CLIENT VAULT– AN WEB APPLICATION

A Project Report

Submitted in partial fulfillment of the

Requirements for the award of the Degree of

BACHELOR OF SCIENCE (INFORMATION TECHNOLOGY)

By

Jitendra Kashiprasad Yadav

3BSCIT65

Under the esteemed guidance of

Mr. Pradnya Nehete

Assistant Professor-Project Coordinator

DEPARTMENT OF INFORMATION TECHNOLOGY

SAHYOG COLLEGE OF MANAGEMENT STUDIES

(Affiliated to University of Mumbai)

THANE(W)- 400 601

MAHARASHTRA

2025-26

PROFORMA FOR THE APPROVAL PROJECT PROPOSAL

PNR No.: **2023016400441671** Roll no: **65**

1. Name of the Student

   Jitendra Kashiprasad Yadav

2. Title of the Project

   Client Vault – An Web Application

3. Name of the Guide

   Prof. Saugat Das

4. Teaching experience of the Guide ______________________

5. Is this your first submission? Yes ☐ No ☐

Signature of the Student Signature of the Guide

Date: ………………… Date: ……………….

Signature of the Coordinator

Date: …………………

CERTIFICATE

This is to certify that the project entitled, "**Client Vault – An Web Application**", is Bonafide work of **Jitendra Kashiprasad Yadav** bearing Seat No: **3BSCIT65** submitted in partial fulfilment of the requirements for the award of degree of BACHELOR OF SCIENCE in INFORMATION TECHNOLOGY from University of Mumbai.

Internal Guide **Coordinator**

External Examiner

Date: **College Seal**

ABSTRACT

Client Vault is a next-generation, comprehensive CRM-style business management application designed and developed using the MERN stack (MongoDB, Express.js, React.js, Node.js) with Tailwind CSS for modern, responsive, and scalable user interfaces. The core idea behind Client Vault is to simplify, centralize, and optimize the way businesses manage their clients, projects, invoices, and tasks by providing a single unified system that eliminates fragmentation and manual inefficiencies.

The platform allows businesses to store and manage client details, link them with associated projects and invoices, and maintain structured invoice records with a printable view feature for professional documentation. The project management module supports status monitoring, while the task management module ensures smooth assignment and tracking of work against deadlines.

A dynamic dashboard consolidates critical information by offering real-time analytics such as the number of clients, active projects, pending tasks, and overdue invoices. All data interactions are synchronized through local storage, ensuring consistency across different modules.

Designed with modular MVC architecture, the system is highly scalable, enabling future enhancements such as SaaS deployment and microservices integration. With its responsive design, interactive interface, and smooth UI animations, Client Vault delivers a professional, user-friendly experience, making it a practical solution for businesses seeking to streamline operations.

ACKNOWLEDGEMENT

We would like to express our deepest gratitude to everyone who contributed to the successful completion of our project, “Client Vault – A Web Application”. This project would not have been possible without the continuous support, guidance, and encouragement we received from the following individuals and sources:

- Our project mentor, whose invaluable advice, insights, and direction helped us navigate challenges and stay focused throughout the development process.
- Our friends and family, who consistently encouraged us, motivated us during difficult times, and provided unwavering emotional support.
- The academic resources and learning materials that enhanced our understanding of web technologies, the MERN stack, and software engineering principles, which enabled us to design and build a professional application.

We are truly grateful for the support and encouragement we have received, and we sincerely acknowledge the contribution of each individual and resource that played a role in the successful completion of this project. Thank you!

DECLARATION

I hereby declare that the project entitled, *"*Client Vault – A Web Application*"* is the outcome of my own work and effort. This project has not been submitted, either fully or partially, to any other university or institution for the award of any degree or diploma. To the best of my knowledge, no part of this work has been duplicated or plagiarized from any other source without proper citation.

The project is carried out in partial fulfillment of the requirements for the award of the degree of BACHELOR OF SCIENCE (INFORMATION TECHNOLOGY*)* under the curriculum of the University of Mumbai.

