DBMS PORJECT REPORT

a) PROJECT NAME : SHIP PORT MANAGEMENT

TEAM MEMBER DETAILS:

NAME	SRN
VARUN S	PES1UG22CS679
VIGNESH	PES1UG22CS688

b) ABSTRACT:

The Ship Port Management System is an advanced solution designed to optimize port logistics and streamline operations. This system provides a centralized platform for managing resources, including ships, containers, port spaces, and bookings. It enables seamless communication between administrators and users, ensuring efficient resource allocation and service management. With features like real-time booking management, role-based access control, and a user-friendly interface, this system aims to enhance the efficiency and transparency of port operations. Built with cutting-edge technologies like React, Node.js, and MySQL, it caters to modern port management needs while supporting scalability and security.

c).User Requirement Specification:

1.Purpose of the Project

The Ship Port Management System is developed to address the challenges associated with managing port operations in a fast-paced global logistics environment. Ports are critical hubs for international trade, and their efficiency directly impacts supply chain operations. This system aims to digitize and optimize port-related activities, offering a streamlined interface for managing ships, containers, employees, and bookings. By automating administrative tasks, the system reduces errors, enhances operational efficiency, and improves the user experience for both port administrators and

customers. The primary goal is to provide a secure, reliable, and scalable solution that facilitates better resource utilization and ensures smooth operations for all stakeholders.

With the increasing complexity of maritime logistics, this system bridges the gap between traditional manual methods and the need for modern, technology-driven solutions. By enabling real-time updates, role-based management, and centralized data access, it simplifies decision-making and supports the growing demands of the shipping industry.

2. Scope of the Project

The Ship Port Management System is designed to cater to the diverse needs of port authorities, employees, and users. For administrators, it provides comprehensive tools to manage various resources, including ships, ports, employees, and container logs. It also ensures efficient handling of bookings, cancellations, and user profiles. For end users, the system offers a straightforward interface to browse available ports, book services, and manage their interactions with the port.

The project aims to implement features such as real-time notifications, advanced analytics for decision-making, and role-based access to enhance security. It is scalable to accommodate future enhancements, such as advanced container tracking, live monitoring, and predictive analytics. This system is intended for deployment across small to large-scale ports, making it versatile and adaptable for diverse logistical scenarios.

3.Detailed Description:

The Ship Port Management System is a full-stack application developed using modern technologies like React (frontend), Node.js with Express.js (backend), and MySQL (database). The system comprises the following key components:

Admin Module:

- Resource Management: Enables administrators to add, update, or delete data related to ports, ships, containers, and employees.
- Booking Management: Provides tools to view, approve, or cancel bookings.
- User Management: Allows admins to create roles, assign permissions, and manage user profiles.
- Logs : Offers detailed logs of bookings and operations for auditing purposes and supports data analytics to provide insights into port performance.

User Module:

- Port Browsing: Lets users explore available ports, view details such as capacity, and check availability.
- Service Booking: Simplifies booking services like port spaces or container handling.
- Profile Management: Allows users to update their information and view booking history.
- Booking Management: Enables users to cancel or modify pending bookings.

Entities and Attributes:

1. **Users**
   - **Attributes:** user_id (Primary Key), name, email, password, role (admin/user), phone_number, address.
   - **Description:** Represents all individuals interacting with the system, including administrators and end-users. The role attribute determines access permissions.
2. **Ports**
   - **Attributes:** port_id (Primary Key), port_name, location, capacity, availability_status.
   - **Description:** Represents the ports managed by the system, with details about their location and current capacity.
3. **Ships**
   - **Attributes:** ship_id (Primary Key), ship_name, capacity, owner, port_id (Foreign Key).
   - **Description:** Represents ships docked at or managed by the ports. Each ship is associated with a specific port.
4. **Bookings**
   - **Attributes:** booking_id (Primary Key), user_id (Foreign Key), port_id (Foreign Key), ship_id (Foreign Key), booking_date, status (confirmed/pending/canceled).
   - **Description:** Represents booking requests made by users to access port services. Each booking links a user, a port, and optionally a ship.
5. **Containers**
   - **Attributes:** container_id (Primary Key), container_type, capacity, status (loaded/unloaded), ship_id (Foreign Key).

- **Description:** Represents the containers handled by the ports. Each container is associated with a ship for logistics purposes.

6. **Employees**
   - **Attributes:** employee_id (Primary Key), name, designation, port_id (Foreign Key), email, phone_number.
   - **Description:** Represents the staff working at the ports, linked to a specific port.
7. **Countries**
   - **Attributes:** country_id (Primary Key), country_name, port_id (Foreign Key).
   - **Description:** Represents the countries where ports are located or operate, establishing international connections.
8. **Booking Logs (Included later as per the teacher's feedback after the project review.)**
   - **Attributes:** booking_id (Primary Key), user_id (Foreign Key), port_id (Foreign Key), ship_id (Foreign Key), booking_date, status (confirmed/pending/canceled).
   - **Description:** It stores all bookings made by users, including those made by users who have been deleted.

Relationships:

1. **Users and Bookings**
   - **Relationship:** A user can create multiple bookings, but each booking is associated with a single user.
   - **Type:** One-to-Many (Users → Bookings).
2. **Ports and Ships**
   - **Relationship:** A port can host multiple ships, but each ship is docked at only one port.
   - **Type:** One-to-Many (Ports → Ships).
3. **Ports and Bookings**
   - **Relationship:** A port can have multiple bookings, and each booking is tied to a single port.
   - **Type:** One-to-Many (Ports → Bookings).
4. **Ships and Containers**
   - **Relationship:** A ship can carry multiple containers, but each container belongs to a single ship.

- **Type:** One-to-Many (Ships → Containers).

5. **Ports and Employees**
    - **Relationship:** A port can have multiple employees working at it, but each employee is associated with one port.
    - **Type:** One-to-Many (Ports → Employees).
6. **Countries and Ports**
    - **Relationship:** A country can have multiple ports, but each port belongs to only one country.
    - **Type:** One-to-Many (Countries → Ports).
7. **Users and Logs**
    - **Relationship:** A user can generate multiple logs through actions, and each log is linked to a single user.
    - **Type:** One-to-Many (Users → Logs).

Key Assumptions

1. Each booking must involve at least one user and one port, and optionally a ship.
2. Containers are always associated with a ship, even when unloaded.
3. Employees are directly tied to the ports where they work, ensuring easy management.

d) List of Software, Tools, and Programming Languages Used

Frontend Development

1. **React.js**: For building the user interface and creating interactive web pages.
2. **Vite**: A fast build tool and development server for optimizing the React project.
3. **Bootstrap**: For responsive and consistent styling of the application.
4. **HTML5**: For structuring the content of the web application.
5. **CSS3**: For designing and enhancing the visual presentation of the application.

Backend Development

1. **Node.js**: A runtime environment for executing JavaScript code on the server side.

2. **Express.js**: A web application framework for creating RESTful APIs and handling backend logic.
3. **JWT (JSON Web Tokens)**: For secure authentication and authorization of users.

Database

1. **MySQL**: A relational database management system to store, retrieve, and manage data such as users, bookings, and ports.

Development Tools

1. **Visual Studio Code**: A code editor for writing and debugging code.
2. **Postman**: For testing and debugging APIs during development.

