1. INTRODUCTION

1.1 INTRODUCTION

An invoice generator is software designed to create and manage invoices efficiently for businesses and freelancers. It simplifies the billing process, allowing users to generate professional invoices quickly and accurately.

This system automates various invoicing tasks, such as calculating totals, applying taxes, and sending a pdf to the concerned businesses. With an invoice generator, businesses can easily track their sales, monitor outstanding payments, and maintain financial records systematically.

An invoice generator is essential for maintaining accurate financial documentation. It allows users to customize invoices with their branding, include detailed item descriptions, and specify payment terms. By using this system, users can streamline their billing process, reduce errors, and ensure timely payments.

The primary purpose of an invoice generator is to provide a user-friendly platform for creating invoices, ultimately saving time and enhancing productivity. With instant access to invoicing data, businesses can focus on growth while ensuring their financial transactions are managed efficiently.

1.2 OBJECTIVE

The Invoice Generator's main goal is to manage client information, invoices, payments, and services provided. It streamlines the creation and tracking of invoices, ensuring that financial data is organized and accessible. The system includes essential client details like contact information and payment history, as well as invoice specifics such as item descriptions and due dates. Access is restricted to authorized personnel to maintain data integrity and

confidentiality. By optimizing the invoicing process, the Invoice Generator aims to improve efficiency, reduce errors, and enhance cash flow management for business.

1.3. FEATURES AND FUNCTIONS

- SIGN IN AND SIGN UP
- CUSTOMER DETAILS
- INVOICE TEMPLATES VIEWER
- CONTENTS UPDATE PAGE
- INVOICE PREVIEW
- CONTENTS TABLE AND INVOICE HISTORY
- PDF GENERATION AND SHARING

Ch 2. TECH STACK USED

2.1 INTEGRATION SOFTWARE

VISUAL STUDIO CODE

Visual Studio Code (VS Code) is a lightweight, open-source code editor designed for versatility and speed. It supports multiple programming languages and offers a rich extension ecosystem that allows for seamless integration with various tools and platforms.

Developers can enhance their workflow with built-in Git support, debugging features, and customizable settings. VS Code also supports live collaboration, making it ideal for team projects. Its user-friendly interface and powerful features make it a top choice for modern web and software development.

2.2 APPLICATION LANGUAGES

We utilized the PERN stack—PostgreSQL, Express.js, React, and Node.js—for our web app development. This robust combination enables efficient database management, seamless backend communication, and dynamic front-end rendering. Leveraging the strengths of each technology, we created a scalable, high-performance application tailored to meet user needs and enhance overall functionality.

2.2.1 REACT

React is a powerful front-end development library that simplifies the creation of interactive user interfaces. Its component-based architecture allows developers to build encapsulated components that manage their own state, promoting reusability and maintainability. This modular approach streamlines the development process, making it easier to manage complex UIs.

One of React's standout features is its virtual DOM, which optimizes rendering by efficiently updating only the parts of the UI that change. This results in improved performance

and a smoother user experience, even in applications with frequent updates. Additionally, React's unidirectional data flow enhances predictability, making it easier to debug and understand how data changes affect the UI.

With a vibrant ecosystem and extensive community support, React offers a wealth of libraries and tools, such as React Router for navigation and Redux for state management

2.2.2 NODE

Node.js is a powerful, open-source JavaScript runtime built on Chrome's V8 engine, enabling server-side execution of JavaScript. Its non-blocking, event-driven architecture allows for high concurrency and scalability, making it ideal for building fast and efficient web applications. Node.js excels in handling real-time applications, such as chat apps and online gaming.

Additionally, Node.js benefits from a rich ecosystem of libraries and packages available through npm (Node Package Manager). This extensive toolkit empowers developers to streamline their workflows and integrate various functionalities, fostering rapid development and innovation in modern web solutions.

2.2.3 EXPRESS

Express.js is a minimalist web framework for Node.js that simplifies the process of building robust web applications and APIs. It offers a range of features, including routing, middleware support, and template engines, allowing developers to create scalable and maintainable server-side applications. With its straightforward syntax and extensive ecosystem, Express.js accelerates development while providing flexibility and performance for handling HTTP requests and responses.

2.2.4 POSTGRESQL

PostgreSQL is a versatile open-source relational database management system well-suited for storing invoice details. Its support for various data types, including JSON and arrays, allows for flexible invoice structures, accommodating complex information like line items and

customer details. The ability to perform complex queries ensures quick retrieval of specific invoices or summaries.

One of PostgreSQL's key features is its adherence to ACID principles, ensuring reliable transactions. This is crucial for invoice management, as it guarantees that all changes are accurately recorded, preventing data loss or corruption during updates. The database’s robust indexing options further enhance performance, allowing for efficient searches and reporting.

Security is paramount in PostgreSQL, with features like role-based access controls, SSL encryption for data in transit, and detailed logging capabilities. These security measures protect sensitive invoice data, ensuring that only authorized personnel can access or modify records, thus maintaining confidentiality and compliance.

SET-UP AND DATA FLOW

3.1 PREREQUISITES AND SET UP

To set up the PERN stack in Visual Studio Code, you'll need Node.js and npm for server-side JavaScript execution and package management. Additionally, PostgreSQL is required for database management. Visual Studio Code serves as the code editor, and extensions like ESLint and Prettier enhance coding practices.

The development process involves essential packages such as Express for building the server, the `pg` client for PostgreSQL integration, and `cors` middleware to handle Cross-Origin Resource Sharing. These components collectively create a robust foundation for developing scalable web applications using the PERN stack, ensuring smooth data management and efficient server-client interactions.

3.2 DATA FLOW

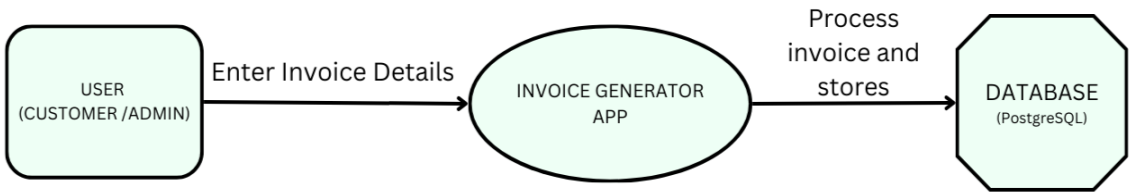
A Data Flow Diagram(DFD) is an important tool used by system analysts to understand how data moves through a system. It shows how data is processed, the
transformation of inputs into outputs and how entities such as files, databases or external entities interact with the system.

In the case of Invoice Generator , the data flow diagram will represent how data such as user information ,invoice details,items and pricing data flow through various processes, interacts with external entities and how the system provides outputs like the generation of invoices or reports.

Level 0 DFD (Context Diagram)

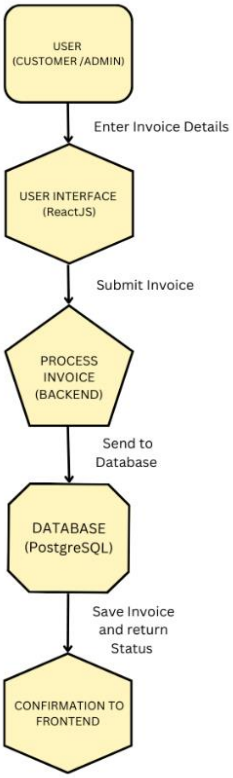
The Level 0 DFD offers a high-level view of the Invoice Generator system, showing the basic interaction between the user (customer or admin) and the application. The user inputs invoice details, which the system processes and stores. The Invoice Generator app then interacts with the database (PostgreSQL) to save the details and retrieve any necessary

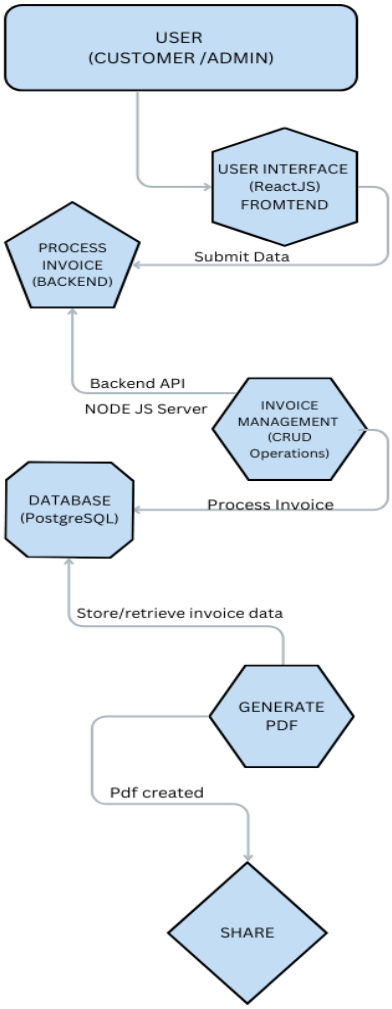
information. This level highlights the core functionality of accepting user input, processing it, and storing it in the database.