Name and Signature of the Student

TABLE OF CONTENTS

List of Tables

List of Figures

Chapter 1

INTRODUCTION

1.1 Background

The management of clients, projects, invoices, and tasks has always been a crucial aspect of business operations across industries. Traditionally, these activities were handled manually through paperwork, spreadsheets, or basic software tools, which often led to inefficiencies, data duplication, and difficulty in maintaining accurate records. As organizations expanded and client bases grew, the need for a more streamlined, centralized, and digital solution became evident.

With the rapid rise of the Internet in the late 1990s and early 2000s, businesses began adopting digital platforms to manage their customer relationships and internal workflows. This gave birth to Customer Relationship Management (CRM) systems, which provided companies with the ability to store client information, monitor ongoing projects, generate invoices, and assign tasks in a structured manner.

Over the years, these systems have evolved significantly, integrating advanced technologies such as cloud computing, responsive web design, and real-time analytics. Today, CRM-style applications are no longer limited to large enterprises but are accessible to small and medium businesses as well, offering professional management tools in an affordable and scalable manner.

Client Vault – A Web Application is built in this context, leveraging the MERN stack with Tailwind CSS to create a comprehensive and interactive platform that brings together client, project, invoice, and task management in one unified system. It addresses the challenges of traditional methods by offering automation, structured data handling, and real-time insights, ensuring businesses can operate more efficiently and make informed decisions.

1.2 Objectives

The main objectives of this system are to manage the details of clients, projects, invoices, and tasks in a structured and efficient way. The project is primarily built for administrative use, ensuring that the administrator has complete control and access over all modules.

The purpose is to build a web application program that reduces manual efforts in managing business operations, enhances productivity, and ensures accurate record-keeping. The system enables storing and updating client details, linking them with respective projects and invoices, and monitoring project statuses in real-time.

It also tracks all pending tasks, overdue invoices, and active projects through a centralized dashboard, ensuring better decision-making and transparency. By automating these workflows, the system eliminates redundancies, reduces errors, and provides a professional and scalable solution for businesses

1.3 Purpose, Scope and Applicability

1.3.1 Purpose

The Purpose of this project is to create an efficient, user-friendly food delivery system that streamlines the process of ordering and delivering food. By digitizing and automating various aspects of food ordering, the project aims to enhance convenience for customers, improve operational efficiency for restaurants, and offer real-time insights and tracking.

This system addresses the growing demand for online food services, reducing delays, enhancing customer satisfaction, providing a scalable solution for modern food delivery need. Its theoretical framework is based on optimizing user experience, logistics, and services management through advanced technology.

1.3.2 Scope

This project covers the development of a web-based CRM-style application that focuses on the key functions of client management**,** project tracking, invoice generation, and task monitoring. The system also integrates a dynamic dashboard that provides real-time analytics such as the number of clients, active projects, pending tasks, and overdue invoices, enabling businesses to make informed decisions efficiently.

The methodology involves using a MERN stack (MongoDB, Express.js, React, and Node.js) for scalable and efficient backend–frontend integration, along with Tailwind CSS for modern and responsive user interface design. The system ensures data consistency across different modules through synchronization, while following MVC architecture to maintain modularity, scalability, and readiness for future SaaS or microservices integration.

Key assumptions include the availability of a stable internet connection and access to modern browsers or devices compatible with the application. The limitations may include potential technical issues such as server downtime, dependency on local storage synchronization, or integration challenges with third-party services like payment gateways and cloud-based APIs when extended in the future.

1.3.3 Applicability

The applicability of this project extends significantly within the client relationship management (CRM) domain and the broader business technology landscape. Directly, the application enhances organizational efficiency by offering a seamless platform for storing client details, linking projects, generating invoices, and monitoring tasks**.** This streamlined workflow reduces manual effort, minimizes errors, and improves transparency across different business operations. For business owners and administrators, the system provides valuable real-time analytics and reports, such as active projects, overdue invoices, and pending tasks, which can inform better strategic and operational decision-making.

