s

CSTS

innovate · integrate · inspire

Compsci Technology solutions

Internship Report on

Invoice Generator

Prepared by

Chandu Chitteti

Intern ID :240402

Under the mentorship of
Mr. Naveen Kumar
&
Dr. Kumar Raja

DECLARATION

I, Chandu Chitteti, hereby declare that this project work titled “Invoice generator” is a genuine work carried out by me and has not been submitted to any other course or University for the award of any degree by me.

CH-Chandu

Signature

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to my institute, NBKR institute of Science And Technology Vidhyanagar, and its Placement Cell for providing me with the opportunity to undertake this internship.

I am also extremely grateful to Mr. Naveen Kumar and Dr. Kumar Raja from Compsci Technology solutions for offering me this internship and providing continuous guidance and support throughout the duration. Their insights and mentorship have been instrumental in making this internship a truly enriching experience.

Additionally, I would like to acknowledge my co-interns for their support and collaboration throughout this internship. Their assistance and camaraderie have made this experience both enjoyable and educational.

Thank you to everyone who has contributed to my learning and growth during this period.

Abstract

This project focuses on the development of an advanced invoice generation system built using React, aimed at delivering a seamless and user-friendly experience. The application starts with a landing page featuring login and signup capabilities, laying the groundwork for a robust and secure system. Following successful authentication, users are directed to a dashboard where the organization's inventory data is loaded. This inventory data is stored persistently, ensuring that it remains accessible even after page reloads. Future enhancements will include connecting this data to a database, allowing for real-time updates and efficient inventory management.

At the heart of the system is the invoice generation functionality. Users can select items from the inventory to add to the invoice, with validation mechanisms in place to ensure that the quantity selected does not exceed the available stock. This feature maintains data accuracy and prevents discrepancies in inventory records. Once the items and customer details are inputted, users can preview the invoice. This preview provides a detailed breakdown of items, prices, discounts, and the total amount. The application offers the flexibility to print the invoice in either A4 or thermal format, catering to different user requirements and printing setups.

In addition to its current capabilities, this project is designed to scale with future improvements. Planned updates include the integration of a backend system to handle data storage and retrieval, further enhancing the application's performance and reliability. This project has been instrumental in refining my skills as a full-stack developer, particularly in the areas of React, front-end development, state management, and data validation. It serves as a strong foundation for creating more complex and feature-rich web applications in the future.

TABLE OF CONTENTS :

Table of contents

1. Introduction

The Invoice Generation Project is a web application designed to simplify and automate the process of creating invoices for products within an organization. Developed using React, the project focuses on delivering a user-friendly interface that allows users to efficiently manage inventory and generate accurate invoices. The primary motivation behind this project is to address the challenges faced by businesses in manually tracking inventory and handling invoicing tasks, which can often lead to errors and inefficiencies. By providing an automated solution, the project aims to enhance the accuracy, speed, and reliability of invoice generation. The system also allows users to preview and print invoices in different formats, catering to diverse business needs. Future plans include expanding the system to support backend integration, enabling real-time updates and scalability.

2. Statement of the Problem

In many organizations, managing inventory and generating invoices are time-consuming tasks that require careful attention to detail. The manual process of tracking stock levels and creating invoices often leads to mistakes, such as over-selling products that are no longer in stock or miscalculating prices and discounts. These errors can result in financial losses, customer dissatisfaction, and reduced operational efficiency. Additionally, the absence of a streamlined system for validating inventory levels during the invoicing process can cause discrepancies between what is sold and what is available. This project addresses these issues by providing an automated system that integrates inventory management with invoice generation, ensuring accuracy and reducing the likelihood of errors.

3.Objectives

The objectives of this project are as follows:

- **Develop a Responsive UI:** Create a React-based user interface that is intuitive, responsive, and accessible across different devices. The UI should support core functions like login, signup, and navigation through the system's features.
- **Implement Inventory Validation:** Ensure that the system validates the quantity of products being added to invoices against the available inventory. This prevents over-selling and maintains accurate stock levels.
- **Enable Invoice Preview and Printing:** Provide users with the ability to preview invoices before finalizing them, including details like item descriptions, prices, discounts, and totals. Users should also be able to print the invoices in A4 and thermal formats, depending on their needs.
- **Build a Scalable Foundation:** Design the application with scalability in mind, allowing for future integration with a backend system and additional features like payment processing, multi-user roles, and detailed reporting.

4. Scope

The current scope of the project focuses on the front-end development of the invoice generation system. Key features include:

- **Landing Page:** A landing page with login and signup functionality, providing secure access to the system.
- **Dashboard:** A dashboard that loads the organization’s inventory data, with persistent data management to ensure that information remains available between sessions.
- **Invoice Generation:** A feature that allows users to create invoices by selecting items from the inventory, entering customer details, and applying discounts or taxes as needed.
- **Quantity Validation:** Built-in validation to ensure that the quantity of items added to the invoice does not exceed available stock.
- **Invoice Preview and Printing:** A preview function that allows users to review invoices before printing them in either A4 or thermal formats.

Future scope includes integrating a backend system to handle data storage and retrieval, real-time updates to inventory data, and the addition of more advanced features like analytics and user roles. The system is also designed to be modular, enabling further customization and enhancement as business needs evolve.

5. Applications

The invoice generation system can be applied across various industries and business sectors that require efficient inventory management and billing solutions. Potential applications include:

- **Retail:** Stores can use the system to manage product sales, track stock levels, and generate customer invoices on the spot.
- **Wholesale:** Wholesale distributors can handle large quantities of products, ensuring that inventory levels are accurately reflected in invoices.
- **Manufacturing:** Manufacturing firms can track raw materials and finished goods, generating invoices for both suppliers and customers.
- **E-commerce:** Online retailers can integrate the system into their platforms to automate billing and inventory management, reducing manual effort and errors.
- **Service Industry:** Businesses providing services alongside products can use the system to generate detailed invoices that include both labor and materials.

6.Limitations

While the current system offers a range of useful features, there are several limitations to be addressed:

1. **No Backend Integration:** The system currently lacks backend integration, which means that data is not stored persistently beyond the session. This limits the application's ability to manage inventory updates and store invoice records over time.

2. **Limited Feature Set:** The system does not yet support advanced features such as payment processing, detailed analytics, or user role management, which could be critical for larger organizations with complex needs.
3. **Static Inventory Data:** As of now, inventory data is manually updated and does not reflect real-time changes, making it less suitable for dynamic environments where stock levels fluctuate frequently.
4. **Customization Requirements:** The system is designed for general use cases and may require customization for specific business needs or industries that have unique billing processes or regulatory requirements.
5. **Scalability:** While the system is designed with scalability in mind, it currently lacks the infrastructure to handle large-scale operations, multi-location inventory management, or high transaction volumes without further development.

Software Requirements Specification for Invoice Generation Project

---

1. Introduction

1.1 Problem Statement

Manual invoice generation is a time-consuming and error-prone process, particularly in businesses dealing with large volumes of transactions. The traditional method often leads to inaccuracies in billing, delays in processing, and inefficiencies in record-keeping. These issues can hinder the overall business operations and customer satisfaction. As businesses expand, the need for a scalable, automated invoice generation system becomes essential.

The solution is to develop an automated invoice generation system that simplifies the process, reduces human errors, and enhances efficiency. This system will enable businesses to generate invoices quickly and accurately, thereby improving overall productivity and customer experience.

1.2 Purpose

The purpose of the Invoice Generation system is to provide a streamlined platform for creating and managing invoices. The system automates the generation of invoices, ensuring accuracy and saving time, which allows businesses to focus on their core operations.

1.3 Intended Audience

The Invoice Generation system is intended for:

- Small and Medium Enterprises (SMEs): To manage invoices for sales, services, and products efficiently.
- Freelancers and Independent Contractors: To generate invoices for clients easily.
- Accountants and Financial Managers: To streamline the invoicing process and maintain accurate financial records.

1.4 Project Scope

The Invoice Generation project aims to deliver a web-based application with the following features:

- Invoice Creation: Generate invoices for products or services, including details like item descriptions, quantities, and prices.
- Customer Management: Manage customer details and retrieve them during invoice creation.
- Invoice Preview and Printing: Preview invoices before printing, with options for A4 and thermal formats.
- User Authentication: Ensure secure access to the system through user login.

1.5 References

- Flask Documentation
- React Documentation
- Axios Library Documentation
- Python 3.12.3 Documentation

---

2. Overall Description

2.1 Product Perspective

The Invoice Generation system is a standalone web application that integrates both frontend and backend components. The backend is developed using Flask, while the frontend uses React. It operates on local servers but can be deployed on cloud infrastructure for scalability.

2.2 Product Features

Key features include:

- Invoice Creation: Create new invoices with detailed line items.
- Customer Management: Store and manage customer details for easy retrieval.
- Invoice Preview & Printing: View and print invoices in multiple formats.
- User Authentication: Secure the system with user login and role-based access.

2.3 User Classes and Characteristics

- Administrators: Have full control over the system, including managing users and settings.
- Standard Users: Can create, view, and print invoices but have limited access to administrative functions.
- Accountants: Can view and manage invoice records and generate financial reports.

2.4 Operating Environment

The system operates within a web environment and can be accessed via modern web browsers (e.g., Chrome, Firefox, Edge). It is developed using Flask for the backend, React for the frontend, and communicates via HTTP.

2.5 Design and Implementation Constraints

- Compatibility: The system must be compatible with various web browsers and devices.
- Security: Implement robust authentication and encryption to protect sensitive data.
- Performance: Ensuring the system can handle multiple users and high data loads efficiently.

2.6 User Documentation

User documentation will include guides on:

- Creating invoices.
- Managing customer data.
- Previewing and printing invoices.

2.7 Assumptions and Dependencies

- Users have basic knowledge of web applications.
- The system will be hosted on a reliable server.
- Dependencies include Flask, React, and a database like MySQL or SQLite.

---

3. System Features

3.1 System Feature 1: Invoice Creation

Description:
Users can create invoices by adding products or services, specifying quantities, prices, and taxes. The system will calculate the total amount automatically.

Functionality:

- Input fields for item details, quantity, price.
- Automatic calculation of totals.

- Option to save or preview the invoice.

3.2 System Feature 2: Customer Management

Description:
The system allows users to store and manage customer details, which can be retrieved during invoice creation.

Functionality:

- Add new customer details.
- Edit or delete existing customer records.
- Auto-fill customer information in invoices.

3.3 System Feature 3: Invoice Preview & Printing

Description:
Users can preview invoices before printing and choose between A4 or thermal print formats.

Functionality:

- Preview invoice layout.
- Print in different formats.
- Option to save a PDF version.

3.5 System Feature 5: User Authentication

Description:
Ensure that only authorized users can access the system and perform actions
Functionality:

- Secure login page.

---

4. External Interface Requirements

4.1 User Interfaces

The user interface should be simple, intuitive, and accessible across devices. Key UI components include:

- Invoice Creation Form: Input fields for item details and customer information.
- Customer Management Interface: Forms for adding, editing, and deleting customer records.
- Authentication Pages: Login and registration forms with clear prompts.

4.2 Hardware Interfaces

As a web application, there are no specific hardware interfaces required. The system should function on any device capable of running a modern web browser.

4.3 Software Interfaces

- Backend: Flask server for handling business logic and data storage.
- Frontend: React-based user interface.
- Database: MySQL or SQLite for storing invoice and customer data.
- APIs: Axios for communicating between frontend and backend.

4.4 Communications Interfaces

- HTTP/HTTPS: For secure communication between the frontend and backend.
- SMTP: For sending invoices via email if required in future enhancements.

---

5. Other Nonfunctional Requirements

5.1 Performance Requirements

- Response Time: Ensure that actions like invoice creation and customer management are processed within 2 seconds.
- Scalability: The system should handle an increasing number of users and data records efficiently.

5.2 Safety Requirements

The system does not deal with critical operations, but basic data protection measures will be in place.

5.3 Security Requirements

- Authentication: Implement secure login with encrypted passwords.
- Data Encryption: Sensitive data, such as customer information and invoices, will be encrypted.

5.4 Software Quality Attributes

- Reliability: The system should run smoothly without unexpected crashes.
- Maintainability: Code should be clean and modular for easy updates.
- Usability: Ensure the interface is user-friendly and accessible.

8.Literature Survey

This section reviews established knowledge related to invoice generation systems, focusing on both traditional and modern approaches. The review encompasses a historical perspective, recent developments, and the theories and techniques applied in the practical implementation of invoice generation systems.

Overview of Invoice Generation Systems

Historically, invoice generation has been a manual process that required significant time and attention to detail, often leading to errors and inefficiencies. The shift to automated invoice systems has been driven by the need for faster, more accurate, and scalable solutions across various industries, including retail, wholesale, and service sectors. Modern systems aim to automate not just the generation of invoices but also the validation of inventory, application of discounts, and tax calculations, ensuring that businesses can operate more efficiently.

Technologies Used in Automated Systems

Advancements in web technologies have revolutionized invoice generation. React, with its component-based architecture, provides an excellent framework for building dynamic and responsive user interfaces. Coupled with backend technologies, these systems enable real-time data handling and scalable operations. The ability to validate inventory levels before finalizing invoices is particularly crucial, preventing errors related to over-selling and stock mismanagement. In this project, React is used for the frontend, with plans for future backend integration to manage persistent data storage and retrieval.

User Experience and Interface Design

Research emphasizes the importance of intuitive user interfaces in enhancing the usability of invoice generation systems. A well-designed UI not only improves user satisfaction but also reduces the time required to complete tasks, such as creating and printing invoices. In this project, the React-based UI is designed to be user-friendly, allowing users to navigate through the system easily, validate inventory, and generate invoices with minimal effort. Future improvements will focus on optimizing the interface for better performance and adding features like customizable invoice templates.

Challenges and Future Directions

Despite advancements, challenges remain in ensuring system scalability, managing large datasets, and maintaining real-time performance as the application grows. ensuring that the system remains secure and reliable as it scales will be a key focus in future iterations. Addressing these challenges will be crucial for the continued improvement and adoption of automated invoice generation systems.

9.Physical Model / Analysis

Overview

In the context of an invoice generation system, a physical model analysis involves designing the system's hardware and infrastructure to ensure it can handle the necessary processing, storage, and networking tasks efficiently. This analysis also addresses performance, scalability, and security to provide a reliable solution for generating and managing invoices.

System Architecture
The physical architecture of the invoice generation system includes several key components:

- **Servers:** The system requires web servers to host the React frontend application and application servers to handle backend processes (e.g., Node.js or Django in future phases). Database servers will store and manage inventory data, customer information, and generated invoices. The current setup involves session storage, but future updates will integrate persistent storage using relational databases like MySQL or NoSQL solutions like MongoDB.
- **Networking:** A robust networking infrastructure is essential for seamless communication between the frontend and backend components. This includes ensuring low-latency connections within the hosting environment through local area networks (LAN) and external user access via wide area networks (WAN). Additionally, secure communication protocols (HTTPS) are vital for protecting data transmission.
- **Storage:** Adequate storage solutions are necessary to manage inventory records, customer details, and invoice data. Future implementations will involve a structured database to persistently store and retrieve this information. File storage solutions may also be used for handling static assets, such as invoice templates or exported PDFs.

Load Balancing and Scalability
To ensure that the system can handle a high volume of invoice generation requests and concurrent users, load balancing techniques will be implemented in future versions. Load balancers will distribute incoming traffic across multiple servers, preventing any single server from becoming a bottleneck. This will enhance the system's scalability, allowing it to accommodate growing user demands and providing a responsive experience even under heavy load.

Security Measures
Security is a critical aspect of the physical model, ensuring that sensitive data such as customer details and financial records are protected:

- **Firewalls:** These protect the servers and network from unauthorized access and potential threats.
- **Encryption:** Data transmission between the frontend and backend components must be encrypted using protocols like SSL/TLS to ensure the privacy and integrity of the data.
- **Regular Backups:** Implementing regular backup routines will prevent data loss and ensure system recovery in case of hardware failure, cyber incidents, or data corruption. This will include backing up both inventory and invoice data.

Performance Optimization
Optimizing system performance involves continuous monitoring and adjustments to hardware and network configurations:

- **Server Resource Management:** Monitoring CPU, memory, and disk usage on servers to prevent resource exhaustion and ensure optimal performance. Scaling resources based on demand will also be important as the system grows.
- **Network Latency:** Minimizing network latency is essential for improving response times and enhancing the overall user experience. This may involve optimizing routing paths and reducing data transfer times between the frontend and backend.
- **Database Tuning:** Regular optimization of database queries, indexes, and schema designs will improve data retrieval speeds, especially as the inventory and invoice data expand.

10.Network Model/mathematical model/Design

The network model describes the architecture and communication flow between different components of the system. Below is a detailed explanation of the current and future versions of the project's network model:

Current Version:

- **Front-end:** The React application runs in a web browser and handles all user interactions.

Advanced Version:

- **Architecture:** The system will incorporate a client-server architecture.
- **Front-end Communication:** The React front-end will communicate with the back-end server through RESTful API calls.
- **Back-end Operations:** These API requests will manage actions like:
  - Logging in
  - Retrieving inventory data
  - Submitting invoices
  - Validating quantities
- **Server-side Processing:** The back-end server will:
  - Process these requests
  - Interact with the database to fetch or update data
  - Return responses to the front-end

Deployed Environment:

Hosting:

- Frontend: Hosted on a cloud platform (e.g., AWS, Netlify).
- Backend: Hosted on a server (e.g., Heroku, AWS EC2).

Security: Secure communication protocols (e.g., HTTPS) will ensure data integrity and security.

Optimization: A Content Delivery Network (CDN) might be used to optimize the delivery of static assets like images and stylesheets.

11.Agile Process Overview

In the invoice generation project, I utilized Agile methodology to ensure a structured and collaborative development process. Here’s a summary of the work I’ve done following the Agile process:

1. Sprint Planning
I began by defining the project goals and user stories related to invoice generation and inventory management. Key objectives included creating a functional landing page, developing the dashboard for inventory management, and implementing the invoice generation feature. I prioritized tasks based on their importance and dependencies, creating a detailed backlog for each sprint.

2. Daily Stand-ups
I participated in daily status updates to update progress, identify roadblocks, and plan next steps. These updates were crucial for maintaining clear idea within the team, ensuring that everyone was aligned on their tasks and deadlines, and facilitating quick resolution of any issues.

3. Development Iterations
The development process was organized into iterative cycles, focusing on specific features:

- **Frontend Development:** I developed the React-based UI, including the landing page with login and signup functionalities, a dashboard for viewing and managing inventory, and an invoice generation page. This involved designing user-friendly interfaces that allow users to add products, validate quantities, and generate invoices.

4. Continuous Feedback
I regularly sought feedback from team members and stakeholders on the functionality and design of the application. This iterative feedback loop allowed for timely adjustments and improvements, ensuring that the features met user needs and project requirements.

5. Testing and Review
During each sprint, I conducted unit tests and integration tests to ensure the quality and functionality of the implemented features. Participation in code reviews allowed me to receive constructive feedback and learn from peers, improving both my code and the overall project.thanks for react test library.

6. Sprint Review and Retrospective
At the end of each sprint, I demonstrated the completed features to stakeholders for approval and feedback. I also engaged in retrospective meetings to reflect on what worked well and what could be improved. These reflections facilitated continuous improvement and adaptability throughout the project.

Contributions

- **UI Development:** Designed and implemented user-friendly interfaces for the landing page, inventory dashboard, and invoice generation functionalities.
- **Feature Implementation:** Developed core features such as product quantity validation, invoice preview, and print options, ensuring a smooth user experience.
- **Collaboration:** Actively collaborated with team members, contributing to discussions and decisions that shaped the project's direction.

Conclusion

By following the Agile methodology, I contributed to a flexible and adaptive development process that emphasized collaboration, continuous feedback, and iterative improvement. This approach resulted in a successful invoice generation system, equipped to handle the needs of users and adaptable to future enhancements.

