BizNex: E-commerce Simplified

Mini project report submitted in partial fulfilment of the requirements for the award of the degree of

Bachelor of Technology
in
Computer Science & Engineering

Submitted by
Abhiram Biju(FIT22CS006)
Alan Biju(FIT22CS017)
Bhoumik B Eugene(FIT22CS054)
Blesson M V(FIT22CS055)

Federal Institute of Science And Technology (FISAT)®
Angamaly, Ernakulam

Affiliated to
APJ Abdul Kalam Technological University
CET Campus, Thiruvananthapuram

April 2025

FEDERAL INSTITUTE OF SCIENCE AND TECHNOLOGY
(FISAT)
Mookkannoor(P.O), Angamaly-683577

CERTIFICATE

This is to certify that the Mini project report for the project entitled ***BizNex*** is a bonafide report of the project presented during VIth semester (CSD334 - Mini Project) by **Abhiram Biju (FIT22CS006)** in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology (B.Tech) in Computer Science & Engineering during the academic year 2024-25.

Project Coordinator	**Project Guide**	**Head of the Department**
Ms Anitha T Nair	Dr.Paul P Mathai	Dr.Paul P Mathai

ABSTRACT

BizNex is a comprehensive, web-based platform developed to assist small-scale businesses in streamlining their operations, managing internal processes, and expanding their digital presence. Designed with an intuitive and responsive interface, the system seamlessly integrates backend services powered by Node.js and frontend functionalities built with React.js, providing users with a unified digital workspace.

The application supports various modules, including employee management, product and category organization, job listings, and client authentication, all of which are secured through robust session handling and authentication mechanisms. Users can interact with real-time dashboards, manage inventory, publish openings, and oversee team roles with minimal friction, encouraging productivity and transparency across departments.

BizNex further empowers businesses by offering analytics and automation through integrated charts, task tracking, and notification systems. The backend, deployed via Render, communicates securely with the PostgreSQL database hosted on AWS EC2, while static assets and uploads are efficiently handled through Amazon S3. The frontend is deployed using Vercel to ensure rapid delivery and performance.

Overall, BizNex serves as a digital backbone for emerging businesses by reducing manual workload, minimizing management overhead, and enabling smarter decision-making. Its modular architecture and scalable design make it an ideal solution for modern business needs in a fast-paced digital economy.

Contribution by Author

My primary contribution to the project was focused on building and integrating backend APIs, managing authentication mechanisms, and implementing various features like OTP verification, salary and job modules, and database operations. I contributed significantly to both the planning and execution phases by setting up AWS S3[12] buckets for file storage, handling user authentication using Google OAuth, and generating invoices. I also assisted in debugging, transitioning from session to token authentication, and worked on subdomain setup and backend deployment. I played a key role in ensuring smooth communication between frontend and backend systems through well-structured APIs. I actively participated in resolving session-related errors and performed security enhancements following a ransomware incident. My consistent involvement in API integration and infrastructure setup helped streamline the overall development workflow.

Abhiram Biju

Contribution by Author

My role mainly involved API development, data handling, and database management. I helped in creating the master database, dynamic client database generation, and API calls for several modules including salary, employee, and job listings. I also focused on improving security post-ransomware attack, integrating payment methods, and debugging. Apart from the backend, I coordinated with the frontend team to ensure proper client-side handling of data and seamless authentication workflows. I was responsible for ensuring data integrity, fixing bugs, and validating database consistency. Additionally, I contributed to testing and refining API endpoints, ensuring they met performance and usability expectations. My proactive involvement in backend architecture greatly contributed to the stability and scalability of the system.

Alan Biju

Contribution by Author

I was primarily responsible for the frontend development of the website. I worked on building the homepage, salary, category, billing, and barcode modules. Additionally, I helped in React integration, dashboard visualization, and frontend document storage. My role also included helping with the second project review presentation and working closely with the backend team to ensure seamless API integration. I took charge of making the user interface intuitive and responsive, implementing designs that prioritized accessibility and clarity. I contributed significantly to the frontend setup for document management and invoice generation modules. My consistent focus on visual elements, combined with efficient data binding from backend services, enhanced the overall user experience of the platform.

Bhoumik B Eugene

Contribution by Author

My contributions covered both frontend and backend development. I helped set up the initial AWS database server, created and managed various API endpoints including authentication, product handling, billing, and finance management. I also led the efforts in implementing OTP verification and transitioning session-based login to token-based authentication. Furthermore, I worked on security improvements after the ransomware incident and handled subdomain management and deployment tasks. I contributed ideas toward the core design of the application and participated in troubleshooting performance bottlenecks. I ensured the backend was production-ready and scalable, optimizing it for real-world use. My work was crucial in finalizing the AWS setup and ensuring our backend services were reliably deployed and functional.

Blesson M V

ACKNOWLEDGMENT

We extend our heartfelt gratitude to all the team members whose dedication and hard work have been instrumental in the success of Project BizNex. We sincerely acknowledge our Principal, Prof. Dr. Jacob Thomas V for the continuous guidance and unwavering support throughout the project's journey.
We also sincerely appreciate our Head of the department as well as our project guide, Dr. Paul P. Mathai, whose expertise and encouragement have been invaluable in steer ing us toward success.
Furthermore, we extend our gratitude to our project mentors, Ms. Anitha T. Nair, Assistant Professor of the Department of Computer Science and Engineering and Ms. Jismy Mathew,Assistant Professor of the Department of Computer Science and Engineering whose insights and feedback have significantly contributed to the refinement and quality of our project.
We also thank all faculty for their support towards completing our project Each team member's contribution has played a crucial role in achieving our goals, and for that, we are truly grateful. Thank you for your dedication, support, and commitment to excellence

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Small-scale businesses often face challenges in embracing digital platforms due to limited technical expertise and resources. **BizNex** is designed to bridge this gap by offering a user-friendly, all-in-one platform that integrates product listing, inventory tracking, financial management, and job postings — empowering local businesses to thrive in the digital marketplace.

1.1 Overview

BizNex provides small businesses with a dedicated web space, offering features like:

- Product listing and management
- Stock tracking with restock notifications
- Integrated billing and finance management
- Job listing feature to connect with local talent
- E-commerce functionality for customer orders

1.2 Problem Statement

- Many local businesses rely on traditional, manual methods for managing inventory, sales, and customer interactions. Without an online presence, they struggle to attract and retain customers in an increasingly digital economy.
- Existing digital solutions are often too complex, costly, or generic, failing to meet the unique needs of small businesses. Inventory mismanagement leads to stock shortages or overstocking, while customers face difficulties accessing

product information, placing orders, or engaging with these businesses. Additionally, businesses miss out on local hiring opportunities due to a lack of a dedicated job listing feature.

- There is a clear demand for an accessible, all-in-one platform that simplifies business operations — from inventory and sales management to customer engagement and hiring — without requiring technical expertise.
- A successful solution must empower small businesses to establish a strong digital presence, streamline operations, enhance customer accessibility, and support job creation — all through an affordable, easy-to-use system designed for their specific needs.
- Lack of visibility for job opportunities

1.3 Objectives

The primary objectives of BizNex are to:

Simplify E-commerce Setup: Provide small businesses with an intuitive, easy-to-navigate platform to quickly set up digital storefronts without requiring technical knowledge.
Streamline Inventory and Finance Management: Enable businesses to track stock levels, receive restocking alerts, manage expenses, and generate invoices — reducing operational inefficiencies.
Enhance Customer Access and Engagement: Ensure customers can seamlessly browse products, place orders, and receive updates, improving overall user experience and boosting sales.
Support Local Job Markets: Include a job listing feature, helping businesses connect with job seekers, fostering local employment opportunities.
Ensure Secure Transactions: Implement data encryption, multi-factor authentication, and secure payment gateways to protect user data and maintain transaction integrity.

1.4 Scope of the Project

The BizNex platform is designed to empower small-scale businesses — including supermarkets, textile shops, shoe stores, and watch retailers — by simplifying their transition to the digital marketplace. It provides a comprehensive, user-friendly solution that streamlines business operations while minimizing technical complexities. BizNex enables businesses to create personalized digital storefronts, efficiently manage inventory with automated restocking alerts, and handle finances through an integrated billing system. Additionally, the platform supports e-commerce operations, allowing customers to browse products, place orders, and track deliveries with ease. To further support business growth, BizNex incorporates a job listing feature, enabling businesses to post vacancies and connect with local talent — promoting both operational efficiency and community engagement.

1.5 Social Relevance

BizNex supports small businesses by empowering them to compete in the digital marketplace. The platform contributes to community development by:

Enhancing Accessibility for Local Businesses: BizNex ensures small businesses can establish an online presence without technical expertise, making it easier for them to reach customers who prefer digital convenience. This improved accessibility helps level the playing field with larger competitors.
Encouraging Efficient Resource Use: The platform automates inventory management, billing, and order processing — reducing manual workload and minimizing waste from overstocking or stockouts. This allows businesses to operate more efficiently and focus on growth.
Promoting Job Creation: By including a job listing feature, BizNex connects businesses with local job seekers, promoting employment opportunities within the community. This not only supports economic growth but also helps businesses quickly fill vacancies with nearby talent.
Strengthening Community Engagement: BizNex helps small businesses improve

their visibility through an integrated marketplace, allowing customers to easily discover and support local stores. This fosters a stronger sense of community and encourages consumers to prioritize local businesses over large, impersonal chains.

1.6 Organization of the Report

The report is structured as follows:
Chapter 1: Introduction - Provides an overview of the project, including objectives, scope, and social relevance.
Chapter 2: Literature Review - Reviews existing solutions and highlights BizNex's unique approach.
Chapter 3: System Design - Describes the architecture, data flow, and key components.
Chapter 4: Implementation - Covers the technologies, frameworks, and development process.
Chapter 5: Results and Discussion - Analyzes performance, user feedback, and system efficiency.
Chapter 6: Conclusion and Future Scope - Summarizes the project and outlines potential enhancements.

Chapter 2

Literature Review

2.1 Related work

Several platforms offer e-commerce and business management capabilities, but they often lack customization for small-scale businesses. Common issues with these systems include:

Shopify: It is an all-in-one e-commerce platform that helps businesses easily create and manage online stores. It offers customizable, mobile-friendly templates, allowing users to build professional websites without coding. Beyond website creation, Shopify handles product and inventory management, supports various payment methods, and includes built-in marketing tools like email campaigns, discounts, and social media integration. It also streamlines shipping with real-time rates and label printing. With an intuitive dashboard providing sales and customer insights, Shopify is a powerful, user-friendly solution for small businesses looking to grow online.
Zoho: It is a versatile business software suite that helps companies manage everything from sales and marketing to accounting and project management. It offers customizable workflows, automation tools, and integrations with apps like Google Workspace and Shopify. Being cloud-based, it supports remote collaboration while providing analytics to track performance and business growth. Zoho's wide range of tools makes it a flexible, all-in-one solution for businesses looking to streamline operations.
Wix: It is a user-friendly website builder that helps businesses create professional, customizable websites without needing coding skills. It offers a variety of templates, drag-and-drop design tools, and built-in features like online stores, blogs, and booking systems. Wix also supports SEO optimization, marketing tools, and integrations with third-party apps to help businesses grow their online presence. Whether you're starting a small store or building a service-based site, Wix makes it easy to create a polished, functional website quickly.

2.2 Proposed Systems

The BizNex platform is designed to revolutionize the way small-scale businesses manage their operations and establish a digital presence. Recognizing the challenges many small businesses face in adopting digital tools due to limited resources and technical expertise, BizNex aims to provide a centralized and user-friendly solution. The core idea is to simplify e-commerce functionalities, streamline business operations, and enhance overall efficiency.

To achieve this, the proposed system will offer a range of essential features. Firstly, it will provide secure user registration and authentication, allowing business owners to create and manage their accounts effectively. The platform will also enable businesses to list, update, and manage their products, which is crucial for establishing an online presence. For smooth transactions, the system will facilitate customer order placement and tracking. Furthermore, BizNex will include tools for finance management, such as budgeting and cash flow tracking, and inventory management, including stock monitoring and automated restocking alerts. An optional job listing feature will also be available, enabling businesses to post job vacancies and connect with potential employees.

In essence, the BizNex platform is envisioned as a comprehensive solution that empowers small-scale entrepreneurs to thrive in the digital marketplace. By focusing on ease of use, automation, and essential business functions, the system aims to bridge the digital gap and foster growth for small businesses

2.3 Comparison of Related Works

Table 2.1: Comparison of BizNex with Other Platforms

Feature	**BizNex**	**Shopify**	**Wix eCommerce**
Product Management	Comprehensive product listing and management	Advanced inventory with AI	Customizable displays with video
Inventory Tracking	Real-time tracking and stock management	Automated updates across channels	Low stock alerts
Financial Management	Integrated billing and finance tools	Multiple payment gateways	Secure checkout options
Job Listings	Connects businesses with local talent	Not available	Not available
E-commerce Functionality	Dynamic subdomain E-Commerce website	Comprehensive multichannel sales	Mobile-optimized store
Mobile Optimization	Mobile-friendly UI	Optimized storefronts	Advanced responsive editor
Marketing Tools	Promotions and engagement tools	Email campaigns, automation	SEO and marketing tools
Pricing	Free of cost	$39–$2000/month	Starts at $29/month

Chapter 3

Design Methodologies

3.1 Software Requirement Specifications

3.1.1 Functional Requirements

User Authentication

Requirement ID: FR-AUTH-001
Description: Users will be authenticated using a username and password. Upon successful authentication, a token is generated for the user and a session is created
Inputs: Username,Password
Outputs: Authentication Status(Success/Failure)
Priority: High
Dependencies: None

Finance Management

Requirement ID: FR-FMGMT-002
Description: Should allow businesses to manage their finances, including budget ing, expense tracking, and financial reporting
Inputs: Finance Information
Outputs: A finance report dashboard
Priority: High
Dependencies: User Authentication

Billing System

Requirement ID: FR-BILLS-003
Description: Facilitates billing for Products

Inputs: Product details
Outputs: Bills and payment information
Priority: High
Dependencies: Inventory Management

Invoice Generation

Requirement ID: FR-INVO-004
Description: Facilitates invoice generation
Inputs: Product details
Outputs: Bills and payment information
Priority: High
Dependencies: Inventory Management

Inventory Management

Requirement ID: FR-IMGMT-005
Description: Allows businesses to monitor stock levels, manage product categories, and automate reordering when necessary
Inputs: Product Details
Outputs: Product Details
Priority: High
Dependencies: None

E-Commerce Management

Requirement ID: FR-EMGMT-006
Description: Allows businesses to list products, process transactions, and manage online orders categories, and automate reordering when necessary
Inputs: Product and Website Details
Outputs: Website and Orders

Priority: High
Dependencies: Inventory Management

Employee Management

Requirement ID: FR-WMGMT-007
Description: Allows businesses to list products, process transactions, and manage online orders categories, and automate reordering when necessary
Inputs: Employee Details
Outputs: Salary Payment,Employee Management
Priority: High
Dependencies: None

Job Listing

Requirement ID: FR-JL-008
Description: Allows businesses to list jobs and aspirants can apply
Inputs: Job Details or Job Application
Outputs: Applicant Details or Application Status
Priority: Medium
Dependencies: None

Market Place

Requirement ID: FR-MP-009
Description: Allows businesses and users to buy and sell products
Inputs: Product Details
Outputs: Orders
Priority: High
Dependencies: E-Commerce Management, Inventory Management

Cloud Storage

Requirement ID: FR-CS-010
Description: Allows businesses owners to upload the necessary documents needed by them, by using s3[12] bucket
Inputs: Documents
Outputs: View documents
Priority: High
Dependencies: None

Business Card creation

Requirement ID: FR-BC-010
Description: Allows businesses to create business cards
Inputs: Business name,address,phone number,website
Outputs: Download business card
Priority: High
Dependencies: None

3.1.2 Non-Functional Requirements

Browser Compatibility

Requirement ID: NFR-BC-001
Description: Ensure the project maintains functionality and consistent display across multiple web browsers including Chrome, Firefox, Safari, and Edge.
Priority: High

Error Handling and Logging

Requirement ID: NFR-EH-001
Description: Implement robust error handling mechanisms within the project to

gracefully manage unexpected errors and exceptions. Log error details for effective debugging, including error messages.
Priority: High

Code Maintainability

Requirement ID: NFR-CM-001
Description: Maintain clean and well-structured code throughout the project to ensure readability, maintainability, and extensibility. Adhere to established coding standards, modularize the codebase, and document appropriately, while utilizing version control effectively.
Priority: High

3.1.3 Security Measures

Requirement ID: NFR-SCM-001
Description: Utilize bcrypt[7] encryption algorithm to securely hash passwords before storing them in the database, ensuring sensitive data remains protected, Verifies phone number and email before creating an account. Uses Google Authentication for login

3.2 Software Design Documents

3.2.1 System Architecture Design

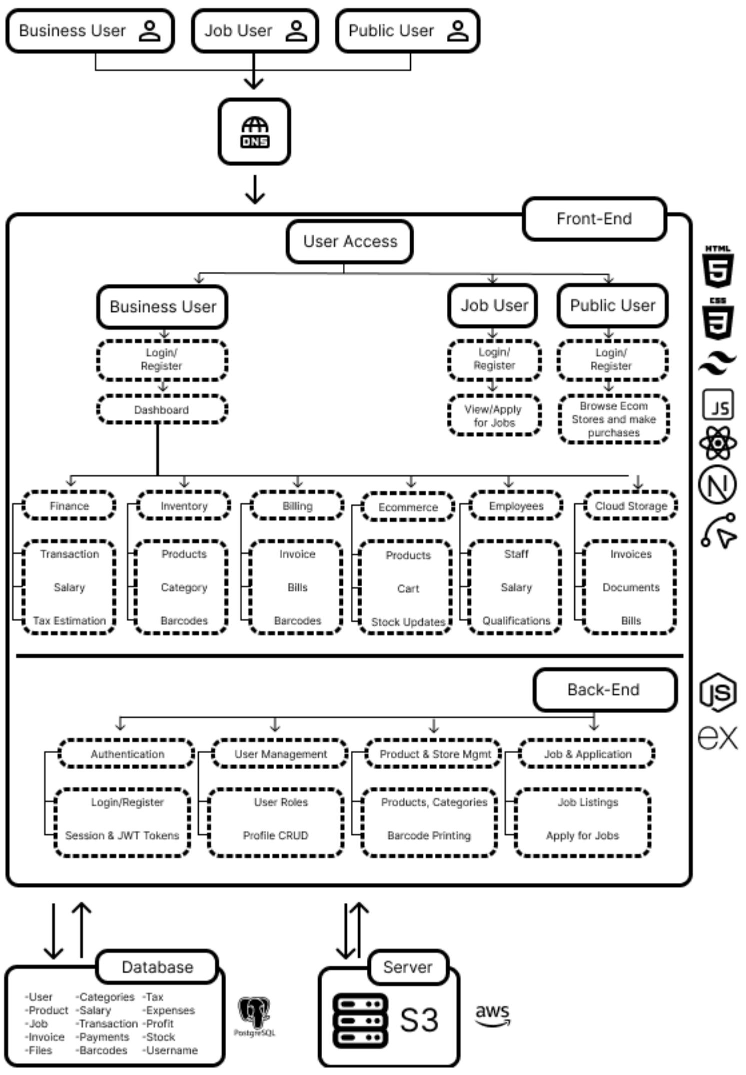


Figure 3.1: System Architecture

Frontend Application (ReactJS,NextJS)

- Responsible for the user interface and interaction with users.
- Built using ReactJS[1] for dynamic UI rendering and Material-UI for design components, while leveraging NextJS for server-side rendering and optimized performance.(In Figure 3.1)

Backend Server (Node.js and Express.js)

- Provides RESTful APIs to the frontend for data exchange.
- Developed using Node.js[3] and Express.js[2] for backend logic and routing.(In Figure 3.1)

Database (PostgreSQL)

- Stores application data, including user information, product details etc.(In Figure 3.1)

System Interaction

- **Frontend-Backend Interaction:**
  - Frontend interacts with the backend server via RESTful APIs for fetching and manipulating data.
  -Backendprocesses requests, retrieves data from the Postgres[4] database, and sends responses to the frontend(In Figure 3.1)

3.2.2 Scalability and Security Measures

Vertical Scalability

- Architecture designed for vertical scalability to handle increased load by upgrading server resources (e.g., CPU, RAM) as needed.