Indirectly, the project demonstrates the effective use of modern web technologies such as the MERN stack, Tailwind CSS, and MVC architecture**,** which can be adapted and extended across various industries for process automation. The modular and scalable design makes it applicable not only for small-to-medium businesses but also as a future-ready SaaS product, enabling subscription-based deployment for multiple organizations.

Additionally, the application contributes to the digital transformation of businesses, fostering productivity, accountability, and data-driven decision-making. By integrating advanced technologies such as cloud computing and potentially AI-driven insights, the system highlights the role of modern web applications in promoting innovation, economic growth, and long-term sustainability in a connected digital economy**.**

1.4 Achievements

Upon completion of this project, a comprehensive understanding of full-stack web application development was gained, particularly in the context of client, project, invoice, and task management systems. This included acquiring skills in user interface design with Tailwind CSS, backend integration using Node.js and Express, database management with MongoDB, and the implementation of real-time data synchronization across multiple modules. The project contributed significantly to the field by providing a robust and scalable solution that enhances organizational efficiency, demonstrating best practices in software development, modular design, and project management**.**

The original objectives of developing a user-friendly platform for managing clients, projects, invoices, and tasks were fully achieved, resulting in a cohesive system that meets the needs of both administrators and business users. Additionally, the integration of features such as dynamic dashboards, structured invoice templates, and real-time analytics exceeded the initial goals, offering valuable insights into organizational performance and improving decision-making.

Overall, the project not only met its objectives but also laid the groundwork for future SaaS deployment and possible microservices integration, demonstrating the potential for further enhancements through cloud computing, automation, and emerging technologies such as AI-driven analytics**.**

1.5 Organisation of Report

The project report is organized into several chapters that provide a detailed account of the development process, methodologies, and findings. Each chapter is structured to ensure a logical flow and an in-depth understanding of the project.

1. **Introduction**
   This chapter provides an overview of the project, including its objectives, scope, and purpose. It lays the foundation for understanding the significance of developing the Client Vault system for secure client, project, task, and invoice management.
2. **Literature Review**
   The literature review covers existing client and project management systems and related technologies. It explores various approaches and solutions used in the industry and highlights the need for a more secure, scalable, and user-friendly solution like Client Vault.
3. **System Analysis and Design**
   This chapter focuses on the analysis of the problem domain and presents the conceptual models, including Data Flow Diagrams (DFD), Entity-Relationship (ER) diagrams, and System Flowcharts. The design considerations for both the administrative dashboard and user functionalities are also covered in this section.

4. **Technologies and Tools Used**

   This chapter explains the MERN stack (MongoDB, Express.js, React, Node.js), along with Tailwind CSS for UI design, JWT for authentication, and optional cloud services for storage and deployment. Each technology is described with its role in the ClientVault system.

5. **Implementation**

   The implementation chapter describes how the system was developed, including coding, integration of various modules such as client management, project tracking, invoice generation, and task assignments. It also discusses challenges faced during development and how they were resolved.

6. **Testing and Validation**

   This chapter outlines the test cases used to validate the functionality of the system. It includes functional and non-functional testing results and explains how issues were identified and resolved. Test cases for registration, login, client management, project creation, and invoice generation are included.

7. **Security Considerations**

   This section discusses the real-time and security challenges faced during development, such as safeguarding client data, ensuring secure authentication, and preventing unauthorized access. It also explains the policies, encryption techniques, and security frameworks implemented.

8. **Conclusion and Future Scope**

   The final chapter summarizes the accomplishments of the project and evaluates whether the objectives were met. It also highlights possible future improvements, such as integrating AI-powered business analytics, automated invoice reminders, or expanding Client Vault into a SaaS-based platform for wider adoption.