Level 1 DFD

The Level 1 DFD breaks down the process further, focusing on the flow between the user interface, backend processing, and the database. Here, the user enters details through the frontend (React interface) and submits them for processing. The backend receives this data, prepares it for storage and communicates with the PostgreSQL database to save the invoice. The database then sends a confirmation back to the frontend, notifying the user of the invoice status, whether successfully stored or if errors occurred.





Level 2 DFD

The Level 2 DFD provides a more detailed view of each stage, emphasizing specific actions taken within the system. When the user enters data, the interface validates it before sending it to the backend. The backend then formats the data, performs any necessary calculations (like tax or discounts), and sends it to the database. The database manages storage, retrieving any required data or updating existing records. After processing, it relays a confirmation or error message, enabling the frontend to display an accurate response to the user. This level shows the system's logic in managing invoices comprehensively, ensuring data integrity and efficient processing.

3.3 DATA DICTIONARIES:

TABLE: invoiceitems

```
InvoiceGenerator=# \d invoiceitems
                                  Table "public.invoiceitems"
  Column   |     Type      | Collation | Nullable |                   Default
-----------+---------------+-----------+----------+-----------------------------------------------
 itemid    | integer       |           | not null | nextval('invoiceitems_itemid_seq'::regclass)
 invoiceid | integer       |           |          |
 quantity  | integer       |           | not null |
 rate      | numeric(10,2) |           | not null |
Indexes:
    "invoiceitems_pkey" PRIMARY KEY, btree (itemid)
Foreign-key constraints:
    "invoiceitems_invoiceid_fkey" FOREIGN KEY (invoiceid) REFERENCES invoices(invoiceid) ON DELETE CASCADE
```

PURPOSE:

This table contains detailed information about items within each invoice, with a unique itemid as the primary key. It includes attributes such as invoiceid, quantity, and rate (with precision up to two decimal places). Each invoiceid references the invoice table, and items are deleted if the associated invoice is removed, ensuring data integrity.

TABLE: invoices

```
InvoiceGenerator=# \d invoices
                                    Table "public.invoices"
   Column    |         Type          | Collation | Nullable |                  Default
-------------+-----------------------+-----------+----------+--------------------------------------------
 invoiceid   | integer               |           | not null | nextval('invoices_invoiceid_seq'::regclass)
 cid         | integer               |           |          |
 invoicedate | date                  |           | not null |
 gst_rate    | numeric(5,2)          |           | not null |
 raddress    | character varying(50) |           | not null |
 rgstno      | character varying(15) |           | not null |
Indexes:
    "invoices_pkey" PRIMARY KEY, btree (invoiceid)
    "invoices_rgstno_key" UNIQUE CONSTRAINT, btree (rgstno)
Foreign-key constraints:
    "invoices_cid_fkey" FOREIGN KEY (cid) REFERENCES users(cid)
Referenced by:
    TABLE "invoiceitems" CONSTRAINT "invoiceitems_invoiceid_fkey" FOREIGN KEY (invoiceid) REFERENCES invoices(invoiceid) ON DELETE CASC
ADE
```

PURPOSE:

This table stores invoice records with a unique invoiceid as the primary key. Attributes include cid (a reference to the users table), invoicedate, gst_rate, raddress (recipient address), and rgstno (recipient GST number, which must be unique). Each invoice links back to a user and is referenced in invoiceitems.

TABLE: company

```
InvoiceGenerator=# \d company
                        Table "public.company"
  Column  |          Type          | Collation | Nullable | Default
----------+------------------------+-----------+----------+---------
 cid      | integer                |           |          |
 cname    | character varying(50)  |           | not null |
 caddress | character varying(100) |           | not null |
 cphone   | character varying(10)  |           | not null |
 gstno    | character varying(15)  |           | not null |
Indexes:
    "company_gstno_key" UNIQUE CONSTRAINT, btree (gstno)
Foreign-key constraints:
    "company_cid_fkey" FOREIGN KEY (cid) REFERENCES users(cid)
```

PURPOSE:

This table holds information about companies, with cid as a foreign key linking to the users table. Key fields include cname (company name), caddress (company address), cphone, and gstno (company GST number, which must be unique). This structure allows each user to be associated with a specific company profile.

TABLE: users

```
InvoiceGenerator=# \d users
                                    Table "public.users"
  Column  |         Type          | Collation | Nullable |              Default
----------+-----------------------+-----------+----------+-----------------------------------
 cid      | integer               |           | not null | nextval('users_cid_seq'::regclass)
 name     | character varying(50) |           | not null |
 email    | character varying(50) |           | not null |
 password | character varying(10) |           | not null |
Indexes:
    "users_pkey" PRIMARY KEY, btree (cid)
    "users_password_key" UNIQUE CONSTRAINT, btree (password)
Referenced by:
    TABLE "company" CONSTRAINT "company_cid_fkey" FOREIGN KEY (cid) REFERENCES users(cid)
    TABLE "invoices" CONSTRAINT "invoices_cid_fkey" FOREIGN KEY (cid) REFERENCES users(cid)
```

PURPOSE:

This table stores user information, with a unique cid as the primary key. Key attributes include name, email, and password. The cid is referenced in both the company and invoices tables. Additionally, each password must be unique, enforced by a unique constraint, to maintain account security.

``````
``````

