IT FINAL YEAR PROJECT

Name: Jeff Tinodashe Nyahuye StudentNo: 402101170 Class: Bsc IT 3rd year

Table of Contents

Content	Page
Phase 1: Proposal	2
Phase 2: Planning	4
Phase 3: Analysis	10
Phase 4: System Design	14
Phase 5: Implementation	17
Conclusion	56
Bibliography	57

Phase 1: Project Proposal

Introduction

The purpose of this project proposal is to outline the development of an inventory management website that allows the remote management of an inventory by multiple users. This is a solution to Nonprofit organizations, restaurant owners, small business owners who seek an efficient solution for tracking daily inventory usage and generating reports.

Aim

The primary aim of this project is to create a user-friendly inventory management website that allows managers to record inventory usage and generate reports. This will enable managers and small business owners to make data-driven decisions about restocking supplies and reduce wastage, ultimately improving profitability and customer satisfaction.

Problem Definition

Small business owners and non-profit organizations currently lack an effective digital inventory management system, leading to inefficiencies in tracking inventory levels, ordering supplies and making informed decisions. Manual record-keeping is time-consuming, error-prone, not easily sharable among managers and does not provide real-time data insights. This results sometimes in overstocking, understocking, waste of resources and sometimes theft of resources as they are not accounted for.

Hypothesis

Implementing an inventory management website will allow managers to manage inventory remotely, improve accuracy in tracking inventory levels, reduce the likelihood of overstocking and understocking, enhance overall operational efficiency and minimize waste. The added benefit of creating a report anywhere anytime also gifts users with an important tool in their toolkit.

Objectives

The project's key objectives are as follows:

- Develop a secure and user-friendly inventory management website.
- Allow restaurant managers to log in and update inventory usage daily.
- Provide real-time inventory data to the managers.
- Allow the inventory to generate reports in either pdf or other formats.

Justification

The justification for this project lies in the need to optimize inventory management processes. By implementing an inventory management website, managers can reduce operational costs by avoiding excess inventory and waste. It can also increase competitiveness by making data-driven decisions. Finally, the system can enhance the owner's ability to plan and budget effectively.

Expectations

Upon successful completion of this project, I anticipate that I will have a functional inventory management website that can help managers easily update daily inventory usage. Users will also be able to generate inventory reports that they can use to make analysis.

Conclusion

In conclusion, the development of an inventory management website is a crucial step towards improving its overall efficiency of any organization. By addressing the current inventory management challenges, I aim to optimize stock levels, reduce waste, and enhance customer satisfaction. This project aligns with the restaurant owner's vision of sustainable growth and will have a positive impact on the restaurant's long-term success.

Phase 2: Project Planning

Identification of Need

The need for an inventory management system arises from the current challenges faced by small businesses in managing inventory. The identified needs include remote tracking of inventory levels, accurate tracking of inventory levels, efficient management of stock and generating reports of stock.

Preliminary Investigation

During the preliminary investigation, I conducted interviews and random surveys from small business owners and nonprofit organizations. From the data collected I assessed the current inventory management processes.

The investigation revealed that most organizations still use manual inventory tracking processes which are prone to errors and inefficiencies. This manual tracking also lacks real-time data for decision-making. Ultimately these business owners have a hard time keeping track of their inventory remotely.

In conclusion of the investigation, I have found out that managers and users are willing to adopt a digital solution.

Feasibility Study

I conducted and three types of feasibility studies and from them I learnt the following: -

The technological feasibility study showed that the required technology stack, to make an online inventory system, is readily available and within budget. I also discovered that I possess the necessary technical skills and expertise to implement such a system. Finally, integration with any existing systems (if any) can be achieved without major hurdles and draw backs.

From the operational feasibility study, I learnt that users, organizations and small businesses are open to adopting the system. In addition, learning material on how to use the system will be greatly appreciated.

Finally, from the economic feasibility study I learnt that the cost-benefit analysis indicates that the benefits (cost savings, reduced waste and time saved not using manual record keeping) outweigh the development and implementation costs. A positive return on investment would be anticipated within a reasonable timeframe.

Project Planning

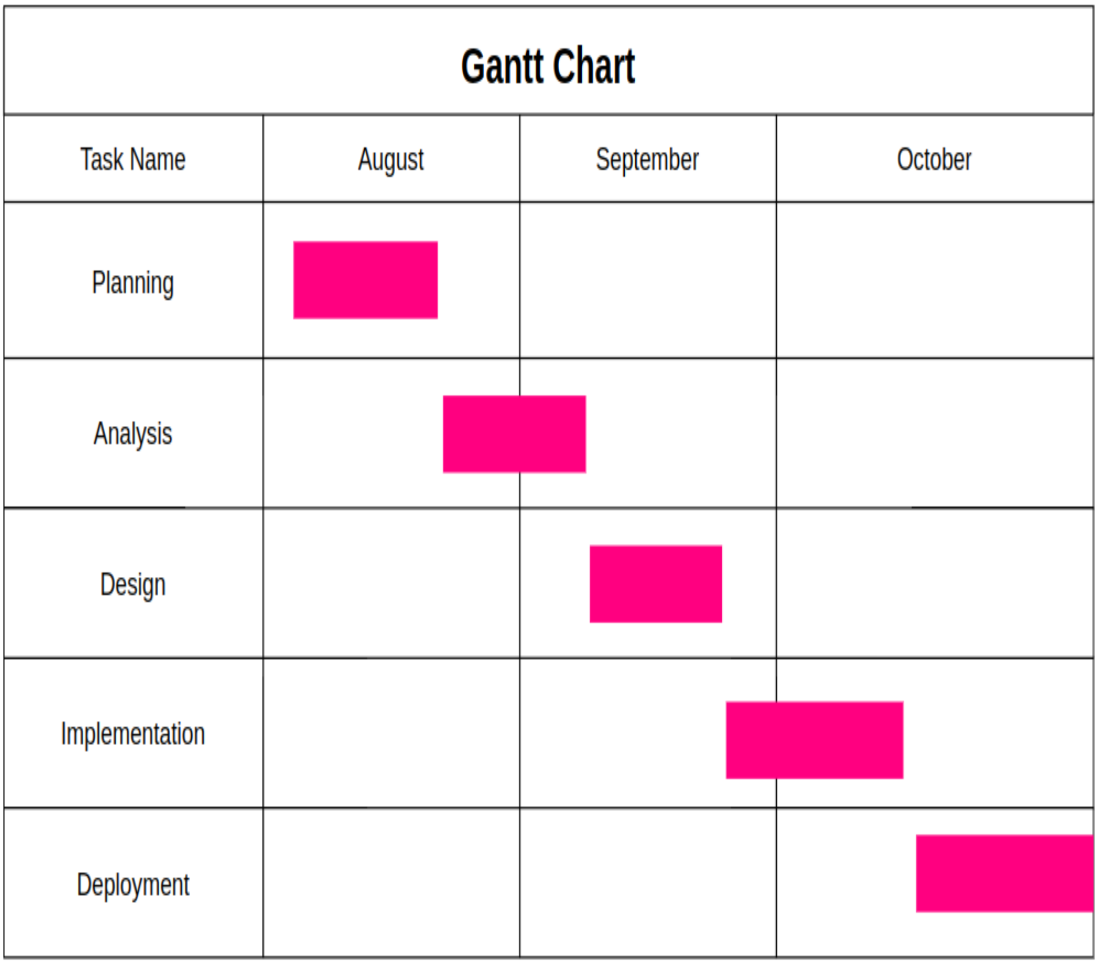
I decided to name the project, Jeff's Inventory Management System (JIMS), after the author.

The project overview will encompass developing a web-based inventory management system using React for the front-end, Express for the back end, and MySQL as the database. The system will allow organization managers to track and update inventory usage.

Furthermore, the project will be divided into several phases which include Planning Phases, Analysis Phases, Implementation phase, deployment phase and maintenance phase.

Project Scheduling

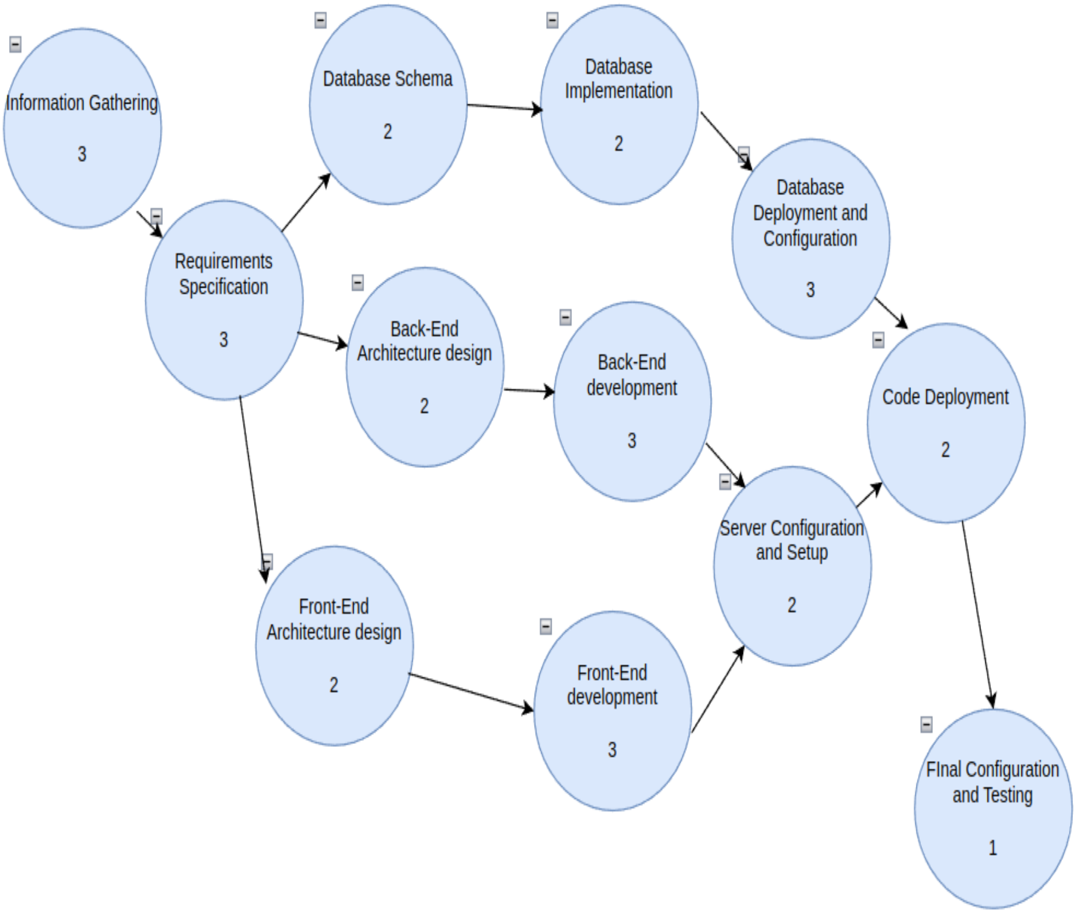
Below is a Gantt Chart showing the project's timeline: -



In addition, a more detailed Pert Chart to shows the dependencies of the phases relative to the project timeline. The critical points of the project are in blue circles and the estimated numbers written withing them below: -

Pert Chart

Legend	
Critical Point	⬭
Units	Days


Information Gathering
3
Requirements Specification
3
Database Schema
2
Database Implementation
2
Database Deployment and Configuration
3
Back-End Architecture design
2
Back-End development
3
Front-End Architecture design
2
Front-End development
3
Server Configuration and Setup
2
Code Deployment
2
FInal Configuration and Testing
1

Software Requirement Specification (SRS):

Introduction to Software Requirements specifications

The Jeff's Restaurant Inventory Management System (JIMS) is a web-based application designed to streamline inventory management processes for organizations. Below are the highlighted functional and non-functional requirements of the system.