12. **Implementation/Simulation**

The processes you've to performe for the IIMR project setup:

1. Cloning the Git Repository

- Create a project folder on your PC where you'll work.

  Inside the folder, clone the repository using:

```
git clone -b UpdatedIIMR https://github.com/CompSci-Technology-Solutions/IIMR.git
```

- After cloning, you'll have your project structure.

2. Setting Up a Virtual Environment

Outside the IIMR folder, create a virtual environment:

```
python -m venv env
```

- You can activate your environment in powershell or commandprompt.

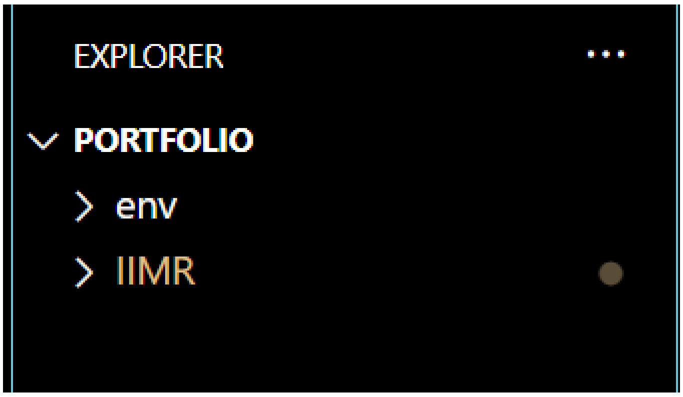
```
For powershell    :    ./env/scripts/activate

make sure your command start from ./

For commandprompt :      env/scripts/activate
```

You have to run these commands in the folder path where you've created the environment.

- **Note:** Make sure to activate the virtual environment every time you work on the codebase. This helps prevent conflicts with global installations by keeping dependencies isolated in the virtual environment.

Example structure of your project expected to be.

Focus on IIMR and the env folders.



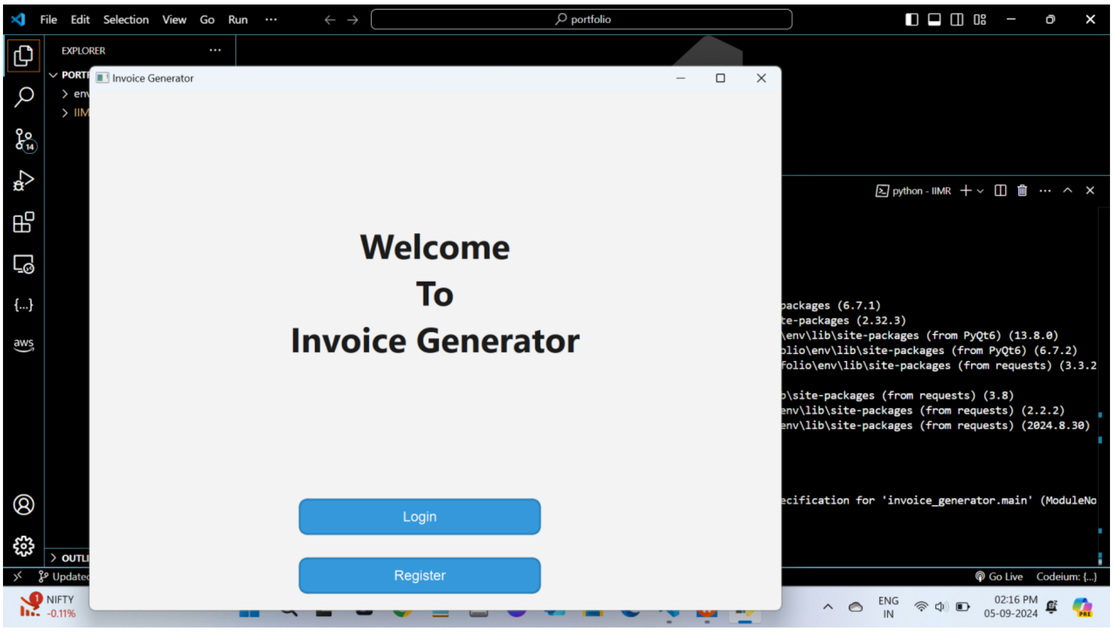
3. Running the IIMR Project Components

Navigate to the IIMR Directory:

```
cd IIMR
```

Running the PyQt UI:

```
pip install PyQt6 requests

python -m invoice_generator.main
```



```
packages (6.7.1)
te-packages (2.32.3)
\env\lib\site-packages (from PyQt6) (13.8.0)
olio\env\lib\site-packages (from PyQt6) (6.7.2)
folio\env\lib\site-packages (from requests) (3.3.2

b\site-packages (from requests) (3.8)
env\lib\site-packages (from requests) (2.2.2)
env\lib\site-packages (from requests) (2024.8.30)

ecification for 'invoice_generator.main' (ModuleNo
```

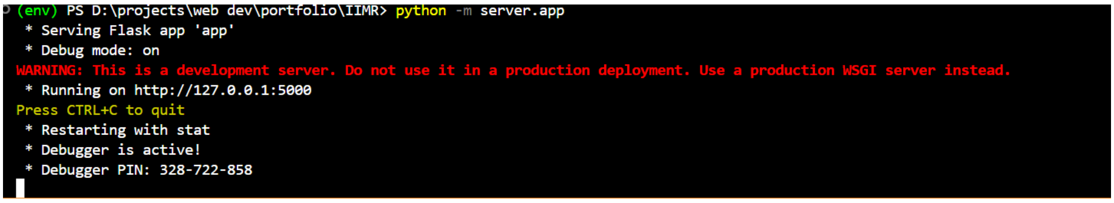
Running the Server:

Run the following command at IIMR directory:
If you have any packages to be installed, follow the command.

```
pip install <packagename>
```

To run the server use :

```
python -m server.app
```



```
(env) PS D:\projects\web dev\portfolio\IIMR> python -m server.app
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 328-722-858
```

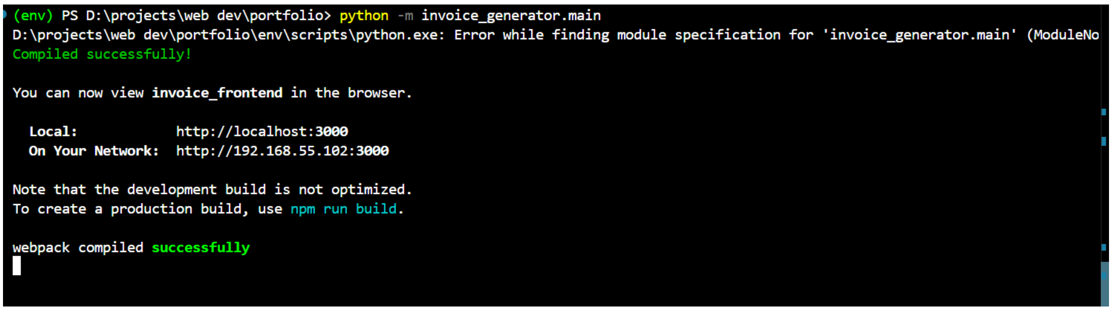
Running the React UI:

make sure you are inside IIMR\invoice_frontend directory

```
npm install   -- installs necessary node modules
npm start    --  starts your development server
```

You can now view invoice_frontend in the browser.

```
http://localhost:3000
```



```
(env) PS D:\projects\web dev\portfolio> python -m invoice_generator.main
D:\projects\web dev\portfolio\env\scripts\python.exe: Error while finding module specification for 'invoice_generator.main' (ModuleNo
Compiled successfully!

You can now view invoice_frontend in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.55.102:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

Appendix :-

13.Programme Listing/Code :

App.js file :

```
// App.js
```

```
import React from 'react';

import {BrowserRouter, Routes, Route} from 'react-router-dom';

import Home from './templates/home';

import Dashboard from './templates/dashboard';

import Invoice from './templates/invoice';

import Login from './templates/login';

import Register from './templates/register';

import Trailapi from './templates/trail_api';

import GenerateInvoice from './templates/generate_invoice';

import MessageBox from './components/messagebox';

function App() {

return (

<div>

<BrowserRouter>

<Routes>

<Route path="/" element={<Home />} />

<Route path='/login' element={<Login />} />

<Route path='/register' element={<Register />} />

<Route path="/dashboard" element={<Dashboard />} />

<Route path="/invoice" element={<Invoice />} />

<Route path="/trail_api" element={<Trailapi />} />

<Route path="/generate_invoice" element={<GenerateInvoice />} />

<Route path="/messagebox" element={<MessageBox />} />

</Routes>

</BrowserRouter>

</div>

);

}

export default App;
```

home.js

```
import React from 'react';

import { Link } from 'react-router-dom';

import '../styles/home.css';

import profilepic from '../assests/profilepic.png';

export default function Home() {

    return (

        <div className="home-container">

            <nav className="navbar">

                <div className="logo-container">

                     <img src={profilepic} alt="CSTS logo" className="logo" />

                    <span className="logo-text">Compsci Technologies</span>

                </div>

                <div className="nav-links">

                    <Link to="/" className="nav-link">Home</Link>

                    <Link to="/dashboard" className="nav-
link">Dashboard</Link>

                    <Link to="/invoice" className="nav-link">Invoice</Link>

                </div>

            </nav>

            <div className="main-content">

                <div className="left-content">

                    <h1 className="main-heading">Compsci Technologies</h1>

                    <div className="content">

                        <p>Welcome to Invoice Generator!</p>

                        <p>Our tool helps you create professional invoices
with ease. Whether you're a freelancer, small business owner, or large
enterprise, our invoice generator simplifies the process of billing your
clients. Keep track of your products, generate detailed invoices, and ensure
timely payments.</p>
```

```
                    </div>
                </div>
                <div className="right-content">
                    <div className="button-container">
                        <Link to="/login" className="link-button">Login</Link>
                        <Link to="/register" className="link-button">Register</Link>
                    </div>
                </div>
            </div>
        </div>
    );
}
```

home.css

```
body {
    margin: 0;
    font-family: Arial, sans-serif;
}

.home-container {
    display: flex;
    flex-direction: column;
    height: 100vh;
    background-color: #282c34;
    color: white;
    text-align: left;
}
.home-container .navbar {
    background-color: #1abc9c;
    padding: 10px;

    display: flex;

    justify-content: space-between;

    align-items: center;

}


.home-container .logo-container {

    display: flex;

    align-items: center;

    overflow: auto;

}


.home-container .logo {

    width: 40px;

    height: 40px;

    border-radius: 50%;

    margin-right: 10px;

    cursor: pointer;

}


.home-container .logo-text {

    font-size: 1.5em;

    color: white;

}


.home-container .nav-links {

    display: flex;

    gap: 20px;

}
```

```
.home-container .nav-link {
    color: white;
    text-decoration: none;
    font-size: 1.2em;
    padding: 10px;
    border-radius: 5px; /* Rounded corners */
    transition: background-color 0.3s ease, color 0.3s ease; /* Smooth transition */
}

.home-container .nav-link:hover {
    background-color: #16a085; /* Darker green background on hover */
    color: #fff; /* White text color on hover */
}

.home-container .main-content {
    display: flex;
    flex: 1;
    padding: 20px;
}

.home-container .left-content {
    flex: 1;
    display: flex;
    flex-direction: column;
    justify-content: center;
    padding-right: 20px;
}
```

```
.home-container .main-heading {

    font-size: 3em;

    margin-bottom: 20px;

}


.home-container .content {

    max-width: 600px;

    font-size: 1.3em;

}


.home-container .right-content {

    flex: 1;

    display: flex;

    flex-direction: column;

    justify-content: center;

    align-items: flex-end;

    padding-left: 20px; /* Add padding for spacing */

    margin-right: 50px;

}


.home-container .button-container {

    display: flex;

    flex-direction: column;

    gap: 20px; /* Increase gap for better spacing */

    margin-top: 20px;

}

.home-container .link-button {

    padding: 15px 30px; /* Increased padding for better button size */

    background-color: #1abc9c; /* Green background */
    border: none;

    border-radius: 10px; /* Rounded corners */

    color: white;

    text-decoration: none;

    font-size: 1rem;

    cursor: pointer;

    text-align: center;

    width: 200px; /* Increased width for better button size */

    transition: background-color 0.3s ease, transform 0.3s ease; /* Smooth
transition */

}



.home-container .link-button:hover {

    background-color: #16a085; /* Darker green on hover */

    transform: scale(1.05); /* Slightly enlarge the button on hover */

}
```

login.js

```
import React, {useState} from 'react';

import axios from 'axios';

import { Link, useNavigate } from 'react-router-dom';

import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';

import { faUser, faLock } from '@fortawesome/free-solid-svg-icons';

import '../styles/login.css';



export default function Login() {

    const [username, setUsername] = useState('');

    const [password, setPassword] = useState('');
    const navigate = useNavigate();


    const validateForm = () => {
        if (!username || !password) {
            alert('Both fields are required');
            return false;
        }


        if (!/^[a-zA-Z][a-zA-Z0-9_-]{4,19}$/.test(username)) {
            alert('Invalid Username: Must start with a letter, be 5-20
characters long, and can only contain letters, numbers, underscores, and
hyphens.');
            return false;
        }


        if (!/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[!@#$%^&*()_+\-
=\\[\]{};':"\\|,.<>\/?])[A-Za-z\d!@#$%^&*()_+\-
=\\[\]{};':"\\|,.<>\/?]{8,}$/.test(password)) {
            alert('Invalid Password: Must be at least 8 characters long,
with at least one uppercase letter, one lowercase letter, one digit, and one
special character.');
            setPassword('');
            return false;
        }
        return true;
    };


    const handleSubmit = (e) => {
        e.preventDefault();


        if (!validateForm()) {
            return;
        }

        const formData = {
            username: username,
            password: password,
        };

        axios.post('http://127.0.0.1:5000/login', formData, {
            headers: {
                'Content-Type': 'application/json'
            }
        })
        .then(response => {
            if (response.data.status === 'success') {
                console.log(response.data);
                localStorage.setItem('username', response.data.username);
                alert('Login successful!');
                // You can save the company_id or any other data as needed
                navigate('/dashboard');
            } else {
                alert(`Error: ${response.data.message}`);
            }
        })
        .catch(error => {
            console.error('There was an error!', error);
            const errorMessage = error.response?.data?.message || 'Login
failed, please try again later.';
            alert(errorMessage);
        });
```

```
    };

    return (
        <div className="login-container">
            <Link to="/" className="back-button">&larr;</Link>
            <div className="login-content">
                <h1>Login</h1>
                <form className="login-form" onSubmit={handleSubmit}>
                    <div className="inputContainer">
                        <FontAwesomeIcon icon={faUser} className="inputIcon"
/>
                        <input
                            type="text"
                            name="username"
                            className="inputField"
                            required
                            data-testid="username"
                            value={username}
                            onChange={(e) => setUsername(e.target.value)}
                            placeholder="Username"
                        />
                    </div>
                    <div className="inputContainer">
                        <FontAwesomeIcon icon={faLock} className="inputIcon"
/>
                        <input
                            type="password"
                            name="password"
                            className="inputField"
                            required
                            data-testid="password"

                            value={password}

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

                            placeholder="Password"

                        />

                    </div>

                    <button type="submit" className="login-
button">Login</button>

                </form>

                <p className="signup-link">

                    Don't have an account? <Link to="/register"
className="signup-button">Sign Up</Link>

                </p>

            </div>

        </div>

    );

}
```

login.css

```
.login-container {

    display: flex;

    flex-direction: column;

    justify-content: center;

    align-items: center;

    height: 100vh;

    background-color: #1e3a2f; /* Dark greenish background */

    color: #000000; /* Ensure the text color is consistent */

    position: relative;

}



.login-content {
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    background-color: #ffffff; /* White background */
    padding: 40px;
    border-radius: 30px;
    box-shadow: 0px 0px 40px rgba(0, 0, 0, 0.1); /* Slightly stronger shadow
for better emphasis */
    width: 400px; /* Increased width for better layout */
    text-align: center;
}


.login-form {
    display: flex;
    flex-direction: column;
    align-items: center;
    gap: 20px;
    width: 100%;
}


.login-form label {
    display: flex;
    flex-direction: column;
    align-items: flex-start; /* Align items to the start (left) */
    width: 100%;
    font-size: 1rem;
    font-weight: 500;
    position: relative;
    color: #333333; /* Darker text color for labels */
```

```
}

.inputContainer {
    position: relative;
    width: 100%;
}

.inputIcon {
    position: absolute;
    left: 18%; /* More space between the icon and the border */
    top: 50%;
    transform: translateY(-50%);
    color: #999999; /* Lighter color for the icons */
    font-size: 1.2rem;
}

.inputField {
    width: 75%;
    height: 45px; /* Slightly taller input fields */
    background-color: #ffffff; /* White background for input */
    border: 2px solid #b0b0b0; /* Lighter border color */
    border-radius: 30px;
    padding-left: 50px; /* Adjusted for icon space */
    padding-right: 15px;
    color: #333333; /* Darker text color */
    font-size: 1rem;
    font-weight: 500;
    box-sizing: border-box;
    transition: border-color 0.3s;
```

```
}

.inputField:focus {
    border-color: #2e8b57; /* Border color changes on focus */
    outline: none;
}

.login-button {
    position: relative;
    width: 90%;
    border: none; /* Removed border */
    background-color: #2e8b57; /* Greenish background */
    height: 40px;
    color: white;
    font-size: 1rem; /* Increased font size */
    font-weight: 500;
    letter-spacing: 1px;
    border-radius: 30px;
    margin: 10px 0;
    cursor: pointer;
    overflow: hidden;
    text-align: center;
    transition: background-color 0.3s;
}

.login-button:hover {
    background-color: #276b46; /* Darker green on hover */
}
```

```
.login-button::after {
    content: "";
    position: absolute;
    background-color: rgba(255, 255, 255, 0.253);
    height: 100%;
    width: 150px;
    top: 0;
    left: -200px;
    border-bottom-right-radius: 100px;
    border-top-left-radius: 100px;
    filter: blur(10px);
    transition: transform 0.5s;
}


.login-button:hover::after {
    transform: translateX(600px);
    transition: transform 0.5s;
}
.signup-link {
    font-size: 0.9rem;
    font-weight: 500;
    color: rgb(26, 22, 22);
    margin-top: 15px;
}


.signup-button {
    font-size: 0.9rem; /* Increased font size for consistency */
    font-weight: 500;
    background-color: #2e2e2e;
    color: white;

    text-decoration: none;

    padding: 8px 15px;

    border-radius: 20px;

    transition: background-color 0.3s;}

.signup-button:hover {

    background-color:rgb(73, 65, 65) ;

}
```

register.js

```
import React, { useState } from 'react';

import { Link, useNavigate } from 'react-router-dom';

import axios from 'axios';

import '../styles/register.css';



export default function Register() {

    const [username, setUsername] = useState('');

    const [password, setPassword] = useState('');

    const [confirmPassword, setConfirmPassword] = useState('');

    const [companyName, setCompanyName] = useState('');

    const [email, setEmail] = useState('');

    const [address, setAddress] = useState('');

    const [gstin, setGstin] = useState('');

    const [pan, setPan] = useState('');

    const [passwordVisible, setPasswordVisible] = useState(false);

    const [confirmPasswordVisible, setConfirmPasswordVisible] =
useState(false);

    const navigate = useNavigate();
    const validateForm = () => {

        if (!username || !password || !confirmPassword || !companyName
|| !email || !address || !gstin || !pan) {

            alert('Please complete all required fields.');

            return false;

        }



        if (!/^[a-zA-Z][a-zA-Z0-9_-]{4,19}$/.test(username)) {

            alert('Username must be between 5-20 characters and start with a
letter. Only letters, numbers, underscores, and hyphens are allowed.');

            return false;

        }



        if (!/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[!@#$%^&*()_+\-
=\[\]{};':"\\|,.<>/?])[A-Za-z\d!@#$%^&*()_+\-
=\[\]{};':"\\|,.<>/?]{8,}$/.test(password)) {

            alert('Password must be at least 8 characters long, include
uppercase, lowercase, digits, and special characters.');

            setPassword('');

            return false;