Security Measures

- Communication between components secured using HTTPS/TLS protocols.

PostgreSQL

- Utilizes PostgreSQL[4] for its flexible relation-based data model and scalability.
- Stores application data, providing high performance and horizontal scalability.

3.2.3 System Features

1.Authentication

- **Description:** This feature ensures secure access to the system by requiring users to authenticate using a unique username and password combination.

2.Homepage Display

- **Description:** Upon accessing the system, users are presented with a homepage that displays the services which will be provided.

3.Finance Management

- **Description:** This feature allows businesses to manage their finances, including budget ing, expense tracking, and financial reporting

(a).Project Creation

- **Description:** Managers can initiate new projects within the system, providing essential details to kick-start project development and execution.

(b).Subtask Creation

- **Description:** Managers can break down projects into smaller, manageable tasks by creating subtasks, ensuring a structured approach to project management.

4.Billing System

- **Description:** This feature automates invoice generation and ensures accurate tax cal culations for business transactions.

5.Inventory Management

- **Description:** This feature allows businesses to monitor stock levels, manage product categories, and automate reordering when necessary.

5.E-Commerce Management

- **Description:** This feature allows businesses to list products, process transactions, and manage online orders.

6.Logout

- **Description:** Users can securely log out of the system, terminating their current session and ensuring the protection of sensitive information

3.2.4 Application Architecture Design

Frontend

Technology: ReactJS[1] with Material UI.
Description: The frontend application will be developed using ReactJS[1], a popular JavaScript library for building user interfaces. Material UI will be used for styling to ensure a modern and responsive design.

Backend

Technology: Node.js[3] with Express.js[2]
Description: The backend of the application will be built using the Node.js[3] runtime environment with the Express.js[2] framework. Express.js[2] will be used to handle routing, middleware, and request/response handling
.

Database

Technology: PostgreSQL
Description: PostgreSQL[4], a powerful open-source relational database, will be used to store application data. It ensures ACID compliance, supports complex queries, and provides robust scalability. Sequelize ORM for Node.js[3] will be utilized to define schemas, interact with the database, and perform data validation
.

User Authentication and Authorization

Method: JSON Web Tokens (JWT) with bcrypt.js[7]
Description: User authentication and authorization will be implemented using JWT. bcrypt.js[7] will be used for securely hashing passwords before storing them in the database. JWT tokens will be generated upon successful authentication and included in subsequent requests for authorization.

Middleware

CORS: Cross-Origin Resource Sharing middleware for handling cross-origin requests.
Body Parsing: Middleware for parsing incoming request bodies in JSON format.
Cookie Parsing: Middleware for parsing cookies attached to incoming requests.

Security Measures

- Hash passwords securely using bcrypt.js[7] before storing them in the database.
- Protect sensitive endpoints with JWT-based authentication.

Deployment Environment

Platform: AWS (Amazon Web Services), Render[13],Vercel[10]
Description: AWS EC2[11] instance is used for hosting database, S3[12] Bucket is used for file storage. The render[13] is used to host backend and vercel[10] is used to host frontend

3.2.5 Constraints

Technology Constraints

Framework Limitations: The choice of frameworks for backend (Express[2] and Node.js[3]) and frontend (React) development may have certain limitations or constraints. Developers need to work within the boundaries of these frameworks while implementing features and functionalities.
Integration Challenges: Integrating different technologies and components, such as Express[2] and Node.js[3] with Postgres[4] for data management and React[1] for frontend, may pose challenges in terms of compatibility and interoperability.
Third-Party Dependencies: The reliance on third-party libraries or services for specific functionalities introduces dependencies and constraints related to their compatibility, availability, and support.

Hardware Constraints

Resource Limitations: The hardware infrastructure on which Collobo will run may have limitations in terms of processing power, memory, and storage capacity. Developers need to optimize the system to perform efficiently within these constraints.
Scalability Considerations: Hardware constraints can affect the scalability of the system, particularly in handling large volumes of concurrent users or data. Scalability solutions need to be implemented to ensure the system can accommodate growth without compromising performance.
Reliability and Availability: Hardware failures or limitations can impact the reliability and availability of the system. Redundancy, fault tolerance, and failover mechanisms may need to be implemented to ensure uninterrupted service.

Security Constraints

Data Security: Ensuring the security of user data stored in Postgres[4] and transmitted over the network is crucial. Implementing encryption, access controls, and secure communication protocols is essential to protect sensitive information.
Authentication and Authorization: Implementing robust authentication and authorization mechanisms, such as JWT for user authentication and RBAC for access control, helps mitigate security risks and enforce user permissions effectively.

3.2.6 API Design

Authentication API

Email OTP Endpoint for client
Endpoint URL: /signup/client/send-email-otp
Method: POST

Description: Send OTP for verification of E-mail of new client .

Request Body:

- **email** (string, required): The E-mail of the user.
- Add any additional fields required for user registration.

Response:

- **200 OK:** Email otp sent successfully.
- **400 Bad Request:** Invalid request format or missing required fields.
- **409 Conflict:** Email already in use.
- **500 Internal server error** Internal server error

Email verify endpoint for client
Endpoint URL: /signup/client/verify-email-otp
Method: POST
Description: Verify the email using OTP

Request Body:

- **email** (string, required): The E-mail of the user.
- **emailOtp** (string, required): The E-mail OTP of the user.
- Add any additional fields required for user registration.

Response:

- **200 OK:** Email verified successfully.
- **400 Bad Request:** Invalid request format or missing required fields.

- **500 Internal server error** Internal server error

Phone OTP Endpoint for client
Endpoint URL: /signup/client/send-phone-otp
Method: POST
Description: Send OTP for verification of Phone of new client .

Request Body:

- **email** (string, required): The E-mail of the user.
- **phone** (string, required): The phone of the user.
- Add any additional fields required for user registration.

Response:

- **200 OK:** Phone otp sent successfully.
- **400 Bad Request:** Invalid request format or missing required fields.
- **500 Internal server error** Internal server error

Phone verify endpoint for client
Endpoint URL: /signup/client/verify-phone-otp
Method: POST
Description: Verify the phone using OTP

Request Body:

- **email** (string, required): The E-mail of the user.
- **phoneOtp** (string, required): The phone OTP of the user.
- Add any additional fields required for user registration.

Response:

- **200 OK:** Phone verified successfully.
- **400 Bad Request:** Invalid request format or missing required fields.
- **500 Internal server error** Internal server error

Creation endpoint for client
Endpoint URL: /signup/client/create-client
Method: POST
Description: Create client account and database

Request Body:

- **username** (string, required): The username of the user.
- **ownername** (string, required): The name of the business owner.
- **address** (string, required): The address of the business.
- **email** (string, required): The E-mail of the user.
- **business_category** (string, required): The category of the business.
- **phone** (string, required): The phone number of the user.
- **password** (string, required): The password of the user.

Response:

- **200 OK:** Phone verified successfully.
- **400 Bad Request:** Invalid request format or missing required fields.
- **500 Internal server error** Internal server error

Login Endpoint
URL: /login/client/login-client
Method: POST
Description: Authenticate user credentials and generate access token.

Request Body:

- **username** (string, required): The Email of the user.
- **password** (string, required): The password of the user.

Response:

- **200 OK:** User successfully logged in. Response body contains user information and access token.
- **400 Bad Request:** Invalid username or password.
- **404 Not Found:** Email not found.
- **500 Internal Server Error:** Failed to generate token.

Display manual transaction Endpoint
URL: /finance/manual-transactions
Method: GET
Description: To display the manual transactions

Response:

- **200 OK:** Data
- **500 Internal Server Error:** Database query failed

Login Endpoint
URL: /dashboard/grouped/all
Method: POST

Description: Returns array of income expense and profit in specific periods of time

Request Body:

- **groupedby** (string, required): used to group the output into separate periods

Response:

- **200 OK:** Data
- **400 Bad Request:** Invalid group by value
- **500 Internal Server Error:** Database query failed

3.2.7 Technology Stack

Backend Technologies:

- **Node.js:** JavaScript runtime environment for server-side development.
- **Express.js:** Web application framework for Node.js[3].

Database:

- **PostgreSQL:** Relational database management system for storing application data.

Frontend Framework:

- **React:** JavaScript library for building user interfaces.

Cloud Services:

- **AWS (Amazon Web Services):** Cloud computing platform for hosting, storage, and scalability.

This technology stack combines the power of Node.js[3] and Express.js[2] for backend development, PostgreSQL[4] for relational database management, React[1] for frontend design, and AWS for cloud-based hosting and scalability.

3.2.8 Database Design

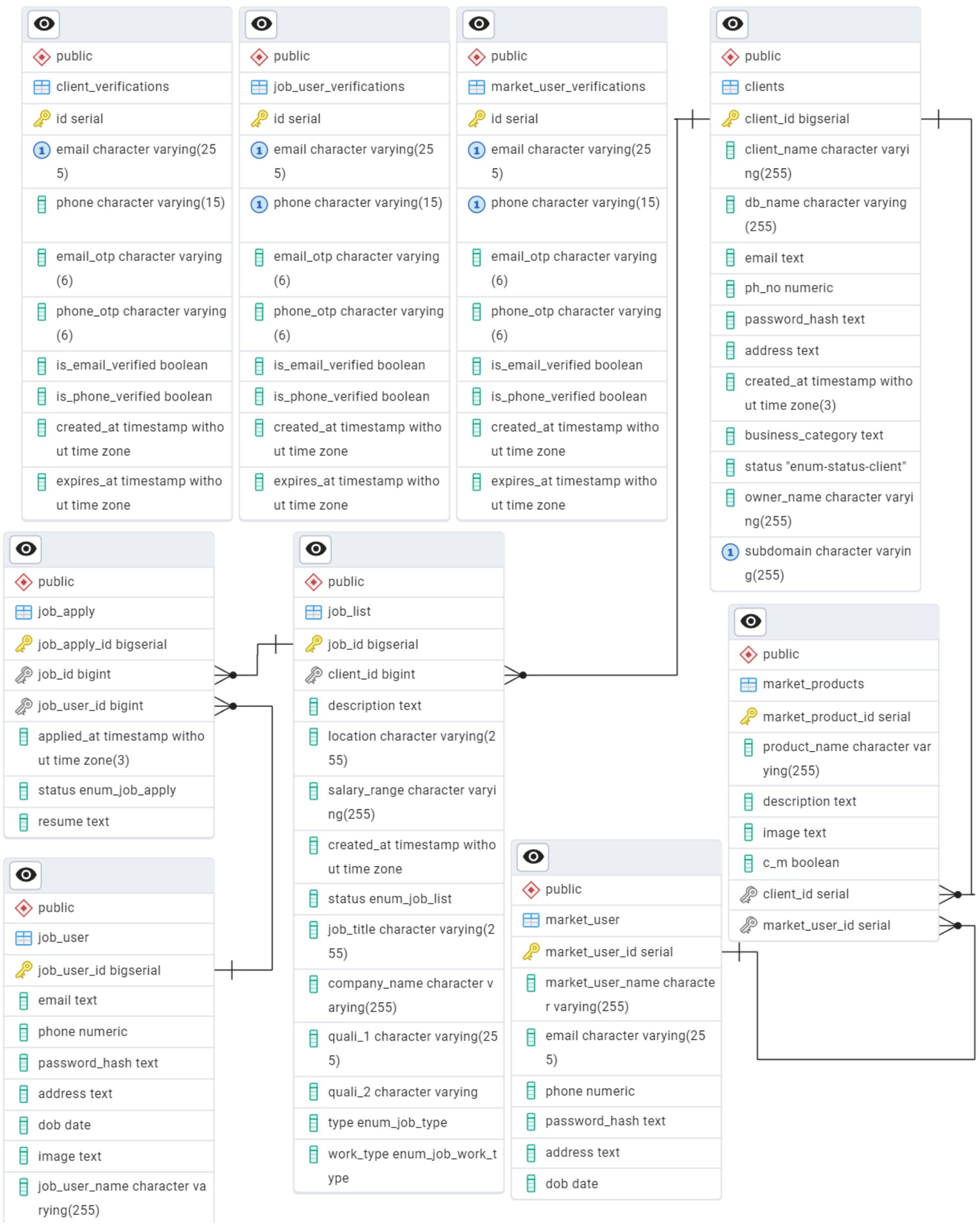

public
client_verifications
id serial
email character varying(255)
phone character varying(15)
email_otp character varying(6)
phone_otp character varying(6)
is_email_verified boolean
is_phone_verified boolean
created_at timestamp without time zone
expires_at timestamp without time zone
public
job_user_verifications
id serial
email character varying(255)
phone character varying(15)
email_otp character varying(6)
phone_otp character varying(6)
is_email_verified boolean
is_phone_verified boolean
created_at timestamp without time zone
expires_at timestamp without time zone
public
market_user_verifications
id serial
email character varying(255)
phone character varying(15)
email_otp character varying(6)
phone_otp character varying(6)
is_email_verified boolean
is_phone_verified boolean
created_at timestamp without time zone
expires_at timestamp without time zone
public
clients
client_id bigserial
client_name character varying(255)
db_name character varying(255)
email text
ph_no numeric
password_hash text
address text
created_at timestamp without time zone(3)
business_category text
status "enum-status-client"
owner_name character varying(255)
subdomain character varying(255)
public
job_apply
job_apply_id bigserial
job_id bigint
job_user_id bigint
applied_at timestamp without time zone(3)
status enum_job_apply
resume text
public
job_list
job_id bigserial
client_id bigint
description text
location character varying(255)
salary_range character varying(255)
created_at timestamp without time zone
status enum_job_list
job_title character varying(255)
company_name character varying(255)
quali_1 character varying(255)
quali_2 character varying
type enum_job_type
work_type enum_job_work_type
public
market_products
market_product_id serial
product_name character varying(255)
description text
image text
c_m boolean
client_id serial
market_user_id serial
public
job_user
job_user_id bigserial
email text
phone numeric
password_hash text
address text
dob date
image text
job_user_name character varying(255)
public
market_user
market_user_id serial
market_user_name character varying(255)
email character varying(255)
phone numeric
password_hash text
address text
dob date

