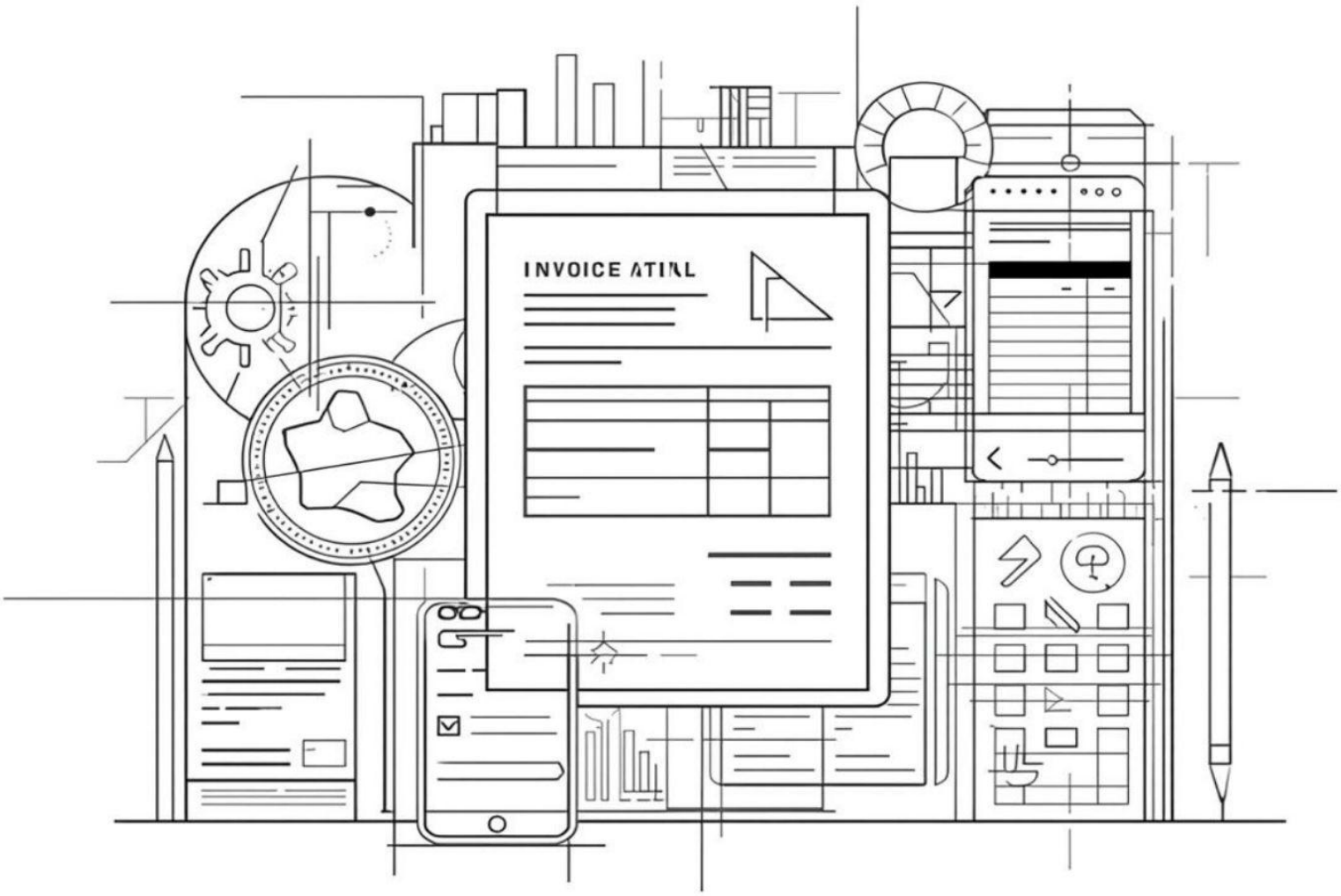

Invoice Management System

Bhagyesh Patil

DECEMBER 2025

Table of Contents

Acknowledgement

It is my privilege to express our sincerest regards to my “Dr. Deepali Patil”, for her valuable inputs, able guidance, encouragement, whole-hearted cooperation and constructive criticism throughout the duration of our project.
We deeply express our sincere thanks to our Head of Department Dr. Sonali Patil for encouraging and allowing us to present the project on the topic “Invoice Management System” at our department premises.
We take this opportunity to thank all our faculties who directly or indirectly helped our project. We pay our respect and love to our parents and all other family members and friends for their love and encouragement throughout our career. Last but not the least we express my thanks to our friends for their cooperation and support.

- Bhagyesh Patil

Abstract

The project titled “Invoice Management System” is a web-based application designed to digitalize and streamline the process of creating, managing, and tracking invoices for small and medium-scale businesses. Traditional manual methods such as paper-based billing or scattered spreadsheet records are time-consuming, error-prone, and difficult to maintain as the number of customers and transactions grows. This system provides a centralized platform where users can manage customer details, define products or services, generate tax-inclusive invoices, and monitor payments and outstanding dues in an organized manner.

The application supports essential functionalities such as customer management, product/service catalog management, automated calculation of totals, taxes, and discounts, and generation of summary reports over customizable time periods. By offering a user-friendly interface and structured data storage, the system reduces human errors, improves accuracy, and increases overall efficiency of the billing process. In addition, it lays the groundwork for future enhancements such as integration with payment gateways, inventory and accounting systems, and advanced analytical dashboards. This project demonstrates how modern web technologies can be used to address common business challenges in billing and financial record-keeping.

Keywords: Invoice Management System, Web Application, Billing Automation, Financial Records, Small and Medium Enterprises (SMEs), Digital Invoicing

1. INTRODUCTION

1.1. Overview

In the modern business environment, maintaining accurate financial records and invoices is critical to ensuring smooth operations, timely payments, and legal compliance. Many small and medium-sized enterprises (SMEs) still rely on manual or semi-digital methods such as handwritten bills, Microsoft Excel sheets, or basic word-processing templates. These approaches often lead to issues such as data duplication, inconsistency, lack of centralized access, difficulty in searching old records, and challenges in generating meaningful reports.

The Invoice Management System addresses these challenges by providing a web-based solution that centralizes all invoice-related activities. The system allows users to store customer information, configure products or services, generate invoices with automatic tax and total calculations, update payment status, and analyze sales over specific periods. The user interacts through a graphical interface (for example, a sidebar-based navigation with modules like Dashboard, Invoices, Customers, and Reports), while the backend handles business logic and database operations. Overall, the system aims to make invoicing more reliable, faster, and easier to manage, even for users with minimal technical background.

1.2. Motivation

The motivation behind developing this system arises from the practical difficulties observed in many small businesses and freelance setups:

1. Manual Workload and Errors: Preparing invoices manually or copying templates repeatedly increases the chances of mistakes in calculations, dates, and customer details. Correcting these errors consumes additional time and may create confusion with clients.

2. Poor Record Management: Paper invoices and isolated digital files are difficult to organize and retrieve, especially when the number of transactions grows. Searching for historical data, such as all invoices for a particular customer or a specific month, becomes time-consuming.

3. Lack of Insights: Without a centralized system, generating summary reports—such as monthly sales, outstanding payments, and tax summaries—is tedious. Decision-making becomes harder due to the absence of clear and timely information.

4. Professionalism and Customer Trust: Well-formatted, consistent invoices and timely reminders reflect professionalism, which positively impacts a company's image and customer satisfaction.

This project is motivated by the need to provide an affordable, easy-to-use tool that can help such businesses automate their billing activities, reduce errors, gain better visibility of their finances, and ultimately save time and cost.

1.3. Problem Statement

Despite the availability of various tools, many small and medium-sized businesses still face the following core problems in their invoicing process:

1. Inefficient and Error-Prone Invoice Creation:
   Businesses often use manual calculation methods or basic templates that do not automatically compute totals, taxes, and discounts. This leads to calculation errors and inconsistencies in invoices.

2. Decentralized and Disorganized Data:
   Customer details, product information, and past invoices are frequently stored in different files or physical locations, making it difficult to track and maintain accurate records.

3. Difficulty in Tracking Payments and Dues:
   Without a centralized system, monitoring which invoices are paid, partially paid, or overdue is challenging. This may lead to delays in follow-ups and cash flow issues.

4. Lack of Quick and Accurate Reports:
   Generating summaries such as total sales per period, customer-wise sales, and tax reports is cumbersome, which affects business analysis and statutory reporting.

Problem Statement:
There is a need for a centralized, user-friendly Invoice Management System that can automate invoice creation, maintain organized records of customers and transactions, provide real-time tracking of payment status, and generate meaningful reports to support effective financial management in small and medium-scale businesses.

2. LITERATURE SURVEY

A comprehensive literature survey was undertaken to understand the current landscape, technological approaches, and research directions in the domain of invoice management and billing systems. The survey encompasses commercial accounting platforms, spreadsheet-based invoicing practices, modern Software-as-a-Service (SaaS) solutions, and relevant academic and open-source systems. The findings highlight both technological opportunities and functional gaps that informed the design of the proposed system.

2.1 Traditional Accounting and Billing Software

Established accounting systems such as Tally, QuickBooks, SAP Business One, and Zoho Books have been widely adopted by small and medium enterprises (SMEs) for financial accounting, inventory control, taxation, and reporting for over two decades [1]–[4]. These platforms provide tightly integrated invoicing modules linked with ledgers and financial statements, enabling end-to-end accounting workflows.

However, multiple studies and industry surveys indicate that these systems are often overly complex for micro-enterprises, freelancers, and early-stage startups [5]. Their extensive configuration requirements, dependence on accounting expertise, and licensing or subscription costs act as barriers to adoption for small-scale users. Research on SME technology adoption highlights that micro-businesses prefer narrowly scoped tools that address core needs such as invoice creation, tax calculation, and basic reporting without excessive overhead [6].

2.2 Spreadsheet-Based Invoicing Practices

Despite the availability of specialized software, a significant proportion of small businesses continue to rely on spreadsheet-based tools such as Microsoft Excel and Google Sheets for invoicing [7]. These tools are favored due to their low cost, flexibility, and ease of customization.

However, extensive research has documented inherent limitations of spreadsheets in financial workflows. Panko's empirical studies demonstrate that spreadsheet errors are pervasive and frequently undetected, even in professional environments [8]. Industry reports further identify issues such as lack of enforced data validation, version fragmentation, limited multi-user collaboration, and absence of reliable audit trails [9]. These limitations significantly reduce the reliability and scalability of spreadsheet-based invoicing systems.

2.3 Web-Based Invoicing Tools and SaaS Platforms

Cloud-based invoicing platforms such as FreshBooks, Zoho Invoice, Wave, and Invoice Ninja have gained popularity by offering cross-device access, automated reminders, hosted databases, and payment gateway integration [10]–[13]. These platforms leverage the SaaS model to reduce local setup complexity.

However, academic and practitioner analyses highlight several drawbacks for micro-enterprises. Recurring subscription fees, feature overload, limited customization, and strict dependence on continuous internet

connectivity restrict usability in low-resource or offline-prone environments [14]. Furthermore, concerns regarding data privacy and vendor lock-in have been raised in studies examining third-party financial data hosting [15].

2.4 Academic and Open-Source Invoice/Billing Systems

Numerous academic projects and open-source billing systems adopt web-based architectures with RESTful backends and relational databases [16], [17]. These systems demonstrate improvements in operational efficiency, accuracy, and transparency when compared to manual methods.

Research emphasizes the importance of data normalization, secure authentication mechanisms, and usability in financial software design [18]. However, many academic prototypes lack modern user interface design, responsive layouts, offline capabilities, and seamless PDF generation, limiting their applicability in real-world business contexts [19].

2.5 Modern Client-Side Technologies in Financial Systems

Recent advancements in front-end technologies such as React, TypeScript, Vite, and Tailwind CSS have enabled the development of high-performance, client-centric web applications [20][22]. Literature in modern web engineering suggests that client-side rendering improves responsiveness, while browser storage technologies such as IndexedDB enable offline-first capabilities [23].

In-browser PDF generation libraries and visualization frameworks further allow document automation and financial analytics without heavy backend dependencies [24]. These advancements support the feasibility of lightweight, privacy-preserving invoice management systems tailored for micro-enterprises.

2.6 Gaps Identified from the Literature

Based on the reviewed literature, the following gaps are consistently observed:

1. Over-complexity of traditional accounting systems
2. High error rates and lack of validation in spreadsheets
3. Cost and connectivity dependence of SaaS platforms
4. Poor user experience in academic prototypes
5. Limited support for local taxation rules
6. Insufficient analytics in lightweight invoicing tools

These gaps establish the need for a simple, customizable, and user-centric invoice management system.

2.7 Summary of Literature Review

The literature supports the development of a client-side invoice management system that combines structured data handling, modern user interface design, and analytical capabilities without the overhead of enterprise accounting platforms. The proposed system aims to bridge the identified gaps by delivering essential invoicing functionality with a low learning curve and extensible architecture.

3. REQUIREMENT SPECIFICATIONS

3.1 Hardware Requirements

The hardware requirements for the proposed system are categorized based on the scale of usage. These specifications ensure smooth development, deployment, and operation of the application under different workloads.