        }



        if (password !== confirmPassword) {

            alert('Passwords do not match.');

            return false;

        }



        if (!/^[a-zA-Z][a-zA-Z0-9_-]{4,19}$/.test(companyName)) {

            alert('Company Name must be between 5-20 characters and start
with a letter. Only letters, numbers, underscores, and hyphens are
allowed.');

            return false;

        }
        if (!/^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/.test(email)) {

            alert('Please enter a valid email address.');

            return false;

        }



        if (!/^[a-zA-Z][a-zA-Z0-9_-]{4,19}$/.test(address)) {

            alert('Address must be between 5-20 characters and start with a
letter. Only letters, numbers, underscores, and hyphens are allowed.');

            return false;

        }

        // GSTIN is a valid 15-character alphanumeric string with uppercase
letters and digits only

        if (!/^[0-9A-Z]{15}$/.test(gstin)) {

            alert('Please enter a valid GSTIN.');

            return false;

        }

        // PAN is a valid 10-character alphanumeric string with uppercase
letters and digits only

        if (!/^[A-Z]{5}[0-9]{4}[A-Z]$/.test(pan)) {

            alert('Please enter a valid PAN.');

            return false;

        }

        return true;

    };



    const togglePasswordVisibility = () => {

        setPasswordVisible(!passwordVisible);

    };



    const toggleConfirmPasswordVisibility = () => {
```

```
    setConfirmPasswordVisible(!confirmPasswordVisible);

};

const handleSubmit = (e) => {

    e.preventDefault();


    if (!validateForm()) {

        return;

    }


    const formData = {

        username: username,

        password: password,

        company_name: companyName,

        email: email,

        address: address,

        gstin: gstin,

        pan: pan,

    };


    axios.post('http://127.0.0.1:5000/register', formData)

        .then(response => {

            if (response.data.status === 'success') {

                console.log(response.data.message);

                alert('User registered successfully!');

                navigate('/login');

            } else {

                alert(`Error: ${response.data.message}`);

            }

        })
```

```
            .catch(error => {
                console.error('There was an error!', error);
                console.log('Error response:', error.response);
                if (error.response.status === 400) {
                    const errorMessage = error.response.data.message ||
'Registration failed, please try again later.';
                    alert(errorMessage);
                } else {
                    alert('Registration failed, please try again later.');
                }
            });
    };

    return (
        <div className="register-container">
            <Link to="/" className="back-button">&larr;</Link>
            <div className="register-content">
                <h1 className="heading">Register</h1>
                <form className="signup-form" onSubmit={handleSubmit}>
                    <div className="input-container">
                        <label className="reg-labels"
htmlFor="username">Username</label>
                        <input
                            type="text"
                            className="input-field"
                            placeholder="Username"
                            value={username}
                            onChange={(e) => setUsername(e.target.value)}
                            data-testid="username"
                        />
                    </div>

                    <div className="input-container">

                        <div className="password-container">

                        <label className="reg-labels"
htmlFor="password">Password</label>

                            <input

                                type={passwordVisible ? 'text' : 'password'}

                                id="password"

                                data-testid="password"

                                className="input-field"

                                placeholder="Enter your password"

                                value={password}

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

                            />

                            {password && (

                                <div

                                    id="toggle-password-visibility"

                                    className="lookicon"

                                    onClick={togglePasswordVisibility}

                                >

                                    {passwordVisible ? '□' : '□'}

                                </div>

                            )}

                        </div>

                    </div>

                    <div className="input-container">

                        <div className="password-container">

                        <label className="reg-labels" htmlFor="confirm-
password">Confirm Password</label>

                            <input
```

```
                                type={confirmPasswordVisible ? 'text' : 'password'}
                                id="confirm-password"
                                data-testid="confirm-password"
                                className="input-field"
                                placeholder="Confirm your password"
                                value={confirmPassword}
                                onChange={(e) => setConfirmPassword(e.target.value)}
                            />
                            {confirmPassword && (
                                <div
                                    id="toggle-confirm-password-visibility"
                                    className="lookicon"
                                     onClick={toggleConfirmPasswordVisibility}
                                >
                                    {confirmPasswordVisible ? '□' : '□'}
                                </div>
                            )}
                        </div>
                    </div>
                    <div className="input-container">
                        <label className="reg-labels" htmlFor="company-name">Company Name</label>
                        <input
                            type="text"
                            className="input-field"
                            placeholder="Company Name"
                            value={companyName}
                            onChange={(e) => setCompanyName(e.target.value)}
                            data-testid="company-name"
                        />

                    </div>

                    <div className="input-container">

                        <label className="reg-labels"
htmlFor="email">Email</label>

                        <input

                            type="email"

                            className="input-field"

                            placeholder="Email"

                            value={email}

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

                            data-testid="email"

                        />

                    </div>

                    <div className="input-container">

                        <label className="reg-labels"
htmlFor="address">Address</label>

                        <input

                            type="text"

                            className="input-field"

                            placeholder="Address"

                            value={address}

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

                            data-testid="address"

                        />

                    </div>

                    <div className="input-container">

                        <label className="reg-labels"
htmlFor="gstin">GSTIN</label>

                        <input

                            type="text"
```

```
                            className="input-field"
                            placeholder="GSTIN"
                            value={gstin}
                            onChange={(e) => setGstin(e.target.value)}
                            data-testid="gstin"
                        />
                    </div>
                    <div className="input-container">
                        <label className="reg-labels"
htmlFor="pan">PAN</label>
                        <input
                            type="text"
                            className="input-field"
                            placeholder="PAN"
                            value={pan}
                            onChange={(e) => setPan(e.target.value)}
                            data-testid="pan"
                        />
                    </div>
                    <button type="submit" className="submit-
button">Register</button>
                </form>
                <p className="login-link">
                    Already have an account? <Link to="/login"
className="login-button">Log In</Link>
                </p>
            </div>
        </div>
    );
}
```

Register.css

```
/* src/styles/signup.css */

.register-container {

    display: flex;

    flex-direction: column;

    align-items: center;

    min-height: 100vh;

    background-color: #282c34; /* Dark greenish background */

    color: #333;

    padding: 20px; /* Padding to ensure content isn't too close to edges */

    box-sizing: border-box;

    overflow-y: auto; /* Allow scrolling if content overflows */

    position: relative; /* Allows absolute positioning inside */

}


.back-button {

    position: absolute;

    top: 20px;

    left: 20px;

    background-color: #2e8b57; /* Dark green for visibility */

    width: 40px;

    height: 40px;

    border-radius: 50%; /* Round button */

    color: white;

    border: none;

    font-size: 1.5rem;

    text-align: center;
    line-height: 40px; /* Center text vertically */
    transition: background-color 0.3s;
    text-decoration: none;
    display: flex;
    justify-content: center;
    align-items: center;
}

.back-button:hover {
    background-color: #1c6d3f; /* Darker green on hover */
}

.register-content {
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    background-image: radial-gradient( circle farthest-corner at 10%
20%,  rgba(176,229,208,1) 42%, rgba(92,202,238,0.41) 93.6% );
    padding: 30px;
    border-radius: 20px;
    width: 90%;
    max-width: 450px; /* Adjusted max-width for form container */
    text-align: center;
    margin-top: 60px; /* Space from the top */
    position: relative;
    overflow: auto; /* Allow scroll if content overflows */
}

.heading {
    font-size: 2.5rem;

    color: #2e8b57; /* Dark green for heading */

    font-weight: 700;

    margin-bottom: 30px;

}


.signup-form {

    display: flex;

    flex-direction: column;

    align-items: center;

    gap: 20px; /* Increased gap between fields */

    width: 100%;

}

.reg-labels {

    display: block; /* Ensure the label takes up the full width */

    text-align: left; /* Align text to the left */

    margin-bottom: 8px; /* Add some space between the label and input field
*/

    font-size: 0.9rem; /* Adjust the font size */

    font-weight: 600; /* Make the text slightly bolder */

    color: #333; /* Dark color for the label */

    padding-left: 35px; /* Add padding to match the input field's padding */

}


.input-container {

    width: 100%;

}


.input-field {

    width: calc(100% - 50px);
    height: 50px; /* Adjusted height for better visibility */
    background-color: white;
    border: 1.5px solid #c2c2c2; /* Light grey border */
    border-radius: 10em;
    padding-left: 15px;
    color: black;
    font-size: 0.9rem;
    font-weight: 500;
    box-sizing: border-box;
    transition: border-color 0.3s ease, box-shadow 0.3s ease; /* Smooth transition */
}

.input-field:focus {
    outline: none;
    border-color: #2e8b57; /* Green border on focus */
    box-shadow: 0 0 8px rgba(46, 139, 87, 0.5); /* Green glow effect on focus */
}

.input-field::placeholder {
    color: rgb(80, 80, 80);
    font-size: 0.9rem;
    font-weight: 500;
    transition: color 0.3s ease;
}

.input-field:focus::placeholder {
    color: grey; /* Grey placeholder on focus */
}
```

```
.password-container {
    position: relative;
    width: 100%;
}

.input-field {
    width: calc(100% - 50px); /* Adjust width to make room for the icon */
    height: 50px;
    background-color: white;
    border: 1.5px solid #c2c2c2;
    border-radius: 10em;
    padding-left: 15px;
    padding-right: 50px; /* Space for the icon */
    color: black;
    font-size: 0.9rem;
    font-weight: 500;
    box-sizing: border-box;
    transition: border-color 0.3s ease, box-shadow 0.3s ease;
}

.input-field:focus {
    outline: none;
    border-color: #2e8b57;
    box-shadow: 0 0 8px rgba(46, 139, 87, 0.5);
}

.input-field::placeholder {
    color: rgb(80, 80, 80);
    font-size: 0.9rem;

    font-weight: 500;

    transition: color 0.3s ease;

}


.input-field:focus::placeholder {

    color: grey;

}


.lookicon {

    position: absolute;

    top: 50%;

    right: 10%; /* Position the icon 15px from the right edge */

    transform: translateY(-50%);

    cursor: pointer;

    font-size: 1.5rem; /* Adjust as needed */

    color: #333;

}


.password-container .input-field {

    padding-right: 50px; /* Ensure there is space for the icon */



}

input[type=password]::-ms-reveal,

input[type=password]::-ms-clear {

  display: none;

}


.submit-button {
    position: relative;

    display: inline-block;

    padding: 15px 30px;

    text-align: center;

    letter-spacing: 1px;

    background: linear-gradient(109.6deg, rgba(61, 245, 167, 1) 11.2%,
rgba(9, 111, 224, 1) 91.1%);

    border: none; /* Remove border */

    border-radius: 10em;

    color: white; /* Text color should be white for better contrast */

    font-size: 18px;

    font-weight: bold;

    height: 50px;

    width: calc(100% - 20px);

    cursor: pointer;

    overflow: hidden;

    transition: background-position 0.5s ease, transform 0.3s ease; /*
Smooth transition */

    background-size: 200% 100%; /* Extend background size */

    background-position: 0% 0%; /* Initial position */

}



.submit-button:hover {

    background-position: 100% 0%; /* Move gradient position to right on
hover */

    transform: scale(1.003); /* Slightly scale up on hover for effect */

}



.submit-button:active {

    transform: scale(0.9);

}
```

```
.login-text{

    font-size: 0.9rem;

    font-weight: 500;

    color: black;

    margin-top: 15px;

}


.login-button {

    font-size: 0.7rem;

    font-weight: 500;

    background-color: #2e2e2e;

    color: white;

    text-decoration: none;

    padding: 8px 15px;

    border-radius: 20px;

}
```

dashboard.js

```
import React, { useState } from 'react';

import { useNavigate } from 'react-router-dom';

import '../styles/dashboard.css';


function Dashboard() {

    const [itemName, setItemName] = useState('');

    const [itemPrice, setItemPrice] = useState('');

    const [quantity, setQuantity] = useState(1);

    const [products, setProducts] = useState([]);

    const [stockQuantities, setStockQuantities] = useState({});
    const [error, setError] = useState('');
    const navigate = useNavigate();

    const handleAddProduct = () => {
        setError(''); // Clear previous errors
        /*
        // check if all fields are filled
        if (!itemName || !itemPrice || !quantity) {
            setError('All fields are required.');
            return;
        }
        */
        // Validate Item Name (letters, numbers, and one underscore or
hyphen)
        if (!/^(?=.*[a-zA-Z])([a-zA-Z0-9]*[_-]?[a-zA-Z0-
9]*)$/.test(itemName)) {
            alert('Invalid Item Name. It can contain letters, numbers, and
only one underscore (_) or hyphen (-).');
            return;
        }
        // Validate Item Price (must be a positive number)
        if (isNaN(itemPrice) || parseFloat(itemPrice) <= 0) {
            setError('Item price must be a positive number.');
            return;
        }
        // Validate Item Price format (no special characters, max two
decimal places)
        if (!/^\d+(\.\d{1,2})?$/.test(itemPrice)) {
            setError('Invalid Item Price format.');
            return;
        }
        // Validate Quantity (must be a positive integer)
        if (!/^\d+$/.test(quantity) || parseInt(quantity) <= 0) {
            setError('Quantity must be a positive integer.');
            return;
        }
        const newItem = { name: itemName, price: parseFloat(itemPrice),
quantity: parseInt(quantity) };
        console.log(newItem);


        const existingProductIndex = products.findIndex(product =>
product.name === itemName);


        if (existingProductIndex !== -1) {
            const existingProduct = products[existingProductIndex];
            const sno = existingProductIndex + 1;
            if (existingProduct.price !== newItem.price) {
                setError('Prices does not match check SNO :' + sno);
                return;
            }


            // Update quantity for the existing product
            const updatedProducts = products.map((product, index) => {
                if (index === existingProductIndex) {
                    const updatedQuantity = product.quantity +
newItem.quantity;
                    return { ...product, quantity: updatedQuantity };
                }
                return product;
            });
            setProducts(updatedProducts);
        } else {
            setProducts([...products, newItem]);

        }



        // Update stock quantities for the new product

        setStockQuantities(prevStock => ({

            ...prevStock,

            [itemName]: (prevStock[itemName] || 0) + quantity

        }));



        setItemName('');

        setQuantity(1);

        setItemPrice('');

    };

    const handleRemoveProduct = (index) => {

        const updatedProducts = [...products];

        updatedProducts.splice(index, 1);  // splice() removes the elements, 
even replaces ,...

        setProducts(updatedProducts);

    };



    const handleGoToInvoice = () => {

        if (products.length === 0) {

            setError('No products to generate an invoice.');

            return;

        }

        // passing products and stockQuantities as state in url but hidden 
with help of useNavigate()

        navigate('/invoice', { state: { products, stockQuantities } });

    };

    const user = localStorage.getItem('username');
```

```
return (
    <div className="dashboard-container">
        <h1 className='dashboard-container-h1'>Welcome {user}</h1>
        <div className="dashboard-content">
            <div className="add-product">
                <h2>Add Your Product</h2>
                <label htmlFor="item-name">
                    Item Name:
                    <input
                        id="item-name"
                        type="text"
                        value={itemName}
                        onChange={(e) => setItemName(e.target.value)}
                        placeholder="Enter your product name."
                        className="input-field"
                        data-testid="item-name"
                    />
                </label>

                <label htmlFor="item-price">
                    Item Price:
                    <input
                        id="item-price"
                        type="number"
                        value={itemPrice}
                        onChange={(e) => setItemPrice(e.target.value)}
                        placeholder="Enter your product price."
                        className="input-field"
                        min="0"
```

```
                            data-testid="item-price"
                        />
                    </label>

                    <label htmlFor='quantity'>
                        Quantity:
                        <input
                            className="input-field"
                            type='number'
                            id='quantity'
                            value={quantity}
                            min="1"
                            onChange={(e) => setQuantity(e.target.value)}
                            placeholder='Enter quantity' />
                    </label>

                    {/* Display error above the Add Product button */}
                    <div className="error-container">
                        {error && <div className="error">{error}</div>}
                    </div>

                    <button className="add-product-button"
onClick={handleAddProduct}>Add Product</button>
                </div>

                <div className="table-container">
                    <div className="table-wrapper">
                        <table>
                            <thead>
                                <tr>
```

```
                                    <th>Sno</th>

                                    <th>Name</th>

                                    <th>Quantity</th>

                                    <th>Price</th>

                                    <th>Actions</th>

                                </tr>

                            </thead>

                            <tbody>

                                {products.map((product, index) => (

                                    <tr key={index}>

                                        <td>{index + 1}</td>

                                        <td>{product.name}</td>

                                        <td>{product.quantity}</td>

                                         <td>₹ {product.price.toFixed(2)}</td>

                                        <td>

                                            <button className="remove-product-button" onClick={() => handleRemoveProduct(index)}>

                                                Remove

                                            </button>

                                        </td>

                                    </tr>

                                ))}

                            </tbody>

                        </table>

                    </div>

                </div>

            </div>



            {/* Global error handling for this page */}
            <button className="invoice-button"
onClick={handleGoToInvoice}>Go to Invoice Page</button>

        </div>

    );

}


export default Dashboard;
```

dashboard.css

```
/* General Dashboard Container */

.dashboard-container {

  display: flex;

  flex-direction: column;

  align-items: center;

  justify-content: center;

  min-height: 100vh;

  background-color: #282c34;

  padding: 20px;

  box-sizing: border-box;

  overflow: hidden; /* Prevent scrolling */

}


/* Dashboard Header */

.dashboard-container-h1 {

  color: white;

}


/* Main Content Layout */

.dashboard-content {
  display: flex;

  justify-content: space-between;

  width: 100%;

  max-width: 1200px; /* Adjust width as needed */

  gap: 20px; /* Space between sections */

}


/* Add Product Section */

.add-product {

  width: 40%; /* Both containers have the same width */

  height: 370px; /* Fixed height for both containers */

  padding: 20px;

  border-radius: 8px;

  background-color: #fff;

  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

  overflow: hidden; /* Hide overflow to prevent scrollbars */

  display: flex;

  flex-direction: column;

  align-items: center;

}


/* Error Message Container */

.error-container {

  width: 100%;

  text-align: center;

  margin-bottom: 10px; /* Space above the button */

}


.add-product label {
  font-size: 1rem;

  color: #333;

  margin-bottom: 8px;

  width: 80%;

  display: block;

  font-family: "Segoe UI", system-ui, sans-serif;

}


.add-product .input-field {

  font-size: 1rem;

  width: 100%;

  padding: 10px;

  border: 1px solid #ddd;

  border-radius: 20px;

  background-color: #fafafa;

  color: #333;

  transition: border-color 0.3s ease, box-shadow 0.3s ease;

}


.add-product .input-field::placeholder {

  color: #aaa;

}


.add-product .input-field:focus {

  outline: none;

  border-color: #007bff;

  box-shadow: 0 0 0 3px rgba(38, 143, 255, 0.3);

}
```

```
.add-product .error {

  color: red;

  font-size: 0.875rem;

}



.add-product .add-product-button {

  font-size: 1rem;

  font-family: "Segoe UI", system-ui, sans-serif;

  font-weight: 500;

  cursor: pointer;

  padding: 10px 20px;

  border: none;

  border-radius: 4px;

  color: #fff;

  background-image: linear-gradient(325deg, hsla(217, 100%, 56%, 1) 0%,
hsla(194, 100%, 69%, 1) 55%, hsla(217, 100%, 56%, 1) 90%);

  background-size: 280% auto;

  transition: background-position 0.8s ease, box-shadow 0.8s ease;

}



.add-product .add-product-button:hover {

  background-position: right top;

}



.add-product .add-product-button:focus, .add-product .add-product-
button:active {

  outline: none;

  box-shadow: 0 0 0 3px #fff, 0 0 0 6px hsla(217, 100%, 56%, 1);

}

/* Table Container */
```

```
.table-container {

  width: 50%; /* Both containers have the same width */

  height: 370px; /* Fixed height for both containers */

  padding: 20px;

  border-radius: 8px;

  background-color: #fff;