Figure 3.2: Master DB

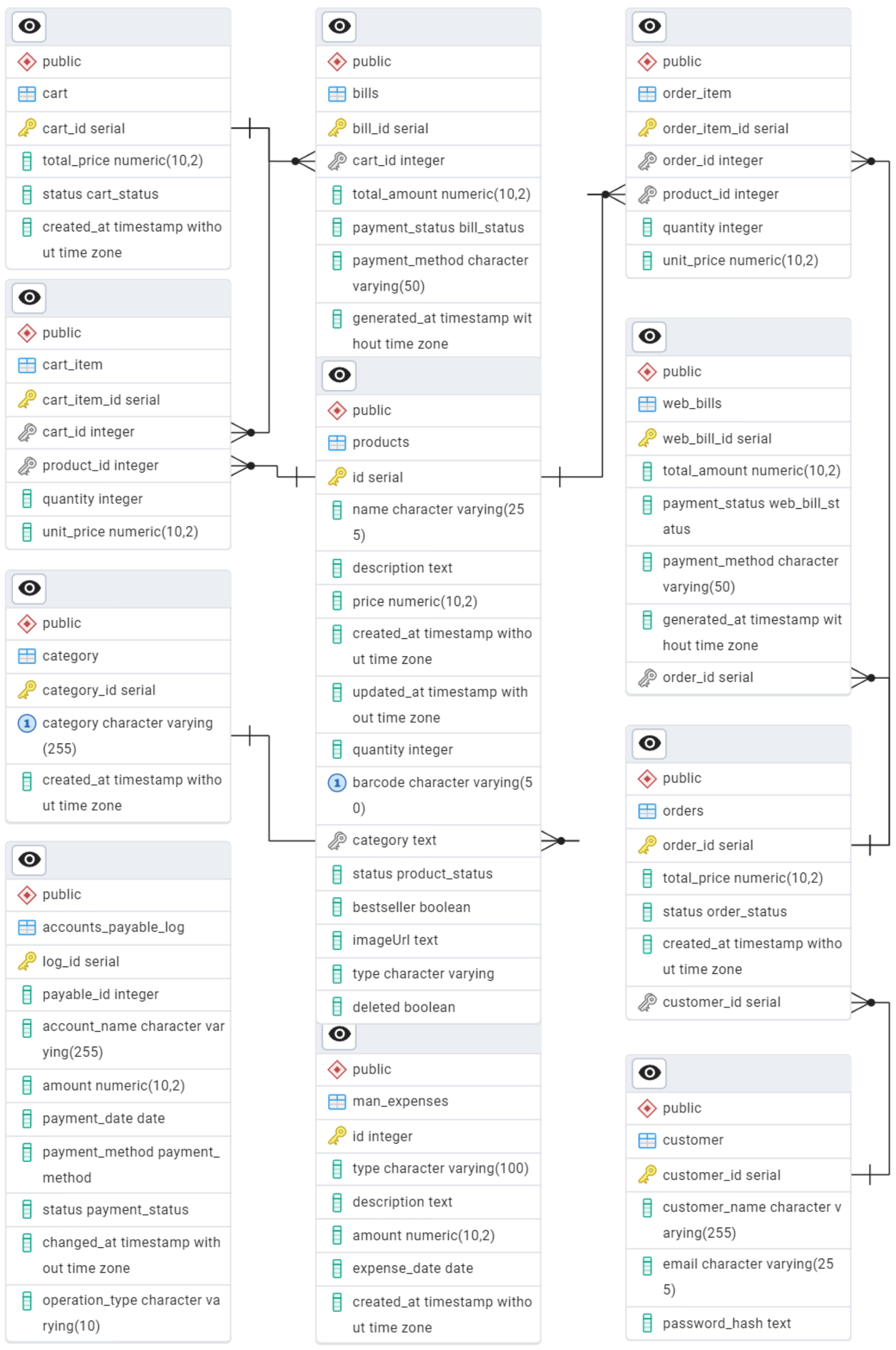


Figure 3.3: Client DB-1

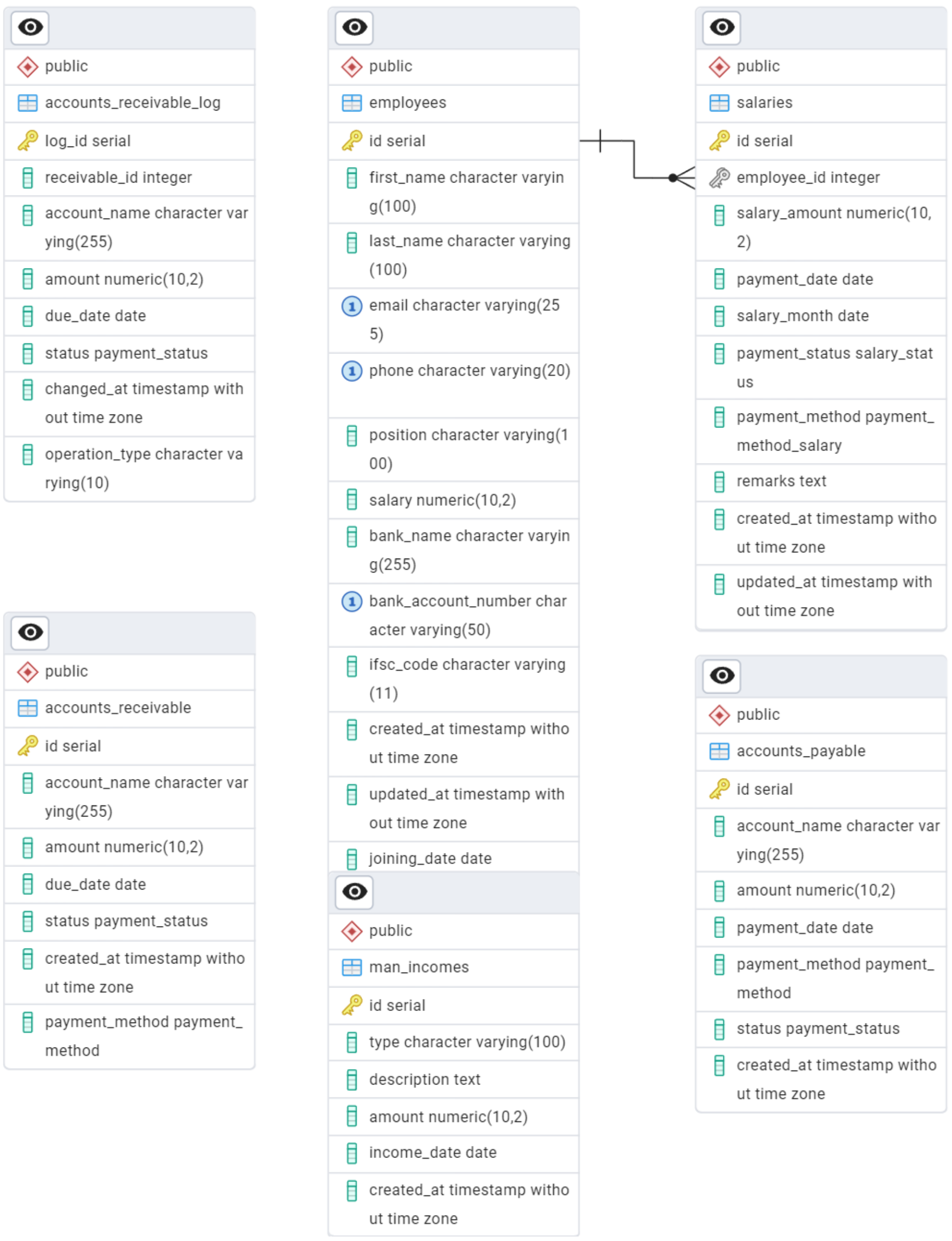

public
accounts_receivable_log
log_id serial
receivable_id integer
account_name character varying(255)
amount numeric(10,2)
due_date date
status payment_status
changed_at timestamp without time zone
operation_type character varying(10)
public
employees
id serial
first_name character varying(100)
last_name character varying (100)
email character varying(255)
phone character varying(20)
position character varying(100)
salary numeric(10,2)
bank_name character varying(255)
bank_account_number character varying(50)
ifsc_code character varying (11)
created_at timestamp without time zone
updated_at timestamp without time zone
joining_date date
public
salaries
id serial
employee_id integer
salary_amount numeric(10,2)
payment_date date
salary_month date
payment_status salary_status
payment_method payment_method_salary
remarks text
created_at timestamp without time zone
updated_at timestamp without time zone
public
accounts_receivable
id serial
account_name character varying(255)
amount numeric(10,2)
due_date date
status payment_status
created_at timestamp without time zone
payment_method payment_method
public
man_incomes
id serial
type character varying(100)
description text
amount numeric(10,2)
income_date date
created_at timestamp without time zone
public
accounts_payable
id serial
account_name character varying(255)
amount numeric(10,2)
payment_date date
payment_method payment_method
status payment_status
created_at timestamp without time zone

Figure 3.4: Client DB-2

The above diagrams illustrate the database structure used in the BizNex application. They show the relationships between users, products, and service categories in the system.(Figure 3.2,Figure3.3 and Figure 3.4)

3.3 Dataflow Diagram/Use Case Diagrams

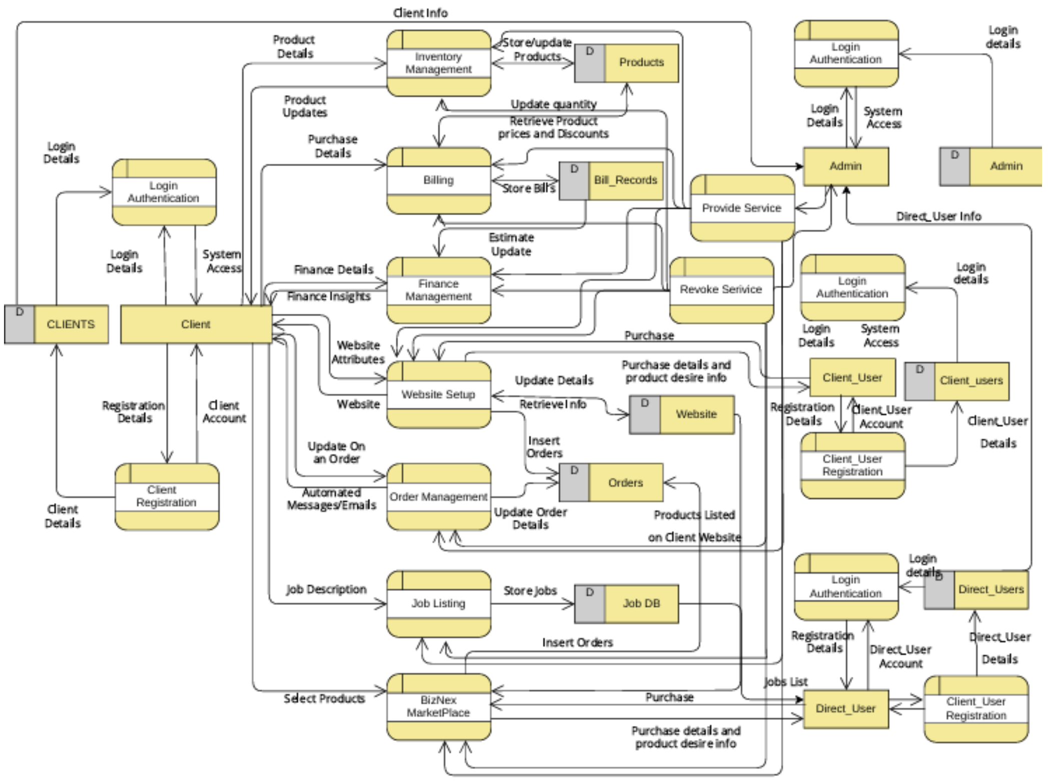


Figure 3.5: Dataflow Diagram of the System

Actors / External Entities

CLIENTS / Clients

They are the main users of the platform: business owners, vendors, or job providers.(Figure 3.5)

Client_Users

End-users who interact with businesses via the BizNex platform.(Figure 3.5)

Job Users

Users who apply for jobs posted on the platform, upload resumes, and track application status.(Figure 3.5)

Market Users

Customers who browse products/services and make purchases through the BizNex platform.(Figure 3.5)

Main System Modules and Flow Explanation (Figure 3.5)

1. Authentication Modules

Login Authentication (Clients / Job Users / Market Users):

Handles login for all roles.
Connected to system authentication and directs users to role-specific access.

System Authentication:

Central node ensuring only valid users access respective modules.

2. Product / Service Management

Inventory:

Manages stock, updates, and product/service details.

Billing:

Handles checkout, purchases, and invoice management.

DL Barcodes:

Generates barcodes for products for easy tracking.

Provide Service / Receive Service:

For clients offering and users availing services.

Products:
End-point of product listing visible to users.

Purchase History:
Logs client purchases for future reference.

3. Order and Transaction Processing

Cart:
Temporarily stores selected products for checkout.

Online Payment:
Handles secure transactions.

Invoice:
Generates digital receipts.

Purchase Details:
Stores transactional data.

4. Website / Business Setup

Website Setup:
Helps clients create their digital storefront.

Update Website:
Enables updates like banners, product listings, etc.

Online Management:

Ongoing backend management for the client's digital presence.

5. Support and Communication

Message Client:
Communication feature between users and service providers.

Update User Details / Account:
For maintaining client user profiles.

6. Job and Employment Section

Job Listing / Job DB:
Clients can post jobs, which are stored in Job DB.

Insert Job Details:
Adding job descriptions.

Job Applications:
Users apply for jobs, resumes get stored.

3.4 Algorithms

3.4.1 Algorithm for Client Sign-Up

Algorithm(Client Sign-Up)

i. Start

ii. Fetch E-mail from user and send OTP request to backend using POST method

iii. Backend sends OTP to the fetched E-mail

iv. Fetch E-mail OTP and send to backend using POST method

v. If OTP matches, Verify E-mail

vi. Fetch phone number from user and send OTP request to backend using POST method

vii. Backend sends OTP to the fetched phone number

viii. Fetch phone OTP and send to backend using POST method

ix. If OTP matches, Verify phone number

x. Read password

xi. If E-mail and phone number is verified, Redirect to complete profile page with E-mail,phone and password as variables

xii. Fetch owner name,business name,business category and business address and send to backend using POST method along with E-mail, phone number and password

xiii. Backend creates client entry and database for the client

xiv. Stop

3.4.2 Algorithm for Client Authentication

Algorithm(Client Authentication)

i. Start

ii. Fetch E-mail and password entered by the user

iii. Check if the E-mail exists in the database

iv. If E-mail exists, compare the entered password with the stored password

v. If passwords match, grant access

vi. Accept and store token , redirect to dashboard

vii. If passwords do not match, display ”Incorrect password” message

viii. If E-mail does not exist, display ”User not found” message

ix. Stop

3.4.3 Algorithm for Client Google Authentication

Algorithm(Client Google Authentication)

i. Start

ii. Client clicks sign-in with google button which redirects them to select accounts section

iii. Select E-mail and send it to the backend

iv. Check if the E-mail exists in the database

v. If E-mail exists, grant access

vi. Accept and store token , redirect to dashboard

vii. If E-mail does not exist, display ”User not found” message

viii. Stop

3.4.4 Algorithm for Dashboard

Algorithm(Dashboard)

i. Start

ii. Fetch data from backend using GET methods

iii. Display the Today’s income, expense, profit and number of orders

iv. Display the percent change between this week and last week of the metrics

v. Fetch the duration for each graph and send the data to backend using POST method

vi. Backend returns the values grouped by time frames

vii. Display the graphs

viii. Stop

3.4.5 Algorithm for Payments

Algorithm(Payments)

i. Start

ii. Fetch data from form and sent to backend using POST method

iii. The payment is added to database

iv. Fetch the data from backend using GET method

v. Display the payment data in the form of a table

vi. When clicked edit button an entry , add the details to form

vii. Fetch data from the form and sent to backend using PUT method for update

viii. The backend update the data in database

ix. When clicked delete button , sent id of the entry to backend for deletion using DELETE method

x. Backend delete the payment from database

xi. Stop

3.4.6 Algorithm for Employees

Algorithm(Employees)

i. Start

ii. Fetch data from form and sent to backend using POST method

iii. The Employee is added to database

iv. Fetch the data from backend using GET method

v. Display the Employees data in the form of a table

vi. When clicked edit button an entry , add the details to form

vii. Fetch data from the form and sent to backend using PUT method for update

viii. The backend update the data in database

ix. When clicked delete button , sent id of the entry to backend for deletion using DELETE method

x. Backend delete the Employee from database

xi. Stop

3.4.7 Algorithm for Products

Algorithm(Products)

i. Start

ii. Fetch data from the form

iii. Store image in S3[12] bucket and get the url

iv. Send the form data and the url to backend using POST method

v. The Product is added to database

vi. Fetch the data from backend using GET method

vii. Display the product data in the form of a table

viii. When clicked edit button an entry , and edit the details

ix. Fetch data and sent to backend using PUT method for update

x. The backend update the data in database

xi. Backend delete the product from database

xii. Stop

3.4.8 Algorithm for Product Category

Algorithm(Category)

i. Start

ii. Fetch data from form and sent to backend using POST method

iii. The category is added to database

iv. Fetch the data from backend using GET method

v. Display the Categories data in the form of a table

vi. When clicked delete button , sent id of the entry to backend for deletion using DELETE method

vii. Backend delete the category from database only if there is no products in the category

viii. Stop

3.4.9 Algorithm for Job-listing Client side

Algorithm(Job-listing Client side)

i. Start

ii. Fetch data from form and sent to backend using POST method

iii. The job is added to database

iv. Fetch the data from backend using GET method

v. Display the Job data and the applicants for the jobs

vi. when clicked delete button , sent id of the entry to backend for deletion using DELETE method

vii. backend delete the job from database

viii. Show the details of applicants when view applicants button is clicked, Using Get method

ix. When reject button is clicked,job application id is sent to backend using POST method

x. The backend changes the application status to rejected

xi. When view details button is clicked,job applicant details are displayed

xii. Stop

3.4.10 Algorithm for Job-listing job-User side

Algorithm(Job-listing job-User side)

i. Start

ii. Fetch job data from backend using GET method

iii. When clicked apply button, sent id of the applicant to backend using POST method

iv. Backend connect the applicant to job applicants database database

v. Get the statuses of application from backend using GET method

vi. Display the statuses of applications

vii. Fetch resume file from job user,and add to S3[12] bucket which return a url

viii. The url is sent to backend for storage using POST METHOD

ix. Stop

Chapter 4

Implementation

4.1 Implementation Details

4.1.1 Code Development

Set Coding Standards

In order to maintain consistency and readability in our codebase, the following coding standards have been established:

- Naming Conventions:
  - Descriptive names are used for variables, functions, and classes to enhance readability and understanding.
  - Consistent camelCase naming convention is followed throughout the codebase.
- Indentation and Formatting:
  - Consistent indentation, using either spaces or tabs, is employed to improve code readability.
  - Built-in VSCode text formatters ensure uniformity in text formatting across the codebase.
- Comments:
  - Clear and concise comments are provided to explain complex code logic, enhancing code comprehension.
  - Unnecessary comments stating the obvious are avoided, keeping the code clutter-free.
- Error Handling:

  - Consistent error handling mechanisms are implemented across the code base to ensure robustness.
  - Meaningful error messages and status codes are provided to aid in de bugging and troubleshooting.

- Version Control:
  - Bestpractices in version control are followed, including meaningful com mit messages, effective branching strategies, and regular code integra tion

4.1.2 Source Code Control Setup

GIT Repository
Weestablished a Git repository to manage the source code of the **BizNex** project. The repository was hosted on a platform, GitHub[8] and GitLab to facilitate collabo ration among team members and enable version control. Various branches were created for feature development, bug fixes, and testing, and changes were merged into the main branch after review and approval.

4.2 Libraries/Applications

The **BizNex** project makes use of several libraries and applications to facilitate development, enhance functionality, and support core features across the frontend, backend, and database layers.

1. **Frontend**
   - **React:** JavaScript library for building dynamic and interactive user interfaces.
   - **Next.js:** React[1]-based framework for building server-side rendered and statically generated web applications.
   - **axios[6]:** Promise-based HTTP client for making API requests from the frontend.

- **react-router-dom:** Library for managing navigation and routing in React[1] applications.
- **react-icons:** Provides a wide variety of icon sets as React[1] components.
- **lucide-react:** Icon library offering modern and consistent SVG icons for React[1].
- **chart.js:** JavaScript library for rendering responsive charts.
- **react-chartjs-2:** React[1] wrapper for Chart.js, enabling easy chart integration.
- **recharts:** Charting library built with React[1] and D3 for creating composable chart components.
- **rc-slider:** React[1] component for creating highly customizable sliders.
- **jspdf:** Library for generating PDF documents on the client-side.
- **jsbarcode:** JavaScript library for generating barcodes.
- **qrcode:** JavaScript library to generate QR codes for various data formats.

2. **Database**

   - **PostgreSQL:** Relational database system used to store and manage application data.
   - **connect-pg-simple:** PostgreSQL[4] session store for Express[2] to maintain user session data.
   - **pg[5]:** PostgreSQL[4] client for Node.js[3].

3. **Backend**

   - **Node.js:** JavaScript runtime for server-side application logic.
   - **Express.js:** Lightweight web framework for building RESTful APIs.
   - **bcrypt:** Used to hash passwords and enhance security.
   - **cors:** Middleware to allow or restrict resources on a web server depending on the origin.

- **dotenv:** Module to load environment variables from a '.env' file into 'process.env'.
- **express-session:** Middleware for managing user sessions in Express[2] applications.
- **passport:** Authentication middleware for Node.js[3] applications.
- **passport-google-oauth2 / passport-google-oauth20:** Google OAuth strategies for user login.