3.1.1 Minimum Hardware Requirements (Development / Small Business Use)

For development purposes and small-scale business usage, the system requires a minimum of a dual-core processor operating at 2.0 GHz or higher. A minimum of 4 GB RAM is necessary to support browser execution, backend services, and development tools simultaneously. At least 20 GB of free disk space is required to store application files, dependencies, and logs. A display resolution of 1366×768 or higher is recommended to ensure proper rendering of the user interface. Since the system is web-based, a stable internet connection is required for uninterrupted access to backend services and cloud resources.

3.1.2 Recommended Hardware Requirements (Server / Multi-User Environment)

For server deployment or multi-user environments, a quad-core processor with a clock speed of 2.4 GHz or higher is recommended to handle concurrent user requests efficiently. The system should be equipped with 8–16 GB RAM to support scalable backend operations and database interactions. A minimum of 100 GB storage is advised, with Solid State Drives (SSD) preferred for improved I/O performance. A high-speed local area network (LAN) or reliable broadband internet connection is essential to ensure low-latency communication between clients and servers.

3.2 Software Requirements

The successful development, deployment, and maintenance of the system require a well-defined set of software requirements across both client-side and server-side environments. These requirements ensure compatibility, optimal performance, security, and accessibility. Since the application is web-based, special emphasis is placed on operating system compatibility, browser support, and runtime environments.

3.2.1 Client-Side Software Requirements

The client-side environment refers to all software components that run on the user’s device, primarily within a web browser. It determines how users interact with the system and ensures responsiveness and accessibility.

Operating System Compatibility

The system supports Microsoft Windows 10 and Windows 11, which are widely used desktop operating systems and offer strong browser and file system support. macOS (Big Sur and above) is supported due to its stability, security, and optimized browser performance through Safari. Various Linux distributions such as Ubuntu, Fedora, Debian, and Arch Linux are also supported, enabling accessibility for developers, academic users, and enterprise environments. Linux-based browsers provide full support for modern web standards.

Supported Web Browsers

The application supports modern web browsers including Google Chrome, Mozilla Firefox, Microsoft Edge, Opera, and Safari. These browsers offer full compatibility with contemporary web technologies such as ES6+ JavaScript, HTML5 Canvas, CSS3 Flexbox and Grid layouts, Web Storage APIs, and Service Workers. This ensures consistent user experience across different platforms.

Additional Client-Side Tools

A PDF reader, such as Adobe Acrobat Reader or a built-in browser viewer, is required to open exported invoices, reports, and financial documents. Spreadsheet viewers such as Microsoft Excel or Google Sheets are necessary for accessing downloaded CSV or Excel-based financial reports. A stable internet connection with a minimum bandwidth of 2 Mbps is recommended to ensure smooth data synchronization. A minimum screen resolution of 1366×768 is required for proper UI rendering, while Full HD resolution (1920×1080) is recommended for dashboards and analytics views.

3.2.2 Runtime Environments

Runtime environments provide the execution layer for backend services and business logic.

Node.js Runtime Environment

Node.js is a JavaScript runtime built on Chrome's V8 engine and is widely used for building scalable backend applications. Its event-driven, non-blocking architecture allows it to handle multiple concurrent requests efficiently, making it suitable for high-performance web applications.

Key Features:
Node.js uses asynchronous, event-driven programming to prevent blocking operations, allowing efficient handling of I/O-intensive tasks. Its non-blocking architecture enables the system to manage thousands of concurrent requests. The Node Package Manager (NPM) provides access to a large ecosystem of reusable libraries, accelerating development.

Use Cases:
Node.js is used for developing RESTful APIs, handling real-time data processing, and supporting microservice-based architectures due to its lightweight and scalable nature.

.NET Runtime Environment

The .NET runtime is a comprehensive development platform provided by Microsoft that supports multiple programming languages, including C#, VB.NET, and F#. It provides a structured and high-performance environment for backend application development.

Key Features:
.NET follows the Model-View-Controller (MVC) architectural pattern, which promotes separation of concerns and improves maintainability. It includes built-in security middleware for authentication, authorization, and protection against common vulnerabilities such as SQL injection and cross-site scripting. The Kestrel web server offers high-performance, cross-platform HTTP request handling.

Use Cases:
.NET is commonly used for enterprise-grade applications that require robustness, scalability, and security. It is also widely adopted in financial and banking systems where data integrity and strong security guarantees are essential.

3.2.3 Database Technology: Supabase

Supabase is used as the primary database and backend platform for the Invoice system. It is an open-source Backend-as-a-Service (BaaS) platform built on PostgreSQL, providing a reliable and scalable backend infrastructure for modern web applications.

3.2.4 Key Features of Supabase

One of the core features of Supabase is its PostgreSQL-based relational database, which ensures ACID-compliant transactions and strong data consistency. This is particularly important for financial systems where invoice records, client data, and payment details must remain accurate and reliable.

Supabase provides Row Level Security (RLS), a database-level access control mechanism that restricts data access based on authenticated user identity. In Invoice, RLS ensures complete data isolation between users, preventing unauthorized access to invoices, clients, and inventory records.

Another significant feature is the built-in authentication and authorization system. Supabase supports secure email and password-based authentication and manages user sessions using JSON Web Tokens (JWT). This simplifies identity management while maintaining high security standards.

Supabase automatically generates RESTful APIs and real-time endpoints for database tables. These APIs allow seamless communication between the frontend application and the database without the need for custom backend logic. Real-time subscriptions enable instant data updates, improving application responsiveness.

Supabase follows a serverless architecture that automatically scales backend resources based on application demand. This eliminates the need for manual infrastructure management and ensures consistent performance as the system grows.

3.2.5 Use Cases of Supabase in Invoice

Supabase is used for user authentication and session management, enabling secure login and logout functionality while maintaining session persistence across browser sessions.

It serves as the primary data storage solution for all financial entities, including invoices, clients, and inventory items. The relational structure ensures referential integrity and prevents data inconsistencies.

Supabase real-time capabilities are used to update dashboard metrics and invoice lists instantly whenever changes occur in the database, providing real-time financial insights to users.

Transactional support in Supabase ensures that invoice creation, inventory updates, and client financial aggregates are executed atomically, preventing partial updates and data corruption.

Row Level Security policies are employed to enforce multi-tenant data isolation, ensuring that each user's business data remains private and secure within the shared database environment.

3.2.6 Backend Development Tools

Backend development for the Invoice System utilizes a set of modern development tools that support efficient coding, debugging, version control, and deployment. These tools enhance developer productivity, maintain code quality, and ensure consistency throughout the software development lifecycle.

3.2.7 Integrated Development Environments (IDEs)

Integrated Development Environments (IDEs) are software tools that provide a comprehensive environment for writing, debugging, and testing application code. In the development of Invoice, IDEs such as Visual Studio Code, Visual Studio, and JetBrains IDEs were used to support backend-related development tasks.

Visual Studio Code is a lightweight, open-source IDE widely used for web and backend development. It provides extensive extension support for languages such as JavaScript, TypeScript, Python, and SQL, making it suitable for full-stack development workflows.

Visual Studio is a full-featured IDE primarily used for enterprise-level development, especially with .NET and C#. It offers advanced debugging, profiling, and integrated testing tools.

JetBrains IDEs, such as IntelliJ IDEA and PyCharm, are professional-grade development environments known for intelligent code analysis and strong language support. These IDEs are commonly used for Java- and Python-based backend development.

Key Features:
These IDEs provide features such as syntax highlighting, intelligent code completion, and real-time error detection, which help reduce development errors. Built-in debugging tools allow developers to identify and resolve runtime issues efficiently. Integration with version control systems such as Git enables seamless code management and collaboration.

Use Cases in Invoice:
IDEs were used for writing backend logic, configuring database interactions, managing API calls, and debugging application behaviour during development and testing phases.

3.2.8 Git Version Control System

Git is a distributed version control system used to track changes in source code and manage collaborative development. It allows multiple developers to work on the same codebase simultaneously while maintaining a complete history of changes.

Key Features:
Git tracks changes to files over time and allows developers to revert to previous versions when necessary. It supports branching and merging, enabling parallel development of features without affecting the main codebase. Integration with platforms such as GitHub, GitLab, and Bitbucket provides additional collaboration features, including pull requests, issue tracking, and code reviews.

Use Cases in Invoice:
Git was used to manage the project source code, track feature development, and maintain version history. Branching strategies were employed to develop and test new features independently before merging them into the main codebase.

3.2.9 Docker (Optional)

Docker is a containerization platform that enables applications to be packaged along with their dependencies into lightweight, portable containers. These containers ensure that applications run consistently across different environments.

Key Features:
Docker provides environment isolation, ensuring that application behavior remains consistent regardless of the underlying operating system. It simplifies deployment by packaging dependencies, runtime libraries, and configuration into a single container. Docker also supports scalable and reproducible deployments.

Use Cases in Invoice:
Docker can be used to containerize backend services and database configurations, ensuring consistent development and testing environments. This is particularly useful for deployment and future scalability, although Docker usage in Invoice remains optional.

3.2.10 Summary

The backend development tools used in Invoice collectively support efficient development, collaboration, and deployment. IDEs enhance code quality and debugging efficiency, Git ensures reliable version control and collaboration, and Docker provides optional support for consistent deployment environments. Together, these tools contribute to a robust and maintainable software development workflow.

3.3. Functional Requirements

Functional requirements define the core capabilities and operations that the system must support to fulfill user needs. These requirements describe *what* the system is expected to do and form the basis for system design, implementation, and testing.

3.3.1 User Authentication and Authorization

The system shall provide a secure authentication mechanism that allows users to register and log in using an email address and password. Authentication shall be handled through a secure backend service to ensure confidentiality

and integrity of user credentials. The system shall maintain user sessions across browser refreshes and enforce automatic session expiration upon logout or inactivity. Role-based access control shall be implemented to ensure that users can only access their own financial data.

3.3.2 Invoice Management

The system shall allow users to create, view, update, and delete invoices. While creating an invoice, users shall be able to select a client, add multiple line items, define quantities, unit prices, and applicable taxes. The system shall automatically calculate subtotals, taxes, and total payable amounts. Users shall be able to update invoice status (paid, pending, overdue) and record payment dates. The system shall provide search and filter functionality to retrieve invoices based on client name, invoice ID, date range, or payment status.

3.3.3 Client Management

The system shall maintain a centralized client directory that stores client details such as name, email address, and transaction history. Users shall be able to add, edit, and delete client records. The system shall automatically associate invoices with their respective clients and maintain aggregated metrics such as total invoice count, total billed amount, and outstanding balance. Historical client data shall be preserved for reporting and analysis purposes.

3.3.4 Inventory Management

The system shall provide an inventory module that allows users to manage products or services offered. Each inventory item shall include attributes such as product name, description, unit price, and availability. Inventory items shall be selectable during invoice creation to reduce manual data entry. The system shall update inventory usage in real time as invoices are generated.

3.3.5 Automated Tax Calculation

The system shall automatically calculate applicable taxes based on predefined tax rules. Tax values shall be applied consistently across all invoices, eliminating manual computation errors. The system shall support configurable tax rates to accommodate regional or business-specific tax requirements.

3.3.6 PDF Generation and Export

The system shall generate professional, print-ready PDF documents for invoices. Generated PDFs shall conform to standard A4 page dimensions and include business details, client information, itemized billing, tax breakdowns, and total amounts. Users shall be able to download and share these PDF invoices directly from the application.

3.3.7 Financial Dashboard and Analytics

The system shall provide a real-time dashboard displaying key financial metrics, including total revenue, outstanding payments, overdue invoices, and monthly trends. Graphical visualizations such as charts and summaries shall be provided to improve financial insight and decision-making.

3.3.8 Client Health Score Evaluation

The system shall compute a client reliability score based on payment behavior, invoice volume, and overdue history. Clients shall be categorized into predefined health levels such as good, average, or poor. This feature shall assist users in identifying high-risk clients and improving cash flow management.

3.4. Non-Functional Requirements

Non-functional requirements define the quality attributes and operational constraints of the system. These requirements ensure that the system performs reliably, securely, and efficiently under expected conditions.

3.4.1 Performance Requirements

The system shall ensure fast load times, with the initial application load completing within 1.5 seconds under normal network conditions. Dashboard data and invoice lists shall load within 800 milliseconds. The system shall support multiple concurrent users without significant performance degradation.

3.4.2 Scalability Requirements

The system shall be designed to scale horizontally to support increased numbers of users and transactions. Backend services and database operations shall handle growth in data volume without requiring major architectural changes. The system shall support future enhancements such as payment gateway integration and AI-based analytics.

3.4.3 Reliability and Availability

The system shall maintain high availability with minimal downtime. All database transactions shall adhere to ACID properties to ensure data consistency and reliability. Automated backups shall be maintained to prevent data loss in case of system failure.

3.4.4 Security Requirements

All data transmission between the client and server shall occur over secure HTTPS connections. The system shall enforce authentication and authorization using secure access tokens. Row-Level Security mechanisms shall ensure

that users can only access their own data. Sensitive information such as credentials shall be securely encrypted and never stored in plain text.

3.4.5 Usability Requirements

The system shall provide an intuitive and user-friendly interface with minimal learning curve. The user interface shall follow modern design principles and support responsive layouts across desktops, tablets, and mobile devices. Visual feedback and consistent navigation shall enhance user engagement and productivity.