  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

  position: absolute; /* Position relative to the parent */

  right: 15px; /* Align to the right */

  overflow: hidden; /* Hide overflow to prevent scrollbars */

  -ms-overflow-style: none; /* IE and Edge */

  scrollbar-width: none; /* Firefox */

}


/* Lets Enable vertical and horizontal scrolling inside the table */

.table-container .table-wrapper {

  overflow-y: auto; /* Enable vertical scrolling */

  overflow-x: auto; /* Enable horizontal scrolling */

  max-height: 100%; /* Ensuring the wrapper takes full height */

}


/* Invoice Table */

.table-container table {

  width: 100%;

  border-collapse: collapse;

  margin-top: 20px;

}


/* Table Header */
```

```
.table-container th {
  padding: 12px;
  border: 1px solid #ddd;
  text-align: center;
  position: sticky;
  top: 0;
  z-index: 2;
  background-color: aquamarine;
}

/* Table Data Cells */
.table-container td {
  padding: 12px;
  border: 1px solid #ddd;
  text-align: left;
  background-color: #fff;
  color: #333;
}

/* Button Styling */
.table-container .remove-product-button {
  padding: 8px 16px;
  background-color: #dc3545;
  color: white;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  transition: background-color 0.3s ease;
}
```

```
.table-container .remove-product-button:hover {
  background-color: #bf081a;
}


/* Button Container for Aligning Actions */
.table-container .button-container {
  display: flex;
  justify-content: center;
  align-items: center;
  gap: 10px;
  padding: 0 10px;
}


/* Invoice Button */
.invoice-button {
  margin-top: 20px;
  font-size: 1rem;
  font-family: "Segoe UI", system-ui, sans-serif;
  font-weight: 500;
  cursor: pointer;
  padding: 10px 20px;
  border: none;
  border-radius: 4px;
  color: #fff;
  background-image: linear-gradient(325deg, hsla(217, 100%, 56%, 1) 0%,
hsla(194, 100%, 69%, 1) 55%, hsla(217, 100%, 56%, 1) 90%);
  background-size: 280% auto;
  transition: background-position 0.8s ease, box-shadow 0.8s ease;
}
```

```
.invoice-button:hover {

  background-position: right top;

}



.invoice-button:focus, .invoice-button:active {

  outline: none;

  box-shadow: 0 0 0 3px #fff, 0 0 0 6px hsla(217, 100%, 56%, 1);

}
```

invoice..js

```
import React, { useState, useEffect } from 'react';

import { useLocation, useNavigate } from 'react-router-dom';

import Select from 'react-select';

import 'jspdf-autotable';

import '../assests/profilepic.png';

import '../styles/invoice.css';

import MessageBox from '../components/messagebox';



const rs = '₹'; // Global currency symbol variable



export default function Invoice() {

    const navigate = useNavigate();

    const location = useLocation();

    const [showDiscountMessage, setShowDiscountMessage] = useState(false);

    const [products, setProducts] = useState([]); // contains all products

    const [selectedProduct, setSelectedProduct] = useState(null);  //
dropdown selected  .value for value selection

    const [quantity, setQuantity] = useState(1);  // input field related

    const [invoiceItems, setInvoiceItems] = useState([]); // table items
(data)
```

```
    const [userDetails, setUserDetails] = useState({

        name: '',

        address: '',

        email: '',

        mobile: ''

    }); //

    const [error, setError] = useState('');

    const [editingIndex, setEditingIndex] = useState(null); // to ckeck
editing index with index with index. so that you wont change other index
value.

    const [editingQuantity, setEditingQuantity] = useState(''); // on
clearin quantity its nothing,..



    const [stockQuantities, setStockQuantities] =
useState(location.state?.stockQuantities || {});

    console.log(stockQuantities);



    /*  ------   fixed discount      ----------

    const discount = 50; // example percentage

    const totalAmount = invoiceItems.reduce((acc, item) => acc +
item.totalPrice, 0);

    const cgst = totalAmount * 0.09;

    const sgst = totalAmount * 0.09;

    const subTotal = totalAmount + cgst + sgst;



    const discountedTotal = Math.max(subTotal - discount, 0); // Ensure
discount doesn't reduce total below zero

    const rounding = Math.round(discountedTotal) - discountedTotal;

    const finalAmount = discountedTotal + rounding;

     */



    const discountPercentage = 15; // 15% discount
    const totalAmount = invoiceItems.reduce((acc, item) => acc +
item.totalPrice, 0);
    const cgst = totalAmount * 0.09;
    const sgst = totalAmount * 0.09;
    const subTotal = totalAmount + cgst + sgst;

    // Calculate the discount as a percentage of the subTotal
    const discount = (subTotal * discountPercentage) / 100;

    const discountedTotal = Math.max(subTotal - discount, 0); // Ensuring
discount doesn't reduce total below zero
    const rounding = Math.round(discountedTotal) - discountedTotal;
    const finalAmount = discountedTotal + rounding;

    useEffect(() => {
        if (location.state && location.state.products) {
            setProducts(location.state.products);
        }
    }, [location.state]);

    const handleAddInvoiceItem = () => {
        if (!selectedProduct) {
            setError('Please select a product.');
            return;
        }

        const requestedQuantity = parseInt(quantity);

        if (requestedQuantity <= 0 || isNaN(requestedQuantity)
|| !Number.isInteger(requestedQuantity)) {
            setError('Quantity must be a positive whole number.');
            return;
        }

        const product = products.find(prod => prod.name ===
selectedProduct.value);
        if (!product) {
            setError('Selected product not found.');
            return;
        }

        const availableStock = parseInt(stockQuantities[product.name]);

        // Check for enough stock
        if (requestedQuantity > availableStock) {
            setError(`Only ${availableStock} units of ${product.name}
available in stock.`);
            return;
        }
        //  if quantity in table is in edit mode,. rise error
        if (editingQuantity) {
            setError(' Quantity in table is still in editmode...');
            return;
        }
        const existingItemIndex = invoiceItems.findIndex(item => item.name
=== product.name);

        if (existingItemIndex >= 0) {
            const existingItem = invoiceItems[existingItemIndex];
            const totalRequestedQuantity = existingItem.quantity +
requestedQuantity;
```

```
            if (totalRequestedQuantity > availableStock +
existingItem.quantity) {
                // Calculate the correct available stock by including the
quantity of the existing item
                setError(`Only ${availableStock + existingItem.quantity}
units of ${product.name} available in stock.`);
                return;
            }

            const updatedItem = {
                ...existingItem,
                quantity: totalRequestedQuantity,
                totalPrice: product.price * totalRequestedQuantity,
                cgst: product.price * totalRequestedQuantity * 0.09,
                sgst: product.price * totalRequestedQuantity * 0.09
            };

            const updatedInvoiceItems = [...invoiceItems];
            updatedInvoiceItems[existingItemIndex] = updatedItem;
            setInvoiceItems(updatedInvoiceItems);

            // Update the stock only by the newly added quantity
            setStockQuantities(prevStock => ({
                ...prevStock,
                [product.name]: availableStock - requestedQuantity
            }));
        } else {
            const newItem = {
                ...product,
                quantity: requestedQuantity,
                totalPrice: product.price * requestedQuantity,
                cgst: product.price * requestedQuantity * 0.09,
                sgst: product.price * requestedQuantity * 0.09
            };

            setInvoiceItems([...invoiceItems, newItem]);

            // Update stock for the new item added
            setStockQuantities(prevStock => ({
                ...prevStock,
                [product.name]: availableStock - requestedQuantity
            }));
            // Show the discount message if it's the first item added
            if (invoiceItems.length === 0) {
                setShowDiscountMessage(true);
            }
        }

        setSelectedProduct(null);
        setQuantity(1);
        setError('');
    };

    const handleRemoveInvoiceItem = (index) => {
        const item = invoiceItems[index];
        // Restore stock quantity
        setStockQuantities(prevStock => ({
            ...prevStock,
            [item.name]: parseInt(prevStock[item.name]) + item.quantity
        }));
```

```
        const newInvoiceItems = invoiceItems.filter((_, i) => i !== index);
        setInvoiceItems(newInvoiceItems);
    };

    const handleEditInvoiceItem = (index) => {
        const item = invoiceItems[index];
        setEditingIndex(index);
        setEditingQuantity(item.quantity);
    };

    const handleQuantityChange = (e) => {
        setEditingQuantity(e.target.value);
    };

    const handleSaveEdit = () => {
        if (!editingQuantity || isNaN(editingQuantity)
|| !Number.isInteger(Number(editingQuantity))) {
            setError('Quantity must be a positive whole number.');
            return;
        }

        const updatedQuantity = parseInt(editingQuantity);

        if (updatedQuantity <= 0) {
            setError('Quantity must be a positive number.');
            return;
        }

        const item = invoiceItems[editingIndex];
        const availableStock = parseInt(stockQuantities[item.name]) +
item.quantity;

        if (updatedQuantity > availableStock) {
            setError(`Only ${availableStock} units of ${item.name} available
in stock.`);
            return;
        }

        const updatedItems = invoiceItems.map((item, index) => {
            if (index === editingIndex) {
                return {
                    ...item,
                    quantity: updatedQuantity,
                    totalPrice: item.price * updatedQuantity,
                    cgst: item.price * updatedQuantity * 0.09,
                    sgst: item.price * updatedQuantity * 0.09
                };
            }
            return item;
        });

        // Update stock quantities
        setStockQuantities(prevStock => ({
            ...prevStock,
            [item.name]: availableStock - updatedQuantity
        }));

        setInvoiceItems(updatedItems);
        setEditingIndex(null);
        setEditingQuantity('');

        setError('');

    };



    const handleChange = (e) => {

        const { id, value } = e.target;

        setUserDetails(prevDetails => ({

            ...prevDetails,

            [id]: value

        }));

    };



    const productOptions = products.map(product => ({

        value: product.name,

        label: product.name

    }));



    const validateUserDetails = () => {

        if (!userDetails.name || !userDetails.address || !userDetails.email
|| !userDetails.mobile) {

            setError('All user details fields are required.');

            return false;

        }

        if (!/\S+@\S+\.\S+/.test(userDetails.email)) {

            setError('Invalid email format.');

            return false;

        }

        if (!/^\d{10}$/.test(userDetails.mobile)) {

            setError('Mobile number must be 10 digits.');

            return false;
        }
        return true;
    };

    const handleGeneratePDF = () => {
        validateUserDetails();
        if (!validateUserDetails() || invoiceItems.length === 0) {
            alert('Please add items and fill in user details');
            return;  // Ensure the function exits here if validation fails
        }
        setError('');  // Clear any previous errors
        console.log('PDF generated:', {
            ...userDetails,
            invoiceItems,
            totalAmount,
            cgst,
            sgst,
            subTotal,
            discount,
            discountedTotal,
            rounding,
            finalAmount
        });
        // navigate to preview page so that user can download invoice from
there even if changes are required,. they can come back,...
        alert('please have a preview of your invoice.');  // Notify the user
on success
        navigate('/generate_invoice', {
            state: {
                ...userDetails,
                invoiceItems,

                totalAmount,

                cgst,

                sgst,

                subTotal,

                discount,

                discountedTotal,

                rounding,

                finalAmount

            }

        });

    };



    return (

        <>

            {showDiscountMessage && (

                <MessageBox

                    message={`Congratulations! Applied ${discountPercentage}%
discount on your purchase.`}

                />

            )}



            <div className="invoice-page-container">

                <div className='userdetails-form-container'>

                    <div className="userdetails-form-group">

                        <label className="userdetails-form-label"
htmlFor='mobile'>Mobile:</label>

                        <input className="userdetails-form-input"
type='text' id='mobile' value={userDetails.mobile} onChange={handleChange}
placeholder='Enter your mobile' />

                    </div>
                    <div className="userdetails-form-group">

                        <label className="userdetails-form-label"
htmlFor='name'>Name:</label>

                        <input className="userdetails-form-input"
type='text' id='name' value={userDetails.name} onChange={handleChange}
placeholder='Enter your name' />

                    </div>

                    <div className="userdetails-form-group">

                        <label className="userdetails-form-label"
htmlFor='email'>Email:</label>

                        <input className="userdetails-form-input"
type='email' id='email' value={userDetails.email} onChange={handleChange}
placeholder='Enter your email' />

                    </div>

                    <div className="userdetails-form-group">

                        <label className="userdetails-form-label"
htmlFor='address'>Address:</label>

                        <input className="userdetails-form-input"
type='text' id='address' value={userDetails.address} onChange={handleChange}
placeholder='Enter your address' />

                    </div>

                </div>



                <div className="invoice-summary">

                    <div className="invoice-summary-left">

                        <div className="invoice-item">

                            CGST (9%): {rs} {cgst.toFixed(2)}

                        </div>

                        <div className="invoice-item">

                            SGST (9%): {rs} {sgst.toFixed(2)}

                        </div>

                        <div className="invoice-item">

                            Rounding: {rs} {rounding.toFixed(2)}

                        </div>
```

```
        <div className="invoice-item">
            SubTotal(after Tax) : {rs} {subTotal.toFixed(2)}
        </div>
    </div>
    <div className="invoice-summary-right">
        <div className="invoice-item discount">
            <label>
                Discount:
            </label>
            {rs} {discount.toFixed(2)}
        </div>
        <div className="invoice-item final-amount">
            <label>
                Final Amount:
            </label>
            {rs} {finalAmount.toFixed(2)}
        </div>
    </div>
</div>

<div className="invoice-form-container">
    <div className="invoice-select-container">
        <label htmlFor="product">Product:</label>
        <Select
            id="product"
            options={productOptions}
            value={selectedProduct}
            onChange={setSelectedProduct}
            placeholder="Select a product"
```

```
                        />

                    </div>

                    <div className="invoice-form-group">

                        <label className="invoice-label"
htmlFor='quantity'>Quantity:</label>

                        <input className="invoice-input" type='number'
min={1} id='quantity' value={quantity} onChange={(e) =>
setQuantity(e.target.value)} placeholder='Enter quantity' />

                    </div>

                    {error && <div className="invoice-error">{error}</div>}

                    <button className="invoice-button"
onClick={handleAddInvoiceItem}>Add to Invoice</button>

                </div>

                <div className="invoice-buttons">

                    <button className="invoice-generate-button"
onClick={handleGeneratePDF}> Generate PDF (Preview) </button>

                </div>



                <div className="invoice-content-container">

                    <div className="invoice-content">

                        <div className="invoice-table-container">

                            <table className="invoice-table">

                                <thead>

                                    <tr>

                                        <th className="invoice-
th">Product</th>

                                         <th className="invoice-th">Price</th>

                                        <th className="invoice-
th">Quantity</th>

                                        <th className='invoice-th'>CGST</th>

                                        <th className='invoice-th'>SGST</th>

                                         <th className="invoice-th">Total</th>

                                        <th className="invoice-
th">Actions</th>
```

```
                                    </tr>
                                </thead>
                                <tbody>
                                    {invoiceItems.map((item, index) => (
                                        <tr key={index}>
                                            <td className="invoice-
td">{item.name}</td>
                                            <td className="invoice-
td">{rs}{item.price.toFixed(2)}</td>
                                            <td className="invoice-td">
                                                {editingIndex === index ? (
                                                    <input
                                                        type="number"
                                                        value={editingQuanti
ty}
                                                        onChange={handleQuan
tityChange}
                                                    />
                                                ) : (
                                                    item.quantity
                                                )}
                                            </td>
                                            <td className="invoice-
td">{rs}{item.cgst.toFixed(2)}</td>
                                            <td className="invoice-
td">{rs}{item.sgst.toFixed(2)}</td>
                                            <td className="invoice-
td">{rs}{item.totalPrice.toFixed(2)}</td>
                                            <td className="invoice-td">
                                                <div className="invoice-
button-container">
                                                     {editingIndex === index ?
(
                                                        // data-
testid="save-button"  is for testing
```

```
                                                        <button data-
testid="save-button" className="invoice-table-editbutton"
onClick={handleSaveEdit}>Save</button>

                                                    ) : (

                                                        <button
className="invoice-table-editbutton" onClick={() =>
handleEditInvoiceItem(index)}>Edit</button>

                                                    )}

                                                    <button
className="invoice-table-removebutton" onClick={() =>
handleRemoveInvoiceItem(index)}>Remove</button>

                                                </div>

                                            </td>



                                        </tr>

                                    ))}

                                </tbody>

                                <div className="invoice-total-amount">

                                    Total Amount:
{rs}{totalAmount.toFixed(2)}

                                </div>

                            </table>

                        </div>

                    </div>

                </div>

            </div>

        </>

    );

}
```

invoice.css

```
/* Invoice Page Container */

.invoice-page-container {

  display: flex;
  flex-direction: column;

  justify-content: center;

  height: 100vh;

  background-image: linear-gradient(109.6deg, rgba(254, 253, 205, 1) 11.2%,
rgba(163, 230, 255, 1) 91.1%);

  padding: 20px;

  box-sizing: border-box;

  overflow-y: auto;

  scrollbar-width: none;

  /* these lines donot allow page to move at any cost,.. */

  position: fixed;

  /* Fix the container to the screen */

  top: 0;

  /* Align to the top of the viewport */

  left: 0;

  /* Align to the left of the viewport */

  width: 100%;

  /* Ensure it covers the full width */

}



/* --------------------------  User Details Form Container  ----------------
---------- */

.userdetails-form-container {

  position: absolute;

  top: 5px;

  left: 5px;

  display: flex;

  flex-direction: column;

  width: 365px;

  padding: 10px;
  background-color: #fff;

  border-radius: 8px;

  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

}


/* User Details Form Group */

.userdetails-form-group {

  display: flex;

  flex-direction: column;

  align-items: flex-start;

  width: 100%;

  margin-bottom: 5px;

}


/* User Details Form Labels */

.userdetails-form-label {

  display: flex;

  align-items: center;

  justify-content: flex-start;

  margin-left: calc(50% - 150px);

  font-size: 1rem;

  font-weight: bold;

  color: #333;

  font-family: "Segoe UI", system-ui, sans-serif;

  width: 80%;

}


/* User Details Form Inputs */

.userdetails-form-input {
  font-size: 1rem;
  width: 80%;
  max-width: 400px;
  padding: 10px;
  border: 1px solid #ddd;
  border-radius: 20px;
  background-color: #fafafa;
  color: #333;
  transition: border-color 0.3s ease, box-shadow 0.3s ease;
  margin: 0 auto;
}


/* Input Placeholder */
.userdetails-form-input::placeholder {
  color: #aaa;
}


/* Input Focus */
.userdetails-form-input:focus {
  outline: none;
  border-color: #007bff;
  box-shadow: 0 0 0 2px rgba(38, 143, 255, 0.25);
}


/* ----------------------- Invoice Form Container --------------------------
--*/
.invoice-form-container {
  position: absolute;
  bottom: 60px;
  left: 0px;
  display: flex;

  flex-direction: column;

  align-items: center;

  width: 350px;

  padding: 20px;

  background-color: #fff;

  border-radius: 8px;

  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

  margin: 0 auto;

}


/* Invoice Title */

.invoice-component-container-h1 {

  color: #282c34;

  font-size: 1.8rem;

  margin-bottom: 20px;

  text-align: center;

}


/* Form Group */

.invoice-form-group {

  display: flex;

  flex-direction: column;

  align-items: flex-start;

  width: 100%;

}


/* Form Labels */

.invoice-label {
  display: flex;

  align-items: center;

  justify-content: flex-start;

  margin-left: calc(50% - 175px);

  font-size: 1rem;

  color: #333;

  margin-bottom: 8px;

  font-family: "Segoe UI", system-ui, sans-serif;

  font-weight: 500;

  width: 80%;

  max-width: 400px;

}


/* Form Inputs */

.invoice-input {

  font-size: 1rem;

  width: 80%;

  max-width: 400px;

  padding: 10px;

  border: 1px solid #ddd;

  border-radius: 20px;

  background-color: #fafafa;

  color: #333;

  transition: border-color 0.3s ease, box-shadow 0.3s ease;

  margin: 0 auto;