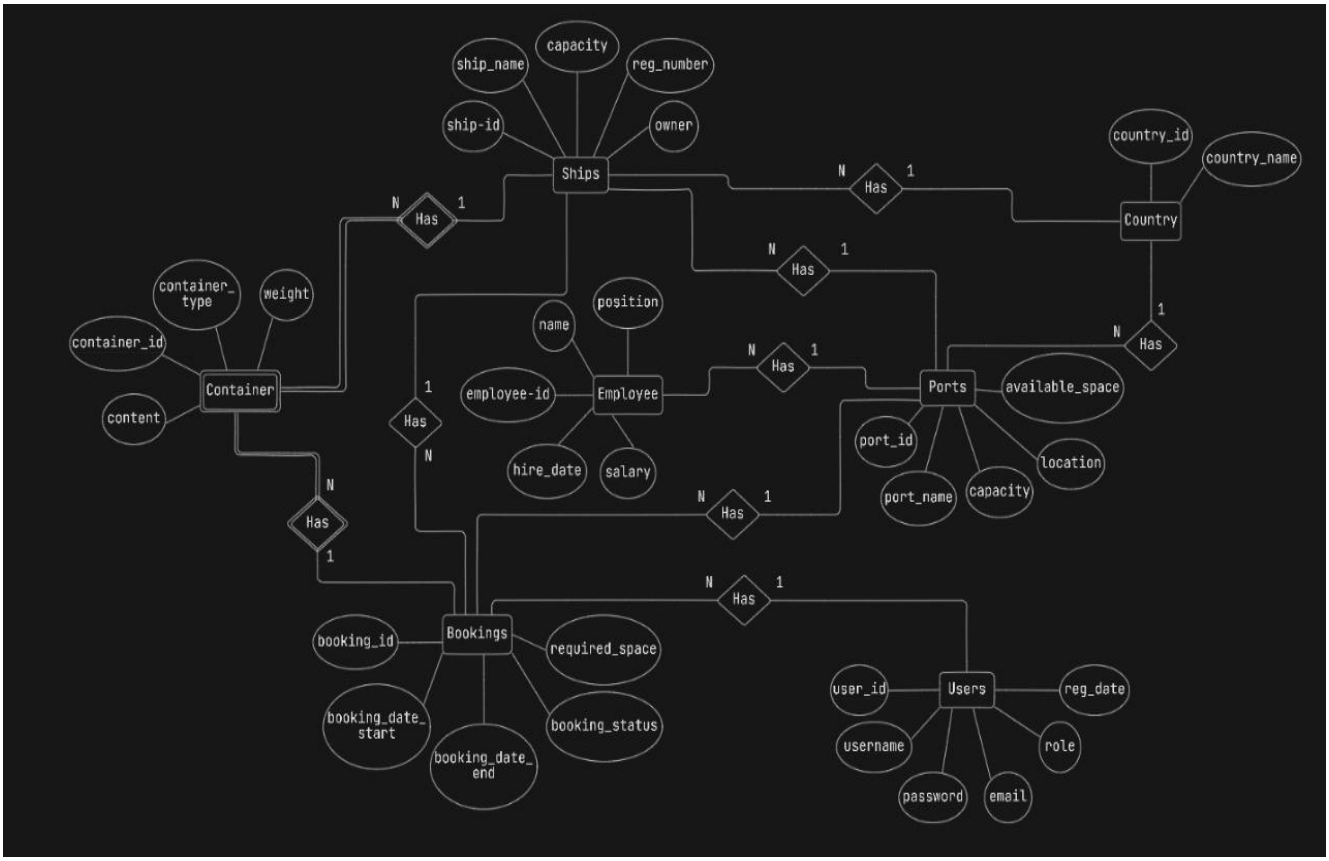
Version Control

1. **Git**: For version control and tracking changes in the codebase.
2. **GitHub**: For collaborative development and hosting the project repository.

Other Tools

1. **npm (Node Package Manager)**: For managing dependencies and libraries used in the project.
2. **MySQL Workbench/DBeaver**: For interacting with the MySQL database during development.

e) ER DIAGRAM:


capacity
ship_name
reg_number
ship-id
owner
Ships
country_id
country_name
Country
N
Has
1
container_type
weight
container_id
Container
content
name
position
employee-id
Employee
hire_date
salary
Ports
available_space
port_id
location
port_name
capacity
Bookings
booking_id
required_space
booking_date_start
booking_date_end
booking_status
user_id
Users
reg_date
username
role
password
email

f) RELATION SCHEMA:

g) DDL QUERIES:

TABLE CREATION QUERY:

```sql
-- Country Table
CREATE TABLE Country (
    country_id INT PRIMARY KEY AUTO_INCREMENT,
    country_name VARCHAR(100) NOT NULL
);

-- Ports Table
CREATE TABLE Ports (
    port_id INT PRIMARY KEY AUTO_INCREMENT,
    port_name VARCHAR(100) NOT NULL,
    country_id INT,
    capacity INT NOT NULL,
    available_space INT NOT NULL,
    location VARCHAR(255),
    FOREIGN KEY (country_id) REFERENCES Country(country_id) ON DELETE CASCADE
);
```

```sql
-- Ships Table
CREATE TABLE Ships (
    ship_id INT PRIMARY KEY AUTO_INCREMENT,
    ship_name VARCHAR(100) NOT NULL,
    capacity INT NOT NULL,
    registration_number VARCHAR(50) UNIQUE,
    owner VARCHAR(100),
    country_id INT,
    port_id INT,
    FOREIGN KEY (country_id) REFERENCES Country(country_id) ON DELETE SET NULL,
    FOREIGN KEY (port_id) REFERENCES Ports(port_id) ON DELETE SET NULL
);

-- Employee Table
CREATE TABLE Employee (
    employee_id INT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50),
    position VARCHAR(100),
    salary DECIMAL(10, 2),
    hire_date DATE,
    port_id INT,
    FOREIGN KEY (port_id) REFERENCES Ports(port_id) ON DELETE SET NULL
);

-- Users Table
CREATE TABLE Users (
    user_id INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) NOT NULL,
    password VARCHAR(255) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    role ENUM('admin', 'user') DEFAULT 'user',
    registration_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Bookings Table
CREATE TABLE Bookings (
    booking_id INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT,
    port_id INT,
    ship_id INT,
    booking_date_start TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    booking_date_end TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    booking_status ENUM('pending', 'confirmed', 'canceled') DEFAULT 'pending',
    required_space INT,
```

```
    FOREIGN KEY (user_id) REFERENCES Users(user_id) ON DELETE CASCADE,
    FOREIGN KEY (port_id) REFERENCES Ports(port_id) ON DELETE CASCADE,
    FOREIGN KEY (ship_id) REFERENCES Ships(ship_id) ON DELETE CASCADE
);

-- Container Table
CREATE TABLE Container (
    container_id INT PRIMARY KEY AUTO_INCREMENT,
    container_type VARCHAR(50) NOT NULL,
    weight INT NOT NULL,
    contents VARCHAR(255),
    ship_id INT,
    booking_id INT,
    FOREIGN KEY (ship_id) REFERENCES Ships(ship_id) ON DELETE CASCADE,
    FOREIGN KEY (booking_id) REFERENCES Bookings(booking_id) ON DELETE CASCADE
);
--Booking_Logs
CREATE TABLE Booking_Logs (
    log_id INT PRIMARY KEY AUTO_INCREMENT,
    booking_id INT,
    user_id INT,
    username varchar(50),
    port_id INT,
    portname varchar(100),
    ship_id INT,
    shipname varchar(100),
    booking_date_start TIMESTAMP,
    booking_date_end TIMESTAMP,
    booking_status ENUM('pending', 'confirmed', 'canceled'),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

h) DML QUERIES (CRUD OPERATIONS):

1.DATA RETERIVAL QUERIES

```
SELECT * FROM Users

SELECT * FROM Users WHERE username = ? OR email = ?