3.4.6 Maintainability

The system shall be modular and maintainable, allowing developers to easily update or extend functionality. Clear separation of concerns between frontend, backend, and database layers shall be maintained. Code readability and documentation shall be prioritized to facilitate future development.

3.4.7 Compatibility and Portability

The system shall be compatible with major operating systems and modern web browsers. As a web-based application, it shall not require platform-specific installation and shall be accessible from any device with a supported browser and internet connection.

3.4.8 Compliance and Standards

The system shall follow standard web development and database design practices. Financial data handling shall comply with applicable data protection and privacy regulations. Generated invoices shall follow standard accounting and documentation formats to ensure professional and legal compliance.

3.5. System Requirements

System requirements define the complete set of technical, operational, and environmental conditions necessary for the successful deployment and execution of the proposed system. These requirements describe how hardware, software, network, and security components interact to deliver a reliable and scalable web-based financial management platform.

3.5.1 Architectural Requirements

The system shall adopt a web-based, client–server architecture using a Single Page Application (SPA) model. The frontend shall be responsible for user interaction, presentation logic, and client-side validation, while backend

services shall manage authentication, data persistence, and business logic. The architecture shall support a decoupled design, enabling independent scaling and maintenance of frontend and backend components.

The system shall utilize a Backend-as-a-Service (BaaS) model for database management, authentication, and real-time data synchronization. This approach reduces infrastructure overhead and improves development efficiency while ensuring high availability and reliability.

3.5.2 Database Requirements

The system shall use a relational database management system to store and manage structured financial data. The database shall support normalized schema design with clearly defined relationships between entities such as users, clients, invoices, and inventory items. Primary and foreign key constraints shall be enforced to maintain referential integrity.

The database shall support advanced querying, indexing, and transactional consistency. All write operations shall be atomic and follow ACID properties to ensure data accuracy, especially for financial records. Real-time synchronization mechanisms shall be supported to reflect data updates instantly on the client interface.

3.5.3 Network and Connectivity Requirements

The system shall require continuous internet connectivity for real-time interaction between the client and backend services. All client–server communication shall occur over secure HTTPS channels to protect data integrity and confidentiality. The system shall be optimized to operate under standard broadband connections with minimal latency.

The network design shall support concurrent connections from multiple users without degradation in performance. WebSocket or real-time communication channels shall be utilized where necessary to ensure instant data updates across sessions.

3.5.4 Security Requirements

The system shall implement robust security mechanisms to protect sensitive financial and user data. Authentication shall be mandatory for all users, and access to resources shall be governed by authorization policies. The system shall enforce row-level data access controls to prevent unauthorized data exposure.

Data at rest and data in transit shall be protected using industry-standard encryption techniques. Secure session management practices shall be followed, including token-based authentication and automatic session expiration. Regular security updates and patches shall be applied to mitigate vulnerabilities.

3.5.5 Performance and Resource Utilization

The system shall be optimized for efficient resource utilization on both client and server sides. Client-side rendering shall minimize unnecessary network requests and reduce load times. Backend services shall efficiently handle database queries and transactions to support real-time dashboard updates and invoice processing.

The system shall be capable of supporting growth in user base and transaction volume without requiring significant architectural changes. Caching strategies and optimized queries shall be employed to enhance responsiveness.

3.5.6 Deployment and Maintenance Requirements

The system shall support cloud-based deployment for ease of scalability and accessibility. Deployment processes shall be automated where possible to reduce configuration errors and downtime. The system shall support environment-based configurations for development, testing, and production.

Maintenance requirements include regular database backups, monitoring of system health, and logging of critical events. The system shall be designed to allow seamless updates and feature enhancements with minimal disruption to users.

3.5.7 Interoperability and Extensibility

The system shall expose standardized interfaces to enable integration with external services such as payment gateways, email services, and analytics tools. The architecture shall support future extensions, including AI-based insights, automated notifications, and third-party integrations, without major refactoring.