}


/* Input Placeholder */

.invoice-input::placeholder {
  color: #aaa;
}

/* Input Focus */
.invoice-input:focus {
  outline: none;
  border-color: #007bff;
  box-shadow: 0 0 0 2px rgba(38, 143, 255, 0.25);
}

/* Select Container */
.invoice-select-container {
  width: 100%;
  max-width: 400px;
  margin: 0 auto;
  margin-bottom: 15px;
}

/* ------------------- invoice summary  -------------------*/
.invoice-summary {
  display: flex;
  justify-content: space-between;
  width: 66%;
  height: 110px;
  position: absolute;
  top: 0px;
  right: 0px;
  padding: 20px;
  border: 1px solid #ddd;
  border-radius: 10px;

  background-color: #f9f9f9;

  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

}


.invoice-summary-left {

  display: flex;

  flex-direction: column;

}


.invoice-summary-right {

  display: flex;

  flex-direction: column;

  align-items: flex-end;

  gap: 10px;

}


.invoice-item {

  font-size: 1rem;

  font-weight: bold;

  margin: 5px 0;

}


.invoice-item.discount {

  margin-top: 5%;

  color: #dc3545;

  font-size: 2rem;

  font-weight: bold;

}
```

```
.invoice-item.discount label {

  font-size: 20px;

  margin-right: 10px;

}


.invoice-item.final-amount {

  color: #28a745;

  font-size: 2rem;

  font-weight: bold;

  position: relative;

}


.invoice-item.final-amount label {

  font-size: 20px;

  margin-right: 10px;

}


/* ---------------- Error Messages  ----------------*/

.invoice-error {

  color: red;

  margin-top: 10px;

  font-size: 0.875rem;

}


/* ---------------  Buttons  ----------------- */

/* Generate PDF Button */

.invoice-generate-button {

  position: absolute;
  bottom: 0;

  left: 5%;

  color: #fff;

  padding: 10px 20px;

  border: none;

  border-radius: 20px;

  cursor: pointer;

  font-size: 1rem;

  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

  margin: 10px;

  /* "to left" / "to right" - affects initial color */

  background: linear-gradient(to left, rgb(0, 49, 117) 50%, rgb(0, 111, 11)
50%) right;

  background-size: 200%;

  transition: .3s ease-out;

}



.invoice-generate-button:hover {

  background-position: left;

  transform: scale(1.05);   /* Slightly enlarge the button on hover */

}



/* -----------------------  Table Container  ----------------------- */

.invoice-content {

  width: 67.5%;

  height: 420px;

  /* Fixed height */

  background-color: #fff;

  /* White background for table container */

  padding: 10px;
  border-radius: 8px;

  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

  overflow: hidden;

  /* Hide any overflowing content */

  position: absolute;

  /* Position relative to the parent */

  bottom: 5px;

  /* Align to the bottom */

  right: 0;

  /* Align to the right */

  -ms-overflow-style: none;

  /* IE and Edge */

  scrollbar-width: none;

  /* Firefox */

}


/* Hide scrollbars but keep scroll functionality */

.invoice-content::-webkit-scrollbar {

  display: none;

  /* Chrome, Safari, and Opera */

}


/* Invoice Table Container */

.invoice-table-container {

  width: 100%;

  overflow-y: auto;

  /* Enable vertical scrolling for the table body */

  height: 100%;

  /* Take full height of the parent */
  position: relative;

  /* Required for the table header to be fixed */

}


.invoice-table {

  width: 100%;

  border-collapse: collapse;

  margin-top: 20px;

  position: relative;

}


.invoice-th {

  padding: 12px;

  border: 1px solid #ddd;

  text-align: center;

  position: sticky;

  /* Fix the header */

  top: 0;

  /* Position the header at the top */

  z-index: 2;

  /* Ensure the header stays above other content */

  background-color: aquamarine;

}


.invoice-td {

  padding: 12px;

  border: 1px solid #ddd;

  text-align: left;

  background-color: #fff;
  color: #333;
}

/* Container for buttons in table cell */
.invoice-button-container {
  display: flex;
  justify-content: center;
  align-items: center;
  gap: 10px;
  padding: 0 10px;
}

/* Remove Button in Table */
.invoice-table-removebutton {
  background-color: #dc3545;
  color: #fff;
  padding: 5px 10px;
  border: none;
  border-radius: 20px;
  cursor: pointer;
  font-size: 0.875rem;
  transition: background-color 0.3s ease;
}

.invoice-table-removebutton:hover {
  background-color: #bf081a;
}

/* Edit Button in Table */
```

```
.invoice-table-editbutton {

  background-color: #007bff;

  color: #fff;

  padding: 5px 25px;

  border: none;

  border-radius: 20px;

  cursor: pointer;

  font-size: 0.875rem;

  transition: background-color 0.3s ease;

}


.invoice-table-editbutton:hover {

  background-color: #0069d9;

}


/* Total Amount */

.invoice-total-amount {

  font-size: 1.5rem;

  font-weight: bold;

  color: #333;

  margin-top: 20px;

  text-align: center;

  margin-bottom: 20px;

}
```

generate_invoice.js

```
import React, { useRef, useState } from "react";

import '../styles/generate_invoice.css';

import profilePic from '../assests/profilepic.png';
import { useLocation } from 'react-router-dom';

import jsPDF from 'jspdf';

import html2canvas from 'html2canvas';



const GenerateInvoice = () => {

    const location = useLocation();

    const { name, address, email, mobile, invoiceItems, totalAmount, cgst,
sgst, finalAmount } = location.state;

    const currentDate = new Date();

    const invoiceRef = useRef();

    const [paperSize, setPaperSize] = useState('A4'); // State to store the
selected paper size



    const downloadInvoice = () => {

        const invoiceElement = invoiceRef.current;

        html2canvas(invoiceElement).then((canvas) => {

            let pdf;

            if (paperSize === 'A4') {

                pdf = new jsPDF('p', 'mm', 'a4');

                const imgWidth = 210;

                const imgHeight = (canvas.height * imgWidth) / canvas.width;

                pdf.addImage(canvas.toDataURL('image/png'), 'PNG', 0, 0,
imgWidth, imgHeight);

            } else if (paperSize === 'Thermal') {

                pdf = new jsPDF('p', 'mm', [80, 200]); // Thermal paper size

                const imgWidth = 80;

                const imgHeight = (canvas.height * imgWidth) / canvas.width;

                pdf.addImage(canvas.toDataURL('image/png'), 'PNG', 0, 0,
imgWidth, imgHeight);

            }

            if (paperSize === 'A4') {
                pdf.save('A4_invoice.pdf');

            } else if (paperSize === 'Thermal') {

                pdf.save('thermal_invoice.pdf');

            }

        });

    };


    return (

        <>

            {/*<Link to="/invoice" className="back-button">&larr;</Link> */}

            <div className="invoice-controls-container">

                <div className="paper-size-selector">

                    <label htmlFor="paper-size">Select Paper Size: </label>

                    <select

                        id="paper-size"

                        value={paperSize}

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

                    >

                        <option value="A4">A4</option>

                        <option value="Thermal">Thermal</option>

                    </select>

                </div>

                <div className="generate-invoice-button">

                    <button onClick={() =>
downloadInvoice(paperSize)}>Download Invoice</button>

                </div>

            </div>

            <div className="invoice-container" ref={invoiceRef}>

                {/* Invoice content goes here */}

                <div className="invoice-header">
                    <div className="company-logo">

                        <img src={profilePic} alt="Company Logo" />

                    </div>

                    <div className="invoice-title">

                        <h1>INVOICE</h1>

                    </div>

                </div>


                <div className="invoice-info">

                    <div className="invoice-to">

                        <strong>INVOICE TO:</strong>

                        <p>{name}</p>

                        <p>{address}</p>

                        <p>{email}</p>

                        <p>{mobile}</p>

                    </div>

                    <div className="invoice-details">

                        <p>Invoice No. 1234</p>

                        <p>{currentDate.toLocaleString()}</p>

                    </div>

                </div>


                <div className="invoice-generation-table-container">

                    <table className="invoice-generation-table">

                        <thead>

                            <tr>

                                <th>NO</th>

                                <th>PRODUCT DESCRIPTION</th>

                                <th>PRICE</th>
```

```
                                <th>QTY</th>

                                <th>TOTAL</th>

                            </tr>

                        </thead>

                        <tbody>

                            {invoiceItems.map((item, index) => (

                                <tr key={index}>

                                    <td>{index + 1}</td>

                                    <td>{item.name}</td>

                                    <td>{item.price}</td>

                                    <td>{item.quantity}</td>

                                    <td>{item.totalPrice}</td>

                                </tr>

                            ))}

                        </tbody>

                    </table>



                    <div className="invoice-payment">

                        <div className="payment-details">

                            <strong>Payment Details:</strong>

                            <p>Account No: 123 456 7890</p>

                            <p>Account Name: CSTS</p>

                        </div>

                        <div className="totals">

                            <p><strong>Subtotal:</strong>
{totalAmount.toFixed(2)}</p>

                            <p><strong>CGST:</strong> {cgst.toFixed(2)}</p>

                            <p><strong>SGST:</strong> {sgst.toFixed(2)}</p>

                            <p><strong>Total:</strong>
{finalAmount.toFixed(2)}</p>
```

```
                        </div>

                    </div>

                </div>



                <div className="invoice-footer">

                    <p className="thank-you">Thank You.</p>

                    <div className="company-details">

                        <p>123 our Business Address, City, Country</p>

                        <p>Compsci Technology Solutions</p>

                        <p>+123-456-7890</p>

                        <p>csts.co.in</p>

                    </div>

                </div>

                <div className="invoice-terms">

                    <p><strong>Tax ID:</strong> 123-456-789</p>

                    <p>Refunds are subject to our refund policy as outlined
on our website.</p>

                    <p>Payment due within 30 days of invoice date. Late
payments may incur a 2% fee per month.</p>

                </div>

            </div>

        </>

    );

};



export default GenerateInvoice;
```

generate_invoice.css

```
.invoice-container {

    max-width: 800px;

    margin: 20px auto; /* Added margin to create space around the invoice */
    background-color: #f7f4ed;

    padding: 30px; /* Increased padding to create space inside the container
*/

    font-family: 'Times New Roman', serif;

    color: #4b2e0f;

    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

/* -------------------------            Invoice css     --------------------
-- */

.invoice-header {

    display: flex;

    justify-content: space-between;

    align-items: center;

}


.company-logo img {

    width: 150px;

    margin-left: 10%;

}


.invoice-title h1 {

    font-size: 4rem;

    color: #4b2e0f;

    text-decoration: none; /* Remove the default underline */

    position: relative;

}


.invoice-title h1::after {

    content: "";

    display: block;
    position: absolute;

    left: 0;

    bottom: -5px; /* Adjust based on font size */

    width: 100%;

    border-bottom: 3px solid #4b2e0f; /* Adjust thickness */

    border-bottom: 3px double #4b2e0f; /* Create a double underline effect
*/

}



.invoice-info {

    display: flex;

    justify-content: space-between;

    align-items: center;

    margin-left: 30px;

}



.invoice-to,

.invoice-details {

    width: 45%;

    font-size: 16px;

}



/* -------------------------            Table css     ----------------------
*/

.invoice-generation-table-container {

    width: 95%;  /* Reduced width to give extra space around the table */

    margin: 30px auto; /* Added margin to create space above and below the
table */

    background-color: #ffffff;

    padding: 20px;

    font-family: 'Times New Roman', serif;
    color: #4b2e0f;

    border-radius: 10px; /* Added slight rounding for a polished look */

}


.invoice-generation-table {

    width: 100%;

    border-collapse: collapse;

    margin-bottom: 30px;

}


.invoice-generation-table th,

.invoice-generation-table td {

    border: 1px solid #4b2e0f;

    padding: 15px; /* Increased padding for a more spacious table layout */

    text-align: left;

}


.invoice-generation-table th {

    background-color: #f7f4ed;

    color: #4b2e0f;

    font-weight: bold;

}

/* -------------------------            Footer css     ---------------------
- */

.invoice-payment {

    display: flex;

    justify-content: space-between;

    margin-top: 30px; /* Increased margin to create more space above the
payment section */

    background-color: #ffffff;
    padding: 30px; /* Increased padding for a roomier layout */
    border: 1px solid #4b2e0f;
    border-radius: 10px; /* Rounded corners to match the table container */
}

.payment-details,
.totals {
    width: 45%;
}

.payment-details p,
.totals p {
    margin: 10px 0; /* Added spacing between payment details lines */
}
.invoice-footer {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-top: 50px;
    padding-top: 20px;
    border-top: 1px solid #4b2e0f; /* Added top border for separation */
}

.thank-you {
    text-align: left;
    font-size: 50px;
    color: #4b2e0f;
    font-style: italic;
    font-weight: bold;
    margin-bottom: 0;

}



.company-details {

    text-align: right;

    font-size: 16px;

}



.invoice-terms {

    margin-top: 30px;

    text-align: center;

    font-size: 12px;

    color: #4b2e0f;

}

/* ------------------------- New Styles ---------------------- */

/* Add spacing and alignment to form controls within the invoice component
*/

.invoice-controls-container {

    margin: 20px auto;

    padding: 20px;

    border: 1px solid #4b2e0f;

    border-radius: 8px;

    background-color: #ffffff;

    box-shadow: 0px 2px 8px rgba(0, 0, 0, 0.1);

    max-width: 820px;

    position: sticky;

    top: 0; /* Sticks to the top of the viewport */

    z-index: 10; /* Ensure it stays on top of other elements */

}
```

```
/* Add margin-bottom to ensure content below isn't covered when sticky */
.invoice-controls-container + .invoice-container {
    margin-top: 20px;
}

.paper-size-selector {
    margin-bottom: 15px;
}

/* Style the legend of the fieldset within the invoice component */
.invoice-controls-container legend {
    font-size: 1.5rem;
    color: #4b2e0f;
    padding: 0 10px;
    font-weight: bold;
}

/* Adjust margins for mobile responsiveness */
@media (max-width: 768px) {
    .invoice-container {
        padding: 20px;
    }

    .company-logo img {
        width: 100px;
    }

    .thank-you {
        font-size: 40px;
    }


    .invoice-generation-table th,
    .invoice-generation-table td {
        padding: 10px;
    }
}
/* ------------------------- New Styles ---------------------- */
/* Add spacing and alignment to form controls within the invoice component
*/
.invoice-controls-container {
    display: flex;
    flex-direction: row;
    justify-content: space-between;
    margin: 20px auto;
    padding: 20px;
    border: 1px solid #4b2e0f;
    border-radius: 8px;
    background-color: #ffffff;
    box-shadow: 0px 2px 8px rgba(0, 0, 0, 0.1);
    max-width: 820px;
    position: sticky;
    top: 0; /* Sticks to the top of the viewport */
    z-index: 10; /* Ensure it stays on top of other elements */
}


/* Add margin-bottom to ensure content below isn't covered when sticky */
.invoice-controls-container + .invoice-container {
    margin-top: 20px;
}
```

```
/* Style the legend of the fieldset within the invoice component */

.invoice-controls-container legend {

    font-size: 1.5rem;

    color: #4b2e0f;

    padding: 0 10px;

    font-weight: bold;

}


/* Add margin-bottom to the paper size selector */

.paper-size-selector {

    margin-bottom: 15px;

}


/* Style for the paper size dropdown */

.paper-size-selector select {

    padding: 8px;

    border: 1px solid #4b2e0f;

    border-radius: 4px;

    font-size: 16px;

    background-color: #f9f9f9;

}


/* Style for the generate invoice button */

.generate-invoice-button button {

    padding: 12px 20px;

    background-color: #4b2e0f;

    color: #ffffff;

    border: none;
    border-radius: 4px;

    font-size: 16px;

    cursor: pointer;

    transition: background-color 0.3s ease, transform 0.2s ease;

}


.generate-invoice-button button:hover {

    background-color: #3a1d0b;

    transform: scale(1.05);

}


.generate-invoice-button button:focus {

    outline: none;

    box-shadow: 0 0 0 2px rgba(75, 46, 15, 0.5);

}


/* Adjust margins for mobile responsiveness */

@media (max-width: 768px) {

    .invoice-container {

        padding: 20px;

    }


    .company-logo img {

        width: 100px;

    }


    .thank-you {

        font-size: 40px;

    }
    .invoice-generation-table th,

    .invoice-generation-table td {

        padding: 10px;

    }

}
```

messagebox.js

```
import React, { useState } from 'react';

import '../components/messagebox.css';



export default function MessageBox({ message }) {

    const [showMessage, setShowMessage] = useState(true);



    const handleClose = () => {

        setShowMessage(false); // Hides the MessageBox

    };



    return (

        <>

            {showMessage && (

                <div className="overlay">

                    <div className="message-box">

                        <button className="close-button"
onClick={handleClose}>X</button>

                        <p>{message}</p>

                    </div>

                </div>

            )}
        </>
    );
}
```

messagebox.css

```
/* Overlay to center the message box */
.overlay {
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    display: flex;
    justify-content: center;
    align-items: center;
    background-color: rgba(0, 0, 0, 0.5); /* semi-transparent background */
    z-index: 20; /* keep it on top */
}

/* Message box styling */
.message-box {
    width: 400px;
    padding: 20px;
    background-color: #fff; /* white background */
    border-radius: 10px;
    box-shadow: 0 8px 16px rgba(0, 0, 0, 0.3); /* cool shadow effect */
    text-align: center;
    position: relative;
    animation: slideDown 0.3s ease-in-out; /* smooth slide-down animation */
```

```
}


/* Text styling */

.message-box p {

    font-size: 20px;

    color: #333;

    margin: 0;

    padding: 20px 0;

}


/* Close button styling */

.close-button {

    position: absolute;

    top: 10px;

    right: 10px;

    background-color: transparent;

    border: none;

    font-size: 18px;

    font-weight: bold;

    cursor: pointer;

    color: #333;

}


.close-button:hover {

    color: #ff0000; /* color change on hover */

}


/* Slide down animation */

@keyframes slideDown {
    from {
        transform: translateY(-50px);
        opacity: 0;
    }
    to {
        transform: translateY(0);
        opacity: 1;
    }
}
```

14. Testing Approach

In the invoice generation project, I implemented a structured testing approach focusing primarily on UI testing with React Testing Library. Here’s a summary of the testing strategy:

UI Testing with React Testing Library:

1. **Setup:**

   ```
   npm install --save-dev @testing-library/react @testing-library/jest-dom
   ```

2. **Testing Strategy:**

   1. **Component Rendering:** Verified that each React component renders correctly based on different props and states. This included checking if components like the invoice form, preview, and results pages appear as expected.
   2. **User Interactions:** Tested user interactions such as form submissions, button clicks, and input changes. Ensured that these interactions trigger the expected updates and behaviors within the application.
   3. **Form Validation:** Validated form inputs for generating invoices to ensure that all required fields are properly validated. This included checking error messages and validation states.
   4. **Integration Testing:** Tested interactions between components, such as submitting a form and rendering the invoice preview. Ensured that data flows correctly through the components and the state updates as intended.

3. Testing Process:

Create Test Cases: Defined test cases for each key feature of the UI, such as rendering components, handling user interactions, and validating form data.

