FINAL YEAR PROJECT REPORT (v2)

ACCOUNTILL — MERN Stack Invoicing Application

A full■stack invoicing and payment tracking web application for freelancers and small businesses

Student Name:	___________________________________
Registration Number:	___________________________________
Department:	___________________________________
Supervisor:	___________________________________
Institution:	___________________________________
Date:	2026-01-20

Repository: Accountill — Client (React) + Server (Express) + MongoDB + Docker

Abstract

This project implements a web■based invoicing system designed for freelancers and small businesses. The system supports client management, invoice creation, invoice status tracking (Unpaid/Partial/Paid), payment recording, and dashboard analytics. The application is built using the MERN stack: MongoDB for persistence, Express/Node.js for the API server, and React for the frontend.

The project includes additional enhancements to improve usability and deployment: optional Google login UI, public invoice payment links using a shareable publicId, a mock payment gateway for demo purposes, and production■style containerization using Docker Compose with Nginx serving the frontend and proxying API calls. A seed script is provided to reset and populate the database with realistic dummy data.

Keywords

MERN, Invoicing, MongoDB, Express, React, Node.js, Docker, Nginx, Payments, Seed Data

Table of Contents

1. Chapter 1: Introduction

2. Chapter 2: Literature Review

3. Chapter 3: System Analysis & Design

4. Chapter 4: Implementation

5. Chapter 5: Testing & Results

6. Chapter 6: Deployment

7. Chapter 7: User Guide

8. Chapter 8: Conclusion & Future Work

9. References

10. Appendix

Chapter 1: Introduction

1.1 Background

Invoicing is essential for tracking revenue, managing customers, and monitoring cash flow. Many freelancers and small businesses require a simple system to create invoices, send them to customers, and keep accurate records of payments.

1.2 Problem Statement

Manual invoicing methods (paper/spreadsheets) are error■prone and time■consuming. Business owners often struggle to track partial payments, due dates, and outstanding balances.

1.3 Aim and Objectives

• Build a full■stack invoicing web application with user authentication.

• Allow users to create and manage clients.

• Allow users to create invoices with line items and totals.

• Track invoice status and payment history, including partial payments.

• Provide a dashboard with invoice statistics.

• Provide a public payment link feature and a demo/mock payment flow.

• Provide Dockerized deployment and database seeding for demonstrations.

1.4 Scope

The system targets small business invoicing workflows and does not attempt to implement real payment processing, complex accounting rules, or tax compliance beyond basic VAT calculation. The mock payment gateway is included for demo/testing only.

1.5 Significance

The project demonstrates practical use of modern full■stack web engineering patterns: REST APIs, authentication via JWT, database modeling with Mongoose, component■based UI development, and containerized deployment.

Chapter 2: Literature Review

2.1 Web Application Architecture

Modern web applications commonly use a client/server architecture. The frontend renders UI and communicates with backend services via APIs. The backend encapsulates business logic and persists data.

2.2 MERN Stack

• MongoDB: Document database suitable for flexible invoice data and nested records.

• Express: Minimal HTTP framework for Node.js APIs.

• React: Component■based UI library for interactive user experiences.

• Node.js: JavaScript runtime for backend services.

2.3 Authentication

JSON Web Tokens (JWT) are widely used for stateless authentication. After login, the client stores a token and includes it in API requests.

2.4 Containerization

Docker containers provide a reproducible runtime environment. Docker Compose orchestrates multi■container setups such as frontend, backend, and database.

Chapter 3: System Analysis & Design

3.1 Functional Requirements

• User registration and login; optional Google login UI.

• Create/update/delete clients and profiles.

• Create/update/delete invoices.

• Record payments and automatically update invoice status.

• Generate public payment links and accept mock payments.

• View dashboards and invoice details.

3.2 Non■Functional Requirements

• Usability: clean UI suitable for business workflows.

• Reliability: persistent storage in MongoDB.

• Portability: Dockerized deployment for demo/production.

• Maintainability: modular client components and server controllers.

3.3 High■Level Architecture

+ React client served by Nginx in Docker.
+ Nginx proxies /api/* to the Express server.
+ Express server uses Mongoose to read/write MongoDB.