4. SYSTEM DESIGN

4.1. System Architecture

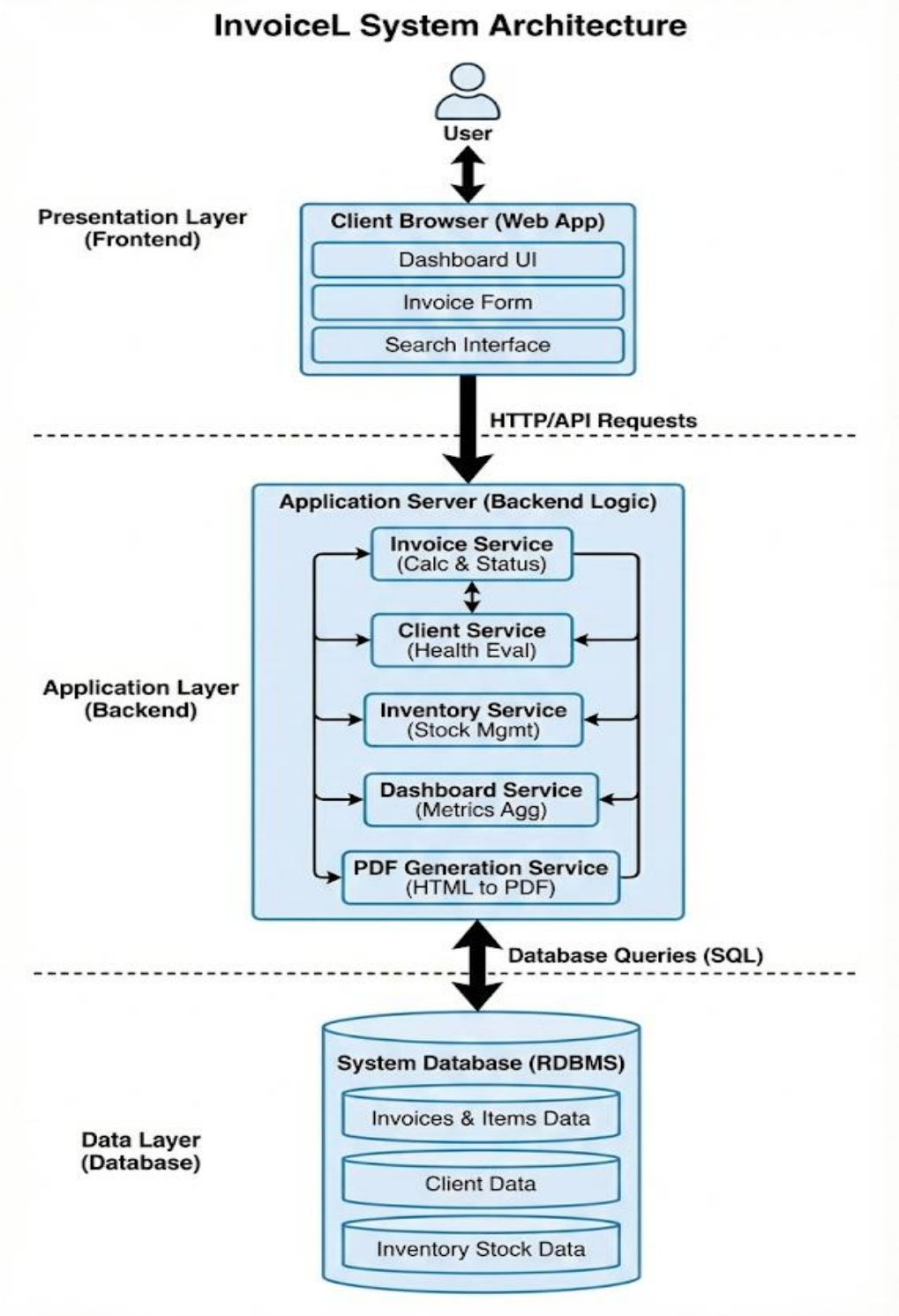

Fig. 4.1. System Architecture

4.2. UML Diagrams

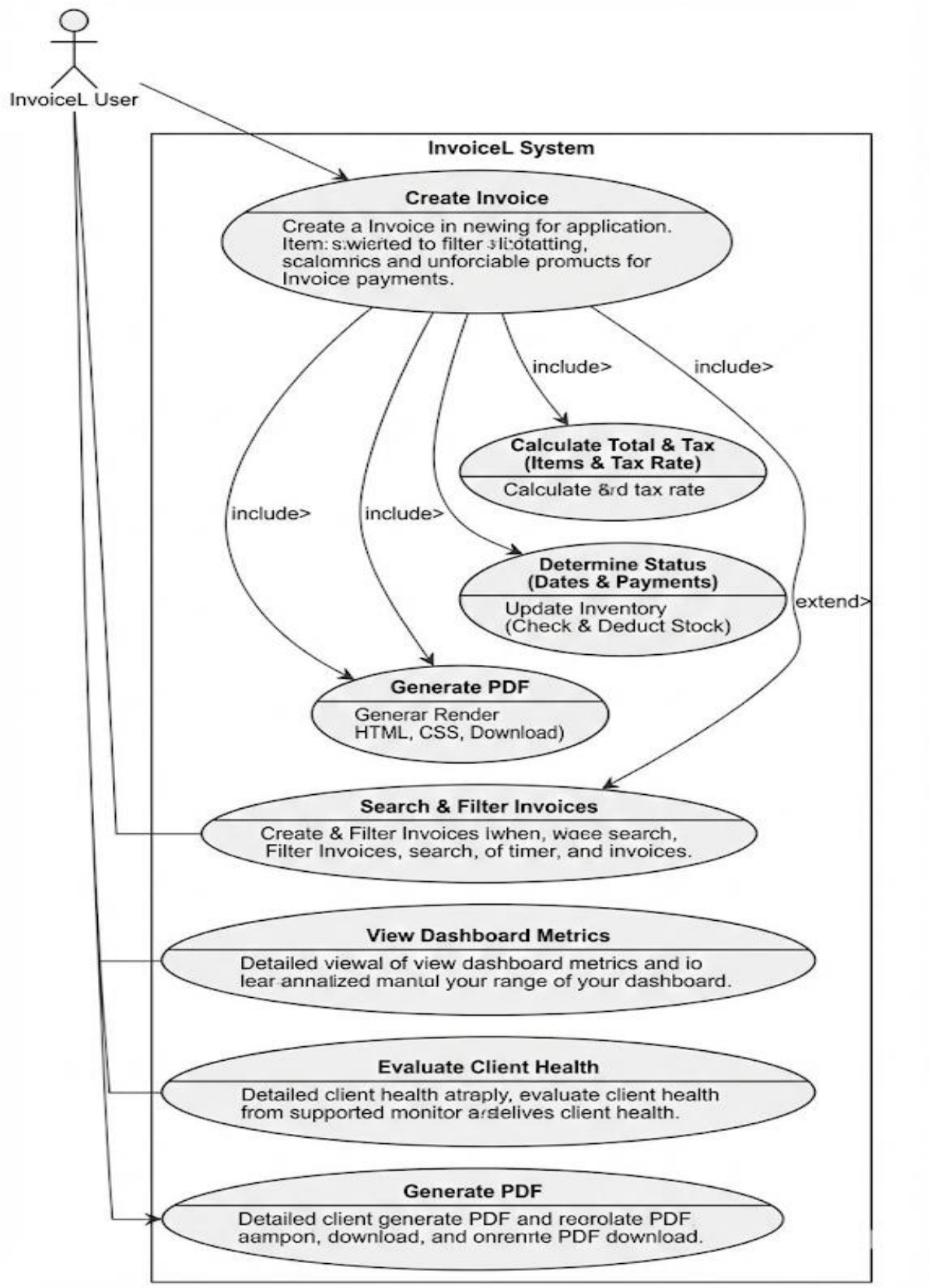

Fig. 4.2. UML Diagrams

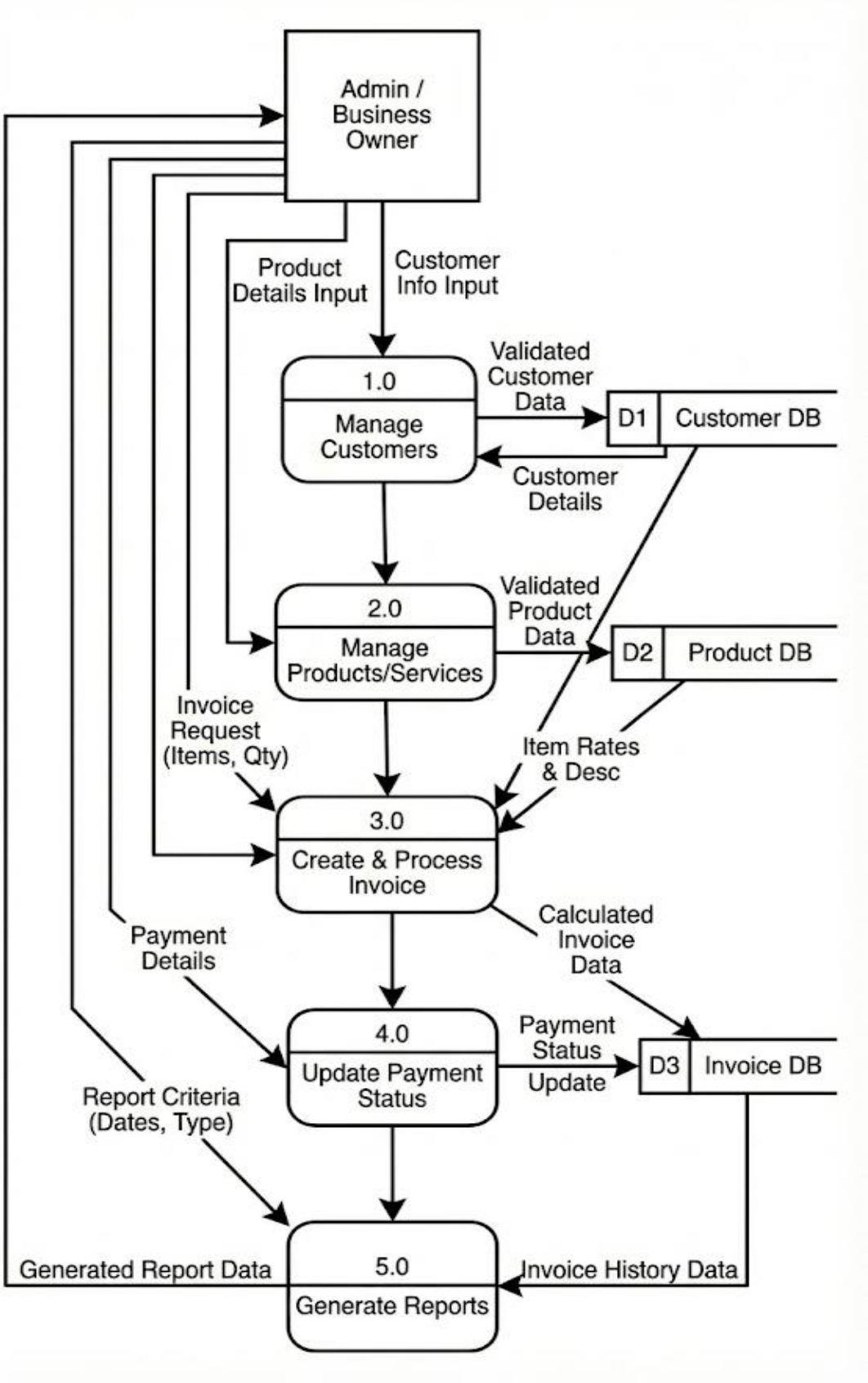

Fig. 4.3. Class Diagram

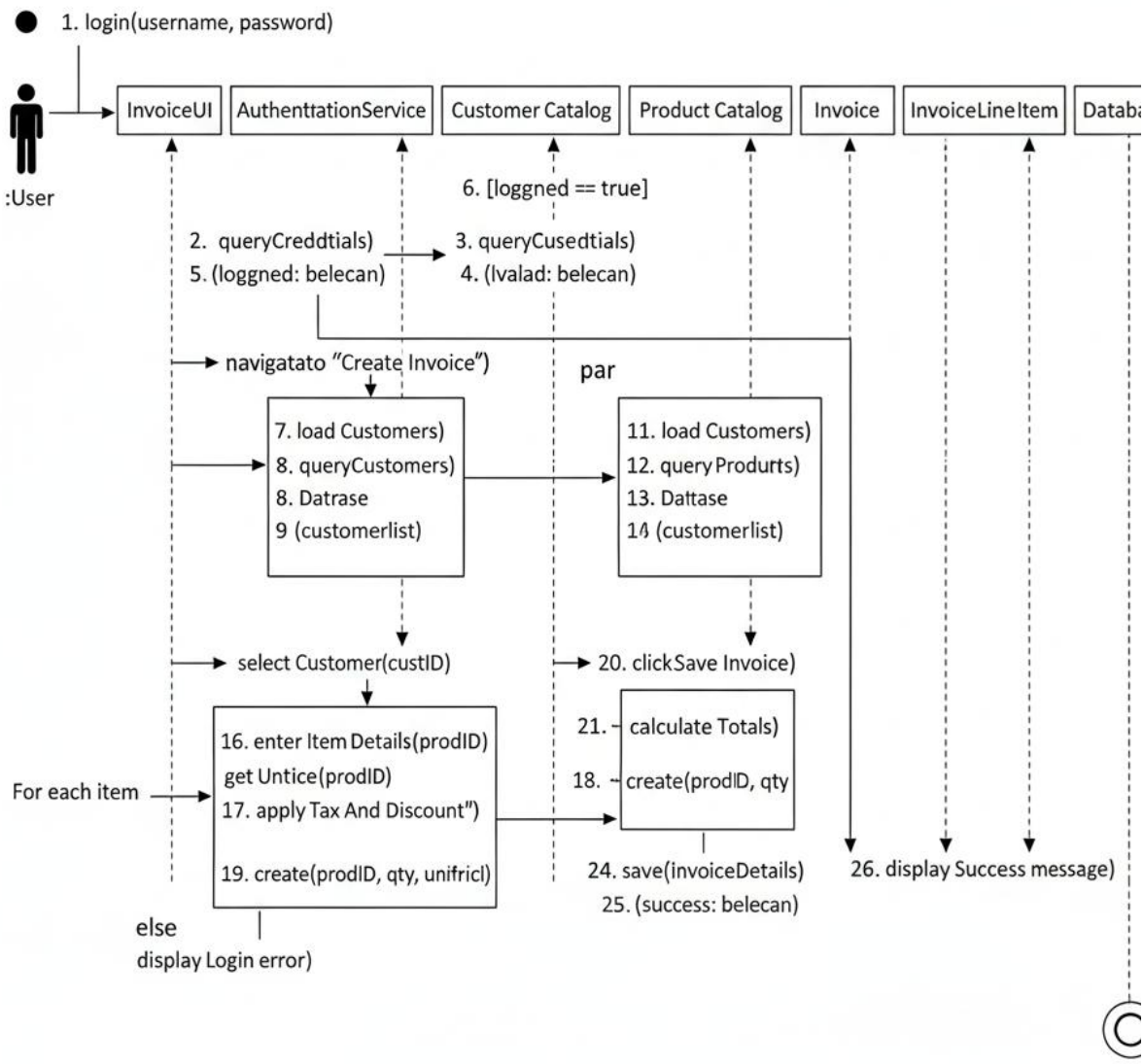

Fig. 4.4. Sequence Diagram

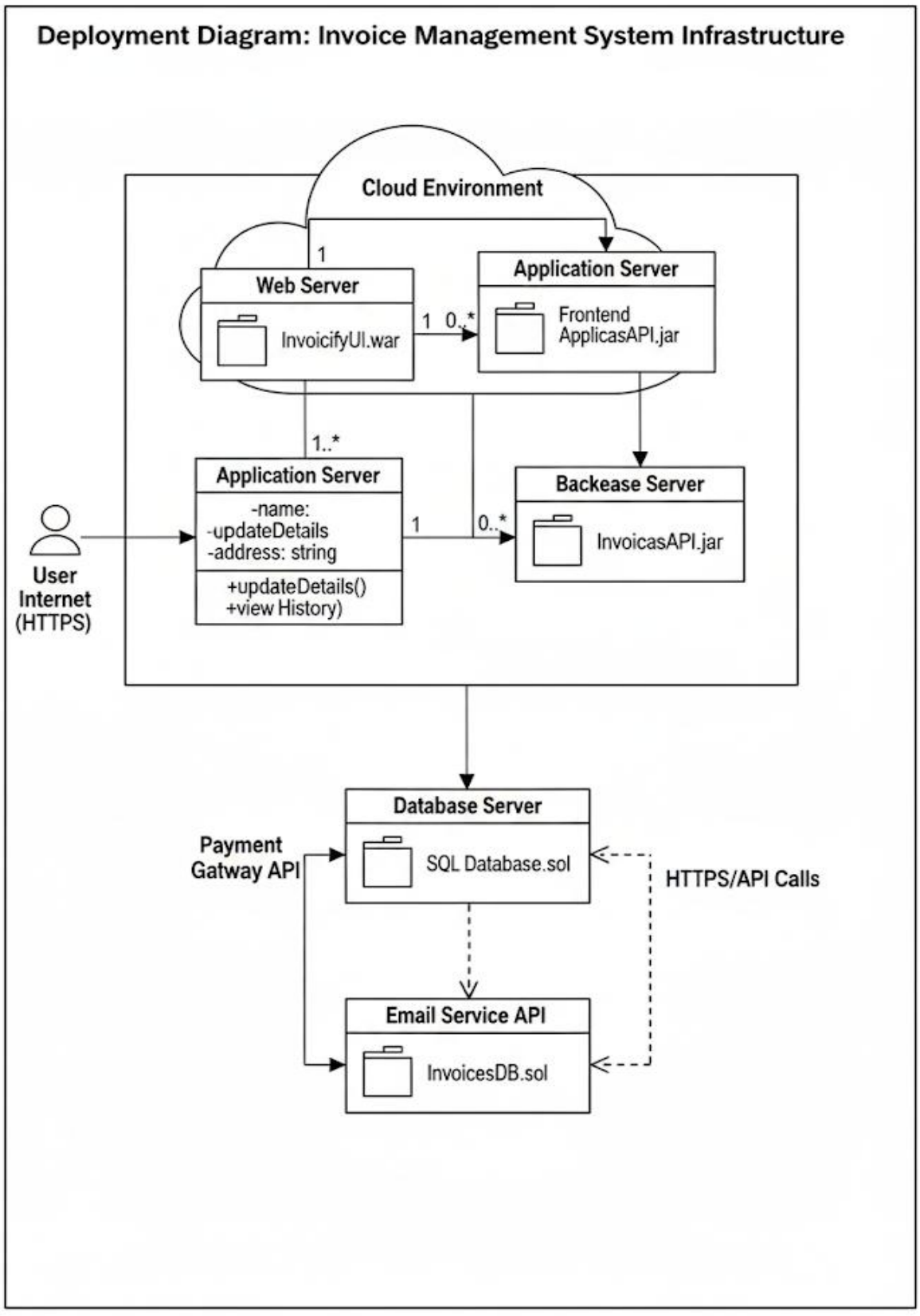

Fig. 4.5. Deployment Diagram

IMPLEMENTATION DETAILS

5.1 Algorithm Details

This section presents the principal algorithms employed in the InvoiceL system. Each algorithm is analyzed in terms of its functional objective, computational logic, and computational complexity. The algorithms are designed to ensure correctness, efficiency, and scalability for small-to-medium datasets typical of SME usage.

5.1.1 Invoice Total and Tax Calculation Algorithm

Objective:
To compute the subtotal, tax amount, and final payable amount for an invoice based on its line items.

Algorithm Description:
The algorithm iterates through each invoice line item, multiplies the quantity by unit price, and accumulates the result to compute the subtotal. A predefined tax percentage is then applied to derive the tax amount, which is added to the subtotal to compute the final total.

Algorithm Steps:

1. Initialize subtotal to zero
2. Iterate through all invoice items
3. Accumulate quantity × unit_price
4. Apply tax percentage
5. Return final total

Time Complexity:

$$O(n)$$

where *n* is the number of line items in the invoice.

Space Complexity:

$$O(1)$$

Only constant auxiliary variables are used.

Justification:
The algorithm scales linearly with the number of invoice items, which is acceptable since invoice sizes are typically small.

5.1.2 Dynamic Invoice Search and Filtering Algorithm

Objective:
To enable real-time filtering of invoices based on client name and invoice status.

Algorithm Description:
The algorithm performs a linear scan of the invoice list and applies string matching for search queries along with conditional checks for status filters. Matching invoices are included in the result set.

Algorithm Steps:

1. Normalize search term
2. Iterate through invoice list
3. Check search and status conditions
4. Append matching invoices to result list

Time Complexity:

$$O(n)$$

where *n* is the total number of invoices.

Space Complexity:

$$O(k)$$

where *k* is the number of filtered invoices returned.

Justification:
Client-side filtering avoids repeated database calls, improving responsiveness for moderate dataset sizes.

5.1.3 Client Health Score Evaluation Algorithm

Objective:

To classify clients based on payment reliability using historical invoice data.

Algorithm Description:

The algorithm calculates the percentage of paid invoices and applies heuristic thresholds to assign a qualitative health category (Good, Average, Poor).

Algorithm Steps:

1. Compute payment success ratio
2. Compare ratio against predefined thresholds
3. Assign corresponding health status

Time Complexity:

$$O(1)$$

Assumes aggregate values are precomputed at query time.

Space Complexity:

$$O(1)$$

Justification:

This constant-time evaluation ensures minimal overhead while providing actionable financial insights.

5.1.4 Invoice Status Determination Algorithm

Objective:

To automatically determine invoice status as *Paid*, *Pending*, or *Overdue*.

Algorithm Description:

The algorithm evaluates the presence of a payment date and compares the due date with the current date to assign the appropriate status.

Algorithm Steps:

1. Check if payment date exists
2. Compare due date with current date
3. Assign status accordingly

Time Complexity:

$$O(1)$$

Space Complexity:

$$O(1)$$

Justification:

This ensures uniform and deterministic invoice state classification across the system.

5.1.5 Inventory Synchronization Algorithm

Objective:

To maintain consistency between invoice creation and inventory stock levels.