SELECT Ports.*, Country.country_name FROM Ports LEFT JOIN Country ON
Ports.country_id = Country.country_id
```

```sql
SELECT port_id, required_space FROM Bookings WHERE booking_id = ?

SELECT * FROM Country;

SELECT * FROM Country WHERE country_name = ?;

SELECT * FROM Ships WHERE registration_number = ?;

SELECT available_space FROM Ports WHERE port_id = ?;

SELECT * FROM Bookings WHERE user_id = ? ORDER BY booking_date_start DESC;

SELECT ship_id, ship_name FROM Ships;

SELECT required_space, port_id FROM Bookings WHERE booking_id = ?;

SELECT * FROM Users WHERE email = ?;

SELECT * FROM Users WHERE user_id = ?;

SELECT * FROM Employee WHERE first_name = ? AND last_name = ?;

SELECT available_space FROM Ports WHERE port_id = ?;
```

2.DATA RETERIVAL WITH JOIN

```sql
SELECT Bookings.*, Users.username, Ports.port_name, Ships.ship_name FROM Bookings
LEFT JOIN Users ON Bookings.user_id = Users.user_id
LEFT JOIN Ports ON Bookings.port_id = Ports.port_id
LEFT JOIN Ships ON Bookings.ship_id = Ships.ship_id;
```

Description:

- Retrieves all columns from the Bookings table.
- Adds the username from the Users table, port_name from the Ports table, and ship_name from the Ships table.
- Uses **LEFT JOIN** to ensure all bookings are included, even if they don't have matching user, port, or ship information.

```sql
SELECT Employee.*, Ports.port_name
```

```
FROM Employee
LEFT JOIN Ports ON Employee.port_id = Ports.port_id;
```

Description:

- Retrieves all columns from the Employee table.
- Adds the port_name from the Ports table.
- Uses **LEFT JOIN** to include all employees,

```
SELECT Ships.*, Country.country_name, Ports.port_name
FROM Ships
LEFT JOIN Country ON Ships.country_id = Country.country_id
LEFT JOIN Ports ON Ships.port_id = Ports.port_id;
```

Description:

- Retrieves all columns from the Ships table.
- Adds the country_name from the Country table and port_name from the Ports table.
- Uses **LEFT JOIN** to ensure all ships are included even if they have no matching country or port.

```
SELECT Container.*, Ships.ship_name, Bookings.booking_status
FROM Container
LEFT JOIN Ships ON Container.ship_id = Ships.ship_id
LEFT JOIN Bookings ON Container.booking_id = Bookings.booking_id;
```

Description:

- Retrieves all columns from the Container table.
- Adds the ship_name from the Ships table, showing which ship is associated with each container.
- Adds the booking_status from the Bookings table to show the status of the booking for each container.
- Uses **LEFT JOIN** to ensure all containers are included

```
SELECT Bookings.booking_id, Bookings.booking_status, Bookings.booking_date_start,
  Bookings.booking_date_end, Bookings.required_space, Ports.port_name,
  Ports.location, Ports.capacity, Ships.ship_name, Ships.ship_id
FROM Bookings JOIN Ports ON Bookings.port_id = Ports.port_id
LEFT JOIN Ships ON Bookings.ship_id = Ships.ship_id
```

```
WHERE Bookings.user_id = ? ORDER BY Bookings.booking_date_start DESC;
```

Description:

- Retrieves booking details (ID, status, start/end dates, and required space) from the Bookings table.
- Retrieves associated port details (name, location, capacity) from the Ports table using an inner JOIN.
- Retrieves associated ship details (name, ID) from the Ships table using a LEFT JOIN (to ensure all bookings are shown, even if no ship is assigned).
- Filters the bookings to only those for a specific user, identified by user_id (specified by a parameter ?).
- Orders the results by the booking's start date in descending order (DESC), showing the most recent bookings first.

```
 SELECT Ports.port_id, Ports.port_name, Ports.capacity, Ports.available_space,
  Country.country_name FROM Ports LEFT JOIN Country ON Ports.country_id =
Country.country_id WHERE Ports.available_space > 0;
```

Description:

- Retrieves information from the Ports table including the port ID, name, capacity, and available space.
- Retrieves the associated country name from the Country table using a LEFT JOIN on the country_id.
- Filters the results to only include ports where the available_space is greater than zero, ensuring that only ports with available capacity are returned.

```
SELECT Container.*
FROM Container
JOIN Bookings ON Container.booking_id = Bookings.booking_id
WHERE Bookings.user_id = ?;
```

Description:

- Retrieves all columns from the Container table.
- Joins the Bookings table to filter containers by the user_id of the bookings.
- The query returns containers associated with a particular user.

```sql
SELECT Ships.*, Bookings.booking_date_start, Bookings.booking_date_end
FROM Ships
JOIN Bookings ON Ships.ship_id = Bookings.ship_id
WHERE Bookings.user_id = ?
  AND Bookings.booking_status = 'pending'
  AND Bookings.booking_date_start > NOW()
ORDER BY Bookings.booking_date_start ASC;
```

Description:

- Retrieves details of ships from the Ships table.
- Filters bookings by user_id, with a status of 'pending' and a start date in the future (> NOW()).
- Orders the results by the start date of the bookings in ascending order.

```sql
SELECT Container.*, Bookings.booking_status
FROM Container
JOIN Bookings ON Container.booking_id = Bookings.booking_id
WHERE Bookings.user_id = ?;
```

Description:

- Retrieves all columns from the Container table along with the booking_status from the Bookings table.
- Filters the containers based on the user_id of the associated bookings, returning the booking status along with container details.

3. SELECT QUERIES WITH SUBQUERIES

```sql
SELECT
  Ports.port_id,
  Ports.port_name,
  Ports.capacity,
  Ports.available_space,
  calculate_port_utilization(Ports.port_id) AS capacityUtilization,
  (SELECT COUNT(*)
   FROM Bookings
   WHERE Bookings.port_id = Ports.port_id
   AND (booking_status = 'confirmed' OR booking_status = 'pending')) AS
activeBookings,
  (SELECT SUM(Container.weight)
   FROM Container
   INNER JOIN Bookings ON Container.booking_id = Bookings.booking_id
```

```
   WHERE Bookings.port_id = Ports.port_id) AS totalCargoLoad
FROM Ports;
```

Description:

- Ports Information: It fetches the port_id, port_name, capacity, and available_space from the Ports table.
- Capacity Utilization: It calculates the port's capacity utilization using the calculate_port_utilization() function.
- Active Bookings: It counts the number of active bookings (both 'confirmed' and 'pending') for each port by querying the Bookings table.
- Total Cargo Load: It calculates the total cargo load by summing the weight of containers associated with active bookings for each port.

4.SET OPERATIONS:

```
(SELECT
  'New Booking' AS activity,
  Bookings.booking_date_start AS timestamp,
  Ports.port_name,
  Bookings.booking_id AS reference_id
 FROM Bookings
 JOIN Ports ON Bookings.port_id = Ports.port_id
 ORDER BY Bookings.booking_date_start DESC)
UNION ALL
(SELECT
  'User Registration' AS activity,
  Users.registration_date AS timestamp,
  'System' AS port_name,
  Users.user_id AS reference_id
 FROM Users
 ORDER BY Users.registration_date DESC)
ORDER BY timestamp DESC;
```

Description:

1. First Query (Bookings):

- Retrieves booking information: booking_date_start, port_name, and booking_id.
- Labels this activity as "New Booking".
- Joins the Ports table to get the corresponding port_name for each booking.
- Orders by the booking_date_start in descending order.

2.Second Query (User Registration):

- Retrieves user registration information: registration_date and user_id.
- Labels this activity as "User Registration".
- Sets 'System' as the port_name (since it's not related to a port).
- Orders by the registration_date in descending order.

3.UNION ALL:

- Combines the two queries into one result set without removing duplicates.
- Sorts the combined results by the timestamp (either booking_date_start or registration_date) in descending order.

5.INSERT QUERIES:

```
INSERT INTO Users (username, email, password, role) VALUES (?, ?, ?, ?)

INSERT INTO Ports (port_name, capacity, available_space, location, country_id)
VALUES (?, ?, ?, ?, ?)

INSERT INTO Country (country_name) VALUES (?);

INSERT INTO Employee (first_name, last_name, position, salary, hire_date,
port_id)
VALUES (?, ?, ?, ?, ?, ?);

INSERT INTO Ships (ship_name, capacity, registration_number, owner, country_id,
port_id) VALUES (?, ?, ?, ?, ?, ?);

INSERT INTO Users (username, email, password, role) VALUES (?, ?, ?, ?);
```

6. UPDATE QUERIES:

```
UPDATE Users SET role = ? WHERE user_id = ?

UPDATE Ports SET port_name = ?, capacity = ?, available_space = ?, location = ?
WHERE port_id = ?

UPDATE Bookings SET booking_status = ? WHERE booking_id = ?
```

```
UPDATE Ports SET available_space = available_space + ? WHERE port_id = ?

UPDATE Country SET country_name = ? WHERE country_id = ?;

UPDATE Employee SET first_name = ?, last_name = ?, position = ?, salary = ?,
hire_date = ?, port_id = ? WHERE employee_id = ?

UPDATE Ships SET ship_name = ?, capacity = ?, registration_number = ?, owner = ?,
country_id = ?, port_id = ? WHERE ship_id = ?;

UPDATE Container SET container_type = ?, weight = ?, contents = ?, ship_id = ?,
booking_id = ? WHERE container_id = ?;

UPDATE Users SET username = ?, email = ? WHERE user_id = ?;
```

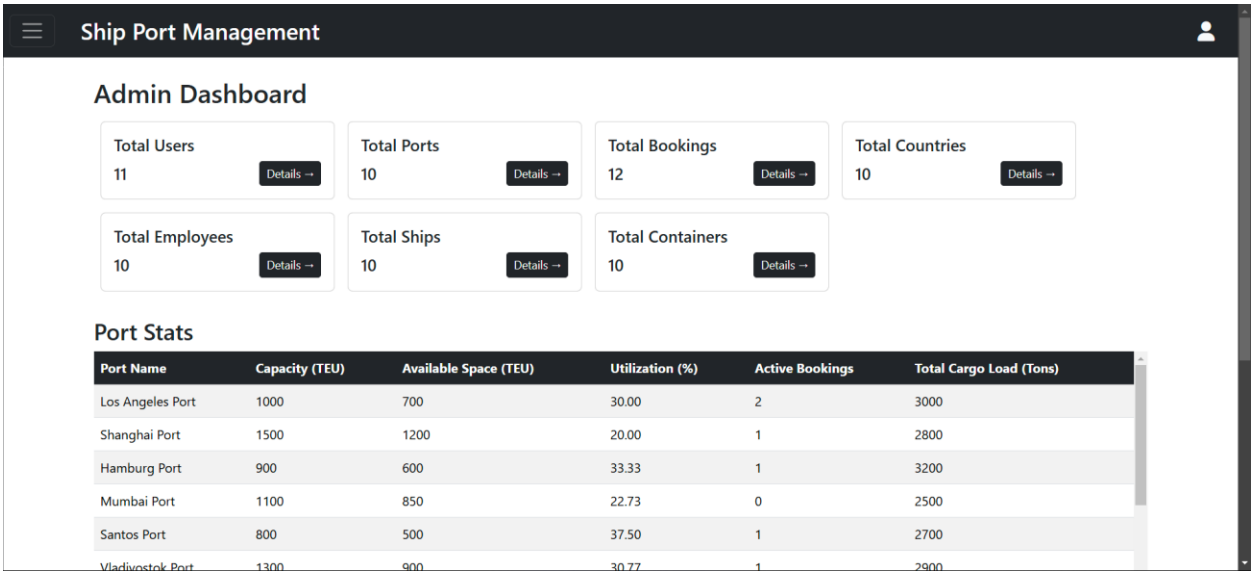
7.DELETE QUERIES:

```
DELETE FROM Users WHERE user_id = ?

DELETE FROM Ports WHERE port_id = ?

DELETE FROM Bookings WHERE booking_id = ?

DELETE FROM Country WHERE country_id = ?;

DELETE FROM Employee WHERE employee_id = ?;

DELETE FROM Ships WHERE ship_id = ?;