```
npm test
```

Review Results: Analyzed test results to identify and fix issues. Ensured that all tests passed and that the application met the expected quality standards.

By focusing on UI testing with React Testing Library, I ensured that the invoice generation application's frontend was functional, user-friendly, and free of critical bugs.

15.Testing :

Manual Testing Approach for Invoice Generation Project

1. Responsiveness and Effectiveness of the UI

Responsive Design:

- **Objective:** Ensure UI elements adjust correctly to different screen sizes (desktops, tablets, mobile devices).
- **Method:** Use browser developer tools to simulate various screen resolutions and manually test UI components, such as the invoice form and preview screens.

User-Friendly Interface:

- **Objective:** Ensure the interface is intuitive and easy to navigate.
- **Method:** Minimize the number of clicks required to generate invoices. Ensure all form elements, input fields, and buttons are accessible and easy to understand.

2. Testing Each Component and Page

Component Functionality:

- **Objective:** Verify that each React component behaves as expected, especially for the invoice form, preview, and submission components.
- **Method:** Check props, state management, and dynamic content rendering, such as updating the total amount when the quantity changes.

Alternative Solutions:

- **Objective:** Search for efficient ways to achieve functionality with fewer lines of code.

- **Method:** Continuously iterate on the codebase to optimize performance, particularly in handling form submissions and data management.

3. Single Page Reload and React Features

Single Page Reload:

- **Objective:** Ensure smooth navigation between different routes (e.g., from the dashboard to the invoice generation page) without full-page reloads.
- **Method:** Use React DevTools to inspect the component tree and test navigation flow, ensuring state persistence during transitions.

Component Updates:

- **Objective:** Ensure components, such as the invoice preview, update correctly without unnecessary re-renders.
- **Method:** Use React DevTools to monitor component updates during user interactions, such as form edits.

4. Login and Signup Pages

Form Validation:

- **Objective:** Test form inputs for edge cases, such as empty fields, invalid email formats, or incorrect password lengths.
- **Method:** Ensure appropriate error messages are displayed and validation rules are enforced.

User Authentication:

- **Objective:** Verify that login and signup processes work correctly, including successful and failed authentication attempts.
- **Method:** Test scenarios for valid and invalid user credentials, ensuring proper session management.

5. Overall Testing Approach

- **Continuous Testing:** Begin testing from the initial development stages and continue throughout the project. Ensure each new feature or component is thoroughly tested upon implementation.
- **Real-World Scenarios:** Simulate real-world user actions, such as filling out forms, adjusting item quantities, generating previews, and saving or printing invoices.
- **Iterative Improvement:** Continuously iterate on design and functionality based on test results, optimizing the user experience and resolving any issues.

16.Test Plan

1. Introduction

- Objective: Ensure the React application for invoice generation meets specified requirements and functions correctly.
- Scope: Covers functional and non-functional aspects, including component rendering, state management, props handling, user interactions, API calls and routing.

2. Test Items

- Components: Invoice form, preview, result page, etc.
- State management: Handling invoice data, user authentication.
- Props passing: Data flow between form elements and invoice preview.
- User interactions: Form submissions, input changes, button clicks.
- API calls and data handling: Fetching and storing inventory data.
- Routing and navigation: Navigating between dashboard and invoice pages.

3. Test Approach

- Unit Testing: Test individual components, such as form inputs and preview sections, using Jest and React Testing Library.
- Integration Testing: Test the interaction between components, especially the flow from form submission to invoice preview.
- End-to-End (E2E) Testing: Test the entire application flow using tools like Cypress or Selenium, from login to invoice generation and saving/printing.

4.. Pass/Fail Criteria

- Pass Criteria: All test cases must pass without errors.
- Fail Criteria: Any critical issues must be resolved before release.

17.Test Cases

// Home.test.js

```
import React from "react";

import { BrowserRouter } from "react-router-dom";

import { render, screen, fireEvent} from '@testing-library/react';

import '@testing-library/jest-dom';

import Home from "../templates/home";
```

```
test('renders home component without crashing', ()=> {
    render(
        <BrowserRouter>
            <Home/>
        </BrowserRouter>
    );
});


test('renders logo, navigation links, and content on pageload', ()=>
{
    render(
        <BrowserRouter>
            <Home/>
        </BrowserRouter>
    );


    // Check for logo image
    expect(screen.getByRole('img', { name: /programming stuff
logo/i })).toBeInTheDocument();


    // Check for heading
    expect(screen.getByRole('heading', { name: /compsci
technologies/i })).toBeInTheDocument();


    // Check navigation links
    expect(screen.getByRole('link', { name:
/home/i })).toHaveAttribute('href', '/');
    expect(screen.getByRole('link', { name:
/dashboard/i })).toHaveAttribute('href', '/dashboard');
```

```
    expect(screen.getByRole('link', { name:
/invoice/i })).toHaveAttribute('href', '/invoice');



    // Check for Login and Register links

    expect(screen.getByRole('link', { name:
/login/i })).toHaveAttribute('href', '/login');

    expect(screen.getByRole('link', { name:
/register/i })).toHaveAttribute('href', '/register');



    // Check content paragraphs

    expect(screen.getByText(/welcome to invoice
generator!/i)).toBeInTheDocument();

    expect(screen.getByText(/our tool helps you create professional
invoices with ease./i)).toBeInTheDocument();

});



test('renders login and register buttons on pageload', ()=> {

    render(

        <BrowserRouter>

            <Home/>

        </BrowserRouter>

    );



    // Check for Login and Register links

    expect(screen.getByRole('link', { name:
/login/i })).toHaveAttribute('href', '/login');

    expect(screen.getByRole('link', { name:
/register/i })).toHaveAttribute('href', '/register');

    // fire them and checking results now,..

    fireEvent.click(screen.getByRole('link', { name: /login/i }));

    fireEvent.click(screen.getByRole('link', { name: /register/i }));
```

```
});

    // Login.test.js

import Login from "../templates/login";

import React from "react";

import { BrowserRouter } from "react-router-dom";

import { render, screen, fireEvent } from '@testing-library/react';

import '@testing-library/jest-dom';



describe('Login Component', () => {

    test('renders login component without crashing', () => {

        render(

            <BrowserRouter>

                <Login />

            </BrowserRouter>

        );

        // Check heading

        expect(screen.getByRole('heading', { name:
/login/i })).toBeInTheDocument();

        // Check username label and textfield


expect(screen.getByLabelText(/username/i)).toBeInTheDocument();

        expect(screen.getByTestId('username')).toBeInTheDocument();

        // Check password label and password field


expect(screen.getByLabelText(/password/i)).toBeInTheDocument();

        expect(screen.getByTestId('password')).toBeInTheDocument();

        // Check login button

        expect(screen.getByRole('button', { name:
/login/i })).toBeInTheDocument();

        // Check signup link
        expect(screen.getByText(/don't have an
account?/i)).toBeInTheDocument();

        expect(screen.getByRole('link', { name: /sign
up/i })).toHaveAttribute('href', '/register');

    });



    test('shows error message when form is submitted with empty
fields', () => {

        render(

            <BrowserRouter>

                <Login />

            </BrowserRouter>

        );

        // Submit form with empty fields

        fireEvent.click(screen.getByRole('button', { name:
/login/i }));

        // Check for error message

        expect(screen.getByTestId('error-
message')).toHaveTextContent('Both fields are required');

    });



    test('submits the form when fields are filled', () => {

        render(

            <BrowserRouter>

                <Login />

            </BrowserRouter>

        );

        // Fill in username and password

        fireEvent.change(screen.getByTestId('username'), { target:
{ value: 'testuser' } });
```

```
        fireEvent.change(screen.getByTestId('password'), { target:
{ value: 'password123' } });
        // Submit form
        fireEvent.click(screen.getByRole('button', { name:
/login/i }));
        // Check that error message is not displayed
        expect(screen.queryByTestId('error-message')).toBeNull();
    });
});
```

// Register.test.js

```
import Register from "../templates/register";
import React from "react";
import { BrowserRouter } from "react-router-dom";
import { render, screen, fireEvent } from '@testing-library/react';
import '@testing-library/jest-dom';

describe('Register Component', () => {
    test('renders register component without crashing', () => {
        render(
            <BrowserRouter>
                <Register />
            </BrowserRouter>
        );
        // Check heading
        expect(screen.getByRole('heading', { name:
/register/i })).toBeInTheDocument();
        // Check input fields
        expect(screen.getByTestId('username')).toBeInTheDocument();
        expect(screen.getByTestId('password')).toBeInTheDocument();
        expect(screen.getByTestId('company-
name')).toBeInTheDocument();

        expect(screen.getByTestId('email')).toBeInTheDocument();

        expect(screen.getByTestId('address')).toBeInTheDocument();

        expect(screen.getByTestId('gstin')).toBeInTheDocument();

        expect(screen.getByTestId('pan')).toBeInTheDocument();

        // Check submit button

        expect(screen.getByRole('button', { name:
/register/i })).toBeInTheDocument();

        // Check login link

        expect(screen.getByText(/already have an
account\?/i)).toBeInTheDocument();

        expect(screen.getByRole('link', { name: /log
in/i })).toHaveAttribute('href', '/login');

    });



    test('shows error message when form is submitted with empty
fields', () => {

        render(

            <BrowserRouter>

                <Register />

            </BrowserRouter>

        );

        // Submit form with empty fields

        fireEvent.click(screen.getByRole('button', { name:
/register/i }));

        // Check for error alert

        expect(screen.getByRole('alert')).toHaveTextContent('Please
complete all required fields.');

    });
    test('submits the form when all fields are filled', () => {

        render(

            <BrowserRouter>

                <Register />

            </BrowserRouter>

        );

        // Fill in all fields

        fireEvent.change(screen.getByTestId('username'), { target:
{ value: 'testuser' } });

        fireEvent.change(screen.getByTestId('password'), { target:
{ value: 'password123' } });

        fireEvent.change(screen.getByTestId('company-name'), { target:
{ value: 'Test Company' } });

        fireEvent.change(screen.getByTestId('email'), { target:
{ value: 'test@example.com' } });

        fireEvent.change(screen.getByTestId('address'), { target:
{ value: '123 Test St' } });

        fireEvent.change(screen.getByTestId('gstin'), { target:
{ value: '1234567890' } });

        fireEvent.change(screen.getByTestId('pan'), { target: { value:
'ABCDE1234F' } });

        // Submit form

        fireEvent.click(screen.getByRole('button', { name:
/Register/i }));

        // Check that error message is not displayed

        expect(screen.queryByTestId('error-message')).toBeNull();

    });



});
```

dashboard.test.js

```
import React from "react";

import { render, screen, fireEvent } from '@testing-library/react';
import '@testing-library/jest-dom';
import Dashboard from "../templates/dashboard";
import Invoice from "../templates/invoice";
import { BrowserRouter, Routes, Route } from "react-router-dom";

test('renders dashboard component without crashing', () => {
    render(
        <BrowserRouter>
            <Dashboard />
        </BrowserRouter>
    );
    // Check for heading
    expect(screen.getByRole('heading', { name:
/Welcome/i })).toBeInTheDocument();
});

test('tests for adding product', () => {
    render(
        <BrowserRouter>
            <Dashboard />
        </BrowserRouter>
    );
    // Check for heading
    expect(screen.getByRole('heading', { name: /Add Your
Product/i })).toBeInTheDocument();

    // Check for labels
    expect(screen.getByLabelText(/Item Name/i)).toBeInTheDocument();
    expect(screen.getByLabelText(/Item Price/i)).toBeInTheDocument();
    // Check for input fields

    expect(screen.getByTestId('item-name')).toBeInTheDocument();

    expect(screen.getByTestId('item-price')).toBeInTheDocument();



    // Check for add button

    expect(screen.getByRole('button', { name: /Add
Product/i })).toBeInTheDocument();



    // Fire event to add product with empty fields

    fireEvent.click(screen.getByRole('button', { name: /Add
Product/i }));

    expect(screen.getByText(/Both item name and price are
required./i)).toBeInTheDocument();



    // Fill the form and add product

    fireEvent.change(screen.getByLabelText(/Item Name/i), { target:
{ value: 'Test Product' } });

    fireEvent.change(screen.getByLabelText(/Item Price/i), { target:
{ value: '10' } });

    fireEvent.click(screen.getByRole('button', { name: /Add
Product/i }));

    // expect no error to be displayed

    expect(screen.queryByText(/Both item name and price are
required./i)).toBeNull();



    // Check if product is added to the table

    expect(screen.getByText(/Test Product/i)).toBeInTheDocument();

    expect(screen.getByText(/\$10\.00/i)).toBeInTheDocument();

});
```

```
test('rendering table and goto invoice button on adding product', ()
=> {
    render(
        <BrowserRouter>
            <Dashboard />
        </BrowserRouter>
    );


    // Fill the form and add product
    fireEvent.change(screen.getByLabelText(/Item Name/i), { target:
{ value: 'Test Product' } });
    fireEvent.change(screen.getByLabelText(/Item Price/i), { target:
{ value: '10' } });
    fireEvent.click(screen.getByRole('button', { name: /Add
Product/i }));


    // Check if product is added to the table
    expect(screen.getByText(/Test Product/i)).toBeInTheDocument();
    expect(screen.getByText(/\$10\.00/i)).toBeInTheDocument();


    // Check for the Go to Invoice Page button
    expect(screen.getByRole('button', { name: /Go to Invoice
Page/i })).toBeInTheDocument();


    // Check for remove button and its functionality
    fireEvent.click(screen.getByRole('button', { name: /Remove/i }));
    expect(screen.queryByText(/Test
Product/i)).not.toBeInTheDocument();
});


console.log(screen.debug()); // Logs the entire rendered HTML
```

```
test('final invoice submission on clicking go to invoice page and
displaying passed items', async () => {

    render(

        <BrowserRouter>

            <Routes>

                <Route path="/" element={<Dashboard />} />

                <Route path="/invoice" element={<Invoice />} />

            </Routes>

        </BrowserRouter>

    );



    // Simulate adding a product

    fireEvent.change(screen.getByTestId('item-name'), { target:
{ value: 'Product 1' } });

    fireEvent.change(screen.getByTestId('item-price'), { target:
{ value: '100' } });

    fireEvent.click(screen.getByRole('button', { name: /Add/i }));



    // Check for the Go to Invoice Page button and navigate

    fireEvent.click(screen.getByRole('button', { name: /Go to Invoice
Page/i }));



    // check for the Invoice page elements to be in the document

        expect(screen.getByText('Invoice
Generator')).toBeInTheDocument();

        expect(screen.getByText('Select a
product')).toBeInTheDocument();



    // Simulate selecting a product from the dropdown
    fireEvent.change(screen.getByLabelText('Product:'), { target: { value: 'Product 1' } });

    // Add the product to the invoice
    fireEvent.click(screen.getByRole('button', { name: /Add to Invoice/i }));

    // Wait for the product to be added to the table
        expect(screen.getByText('Product 1')).toBeInTheDocument();
});
```

invoice.test.js

```
import { render, screen, fireEvent } from '@testing-library/react';
import Invoice from '../templates/invoice';
import { MemoryRouter } from 'react-router-dom';
import '@testing-library/jest-dom';

const mockProducts = [
    { name: 'Product 1', price: 100.00 },
    { name: 'Product 2', price: 200.00 }
];

const renderInvoice = (initialState = {}) => {
    return render(
        <MemoryRouter initialEntries={[{ state: { products: mockProducts }, ...initialState }]}>
            <Invoice />
        </MemoryRouter>
    );
```

```
};

describe('Invoice Component', () => {
    let consoleSpy;

    beforeEach(() => {
        consoleSpy = jest.spyOn(console, 'log').mockImplementation(()
=> {});
    });

    afterEach(() => {
        consoleSpy.mockRestore();
    });

    it('renders Invoice component and checks initial state', () => {
        renderInvoice();

        // Check if the title is rendered
        expect(screen.getByText(/Invoice
Generator/i)).toBeInTheDocument();

        // Check if input fields are rendered
        expect(screen.getByPlaceholderText(/Enter your
name/i)).toBeInTheDocument();
        expect(screen.getByPlaceholderText(/Enter your
address/i)).toBeInTheDocument();
        expect(screen.getByPlaceholderText(/Enter your
email/i)).toBeInTheDocument();
        expect(screen.getByPlaceholderText(/Enter your
mobile/i)).toBeInTheDocument();
```

```
        // Check if the "Add to Invoice" button is rendered

        expect(screen.getByText(/Add to
Invoice/i)).toBeInTheDocument();

    });



    test('displays error when adding an invoice item without
selecting a product', () => {

        renderInvoice();



        // Click "Add to Invoice" without selecting a product

        fireEvent.click(screen.getByText(/Add to Invoice/i));



        // Check if the error message is displayed

        expect(screen.getByText(/Please select a
product/i)).toBeInTheDocument();

    });



    test('adds an item to the invoice and displays it in the table',
() => {

        renderInvoice();



        // Select a product

        fireEvent.change(screen.getByLabelText(/Product:/i), { target:
{ value: 'Product 1' } });

        fireEvent.keyDown(screen.getByLabelText(/Product:/i), { key:
'Enter', code: 'Enter' });



        // Set the quantity

        fireEvent.change(screen.getByPlaceholderText(/Enter
quantity/i), { target: { value: '2' } });
```

```
        // Click "Add to Invoice"
        fireEvent.click(screen.getByText(/Add to Invoice/i));

        // Check if the item is added to the invoice table
        expect(screen.getByText(/Product 1/i)).toBeInTheDocument();
        expect(screen.getByText('$100.00')).toBeInTheDocument();
        expect(screen.getByText('2')).toBeInTheDocument();
        expect(screen.getByText('$200.00')).toBeInTheDocument();
    });

    test('removes an item from the invoice', () => {
        renderInvoice();

        // Add a product to the invoice
        fireEvent.change(screen.getByLabelText(/Product:/i), { target:
{ value: 'Product 1' } });
        fireEvent.keyDown(screen.getByLabelText(/Product:/i), { key:
'Enter', code: 'Enter' });
        fireEvent.change(screen.getByPlaceholderText(/Enter
quantity/i), { target: { value: '2' } });
        fireEvent.click(screen.getByText(/Add to Invoice/i));

        // Remove the item
        fireEvent.click(screen.getByText(/Remove/i));

        // Check if the item is removed from the table
        expect(screen.queryByText(/Product
1/i)).not.toBeInTheDocument();
    });
```

```
// ------------------------------------------------------------------
------------------------------------------------------------------

    test('edits the quantity of an invoice item', () => {

        renderInvoice();



        // Add a product to the invoice

        fireEvent.change(screen.getByLabelText(/Product:/i), { target:
{ value: 'Product 1' } });

        fireEvent.keyDown(screen.getByLabelText(/Product:/i), { key:
'Enter', code: 'Enter' });

        fireEvent.change(screen.getByPlaceholderText(/Enter
quantity/i), { target: { value: '2' } });

        fireEvent.click(screen.getByText(/Add to Invoice/i));



        // Click "Edit" on the item

        fireEvent.click(screen.getByText(/Edit/i));



        // Change the quantity

        fireEvent.change(screen.getByDisplayValue('2'), { target:
{ value: '3' } });



        // Ensure the "Save" button is present before clicking

        fireEvent.click(screen.getByTestId('save-button'));



        // Check if the item is updated with the new quantity

        expect(screen.getByText('3')).toBeInTheDocument();

        expect(screen.getByText('$300.00')).toBeInTheDocument();

    });

// ------------------------------------------------------------------
------------------------------------------------------------------
```

```
    test('validates user details before saving the invoice', () => {

        renderInvoice();



        // Add a product to the invoice

        fireEvent.change(screen.getByLabelText(/Product:/i), { target:
{ value: 'Product 1' } });

        fireEvent.keyDown(screen.getByLabelText(/Product:/i), { key:
'Enter', code: 'Enter' });

        fireEvent.change(screen.getByPlaceholderText(/Enter
quantity/i), { target: { value: '2' } });

        fireEvent.click(screen.getByText(/Add to Invoice/i));



        // Try to save the invoice without entering user details

        fireEvent.click(screen.getByText(/Save Invoice/i));



        // Check if the error message is displayed

        expect(screen.getByText(/All user details fields are
required/i)).toBeInTheDocument();

    });



    test('saves the invoice when all fields are valid', () => {

        renderInvoice();



        // Add a product to the invoice

        fireEvent.change(screen.getByLabelText(/Product:/i), { target:
{ value: 'Product 1' } });

        fireEvent.keyDown(screen.getByLabelText(/Product:/i), { key:
'Enter', code: 'Enter' });

        fireEvent.change(screen.getByPlaceholderText(/Enter
quantity/i), { target: { value: '2' } });

        fireEvent.click(screen.getByText(/Add to Invoice/i));
        // Enter user details

        fireEvent.change(screen.getByPlaceholderText(/Enter your
name/i), { target: { value: 'John Doe' } });

        fireEvent.change(screen.getByPlaceholderText(/Enter your
address/i), { target: { value: '123 Main St' } });

        fireEvent.change(screen.getByPlaceholderText(/Enter your
email/i), { target: { value: 'john@example.com' } });

        fireEvent.change(screen.getByPlaceholderText(/Enter your
mobile/i), { target: { value: '1234567890' } });