ER DIAGRAM

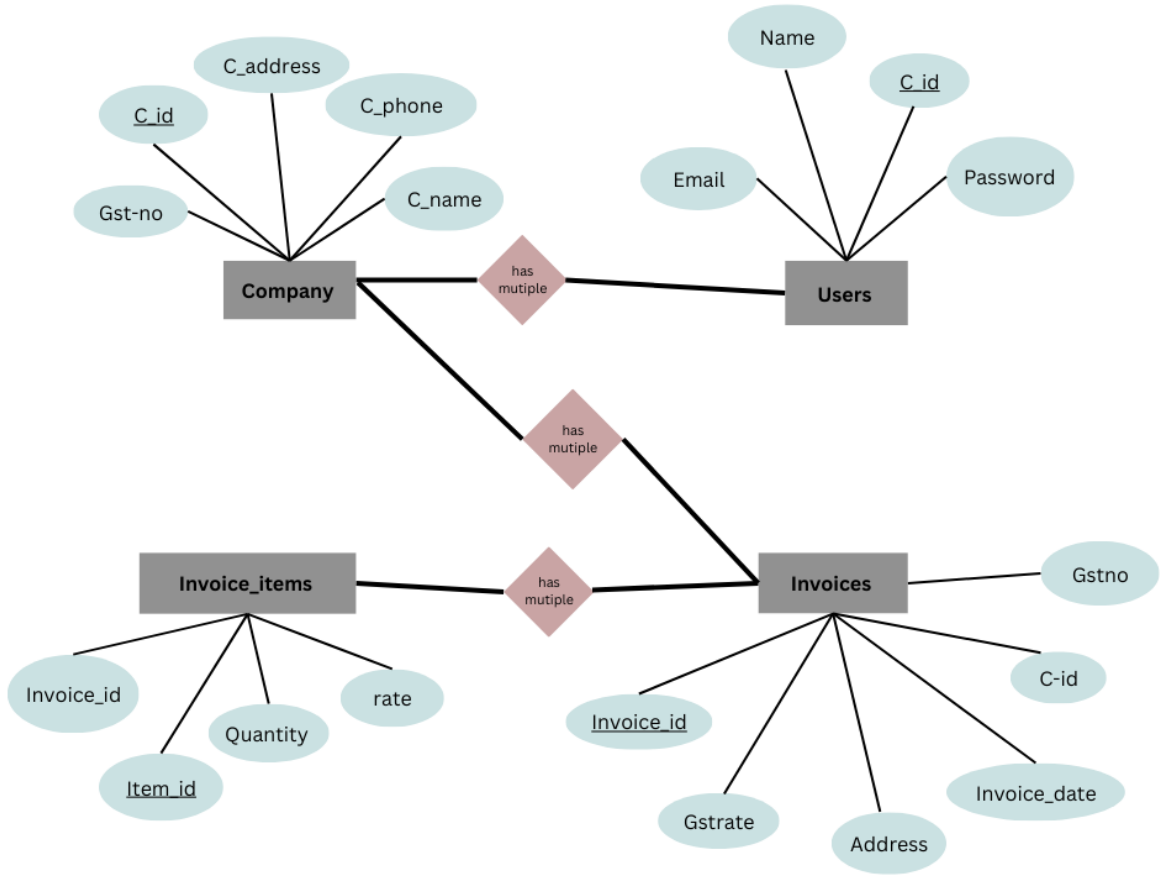

Name
C_id
Email
Password
C_address
C_id
C_phone
C_name
Gst-no
Company
has mutiple
Users
has mutiple
Invoice_items
has mutiple
Invoices
Gstno
Invoice_id
Quantity
rate
Item_id
Invoice_id
C-id
Gstrate
Address
Invoice_date

Table	Attribute	Key	Relationship
Company	cid	Primary Key (PK)	
	cname		
	caddress		
	cphone		
	gstno		
Users	cid	Primary Key (PK), FK to Company.cid	One-to-Many with Company
	name		
	email		
	password		
Invoices	invoiceid	Primary Key (PK)	
	cid	Foreign Key (FK) to Company.cid	One-to-Many with Company
	invoicedate		
	gst_rate		
	raddress		
	rgstno		
InvoiceItems	itemid	Primary Key (PK)	
	invoiceid	Foreign Key (FK) to Invoices.invoiceid	One-to-Many with Invoices
	quantity		
	rate		
	rate		

PROGRAM CODE

BACKEND CONNECTIVITY DESCRIPTION

The backend, built with Express.js in Node.js, manages invoices, users, and sharing features through RESTful endpoints for CRUD operations. It uses PostgreSQL to securely store user and invoice data. When a new invoice is created, the frontend sends form data to the backend via a POST request, which validates, saves, and returns an invoice ID or confirmation. User authentication is handled with JWT or sessions to ensure secure access.

For generating PDFs, the backend uses libraries like pdfkit or Puppeteer, enabling users to download or email the document via a dedicated route. Sharing options—such as email, WhatsApp, or social media—are supported by specific endpoints. On clicking "Share Invoice," the backend formats the invoice for the chosen platform, such as using Nodemailer for emails.

Buttons in the app streamline workflows: "Generate PDF" downloads a formatted PDF, "Review Invoice" opens a modal for final checks, and "Share Invoice" pre-fills messages on selected platforms. Each button leverages backend services to make invoice creation, review, and sharing smooth and efficient.

1.creating react app

```
npx create-react-app invoice-generator
cd invoice-generator
npm install axios react-router-dom formik yup
```

2. project structure

```
src/
├── components/
│   ├── CustomerDetails.js
│   ├── BillFrom.js
│   ├── InvoiceItems.js
│   ├── InvoiceSummary.js
│   └── App.js
├── App.css
└── index.js
```

3. Basic components

CustomerDetails.js

```
// src/components/CustomerDetails.js
import React from 'react';
import { useFormik } from 'formik';
import * as Yup from 'yup';


const CustomerDetails = ({ onChange }) => {
  const formik = useFormik({
    initialValues: {
      name: '',
      email: '',
      address: '',
      gstin: ''
    },
    validationSchema: Yup.object({
      name: Yup.string().required('Required'),
      email: Yup.string().email('Invalid email').required('Required'),
      address: Yup.string().required('Required'),
      gstin: Yup.string().required('Required')
    }),
    onSubmit: values => {
      onChange(values);
    }
  });


  return (
    <form onSubmit={formik.handleSubmit}>
      <input type="text" name="name" placeholder="Customer Name"
onChange={formik.handleChange} />
      {formik.errors.name && <div>{formik.errors.name}</div>}
      <input type="email" name="email" placeholder="Customer Email"
onChange={formik.handleChange} />
      {formik.errors.email && <div>{formik.errors.email}</div>}
      <input type="text" name="address" placeholder="Customer Address"
onChange={formik.handleChange} />
      {formik.errors.address && <div>{formik.errors.address}</div>}
      <input type="text" name="gstin" placeholder="Customer GSTIN"
onChange={formik.handleChange} />
      {formik.errors.gstin && <div>{formik.errors.gstin}</div>}
    </form>
  );
};
```

```
export default CustomerDetails
```

BillForm.js

```
// src/components/BillFrom.js
import React from 'react';
import { useFormik } from 'formik';
import * as Yup from 'yup';


const BillFrom = ({ onChange }) => {
   const formik = useFormik({
      initialValues: {
         companyName: '',
         email: '',
         address: '',
         gstin: ''
      },
      validationSchema: Yup.object({
         companyName: Yup.string().required('Required'),
         email: Yup.string().email('Invalid email').required('Required'),
         address: Yup.string().required('Required'),
         gstin: Yup.string().required('Required')
      }),
      onSubmit: values => {
         onChange(values);
      }
   });


   return (
      <form onSubmit={formik.handleSubmit}>
         <input type="text" name="companyName" placeholder="Company Name"
onChange={formik.handleChange} />
         {formik.errors.companyName && <div>{formik.errors.companyName}</div>}
         <input type="email" name="email" placeholder="Company Email"
onChange={formik.handleChange} />
         {formik.errors.email && <div>{formik.errors.email}</div>}
         <input type="text" name="address" placeholder="Company Address"
onChange={formik.handleChange} />
         {formik.errors.address && <div>{formik.errors.address}</div>}
         <input type="text" name="gstin" placeholder="Company GSTIN"
onChange={formik.handleChange} />
         {formik.errors.gstin && <div>{formik.errors.gstin}</div>}
      </form>
   );
};  export default BillForm;
```

InvoiceItems.js

```
// src/components/InvoiceItems.js
import React, { useState } from 'react';


const InvoiceItems = ({ items, setItems }) => {
  const [item, setItem] = useState({ name: '', description: '', price: '', quantity: '' });


  const addItem = () => {
    setItems([...items, item]);
    setItem({ name: '', description: '', price: '', quantity: '' });
  };


  return (
    <div>
      <input type="text" placeholder="Item Name" value={item.name} onChange={e =>
setItem({ ...item, name: e.target.value })} />
      <input type="text" placeholder="Description" value={item.description}
onChange={e => setItem({ ...item, description: e.target.value })} />
      <input type="number" placeholder="Price" value={item.price} onChange={e =>
setItem({ ...item, price: e.target.value })} />
      <input type="number" placeholder="Quantity" value={item.quantity} onChange={e
=> setItem({ ...item, quantity: e.target.value })} />
      <button type="button" onClick={addItem}>Add Item</button>
      <ul>
        {items.map((i, index) => (
          <li key={index}>{i.name} - {i.price} x {i.quantity}</li>
        ))}
      </ul>
    </div>
  );
};


export default InvoiceItems;
```