3.4 Data Design (Core Collections)

Collection	Purpose	Key Fields
Users	Authentication and identity	name, email (unique), password (hashed), resetToken
Profiles	Business details used on invoices	businessName, contactAddress, email, userId[]
Clients	Customer directory	name, email, phone, address, userId[]
Invoices	Invoice documents + payment history	invoiceNumber, items[], total, status, paymentRecords[], publicI

3.5 Public Payment Link Design

Each invoice can be assigned a shareable publicId. A public route (/pay/<publicId>) allows a customer to view the invoice summary and submit a mock payment. The mock payment is stored in the invoice payment history.

Chapter 4: Implementation

4.1 Backend

• REST endpoints for invoices, clients, profiles, and users.

• Authentication middleware using JWT.

• Invoice payment history and automatic status updates.

• Public invoice endpoints and mock payment submission.

4.2 Frontend

• React Router routes for login, dashboard, invoices, invoice details, and public payment page.

• Redux actions/reducers for API data flows.

• Material■UI components and CSS modules for styling.

• Payment links built from the current site origin (works in Docker/production).

4.3 Dockerized Deployment

The production compose setup runs three services: client (Nginx + built React), server (Express API), and mongo (MongoDB with persistent volume).

Client: http://localhost
API: http://localhost/api/
Server (host mapped): http://localhost:5001

4.4 Database Seeding

A seed script populates Users/Profiles/Clients/Invoices with payment history and supports clearing/resetting the database.

```
docker compose -f docker-compose.prod.yml exec server npm run db:reset
```

Chapter 5: Testing & Results

5.1 Test Approach

• Manual functional testing of login, client creation, invoice creation, payment recording.

• Validation of public payment link route and mock payment updates.

• Deployment validation using Docker Compose.

5.2 Results Summary

• Invoices can be created, listed, and viewed in detail.

• Payments update totals and invoice status (Unpaid/Partial/Paid).

• Public payment links work without authentication and record mock payments.

• Docker stack builds and runs locally.

Chapter 6: Deployment

6.1 Environment Variables

Component	Variable	Description
Server	DB_URL	MongoDB connection string
Server	PORT	Server port inside container (typically 5000)
Server	SECRET	JWT secret
Client	REACT_APP_API	API base URL (defaults to /api via Nginx proxy)
Client	REACT_APP_GOOGLE_CLIENT_ID	Optional, shows Google login if set

6.2 Docker Commands

```
docker compose -f docker-compose.prod.yml build docker compose -f
docker-compose.prod.yml up docker compose -f docker-compose.prod.yml exec server
npm run db:reset
```

Chapter 7: User Guide

7.1 Login

• Create an account or login using email/password.

• Google login UI is optional and appears only when configured.

7.2 Create Clients

• Navigate to Clients and add customer details (name, email, phone, address).

7.3 Create Invoices

• Select a client, add items (name, unit price, quantity, discount), and set due date.

• Review totals and VAT calculations.

7.4 Record Payments

• Open an invoice and add a payment record.

• Status automatically updates based on payments received.

7.5 Public Payment Link

• Open an invoice and click “Copy payment link”.

• Share the link with a customer; they can submit a demo payment.

Chapter 8: Conclusion & Future Work

8.1 Conclusion

Accountill provides a practical invoicing solution built on the MERN stack. The application supports core invoicing workflows and provides a modern foundation for further improvements. Containerization and seeding make it easy to demonstrate and deploy.

8.2 Future Work

• Integrate a real payment gateway (Stripe/PayPal) with webhooks.

• Add roles/permissions for teams and multiple staff accounts.

• Improve PDF rendering reliability in containers.

• Add automated test suites (unit + integration + e2e).

• Add more advanced reporting and export formats.

References

• MongoDB Documentation — https://www.mongodb.com/docs/

• Express Documentation — https://expressjs.com/

• React Documentation — https://react.dev/

• Docker Documentation — https://docs.docker.com/

Appendix

Project Structure

```
root/ client/ (React frontend) server/ (Express backend) docker-compose.prod.yml
```