DELETE FROM Container WHERE container_id = ?;
```

I) Functionalities/Features

FRONTEND:

Admin:


Ship Port Management
Admin Dashboard
Total Users
11
Details →
Total Ports
10
Details →
Total Bookings
12
Details →
Total Countries
10
Details →
Total Employees
10
Details →
Total Ships
10
Details →
Total Containers
10
Details →
Port Stats
Port Name
Capacity (TEU)
Available Space (TEU)
Utilization (%)
Active Bookings
Total Cargo Load (Tons)
Los Angeles Port
1000
700
30.00
2
3000
Shanghai Port
1500
1200
20.00
1
2800
Hamburg Port
900
600
33.33
1
3200
Mumbai Port
1100
850
22.73
0
2500
Santos Port
800
500
37.50
1
2700
Vladivostok Port
1300
900
30.77
1
2900

Port Stats

Port Name	Capacity (TEU)	Available Space (TEU)	Utilization (%)	Active Bookings	Total Cargo Load (Tons)
Santos Port	800	500	37.50	1	2700
Vladivostok Port	1300	900	30.77	1	2900
Tokyo Port	1400	1100	21.43	1	2600
Vancouver Port	1000	700	30.00	1	2750
Sydney Port	950	650	31.58	1	3050
London Port	1050	800	23.81	0	2550

Activity Logs

Activity	Port Name	Reference ID	Timestamp
New Booking	Los Angeles Port	14	11/13/2024, 9:18:00 AM
New Booking	Los Angeles Port	15	11/13/2024, 12:00:00 AM
User Registration	System	11	11/12/2024, 11:29:31 PM
User Registration	System	10	11/12/2024, 11:29:00 PM
User Registration	System	9	11/12/2024, 11:28:38 PM
User Registration	System	8	11/12/2024, 11:28:22 PM

Ship Port Management

Manage Users

Search by ID, Username, or Email

All User Admin

Add New User

ID	Username	Email	Role	Actions
1	Admin	admin@test.com	admin	Edit Delete
2	User 1	user1@test.com	user	Edit Delete
3	User 2	user2@test.com	user	Edit Delete
4	User 3	user3@test.com	user	Edit Delete
5	User 4	user4@test.com	user	Edit Delete
6	User 5	user5@test.com	user	Edit Delete
7	User 6	userr6@test.com	user	Edit Delete

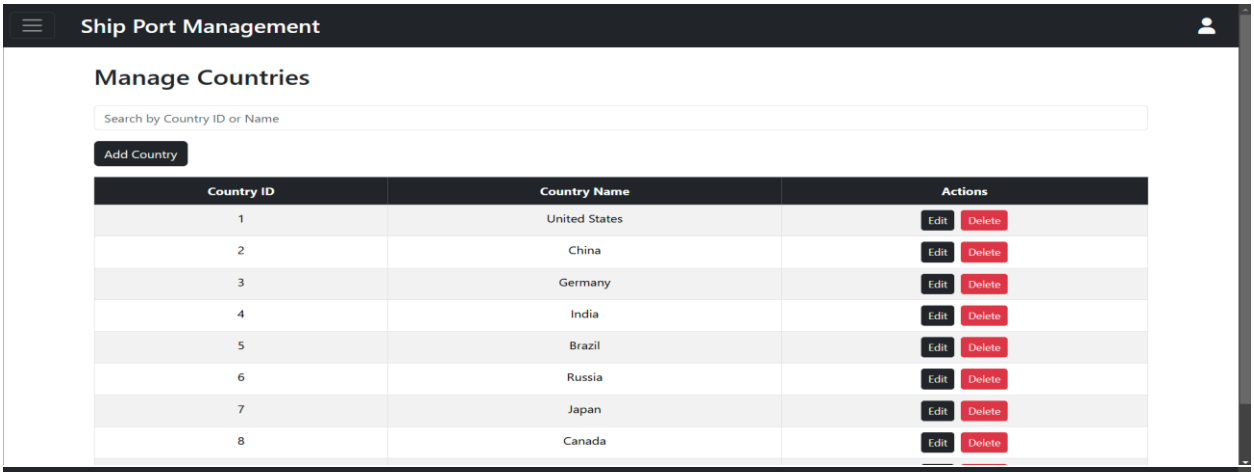
Ship Port Management

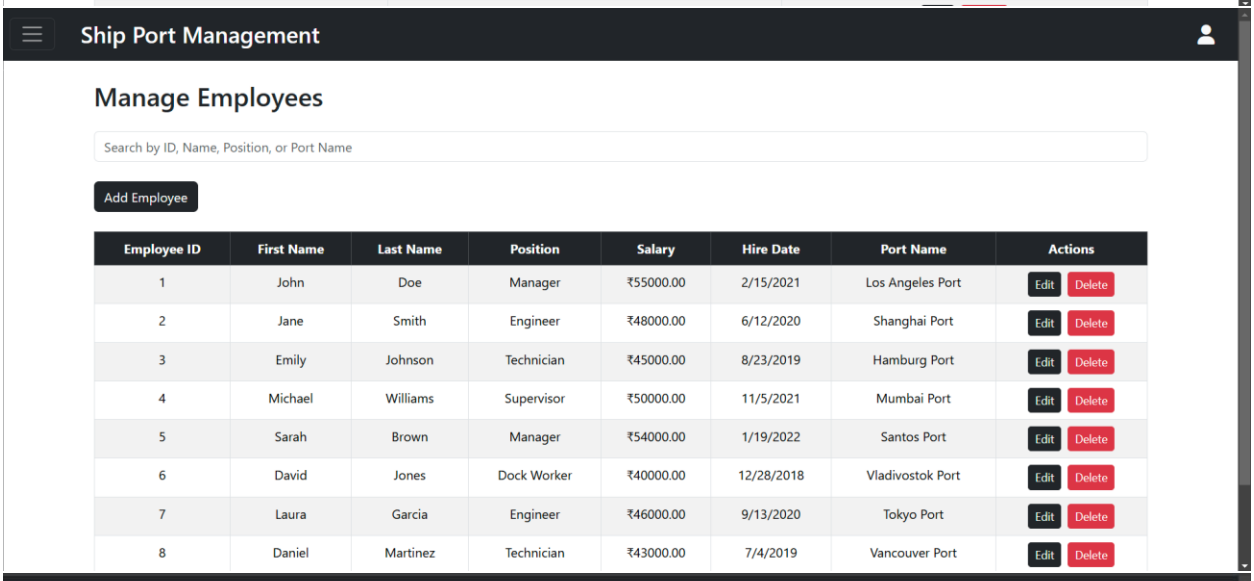
Manage Bookings

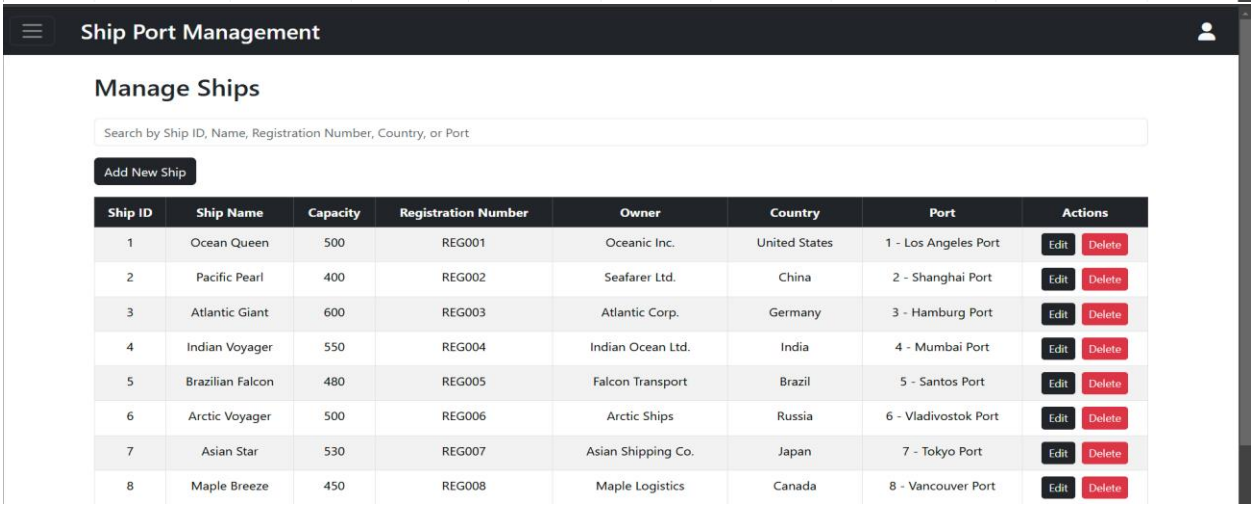
Search by Booking ID, User ID, Name, Port ID, Port Name, Ship ID, Ship Name

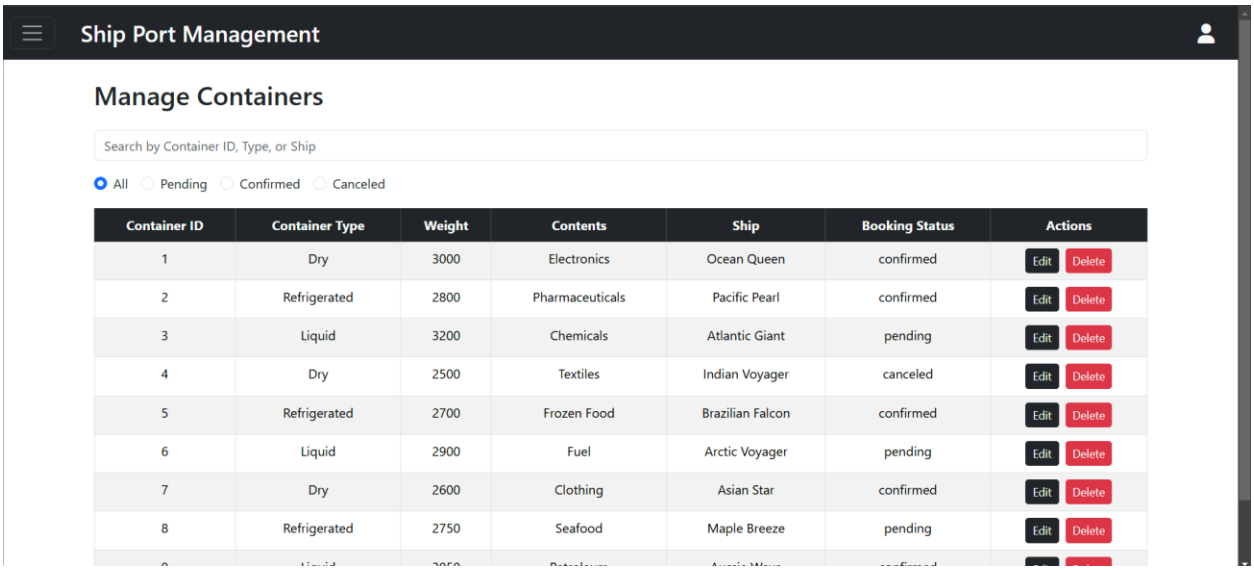
All Pending Confirmed Canceled

Booking ID	User	Port	Ship	From Booking Date	To Booking Date	Required Space	Status	Actions
1	2 - User 1	1 - Los Angeles Port	1 - Ocean Queen	5/10/2023	5/15/2023	200	confirmed	Edit Delete
2	3 - User 2	2 - Shanghai Port	2 - Pacific Pearl	6/12/2023	6/17/2023	150	confirmed	Edit Delete