Algorithm Description:

For each invoiced product, the algorithm validates available stock and deducts the invoiced quantity. The operation is performed within a transactional context to preserve atomicity.

Algorithm Steps:

1. Iterate through invoice items
2. Validate stock availability
3. Deduct quantity from inventory
4. Commit transaction

Time Complexity:

$$O(n)$$

where *n* is the number of items in the invoice.

Space Complexity:

$$O(1)$$

Justification:
Linear complexity ensures correctness while preventing over-selling or inconsistent inventory records.

5.1.6 Real-Time Dashboard Aggregation Algorithm

Objective:
To compute financial metrics for dashboard visualization in real time.

Algorithm Description:
The algorithm iterates through all invoices and aggregates amounts based on invoice status to compute revenue, outstanding balance, and overdue totals.

Algorithm Steps:

1. Initialize aggregate counters
2. Iterate through invoices
3. Update counters based on status
4. Return aggregated values

Time Complexity:

$$O(n)$$

where *n* is the number of invoices.

Space Complexity:

$$O(1)$$

Justification:

This single-pass aggregation minimizes computational overhead and supports real-time UI updates.

5.1.7 PDF Invoice Generation Algorithm

Objective:

To generate professional, printable invoice documents in PDF format.

Algorithm Description:

Invoice data is rendered into a structured HTML template, styled using print-specific CSS rules, and converted into a PDF document using a client-side rendering engine.

Algorithm Steps:

1. Populate HTML template with invoice data
2. Apply print layout styling
3. Convert HTML to PDF
4. Trigger file download

Time Complexity:

$$O(n)$$

where *n* is the number of invoice elements rendered.

Space Complexity:

$$O(n)$$

Temporary DOM and rendering buffers are required.

Justification:

The algorithm ensures portability and professional document standards while maintaining acceptable performance.

5.1.8 Summary of Algorithm Complexity

Algorithm	Time Complexity	Space Complexity
Invoice Calculation	O(n)	O(1)
Search & Filter	O(n)	O(k)
Client Health Score	O(1)	O(1)
Status Determination	O(1)	O(1)
Inventory Sync	O(n)	O(1)
Dashboard Aggregation	O(n)	O(1)
PDF Generation	O(n)	O(n)

Algorithm (Pseudocode):

Step 1: Start the application.

Step 2: Initialize system database and load Main Dashboard.

Step 3: Display **Main Menu** options:

1. Create New Invoice
2. Search & Filter Invoices
3. Check Client Health Score
4. View Dashboard Metrics
5. Exit

Step 4: Read user input as User_Choice.

Step 5: IF User_Choice is **"Create New Invoice"**, then:

5.1: Input Client Details, Line Items, and Tax Rate.

5.2: FOR each item in Line Items:

Check Inventory_Stock.

- **IF** stock is insufficient, **PRINT** "Error: Low Stock" and **GOTO** Step 3.

- **ELSE**, deduct quantity from Inventory_Stock and add to Subtotal.

5.3: Calculate Tax_Amount = (Subtotal × Tax_Rate) / 100.

5.4: Calculate Total_Amount = Subtotal + Tax_Amount.

5.5: Input Payment_Date, Due_Date, and Current_Date.

5.6: Determine Status:

- **IF** Payment_Date exists -> Status is "Paid".
- **ELSE IF** Current_Date > Due_Date -> Status is "Overdue".
- **ELSE** -> Status is "Pending".

5.7: Save Invoice to Database.

5.8: IF user requests PDF, Render HTML -> Convert to PDF -> Download.

5.9: GOTO Step 3.

Step 6: ELSE IF User_Choice is **"Search & Filter Invoices"**, then:

6.1: Input Search_Term (Client Name) and Status_Filter.

6.2: FOR each Invoice in Database:

- **IF** Invoice Name matches Search_Term **AND** Status matches Status_Filter:
  - Add to Results_List.

6.3: Display Results_List to user.

6.4: GOTO Step 3.

Step 7: ELSE IF User_Choice is **"Check Client Health Score"**, then:

7.1: Input Client ID.

7.2: Retrieve Paid_Invoices, Total_Invoices, and Total_Revenue.

7.3: Calculate Paid_Percentage = (Paid_Invoices / Total_Invoices) * 100.

7.4: Determine Health:

- **IF** Paid_Percentage > 90 **AND** Total_Revenue > 1000 -> "Green".
- **ELSE IF** Paid_Percentage > 70 -> "Yellow".
- **ELSE** -> "Red".

7.5: Display Health Status Color.

7.6: GOTO Step 3.

Step 8: ELSE IF User_Choice is **"View Dashboard Metrics"**, then:

8.1: Set Revenue = 0, Outstanding = 0, Overdue = 0.

8.2: FOR each Invoice in Database:

- **IF** Status is "Paid", add amount to Revenue.
- **IF** Status is "Pending", add amount to Outstanding.
- **IF** Status is "Overdue", add amount to Overdue.

8.3: Update and Display Dashboard Widgets.

8.4: GOTO Step 3.

Step 9: ELSE IF User_Choice is **"Exit"**, then:

9.1: Save all data.

9.2: Close Application.

9.3: GOTO Step 10.

Step 10: End.

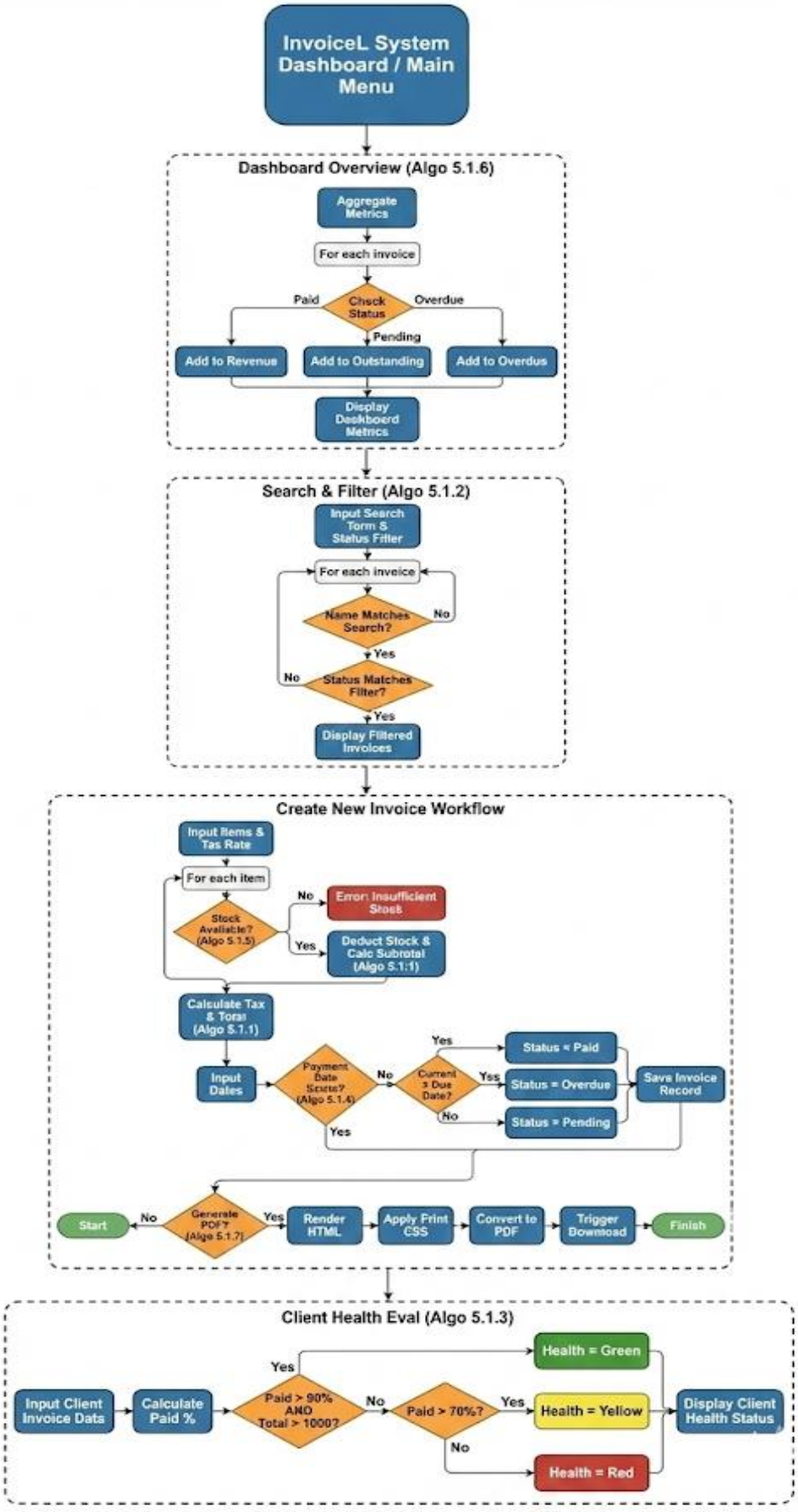

Fig. 5.1.1 System Functionalities

5.2. Output of Implementation

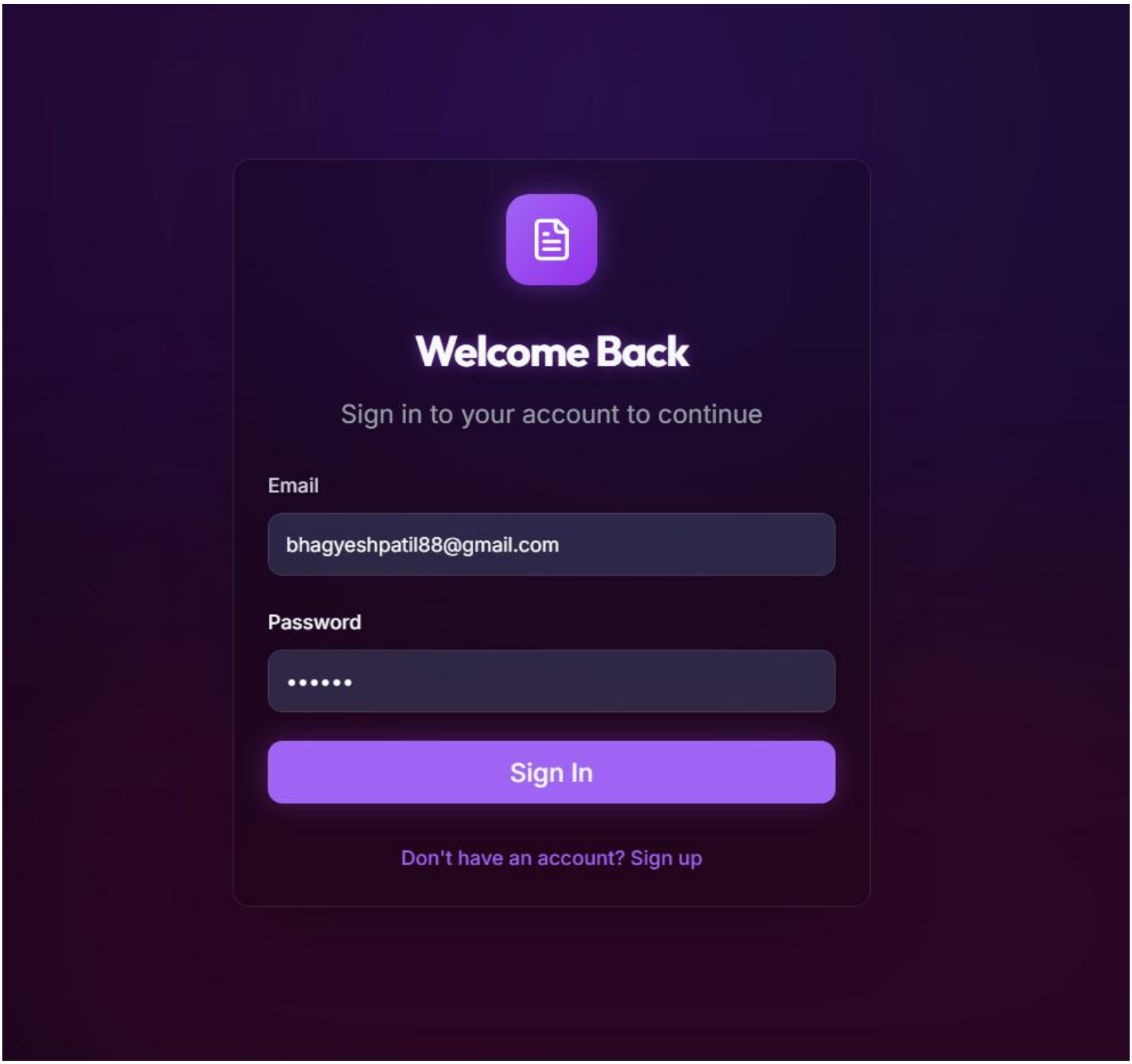

Fig. 5.2.1. Sign In/Sign Up Page.