Invoice summary.js

```
// src/components/InvoiceSummary.js
import React from 'react';


const InvoiceSummary = ({ items, taxRate, discountRate }) => {
  const subtotal = items.reduce((acc, item) => acc + (item.price * item.quantity), 0);
  const tax = (subtotal * taxRate) / 100;
  const discount = (subtotal * discountRate) / 100;
  const total = subtotal + tax - discount;


  return (
    <div>
      <h3>Invoice Summary</h3>
      <p>Subtotal: {subtotal}</p>
      <p>Tax: {tax}</p>
      <p>Discount: {discount}</p>
      <h4>Total: {total}</h4>
    </div>
  );
};


export default InvoiceS
```

App.js

```
// src/components/App.js
import React, { useState } from 'react';
import CustomerDetails from './CustomerDetails';
import BillFrom from './BillFrom';
import InvoiceItems from './InvoiceItems';
import InvoiceSummary from './InvoiceSummary';


const App = () => {
  const [customerDetails, setCustomerDetails] = useState({});
  const [billFromDetails, setBillFromDetails] = useState({});
  const [items, setItems] = useState([]);
  const [taxRate, setTaxRate] = useState(0);
  const [discountRate, setDiscountRate] = useState(0);
  return (
    <div>
      <h1>Invoice Generator</h1>
```

```
            <CustomerDetails onChange={setCustomerDetails} />
            <BillFrom onChange={setBillFromDetails} />
            <InvoiceItems items={items} setItems={setItems} />
            <input type="number" placeholder="Tax Rate (%)" onChange={e =>
setTaxRate(e.target.value)} />
            <input type="number" placeholder="Discount Rate (%)" onChange={e =>
setDiscountRate(e.target.value)} />
            <InvoiceSummary items={items} taxRate={taxRate} discountRate={discountRate}
/>
        </div>
    );
  };


  export default App;
```

Back-End(Node.js & Express)

1.set up Express Application mkdir backend

```
  cd backend
  npm init -y
  npm install express pg body-parser cors nodemailer
```

PROJECT STRUCTURE

```
  backend/
  ├── server.js
  ├── routes/
  │   ├── customers.js
  │   ├── companies.js
  │   └── invoices.js
  └── db/
      └── index.js
```

DATABASE CONNECTION

```
  // backend/db/index.js
  const { Pool } = require('pg');


  const pool = new Pool({
    user: 'your_user',
    host: 'localhost',
    database: 'invoice_db',
    password: 'your_password',
    port: 5432,
  });


  module.exports = pool;
```

SERVER SETUP

```
// backend/server.js
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const customerRoutes = require('./routes/customers');
const companyRoutes = require('./routes/companies');
const invoiceRoutes = require('./routes/invoices');


const app = express();
app.use(cors());
app.use(bodyParser.json());


app.use('/api/customers', customerRoutes);
app.use('/api/companies', companyRoutes);
app.use('/api/invoices', invoiceRoutes);


const PORT = process.env.PORT || 5000;
app.listen(PORT, () => {
   console.log(`Server is running on port ${PORT}`);
});
```

RUNNING THE APPLICATION

1.Start the Backend

```
cd backend
node server.js
```

2.Start the Front_End

```
cd invoice-generator
npm start
```

FRONTEND DESCRIPTION

The front end of the Invoice Generator app, built with React and React-Bootstrap, provides a user-friendly, professional interface for creating, viewing, and sharing invoices. The main invoice form lets users input essential customer and company details with clearly labeled fields, while a sidebar offers quick settings for currency, tax, and discounts. The "Product Details" section allows users to add items with automatic total calculations via a custom hook.

Key features include a "Generate PDF" button, which formats the invoice in Times New Roman with a green header, a border, and sections for dates and digital signatures. Sharing options for email, WhatsApp, and Facebook, plus a "Review Invoice" preview modal, enhance accessibility and accuracy. The interface uses a modern glass morphism style, ensuring a clean, responsive layout across devices. This design creates an effective, visually appealing tool for seamless invoice management.