Functional Requirements

The most important of the function requirements is that users must be able to log in with unique usernames and passwords, otherwise the website must be inaccessible. Also, failed login attempts should trigger appropriate error messages. In addition, users should be able to log out securely.

Another one of the functional requirements is that managers should be able to view the current inventory status, update items and remove them. The inventory status should include item names and quantities. In addition, Managers must have the ability to add or remove users. Managers should be able to generate daily, weekly, and monthly inventory usage reports.

Non-functional Requirements

For non-function requirements the system shall respond to user actions within 2 seconds. The system must also prevent SQL injection through input sanitization. Finally, the system should be designed to accommodate future growth in terms of users and inventory items.

System Architecture Requirements

In relation to the system Architecture, the front-end will be developed using React to provide a responsive user interface. The back-end will be developed using Express.js to handle API requests. Finally, MySQL will be used for data storage.

User Interfaces Requirements

The login page should provide a secure login mechanism for users. The dashboard should display current inventory status. Users should be able to update inventory from this interface. Managers should also be able to view users and delete or add users.

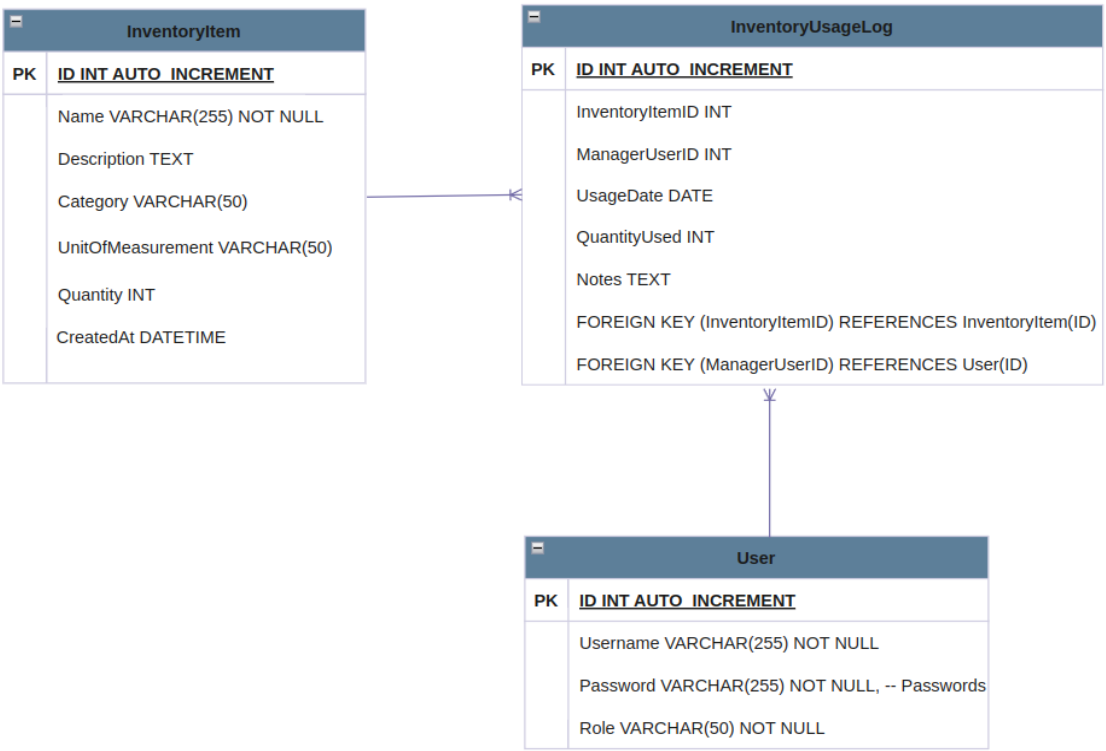
Data Models Requirements

For the users there should be fields for the ID, username and passwords. For the Inventory items there should be fields for ID, item name, quantity, reorder point, date.

Data Models

Below is a simplified Data Model for the MySQL database for Jeff’s Inventory Management System: -

ER Diagram for the data model: -



InventoryItem	
PK	ID INT AUTO_INCREMENT
	Name VARCHAR(255) NOT NULL
	Description TEXT
	Category VARCHAR(50)
	UnitOfMeasurement VARCHAR(50)
	Quantity INT
	CreatedAt DATETIME

InventoryUsageLog	
PK	ID INT AUTO_INCREMENT
	InventoryItemID INT
	ManagerUserID INT
	UsageDate DATE
	QuantityUsed INT
	Notes TEXT
	FOREIGN KEY (InventoryItemID) REFERENCES InventoryItem(ID)
	FOREIGN KEY (ManagerUserID) REFERENCES User(ID)

User	
PK	ID INT AUTO_INCREMENT
	Username VARCHAR(255) NOT NULL
	Password VARCHAR(255) NOT NULL, -- Passwords
	Role VARCHAR(50) NOT NULL

Phase 3: Analysis Phase

Introduction to Analysis

The purpose of this report is to provide a comprehensive analysis of traditional manual inventory management systems. It aims to identify the system's weaknesses and inefficiencies, laying the foundation for the design and development of an improved system.

The primary project objectives include streamlining inventory management, enhancing it, reducing operational inefficiencies and improving reporting capabilities.

Information Gathering

The data was collected using observations of a nonprofit church, Interviews of a small business owner and examination of historical inventory data from other related small organizations. The information collected was in relation to how inventory records are kept and managed, and the who has access to them.

Analysis of the Data Collected

Managers of organizations and small businesses use handwritten ledger to track inventory. The analysis showed that new paper sheets must constantly be used to keep track of inventory. It also showed that reports must be typed in a spreadsheets or pictures taken in order to store them online or send the information to someone else. The analysis also showed that printing or generating reports of the inventory is a tedious process that requires someone manually typing a word document from scratch every time.

Weaknesses of the Current System

With the current inventory system business owners cannot see the inventory status remotely. There is also a risk of perishable items expiring due to poor tracking. Additionally, there is a limited capability to generate detailed reports. Conducting the manual processes such as writing is hard and consumes excessive time. Also, there is risk of human errors that can lead to overstocking or understocking. In addition, the inventory documents are capable of misplacement or loss. The inventory documents are also not easily sharable, and not readily available. Finally, the inventory lists are not safe and secure, as anyone can easily open them and edit the books.

Proposed System Objectives

The proposed system aims to mitigate these weaknesses by digitalizing inventory management. It aims to make the inventory accessible online from anywhere by authorized users which in turn will improve overall operational efficiency. It also aims to create a secure login system, safely storing the inventory lists.

Functional Requirements

From the analysis the functional requirements should include user authentication where users can register by providing a username, password, and email. Registered users can log in using their credentials. Authentication should be secure and user information should be stored securely.

Users should be able to add new inventory items, including details like item name, quantity, unit price, and category. Users can update existing inventory items. Users can delete inventory items. The system should maintain a history of inventory item changes.

Users must also have the ability to generate inventory reports that should be exportable in common formats (e.g., png, PDF and txt).

Non-Functional Requirements

The system should provide a responsive and efficient user experience. Inventory reports should also be generated in a reasonable time frame.

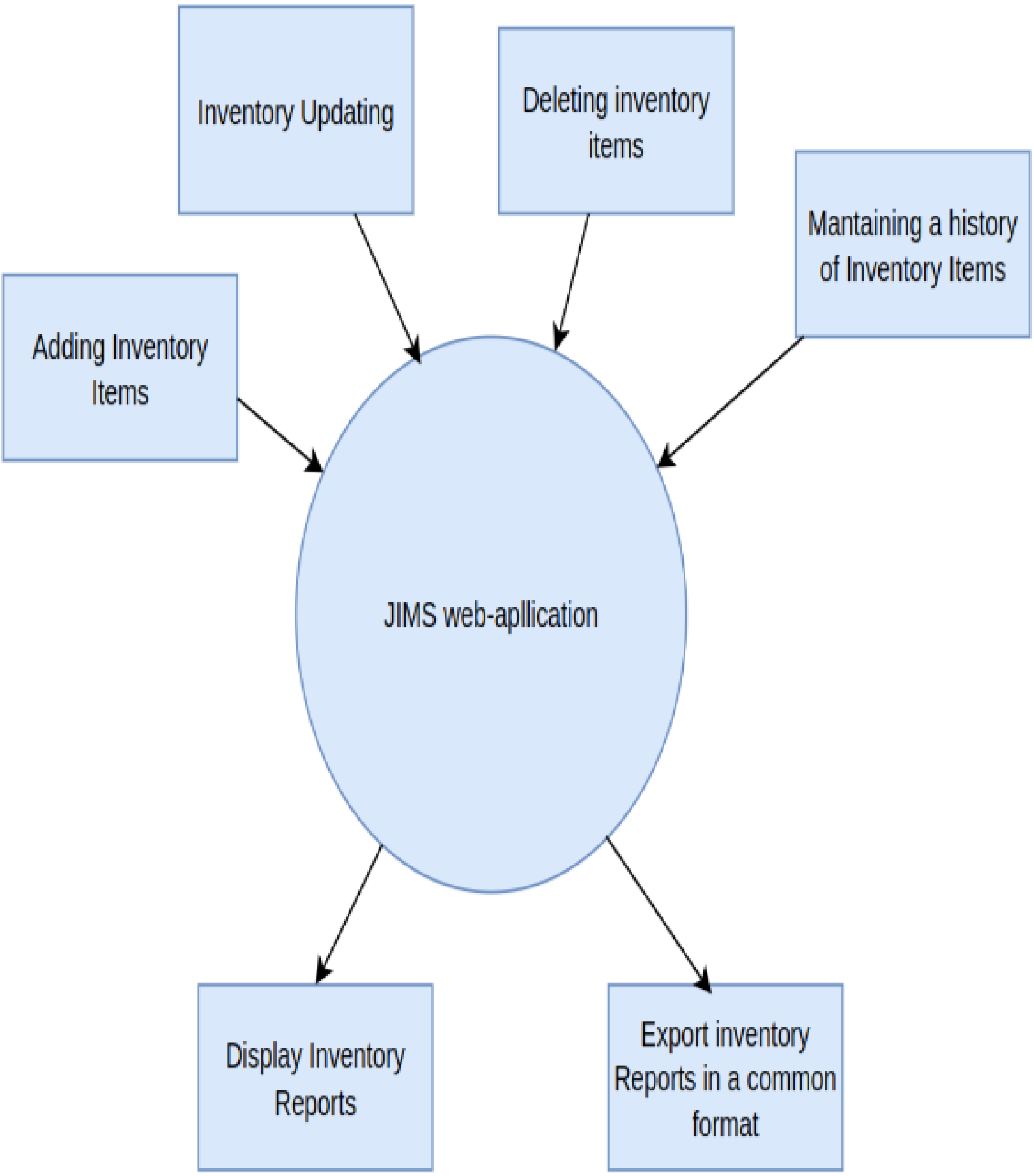
User authentication data should also be securely stored and transmitted. Only authorized users can perform inventory management operations. Finally, the system must be protected against common security vulnerabilities (e.g., SQL injection, XSS).

The system should handle a growing volume of inventory data without significant performance degradation. The system must also be able to handle new features. The user interface should be intuitive and user-friendly. In addition, adequate feedback should be provided for user actions and errors. Also, regular backups of the database should be performed to prevent data loss. The system should also have a data recovery mechanism in case of failures. Finally, the system should be compatible with modern web browsers (e.g., Chrome, Firefox, Safari).