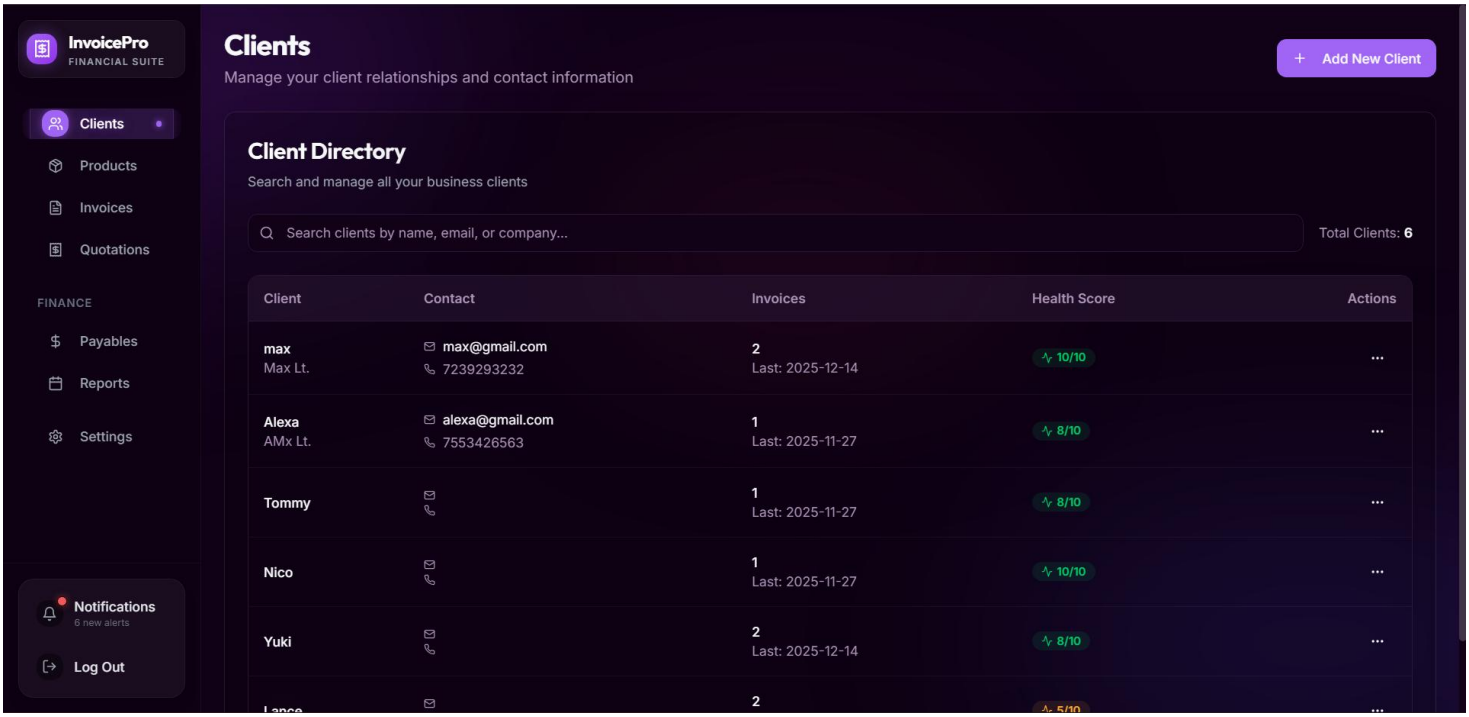

Fig 5.2.2. Clients Session.

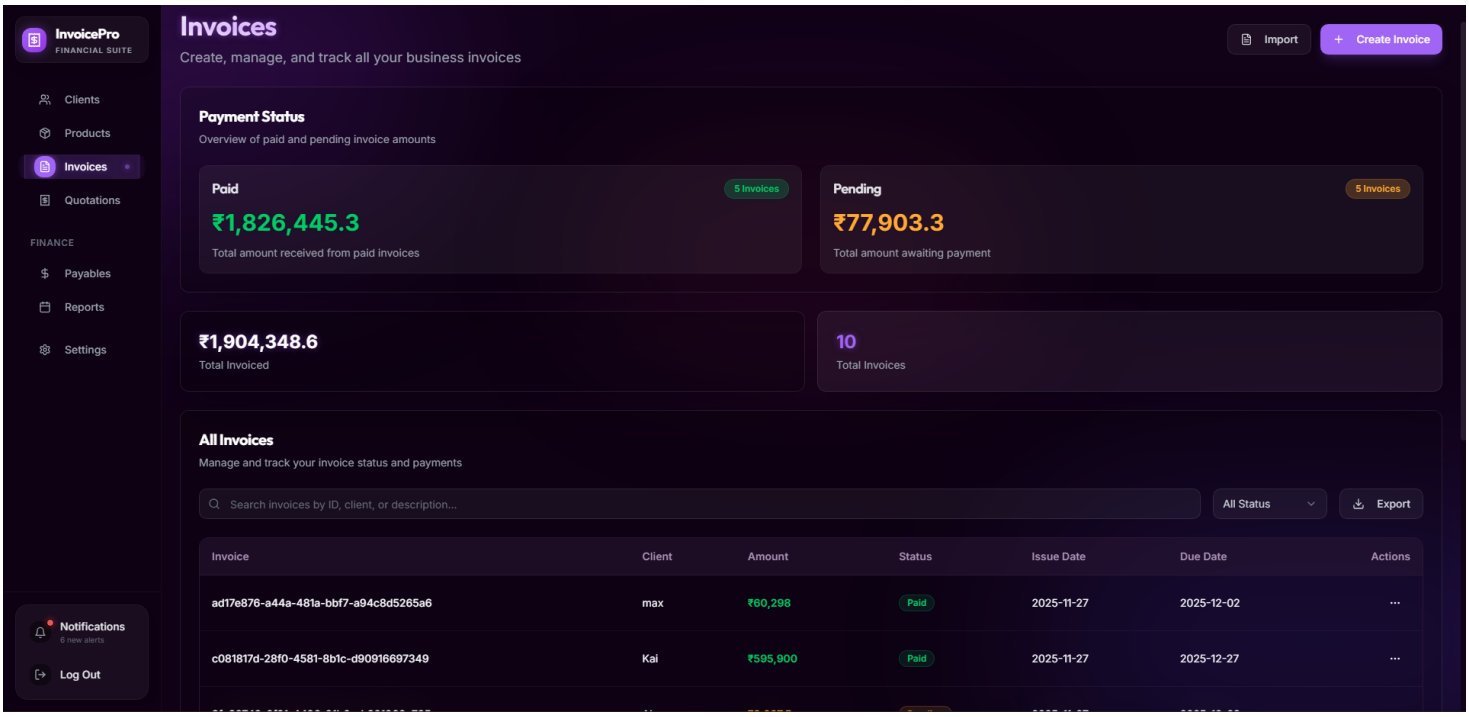

Fig. 5.2.3. Invoice Session.

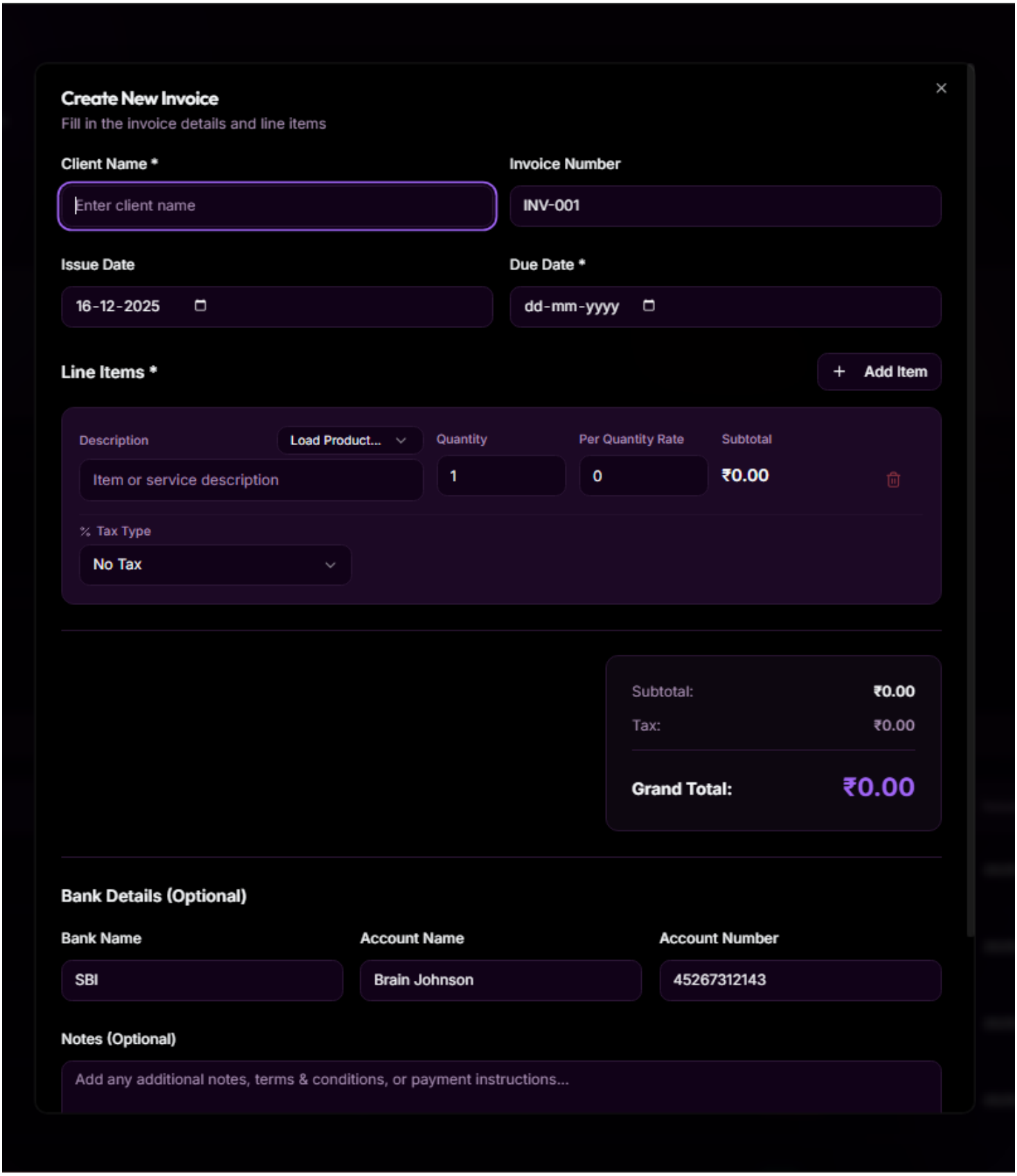

Fig. 5.2.4. Creating Invoice.

JB INT.

Sr.71, Fr.no.13, JWRT Pavillion, SA, USA

Mobile: +1 (911) 901-3050 **Email:** jbrain@gmail.com

Bill To:

max

INVOICE

Invoice No : **ad17e876-a44a-481a-bbf7-a94c8d5265a6**

Invoice Date : 2025-11-27

Due Date : 2025-12-02

DESCRIPTION	UNIT PRICE	QTY	AMOUNT
	60298	1	₹60298.00

SUBTOTAL	₹60298.00
Tax 0%	₹0.00
TOTAL	**₹60298.00**

BANK DETAILS

Bank: SBI
Account Name: Brain Johnson
Account No.: 45267312143
IFSC Code: SBI4563728