FRONT END (REACT)

```
import React, { useState, useEffect } from "react";
import { Button, Col, Container, Form, Row, Modal } from "react-bootstrap";
import { generatePDF } from "./pdf/pdfGenerator";
import InvoiceItem from "./reusable/InvoiceItem";
import "./InvoiceForm.css";
export default function InvoiceForm() {
  const [invoiceData, setInvoiceData] = useState({
    currency: "₹",
    invoiceNumber: 1,
    billTo: "",
    billToAddress: "",
    billToEmail: "",
    billToGSTIN: "",
    billFrom: "",
    billFromAddress: "",
    billFromEmail: "",
    billFromGSTIN: "",
    notes: "",
    taxRate: 8,
    discountRate: 15,
  });


  const [items, setItems] = useState([
    { id: 1, name: "", description: "", quantity: 1, price: 0 },
  ]);


  const [totals, setTotals] = useState({
```

```
  subTotal: 0.0,
  taxAmount: 0.0,
  discountAmount: 0.0,
  total: 0.0,
});

const [showModal, setShowModal] = useState(false);

// Fetch initial data
useEffect(() => {
  fetchInvoiceData();
}, []);

const fetchInvoiceData = async () => {
  try {
    const response = await axios.get("/api/invoice");
    setInvoiceData(response.data.invoiceData);
    setItems(response.data.items);
  } catch (error) {
    console.error("Error fetching invoice data:", error);
  }
};

// Update totals whenever items, tax, or discount changes
useEffect(() => {
  const calculateTotals = () => {
    const subTotal = items.reduce((total, item) => total + item.quantity * item.price, 0);
    const taxAmount = (subTotal * invoiceData.taxRate) / 100;
    const discountAmount = (subTotal * invoiceData.discountRate) / 100;
    const total = subTotal + taxAmount - discountAmount;

    setTotals({
      subTotal: subTotal.toFixed(2),
      taxAmount: taxAmount.toFixed(2),
      discountAmount: discountAmount.toFixed(2),
      total: total.toFixed(2),
    });
  };
  calculateTotals();
}, [items, invoiceData.taxRate, invoiceData.discountRate]);

  const handleChange = (event) => {
  const { name, value } = event.target;
```

```
  setInvoiceData((prevData) => ({ ...prevData, [name]: value }));
 };

 const addItem = () => {
  const newItem = { id: Date.now(), name: "", description: "", quantity: 1, price: 0 };
  setItems((prevItems) => [...prevItems, newItem]);
 };

 const updateItem = (id, field, value) => {
  const updatedItems = items.map((item) =>
   item.id === id ? { ...item, [field]: value } : item
  );
  setItems(updatedItems);
 };

 const deleteItem = (id) => {
  const filteredItems = items.filter((item) => item.id !== id);
  setItems(filteredItems);
 };

 const handleSaveInvoice = async () => {
  try {
   await axios.post("/api/invoice", { invoiceData, items });
   alert("Invoice saved successfully!");
  } catch (error) {
   console.error("Error saving invoice:", error);
  }
 };

 const handleGeneratePDF = () => {
  generatePDF(invoiceData, items, totals);
 };

 const handleShare = () => {
  const invoiceDetails = `Invoice Number: ${invoiceData.invoiceNumber}\nTotal:
${totals.total} ${invoiceData.currency}\nBill To: ${invoiceData.billTo}\nBill From:
${invoiceData.billFrom}`;

  const emailBody = encodeURIComponent(invoiceDetails);
  const emailSubject = encodeURIComponent("Invoice Details");
 return (
  <Container className="invoice-container">
   <h1 className="text-center mb-4">Invoice Generator</h1>
```

```
    <Form onSubmit={(e) => e.preventDefault()} style={{ width: "100%", maxWidth:
"900px" }}>
      <Row>
        <Col md={8} lg={9}>
          <div className="mb-3">
            <span className="fw-bold">Current Date: </span>
            <span className="current-date">{new Date().toLocaleDateString()}</span>
            <span className="fw-bold ms-4">Invoice Number: </span>
            <span className="current-date">{invoiceData.invoiceNumber}</span>
          </div>
          <hr className="my-4" />
          <Row className="mb-5">
            <Col>
              <Form.Label className="fw-bold">Customer Details:</Form.Label>
              <Form.Group className="my-2">
                <Form.Control
                  placeholder="Enter Name"
                  value={invoiceData.billTo}
                  type="text"
                  name="billTo"
                  onChange={handleChange}
                  required
                />
              </Form.Group>
              <Form.Group className="my-2">
                <Form.Control
                  placeholder="Enter Email"
                  value={invoiceData.billToEmail}
                  type="email"
                  name="billToEmail"
                  onChange={handleChange}
                />
              </Form.Group>
              <Form.Group className="my-2">
                <Form.Control
                  placeholder="Enter Address"
                  value={invoiceData.billToAddress}
                  type="text"
                  name="billToAddress"
                  onChange={handleChange}
                  required
                />
              </Form.Group>
              <Form.Group className="my-2">
                <Form.Control
                  placeholder="Enter GSTIN"
```

```
          value={invoiceData.billToGSTIN}
          type="text"
          name="billToGSTIN"
          onChange={handleChange}
        />
      </Form.Group>
    </Col>
    <Col>
      <Form.Label className="fw-bold">Bill From:</Form.Label>
      <Form.Group className="my-2">
        <Form.Control
          placeholder="Enter Company Name"
          value={invoiceData.billFrom}
          type="text"
          name="billFrom"
          onChange={handleChange}
          required
        />
      </Form.Group>
      <Form.Group className="my-2">
        <Form.Control
          placeholder="Enter Email"
          value={invoiceData.billFromEmail}
          type="email"
          name="billFromEmail"
          onChange={handleChange}
          required
        />
      </Form.Group>
      <Form.Group className="my-2">
        <Form.Control
          placeholder="Enter Address"
          value={invoiceData.billFromAddress}
          type="text"
          name="billFromAddress"
          onChange={handleChange}
          required
        />
      </Form.Group>
      <Form.Group className="my-2">
        <Form.Control
          placeholder="Enter GSTIN"
          value={invoiceData.billFromGSTIN}
          type="text"
          name="billFromGSTIN"
          onChange={handleChange}
```

```
              required
            />
          </Form.Group>
        </Col>
      </Row>


      {/* Product Details Table */}
      <InvoiceItem items={items} setItems={setItems} addItem={addItem}
updateItem={updateItem} deleteItem={deleteItem} currency={invoiceData.currency} />


      <hr className="my-4" />
    </Col>


    <Col md={4} lg={3}>
      <div className="text-center">
        <Button variant="primary" onClick={handleGeneratePDF} className="d-block w-
100 my-2">
          Generate PDF
        </Button>
        <Button variant="secondary" onClick={() => setShowModal(true)} className="d-
block w-100 my-2">
          Review Invoice
        </Button>
        <Button variant="success" onClick={handleShare} className="d-block w-100 my-
2">
          Share Invoice
        </Button>
        <Button variant="info" onClick={handleSaveInvoice} className="d-block w-100
my-2">
          Save Invoice
        </Button>
        <hr />
        <Form.Group className="my-3">
          <Form.Label className="fw-bold">Currency:</Form.Label>
          <Form.Select
            value={invoiceData.currency}
            onChange={(e) => setInvoiceData({ ...invoiceData, currency: e.target.value })}
          >
            <option>₹</option>
            <option>$</option>
            <option>€</option>
            <option>£</option>
          </Form.Select>
        </Form.Group>
        <Form.Group className="my-3">
```

```
            <Form.Label className="fw-bold">Discount Rate (%):</Form.Label>
            <Form.Control
              type="number"
              min="0"
              value={invoiceData.discountRate}
              onChange={(e) => setInvoiceData({ ...invoiceData, discountRate:
parseFloat(e.target.value) || 0 })}
            />
          </Form.Group>
          <Form.Group className="my-3">
            <Form.Label className="fw-bold">Tax Rate (%):</Form.Label>
            <Form.Control
              type="number"
              min="0"
              value={invoiceData.taxRate}
              onChange={(e) => setInvoiceData({ ...invoiceData, taxRate:
parseFloat(e.target.value) || 0 })}
            />
          </Form.Group>
          <hr />
          <div className="text-center">
            <div className="my-2">
              <strong>Sub Total:</strong> {totals.subTotal} {invoiceData.currency}
            </div>
            <div className="my-2">
              <strong>Tax:</strong> {totals.taxAmount} {invoiceData.currency}
            </div>
            <div className="my-2">
              <strong>Discount:</strong> -{totals.discountAmount} {invoiceData.currency}
            </div>
            <div className="my-2">
              <strong>Total:</strong> {totals.total} {invoiceData.currency}
            </div>
          </div>
        </div>
      </Col>
    </Row>
    {/* Review Modal */}
    <Modal show={showModal} onHide={() => setShowModal(false)} centered>
      <Modal.Header closeButton>
        <Modal.Title>Invoice Preview</Modal.Title>
      </Modal.Header>
      <Modal.Body>
        <div>
          <strong>Invoice Number:</strong> {invoiceData.invoiceNumber}
          <hr />
```

```
            {items.map((item) => (
              <div key={item.id}>
                <p>{item.name} (x{item.quantity}) - {item.price * item.quantity}
{invoiceData.currency}</p>
              </div>
            ))}
            <hr />
            <p><strong>Total Amount:</strong> {totals.total} {invoiceData.currency}</p>
          </div>
        </Modal.Body>
        <Modal.Footer>
          <Button variant="secondary" onClick={() => setShowModal(false)}>
            Close
          </Button>
        </Modal.Footer>
      </Modal>
    </Form>
  </Container>
 );
}
```

```
import React, { useState, useEffect } from "react";
import { Button, Col, Container, Form, Row, Modal } from "react-bootstrap";
import { generatePDF } from "./pdf/pdfGenerator";
import InvoiceItem from "./reusable/InvoiceItem";
import "./InvoiceForm.css";

export default function InvoiceForm() {
 const [invoiceData, setInvoiceData] = useState({
  currency: "₹",
  invoiceNumber: 1,
  billTo: "",
  billToAddress: "",
  billToEmail: "",
  billToGSTIN: "",
  billFrom: "",
  billFromAddress: "",
  billFromEmail: "",
  billFromGSTIN: "",
  notes: "",
  taxRate: 8,
  discountRate: 15,
 });

 const [items, setItems] = useState([
  { id: 1, name: "", description: "", quantity: 1, price: 0 },
 ]);

 const [totals, setTotals] = useState({
  subTotal: 0.0,
  taxAmount: 0.0,
  discountAmount: 0.0,
```

```
  total: 0.0,
});

const [showModal, setShowModal] = useState(false);

// Fetch initial data
useEffect(() => {
  fetchInvoiceData();
}, []);

const fetchInvoiceData = async () => {
  try {
    const response = await axios.get("/api/invoice");
    setInvoiceData(response.data.invoiceData);
    setItems(response.data.items);
  } catch (error) {
    console.error("Error fetching invoice data:", error);
  }
};

// Update totals whenever items, tax, or discount changes
useEffect(() => {
  const calculateTotals = () => {
    const subTotal = items.reduce((total, item) => total + item.quantity * item.price, 0);
    const taxAmount = (subTotal * invoiceData.taxRate) / 100;
    const discountAmount = (subTotal * invoiceData.discountRate) / 100;
    const total = subTotal + taxAmount - discountAmount;

    setTotals({
      subTotal: subTotal.toFixed(2),
      taxAmount: taxAmount.toFixed(2),
```

```
    discountAmount: discountAmount.toFixed(2),
    total: total.toFixed(2),
   });
  };
  calculateTotals();
}, [items, invoiceData.taxRate, invoiceData.discountRate]);

// Input change handler
const handleChange = (event) => {
  const { name, value } = event.target;
  setInvoiceData((prevData) => ({ ...prevData, [name]: value }));
};

// CRUD Operations
const addItem = () => {
  const newItem = { id: Date.now(), name: "", description: "", quantity: 1, price: 0 };
  setItems((prevItems) => [...prevItems, newItem]);
};

const updateItem = (id, field, value) => {
  const updatedItems = items.map((item) =>
   item.id === id ? { ...item, [field]: value } : item
  );
  setItems(updatedItems);
};

const deleteItem = (id) => {
  const filteredItems = items.filter((item) => item.id !== id);
  setItems(filteredItems);
};
```

```
 const handleSaveInvoice = async () => {
  try {
   await axios.post("/api/invoice", { invoiceData, items });
   alert("Invoice saved successfully!");
  } catch (error) {
   console.error("Error saving invoice:", error);
  }
 };

 const handleGeneratePDF = () => {
  generatePDF(invoiceData, items, totals);
 };

 const handleShare = () => {
  const invoiceDetails = `Invoice Number: ${invoiceData.invoiceNumber}\nTotal: ${totals.total} ${invoiceData.currency}\nBill To: ${invoiceData.billTo}\nBill From: ${invoiceData.billFrom}`;

  // Email share
  const emailBody = encodeURIComponent(invoiceDetails);
  const emailSubject = encodeURIComponent("Invoice Details");
  const emailLink = `mailto:${invoiceData.billToEmail}?subject=${emailSubject}&body=${emailBody}`;

  // WhatsApp share
  const whatsappLink = `https://wa.me/?text=${encodeURIComponent(invoiceDetails)}`;

  // Facebook share
  const facebookLink = `https://www.facebook.com/sharer/sharer.php?u=${encodeURIComponent(window.location.href)}`;

  // Open the share links
```

```
 window.open(emailLink, "_self"); // Open mail client
 window.open(whatsappLink, "_blank"); // Open WhatsApp
 window.open(facebookLink, "_blank"); // Open Facebook
};

return (
 <Container className="invoice-container">
  <h1 className="text-center mb-4">Invoice Generator</h1>
  <Form onSubmit={(e) => e.preventDefault()} style={{ width: "100%", maxWidth: "900px" }}>
   <Row>
    <Col md={8} lg={9}>
     <div className="mb-3">
      <span className="fw-bold">Current Date: </span>
      <span className="current-date">{new Date().toLocaleDateString()}</span>
      <span className="fw-bold ms-4">Invoice Number: </span>
      <span className="current-date">{invoiceData.invoiceNumber}</span>
     </div>
     <hr className="my-4" />
     <Row className="mb-5">
      <Col>
       <Form.Label className="fw-bold">Customer Details:</Form.Label>
       <Form.Group className="my-2">
        <Form.Control
         placeholder="Enter Name"
         value={invoiceData.billTo}
         type="text"
         name="billTo"
         onChange={handleChange}
         required
        />
       </Form.Group>
```

```
 <Form.Group className="my-2">
  <Form.Control
   placeholder="Enter Email"
   value={invoiceData.billToEmail}
   type="email"
   name="billToEmail"
   onChange={handleChange}
  />
 </Form.Group>
 <Form.Group className="my-2">
  <Form.Control
   placeholder="Enter Address"
   value={invoiceData.billToAddress}
   type="text"
   name="billToAddress"
   onChange={handleChange}
   required
  />
 </Form.Group>
 <Form.Group className="my-2">
  <Form.Control
   placeholder="Enter GSTIN"
   value={invoiceData.billToGSTIN}
   type="text"
   name="billToGSTIN"
   onChange={handleChange}
  />
 </Form.Group>
</Col>
<Col>
 <Form.Label className="fw-bold">Bill From:</Form.Label>
```

```
<Form.Group className="my-2">
 <Form.Control
  placeholder="Enter Company Name"
  value={invoiceData.billFrom}
  type="text"
  name="billFrom"
  onChange={handleChange}
  required
 />
</Form.Group>
<Form.Group className="my-2">
 <Form.Control
  placeholder="Enter Email"
  value={invoiceData.billFromEmail}
  type="email"
  name="billFromEmail"
  onChange={handleChange}
  required
 />
</Form.Group>
<Form.Group className="my-2">
 <Form.Control
  placeholder="Enter Address"
  value={invoiceData.billFromAddress}
  type="text"
  name="billFromAddress"
  onChange={handleChange}
  required
 />
</Form.Group>
<Form.Group className="my-2">
```

```
            <Form.Control
              placeholder="Enter GSTIN"
              value={invoiceData.billFromGSTIN}
              type="text"
              name="billFromGSTIN"
              onChange={handleChange}
              required
             />
            </Form.Group>
           </Col>
          </Row>

          {/* Product Details Table */}
          <InvoiceItem items={items} setItems={setItems} addItem={addItem}
updateItem={updateItem} deleteItem={deleteItem} currency={invoiceData.currency} />

          <hr className="my-4" />
         </Col>

         <Col md={4} lg={3}>
          <div className="text-center">
           <Button variant="primary" onClick={handleGeneratePDF} className="d-block w-100
my-2">
            Generate PDF
           </Button>
           <Button variant="secondary" onClick={() => setShowModal(true)} className="d-block
w-100 my-2">
            Review Invoice
           </Button>
           <Button variant="success" onClick={handleShare} className="d-block w-100 my-2">
            Share Invoice
           </Button>
```

```
<Button variant="info" onClick={handleSaveInvoice} className="d-block w-100 my-2">
 Save Invoice
</Button>
<hr />
<Form.Group className="my-3">
 <Form.Label className="fw-bold">Currency:</Form.Label>
 <Form.Select
  value={invoiceData.currency}
  onChange={(e) => setInvoiceData({ ...invoiceData, currency: e.target.value })}
 >
  <option>₹</option>
  <option>$</option>
  <option>€</option>
  <option>£</option>
 </Form.Select>
</Form.Group>
<Form.Group className="my-3">
 <Form.Label className="fw-bold">Discount Rate (%):</Form.Label>
 <Form.Control
  type="number"
  min="0"
  value={invoiceData.discountRate}
  onChange={(e) => setInvoiceData({ ...invoiceData, discountRate:
parseFloat(e.target.value) || 0 })}
 />
</Form.Group>
<Form.Group className="my-3">
 <Form.Label className="fw-bold">Tax Rate (%):</Form.Label>
 <Form.Control
  type="number"
  min="0"
```

```
            value={invoiceData.taxRate}
            onChange={(e) => setInvoiceData({ ...invoiceData, taxRate: parseFloat(e.target.value) ||
0 })}
           />
          </Form.Group>
          <hr />
          <div className="text-center">
           <div className="my-2">
            <strong>Sub Total:</strong> {totals.subTotal} {invoiceData.currency}
           </div>
           <div className="my-2">
            <strong>Tax:</strong> {totals.taxAmount} {invoiceData.currency}
           </div>
           <div className="my-2">
            <strong>Discount:</strong> -{totals.discountAmount} {invoiceData.currency}
           </div>
           <div className="my-2">
            <strong>Total:</strong> {totals.total} {invoiceData.currency}
           </div>
          </div>
         </div>
        </Col>
       </Row>

       {/* Review Modal */}
       <Modal show={showModal} onHide={() => setShowModal(false)} centered>
        <Modal.Header closeButton>
         <Modal.Title>Invoice Preview</Modal.Title>
        </Modal.Header>
        <Modal.Body>
         <div>
```

```
          <strong>Invoice Number:</strong> {invoiceData.invoiceNumber}
          <hr />
          {items.map((item) => (
            <div key={item.id}>
              <p>{item.name} (x{item.quantity}) - {item.price * item.quantity}
{invoiceData.currency}</p>
            </div>
          ))}
          <hr />
          <p><strong>Total Amount:</strong> {totals.total} {invoiceData.currency}</p>
        </div>
      </Modal.Body>
      <Modal.Footer>
        <Button variant="secondary" onClick={() => setShowModal(false)}>
          Close
        </Button>
      </Modal.Footer>
    </Modal>
  </Form>
 </Container>
 );
}
```