Data modeling for proposed system

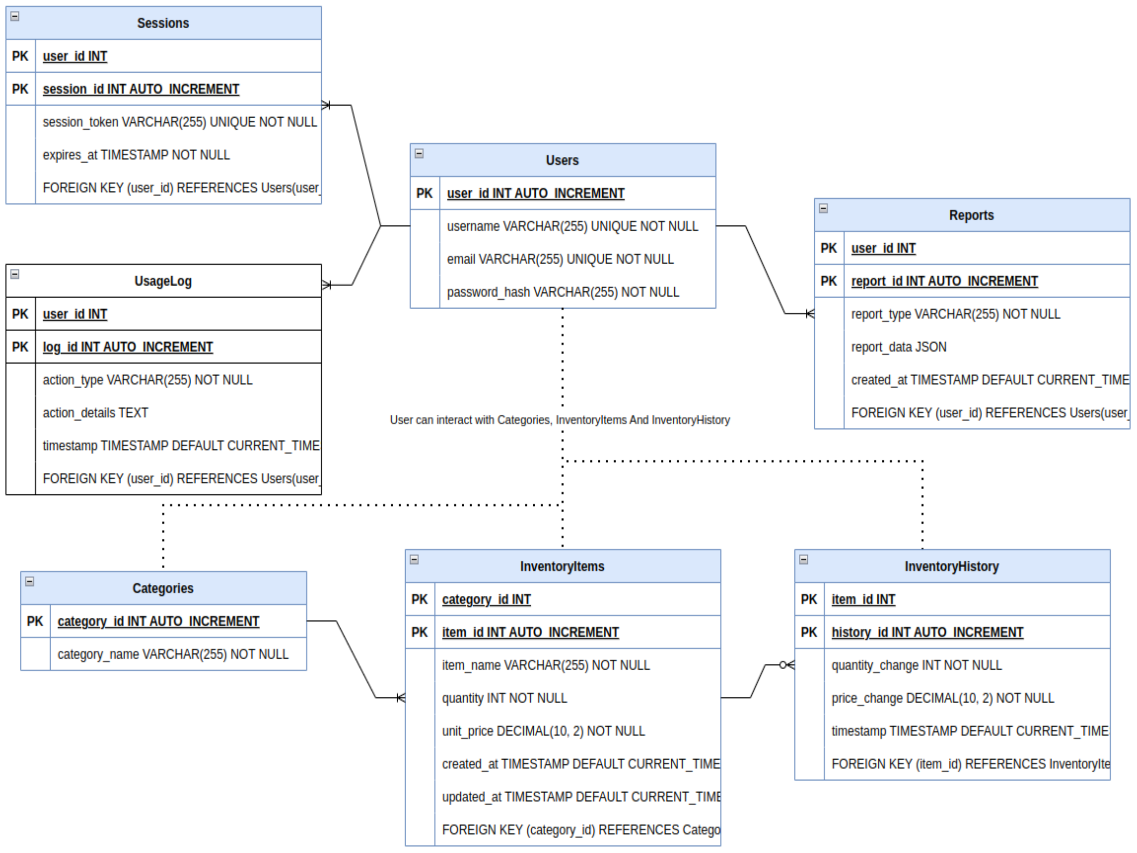
The data model below shows how the system functions and what are its expected inputs and outputs: -

Diagram Showing the functions of the Inventory Management System


Inventory Updating
Deleting inventory items
Mantaining a history of Inventory Items
Adding Inventory Items
JIMS web-apllication
Display Inventory Reports
Export inventory Reports in a common format

The database model below shows the database schema of the JIMS system: -

Jeff's Inventory Management System Database Schema



	Sessions
PK	**user_id INT**
PK	**session_id INT AUTO_INCREMENT**
	session_token VARCHAR(255) UNIQUE NOT NULL
	expires_at TIMESTAMP NOT NULL
	FOREIGN KEY (user_id) REFERENCES Users(user_

	Users
PK	**user_id INT AUTO_INCREMENT**
	username VARCHAR(255) UNIQUE NOT NULL
	email VARCHAR(255) UNIQUE NOT NULL
	password_hash VARCHAR(255) NOT NULL

	Reports
PK	**user_id INT**
PK	**report_id INT AUTO_INCREMENT**
	report_type VARCHAR(255) NOT NULL
	report_data JSON
	created_at TIMESTAMP DEFAULT CURRENT_TIME
	FOREIGN KEY (user_id) REFERENCES Users(user_

	UsageLog
PK	**user_id INT**
PK	**log_id INT AUTO_INCREMENT**
	action_type VARCHAR(255) NOT NULL
	action_details TEXT
	timestamp TIMESTAMP DEFAULT CURRENT_TIME
	FOREIGN KEY (user_id) REFERENCES Users(user_

User can interact with Categories, InventoryItems And InventoryHistory

	Categories
PK	**category_id INT AUTO_INCREMENT**
	category_name VARCHAR(255) NOT NULL

	InventoryItems
PK	**category_id INT**
PK	**item_id INT AUTO_INCREMENT**
	item_name VARCHAR(255) NOT NULL
	quantity INT NOT NULL
	unit_price DECIMAL(10, 2) NOT NULL
	created_at TIMESTAMP DEFAULT CURRENT_TIME
	updated_at TIMESTAMP DEFAULT CURRENT_TIME
	FOREIGN KEY (category_id) REFERENCES Catego

	InventoryHistory
PK	**item_id INT**
PK	**history_id INT AUTO_INCREMENT**
	quantity_change INT NOT NULL
	price_change DECIMAL(10, 2) NOT NULL
	timestamp TIMESTAMP DEFAULT CURRENT_TIME
	FOREIGN KEY (item_id) REFERENCES InventoryIte

Phase 4: System Design Phase

Purpose

The purpose of the system design is to outline the design and architecture of the proposed Inventory Management System. This system aims to address weaknesses related to manual inventory management, reporting, and operational efficiency while ensuring security and scalability.

System Design

The proposed Restaurant Management System is a cutting-edge, web-based application designed to automate inventory management, enhance reporting capabilities, reduce employee training needs, improve operational efficiency, provide secure login features, and offer remote accessibility. The system will consist of three main components: User Authentication, Inventory Management, and Inventory Reporting.

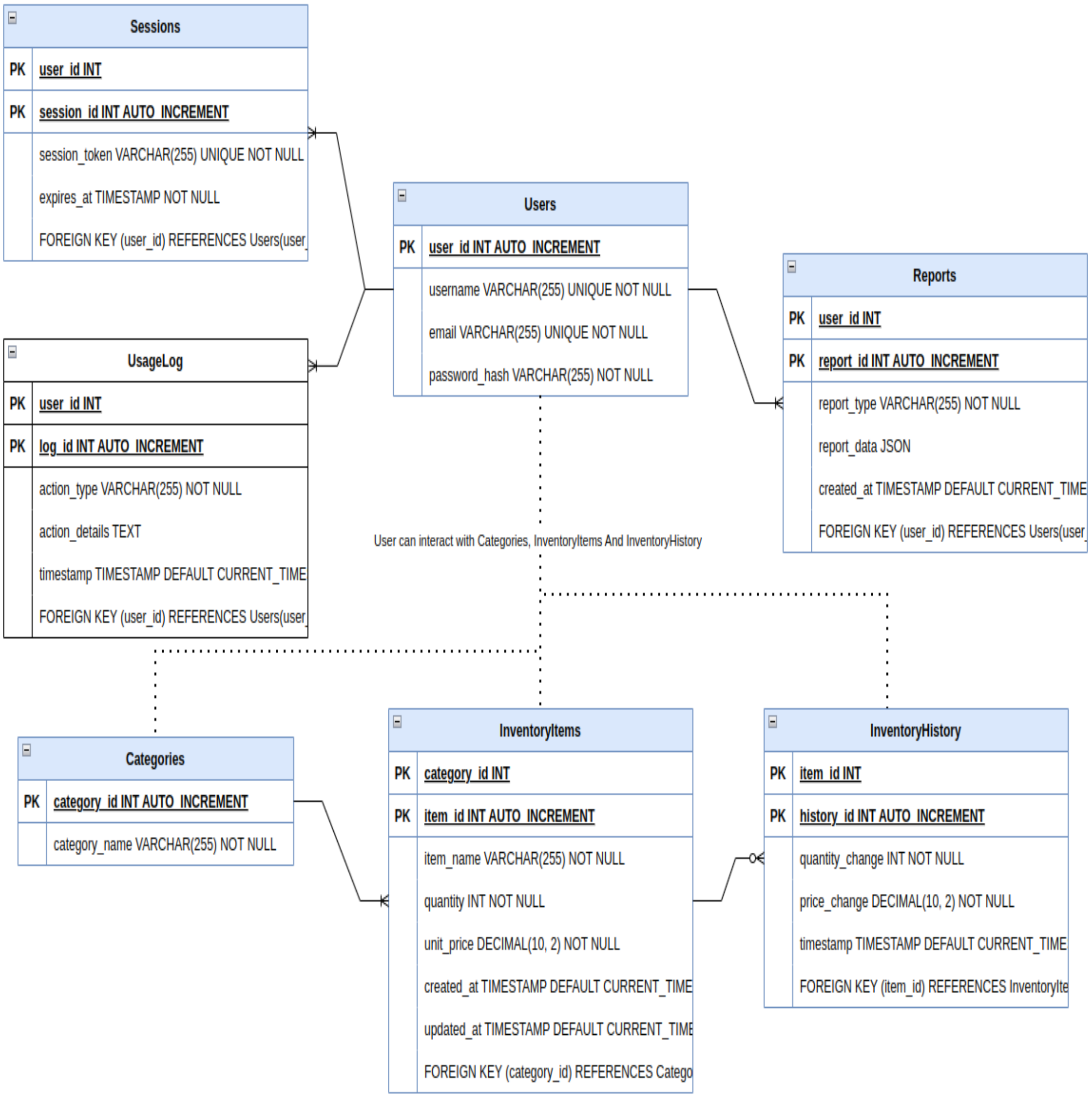
Architectural Design

The system will follow a three-tier architecture with the following components: The frontend made using React framework and written in JavaScript, the backend made using Express.js and Node.js also written in JavaScript and finally the Database written in MySQL. Axios library will be used to make API requests.

The Database Schema

Below is the database schema for the JIMS:-

Jeff's Inventory Management System Database Schema



Sessions

PK	user_id INT
PK	session_id INT AUTO_INCREMENT
	session_token VARCHAR(255) UNIQUE NOT NULL
	expires_at TIMESTAMP NOT NULL
	FOREIGN KEY (user_id) REFERENCES Users(user_

Users

PK	user_id INT AUTO_INCREMENT
	username VARCHAR(255) UNIQUE NOT NULL
	email VARCHAR(255) UNIQUE NOT NULL
	password_hash VARCHAR(255) NOT NULL

Reports

PK	user_id INT
PK	report_id INT AUTO_INCREMENT
	report_type VARCHAR(255) NOT NULL
	report_data JSON
	created_at TIMESTAMP DEFAULT CURRENT_TIME
	FOREIGN KEY (user_id) REFERENCES Users(user_

UsageLog

PK	user_id INT
PK	log_id INT AUTO_INCREMENT
	action_type VARCHAR(255) NOT NULL
	action_details TEXT
	timestamp TIMESTAMP DEFAULT CURRENT_TIME
	FOREIGN KEY (user_id) REFERENCES Users(user_

User can interact with Categories, InventoryItems And InventoryHistory

Categories

PK	category_id INT AUTO_INCREMENT
	category_name VARCHAR(255) NOT NULL

InventoryItems

PK	category_id INT
PK	item_id INT AUTO_INCREMENT
	item_name VARCHAR(255) NOT NULL
	quantity INT NOT NULL
	unit_price DECIMAL(10, 2) NOT NULL
	created_at TIMESTAMP DEFAULT CURRENT_TIME
	updated_at TIMESTAMP DEFAULT CURRENT_TIME
	FOREIGN KEY (category_id) REFERENCES Catego

InventoryHistory

PK	item_id INT
PK	history_id INT AUTO_INCREMENT
	quantity_change INT NOT NULL
	price_change DECIMAL(10, 2) NOT NULL
	timestamp TIMESTAMP DEFAULT CURRENT_TIME
	FOREIGN KEY (item_id) REFERENCES InventoryIte

Physical Design

For the physical design Node JS will be used to build the application, the React framework will be used for the frontend, the Express framework will be used for the backend and

XAMPP will be used to build the MySQL database. The backend will be used to connect, retrieve, update and delete data in the MySQL database.