        // Save the invoice

        fireEvent.click(screen.getByText(/Save Invoice/i));



        // Check if the invoice is saved (for simplicity, we just log
it in the console)

        expect(console.log).toHaveBeenCalledWith('Invoice saved:',
expect.any(Object));

    });



    test('generates a PDF when all fields are valid', () => {

        renderInvoice();



        // Add a product to the invoice

        fireEvent.change(screen.getByLabelText(/Product:/i), { target:
{ value: 'Product 1' } });

        fireEvent.keyDown(screen.getByLabelText(/Product:/i), { key:
'Enter', code: 'Enter' });

        fireEvent.change(screen.getByPlaceholderText(/Enter
quantity/i), { target: { value: '2' } });

        fireEvent.click(screen.getByText(/Add to Invoice/i));



        // Enter user details
        fireEvent.change(screen.getByPlaceholderText(/Enter your
name/i), { target: { value: 'John Doe' } });

        fireEvent.change(screen.getByPlaceholderText(/Enter your
address/i), { target: { value: '123 Main St' } });

        fireEvent.change(screen.getByPlaceholderText(/Enter your
email/i), { target: { value: 'john@example.com' } });

        fireEvent.change(screen.getByPlaceholderText(/Enter your
mobile/i), { target: { value: '1234567890' } });



        // Generate the PDF

        fireEvent.click(screen.getByText(/Generate PDF/i));



        // Check if the PDF is generated (for simplicity, we just log
it in the console)

        expect(console.log).toHaveBeenCalledWith('PDF generated:',
expect.any(Object));

    });  });
```

18.Bug List

1. Invoice Page CSS Not Responsive

- **Description**: The invoice page's CSS is not aligning and resizing correctly when the device size changes. This issue affects the layout, making it difficult to view and interact with the invoice on different screen sizes.
- **Priority**: High
- **Status**: Identified
- **Steps to Reproduce**:
    1. Open the invoice page on a desktop browser.
    2. Resize the browser window to simulate different screen sizes (e.g., tablet, mobile).
    3. Observe that the page layout does not adjust properly, leading to misalignment and overflow issues.
- **Expected Behavior**: The invoice page should be fully responsive, with elements adjusting correctly to different screen sizes without losing alignment or readability.

19. Agile Methodology in the Invoice Generation Project

In the development of the invoice generation project, we utilized Agile methodology to ensure a flexible, collaborative, and efficient approach. Agile enabled us to quickly adapt to changes and deliver incremental improvements, ensuring the project met user needs and maintained high quality.

1. Introduction to Agile Methodology

Agile methodology is a project management and software development approach that emphasizes iterative progress, collaboration, and flexibility. Unlike traditional methodologies, Agile focuses on delivering small, functional pieces of the project frequently, enabling continuous feedback and adjustments.

2. Key Principles of Agile Methodology

- **Iterative Development:** Work is divided into small, manageable iterations or sprints, usually lasting two to four weeks. Each iteration results in a potentially shippable product increment.
- **Customer Collaboration:** Regular interaction with stakeholders to gather feedback and make necessary adjustments.
- **Responding to Change:** Flexibility to adapt to changing requirements, even late in the development process.
- **Cross-Functional Teams:** Collaboration among team members with different expertise to ensure comprehensive development and testing.

3. Agile Process in the Invoice Generation Project

The following steps outline how we implemented Agile methodology in the invoice generation project:

1. Sprint Planning

- **Sprint Planning Meetings:** At the beginning of each sprint, we held planning meetings to define the goals and tasks for the upcoming iteration. We prioritized user stories and features such as invoice creation, quantity validation, and PDF generation based on their importance and complexity.
- **Task Breakdown:** Each user story was broken down into smaller, actionable tasks. This made it easier to estimate effort and assign tasks to team members, such as designing UI components or integrating the Flask backend.

2. Daily Status Updates

- **Daily Status Updates:** Instead of traditional stand-up meetings, we had daily status updates where each team member reported their progress to their respective reporting manager. This facilitated communication, helped identify blockers, and ensured alignment among team members.

3. Iterative Development and Testing

- **Development:** Each team member focused on their assigned tasks, developing components and features iteratively. This included both frontend (React components for invoice forms and previews) and backend (API integration, data handling) work.
- **Continuous Testing:** Testing was an integral part of each iteration. As features were developed, they were tested manually for responsiveness, functionality, and performance. This ensured that any issues were identified and addressed early.

4. Sprint Review and Retrospective

- **Sprint Review:** At the end of each sprint, we held a review meeting to demonstrate the completed work to stakeholders. Feedback was gathered and used to refine future iterations, focusing on aspects like invoice format or user experience improvements.
- **Retrospective Meeting:** Conducted retrospective meetings to reflect on the sprint. The team discussed what went well, what could be improved, and any actions needed to enhance future sprints.

5. Continuous Improvement

- **Adaptation and Flexibility:** Based on feedback from reviews and retrospectives, we adapted our approach and made necessary changes to improve the process. This included refining user stories, adjusting priorities, and improving communication.
- **Incremental Delivery:** By delivering functional increments at the end of each sprint, we ensured continuous progress and allowed stakeholders to see tangible results. This incremental approach helped in building a reliable and user-friendly invoice generation application.

20. Enhancements and Future

1. Enhanced User Interface

Future Work: Improve the user interface by incorporating modern design principles and user experience (UX) best practices. This includes enhancing visual elements, streamlining navigation, and making the invoice generation process more intuitive for users.

2. Automated Testing

Future Work: Implement automated testing frameworks to streamline the testing process. This would include unit tests, integration tests, and end-to-end tests to ensure that new features and functionalities, like invoice creation and preview, are thoroughly vetted without extensive manual testing.

3. Performance Optimization

Future Work: Further optimize performance by analyzing and improving database queries, implementing caching strategies, and exploring lazy loading techniques for large datasets. This will enhance response times, especially during bulk invoice generation and user interactions.

4. User Role Management

Future Work: Develop a user role management system that allows different access levels for users. This could enable functionalities such as admin dashboards, different invoice templates for different user roles, and controlled access to sensitive areas of the application.

5. Real-Time Notifications

Future Work: Integrate real-time notification features using WebSockets or similar technologies. This would allow users to receive updates on their invoice status instantly, improving engagement and user satisfaction, such as being notified when an invoice is successfully generated or sent.

6. Expanded Invoice Customization Options

Future Work: Allow users to customize invoice templates more extensively. This could include options for different styles, fonts, and layouts, enhancing personalization and making the invoices more appealing to different business needs.

7. Mobile Application

Future Work: Consider developing a mobile application or a responsive web app to provide users with the ability to generate and manage invoices on-the-go. This would cater to a wider audience, particularly small business owners who need mobile access.

8. Integration with Third-Party Services

Future Work: Explore integrations with third-party services, such as payment gateways for processing payments directly through the invoice or cloud storage solutions for storing and sharing invoices securely.

9. Enhanced Analytics and Reporting

Future Work: Implement advanced analytics and reporting features that provide insights into user behavior, invoice generation trends, and system performance. This data can inform future improvements and help in making data-driven business decisions.

21.Release Notes for Invoice Generation Project

Version 1.0.0

Overview

This release introduces the **Invoice Generation System** with a fully functional web-based user interface for generating and managing invoices. The system allows users to add products, calculate taxes and discounts, and generate detailed invoices that can be previewed and downloaded in PDF format.

Features

Product Management

1. Users can add products by specifying the item name, price, and quantity.
2. The system supports quantity validation to ensure stock availability.
3. Duplicate products are managed with automatic quantity updates.

Stock Management

1. Real-time tracking of available stock for each product.
2. Automatic stock updates as products are added or removed from invoices.

Invoice Generation

1. Generate invoices with detailed item breakdowns, including CGST, SGST, and discounts.
2. Flexible discounting system with configurable discount percentages.
3. Automatic rounding of final amounts to the nearest whole number.

User Details

1. Capture user details including name, address, email, and mobile number.
2. Validation of email format and mobile number for accuracy.

Invoice Preview and Download

1. Users can preview their invoices before finalizing and downloading them in PDF format.
2. The preview page provides an option to make last-minute adjustments.

Responsive UI

1. A user-friendly interface designed to work seamlessly across different devices and screen sizes.
2. Clear error messaging and interactive elements ensure a smooth user experience.

Bug Fixes

- Fixed issues with product price validation to ensure only valid prices with up to two decimal places are accepted.
- Resolved issues related to stock quantity updates when products are removed from the invoice.
- Corrected calculation errors in discount and tax application to ensure accurate final amounts.

Known Issues

- Editing invoice items is restricted to one item at a time due to the current handling of stock quantities.
- The system does not currently support complex tax scenarios beyond CGST and SGST.

Enhancements

- Added error handling for quantity in edit mode to avoid conflicts with stock management.
- Enhanced discount messaging to inform users when a discount has been applied to their purchase.

Future Work

- Implementation of more advanced tax calculations, including support for different tax rates.
- Integration with a persistent database to allow for long-term storage of products and invoices.
- Expansion of the system to support multiple users and organizations.

Technical Details

- **Frontend**: React, with custom CSS for styling.
- **Backend**: Flask, serving as the API layer.
- **State Management**: React state hooks for managing user inputs and dynamic data.

22.List of Abbreviations / Nomenclature:

The list of abbreviations and nomenclature used in the codebase:

React Components and Hooks:

1. **JSX**: JavaScript XML
2. **DOM**: Document Object Model
3. **useState**: A React Hook for managing state in functional components
4. **useEffect**: A React Hook for managing side effects in functional components
5. **useContext**: A React Hook for consuming context values

Props and State:

1. **props**: Properties passed to components
2. **state**: Local state within a component
3. **setState**: Function to update the state
4. **products**: State variable for storing the list of products
5. **selectedProduct**: State variable for storing the selected product
6. **quantity**: State variable for storing the quantity of the selected product
7. **totalPrice**: State variable for storing the total price of the invoice
8. **customerName**: State variable for storing the customer's name
9. **customerEmail**: State variable for storing the customer's email
10. **invoiceDate**: State variable for storing the date of the invoice
11. **invoiceItems**: State variable for storing the list of items in the invoice
12. **errorMessage**: State variable for storing error messages
13. **submittedInvoiceId**: State variable for storing the ID of the submitted invoice
14. **invoiceType**: State variable for storing the type of invoice (A4 or thermal)

Routing and URLs:

1. **BrowserRouter**: Component for enabling routing using the HTML5 history API
2. **Route**: Component for defining a route
3. **Routes**: Component for grouping multiple Route components
4. **window.location.href**: Property for redirecting to a different URL

FormData and API:

1. **FormData**: Interface for creating a set of key/value pairs representing form fields and their values
2. **axios**: Promise-based HTTP client for making requests
3. **response**: Object containing the response from an HTTP request
4. **sessionStorage**: Web storage object for storing data that is accessible only within the current session
5. **API**: Application Programming Interface

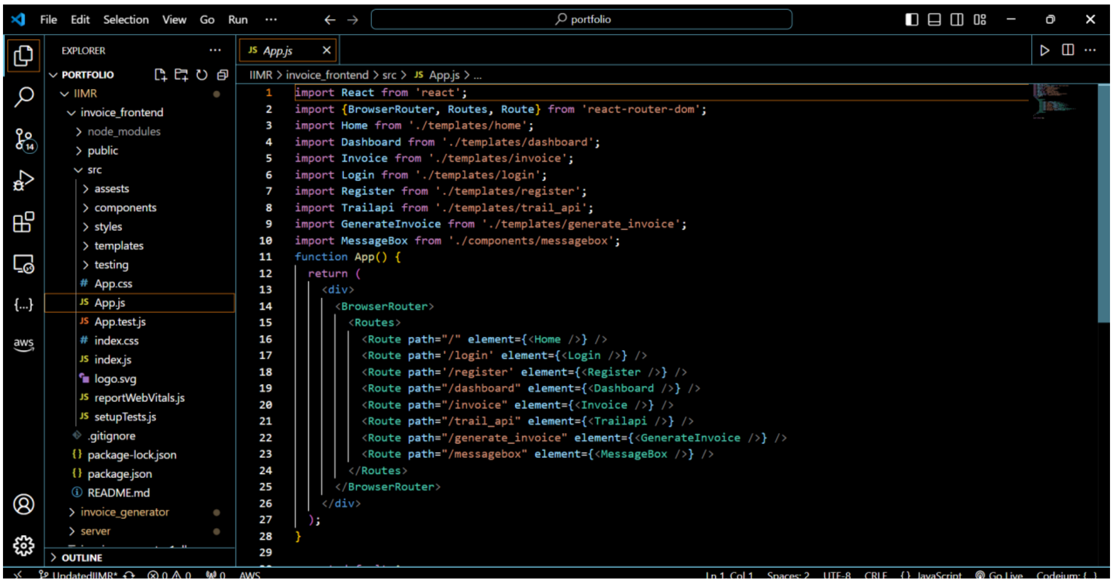
Miscellaneous:

1. **UI**: User Interface
2. **invoice-table**: Class name for styling the invoice table
3. **product-options**: Class name for styling the product options section
4. **generate-invoice-btn**: Class name for styling the generate invoice button
5. **form-container**: Class name for styling the form container
6. **form-group**: Class name for styling form groups
7. **form-submit-btn**: Class name for styling the form submit button
8. **invoice-preview-container**: Class name for styling the invoice preview container
9. **print-section**: Class name for styling the print section
10. **download-button**: Class name for styling the download button

23. List Of Figures :

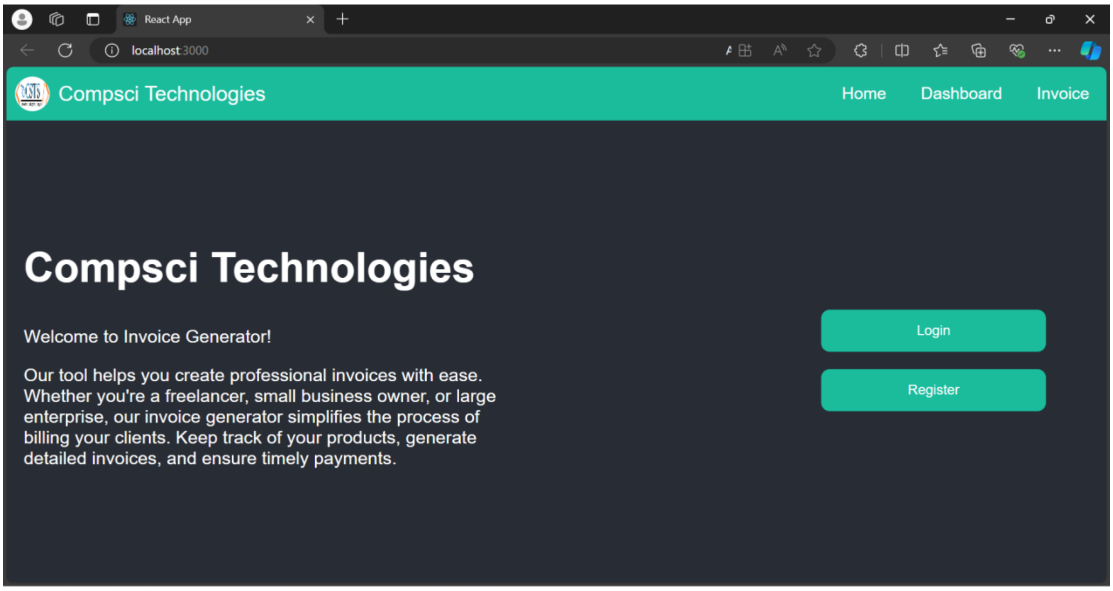
24. ScreenShots :

CODEBASE :