4. **Utilities and Integrations**

   - **nodemailer:** Module for sending emails such as verifications and alerts.
   - **twilio:** Communication API for sending SMS and voice messages.
   - **moment:** Library for parsing, validating, and formatting dates and times.
   - **nodemon:** Development tool that restarts the server automatically on code changes.

These libraries and frameworks work cohesively to deliver a modern, interactive, and secure web application through BizNex, ensuring robust functionality and a seamless user experience.

4.3 Deployment

The deployment architecture for the BizNex project was designed to be cost-effective, scalable, and reliable by utilizing modern cloud platforms and services. The deployment setup includes hosting the frontend on Vercel[10], the backend on Render[13], PostgreSQL[4] on an AWS EC2[11] instance, and file storage on Amazon S3[12].

1. **Frontend Deployment on Vercel:**

   - The React[1]-based frontend was deployed using Vercel[10], a platform optimized for frontend frameworks and static sites.
   - Vercel[10] provided seamless integration with GitHub[8], enabling continuous deployment on every push to the main branch.

- Environment variables required for API interactions were securely configured within the Vercel[10] dashboard.

2. **Backend Deployment on Render:**
   - The Node.js[3] backend was deployed on Render[13], a cloud service suitable for dynamic web applications and APIs.
   - Render's[13] automated deployment pipeline was connected to the GitHub[8] repository for efficient CI/CD.
   - Environment variables such as database credentials, secret keys, and S3[12] configuration were added securely via the Render[13] dashboard.

3. **PostgreSQL[4] Database on EC2:**
   - An AWS EC2[11] instance was provisioned to host a PostgreSQL[4] database.
   - The instance was configured with appropriate compute resources and a secure network setup to allow controlled access from the backend.
   - Regular backups and secure SSH access were implemented to maintain database reliability and integrity.

4. **File Storage using Amazon S3:**
   - Amazon S3[12] was used as a secure and scalable solution for storing files such as profile images, product images, and job-related documents.
   - The backend generated signed URLs to enable clients to upload files directly to S3[12], ensuring both security and efficiency.
   - Bucket policies and access permissions were configured to prevent unauthorized access.

This distributed deployment setup ensures that the BizNex platform is modular, scalable, and easy to maintain. By leveraging Vercel[10] for the frontend, Render[13] for the backend, EC2[11] for PostgreSQL[4] hosting, and S3[12] for file storage, the system achieves optimal performance while remaining budget-conscious.

Chapter 5

Testing

5.1 Test Plan

Test Objectives

The testing validates the functionalities of the BizNex system according to the Software Require ments Specification (SRS). It ensures that the system is reliable, secure, and performs efficiently. Before deployment, it provides a way to identify and address various defects and issues in the system if any.

Test Scope

The testing is conducted for both the frontend and backend components of the EventHub website. This includes unit testing of all the components

Test Approach

The testing approach used involves conducting both manual and automated testing tech niques. Unit testing has been performed on each component.

Test Environment

Test cases

Unit Testing:Individual components in project is tested.
Integral Testing:Backend and Frontend features are integrated and tested.
Functional Testing:Functionality as a whole is put to test.
Load Testing:The performance under stress is put to test.

Test Execution

Executing each test cases and recording the results

Test schedule

1.Preparing each test cases.
2.Executing each test cases.
3.Bug fixing
4.Regression Testing after bug fix.

Risks and Contingencies

Changes in evolving needs may find it difficult to adapt to changes.

Conclusion

According to the test objectives,scope,approach and schedule a detailed test plan is drawn out for implementation.

5.2 Test Scenarios

1. Client Sign-Up

   TC-01: Verify that OTP is sent to the email address provided by the user.
   TC-02: Verify that email OTP verification is successful with correct OTP.
   TC-03: Verify that OTP is sent to the phone number provided by the user.
   TC-04: Verify that phone OTP verification is successful with correct OTP.
   TC-05: Verify that after successful verification, user is redirected to the profile completion page.
   TC-06: Verify that all user information is correctly submitted to the backend and client account is created.

2. Client Authentication

   TC-07: Verify that login is successful with valid email and password.
   TC-08: Verify that login fails with incorrect password.
   TC-09: Verify that error message ”User not found” is displayed for non-existent email.
   TC-10: Verify redirection to dashboard after successful login and token storage.

3. Google Authentication

   TC-11: Verify that the Google login popup is triggered upon clicking the button.
   TC-12: Verify access is granted if email exists in database.
   TC-13: Verify proper error message is shown if email does not exist.

4. Dashboard

   TC-14: Verify correct data is fetched and displayed (income, expenses, profit, orders).
   TC-15: Verify that graph duration filters send appropriate requests and update charts.

5. Payments
   TC-16: Verify that new payments are added with valid form submission.
   TC-17: Verify that payments are correctly displayed in table format.
   TC-18: Verify payment updates reflect correctly after editing.
   TC-19: Verify that deleted payments are removed from the table.

6. Employees
   TC-20: Verify that employee data is stored and displayed correctly.
   TC-21: Verify that editing employee updates their data.
   TC-22: Verify that employee is deleted on delete action.

7. Products
   TC-23: Verify that product image is stored in S3[12] and URL is retrieved.
   TC-24: Verify that new products are correctly added and displayed.
   TC-25: Verify that products can be updated and deleted.

8. Categories
   TC-26: Verify that new categories are added and shown in table.
   TC-27: Verify that categories cannot be deleted if products exist under them.

9. Job Listing – Client Side
   TC-28: Verify that jobs are added and displayed with applicants.
   TC-29: Verify deletion and rejection of applicants updates UI and backend.

10. Job Listing – Job User Side
    TC-30: Verify that job applications are sent and statuses are tracked.
    TC-31: Verify that resume is uploaded to S3[12] and the URL is stored.

5.3 Unit Testing

Unit testing was conducted to validate individual components and ensure each module of BizNex works as expected in isolation. The testing framework used was **Jest** for JavaScript/React and **Supertest** + **Mocha/Chai** for Node.js APIs.

Frontend Testing (React)

1. **UT-01:** Verify login form renders correctly with all fields and button.

2. **UT-02:** Ensure form validation triggers for empty or invalid input fields.

3. **UT-03:** Test navigation from login to dashboard upon successful login.

4. **UT-04:** Check that product listing displays correct data from API.

5. **UT-05:** Test "Add to Cart" button adds product to cart state.

6. **UT-06:** Verify job listing component displays all job posts accurately.

7. **UT-07:** Check that messages sent through the contact form are dispatched correctly.
8. **UT-08:** Validate QR/Barcode components render and generate codes based on input.

Backend Testing (Node.js + Express)

1. **UT-09:** Validate POST /register route creates a new user with valid data.
2. **UT-10:** Ensure POST /login route authenticates users with correct credentials.
3. **UT-11:** Verify GET /products returns all product entries.
4. **UT-12:** Check POST /products creates a new product.
5. **UT-13:** Ensure PUT /products/:id updates product data correctly.
6. **UT-14:** Confirm DELETE /products/:id removes a product from DB.
7. **UT-15:** Validate job application submission via POST /jobs/apply.
8. **UT-16:** Check resume upload route sends file to S3 and stores URL in DB.

Database Testing (PostgreSQL)

1. **UT-17:** Ensure user data is saved correctly upon registration.
2. **UT-18:** Verify product data integrity after creation and updates.
3. **UT-19:** Validate proper deletion cascades (e.g., delete category, not products).
4. **UT-20:** Test job listings table correctly stores job post data.
5. **UT-21:** Check foreign key constraints between users and jobs/applications.

5.4 Integration Testing

Integration testing ensures that different modules or components of a software application, such as BizNex, interact with each other effectively. BizNex uses Node.js[3] for the backend and a web-based frontend to facilitate business operations like user registration, payment management, employee tracking, and product listing.

In this phase, we validate the interaction between frontend inputs and backend functionalities. Below are some example test scenarios for integration:

Backend-Frontend Integration Testing:

1. **Client Sign-Up Integration:**
   - Verify that email and phone OTP requests initiated from the frontend are correctly handled and verified by the backend.
   - Confirm that verified details are successfully used to create a new client entry in the database.
   - Ensure smooth redirection to the profile completion page after successful verification.
2. **Client Authentication Integration:**
   - Ensure the login form (frontend) correctly sends login credentials to the backend.
   - Verify that token-based session creation and redirection to the dashboard occurs on successful login.
   - Test proper handling and display of error messages for incorrect email or password.
3. **Dashboard Integration:**
   - Test the communication between the frontend dashboard and backend API for fetching income, expenses, and orders.
   - Validate that graphical data is rendered accurately based on data grouped by time frames.

4. **Payment Management Integration:**

   - Verify that the payment form submits data correctly and triggers the corresponding backend functions.
   - Confirm that payment updates and deletions are reflected instantly on the frontend.

5. **Employee Management Integration:**

   - Ensure that employee records added via the frontend are saved correctly through the backend API.
   - Validate that edit and delete actions on employees sync properly between backend and frontend.

6. **Product and Category Integration:**

   - Test the integration for uploading product images to S3[12] and storing URLs via backend.
   - Confirm that product-category relationships are respected and displayed correctly.

7. **Job Listing Integration:**

   - Validate job posting and applicant viewing flows between the frontend and backend.
   - Ensure that application statuses are fetched, updated, and displayed accurately to both clients and job users.

These integration scenarios help ensure that each component of the BizNex platform works harmoniously. Successful integration testing guarantees that the user experience is consistent and reliable across various modules.

5.5 Functional Testing

Functional Testing is done to verify that all the functionalities specified in software requirements are met. The functional testing can be done on various scenarios like:

1. Client Sign-Up

- Verifying that OTP is sent to the email address entered by the user.
- Verifying that the email OTP entered by the user is successfully verified.
- Verifying that OTP is sent to the phone number entered by the user.
- Verifying that the phone OTP entered by the user is successfully verified.
- Ensuring redirection to profile page only after email and phone are verified.
- Verifying that complete client data is submitted and stored in the backend database.

2. Client Authentication

- Checking login functionality with valid email and password.
- Verifying correct error message for invalid password.
- Verifying error message display when email is not found.
- Verifying that user is redirected to dashboard after successful login and token is stored.

3. Client Google Authentication

- Verifying redirection to Google account selector on click of Google login.
- Checking that access is granted if email exists in database.
- Ensuring proper error message is shown if email does not exist.
- Verifying token storage and dashboard redirection after login.

4. Dashboard

- Verifying that income, expense, profit, and order data is fetched and displayed correctly.
- Ensuring percentage change of metrics compared to previous week is calculated accurately.
- Verifying that graph duration filters fetch and display correct data.

5. Payments

- Ensuring new payment entries are added using form submission.
- Verifying that all payments are displayed in tabular format.
- Checking that edit operation updates the backend and refreshes the table.
- Verifying deletion of a payment entry removes it from database and UI.

6. Employees

- Verifying that employee data is correctly added via form.
- Ensuring employee data is displayed in table format.
- Verifying that employee records can be updated through edit option.
- Ensuring deletion of employee removes it from backend and table.

7. Products

- Verifying that product image is uploaded to S3[12] and URL is received.
- Checking that product data with image URL is stored in the backend.
- Ensuring correct retrieval and display of product data in table format.
- Verifying edit and delete functionalities update and remove products appropriately.

8. Product Category

- Ensuring new category entries are added using form and stored in database.
- Verifying categories are displayed in tabular format.
- Verifying that category cannot be deleted if products exist under it.

9. Job-Listing (Client Side)

- Ensuring job entries are submitted correctly via form and stored in database.
- Verifying job data and applicant details are fetched and displayed properly.
- Ensuring job and applicant entries can be deleted and status can be updated to “Rejected”.
- Verifying view functionality shows detailed applicant information.

10. Job-Listing (Job User Side)

- Verifying job listings are correctly retrieved and displayed to job users.
- Ensuring application data is sent and stored in database on apply action.
- Verifying that application status is fetched and shown correctly.
- Verifying resume upload to S3[12] and URL is correctly stored in backend.

Chapter 6

Result

6.1 Result Summary

There are several key outcomes and conclusions derived from our project aimed at empowering local entrepreneurs by providing a digital platform for business growth. This summary highlights the results observed during development and testing stages.

Main Findings

The BizNex platform successfully functions as a centralized marketplace where users can manage their products, access business support, and engage customers without technical barriers.

The integration of AWS S3 for file storage and PostgreSQL via EC2 ensures robust, scalable backend infrastructure while maintaining data reliability.

Users have shown positive feedback regarding the simplicity of onboarding, localized language support, and the intuitive interface that caters to diverse user groups.

The platform effectively bridges the gap between business owners and support institutions, enhancing outreach and operational efficiency.

Recommendations for Future Development

Introduce multilingual chatbot support for real-time assistance and user engagement.

Implement detailed analytics dashboards to help users track sales, customer behavior, and optimize offerings.

Add integrations with more digital payment gateways to widen financial accessibility.

These results, aligned with the project's core objectives, confirm that BizNex is an effective digital solution tailored for underserved communities. The system has shown strong promise in usability and impact, with opportunities for expansion and improvement noted in the appendices for further data and analysis.

6.2 Comparison with Existing Systems

Traditional methods for promoting and managing small-scale businesses often rely on offline processes, such as word-of-mouth marketing, manual inventory tracking, and limited regional customer reach. These practices result in inefficient business growth, poor scalability, and restricted access to broader markets. The BizNex platform addresses these challenges by offering:

Digital Onboarding: Simplifies the process for local entrepreneurs to create an online presence without requiring technical skills.
Centralized Marketplace: Provides a unified platform where users can showcase products, access support services, and reach a larger customer base.
Automated Listings: Enables vendors to update inventory and product details with ease, minimizing errors and saving time.
Language & UI Accessibility: Offers a user-friendly interface in regional languages to increase adoption in rural and semi-urban areas.
Integrated Support System: Connects users to NGOs, SHGs, and support centers for financial literacy, business training, and technical assistance.

Chapter 7
Conclusion

BizNex is a digital enablement platform designed to support small-scale and rural entrepreneurs by bridging the gap in digital access and market connectivity. Focused on inclusivity, the platform allows local business owners to register, manage products or services, and engage with broader markets through an intuitive interface. Key features include vendor onboarding, catalog management, language localization, and community-based support. The system's backend ensures secure data flow and scalability, while usability testing informed iterative improvements to align with user needs. The platform integrates technologies like Firebase, cloud hosting, and responsive UI to make it accessible even for first-time digital users. A mobile-first approach further enhances usability in semi-urban and rural areas where desktop access may be limited.

BizNex has been designed to minimize management overhead while enabling smarter, data-driven decision-making for entrepreneurs. It encourages collaboration between business owners, NGOs, and government agencies, creating a shared digital ecosystem. With modules for job listings, inventory, and financial reporting, users can manage operations end-to-end. The application's scalability and modular design also support future enhancements such as AI-driven insights and integration with public schemes. Overall, BizNex empowers micro-enterprises to thrive in a digital economy by simplifying workflows, expanding reach, and promoting sustainable growth.

References

[1] React Documentation: https://reactjs.org/docs/getting-started.html

[2] Express Documentation: https://expressjs.com/en/4x/api.html

[3] Node Documentation: https://nodejs.org/en/docs/

[4] PostgreSQL Documentation: https://www.postgresql.org/docs/current/index.html

[5] pg (PostgreSQL client for Node.js): https://node-postgres.com/

[6] Axios (HTTP client for Node.js): https://axios-http.com/docs/intro

[7] bcrypt.js (Password hashing for Node.js): https://www.npmjs.com/package/bcryptjs