The frontend will allow the user to interact with the application. Requests to the backend will be sent from the frontend using the Axios HTTP library. Html2canvas will be used to capture the report data to convert it to png while the Jspdf will be used to convert the png to PDF resulting in an export.. React-boostrap library will be used to style the buttons and give the website a sleek feel and lastly the react router-dom will be used to navigate links to page components.

The backend will use CORS mechanism will be used for secure cross-origin requests and data transfers between the browser and the server. It will also use the express-session library to handle user sessions.

Database Design

The database will include tables for users, inventory items, and inventory history, with appropriate relationships and data types. Data will be stored securely, and access controls will be implemented. The database will be MySQL. MySQL2 JavaScript library will be used to connect to the database.

Program Design

implement user authentication with secure password. The program will Implement APIs for adding, updating, and deleting inventory items, and maintain a history of changes. The system will also display inventory features upon request and have an option to export in PDF. The program code will be written in HTML, CSS and JavaScript.

Interface Design

For the interface a user-friendly menu with sections for authentication, inventory management, and reporting. It will also Include navigation and search functionalities.

There should also be creative intuitive forms for user registration and login. In addition, input validation for inventory data entry must be implemented.

The buttons and links will be styled using the react-bootstrap JavaScript library.

There should also be export features for CSV and PDF formats, with a date tag of when the export was made.

Finally, the system will be a blue themed colour which aligns with the system logo design below;-

JIMS
MANAGEMENT MADE SIMPLE

Security Back up Design

The website forms must Protect against common security vulnerabilities (e.g., SQL injection, XSS). The login should also enforce user authorization for inventory management operations. There should be an export inventory to prevent data loss.

Phase5: Implementation Phase

The Code

The code is divided into the frontend code, the backend code and the database code.

Frontend

For the frontend the code is divided into the following components:- Home.js, Login.js GenerateReport.js, Navigation.js, AddUser.js, Add Item.js, View Inventory.js, ManageUsers.js. The components are implemented using the App.js file.

globalStyles.css

```
body {
/*background-image: url('images/background.png');*/
background-color: #8ba1b4;
color: #333;
display: flex;
flex-direction: column;
align-items: center;
justify-content: center;
background-repeat: no-repeat;
background-size: cover;
}

.navbar {
background-color: #007bff;
color: #fff;
}

.highlighted-text span {
color: #007bff;
}
```

App.js

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom';
import { useAuth } from './components/AuthContext';
import 'bootstrap/dist/css/bootstrap.min.css';
import Home from './components/Home';
import Login from './components/Login';
import GenerateReport from './components/GenerateReport';
import Navigation from './components/Navigation';
import AddUser from './components/AddUser';
import AddItem from './components/AddItem';
import ViewInventory from './components/ViewInventoryList';
import ManageUsers from './components/ManageUsers';
import './globalStyles.css';
import './App.css';

const PrivateRoute = ({ element }) => {
const { isLoggedIn } = useAuth();

return isLoggedIn ? element : <Navigate to="/login" />;
};

function App() {
```

```
return (
<Router>
<div>
<h1>Jeff's Inventory Management System</h1>
<Routes>
<Route path="/" element={<Home />} />
<Route path="/login" element={<Login />} />
<Route
path="/AddUser"
element={<PrivateRoute element={<AddUser />} />}
/>
<Route
path="/AddItem"
element={<PrivateRoute element={<AddItem />} />}
/>
<Route
path="/ViewInventoryList"
element={<PrivateRoute element={<ViewInventory />} />}
/>
<Route
path="/GenerateReport"
element={<PrivateRoute element={<GenerateReport />} />}
/>
<Route
path="/ManageUsers"
element={<PrivateRoute element={<ManageUsers />} />}
/>

<Route
path="/Navigation"
element={<PrivateRoute element={<Navigation />} />}
/>
</Routes>
</div>
</Router>
);
}

export default App;
```

Login.js

```
import React, { useState } from 'react';
import axios from 'axios';
import { Link, useNavigate } from 'react-router-dom';
import { Container, Form, Button, Row, Col, Alert } from 'react-bootstrap'; // Import Bootstrap components
import { useAuth } from './AuthContext';
 function Login() {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [loginError, setLoginError] = useState('');
  const navigate = useNavigate();
  const { login } = useAuth();
  function handleLogin(event) {
    event.preventDefault();

    axios.post('http://localhost:5000/login', { email, password })
    .then(res => {
  console.log(res.data.success);
 if (res.data.success) {
        login();
        navigate('/Navigation');
      } else {
        setLoginError('Login failed. Please check your credentials.');
      }
    })
    .catch(err => {
      console.log(err);
      setLoginError('Login failed. Please check your credentials.');
    });
}
  return (
    <Container className="mt-5">
      <Row className="justify-content-center">
        <Col md={6}>
          <Form onSubmit={handleLogin}>
            <h2 className="mb-4 text-center">Login</h2>
            <Form.Group controlId="formEmail">
              <Form.Label>Email:</Form.Label>
              <Form.Control
                type="text"
                placeholder="Enter your email"
                value={email}
                onChange={e => setEmail(e.target.value)}
              />
            </Form.Group>
            <Form.Group controlId="formPassword">
              <Form.Label>Password:</Form.Label>
              <Form.Control
                type="password"
                placeholder="Enter your password"
                value={password}
        onChange={e => setPassword(e.target.value)}
       />
      </Form.Group>
      <Button variant="primary" type="submit" className="w-100">
       Login
      </Button>
      {loginError && <Alert variant="danger" className="mt-3">{loginError}</Alert>}
      <div className="mt-3 text-center">
       <Button variant="outline-primary" as={Link} to="/">
        Go Back
       </Button>
     </div>
     </Form>
    </Col>
   </Row>
  </Container>
 );
}

export default Login;
```

AddItem.js

```
import React, { useState } from 'react';
import { Modal, Button, Form } from 'react-bootstrap';
import axios from 'axios';
function AddItemDialog({ isOpen, onClose, onAdd }) {
const [itemName, setItemName] = useState('');
const [categoryName, setCategoryName] = useState('');
const [quantity, setQuantity] = useState('');
 const handleAddItem = () => {
```

```
// Validate input fields before making the request
if (!itemName || !categoryName || !quantity) {
// You may want to display an error message to the user
console.error('Please fill in all fields');
return;
}
// Make the request to add the item

axios .post('http://localhost:5000/addItem', {
itemName, categoryName,  quantity,
})
.then((res) => {

// Close the dialog and trigger a refresh of the items in the parent component
onClose();
onAdd();


})
.catch((err) => {
// Handle error (display error message, log, etc.)
console.error(err);
});
};

return (
<Modal show={isOpen} onHide={onClose}>
<Modal.Header closeButton>
<Modal.Title>Add Item</Modal.Title>
</Modal.Header>
<Modal.Body>
```

```
<Form>
<Form.Group controlId="itemName">
<Form.Label>Item Name</Form.Label>
<Form.Control  type="text" placeholder="Enter item name" value={itemName} onChange={(e) => setItemName(e.target.value)}
/>
</Form.Group>
<Form.Group controlId="categoryName">
<Form.Label>Category Name</Form.Label>
<Form.Control

type="text"

placeholder="Enter category name"

value={categoryName}

onChange={(e) => setCategoryName(e.target.value)}

/>

</Form.Group>

<Form.Group controlId="quantity">

<Form.Label>Quantity</Form.Label>

<Form.Control type="number" placeholder="Enter quantity" value={quantity} onChange={(e) => setQuantity(e.target.value)} />
</Form.Group>
</Form>
</Modal.Body>
```

```
<Modal.Footer>
<Button variant="primary" onClick={handleAddItem}>
```

Add Item

```
 </Button>
<Button variant="secondary" onClick={onClose}>
Cancel
</Button>
</Modal.Footer>
</Modal>
);
 }
export default AddItemDialog;
```

UpdateItem

```
import React, { useState } from 'react';
import axios from 'axios';
import { Modal, Button, Form } from 'react-bootstrap';
 function UpdateItemDialog({ isOpen, onClose, itemId, onUpdate }) {
const [quantity, setQuantity] = useState('');
const [updateItemError, setUpdateItemError] = useState('');
const handleUpdateItem = (event) => {
event.preventDefault();

axios .put(`http://localhost:5000/updateItem/${itemId}`, { quantity })
.then((res) => { console.log(res.data.success);
if (res.data.success) {
// Item updated successfully
onUpdate(); // Trigger a callback to refresh the item list or perform other actions
onClose(); // Close the dialog
} else {
```

```
setUpdateItemError('Failed to update item. Please check your input.');
}
})
.catch((err) => {
console.log(err);
setUpdateItemError('Failed to update item. Please check your input.');

});
};

return (
<Modal show={isOpen} onHide={onClose} centered>
<Modal.Header closeButton>
<Modal.Title>Update Item</Modal.Title>
</Modal.Header>
<Modal.Body>
<Form onSubmit={handleUpdateItem}>
<Form.Group controlId="formQuantity">
<Form.Label>New Quantity:</Form.Label>
<Form.Control type="number" name="quantity" value={quantity} onChange={(e) => setQuantity(e.target.value)} />
</Form.Group>
{updateItemError && (
<p className="text-danger">{updateItemError}</p>
)}
</Form>
</Modal.Body>
<Modal.Footer>
<Button variant="secondary" onClick={onClose}>
Cancel
</Button>
```

```
<Button variant="primary" type="submit" onClick={handleUpdateItem}>
Update Item
</Button>
</Modal.Footer>
</Modal>
);
}

export default UpdateItemDialog;
```

RemoveItem

```
import React, { useState } from 'react';
import axios from 'axios';
import { Modal, Button } from 'react-bootstrap';
function RemoveItemDialog({ isOpen, onClose, itemId, onRemove }) {
const [removeItemError, setRemoveItemError] = useState('');

const handleRemoveItem = () => {
axios .delete(`http://localhost:5000/removeItem/${itemId}`)
.then((res) => {
console.log(res.data.success);
if (res.data.success) {
// if  Item removed successfully
onRemove(); // Trigger a callback to refresh the item list or perform other actions
onClose(); // Close the dialog
} else {
```

```
setRemoveItemError('Failed to remove item. Please try again.');
}
})

.catch((err) => {
console.error(err);
setRemoveItemError('Failed to remove item. Please try again.');
});
};

return (
<Modal show={isOpen} onHide={onClose} centered>
<Modal.Header closeButton>
<Modal.Title>Remove Item</Modal.Title>
</Modal.Header>
<Modal.Body>
<p>Are you sure you want to remove this item?</p>
{removeItemError && <p className="text-danger">{removeItemError}</p>}
</Modal.Body>
<Modal.Footer>
<Button variant="secondary" onClick={onClose}>
No
</Button>