Pdf generator code

```
import jsPDF from "jspdf";
import "jspdf-autotable";

export const generatePDF = (invoiceData, items, totals) => {
 const doc = new jsPDF();

 // Set Times New Roman font and create a border
 doc.setFont("times", "normal");
 doc.rect(10, 10, 190, 277); // Creates a border

 // 'INVOICE' as heading
 doc.setFontSize(20);
 doc.text("INVOICE", 105, 20, { align: "center" });

 // Company Details
 doc.setFontSize(12);
 doc.text(`From: ${invoiceData.billFrom}`, 14, 30);
 doc.text(`Address: ${invoiceData.billFromAddress}`, 14, 36);
 doc.text(`Email: ${invoiceData.billFromEmail}`, 14, 42);
 doc.text(`GSTIN: ${invoiceData.billFromGSTIN}`, 14, 48);

 // Customer Details
 doc.text(`Bill To: ${invoiceData.billTo}`, 14, 60);
 doc.text(`Address: ${invoiceData.billToAddress}`, 14, 66);
 doc.text(`Email: ${invoiceData.billToEmail}`, 14, 72);
 doc.text(`GSTIN: ${invoiceData.billToGSTIN}`, 14, 78);

 // Product details in a table with green header
 doc.autoTable({
  head: [['Product Name', 'Description', 'Price', 'Quantity', 'Total']],
```

```
  body: items.map(item => [
    item.name,
    item.description,
    item.price,
    item.quantity,
    (item.price * item.quantity).toFixed(2)
  ]),
  startY: 90,
  headStyles: { fillColor: [0, 128, 0] }, // Green header
 });

 const yPosition = doc.autoTable.previous.finalY;

 // Total calculations in a tabular format
 doc.text(`Subtotal: ${totals.subTotal}`, 14, yPosition + 10);
 doc.text(`Tax: ${totals.taxAmount}`, 14, yPosition + 16);
 doc.text(`Discount: ${totals.discountAmount}`, 14, yPosition + 22);
 doc.text(`Total: ${totals.total}`, 14, yPosition + 28);



 // Digital Signature
 doc.text(`Signature: _____`, 14, yPosition + 50);

 doc.save(`Invoice-${invoiceData.invoiceNumber}.pdf`);
};
```