3	4 - User 3	3 - Hamburg Port	3 - Atlantic Giant	7/14/2023	7/19/2023	180	pending	Edit Delete
4	5 - User 4	4 - Mumbai Port	4 - Indian Voyager	8/10/2023	8/15/2023	220	canceled	Edit Delete
5	6 - User 5	5 - Santos Port	5 - Brazilian Falcon	9/8/2023	9/13/2023	210	confirmed	Edit Delete
6	7 - User 6	6 - Vladivostok Port	6 - Arctic Voyager	10/18/2023	10/23/2023	190	pending	Edit Delete
7	8 - User 7	7 - Tokyo Port	7 - Asian Star	11/22/2023	11/27/2023	230	confirmed	Edit Delete
8	9 - User 8	8 - Vancouver Port	8 - Maple Breeze	12/1/2023	12/6/2023	170	pending	Edit Delete

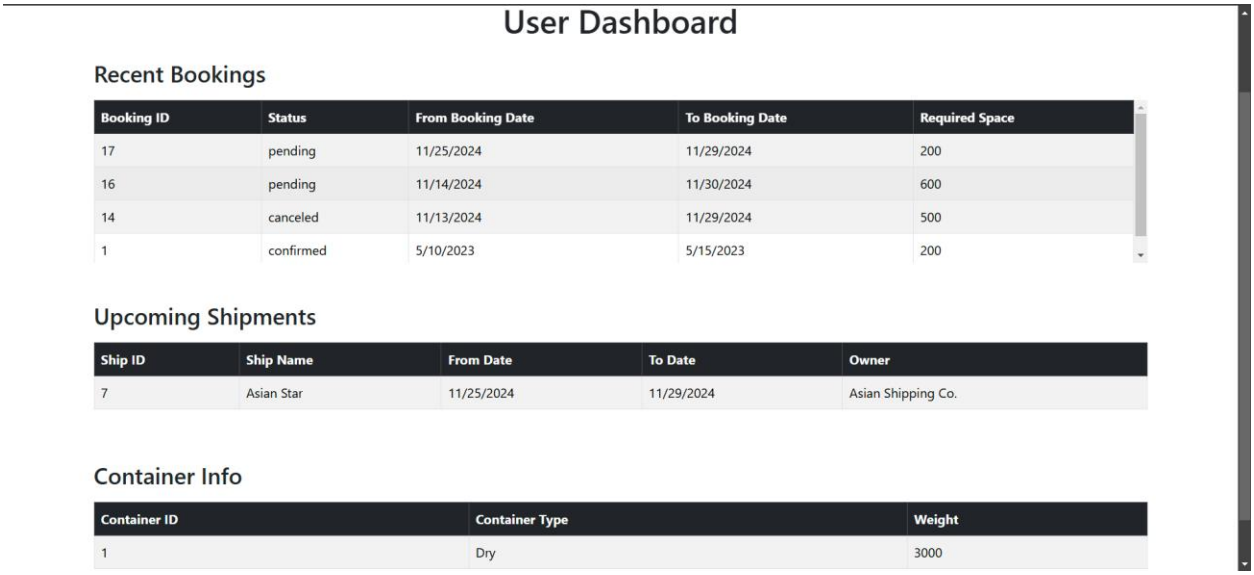

Ship Port Management
Manage Countries
Search by Country ID or Name
Add Country
Country ID
Country Name
Actions
1 United States Edit Delete
2 China Edit Delete
3 Germany Edit Delete
4 India Edit Delete
5 Brazil Edit Delete
6 Russia Edit Delete
7 Japan Edit Delete
8 Canada Edit Delete

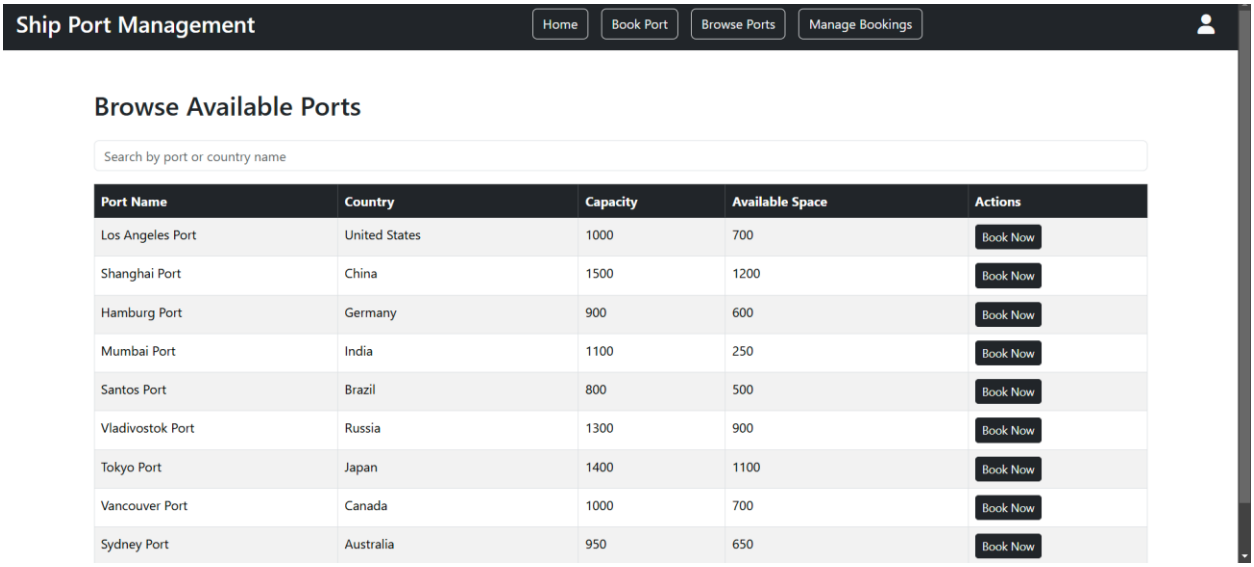

Ship Port Management
Manage Employees
Search by ID, Name, Position, or Port Name
Add Employee
Employee ID
First Name
Last Name
Position
Salary
Hire Date
Port Name
Actions
1 John Doe Manager ₹55000.00 2/15/2021 Los Angeles Port Edit Delete
2 Jane Smith Engineer ₹48000.00 6/12/2020 Shanghai Port Edit Delete
3 Emily Johnson Technician ₹45000.00 8/23/2019 Hamburg Port Edit Delete
4 Michael Williams Supervisor ₹50000.00 11/5/2021 Mumbai Port Edit Delete
5 Sarah Brown Manager ₹54000.00 1/19/2022 Santos Port Edit Delete
6 David Jones Dock Worker ₹40000.00 12/28/2018 Vladivostok Port Edit Delete
7 Laura Garcia Engineer ₹46000.00 9/13/2020 Tokyo Port Edit Delete
8 Daniel Martinez Technician ₹43000.00 7/4/2019 Vancouver Port Edit Delete


Ship Port Management
Manage Ships
Search by Ship ID, Name, Registration Number, Country, or Port
Add New Ship
Ship ID
Ship Name
Capacity
Registration Number
Owner
Country
Port
Actions
1 Ocean Queen 500 REG001 Oceanic Inc. United States 1 - Los Angeles Port Edit Delete
2 Pacific Pearl 400 REG002 Seafarer Ltd. China 2 - Shanghai Port Edit Delete
3 Atlantic Giant 600 REG003 Atlantic Corp. Germany 3 - Hamburg Port Edit Delete
4 Indian Voyager 550 REG004 Indian Ocean Ltd. India 4 - Mumbai Port Edit Delete
5 Brazilian Falcon 480 REG005 Falcon Transport Brazil 5 - Santos Port Edit Delete
6 Arctic Voyager 500 REG006 Arctic Ships Russia 6 - Vladivostok Port Edit Delete
7 Asian Star 530 REG007 Asian Shipping Co. Japan 7 - Tokyo Port Edit Delete
8 Maple Breeze 450 REG008 Maple Logistics Canada 8 - Vancouver Port Edit Delete