<Button variant="primary" onClick={handleRemoveItem}>
Yes
</Button>
</Modal.Footer>
</Modal>
);
}
```

```
export default RemoveItemDialog;
```

AddUser

```
import React, { useState } from 'react';
 import axios from 'axios';
 import { Link} from 'react-router-dom';
import { Container, Form, Button, Row, Col, Alert } from 'react-bootstrap';
 function AddUser() {
 const [name, setName] = useState('');
 const [email, setEmail] = useState('');
const [password, setPassword] = useState('');
 const [addUserError, setAddUserError] = useState('');
 const [userAdded, setUserAdded] = useState(false); // New state for success message

function handleAddUser(event) {
 event.preventDefault();
// Check if email already exists
axios.get(`http://localhost:5000/checkEmail/${email}`)
.then(emailCheckRes => {
if (emailCheckRes.data.exists) {
setAddUserError('Email already exists. Please choose a different email.');
} else {
// If email doesn't exist, proceed with adding the user
axios.post('http://localhost:5000/addUser', { name, email, password })
.then(res => {
if (res.data.success) {
setUserAdded(true);
setAddUserError('');
} else {
setAddUserError('Failed to add user. Please check your inputs.');
}
})
.catch(err => {
console.error(err);
setAddUserError('Failed to add user. Please check your inputs.');
});
}
})
.catch(emailCheckErr => {
console.error(emailCheckErr);
setAddUserError('Failed to check email existence. Please try again.');
});
}
return (
<Container className="mt-5">
<Row className="justify-content-center">
<Col md={6}>
<Form onSubmit={handleAddUser}>
<Form.Group controlId="formName">
<Form.Label>Name:</Form.Label>
<Form.Control
type="text"
placeholder="Enter the user's name"
value={name}
onChange={e => setName(e.target.value)}
/>

</Form.Group>
<Form.Group controlId="formEmail">
```

```
<Form.Label>Email:</Form.Label>
<Form.Control
type="text"
placeholder="Enter the user's email"
value={email}
onChange={e => setEmail(e.target.value)}
/>

</Form.Group>
<Form.Group controlId="formPassword">
<Form.Label>Password:</Form.Label>
<Form.Control type="password" placeholder="Enter the user's password" value={password} onChange={e => setPassword(e.target.value)} />
</Form.Group>

<Button variant="primary" type="submit" className="w-100"> Add User
</Button>
{addUserError && <Alert variant="danger" className="mt-3">{addUserError}</Alert>}
{userAdded && (
<Alert variant="success" className="mt-3">
User added successfully!
</Alert>
)}
<div className="mt-3 text-center">
<Button variant="outline-primary" as={Link} to="/Navigation"> Back
</Button>
</div>
</Form>
</Col>
</Row>
</Container>
```

```
);
}
 export default AddUser;
```

ViewItems

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { Button, Table, Pagination, Modal } from 'react-bootstrap';
import UpdateItemDialog from './UpdateItem';
import RemoveItemDialog from './RemoveItem';
import AddItemDialog from './AddItem';
import SearchBar from './SearchBar';
import { Link} from 'react-router-dom';

function OptionsDialog({ isOpen, onClose, onUpdate, onRemove }) {
return (
<Modal show={isOpen} onHide={onClose}>
<Modal.Header closeButton>
 <Modal.Title>Options</Modal.Title>
</Modal.Header>
<Modal.Body>
<Button variant="primary" onClick={onUpdate}>
Update Item
</Button>
<Button variant="danger" onClick={onRemove}>
Remove Item
</Button>
</Modal.Body>
<Modal.Footer>
<Button variant="secondary" onClick={onClose}>
Cancel
```

```
</Button>
</Modal.Footer>
</Modal>
);
}
function ViewItems() {
const [items, setItems] = useState([]);
const [currentPage, setCurrentPage] = useState(1);
const [itemsPerPage] = useState(10);
const [selectedItemId, setSelectedItemId] = useState(null);
const [isOptionsDialogOpen, setIsOptionsDialogOpen] = useState(false);
const [isUpdateDialogOpen, setIsUpdateDialogOpen] = useState(false);
const [isRemoveDialogOpen, setIsRemoveDialogOpen] = useState(false);
const [isAddDialogOpen, setIsAddDialogOpen] = useState(false);
const [searchResults, setSearchResults] = useState([]);
const formatDateTime = (isoDate) => {
const date = new Date(isoDate);
return date.toLocaleString();
};

useEffect(() => {

// Fetch items from the server

axios .get(`http://localhost:5000/viewItems`)
.then((res) => {
setItems(res.data.items);
setSearchResults(res.data.items);
})
.catch((err) => {
console.error(err);
});
}, []);
const indexOfLastItem = currentPage * itemsPerPage;
const indexOfFirstItem = indexOfLastItem - itemsPerPage;
const currentItems = searchResults.slice(indexOfFirstItem, indexOfLastItem);
const paginate = (pageNumber) => setCurrentPage(pageNumber);
 const handleItemClick = (itemId) => {
setSelectedItemId(itemId);
setIsOptionsDialogOpen(true);
};

const handleCloseOptionsDialog = () => {
setSelectedItemId(null);
setIsOptionsDialogOpen(false);
};
const handleOpenUpdateDialog = () => {
setIsOptionsDialogOpen(false);
setIsUpdateDialogOpen(true);
};
const handleCloseUpdateDialog = () => {
setIsUpdateDialogOpen(false);
};
const handleOpenRemoveDialog = () => {
setIsOptionsDialogOpen(false);
 setIsRemoveDialogOpen(true);
};
 const handleCloseRemoveDialog = () => {
setIsRemoveDialogOpen(false);
};
const handleOpenAddDialog = () => {
setIsAddDialogOpen(true);
```

```
};
const handleCloseAddDialog = () => {
setIsAddDialogOpen(false);
};
const handleRemoveItem = () => {
console.log(`Removing item with ID: ${selectedItemId}`);
handleCloseOptionsDialog();


};
const handleUpdate = () => {
axios
.get(`http://localhost:5000/viewItems`)
.then((res) => {
setItems(res.data.items);
setSearchResults(res.data.items);
})
.catch((err) => {
console.error(err);
});
};


const handleSearch = (searchTerm) => {
const filteredItems = items.filter(
(item) =>
 item.itemName.toLowerCase().includes(searchTerm.toLowerCase()) ||
item.categoryName.toLowerCase().includes(searchTerm.toLowerCase())
);
setSearchResults(filteredItems);
setCurrentPage(1);
};
return (
<div>
<br/>
{/* SearchBar */}
<SearchBar onSearch={handleSearch} />
<Table striped bordered hover>
<thead>
<tr>
<th>Item ID</th>
<th>Category Name</th>
<th>Item Name</th>
<th>Quantity</th>
<th>Created At</th>
<th>Updated At</th>
</tr>
</thead>
<tbody>
{currentItems.map((item) => (
<tr key={item.itemId} onClick={() => handleItemClick(item.itemId)}>
<td>{item.itemId}</td>
<td>{item.categoryName}</td>
<td>{item.itemName}</td>
<td>{item.quantity}</td>
<td>{formatDateTime(item.created_at)}</td>
<td>{formatDateTime(item.updated_at)}</td>
</tr>
))}
</tbody>
</Table>
<Pagination>
{Array.from({ length: Math.ceil(searchResults.length / itemsPerPage) }, (_, index) => (
<Pagination.Item key={index + 1} onClick={() => paginate(index + 1)}>
{index + 1}
</Pagination.Item>
))}
</Pagination>
{/* Options Dialog */}
<OptionsDialog
isOpen={isOptionsDialogOpen}
onClose={handleCloseOptionsDialog}
onUpdate={handleOpenUpdateDialog}
onRemove={handleOpenRemoveDialog}
/>

{/* Update Item Dialog */}
<UpdateItemDialog
isOpen={isUpdateDialogOpen}
onClose={handleCloseUpdateDialog}
itemId={selectedItemId}
onUpdate={handleUpdate}
/>

{/* Remove Item Dialog */}
<RemoveItemDialog
isOpen={isRemoveDialogOpen}
onClose={handleCloseRemoveDialog}
itemId={selectedItemId}
onRemove={handleRemoveItem}
/>

{/* Add Item Dialog */}
<AddItemDialog isOpen={isAddDialogOpen} onClose={handleCloseAddDialog}
onAdd={handleUpdate} />
```

```
{/* Add Item Button */}
<Button variant="success" onClick={handleOpenAddDialog}>
Add Item
</Button>
<div className="mt-3 text-center">
<Button variant="outline-primary" as={Link} to="/Navigation">
Back
</Button>
</div>
</div>
);
}

export default ViewItems;
```

ViewUsers.js

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { Table, Button, Modal } from 'react-bootstrap';
import { Link} from 'react-router-dom';

function ManageUsers() {
const [users, setUsers] = useState([]);
const [showModal, setShowModal] = useState(false);
const [selectedUserEmail, setSelectedUserEmail] = useState(null);
useEffect(() => {
 // Fetch the list of users when the component mounts
axios.get('http://localhost:5000/viewUsers')
.then(res => {
setUsers(res.data.users);
})
.catch(err => {
console.error('Error fetching users:', err);
});
}, []);

const handleRemoveUser = (userEmail) => {
setSelectedUserEmail(userEmail);
setShowModal(true);
};

const confirmRemoveUser = () => {
// Make API call to remove user by email
axios.delete(`http://localhost:5000/removeUser/${selectedUserEmail}`)
.then(res => {
// Refresh the user list after successful removal
axios.get('http://localhost:5000/viewUsers')
.then(res => {
setUsers(res.data.users);
})

.catch(err => {
console.error('Error fetching users:', err);
});
})

.catch(err => {
console.error('Error removing user:', err);
})
```

```
.finally(() => {
// Close the modal after removal attempt
setShowModal(false);
});
};

const handleCloseModal = () => {
setShowModal(false);
};

return (
<div className="mt-5">
<Table striped bordered hover>
<thead>
<tr>
<th>Name</th>
<th>Email</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
{users.map(user => (
<tr key={user.id}>
<td>{user.username}</td>
<td>{user.email}</td>
<td>
<Button variant="danger" onClick={() => handleRemoveUser(user.email)}>
Remove
</Button>
</td>
</tr>
))}
</tbody>
</Table>
{/* Modal for confirming user removal */}
<Modal show={showModal} onHide={handleCloseModal}>
<Modal.Header closeButton>
<Modal.Title>Confirm Removal</Modal.Title>
</Modal.Header>
<Modal.Body>
Are you sure you want to remove this user?
</Modal.Body>
<Modal.Footer>
<Button variant="secondary" onClick={handleCloseModal}>
Cancel
</Button>
<Button variant="danger" onClick={confirmRemoveUser}>
Remove
</Button>
</Modal.Footer>
</Modal>
<div className="mt-3 text-center">
<Button variant="outline-primary" as={Link} to="/Navigation">
Back
</Button>
</div>
</div>
);
}
export default ManageUsers;
```

Backend code

For the Backend the code is divided into the following routers : Login.js, AddItem.js, viewTable.js, UpdateItem.js, RemoveItem.js, AddUser.js, ViewItems.js, CheckEmail.js, ViewUsers.js and RemoveUsers.js.The routers are accessed using the server.js.

AddItem.js

```
const express = require('express');

const router = express.Router();

const db = require('../db/db');

router.post('/addItem', (req, res) => {

const { categoryName, itemName, quantity } = req.body;

 // Ensure required fields are provided

if (!categoryName || !itemName || !quantity) {

return res.status(400).json({ success: false, message: "Missing required fields" });

