built-in basic image editing interfaces.

- **Robust Security Features:** Even if you use the document scanner SDK for personal use, you cannot compromise the security part. Hence, look for standard security features such as data encryption, built-in HTTPS support, digital signature, authorization access for accessing local files, etc.

2. Is to Build shipping Container Tracking:

The idea of adding this process to our website is to create an interactive map where users can enter vessel identifiers and see their shipment location on the map & Retrieving Real-time Vessels arrival time.



So, website will be designed to give instructions to the user by using APIs to get the current location of their shipment vessel.

In addition, we plan to use Google Maps service as a map engine and its Google Maps JavaScript API for interacting with it.

Lastly, the entire project is a type of front-end project, which we will implement using React JS library in combination with TypeScript.

❖ **References:**

- Wikipedia What is Language Integrated Query (LINQ)?
- Why. NETCORE? (.NET - Wikipedia)

- Code guru
- What is a GUI (Graphical User Interface)? (computerhope.com)
- GitHub
- COPILOT
- react@18.3.1