THANK YOU

Fig. 5.2.5. The Invoice

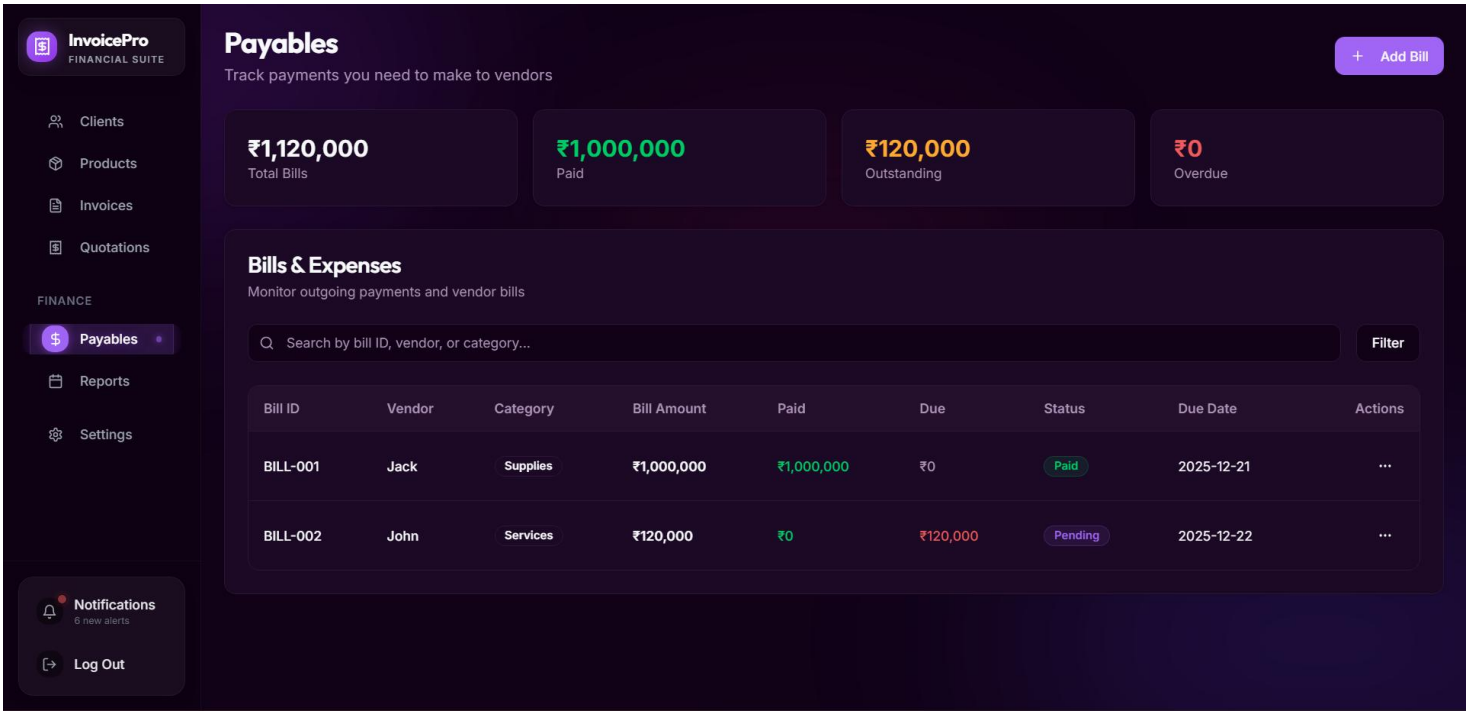

Fig. 5.2.6. Payable Session

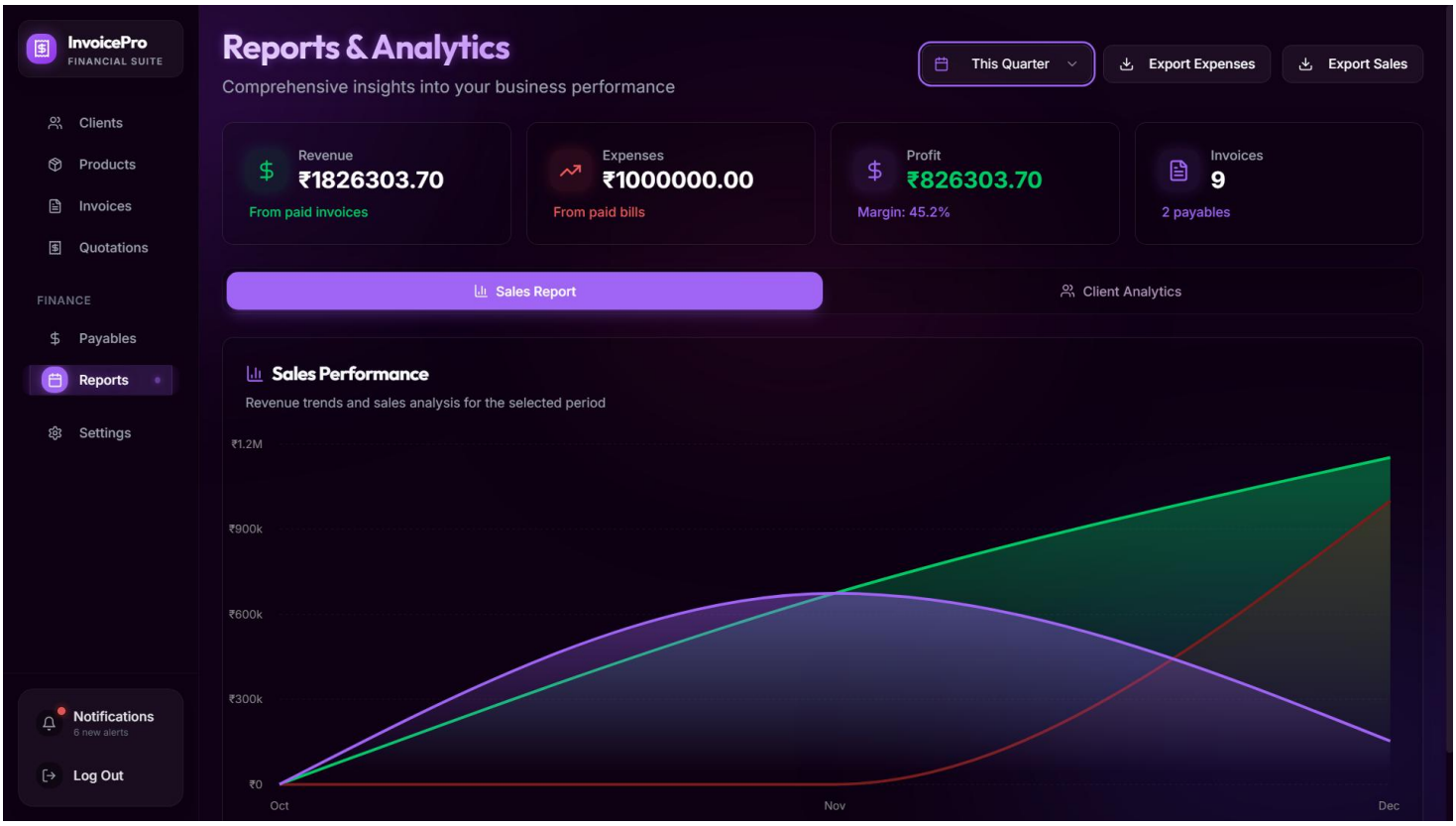

Figure no. 5.2.7. Report & Analytics Session.

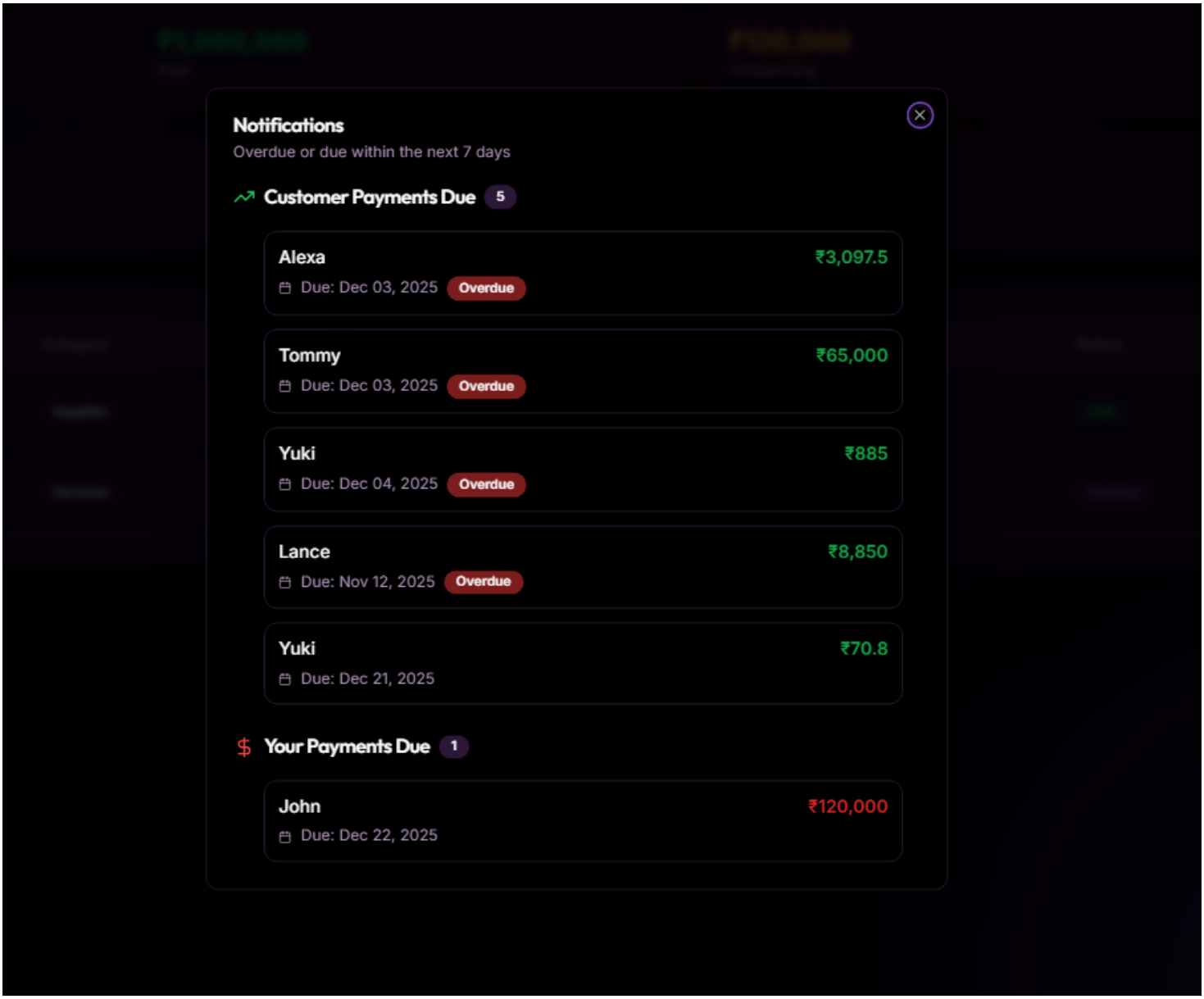

Fig. 5.2.8. Notification of Due which User have to pay and will Get paid within a Week.