}


const sql = "INSERT INTO inventory (categoryName, itemName, quantity, created_at, updated_at) VALUES (?, ?, ?, CURRENT_TIMESTAMP, CURRENT_TIMESTAMP)";


const values = [categoryName, itemName, quantity];

db.query(sql, values, (err, data) => {

if (err) {

console.error("Database error:", err);

return res.status(500).json({ success: false, message: "Internal Server Error" });

}

return res.status(200).json({ success: true, message: "Item added successfully" });

});

});
```

```
module.exports = router;
```

Login.js

```
const express = require('express');

const router = express.Router();

 const db = require('../db/db');

router.post('/login', (req, res) => {

const sql = "SELECT * FROM users WHERE email = ? AND password = ?";

const values = [req.body.email, req.body.password];


db.query(sql, values, (err, data) => {

if (err) {

console.error("Database error:", err);

return res.status(500).json({ success: false, message: "Internal Server Error" });

}


if (data.length > 0) {

req.session.userId = data[0].userid;

req.session.userName = data[0].username;

// Send the username in the response

return res.status(200).json({

success: true,

message: "Login Successfully",

userName: req.session.userName,

});

} else {

return res.status(401).json({ success: false, message: "Invalid email or password" });

}

});

});
```

```
module.exports = router;
```

ViewTable.js

```
const express = require('express');
const router = express.Router();
const db = require('../db/db');

router.get('/viewTable', (req, res) => {

const sql = "SELECT `userdata` FROM `userdata` WHERE `userdataid` = 2";
db.query(sql, (err, data) => {
if (err) {
console.error("Database error:", err);
return res.status(500).json({ success: false, message: "Internal Server Error" });
}
// Assuming data is an array and you want the first item
const userData = data.length > 0 ? data[0] : null;
return res.status(200).json({ success: true, items: userData });
});
});
module.exports = router;
```

UpdateItem.js

```
const express = require('express');
const router = express.Router();
const db = require('../db/db');
router.put('/updateItem/:itemId', (req, res) => {
const itemId = req.params.itemId;
const { quantity } = req.body;
// Ensure required fields are provided
if (!quantity) {
```

```
return res.status(400).json({ success: false, message: "Missing required fields" });
}
const sql = "UPDATE inventory SET quantity = ? WHERE itemId = ?";
const values = [quantity, itemId];
db.query(sql, values, (err, data) => {
if (err) {
console.error("Database error:", err);
return res.status(500).json({ success: false, message: "Internal Server Error" });
}




// Check if any rows were affected
if (data.affectedRows === 0) {
return res.status(404).json({ success: false, message: "Item not found" });
}
return res.status(200).json({ success: true, message: "Item updated successfully" });
});
});
module.exports = router;
```

RemoveItem .js

```
const express = require('express');
const router = express.Router();
const db = require('../db/db');

router.delete('/removeItem/:itemId', (req, res) => {
const itemId = req.params.itemId;
const sql = "DELETE FROM inventory WHERE itemId = ?";
const values = [itemId];
db.query(sql, values, (err, data) => {
if (err) {
console.error("Database error:", err);
return res.status(500).json({ success: false, message: "Internal Server Error" });
}
// Check if any rows were affected
if (data.affectedRows === 0) {
return res.status(404).json({ success: false, message: "Item not found" });
}



return res.status(200).json({ success: true, message: "Item removed successfully" });
});
});


module.exports = router;
```

AddUser.js

```
const express = require('express');
const router = express.Router();
const db = require('../db/db');

router.post('/addUser', (req, res) => {
const { name, email, password } = req.body;
// Ensure required fields are provided
if (!name || !email || !password) {
```

```
return res.status(400).json({ success: false, message: "Missing required fields" });
}

const sql = "INSERT INTO users (username, email, password) VALUES (?, ?, ?)";
const values = [name, email, password];
db.query(sql, values, (err, data) => {
if (err) {
console.error("Database error:", err);
return res.status(500).json({ success: false, message: "Internal Server Error" });
}
return res.status(200).json({ success: true, message: "User added successfully" });
});
});

module.exports = router;
```

ViewItems.js

```
const express = require('express');
const router = express.Router();
const db = require('../db/db');
router.get('/viewItems', (req, res) => {
const sql = "SELECT * FROM inventory";
db.query(sql, (err, data) => {
if (err) {
console.error("Database error:", err);
return res.status(500).json({ success: false, message: "Internal Server Error" });
}
return res.status(200).json({ success: true, items: data });
});
});
```

```
module.exports = router;
```

CheckEmail.js

```
const express = require('express');
const router = express.Router();
const db = require('../db/db');
router.get('/checkEmail/:email', (req, res) => {
const emailToCheck = req.params.email;

// Ensure email is provided
if (!emailToCheck) {
return res.status(400).json({ exists: false, message: "Email not provided" });
}

const sql = "SELECT COUNT(*) AS count FROM users WHERE email = ?";
 const values = [emailToCheck];
db.query(sql, values, (err, data) => {
if (err) {
console.error("Database error:", err);
return res.status(500).json({ exists: false, message: "Internal Server Error" });
}
const emailExists = data[0].count > 0;
return res.status(200).json({ exists: emailExists, message: "Email check successful" });
});
});

module.exports = router;
```

ViewUsers.js

```
// routes/viewUsers.js
```

```
const express = require('express');
const router = express.Router();
const db = require('../db/db');

router.get('/viewUsers', (req, res) => {
const sql = "SELECT username, email FROM users"; // Adjust the SQL query based on your actual database schema
db.query(sql, (err, data) => {
if (err) {
console.error("Database error:", err);
return res.status(500).json({ error: "Internal Server Error" });
}
return res.status(200).json({ users: data });
});
});
module.exports = router;
```

RemoveUsers.js

```
// routes/removeUser.js
const express = require('express');
const router = express.Router();
const db = require('../db/db');

router.delete('/removeUser/:email', (req, res) => {
const userEmail = req.params.email;
// Ensure userEmail is provided
if (!userEmail) {
return res.status(400).json({ error: "User email not provided" });
}
const sql = "DELETE FROM users WHERE email = ?";
const values = [userEmail];

db.query(sql, values, (err, data) => {
if (err) {
console.error("Database error:", err);
return res.status(500).json({ error: "Internal Server Error" });
}

// Check if any rows were affected (user with given email exists)
if (data.affectedRows === 0) {
return res.status(404).json({ error: "User not found" });
}
return res.status(200).json({ success: true, message: "User removed successfully" });
});
});




module.exports = router;
```

Database

The database connection is made using the following API endpoints Db.js and Session.js

DB.js

```
const mysql = require('mysql2');
const db = mysql.createConnection({
host: "localhost",
user: "jeff",
password: "jeff",
database: "JIMS",
});

module.exports = db;
```

Session.js

```
const session = require('express-session');
const MySQLStore = require('express-mysql-session')(session);
const db = require('./db');
const sessionStore = new MySQLStore({
createDatabaseTable: true,
schema: {
tableName: 'sessions',
columnNames: {
session_id: 'session_id',
expires: 'expires',
data: 'data',
,  },
expiration: 20 * 60 * 1000, // 20 minutes
}, db.promise());
const sessionConfig = {
secret: 'sdfsdfsdfs',
resave: false,
saveUninitialized: false,
store: sessionStore,
cookie: {
```

```
maxAge: 20 * 60 * 1000, // 20 minutes

},

};

module.exports = { sessionConfig, sessionStore };
```

Server.js

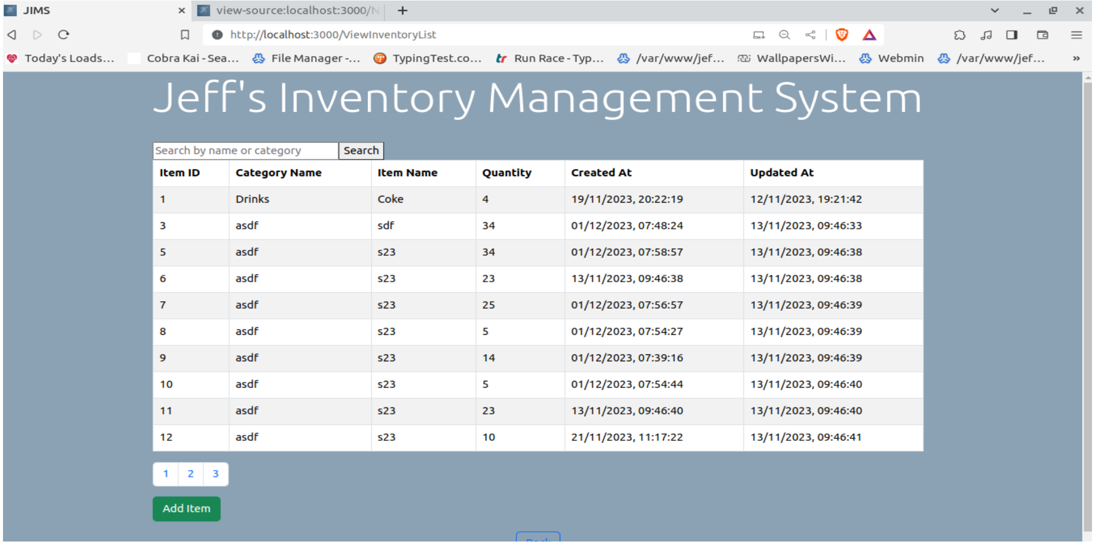
```
const express = require('express');

const cors = require('cors');

const session = require('express-session');

const db = require('./db/db');

const { sessionConfig, sessionStore } = require('./db/session'); // Corrected import path

const app = express();

const port = 5000;

// Use session middleware

app.use(session(sessionConfig));

app.use(cors());

app.use(express.json());

// Import your routers

const loginRouter = require('./routes/login.js');

const itemsRouter = require('./routes/addItem.js');

const viewTableRouter = require('./routes/viewTable.js');

const updateItemRouter = require('./routes/updateItem.js');

const removeItemRouter = require('./routes/removeItem.js');

const addUserRouter = require('./routes/addUser.js');


const viewItems = require('./routes/viewItems.js');

const checkEmail = require('./routes/checkEmail.js');

const viewUsersRouter = require('./routes/viewUsers.js');

const removeUserRouter = require('./routes/removeUser.js');