App.js

```
import React, { useState } from "react";
import './App.css';
import { Container } from "react-bootstrap";
import InvoiceForm from './components/InvoiceForm';

function App() {
 const [count, setCount] = useState(0);

 return (
  <div className="App d-flex flex-column align-items-center justify-content-center">
   <Container>
    <InvoiceForm />
   </Container>
  </div>
 );
}

export default App;
```

INVOICE FORM.CSS

```
import React from "react";
import { Table, Button } from "react-bootstrap";
import { FaTrash } from "react-icons/fa"; // Import trash icon from react-icons

export default function InvoiceItem({ items, setItems, currency }) {
  const handleChange = (index, event) => {
    const updatedItems = [...items];
    updatedItems[index][event.target.name] = event.target.value;
    setItems(updatedItems);
  };

  const handleAddItem = () => {
    setItems([
      ...items,
      { id: items.length + 1, name: "", description: "", price: 0, quantity: 1 },
    ]);
  };

  const handleRemoveItem = (index) => {
    const updatedItems = items.filter((_, i) => i !== index);
    setItems(updatedItems);
  };

  return (
    <div>
      <Table striped bordered hover>
        <thead>
          <tr>
            <th>Item</th>
```

```
    <th>Description</th>
    <th>Price</th>
    <th>Quantity</th>
    <th>Action</th>
  </tr>
</thead>
<tbody>
  {items.map((item, index) => (
    <tr key={item.id}>
      <td>
        <input
          type="text"
          placeholder="Item Name"
          name="name"
          value={item.name}
          onChange={(event) => handleChange(index, event)}
          className="form-control"
        />
      </td>
      <td>
        <input
          type="text"
          placeholder="Description"
          name="description"
          value={item.description}
          onChange={(event) => handleChange(index, event)}
          className="form-control"
        />
      </td>
      <td>
```

```
          <input
            type="number"
            placeholder="Price"
            name="price"
            value={item.price}
            onChange={(event) => handleChange(index, event)}
            className="form-control"
          />
        </td>
        <td>
          <input
            type="number"
            placeholder="Qty"
            name="quantity"
            value={item.quantity}
            onChange={(event) => handleChange(index, event)}
            className="form-control"
          />
        </td>
        <td>
          <Button variant="danger" onClick={() => handleRemoveItem(index)}>
            <FaTrash /> {/* Trash icon */}
          </Button>
        </td>
      </tr>
    ))}
  </tbody>
</Table>
<Button variant="primary mt-2" onClick={handleAddItem}>
  Add Item
```

```
      </Button>
    </div>
  );
}
```

INVOICE FORM.CSS

```
import React from "react";
import { Table, Button } from "react-bootstrap";
import { FaTrash } from "react-icons/fa"; // Import trash icon from react-icons

export default function InvoiceItem({ items, setItems, currency }) {
  const handleChange = (index, event) => {
    const updatedItems = [...items];
    updatedItems[index][event.target.name] = event.target.value;
    setItems(updatedItems);
  };

  const handleAddItem = () => {
    setItems([
      ...items,
      { id: items.length + 1, name: "", description: "", price: 0, quantity: 1 },
    ]);
  };

  const handleRemoveItem = (index) => {
    const updatedItems = items.filter((_, i) => i !== index);
    setItems(updatedItems);
  };

  return (
```

```
<div>
  <Table striped bordered hover>
    <thead>
      <tr>
        <th>Item</th>
        <th>Description</th>
        <th>Price</th>
        <th>Quantity</th>
        <th>Action</th>
      </tr>
    </thead>
    <tbody>
      {items.map((item, index) => (
        <tr key={item.id}>
          <td>
            <input
              type="text"
              placeholder="Item Name"
              name="name"
              value={item.name}
              onChange={(event) => handleChange(index, event)}
              className="form-control"
            />
          </td>
          <td>
            <input
              type="text"
              placeholder="Description"
              name="description"
              value={item.description}
```

```
        onChange={(event) => handleChange(index, event)}
        className="form-control"
      />
    </td>
    <td>
      <input
        type="number"
        placeholder="Price"
        name="price"
        value={item.price}
        onChange={(event) => handleChange(index, event)}
        className="form-control"
      />
    </td>
    <td>
      <input
        type="number"
        placeholder="Qty"
        name="quantity"
        value={item.quantity}
        onChange={(event) => handleChange(index, event)}
        className="form-control"
      />
    </td>
    <td>
      <Button variant="danger" onClick={() => handleRemoveItem(index)}>
        <FaTrash /> {/* Trash icon */}
      </Button>
    </td>
  </tr>
```

```
          ))}
        </tbody>
      </Table>
      <Button variant="primary mt-2" onClick={handleAddItem}>
        Add Item
      </Button>
    </div>
  );
}
```

RESULT AND DISCUSSION

Usage Guide

This guide will help you navigate the application and perform key actions like creating invoices, adding details, calculating totals, and sharing your invoices. Follow these steps to streamline your workflow.