```
import React from 'react';
import {BrowserRouter, Routes, Route} from 'react-router-dom';
import Home from './templates/home';
import Dashboard from './templates/dashboard';
import Invoice from './templates/invoice';
import Login from './templates/login';
import Register from './templates/register';
import Trailapi from './templates/trail_api';
import GenerateInvoice from './templates/generate_invoice';
import MessageBox from './components/messagebox';
function App() {
  return (
    <div>
      <BrowserRouter>
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path='/login' element={<Login />} />
          <Route path='/register' element={<Register />} />
          <Route path="/dashboard" element={<Dashboard />} />
          <Route path="/invoice" element={<Invoice />} />
          <Route path="/trail_api" element={<Trailapi />} />
          <Route path="/generate_invoice" element={<GenerateInvoice />} />
          <Route path="/messagebox" element={<MessageBox />} />
        </Routes>
      </BrowserRouter>
    </div>
  );
}

```

Home Page:

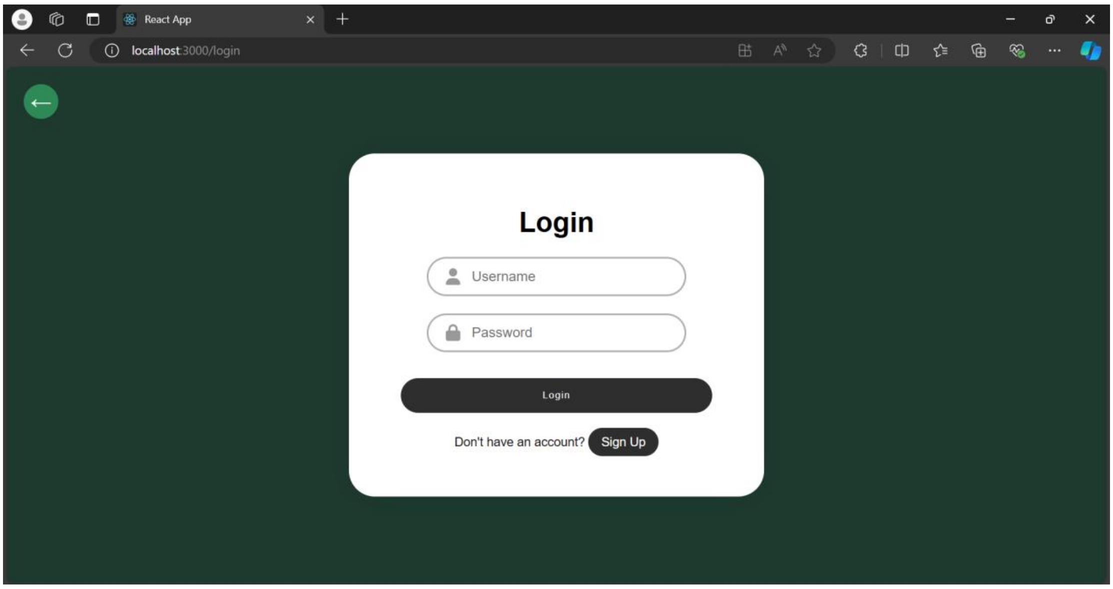


Compsci Technologies

Welcome to Invoice Generator!

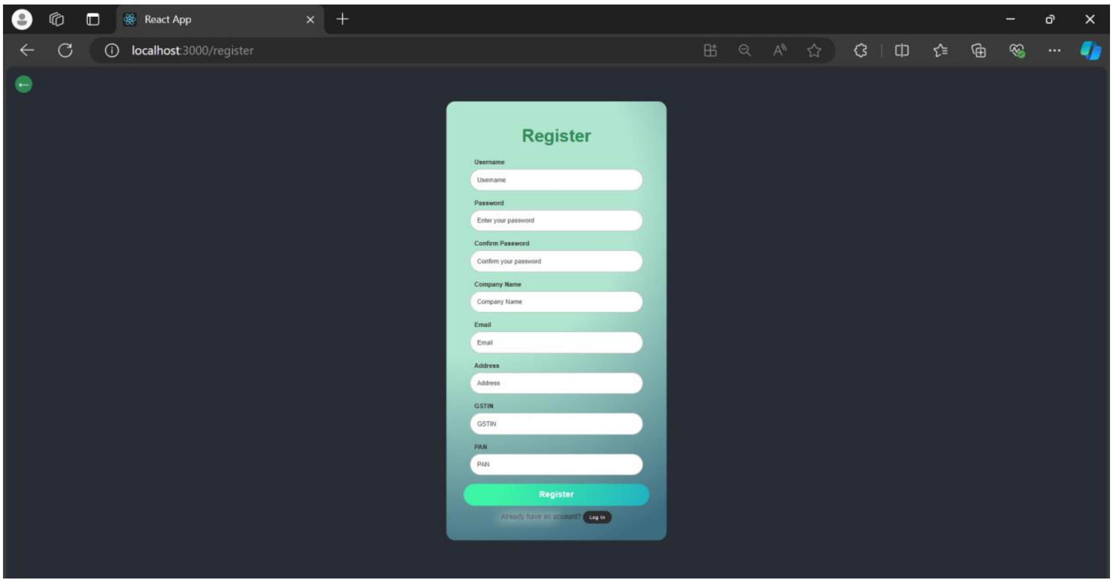
Our tool helps you create professional invoices with ease. Whether you're a freelancer, small business owner, or large enterprise, our invoice generator simplifies the process of billing your clients. Keep track of your products, generate detailed invoices, and ensure timely payments.

LOGIN PAGE:



Don't have an account?

REGISTER PAGE:



Register

Username

Username

Password

Enter your password

Confirm Password

Confirm your password

Company Name

Company Name

Email

Email

Address

Address

GSTIN

GSTIN

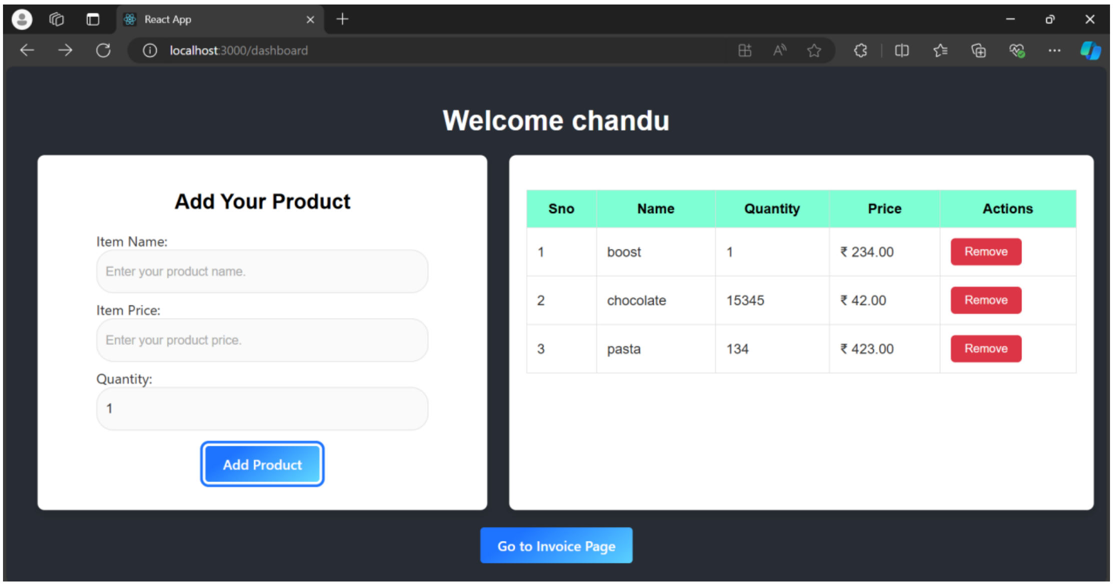
PAN

PAN

Register

Already have an account? Log In

DASHBOARD PAGE:



Welcome chandu

Add Your Product

Item Name:

Enter your product name.

Item Price:

Enter your product price.

Quantity:

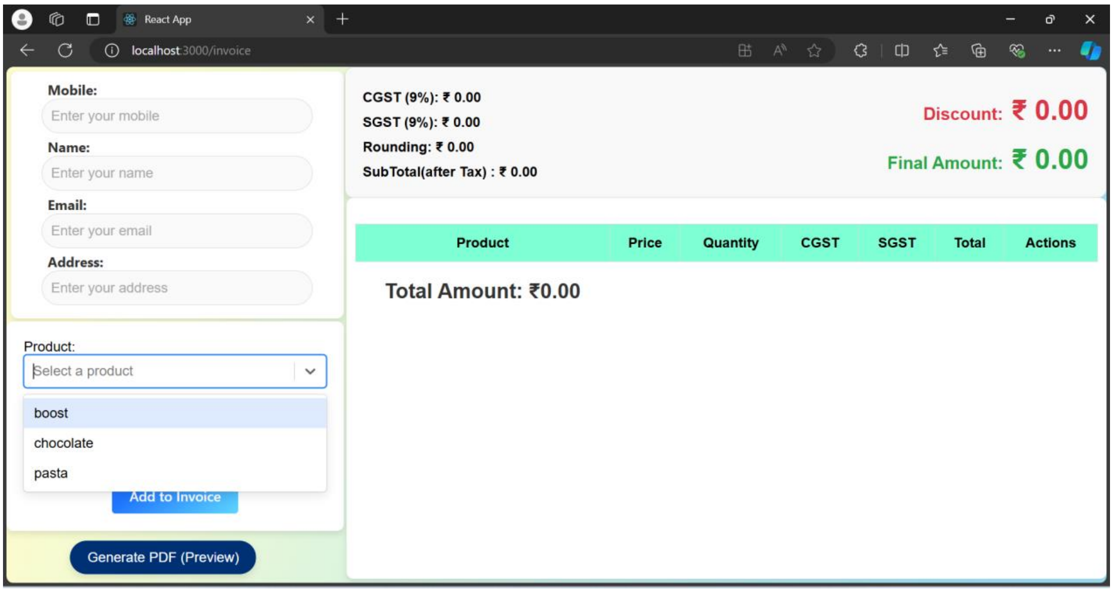
1

Add Product

Sno	Name	Quantity	Price	Actions
1	boost	1	₹ 234.00	Remove
2	chocolate	15345	₹ 42.00	Remove
3	pasta	134	₹ 423.00	Remove

Go to Invoice Page

INVOICE PAGE:



Mobile: Enter your mobile

Name: Enter your name

Email: Enter your email

Address: Enter your address

Product: Select a product

- boost
- chocolate
- pasta

CGST (9%): ₹ 0.00
SGST (9%): ₹ 0.00
Rounding: ₹ 0.00
SubTotal(after Tax) : ₹ 0.00

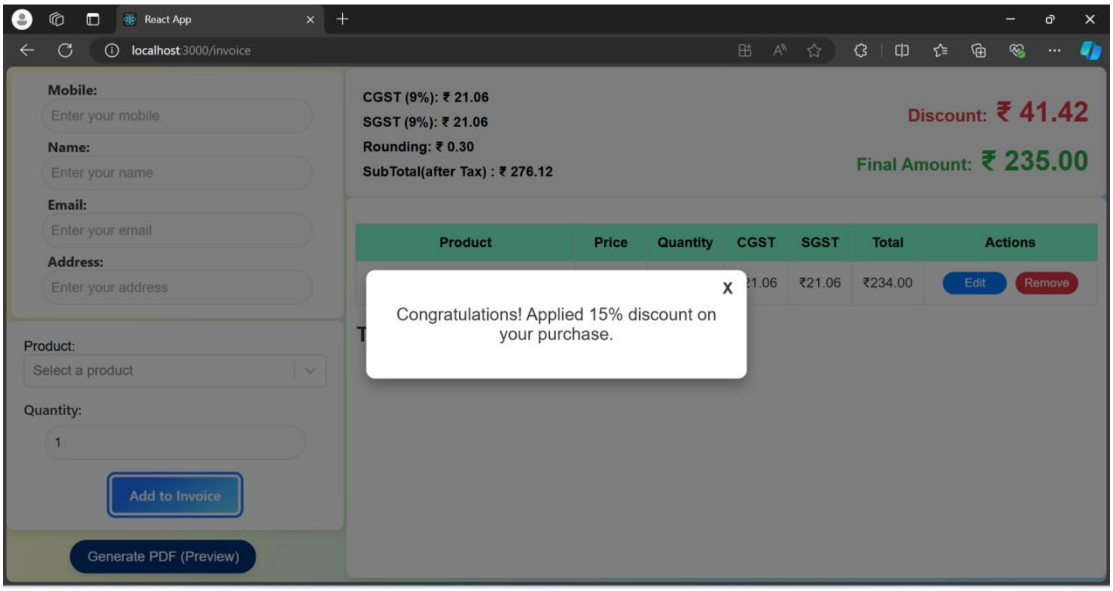
Discount: ₹ 0.00

Final Amount: ₹ 0.00

Product	Price	Quantity	CGST	SGST	Total	Actions

Total Amount: ₹0.00

MESSAGE BOX ON FIRST PRODUCT ADDITION:



Mobile: Enter your mobile

Name: Enter your name

Email: Enter your email

Address: Enter your address

Product: Select a product

Quantity: 1

CGST (9%): ₹ 21.06
SGST (9%): ₹ 21.06
Rounding: ₹ 0.30
SubTotal(after Tax) : ₹ 276.12

Discount: ₹ 41.42

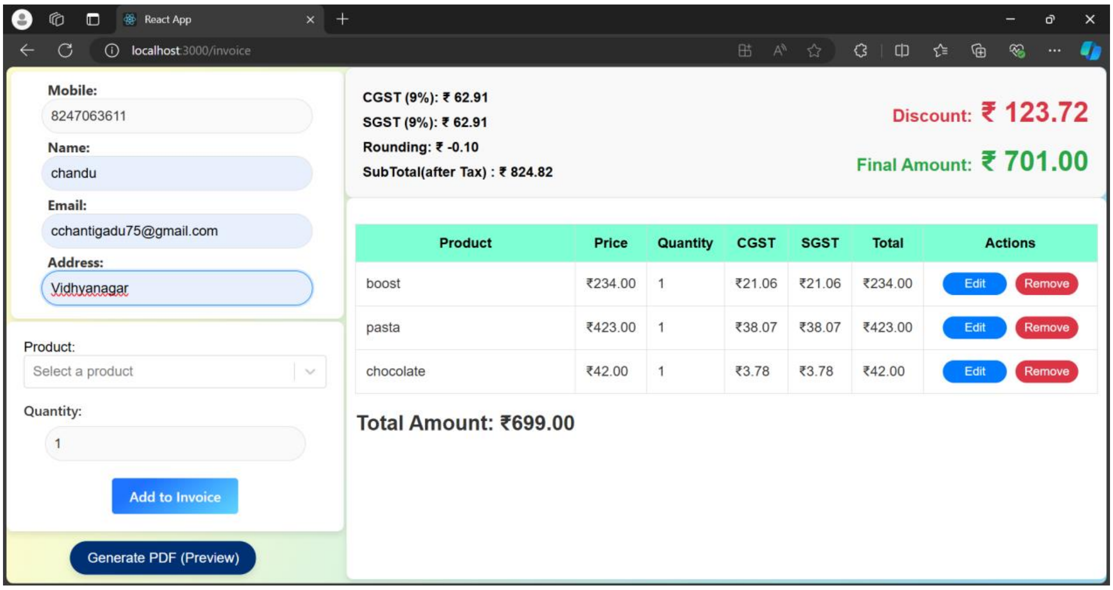
Final Amount: ₹ 235.00

Product	Price	Quantity	CGST	SGST	Total	Actions
			21.06	₹21.06	₹234.00	Edit Remove

Congratulations! Applied 15% discount on your purchase.

T

INVOICE PAGE FILLED:



Mobile: 8247063611

Name: chandu

Email: cchantigadu75@gmail.com

Address: Vidhyanagar

Product: Select a product

Quantity: 1

CGST (9%): ₹ 62.91
SGST (9%): ₹ 62.91
Rounding: ₹ -0.10
SubTotal(after Tax) : ₹ 824.82

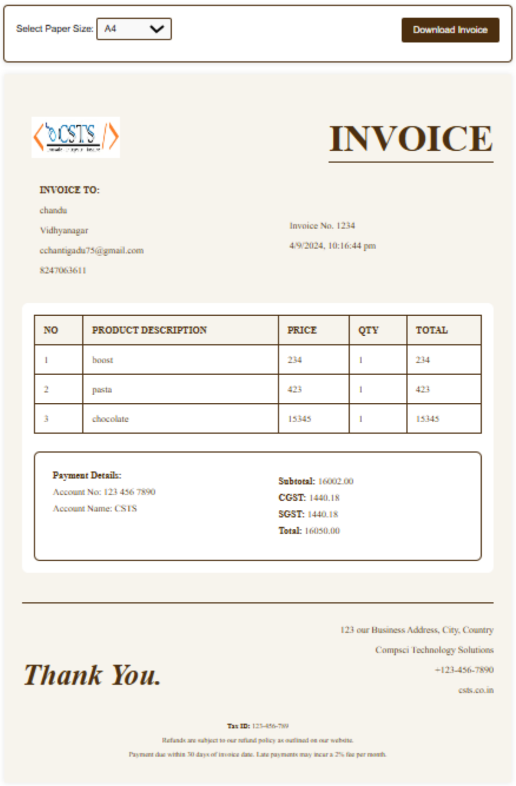
Discount: ₹ 123.72

Final Amount: ₹ 701.00

Product	Price	Quantity	CGST	SGST	Total	Actions
boost	₹234.00	1	₹21.06	₹21.06	₹234.00	Edit Remove
pasta	₹423.00	1	₹38.07	₹38.07	₹423.00	Edit Remove
chocolate	₹42.00	1	₹3.78	₹3.78	₹42.00	Edit Remove

Total Amount: ₹699.00

INVOICE PREVIEW:



Select Paper Size: A4

INVOICE

INVOICE TO:
chandu
Vidhyanagar
cchantigadu75@gmail.com
8247063611

Invoice No. 1234
4/9/2024, 10:16:44 pm

NO	PRODUCT DESCRIPTION	PRICE	QTY	TOTAL
1	boost	234	1	234
2	pasta	423	1	423
3	chocolate	15345	1	15345

Payment Details:
Account No: 123 456 7890
Account Name: CSTS

Subtotal: 16002.00
CGST: 1440.18
SGST: 1440.18
Total: 16050.00

Thank You.

123 our Business Address, City, Country
Compsci Technology Solutions
+123-456-7890
csts.co.in

Tax ID: 123-456-789
Refunds are subject to our refund policy as outlined on our website.
Payment due within 30 days of invoice date. Late payments may incur a 2% fee per month.

GENERATED INVOICE PDFS:

A4_invoice.pdf

thermal_invoice (1).pdf

25. Conclusion & Future Work

The completion of the invoice generation project signifies a major advancement in automating the invoicing process, making it more efficient and error-free. By utilizing modern web technologies, including React for the frontend and Flask for the backend, the system offers a robust solution for generating and managing invoices in both A4 and thermal formats.

Key Achievements:

User Interface and Backend Integration: Successfully integrated React for frontend development and Flask for backend operations, ensuring seamless data exchange and process execution between the client and server.

Enhanced User Experience: Developed a user-friendly interface that simplifies the invoicing workflow, including features like product selection, quantity validation, real-time invoice previews, and customizable invoice formats.

Scalability and Performance: Implemented strategies to optimize the system's scalability and performance, including efficient data handling and resource management, allowing the system to handle varying inventory sizes and workloads.

Contributions to Professional Development:

This project has greatly enhanced my full-stack development skills, particularly in integrating React and Flask to build web applications. It provided practical experience in creating responsive UIs, managing state and props, implementing backend logic, and ensuring seamless API communication. Collaboration across various project phases deepened my understanding of web architecture and deployment strategies.

Future Work:

User Experience Improvements:

Feedback Mechanisms: Implement feedback mechanisms to collect user insights and make continuous improvements to the interface based on real-world usage. Regular usability testing and iterative design enhancements will ensure the system remains user-friendly and efficient.

Security and Reliability:

Enhanced Security Measures: Strengthen data security protocols, including encryption and access control, and conduct regular security audits to protect sensitive business and user information. This will enhance the reliability and trustworthiness of the invoice generation system.

Research and Innovation:

Exploration of Emerging Technologies: Investigate the potential of emerging technologies such as machine learning for automating invoice categorization and blockchain for secure transaction verification. These advancements could further enhance the system's functionality and reliability.

26.Invoice Generation System - User Manual

1. Introduction

This user manual provides instructions on how to use the Invoice Generation System. The system allows users to generate, preview, and print invoices based on their organization's inventory data.

2. Accessing the System

- **Login**: Users need to log in with their credentials. Enter your username and password, then click on the 'Login' button.
- **Signup**: If you don't have an account, click on 'Signup' and provide the required details to create one.

3. Dashboard Overview

Once logged in, you will be directed to the dashboard. Here, you can view your organization's inventory data and navigate to different sections of the system.

4. Generating Invoices

- **Select Products**: Browse through the inventory list and select the products you wish to include in the invoice.
- **Enter Quantity**: Specify the quantity for each selected product.
- **Enter User Details**: Input the customer's name, contact information, and any other necessary details.
- **Preview Invoice**: After entering all necessary details, click on 'Preview' to view a draft of the invoice.

- **Print Invoice**: If everything is correct, click on 'Print' to generate the invoice. You can choose between A4 and thermal print formats.

5. Managing Inventory

- **View Inventory**: On the dashboard, you can view your current inventory, including product names, quantities, and prices.
- **Update Inventory**: If you have permission, you can update inventory details as needed.

6. Additional Features

- **Search Function**: Use the search bar to quickly find specific products in your inventory.

7. Troubleshooting

- **Forgot Password**: Click on 'Forgot Password' on the login page and follow the instructions to reset your password.
- **Error Messages**: If you encounter any error messages, follow the on-screen instructions or contact support.

8. Contact Support

For further assistance, please reach out to the support team at **csts.co.in**

27. References

References Used in the Invoice Generation Project

Flask Documentation
Flask is a lightweight WSGI web application framework in Python. It is designed to make getting started quick and easy, with the ability to scale up to complex applications.
https://flask.palletsprojects.com/

React Documentation
React is a JavaScript library for building user interfaces, maintained by Facebook. It allows developers to create large web applications that can update and render efficiently in response to data changes.
https://reactjs.org/docs/getting-started.html

Axios Library Documentation
Axios is a promise-based HTTP client for the browser and Node.js. It makes it easy to send asynchronous HTTP requests to REST endpoints and perform CRUD operations.
https://axios-http.com/docs/intro

Flask-CORS Documentation
Flask-CORS is a Flask extension for handling Cross-Origin Resource Sharing (CORS), making it easier to configure your server to allow for cross-origin requests.
https://flask-cors.readthedocs.io/en/latest/

React-Bootstrap Documentation
React-Bootstrap is a popular front-end framework rebuilt for React. It allows for the use of Bootstrap components with React's declarative syntax.
https://react-bootstrap.github.io/

Convert HTML to PDF with JavaScript
This resource provides methods and best practices for converting HTML content to PDF format using JavaScript, a crucial part of generating printable invoices.
https://parall.ax/blog/view/3313/converting-html-to-pdf

Python 3.12.3 Documentation
Python is an interpreted, high-level, general-purpose programming language. This documentation provides detailed guidance on Python 3.12.3 features and libraries.
https://docs.python.org/3/

MDN Web Docs (Mozilla Developer Network)
MDN provides comprehensive documentation and tutorials for web technologies, including HTML, CSS, and JavaScript, which were essential for front-end development.
https://developer.mozilla.org/en-US/

Stack Overflow
A platform where developers ask and answer questions. It was invaluable for troubleshooting and finding solutions to technical challenges during development.
https://stackoverflow.com/

GitHub
GitHub is a platform for hosting and collaborating on code repositories. It facilitated version control and collaboration throughout the project.
https://github.com/