// Use your routers
```

```
app.delete('/removeItem/:itemId', removeItemRouter);
app.post('/login', loginRouter);
app.get('/viewItems', viewItems);
app.post('/addItem', itemsRouter);
app.get('/viewTable', viewTableRouter);
app.put('/updateItem/:itemId', updateItemRouter);
app.post('/addUser', addUserRouter);
app.get('/checkEmail/:email', checkEmail);
app.get('/viewUsers', viewUsersRouter);
app.delete('/removeUser/:email', removeUserRouter);
app.get('/', (req, res) => {
res.send('Hello from the backend!');
});
// Error handling for db.promise() connection
db.promise()
.connect()
.then(() => {
console.log('Connected to the database');
app.listen(port, () => {
console.log(`Server is running on port ${port}`);
});
})
.catch((err) => {
console.error('Error connecting to the database:', err);
});
```

Testing

Below are screeshots of tests that were conducted on the final project.

The Inventory List Page

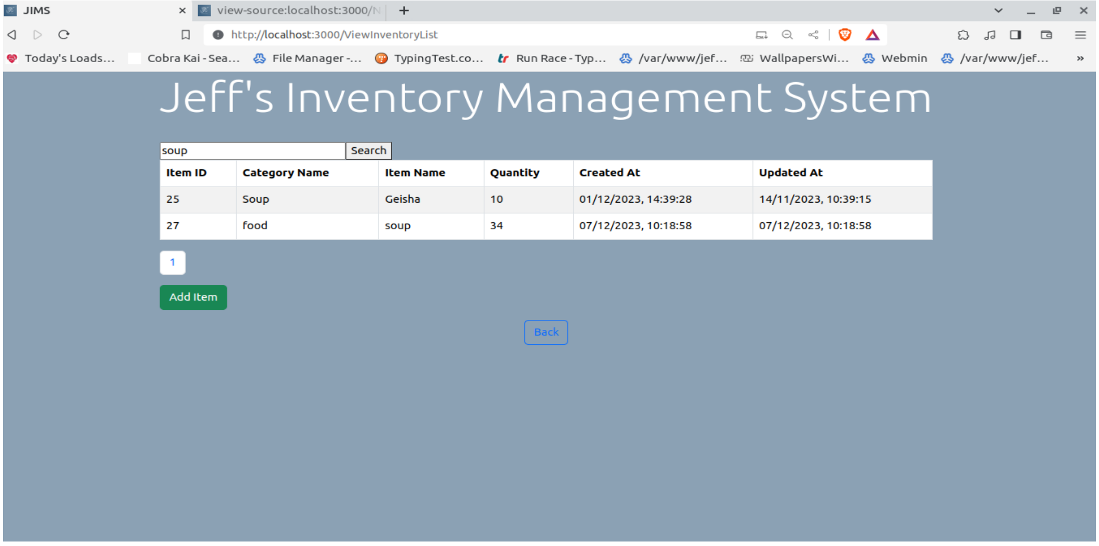


Jeff's Inventory Management System

Search by name or category Search

Item ID	Category Name	Item Name	Quantity	Created At	Updated At
1	Drinks	Coke	4	19/11/2023, 20:22:19	12/11/2023, 19:21:42
3	asdf	sdf	34	01/12/2023, 07:48:24	13/11/2023, 09:46:33
5	asdf	s23	34	01/12/2023, 07:58:57	13/11/2023, 09:46:38
6	asdf	s23	23	13/11/2023, 09:46:38	13/11/2023, 09:46:38
7	asdf	s23	25	01/12/2023, 07:56:57	13/11/2023, 09:46:39
8	asdf	s23	5	01/12/2023, 07:54:27	13/11/2023, 09:46:39
9	asdf	s23	14	01/12/2023, 07:39:16	13/11/2023, 09:46:39
10	asdf	s23	5	01/12/2023, 07:54:44	13/11/2023, 09:46:40
11	asdf	s23	23	13/11/2023, 09:46:40	13/11/2023, 09:46:40
12	asdf	s23	10	21/11/2023, 11:17:22	13/11/2023, 09:46:41

The Inventory Page After Search

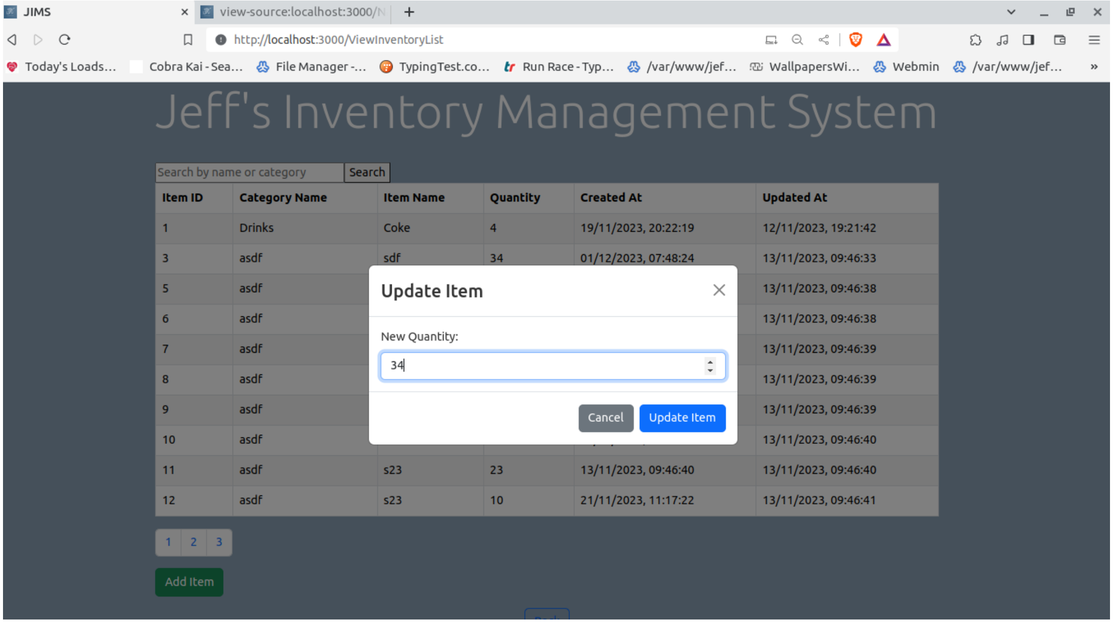


Jeff's Inventory Management System

soup Search

Item ID	Category Name	Item Name	Quantity	Created At	Updated At
25	Soup	Geisha	10	01/12/2023, 14:39:28	14/11/2023, 10:39:15
27	food	soup	34	07/12/2023, 10:18:58	07/12/2023, 10:18:58

Update Item



Jeff's Inventory Management System

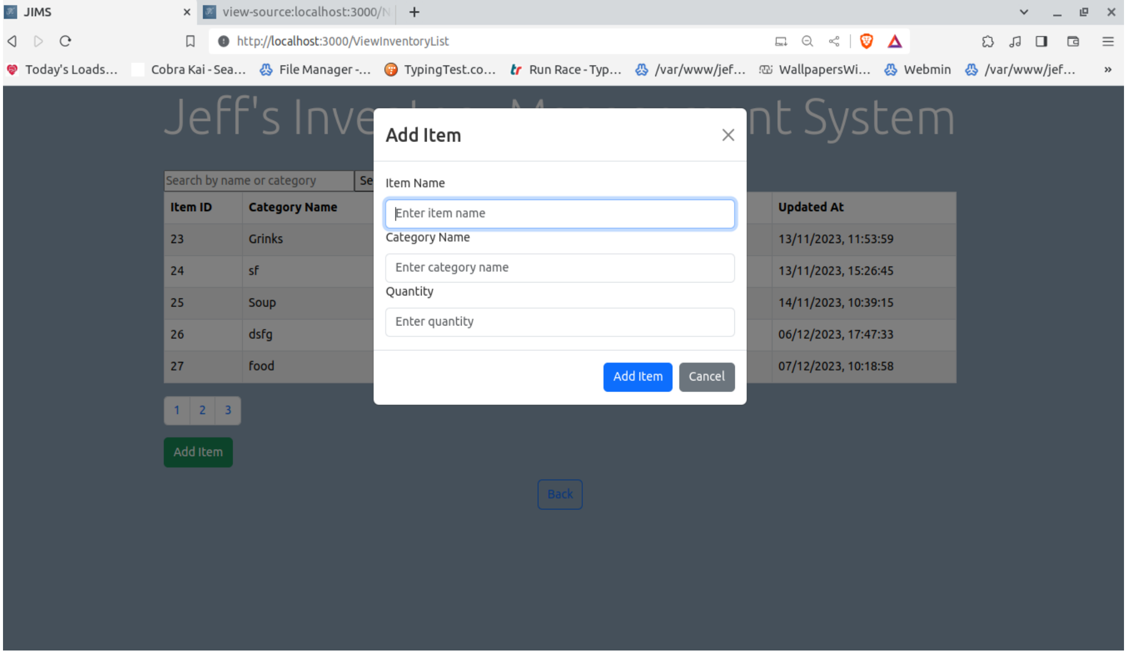
Search by name or category | Search

Item ID	Category Name	Item Name	Quantity	Created At	Updated At
1	Drinks	Coke	4	19/11/2023, 20:22:19	12/11/2023, 19:21:42
3	asdf	sdf	34	01/12/2023, 07:48:24	13/11/2023, 09:46:33
5	asdf				13/11/2023, 09:46:38
6	asdf				13/11/2023, 09:46:38
7	asdf				13/11/2023, 09:46:39
8	asdf				13/11/2023, 09:46:39
9	asdf				13/11/2023, 09:46:39
10	asdf				13/11/2023, 09:46:40
11	asdf	s23	23	13/11/2023, 09:46:40	13/11/2023, 09:46:40
12	asdf	s23	10	21/11/2023, 11:17:22	13/11/2023, 09:46:41

Update Item

New Quantity: 34

Add Item



Jeff's Inve nt System

Search by name or category | Se

Item ID	Category Name	Updated At
23	Grinks	13/11/2023, 11:53:59
24	sf	13/11/2023, 15:26:45
25	Soup	14/11/2023, 10:39:15
26	dsfg	06/12/2023, 17:47:33
27	food	07/12/2023, 10:18:58

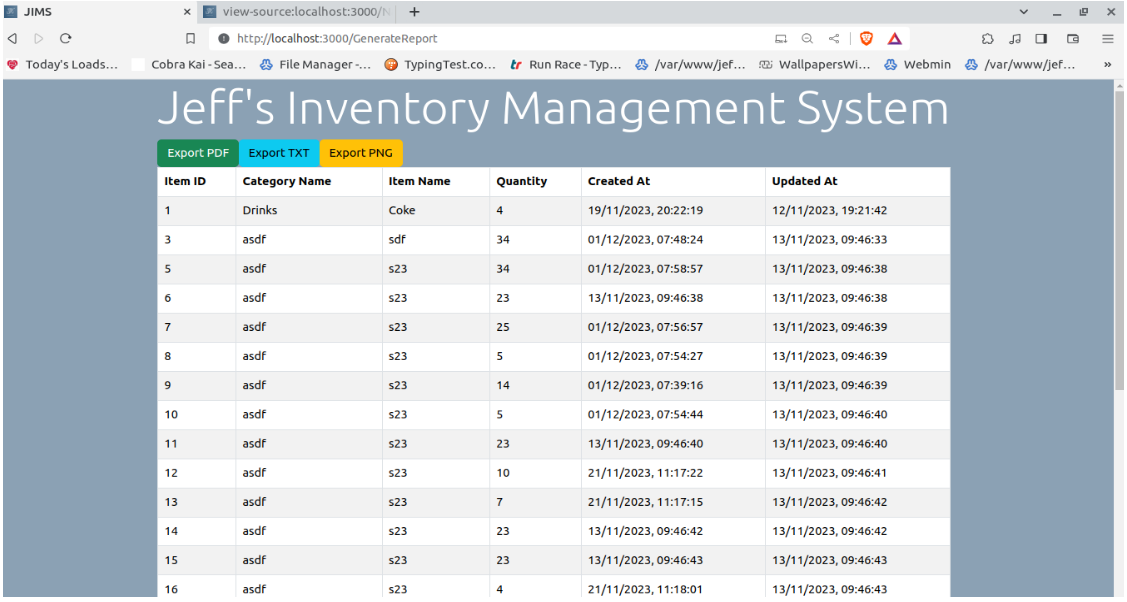
Add Item

Item Name: Enter item name

Category Name: Enter category name

Quantity: Enter quantity

The generate report page



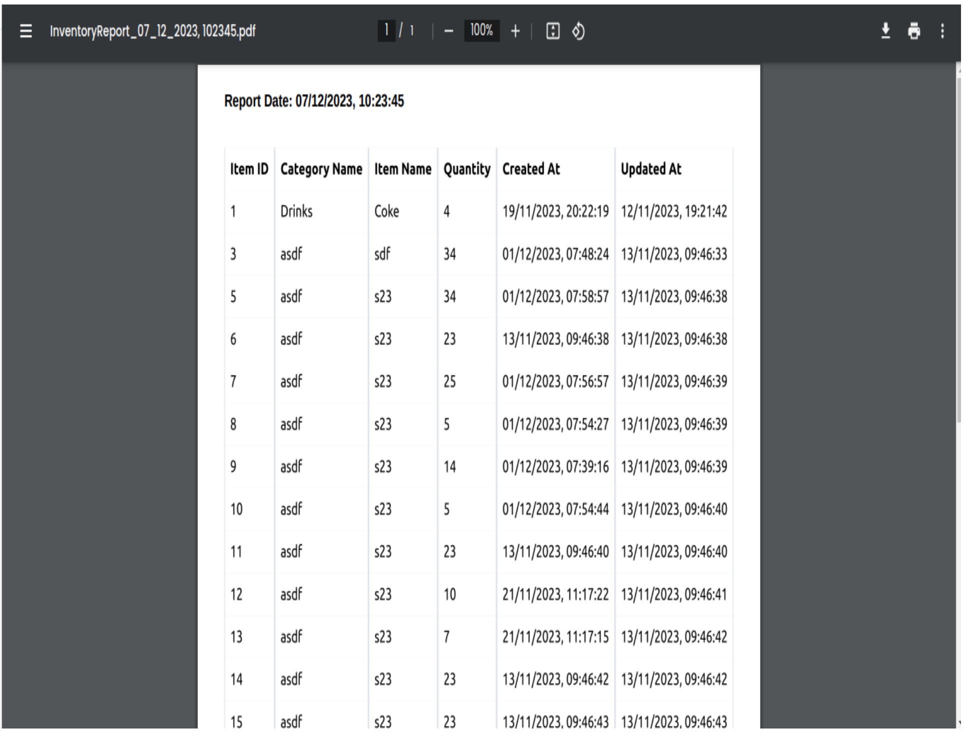
Jeff's Inventory Management System

Item ID	Category Name	Item Name	Quantity	Created At	Updated At
1	Drinks	Coke	4	19/11/2023, 20:22:19	12/11/2023, 19:21:42
3	asdf	sdf	34	01/12/2023, 07:48:24	13/11/2023, 09:46:33
5	asdf	s23	34	01/12/2023, 07:58:57	13/11/2023, 09:46:38
6	asdf	s23	23	13/11/2023, 09:46:38	13/11/2023, 09:46:38
7	asdf	s23	25	01/12/2023, 07:56:57	13/11/2023, 09:46:39
8	asdf	s23	5	01/12/2023, 07:54:27	13/11/2023, 09:46:39
9	asdf	s23	14	01/12/2023, 07:39:16	13/11/2023, 09:46:39
10	asdf	s23	5	01/12/2023, 07:54:44	13/11/2023, 09:46:40
11	asdf	s23	23	13/11/2023, 09:46:40	13/11/2023, 09:46:40
12	asdf	s23	10	21/11/2023, 11:17:22	13/11/2023, 09:46:41
13	asdf	s23	7	21/11/2023, 11:17:15	13/11/2023, 09:46:42
14	asdf	s23	23	13/11/2023, 09:46:42	13/11/2023, 09:46:42
15	asdf	s23	23	13/11/2023, 09:46:43	13/11/2023, 09:46:43
16	asdf	s23	4	21/11/2023, 11:18:01	13/11/2023, 09:46:43

Test Case

To test the system a typical task was used. The task was to login and make an inventory list. Then generate an inventory report in pdf and txt. The system was able to complete the task flawlessly. The test results are shown below:- A screenshot of the report in pdf format and a screenshot of the report in text format.

<u>Screenshot of the report in pdf format</u>

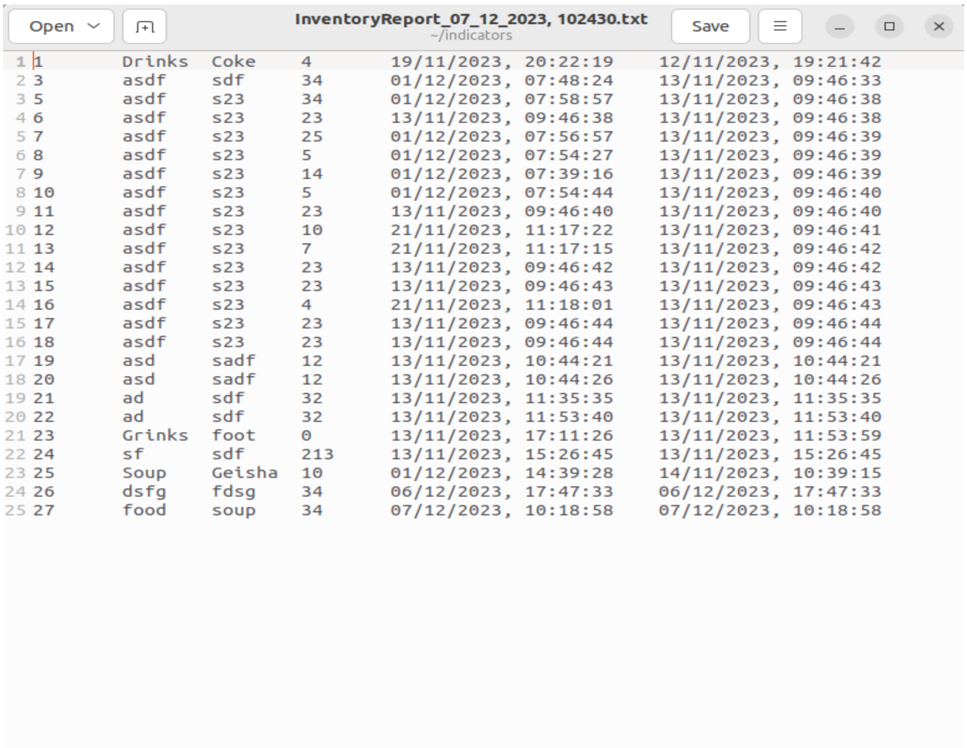


Report Date: 07/12/2023, 10:23:45

Item ID	Category Name	Item Name	Quantity	Created At	Updated At
1	Drinks	Coke	4	19/11/2023, 20:22:19	12/11/2023, 19:21:42
3	asdf	sdf	34	01/12/2023, 07:48:24	13/11/2023, 09:46:33
5	asdf	s23	34	01/12/2023, 07:58:57	13/11/2023, 09:46:38
6	asdf	s23	23	13/11/2023, 09:46:38	13/11/2023, 09:46:38
7	asdf	s23	25	01/12/2023, 07:56:57	13/11/2023, 09:46:39
8	asdf	s23	5	01/12/2023, 07:54:27	13/11/2023, 09:46:39
9	asdf	s23	14	01/12/2023, 07:39:16	13/11/2023, 09:46:39
10	asdf	s23	5	01/12/2023, 07:54:44	13/11/2023, 09:46:40
11	asdf	s23	23	13/11/2023, 09:46:40	13/11/2023, 09:46:40
12	asdf	s23	10	21/11/2023, 11:17:22	13/11/2023, 09:46:41
13	asdf	s23	7	21/11/2023, 11:17:15	13/11/2023, 09:46:42
14	asdf	s23	23	13/11/2023, 09:46:42	13/11/2023, 09:46:42
15	asdf	s23	23	13/11/2023, 09:46:43	13/11/2023, 09:46:43

X

Screenshot of the report in text format