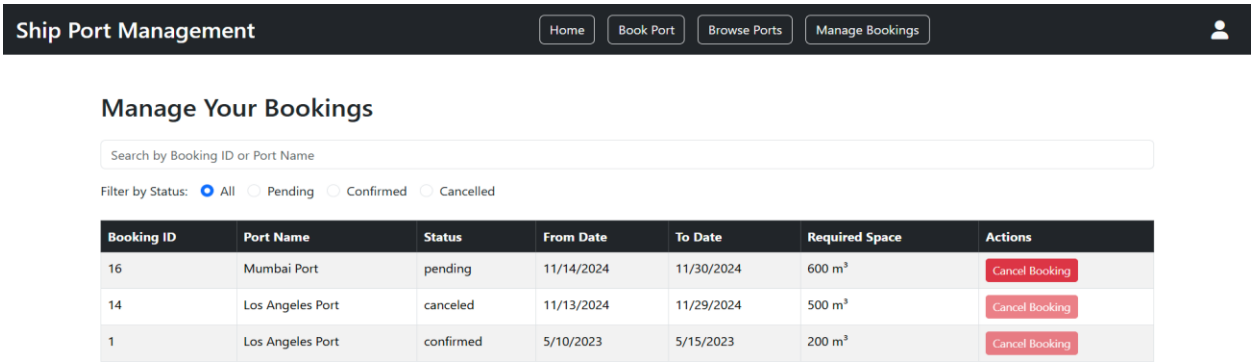

Ship Port Management
Manage Containers
Search by Container ID, Type, or Ship
All Pending Confirmed Canceled
Container ID Container Type Weight Contents Ship Booking Status Actions
1 Dry 3000 Electronics Ocean Queen confirmed Edit Delete
2 Refrigerated 2800 Pharmaceuticals Pacific Pearl confirmed Edit Delete
3 Liquid 3200 Chemicals Atlantic Giant pending Edit Delete
4 Dry 2500 Textiles Indian Voyager canceled Edit Delete
5 Refrigerated 2700 Frozen Food Brazilian Falcon confirmed Edit Delete
6 Liquid 2900 Fuel Arctic Voyager pending Edit Delete
7 Dry 2600 Clothing Asian Star confirmed Edit Delete
8 Refrigerated 2750 Seafood Maple Breeze pending Edit Delete

User:


User Dashboard
Recent Bookings
Booking ID Status From Booking Date To Booking Date Required Space
17 pending 11/25/2024 11/29/2024 200
16 pending 11/14/2024 11/30/2024 600
14 canceled 11/13/2024 11/29/2024 500
1 confirmed 5/10/2023 5/15/2023 200
Upcoming Shipments
Ship ID Ship Name From Date To Date Owner
7 Asian Star 11/25/2024 11/29/2024 Asian Shipping Co.
Container Info
Container ID Container Type Weight
1 Dry 3000


Ship Port Management
Home
Book Port
Browse Ports
Manage Bookings
Browse Available Ports
Search by port or country name
Port Name
Country
Capacity
Available Space
Actions
Los Angeles Port
United States
1000
700
Book Now
Shanghai Port
China
1500
1200
Book Now
Hamburg Port
Germany
900
600
Book Now
Mumbai Port
India
1100
250
Book Now
Santos Port
Brazil
800
500
Book Now
Vladivostok Port
Russia
1300
900
Book Now
Tokyo Port
Japan
1400
1100
Book Now
Vancouver Port
Canada
1000
700
Book Now
Sydney Port
Australia
950
650
Book Now


Book a Port
Booking successful!
Select Port
Choose a port...
Select Ship
Choose a ship...
Required Space (in cubic meters)
0
From Booking Date
dd-----yyyy --:-- --
To Booking Date
dd-----yyyy --:-- --
Book Port


Ship Port Management
Home
Book Port
Browse Ports
Manage Bookings
Manage Your Bookings
Search by Booking ID or Port Name
Filter by Status:
All
Pending
Confirmed
Cancelled
Booking ID
Port Name
Status
From Date
To Date
Required Space
Actions
16
Mumbai Port
pending
11/14/2024
11/30/2024
600 m³
Cancel Booking
14
Los Angeles Port
canceled
11/13/2024
11/29/2024
500 m³
Cancel Booking
1
Los Angeles Port
confirmed
5/10/2023
5/15/2023
200 m³
Cancel Booking

J) Triggers, Procedure and Functions

1. PROCEDURE