Navigating the Application

1. **Home Screen**: Access features like creating new invoices, viewing past ones, and managing settings.
2. **Menu**: Use the menu bar to quickly switch between sections.
3. **Search & Filters**: Find specific invoices or customers using the search tool.

Creating an Invoice

1. From the home screen, click **Create Invoice**.
2. Select an existing customer or enter new customer details.
3. Add items, services, or products for the invoice.
4. Review the invoice summary before proceeding.

Filling Customer and Sender Details

- **Customer Information**: Add customer name, address, and contact details.
- **Sender Information**: Ensure your business name, address, and payment details are correct. This can usually be pre-set in the profile settings for future invoices.

Adding Invoice Items

1. Click **Add Item**.
2. Fill in the item name, description, quantity, price, and tax details.
3. Repeat the process for additional items or services.

Calculating Totals

The app automatically calculates:

- **Subtotal**: Based on item quantity and price.
- **Tax**: Applied to the subtotal if specified.
- **Total**: The sum of subtotal and taxes.

Generating and Sharing Invoices

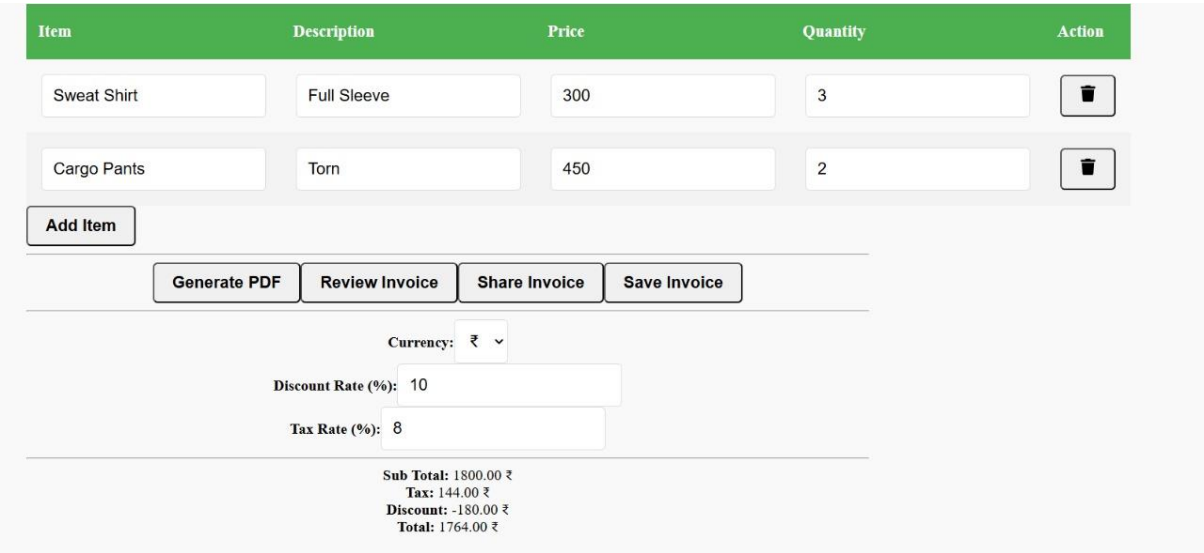
1. Once all details are filled in, click **Generate Invoice**.
2. Choose a format (PDF, email, or other options).
3. Click **Share** to send the invoice to your customer via email or other channels.

Error Handling and Troubleshooting

- **Missing Fields**: If any required fields are incomplete, the app will highlight them and prevent invoice generation until they are filled.
- **Calculation Issues**: Check if the item prices and taxes are correctly entered.
- **Unable to Share**: Ensure internet connectivity or check your app permissions if you can't send the invoice.

BASIC FRONT-END LAYOUT

Customer details



Invoice description page

The InvoiceItem component manages a dynamic list of invoice items, each with input fields for name, description, price, and quantity. Users can add items with default values or remove items using a "Remove" button, which filters out the selected row. The component uses React state to track changes, ensuring updates are captured in real-time. Styled with Bootstrap's Table and Button components, it provides a clean, responsive layout for easy invoice management.

Preview of generated Invoice

Invoice Preview

Invoice Number: 1

Sweat Shirt (x3) - 900 ₹

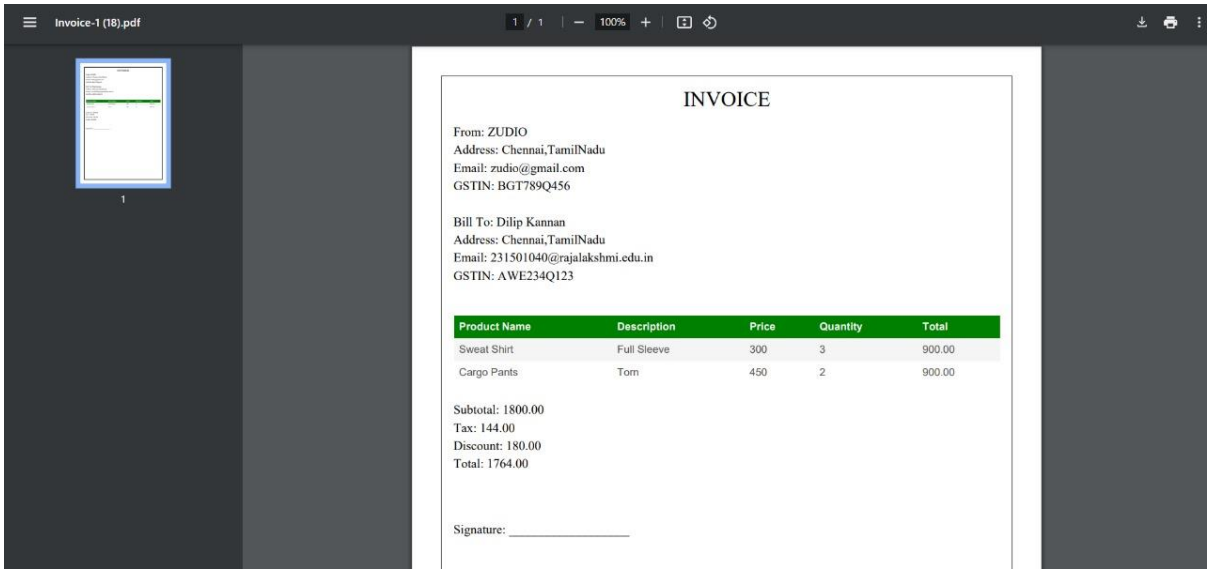
Cargo Pants (x2) - 900 ₹

Total Amount: 1764.00 ₹

Close

Generated PDF Content

The generatePDF function is designed to create a PDF invoice using the jsPDF library. It begins by setting the font to Times New Roman and creating a border around the page. The function then adds a centered "INVOICE" heading followed by the company details, such as the name, address, email, and GSTIN. The customer's details, including their name, address, email, and GSTIN, are also displayed. The function uses jsPDF methods to format and position these details clearly on the page.



Conclusion:

Upon completing the "Invoice Generator" project, we are confident that it will effectively address the issues present in traditional invoicing methods. By automating and streamlining the invoicing process, this system minimizes human errors, increases efficiency, and reduces manual effort. The primary aim of the project is to offer MSME users an easy-to-use, online, and instant invoicing solution that can be customized to fit diverse business needs.

Each invoice is assigned a unique identifier to ensure accurate access and efficient record management. All records are securely stored in a database, allowing for quick data retrieval and simplified navigation. Editing and updating invoice details are made straightforward, with intuitive options to update fields as needed.

The "Invoice Generator" also provides the added advantage of auto filling details for frequent clients, saving time and improving accuracy. By reducing repetitive tasks, this system enhances productivity and frees up users to focus on other business-critical activities.

With this solution, we are confident that users will experience a smoother, more reliable invoicing process that supports their operational goals and facilitates growth in an increasingly digital business landscape.