[8] GitHub: https://github.com/

[9] ThunderClient (API testing tool): https://www.thunderclient.io/

[10] Vercel: https://vercel.com/docs

[11] Amazon EC2 Documentation: https://docs.aws.amazon.com/ec2/

[12] Amazon S3 Documentation: https://docs.aws.amazon.com/s3/

[13] Render Documentation: https://render.com/docs

Appendices

Appendix A

CODE

Sign-up Client

```
Import required packages
const express = require('express');
const { Pool } = require('pg');
require('dotenv').config();
const router = express.Router();
const bcrypt = require('bcrypt');
const nodemailer = require("nodemailer");
const twilio = require('twilio');
const client = new twilio(process.env.TWILIO_ACCOUNT_SID, process.
    env.TWILIO_AUTH_TOKEN);
const crypto = require("crypto");
const port = 5000;

const masterPool = new Pool({
    user: process.env.DB_USER,
    password: process.env.DB_PASSWORD,
    host: process.env.DB_HOST,
    port: process.env.DB_PORT,
    database: process.env.DB_NAME,
  });

router.use(express.json());


const transporter = nodemailer.createTransport({
  service: "Gmail",
  auth: {
    user: process.env.EMAIL_USER,
    pass: process.env.EMAIL_PASS,
  },
});
```

```

const generateOTP = () => crypto.randomInt(100000, 999999).toString();

//  1. Send Email OTP
router.post("/send-email-otp", async (req, res) => {
  const { email} = req.body;
  if (!email) return res.status(400).json({ error: "Email is required" });

  const result=await masterPool.query('SELECT * FROM clients WHERE email = $1', [email]);
  if (result.rows.length === 0){
    try {
        const emailOtp = generateOTP();
        const expiresAt = new Date(Date.now() + 10 * 60 * 1000);

        await masterPool.query('DELETE FROM client_verifications WHERE email = $1', [email]);

        await masterPool.query(
          `INSERT INTO client_verifications (email, email_otp, expires_at)
           VALUES ($1, $2, $3)
           ON CONFLICT (email)
           DO UPDATE SET email_otp = $2, expires_at = $3, is_email_verified = FALSE;`,
          [email, emailOtp, expiresAt]
        );2

        await transporter.sendMail({
          from: `"Service Platform" <${process.env.EMAIL_USER}>`,
          to: email,
          subject: "Your Email OTP",
          text: `Your email OTP is: ${emailOtp}. It is valid for 10 minutes.`,
        });
        console.log(` Sent OTP to ${email}: ${emailOtp}`);
        res.json({ message: "Email OTP sent successfully." });
```

```
      } catch (error) {
        console.error("Error sending email OTP:", error);
        res.status(500).json({ error: "Internal server error" });
      }
  }
  else{
    console.error("Email already in use:", error);
    res.status(409).json({ error: "Email already in use" });
  }

});

//  2. Send Phone OTP

router.post("/send-phone-otp", async (req, res2) => {
  const { email, phone } = req.body;
  if (!email || !phone) return res.status(400).json({ error: "Email
     and phone are required" });

  try {
    const userCheck = await masterPool.query(
      `SELECT * FROM client_verifications WHERE email = $1 ;`,
      [email]
    );

    if (userCheck.rows.length === 0) {
      return res.status(404).json({ error: "User not found." });
    }

    const phoneOtp = generateOTP();
    const expiresAt = new Date(Date.now() + 10 * 60 * 1000);

    await masterPool.query(
      `UPDATE client_verifications SET phone_otp = $1, expires_at =
     $2, phone= $3 ,is_phone_verified = FALSE
       WHERE email = $4 ;`,
      [phoneOtp, expiresAt, phone, email]
    );

    await client.messages.create({
      body: ‘Your OTP is: ${phoneOtp}‘,
      from: process.env.TWILIO_PHONE_NUMBER,
      to: phone,
    });

    console.log(‘       Sent OTP to ${phone}: ${phoneOtp}‘);
    res.json({ message: "Phone OTP sent successfully." });
  } catch (error) {
    console.error("Error sending phone OTP:", error);
    res.status(500).json({ error: "Internal server error" });
  }
});


//  3. Verify Email OTP
router.post("/verify-email-otp", async (req, res) => {
  const { email, emailOtp } = req.body;
  if (!email || !emailOtp) return res.status(400).json({ error: "Email and OTP are required." });

  try {
    const result = await masterPool.query(‘SELECT * FROM client_verifications WHERE email = $1;‘, [email]);

    if (result.rows.length === 0) return res.status(400).json({ error: "Invalid email." });

    const { email_otp, expires_at, is_email_verified } = result.rows[0];

    if (is_email_verified) return res.status(400).json({ error: "Email already verified." });
    if (new Date() > new Date(expires_at)) return res.status(400).json({ error: "OTP expired." });
    if (email_otp !== emailOtp) return res.status(400).json({ error: "Invalid email OTP." });

    await masterPool.query(‘UPDATE client_verifications SET is_email_verified = TRUE WHERE email = $1;‘, [email]);

    res.json({ message: "Email verified successfully." });
  } catch (error) {
    console.error("Error verifying email OTP:", error);
    res.status(500).json({ error: "Internal server error" });
  }
});

//  4. Verify Phone OTP
router.post("/verify-phone-otp", async (req, res) => {
  const { email, phoneOtp } = req.body;
  if (!email || !phoneOtp) return res.status(400).json({ error: "Email and phone OTP are required." });

  try {
    const result = await masterPool.query(‘SELECT * FROM client_verifications WHERE email = $1;‘, [email]);

    if (result.rows.length === 0) return res.status(400).json({ error: "Invalid email." });

    const { phone_otp, expires_at, is_phone_verified, is_email_verified } = result.rows[0];

    if (is_phone_verified) return res.status(400).json({ error: "Phone already verified." });
    if (new Date() > new Date(expires_at)) return res.status(400).json({ error: "OTP expired." });
    if (phone_otp !== phoneOtp) return res.status(400).json({ error: "Invalid phone OTP." });

    await masterPool.query(‘UPDATE client_verifications SET is_phone_verified = TRUE WHERE email = $1;‘, [email]);


    res.json({ message: "Phone verified successfully." });
  } catch (error) {
    console.error("Error verifying phone OTP:", error);
    res.status(500).json({ error: "Internal server error" });
  }
});

async function hashPassword(password) {
    const saltRounds = 10; // Higher is more secure but slower
    try {
      const hashedPassword = await bcrypt.hash(password, saltRounds)
    ;
      console.log("Hashed Password:", hashedPassword);
      return hashedPassword;
    } catch (err) {
      console.error("Error hashing password:", err);
    }
  }

router.post('/create-client', async (req, res) => {
  const { username,ownername, address, email,business_category,phone
    , password } = req.body;
  if (!username || !email || !password ||!ownername||!address||!
    business_category||!phone) {
    return res.status(400).json({ error: 'Missing required fields' ,
     message : {username,ownername,address, email, phone ,password}
    });
  }

  const result=await masterPool.query('SELECT * FROM clients WHERE
    email = $1', [email]);
  if (result.rows.length === 0){
      try {

        const result = await masterPool.query(`SELECT * FROM
    client_verifications WHERE email = $1;`, [email]);

        if (result.rows.length === 0) return res.status(400).json({
    error: "Invalid email." });

    const { phone_otp, expires_at, is_phone_verified, is_email_verified } = result.rows[0];

    const hpassword= await hashPassword(password);

    if(is_email_verified && is_phone_verified){
        const userResult = await masterPool.query(
            'INSERT INTO clients (client_name,owner_name, email, address,business_category,status, password_hash,ph_no) VALUES ($1, $2, $3,$4,$5,$6,$7,$8) RETURNING client_id;',
            [username,ownername, email, address, business_category,'active',hpassword,phone]
          );

          const userId = userResult.rows[0].client_id;

          const dbName = `user_db_${userId}`;

          const dbUpdateResult = await masterPool.query(
          'UPDATE clients SET db_name = $1 WHERE client_id = $2 RETURNING *;',
          [dbName, userId]
          );

          if (userResult.rowCount === 0) {
            console.log('No client found with that ID.');
          } else {
            console.log('Updated client:', userResult.rows[0]);
          }
          await masterPool.query(`CREATE DATABASE ${dbName};`);

          const userPool = new Pool({
            user: process.env.DB_USER,
            password: process.env.DB_PASSWORD,
            host: process.env.DB_HOST,
            port: process.env.DB_PORT,
            database: dbName,
          });

```

```
            // Create enums, tables, and triggers
            await userPool.query(`
              CREATE TYPE cart_status AS ENUM ('pending', 'processing', 'completed', 'cancelled');
              CREATE TYPE cart_log_status AS ENUM ('Created', 'processing', 'shipped', 'delivered', 'cancelled');
              CREATE TYPE bill_status AS ENUM ('paid', 'pending', 'failed');
              CREATE TYPE bill_log_status AS ENUM ('generated', 'paid', 'refunded');
              CREATE TYPE web_bill_status AS ENUM ('paid', 'pending', 'failed');
              CREATE TYPE web_bill_log_status AS ENUM ('generated', 'paid', 'refunded');
              CREATE TYPE order_status AS ENUM ('pending', 'processing', 'completed', 'cancelled');
              CREATE TYPE order_log_status AS ENUM ('created', 'processing', 'shipped', 'cancelled', 'delivered');
              CREATE TYPE payment_status AS ENUM ('Pending', 'Paid', 'Overdue');
              CREATE TYPE payment_method AS ENUM ('Cash', 'Credit Card', 'Bank Transfer', 'Other');
              CREATE TYPE product_status AS ENUM ('active', 'inactive');
              CREATE TYPE salary_status AS ENUM ('pending', 'paid', 'failed');
              CREATE TYPE payment_method_salary AS ENUM ('bank_transfer', 'cash', 'cheque', 'upi');

             CREATE TABLE products (
              id SERIAL PRIMARY KEY,
              name VARCHAR(255) NOT NULL,
              description TEXT,
              price DECIMAL(10,2) NOT NULL CHECK (price >= 0),
              created_at TIMESTAMP DEFAULT NOW(),
              updated_at TIMESTAMP DEFAULT NOW(),
              quantity INT NOT NULL CHECK (quantity >= 0),
              barcode VARCHAR(50) UNIQUE NOT NULL,
              category VARCHAR(100) NOT NULL,
              status VARCHAR(20) DEFAULT 'active',
              bestseller BOOLEAN DEFAULT FALSE,
              imageUrl TEXT,
              type VARCHAR(50)
            );


              CREATE TABLE category (
                  category_id SERIAL PRIMARY KEY,
                  category VARCHAR(255) NOT NULL,
                  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
              );



              CREATE TABLE employees (
                  id SERIAL PRIMARY KEY,
                  first_name VARCHAR(100) NOT NULL,
                  last_name VARCHAR(100) NOT NULL,
                  email VARCHAR(255) UNIQUE NOT NULL,
                  phone VARCHAR(20) UNIQUE NOT NULL,
                  position VARCHAR(100) NOT NULL,
                  salary DECIMAL(10,2) NOT NULL CHECK (salary >= 0),
                  bank_name VARCHAR(255) NOT NULL,
                  bank_account_number VARCHAR(50) UNIQUE NOT NULL,
                  ifsc_code VARCHAR(11) NOT NULL,
                  created_at TIMESTAMP DEFAULT NOW(),
                  updated_at TIMESTAMP DEFAULT NOW(),
                  joining_date DATE
              );

              CREATE TABLE salaries (
                  id SERIAL PRIMARY KEY,
                  employee_id INTEGER NOT NULL REFERENCES employees(id) ON DELETE CASCADE,
                  salary_amount DECIMAL(10,2) NOT NULL CHECK (salary_amount >= 0),
                  payment_date DATE NOT NULL DEFAULT CURRENT_DATE,
                  salary_month DATE NOT NULL,
                  payment_method payment_method_salary NOT NULL,
                  created_at TIMESTAMP DEFAULT NOW(),
                  updated_at TIMESTAMP DEFAULT NOW()
              );

              CREATE TABLE cart (
                cart_id SERIAL PRIMARY KEY,
                total_price DECIMAL(10, 2) NOT NULL,
                status cart_status,
                created_at TIMESTAMP DEFAULT NOW()
              );

              CREATE TABLE cart_logs (
                log_id SERIAL PRIMARY KEY,
                cart_id INT NOT NULL,
                status cart_log_status,
                updated_at TIMESTAMP DEFAULT NOW(),
                CONSTRAINT fk_cart_log FOREIGN KEY (cart_id) REFERENCES cart(cart_id)
              );

              CREATE TABLE cart_item (
                cart_item_id SERIAL PRIMARY KEY,
                cart_id INT NOT NULL,
                product_id INT NOT NULL,
                quantity INT NOT NULL CHECK (quantity > 0),
                unit_price DECIMAL(10,2) NOT NULL CHECK (unit_price >= 0), -- Store price at the time of purchase
                CONSTRAINT fk_cart FOREIGN KEY (cart_id) REFERENCES cart(cart_id) ON DELETE CASCADE,
                CONSTRAINT fk_product FOREIGN KEY (product_id) REFERENCES products(id) ON DELETE CASCADE
              );

              CREATE TABLE bills (
                bill_id SERIAL PRIMARY KEY,
                cart_id INT NOT NULL,
                total_amount DECIMAL(10, 2) NOT NULL,
                payment_status bill_status,
                payment_method VARCHAR(50) CHECK (payment_method IN ('card', 'UPI', 'cash')),
                generated_at TIMESTAMP DEFAULT NOW(),
                CONSTRAINT fk_cart_bill FOREIGN KEY (cart_id) REFERENCES cart(cart_id)
              );

              CREATE TABLE bill_logs (
                bill_log_id SERIAL PRIMARY KEY,
                bill_id INT NOT NULL,
                status bill_log_status,
                updated_at TIMESTAMP DEFAULT NOW(),
                CONSTRAINT fk_bill FOREIGN KEY (bill_id) REFERENCES bills(bill_id)
              );

              CREATE TABLE web_bills (
                web_bill_id SERIAL PRIMARY KEY,
                order_id INT NOT NULL,
                total_amount DECIMAL(10, 2) NOT NULL,
                payment_status web_bill_status,
                payment_method VARCHAR(50) CHECK (payment_method IN ('card', 'UPI', 'cash')),
                generated_at TIMESTAMP DEFAULT NOW(),
                CONSTRAINT fk_cart_web_bill FOREIGN KEY (cart_id) REFERENCES cart(cart_id)
              );


              CREATE TABLE web_bill_logs (
                web_log_id SERIAL PRIMARY KEY,
                bill_id INT NOT NULL,
                status web_bill_log_status,
                updated_at TIMESTAMP DEFAULT NOW(),
                CONSTRAINT fk_web_bill FOREIGN KEY (bill_id) REFERENCES web_bills(web_bill_id)
              );


              CREATE TABLE orders (
                order_id SERIAL PRIMARY KEY,
                total_price DECIMAL(10, 2) NOT NULL,
                status order_status,
                created_at TIMESTAMP DEFAULT NOW()
              );

              CREATE TABLE order_logs (
                log_id SERIAL PRIMARY KEY,
                order_id INT NOT NULL,
                status order_log_status,
                updated_at TIMESTAMP DEFAULT NOW(),
                CONSTRAINT fk_order_log FOREIGN KEY (order_id) REFERENCES orders(order_id)
              );

              CREATE TABLE order_item (
                  order_item_id SERIAL PRIMARY KEY,
                  order_id INT NOT NULL,
                  product_id INT NOT NULL,
                  quantity INT NOT NULL CHECK (quantity > 0),
                  unit_price DECIMAL(10, 2) NOT NULL CHECK (unit_price >= 0),
                  CONSTRAINT fk_order FOREIGN KEY (order_id) REFERENCES orders(order_id) ON DELETE CASCADE,
                  CONSTRAINT fk_product FOREIGN KEY (product_id) REFERENCES products(id) ON DELETE CASCADE
              );


              CREATE TABLE documents (
                document_id SERIAL PRIMARY KEY,
                file_url TEXT NOT NULL,
                description TEXT,
                created_at TIMESTAMP DEFAULT NOW()
              );

              CREATE TABLE man_expenses (
                id SERIAL PRIMARY KEY,
                type VARCHAR(100) NOT NULL,
                description TEXT,
                amount NUMERIC(10,2) NOT NULL CHECK (amount >= 0),
                payment_method VARCHAR(50) CHECK (payment_method
  IN ('Cash', 'Card', 'UPI', 'Bank Transfer', 'Other')),
                expense_date DATE NOT NULL,
                created_at TIMESTAMP DEFAULT NOW()
              );

              CREATE TABLE man_incomes (
                id SERIAL PRIMARY KEY,
                type VARCHAR(100) NOT NULL,
                description TEXT,
                amount NUMERIC(10,2) NOT NULL CHECK (amount >= 0),
                payment_method VARCHAR(50) CHECK (payment_method
  IN ('Cash', 'Card', 'UPI', 'Bank Transfer', 'Other')),
                income_date DATE NOT NULL,
                created_at TIMESTAMP DEFAULT NOW()
              );

              CREATE TABLE accounts_payable (
                id SERIAL PRIMARY KEY,
                account_name VARCHAR(255) NOT NULL,
                amount DECIMAL(10,2) NOT NULL,
                payment_date DATE NOT NULL,
                payment_method payment_method NOT NULL,
                status payment_status DEFAULT 'Pending',
                created_at TIMESTAMP DEFAULT NOW()
              );

              CREATE TABLE accounts_receivable (
                id SERIAL PRIMARY KEY,
                account_name VARCHAR(255) NOT NULL,
                amount DECIMAL(10,2) NOT NULL,
                due_date DATE NOT NULL,
                status payment_status DEFAULT 'Pending',
                created_at TIMESTAMP DEFAULT NOW()
              );



              CREATE TABLE accounts_payable_log (
                log_id SERIAL PRIMARY KEY,
                payable_id INT NOT NULL,
                account_name VARCHAR(255) NOT NULL,
                amount DECIMAL(10,2) NOT NULL,
                payment_date DATE NOT NULL,
                payment_method payment_method NOT NULL,
                status payment_status NOT NULL,
                changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                operation_type VARCHAR(10) CHECK (operation_type IN ('INSERT', 'UPDATE', 'DELETE'))
              );

              CREATE TABLE accounts_receivable_log (
                log_id SERIAL PRIMARY KEY,
                receivable_id INT NOT NULL,
                account_name VARCHAR(255) NOT NULL,
                amount DECIMAL(10,2) NOT NULL,
                due_date DATE NOT NULL,
                status payment_status NOT NULL,
                changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                operation_type VARCHAR(10) CHECK (operation_type IN ('INSERT', 'UPDATE', 'DELETE'))
              );


              -- Trigger functions for logs

              CREATE OR REPLACE FUNCTION log_cart_changes() RETURNS TRIGGER AS $$
              BEGIN
                INSERT INTO cart_logs(cart_id, status) VALUES (NEW.cart_id, NEW.status);
                RETURN NEW;
              END;
              $$ LANGUAGE plpgsql;
```

```

            CREATE TRIGGER cart_log_trigger
            AFTER INSERT OR UPDATE ON cart
            FOR EACH ROW EXECUTE FUNCTION log_cart_changes();

            CREATE OR REPLACE FUNCTION log_bill_changes() RETURNS TRIGGER AS $$
            BEGIN
              INSERT INTO bill_logs(bill_id, status) VALUES (NEW.bill_id, NEW.payment_status);
              RETURN NEW;
            END;
            $$ LANGUAGE plpgsql;

            CREATE TRIGGER bill_log_trigger
            AFTER INSERT OR UPDATE ON bills
            FOR EACH ROW EXECUTE FUNCTION log_bill_changes();

            CREATE OR REPLACE FUNCTION log_web_bill_changes() RETURNS TRIGGER AS $$
            BEGIN
              INSERT INTO web_bill_logs(bill_id, status) VALUES (NEW.web_bill_id, NEW.payment_status);
              RETURN NEW;
            END;
            $$ LANGUAGE plpgsql;

            CREATE TRIGGER web_bill_log_trigger
            AFTER INSERT OR UPDATE ON web_bills
            FOR EACH ROW EXECUTE FUNCTION log_web_bill_changes();

            CREATE OR REPLACE FUNCTION log_order_changes() RETURNS TRIGGER AS $$
            BEGIN
              INSERT INTO order_logs(order_id, status) VALUES (NEW.order_id, NEW.status);
              RETURN NEW;
            END;
```

```
                $$ LANGUAGE plpgsql;

                CREATE TRIGGER order_log_trigger
                AFTER INSERT OR UPDATE ON orders
                FOR EACH ROW EXECUTE FUNCTION log_order_changes();


                CREATE OR REPLACE FUNCTION
    log_accounts_payable_changes()
                RETURNS TRIGGER AS $$
                BEGIN
                    IF TG_OP = 'DELETE' THEN
                        INSERT INTO accounts_payable_log (payable_id
    , account_name, amount, payment_date, payment_method, status,
    operation_type)
                        VALUES (OLD.id, OLD.account_name, OLD.amount
    , OLD.payment_date, OLD.payment_method, OLD.status, 'DELETE');
                    ELSIF TG_OP = 'UPDATE' THEN
                        INSERT INTO accounts_payable_log (payable_id
    , account_name, amount, payment_date, payment_method, status,
    operation_type)
                        VALUES (NEW.id, NEW.account_name, NEW.amount
    , NEW.payment_date, NEW.payment_method, NEW.status, 'UPDATE');
                    ELSE
                        INSERT INTO accounts_payable_log (payable_id
    , account_name, amount, payment_date, payment_method, status,
    operation_type)
                        VALUES (NEW.id, NEW.account_name, NEW.amount
    , NEW.payment_date, NEW.payment_method, NEW.status, 'INSERT');
                    END IF;
                    RETURN NULL;
                END;
                $$ LANGUAGE plpgsql;

                CREATE TRIGGER trigger_accounts_payable
                AFTER INSERT OR UPDATE OR DELETE ON accounts_payable
                FOR EACH ROW EXECUTE FUNCTION
    log_accounts_payable_changes();