```
-- Procedure for adding booking
DELIMITER //
CREATE PROCEDURE add_booking(
  IN p_user_id INT,
  IN p_port_id INT,
  IN p_ship_id INT,
  IN p_start_date TIMESTAMP,
  IN p_end_date TIMESTAMP,
  IN p_required_space INT
)
BEGIN
  DECLARE available INT;
  DECLARE new_booking_id INT;

  SELECT available_space INTO available FROM Ports WHERE port_id = p_port_id;

  IF available >= p_required_space THEN
    INSERT INTO Bookings (user_id, port_id, ship_id, booking_date_start,
booking_date_end, booking_status, required_space)
    VALUES (p_user_id, p_port_id, p_ship_id, p_start_date, p_end_date, 'pending',
p_required_space);

    SET new_booking_id = LAST_INSERT_ID();

    UPDATE Ports
    SET available_space = available_space - p_required_space
    WHERE port_id = p_port_id;

    SELECT 'Booking created successfully' AS message, new_booking_id AS
booking_id;
  ELSE
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Insufficient space available at the selected port for
this booking.';
  END IF;
END //
DELIMITER ;
```

Description:

The procedure is responsible for adding a booking to the system, ensuring that the required space is available at the selected port, and adjusting the

port's available space accordingly. If there isn't enough space, it raises an error.

2. FUNCTIONS

```
-- Function to calculate port utilization
DELIMITER //
CREATE FUNCTION calculate_port_utilization(pid INT)
RETURNS DECIMAL(5,2)
DETERMINISTIC
BEGIN
  DECLARE total_capacity INT;
  DECLARE space_available INT;
  DECLARE utilization DECIMAL(5,2);

  SELECT capacity, available_space INTO total_capacity, space_available
  FROM Ports
  WHERE port_id = pid;

  IF total_capacity > 0 THEN
    SET utilization = ((total_capacity - space_available) / total_capacity) *
100;
  ELSE
    SET utilization = 0;
  END IF;

  RETURN utilization;
END //
DELIMITER ;
```

Description:

This function calculates the percentage of a port's capacity that is currently being used by comparing the total capacity to the available space. It returns this value as a percentage of utilization. If the port has no capacity, it returns a utilization of 0%.

3. TRIGGERS

i) Trigger for preventing deletion of country if associated with any port or ship

```sql
-- Trigger for preventing deletion of country if associated with any port or ship
DELIMITER //
CREATE TRIGGER
BEFORE DELETE ON Country
FOR EACH ROW
BEGIN
  DECLARE port_count INT;
  DECLARE ship_count INT;

  SELECT COUNT(*) INTO port_count FROM Ports WHERE country_id = OLD.country_id;
  SELECT COUNT(*) INTO ship_count FROM Ships WHERE country_id = OLD.country_id;

  IF port_count > 0 OR ship_count > 0 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Cannot delete country with associated ports or ships.';
  END IF;
END //
DELIMITER ;
```

Description:

This trigger ensures that a country cannot be deleted if there are any associated ports or ships. If such associations exist, the trigger raises an error message, preventing the deletion and maintaining data integrity.

ii) Trigger for prevention of deletion of ports if associated with any ship

```sql
-- Trigger for prevention of deletion of ports if associated with any ship
DELIMITER //
CREATE TRIGGER prevent_port_deletion
BEFORE DELETE ON Ports
FOR EACH ROW
BEGIN
  DECLARE ship_count INT;

  SELECT COUNT(*) INTO ship_count FROM Ships WHERE port_id = OLD.port_id;

  IF ship_count > 0 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Cannot delete port with associated ships.';
  END IF;
END //
DELIMITER ;
```

Description:

This trigger prevents the deletion of a port if there are ships associated with it. If such ships exist, the trigger raises an error with a custom message, ensuring that ports with active ship associations cannot be deleted.

K) Code Snippets For Invoking Procedure/Functions and Trigger

i) functions

```
getStats: async (req, res) => {
    const statsQuery = `
      SELECT
        Ports.port_id,
        Ports.port_name,
        Ports.capacity,
        Ports.available_space,
        calculate_port_utilization(Ports.port_id) AS capacityUtilization,
        (SELECT COUNT(*)
        FROM Bookings
        WHERE Bookings.port_id = Ports.port_id
        AND (booking_status = 'confirmed' OR booking_status = 'pending')) AS
activeBookings,
        (SELECT SUM(Container.weight)
        FROM Container
        INNER JOIN Bookings ON Container.booking_id = Bookings.booking_id
        WHERE Bookings.port_id = Ports.port_id) AS totalCargoLoad
      FROM Ports;

    `;
    db.query(statsQuery, (err, results) => {
      if (err) {
        console.error("Error fetching stats:", err);
        return res
          .status(500)
          .json({ message: "Error fetching stats", error: err.message });
      }
      res.json({ data: results });
    });
  },
```

->Above the function port_utilization is invoked.

ii) Procedure

```
exports.bookPort = (req, res) => {
  const { user_id, port_id, ship_id, booking_date_start, booking_date_end,
required_space } = req.body;

  const addBookingQuery = `
    CALL add_booking(?, ?, ?, ?, ?, ?)
  `;

  connection.query(
    addBookingQuery,
    [user_id, port_id, ship_id, booking_date_start, booking_date_end,
required_space],
    (err, results) => {
      if (err) {
        return res.status(500).json({ error: "Error creating booking" });
      }

      res.status(201).json({
        message: "Booking created successfully",
        bookingId: results[0][0].booking_id,
      });
    }
  );
};
```

-> Above the procedure add_booking is invoked

iii) Triggers

First Trigger

```
  deletePort: async (req, res) => {
    const { port_id } = req.params;

    db.query(
      "DELETE FROM Ports WHERE port_id = ?",
      [port_id],
      (err, results) => {
        if (err) {
          console.error("Error deleting port:", err);
          return res
            .status(500)
            .json({ message: "Error deleting port", error: err.message });
        }
```

```
        if (results.affectedRows === 0) {
          return res.status(404).json({ message: "Port not found" });
        }
        res.json({ message: "Port deleted successfully" });
      }
    );
  },
```

-> Above the trigger prevent_port_deletion trigger is invoked

Second Trigger:

```
  deleteCountries: async (req, res) => {
    const { country_id } = req.params;

    db.query(
      "DELETE FROM Country WHERE country_id = ?",
      [country_id],
      (err, results) => {
        if (err) {
          console.error("Error deleting countries:", err);
          return res
            .status(500)
            .json({ message: "Error deleting countries", error: err.message });
        }
        if (results.affectedRows === 0) {
          return res.status(404).json({ message: "Country not found" });
        }
        res.json({ message: "Country deleted successfully" });
      }
    );
  },
```

-> Above the trigger prevent_country_deletion is invoked

L) Github Repo Link:

Link-> https://github.com/varuns2903/Shipping-Port-Management-System.git