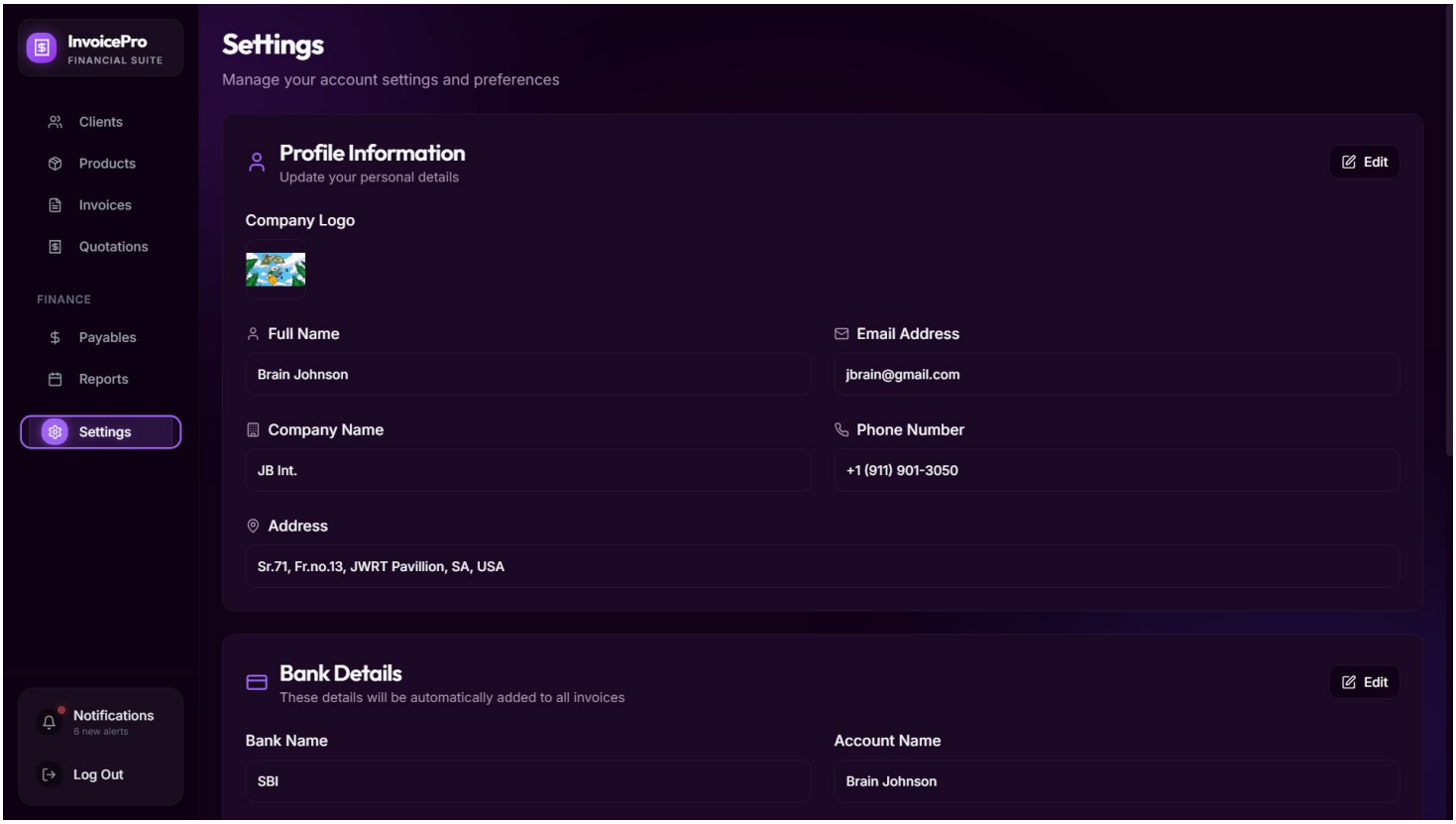

Fig. 5.2.9. Settings

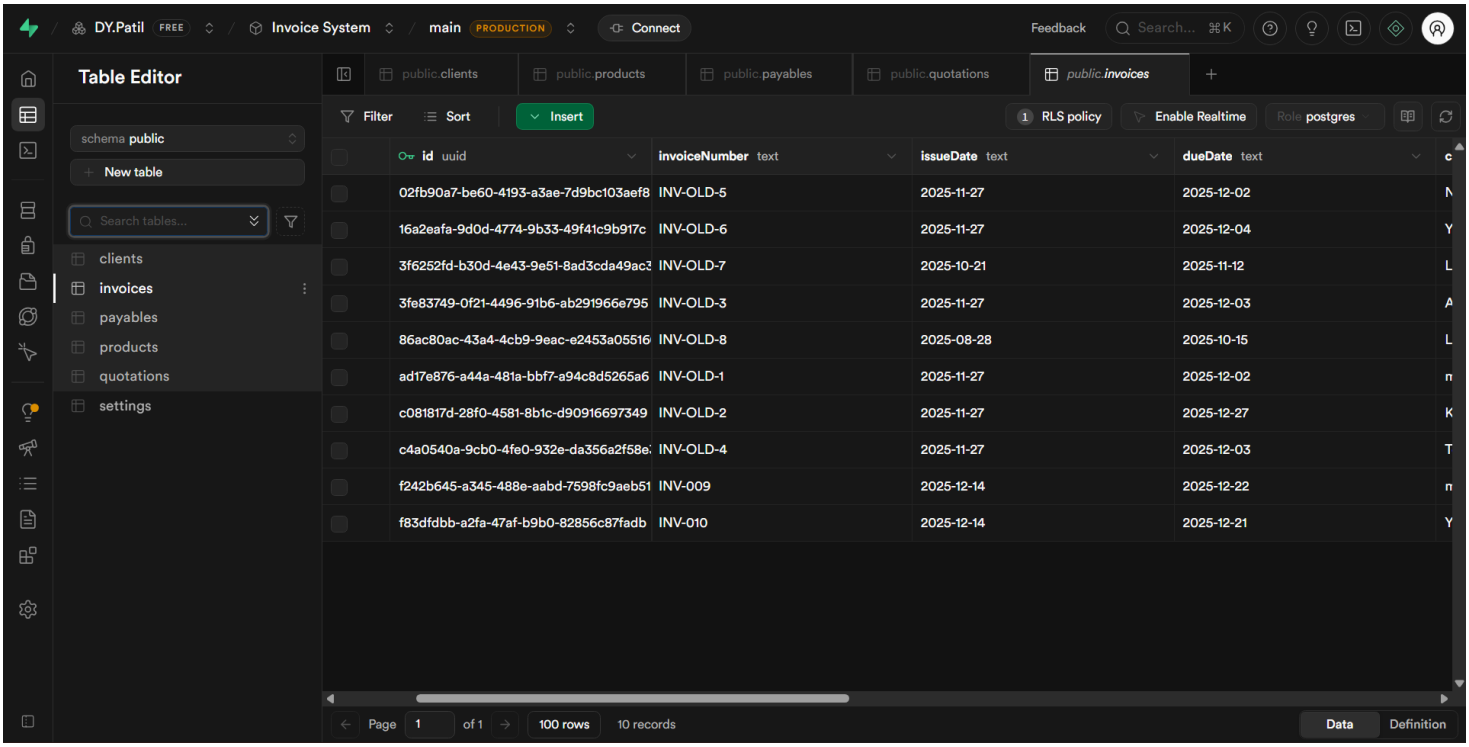

Fig. 5.2.10. Database (Supabase)

RESULTS AND CONCLUSION

6.1. Conclusion

The Invoice Management System developed in this project demonstrates an effective transition from fragmented, spreadsheet-based billing practices to a structured, automated, and accessible digital workflow. By consolidating customer information, quotation and invoice generation, payment tracking, and financial reporting within a unified interface, the system significantly reduces the probability of human error that commonly arises during manual data entry and document preparation. The streamlined workflow ensures faster invoice creation, improved consistency in financial documentation, and enhanced transparency in the tracking of receivables and outstanding dues.

The platform's automated handling of tax calculations, discount applications, and amount validations ensures that generated invoices remain compliant with standard financial practices. Additionally, the embedded analytics module offers stakeholders actionable insights into revenue patterns, payment delays, and cash-flow behavior, thereby enabling more informed decision-making. The offline-capable architecture further enhances usability by allowing uninterrupted access to historical billing data without dependency on network connectivity.

Overall, the project validates the capability of modern front-end technologies—combined with efficient client-side data management—to deliver a robust alternative to expensive enterprise software. The resulting system serves as a reliable, cost-effective solution for small and medium-sized businesses seeking to improve operational efficiency, strengthen financial accuracy, and elevate the professionalism of their billing processes.

6.2. Future Scope

The current system provides a strong foundation for invoice lifecycle management; however, several enhancements can extend its functionality and support larger-scale operational needs:

6.2.1. Payment Gateway Integration:

Integrating online payment gateways (such as Razorpay, Stripe, or PayPal) would allow businesses to embed payment links directly into invoices. This feature would streamline the payment collection process, enable automatic status updates upon successful transactions, and reduce manual reconciliation efforts. Real-time tracking of payment flows would further strengthen the accuracy and timeliness of financial records.

6.2.2. GST/e-Invoicing Compliance:

Future iterations may incorporate automated generation of GST-compliant invoices, e-way bills, and standardized tax summaries. Integration with government platforms for real-time e-invoice generation and return filing would

make the system suitable for businesses operating under regulatory frameworks. This enhancement would significantly minimize compliance burdens and reduce errors in tax reporting.

6.2.3. Mobile Application & Enhanced Offline Mode:

Developing dedicated Android and iOS applications would expand accessibility for field agents, freelancers, and business owners who frequently operate on mobile devices. An enhanced offline mode, allowing invoice creation and editing without internet access, could synchronize data when connectivity is restored. This functionality would be particularly useful in remote locations or during on-site client interactions.

6.2.4. Advanced Analytics, Insights & Forecasting:

Introducing richer analytics—such as customer purchasing trends, seasonal sales patterns, and profitability metrics—would provide deeper business intelligence. Predictive algorithms could analyze payment histories to forecast overdue invoices, highlight high-priority follow-ups, and recommend optimal billing cycles. Such features would elevate the system from a transactional tool to a strategic decision-support platform.

6.2.5. Inventory & Accounting System Integration

Seamless integration with established platforms such as Tally, Zoho Books, or QuickBooks would eliminate duplicate data entry and ensure consistency between billing, accounting, and inventory systems. Linking invoice items to stock levels would enable real-time inventory updates and low-stock alerts, further increasing operational efficiency.

6.2.6. Role-Based Access Control & Audit Trails:

Introducing multi-user support with granular permissions would allow businesses to assign roles such as Admin, Accountant, and Sales Executive. Built-in audit trails could track all user activities—such as edits, approvals, and deletions—ensuring accountability and facilitating compliance with internal and external audits.

6.2.7. Multi-Currency & Localization Support:

To serve international users, the system can be extended with multi-currency handling, region-specific tax rules, and multi-language support. These capabilities would broaden the system's applicability for businesses operating across borders or serving global clients.

6.2.8. AI-Assisted Data Extraction and Automation:

Incorporating OCR (Optical Character Recognition) and AI-based extraction models would enable automatic reading of purchase orders, receipts, and vendor invoices. Such automation would significantly reduce manual entry time, improve accuracy, and accelerate workflow completion. AI could also suggest corrections, flag anomalies, and recommend missing fields based on historical patterns.

Appendices

Appendix A – Sample Invoice Format

This appendix includes a representative sample of a generated invoice produced by the system. The invoice layout consists of clearly delineated sections such as company identity, customer details, invoice number and date, itemized line entries, applicable taxes, subtotals, discounts, grand totals, and payment terms. Annotations may be added around the screenshot or embedded PDF to highlight specific design decisions—for example, the placement of the company logo for branding consistency, the grid alignment of line items for readability, or the use of bold typography for financial totals to enhance visibility. The format illustrates how the system ensures document clarity and professional presentation.

Appendix B – Data Dictionary

This appendix provides a structured data dictionary describing the core entities used in the system. Each entity is documented with its attributes, data types, constraints, and validation rules. A sample is shown below:

Entity	**Attribute**	**Type**	**Constraint / Rule**
Customer	customerId	string	Unique identifier
Customer	name	string	Required, min length 2
Invoice	invoiceNumber	string	Unique, pattern `INV-YYYYMM-####`
Invoice	issueDate	date	Required
Invoice	totalAmount	number	Must be ≥ 0
InvoiceLine	quantity	number	Must be ≥ 1
Payment	paymentDate	date	Optional
Payment	amount	number	Must not exceed invoice balance

References

[1] R. G. Eccles and M. P. Krzus, *One Report: Integrated Reporting for a Sustainable Strategy*. Hoboken, NJ, USA: Wiley, 2010.
Uploaded on: Wiley Online Library (Academic Publisher)

[2] Intuit Inc., "QuickBooks Accounting Software," Intuit, 2024.
Uploaded on: Official Intuit Product Documentation

[3] SAP SE, "SAP Business One – ERP for Small and Medium Enterprises," SAP Documentation, 2024.
Uploaded on: SAP Official Documentation Portal

[4] Zoho Corporation, "Zoho Books: Online Accounting Software," Zoho Documentation, 2024.
Uploaded on: Zoho Official Product Website

[5] Tally Solutions Pvt. Ltd., "TallyPrime Product Overview," Tally Solutions, 2024.
Uploaded on: Official Tally Solutions Website

[6] Organisation for Economic Co-operation and Development (OECD), *SME Digitalisation: Policies, Challenges and Opportunities*. Paris, France: OECD Publishing, 2021.
Uploaded on: OECD iLibrary (International Policy Publisher)

[7] V. Venkatesh, M. G. Morris, G. B. Davis, and F. D. Davis, "User Acceptance of Information Technology: Toward a Unified View," *MIS Quarterly*, vol. 27, no. 3, pp. 425–478, 2003.
Uploaded on: MIS Quarterly & JSTOR Digital Library

[8] Microsoft Corporation, "Microsoft Excel for Small Business," Microsoft Documentation, 2023.
Uploaded on: Microsoft Official Documentation Portal

[9] R. R. Panko, "What We Know About Spreadsheet Errors," *Journal of End User Computing*, vol. 10, no. 2, pp. 15–21, 1998.
Uploaded on: IGI Global Academic Digital Library

[10] CFO Magazine Editorial Team, "The Hidden Risks of Spreadsheet Accounting," *CFO Magazine*, 2020.
Uploaded on: CFO Magazine Online (Industry Publication)

[11] FreshBooks Inc., "FreshBooks Invoicing Software Overview," FreshBooks, 2024.
Uploaded on: FreshBooks Official Website

[12] Wave Financial Inc., “Wave Invoicing Platform,” Wave Accounting, 2024.
Uploaded on: Wave Accounting Official Website

[13] Invoice Ninja LLC, “Invoice Ninja Open Source Invoicing System,” Invoice Ninja, 2024.
Uploaded on: Invoice Ninja Official Website

[14] Gartner Research, “Challenges of SaaS Adoption for Small Enterprises,” Gartner Inc., 2022.
Uploaded on: Gartner Research Portal (Subscription-Based)

[15] A. Ghosh, “Data Privacy and Security Challenges in Cloud-Based Accounting Systems,” *IEEE Security & Privacy*, vol. 18, no. 4, pp. 72–76, 2020.
Uploaded on: IEEE Xplore Digital Library
Available: https://ieeexplore.ieee.org

[16] S. Kumar, A. Singh, and R. Sharma, “Design and Implementation of a Web-Based Billing System Using PHP and MySQL,” *International Journal of Computer Applications*, vol. 174, no. 5, pp. 25–30, 2017.
Uploaded on: IJCA Online Journal Portal

[17] GitHub Inc., “Open Source Invoicing and Billing Systems,” GitHub Topics, 2024.
Uploaded on: GitHub Open Source Repository Index

[18] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts*, 7th ed. New York, NY, USA: McGraw-Hill Education, 2019.
Uploaded on: McGraw-Hill Education & Google Books

[19] IEEE Software Editorial Board, “Usability Challenges in Academic Software Prototypes,” *IEEE Software*, vol. 38, no. 3, pp. 112–118, 2021.
Uploaded on: IEEE Xplore Digital Library
Available: https://ieeexplore.ieee.org

[20] Meta Platforms Inc., “React – A JavaScript Library for Building User Interfaces,” React Documentation, 2024.
Uploaded on: Official React Documentation Website

[21] Microsoft Corporation, “TypeScript Handbook,” Microsoft Docs, 2024.
Uploaded on: Microsoft Developer Documentation

[22] Tailwind Labs Inc., “Tailwind CSS Documentation,” Tailwind CSS, 2024.
Uploaded on: Tailwind CSS Official Documentation

[23] Mozilla Developer Network (MDN), “IndexedDB API,” Mozilla Foundation, 2024.
Uploaded on: MDN Web Docs (Mozilla)

[24] J. Paraschiv, “Client-Side PDF Generation Techniques Using JavaScript,” *arXiv preprint*, arXiv:2106.04521, 2021.
Uploaded on: arXiv.org (Cornell University Open Repository)