```

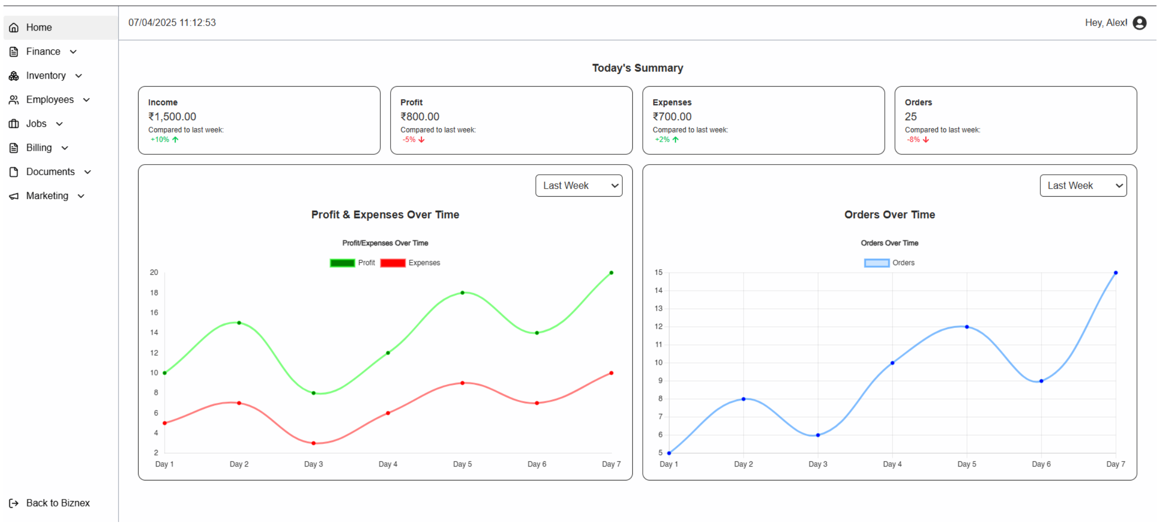
```

                CREATE OR REPLACE FUNCTION log_accounts_receivable_changes()
                RETURNS TRIGGER AS $$
                BEGIN
                    IF TG_OP = 'DELETE' THEN
                        INSERT INTO accounts_receivable_log (receivable_id, account_name, amount, due_date, status, operation_type)
                        VALUES (OLD.id, OLD.account_name, OLD.amount, OLD.due_date, OLD.status, 'DELETE');
                    ELSIF TG_OP = 'UPDATE' THEN
                        INSERT INTO accounts_receivable_log (receivable_id, account_name, amount, due_date, status, operation_type)
                        VALUES (NEW.id, NEW.account_name, NEW.amount, NEW.due_date, NEW.status, 'UPDATE');
                    ELSE
                        INSERT INTO accounts_receivable_log (receivable_id, account_name, amount, due_date, status, operation_type)
                        VALUES (NEW.id, NEW.account_name, NEW.amount, NEW.due_date, NEW.status, 'INSERT');
                    END IF;
                    RETURN NULL;
                END;
                $$ LANGUAGE plpgsql;

                CREATE TRIGGER trigger_accounts_receivable
                AFTER INSERT OR UPDATE OR DELETE ON accounts_receivable
                FOR EACH ROW EXECUTE FUNCTION log_accounts_receivable_changes();


              `);

              await userPool.end();

              res.status(201).json({ message: 'User created and database initialized', dbName });

              }
              else
              {

                  console.error('Email or phone not verified.');
                  return res.status(400).json({ error: 'Email or phone not verified.' });


              }
            }
            catch (err) {
              console.error('Error creating user or database:', err);
              res.status(500).json({ error: 'Internal server error' });
            }
        }
        else{
          console.error("Email already in use:", error);
          res.status(409).json({ error: "Email already in use" });
        }
});
module.exports = router;
```

Login Client

```
const express = require('express');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const router = express.Router();
const masterPool = require('./master_db');
router.use(express.json());

const JWT_SECRET = process.env.JWT_SECRET || 'your_jwt_secret';

router.post("/login-client", async (req, res) => {
  console.log("Login client request received:", req.body);
  const { email, password } = req.body;

  if (!email || !password) {
    return res.status(400).json({ error: "Email and password are required" });
  }

  try {
    const result = await masterPool.query('SELECT * FROM clients WHERE email = $1', [email]);

    if (result.rows.length === 0) {
      return res.status(401).json({ error: 'Invalid email or password' });
    }

    const user = result.rows[0];

    const isMatch = await bcrypt.compare(password, user.password_hash);
    if (!isMatch) {
      return res.status(401).json({ error: 'Invalid email or password' });
    }

    const token = jwt.sign({
      id: user.client_id || user.id,
      email: user.email,
      dbname: user.db_name,
      userType: 'client'
    }, JWT_SECRET, { expiresIn: '1h' });

    res.status(200).json({
      message: 'Login successful',
      client: user.client_name,
      token
    });
  } catch (error) {
    console.error('Login error:', error);
    res.status(500).json({ error: 'Internal Server Error' });
  }
});

module.exports = router;
```

Appendix B

Screenshots



07/04/2025 11:12:53

Hey, Alex!

Today's Summary

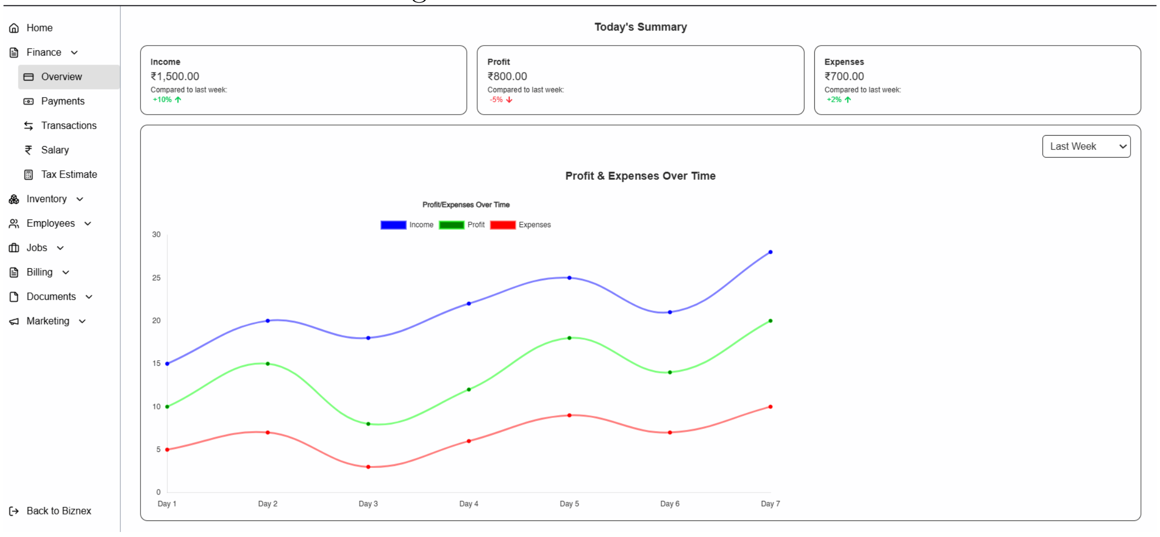
Income
₹1,500.00
Compared to last week:
+10% ↑

Profit
₹800.00
Compared to last week:
-5% ↓

Expenses
₹700.00
Compared to last week:
+2% ↑

Orders
25
Compared to last week:
-8% ↓

Figure B.1: Main Dashboard



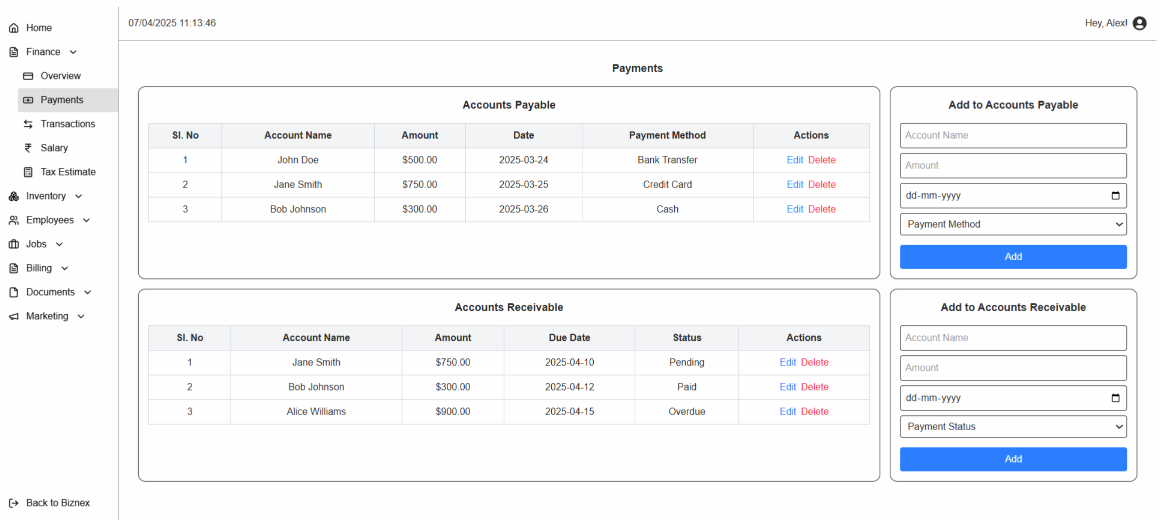
Today's Summary

Income
₹1,500.00
Compared to last week:
+10% ↑

Profit
₹800.00
Compared to last week:
-5% ↓

Expenses
₹700.00
Compared to last week:
+2% ↑

Figure B.2: Finance Overview



07/04/2025 11:13:46

Hey, Alex!

Payments

Accounts Payable

Sl. No	Account Name	Amount	Date	Payment Method	Actions
1	John Doe	$500.00	2025-03-24	Bank Transfer	Edit Delete
2	Jane Smith	$750.00	2025-03-25	Credit Card	Edit Delete
3	Bob Johnson	$300.00	2025-03-26	Cash	Edit Delete

Add to Accounts Payable

Account Name
Amount
dd-mm-yyyy
Payment Method
Add

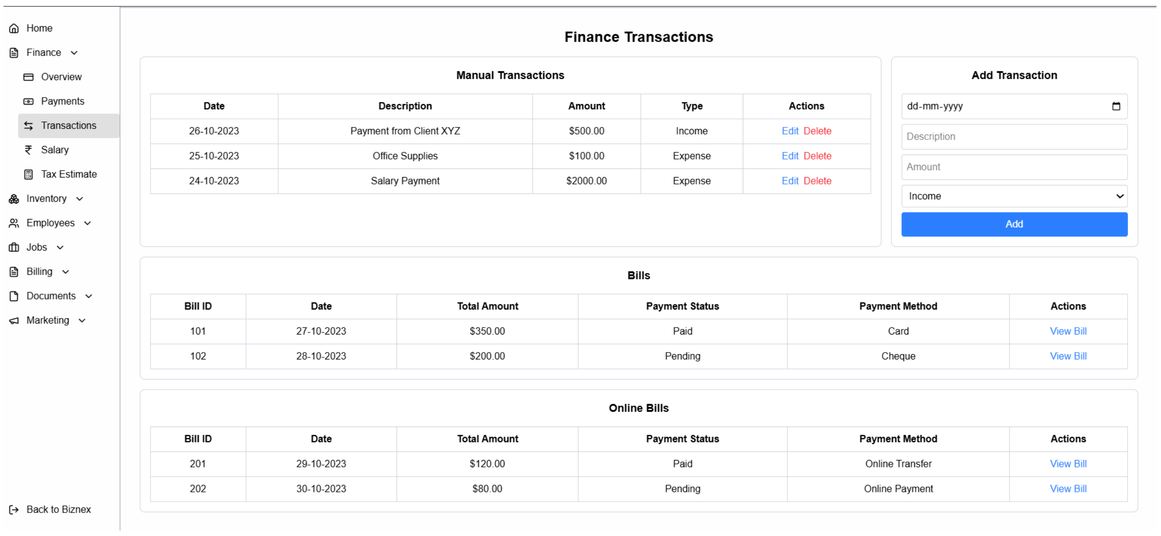
Accounts Receivable

Sl. No	Account Name	Amount	Due Date	Status	Actions
1	Jane Smith	$750.00	2025-04-10	Pending	Edit Delete
2	Bob Johnson	$300.00	2025-04-12	Paid	Edit Delete
3	Alice Williams	$900.00	2025-04-15	Overdue	Edit Delete

Add to Accounts Receivable

Account Name
Amount
dd-mm-yyyy
Payment Status
Add

Figure B.3: Finance Payments



Finance Transactions

Manual Transactions

Date	Description	Amount	Type	Actions
26-10-2023	Payment from Client XYZ	$500.00	Income	Edit Delete
25-10-2023	Office Supplies	$100.00	Expense	Edit Delete
24-10-2023	Salary Payment	$2000.00	Expense	Edit Delete

Add Transaction

dd-mm-yyyy
Description
Amount
Income
Add

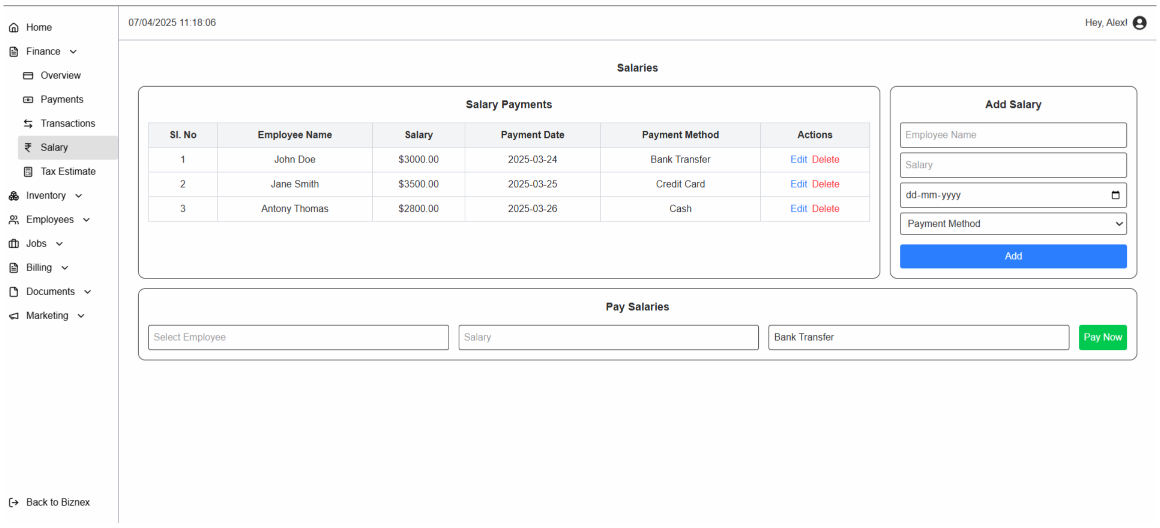
Bills

Bill ID	Date	Total Amount	Payment Status	Payment Method	Actions
101	27-10-2023	$350.00	Paid	Card	View Bill
102	28-10-2023	$200.00	Pending	Cheque	View Bill

Online Bills

Bill ID	Date	Total Amount	Payment Status	Payment Method	Actions
201	29-10-2023	$120.00	Paid	Online Transfer	View Bill
202	30-10-2023	$80.00	Pending	Online Payment	View Bill

Figure B.4: Finance Transactions



07/04/2025 11:18:06

Salaries

Salary Payments

Sl. No	Employee Name	Salary	Payment Date	Payment Method	Actions
1	John Doe	$3000.00	2025-03-24	Bank Transfer	Edit Delete
2	Jane Smith	$3500.00	2025-03-25	Credit Card	Edit Delete
3	Antony Thomas	$2800.00	2025-03-26	Cash	Edit Delete

Add Salary

Employee Name

Salary

dd-mm-yyyy

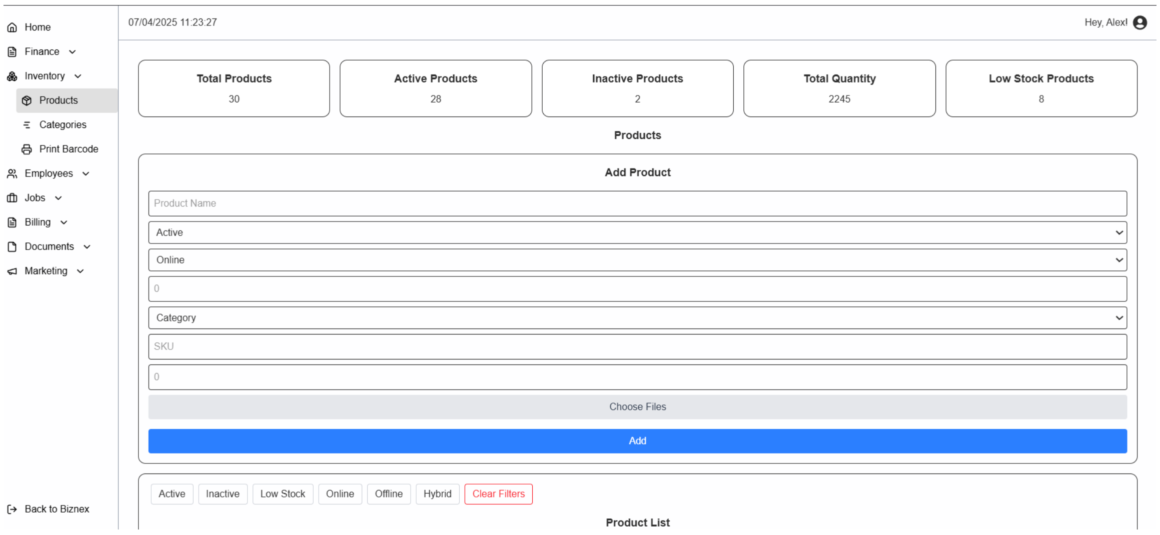
Payment Method

Add

Pay Salaries

Select Employee | Salary | Bank Transfer | Pay Now

Figure B.5: Salary Payment



07/04/2025 11:23:27

Total Products	Active Products	Inactive Products	Total Quantity	Low Stock Products
30	28	2	2245	8