```
1       Drinks  Coke    4       19/11/2023, 20:22:19    12/11/2023, 19:21:42
3       asdf    sdf     34      01/12/2023, 07:48:24    13/11/2023, 09:46:33
5       asdf    s23     34      01/12/2023, 07:58:57    13/11/2023, 09:46:38
6       asdf    s23     23      13/11/2023, 09:46:38    13/11/2023, 09:46:38
7       asdf    s23     25      01/12/2023, 07:56:57    13/11/2023, 09:46:39
8       asdf    s23     5       01/12/2023, 07:54:27    13/11/2023, 09:46:39
9       asdf    s23     14      01/12/2023, 07:39:16    13/11/2023, 09:46:39
10      asdf    s23     5       01/12/2023, 07:54:44    13/11/2023, 09:46:40
11      asdf    s23     23      13/11/2023, 09:46:40    13/11/2023, 09:46:40
12      asdf    s23     10      21/11/2023, 11:17:22    13/11/2023, 09:46:41
13      asdf    s23     7       21/11/2023, 11:17:15    13/11/2023, 09:46:42
14      asdf    s23     23      13/11/2023, 09:46:42    13/11/2023, 09:46:42
15      asdf    s23     23      13/11/2023, 09:46:43    13/11/2023, 09:46:43
16      asdf    s23     4       21/11/2023, 11:18:01    13/11/2023, 09:46:43
17      asdf    s23     23      13/11/2023, 09:46:44    13/11/2023, 09:46:44
18      asdf    s23     23      13/11/2023, 09:46:44    13/11/2023, 09:46:44
19      asd     sadf    12      13/11/2023, 10:44:21    13/11/2023, 10:44:21
20      asd     sadf    12      13/11/2023, 10:44:26    13/11/2023, 10:44:26
21      ad      sdf     32      13/11/2023, 11:35:35    13/11/2023, 11:35:35
22      ad      sdf     32      13/11/2023, 11:53:40    13/11/2023, 11:53:40
23      Grinks  foot    0       13/11/2023, 17:11:26    13/11/2023, 11:53:59
24      sf      sdf     213     13/11/2023, 15:26:45    13/11/2023, 15:26:45
25      Soup    Geisha  10      01/12/2023, 14:39:28    14/11/2023, 10:39:15
26      dsfg    fdsg    34      06/12/2023, 17:47:33    06/12/2023, 17:47:33
27      food    soup    34      07/12/2023, 10:18:58    07/12/2023, 10:18:58
```

Software Installation

To install the software Node.js and npm is required on your machine. You can download them from https://nodejs.org/. In the frontend end folder, open terminal and use the command “npm install”, then do the same for the backend folder. Run npm start for both folders in separate terminals and open your browser and navigate to http://localhost:3000 to view the app. Setup the JIMS database and implement the users and inventoryList table, then setup the necessary username and password in the db.js file.

Project Conclusion

The goal of this project was to create an inventory management system that could replace the traditional pen-and-paper method. I went above and beyond, ending up with a system that's not only effective but also fast and dependable.

Despite some challenges I faced along the way, I managed to overcome them easily. The result is a system that can be used in various aspects of life where a group of people needs to handle inventory.

The system is versatile and can be applied in different industries like retail, manufacturing, or healthcare, providing a solution for collaborative inventory management. It's designed to give real-time updates, making decision-making quicker, and it's reliable, ensuring accurate tracking of inventory levels.

This project delivered more than expected, providing a simple and efficient solution for anyone looking to transition from manual inventory tracking to a more streamlined and accurate system.

Cost Estimation of the Project

The project used free software from start to finish. The only cost was time.

Reports

Users who used the system where satisfied with its ingenuity.

Future scope and further enhancement of the Project

In the future I plan to add a monitoring system, that monitors the time users spend accessing the inventory. I would also want to add a color and theme changing system that would assist the visually impared. More advanced tools like automated QR scanning systems would also help the system be more efficient.

Bibliography

Information Technology Project Management, Kathy Schwalbe, 4th Edition, Thomson, ISBN 0-619-21528-3

Smith, J. (2005). The Art of Project Management. New York: ABC Publishers.

Johnson, A. (2010). Project Planning in Contemporary Organizations. Journal of Project Management, 15(2), 45-60. https://doi.org/xxxx

Brown, M. (2018). Project Management Best Practices. Retrieved from https://www.example.com/best-practices (Accessed on June 1, 2023)