Products

Add Product

Product Name

Active

Online

0

Category

SKU

0

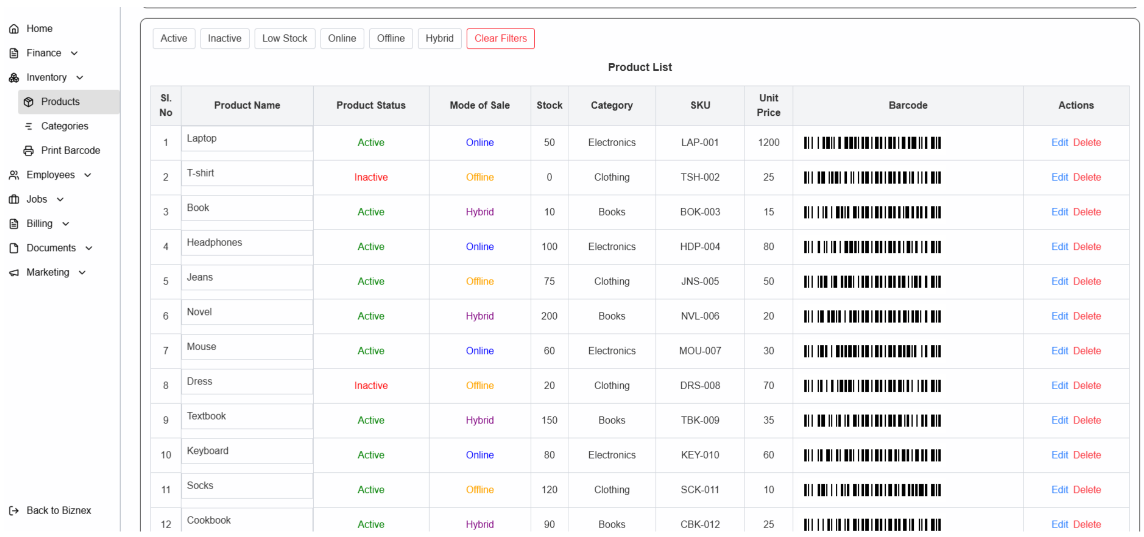
Choose Files

Add

Active | Inactive | Low Stock | Online | Offline | Hybrid | Clear Filters

Product List

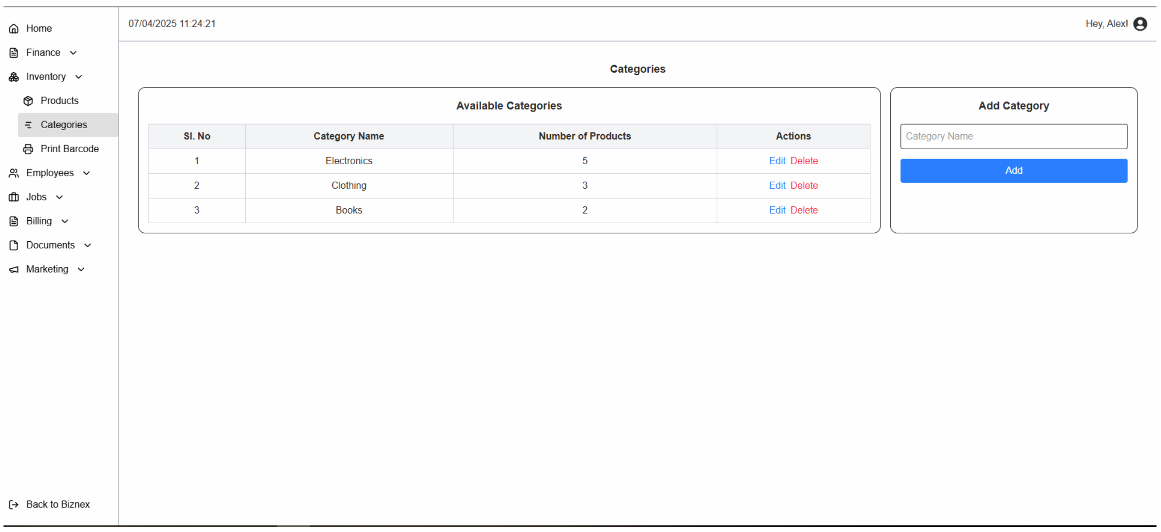
Figure B.6: Inventory 1



Product List

Sl. No	Product Name	Product Status	Mode of Sale	Stock	Category	SKU	Unit Price	Barcode	Actions
1	Laptop	Active	Online	50	Electronics	LAP-001	1200		Edit Delete
2	T-shirt	Inactive	Offline	0	Clothing	TSH-002	25		Edit Delete
3	Book	Active	Hybrid	10	Books	BOK-003	15		Edit Delete
4	Headphones	Active	Online	100	Electronics	HDP-004	80		Edit Delete
5	Jeans	Active	Offline	75	Clothing	JNS-005	50		Edit Delete
6	Novel	Active	Hybrid	200	Books	NVL-006	20		Edit Delete
7	Mouse	Active	Online	60	Electronics	MOU-007	30		Edit Delete
8	Dress	Inactive	Offline	20	Clothing	DRS-008	70		Edit Delete
9	Textbook	Active	Hybrid	150	Books	TBK-009	35		Edit Delete
10	Keyboard	Active	Online	80	Electronics	KEY-010	60		Edit Delete
11	Socks	Active	Offline	120	Clothing	SCK-011	10		Edit Delete
12	Cookbook	Active	Hybrid	90	Books	CBK-012	25		Edit Delete

Figure B.7: Inventory 2



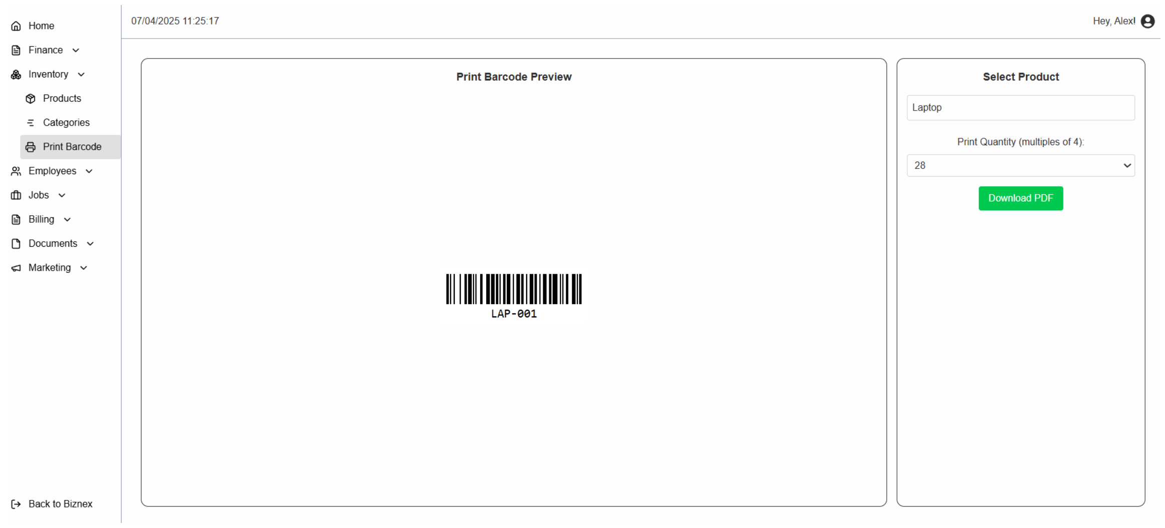
07/04/2025 11:24:21

Categories

Available Categories

Sl. No	Category Name	Number of Products	Actions
1	Electronics	5	Edit Delete
2	Clothing	3	Edit Delete
3	Books	2	Edit Delete

Figure B.8: Inventory Categories



07/04/2025 11:25:17

Hey, Alex!

Print Barcode Preview

LAP-001

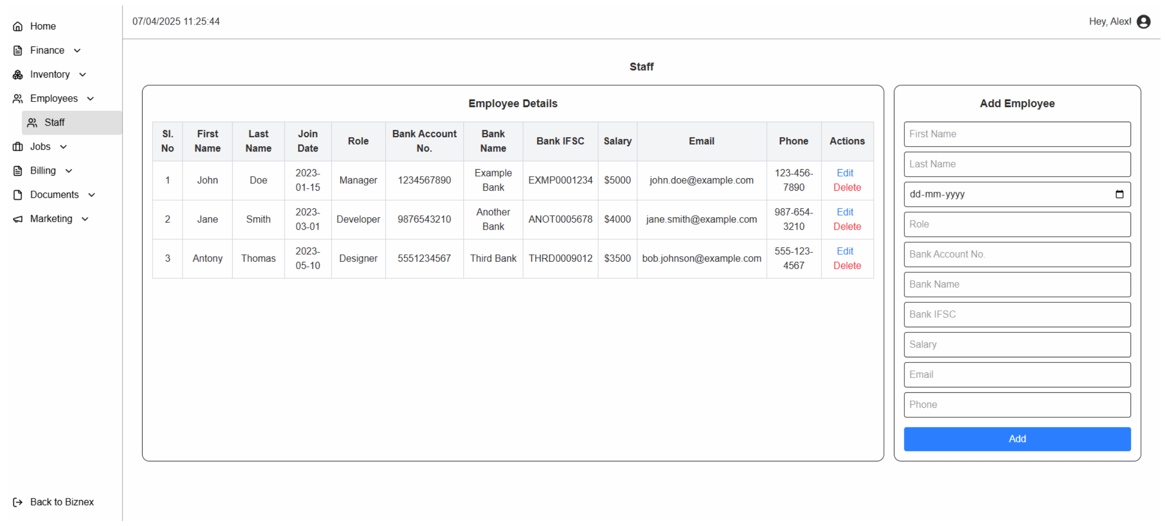
Select Product

Laptop

Print Quantity (multiples of 4):

28

Figure B.9: Inventory Barcode



07/04/2025 11:25:44

Hey, Alex!

Staff

Employee Details

Sl. No	First Name	Last Name	Join Date	Role	Bank Account No.	Bank Name	Bank IFSC	Salary	Email	Phone	Actions
1	John	Doe	2023-01-15	Manager	1234567890	Example Bank	EXMP0001234	$5000	john.doe@example.com	123-456-7890	Edit Delete
2	Jane	Smith	2023-03-01	Developer	9876543210	Another Bank	ANOT0005678	$4000	jane.smith@example.com	987-654-3210	Edit Delete
3	Antony	Thomas	2023-05-10	Designer	5551234567	Third Bank	THRD0009012	$3500	bob.johnson@example.com	555-123-4567	Edit Delete

Add Employee

First Name

Last Name

dd-mm-yyyy

Role

Bank Account No.

Bank Name

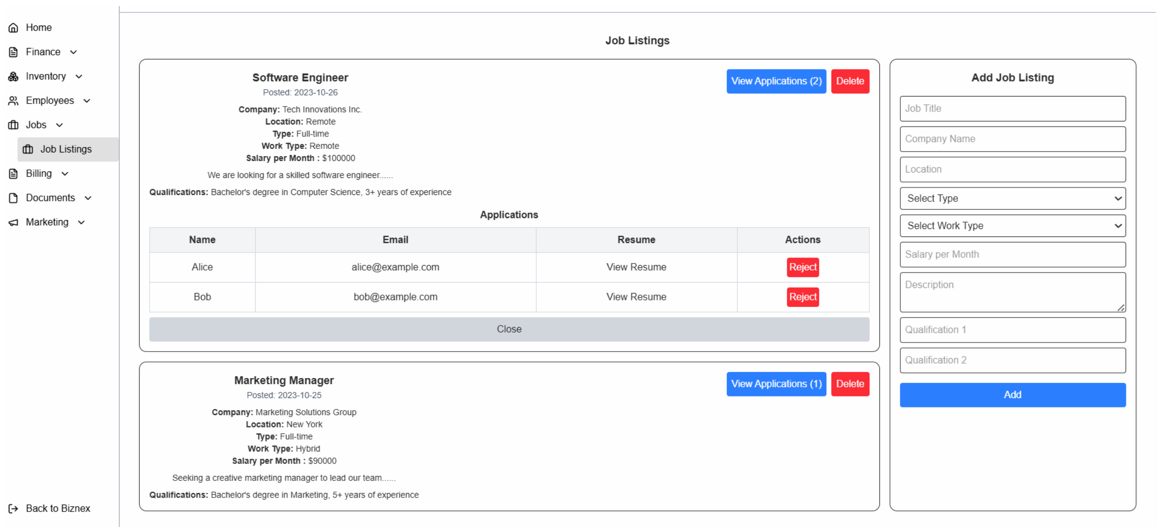
Bank IFSC

Salary

Email

Phone

Figure B.10: Employees



Job Listings

Software Engineer
Posted: 2023-10-26
Company: Tech Innovations Inc.
Location: Remote
Type: Full-time
Work Type: Remote
Salary per Month : $100000
We are looking for a skilled software engineer......
Qualifications: Bachelor's degree in Computer Science, 3+ years of experience

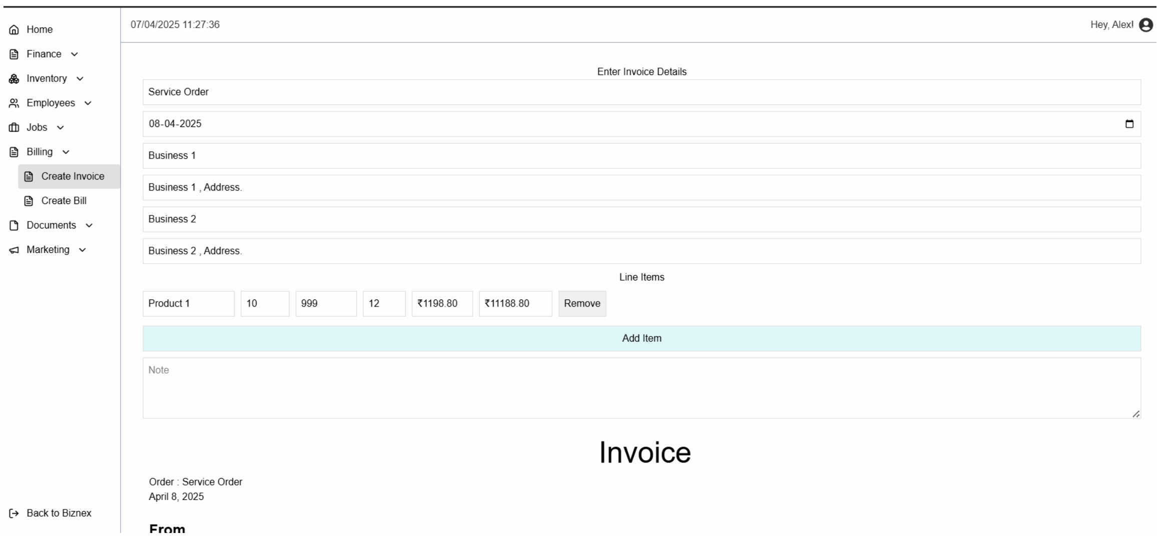
Applications

Name	Email	Resume	Actions
Alice	alice@example.com	View Resume	Reject
Bob	bob@example.com	View Resume	Reject

Marketing Manager
Posted: 2023-10-25
Company: Marketing Solutions Group
Location: New York
Type: Full-time
Work Type: Hybrid
Salary per Month : $90000
Seeking a creative marketing manager to lead our team......
Qualifications: Bachelor's degree in Marketing, 5+ years of experience

Add Job Listing
Job Title
Company Name
Location
Select Type
Select Work Type
Salary per Month
Description
Qualification 1
Qualification 2

Figure B.11: Job Listing



07/04/2025 11:27:36   Hey, Alex!

Enter Invoice Details
Service Order
08-04-2025
Business 1
Business 1 , Address.
Business 2
Business 2 , Address.

Line Items

Product 1	10	999	12	₹1198.80	₹11188.80

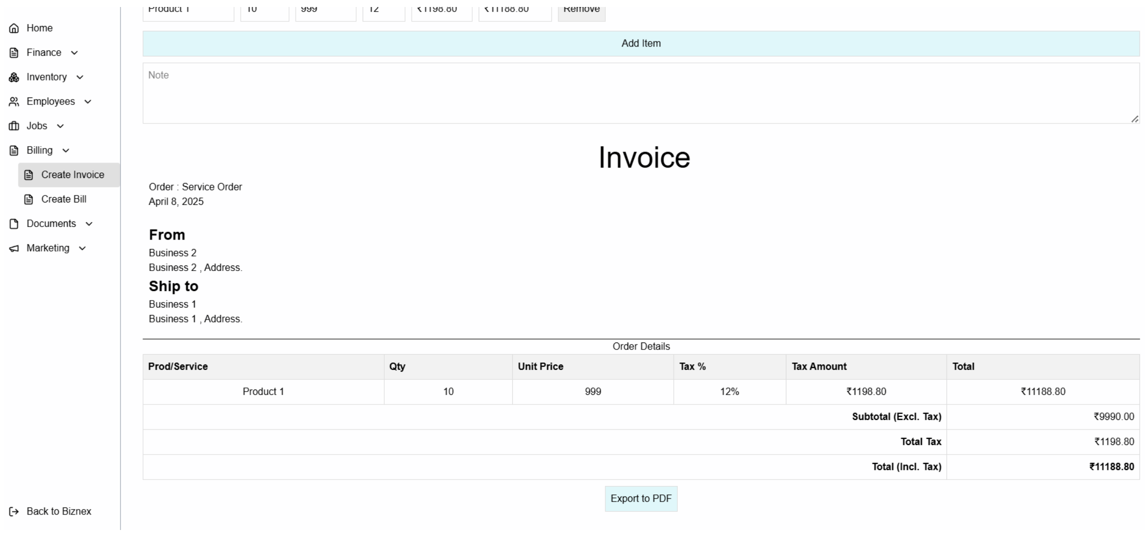
Note

Invoice

Order : Service Order
April 8, 2025

From

Figure B.12: Invoice Creation 1



Product 1	10	999	12	₹1198.80	₹11188.80

Note

Invoice

Order : Service Order
April 8, 2025

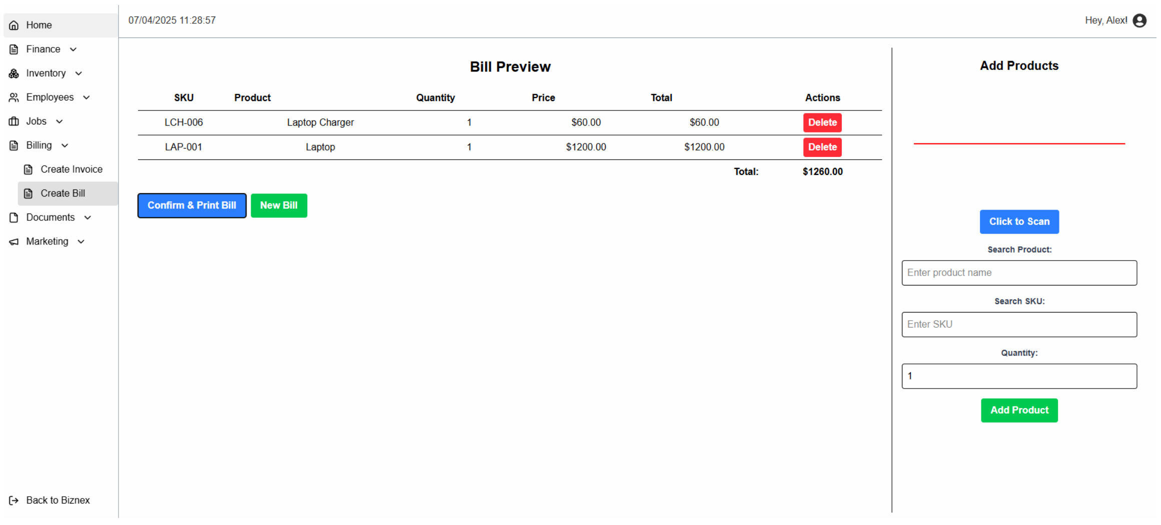
From
Business 2
Business 2 , Address.

Ship to
Business 1
Business 1 , Address.

Order Details

Prod/Service	Qty	Unit Price	Tax %	Tax Amount	Total
Product 1	10	999	12%	₹1198.80	₹11188.80
				Subtotal (Excl. Tax)	₹9990.00
				Total Tax	₹1198.80
				Total (Incl. Tax)	**₹11188.80**

Figure B.13: Invoice Creation 2



07/04/2025 11:28:57

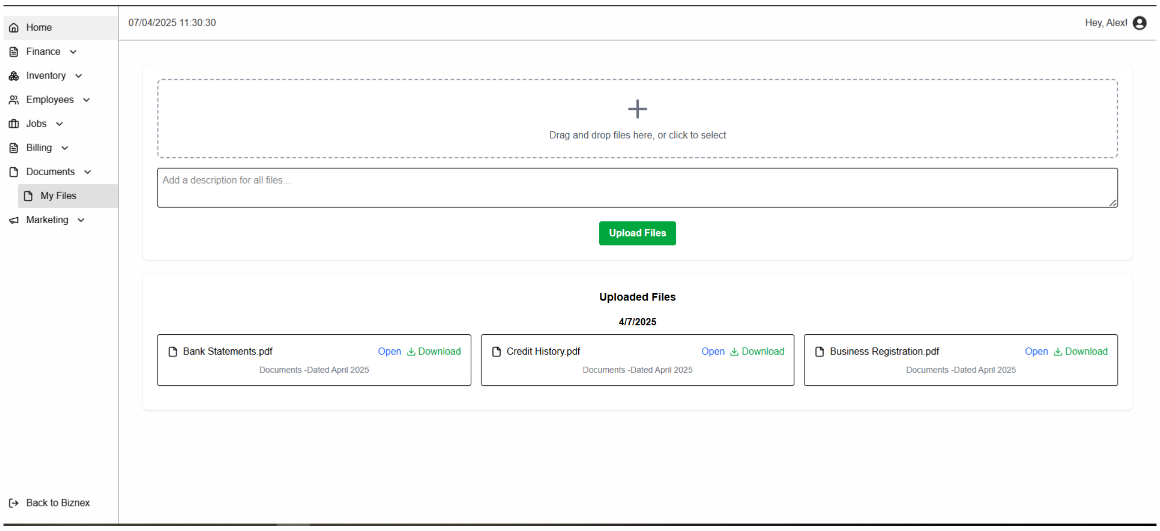
Bill Preview

SKU	Product	Quantity	Price	Total	Actions
LCH-006	Laptop Charger	1	$60.00	$60.00	Delete
LAP-001	Laptop	1	$1200.00	$1200.00	Delete
				Total:	**$1260.00**

Add Products

Search Product: Enter product name
Search SKU: Enter SKU
Quantity: 1

Figure B.14: Bill Creation



07/04/2025 11:30:30

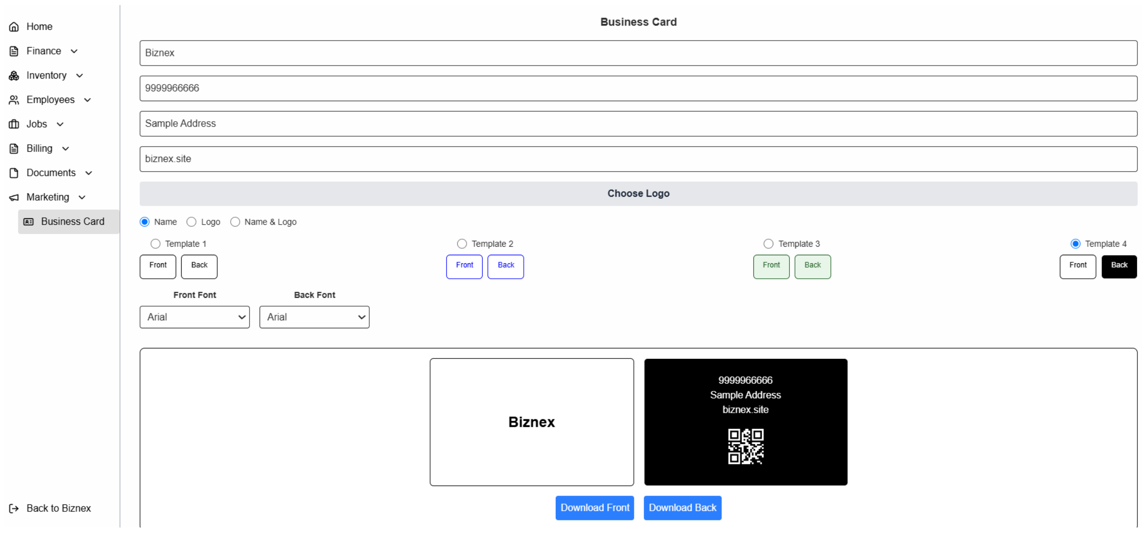
Hey, Alex!

Uploaded Files

4/7/2025

Bank Statements.pdf — Open Download	Credit History.pdf — Open Download	Business Registration.pdf — Open Download
Documents -Dated April 2025	Documents -Dated April 2025	Documents -Dated April 2025

Figure B.15: File Storage



Business Card

Biznex

9999966666

Sample Address

biznex.site

Choose Logo

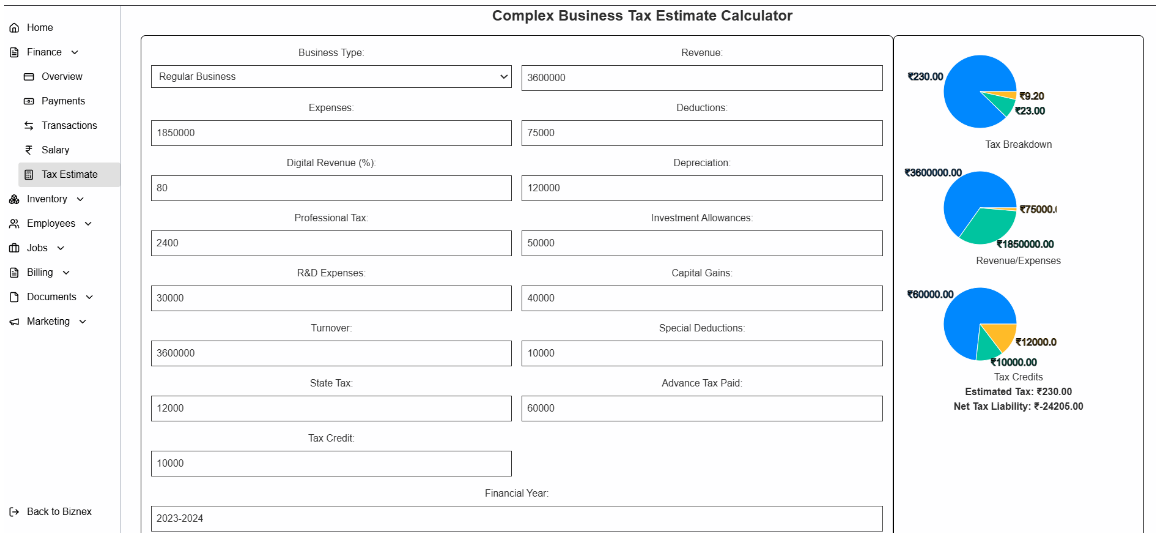
[✓ Name] Logo Name & Logo

Template 1 Template 2 Template 3 [✓ Template 4]

Front Font: Arial

Back Font: Arial

Figure B.16: Business Card



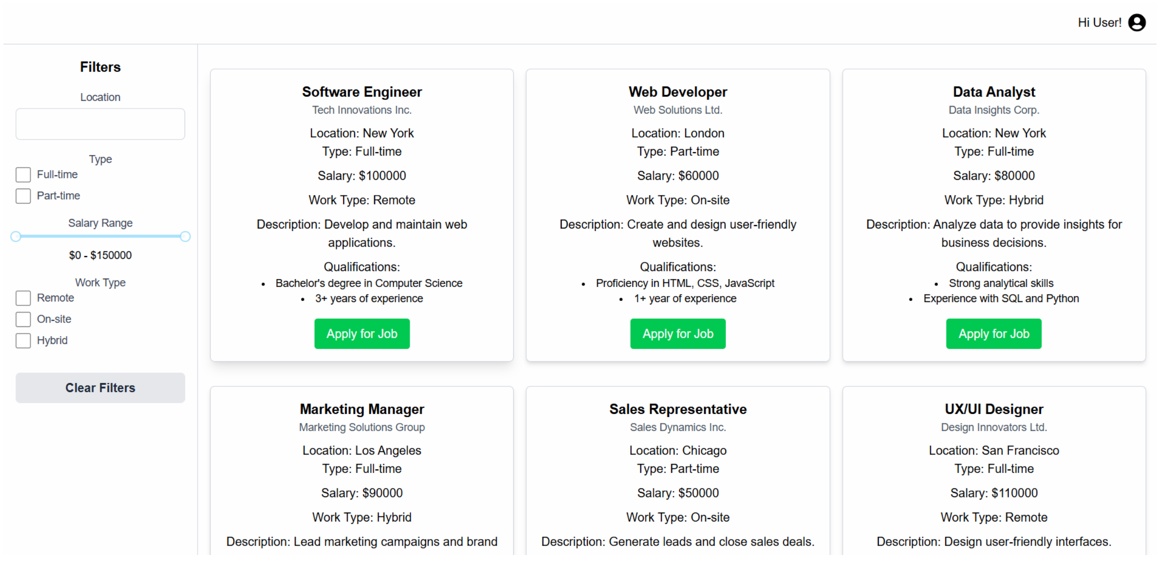
Complex Business Tax Estimate Calculator

Business Type: Regular Business	Revenue: 3600000
Expenses: 1850000	Deductions: 75000
Digital Revenue (%): 80	Depreciation: 120000
Professional Tax: 2400	Investment Allowances: 50000
R&D Expenses: 30000	Capital Gains: 40000
Turnover: 3600000	Special Deductions: 10000
State Tax: 12000	Advance Tax Paid: 60000
Tax Credit: 10000	

Financial Year: 2023-2024

Estimated Tax: ₹230.00
Net Tax Liability: ₹-24205.00

Figure B.17: Tax Estimation



Software Engineer
Tech Innovations Inc.
Location: New York
Type: Full-time
Salary: $100000
Work Type: Remote
Description: Develop and maintain web applications.
Qualifications:
- Bachelor's degree in Computer Science
- 3+ years of experience

Apply for Job

Web Developer
Web Solutions Ltd.
Location: London
Type: Part-time
Salary: $60000
Work Type: On-site
Description: Create and design user-friendly websites.
Qualifications:
- Proficiency in HTML, CSS, JavaScript
- 1+ year of experience

Apply for Job

Data Analyst
Data Insights Corp.
Location: New York
Type: Full-time
Salary: $80000
Work Type: Hybrid
Description: Analyze data to provide insights for business decisions.
Qualifications:
- Strong analytical skills
- Experience with SQL and Python

Apply for Job

Marketing Manager
Marketing Solutions Group
Location: Los Angeles
Type: Full-time
Salary: $90000
Work Type: Hybrid
Description: Lead marketing campaigns and brand

Sales Representative
Sales Dynamics Inc.
Location: Chicago
Type: Part-time
Salary: $50000
Work Type: On-site
Description: Generate leads and close sales deals.

UX/UI Designer
Design Innovators Ltd.
Location: San Francisco
Type: Full-time
Salary: $110000
Work Type: Remote
Description: Design user-friendly interfaces.

Figure B.18: Job User Dashboard

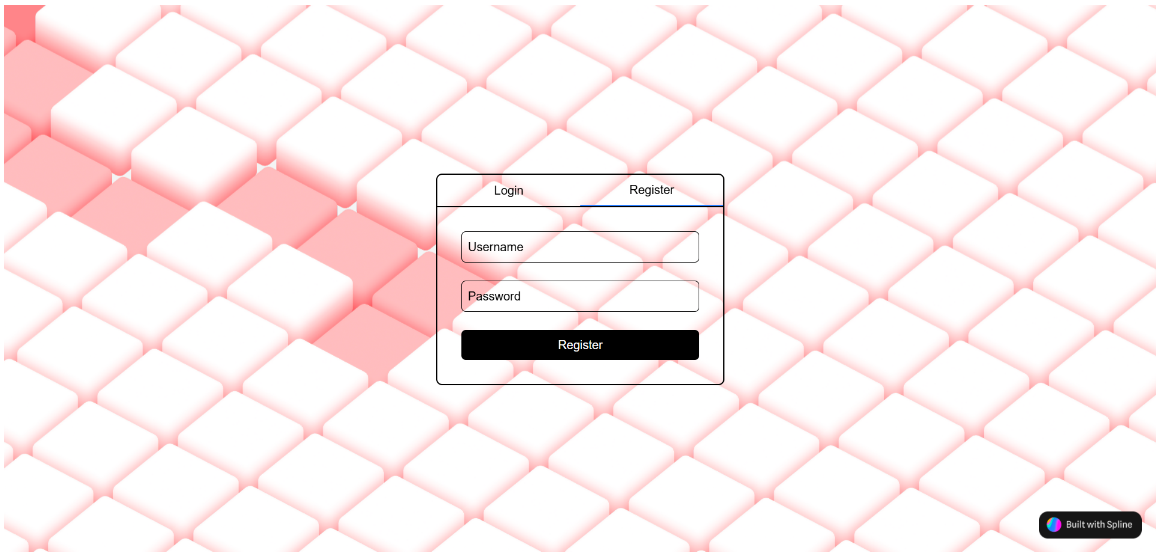


Figure B.19: Subdomain login/register

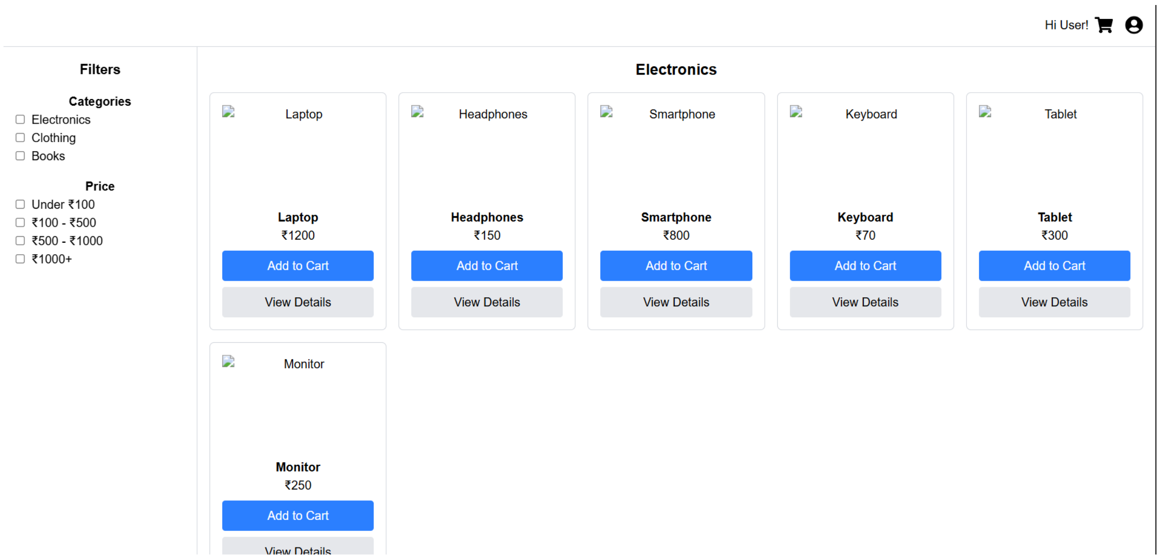


Figure B.20: Subdomain dashboard 1

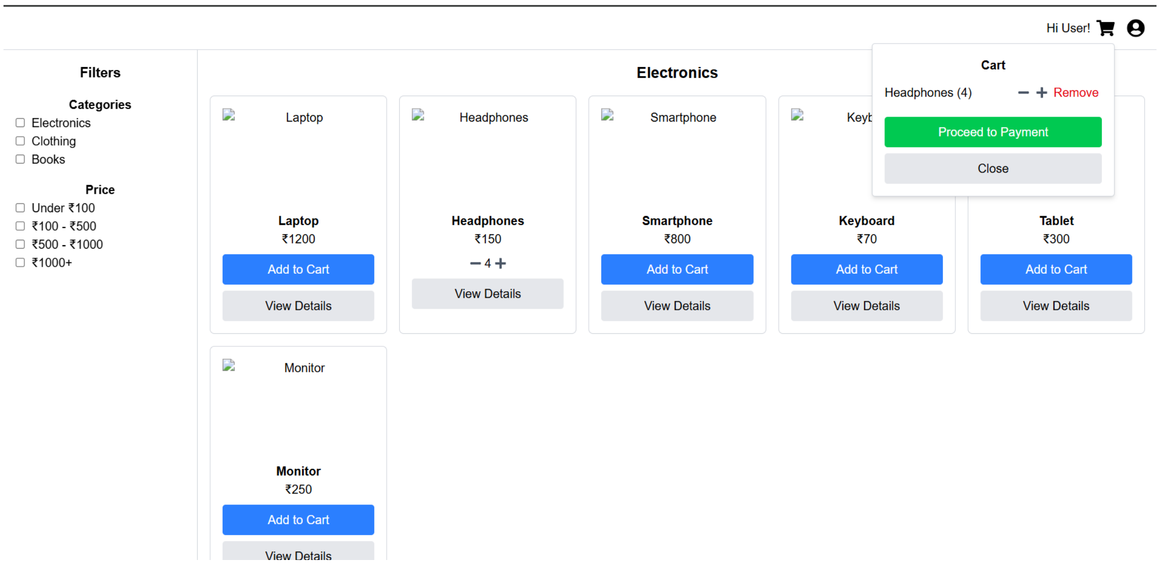


Figure B.21: Subdomain dashboard 2