Chapter 2

Survey of Technologies

Introduction

In the development of secure client and project management systems, selecting an appropriate technology stack is essential for ensuring scalability, security, performance, and long-term maintainability. For Client Vault, the technologies chosen must support a wide range of features, including client information storage, project and task tracking, invoice generation, user authentication, and data security

The system requires a stack that can deliver real-time data processing, a responsive user interface, and robust database management, while also maintaining strict security protocols to safeguard sensitive client and financial data. The project adopts a modern full-stack approach using powerful frameworks and libraries tailored for web-based enterprise applications.

Client Vault leverages the MERN stack (MongoDB, Express.js, React, Node.js) as the core technology for development, supported by additional tools and frameworks such as Tailwind **CSS** for styling for authentication, and cloud-based services for deployment and backup. This combination ensures a highly scalable, secure, and user-friendly system suitable for handling diverse business needs efficiently.

Technologies Used – Web

Client Vault is built upon the robust and versatile MERN stack, a popular JavaScript-based technology stack for developing secure and dynamic web applications. It consists of four core technologies**:** MongoDB, Express.js, React.js, and Node.js, all of which work seamlessly together to provide a full-stack solution. To further enhance the system’s reliability, TypeScript was integrated for strong type safety, while Zod was adopted for runtime validation, ensuring that client and project data remains accurate and secure.

MongoDB

MongoDB is a NoSQL database that stores information in flexible, JSON-like documents, instead of traditional rows and columns. This structure makes it highly effective for managing hierarchical business data, such as client records, project details, invoices, and user roles. For Client Vault, MongoDB provides the flexibility to handle complex and varied data structures while maintaining scalability. Its ability to distribute data across multiple servers ensures smooth performance even as the number of clients and projects grows, making it ideal for enterprise-level client management systems

Express.js

Express.js is a lightweight and powerful web application framework for Node.js. It simplifies the backend development process by offering routing, middleware, and error-handling functionalities. Within Client Vault, Express.js manages the server-side logic, including client authentication, project data management, and secure communication between the frontend and the database. By enabling the creation of RESTful APIs, Express.js ensures efficient and secure interaction between the React-based frontend and the backend services.

React.js

React.js is a frontend JavaScript library used to build fast, interactive, and scalable user interfaces. It follows a component-based architecture, allowing developers to reuse UI elements such as client dashboards, project boards, and invoicing panels. With its virtual DOM, React updates only the parts of the interface that change, leading to better performance and responsiveness. In Client Vault, React.js powers dynamic features like project tracking, client profile management, and real-time notifications, ensuring that users have a smooth and intuitive experience while interacting with the system.

Node.js

Node.js is a runtime environment that enables JavaScript code to run on the server side. Its non-blocking, event-driven architecture makes it highly efficient at handling multiple client requests simultaneously. In Client Vault, Node.js facilitates fast and reliable server-side execution for tasks such as authenticating users, handling large amounts of project data, and managing secure financial transactions. Since both frontend and backend share the same programming language (JavaScript/TypeScript), development is simplified, reducing complexity and increasing maintainability.

GitHub

GitHub is a cloud-based platform for version control and collaborative software development, built around Git. It allows developers of Client Vault to efficiently manage code, track changes, and collaborate across teams through features like branching, pull requests, and code reviews. By using GitHub, Client Vault ensures organized development workflows, continuous integration with CI/CD pipelines, and secure storage of source code. Its issue tracking and project management tools also help streamline task assignment and progress monitoring, making it ideal for scalable, team-based development

Docker

Docker is a containerization platform that packages applications and their dependencies into lightweight, portable containers. For Client Vault, Docker ensures consistent environments across development, testing, and production, eliminating the common “works on my machine” problem. By containerizing the application, Docker enables seamless deployment, easier scaling, and efficient resource usage. Its support for microservices architecture also aligns with Client Vault’s future SaaS readiness, making the system modular, maintainable, and cloud-friendly.

Chapter 3

Requirements and Analysis

3.1 Problem Definition

The core problem that the project addresses is the lack of a centralized, efficient, and secure client management platform that enables businesses to organize client information, track project progress, manage documents, and handle financial transactions seamlessly. Many existing systems are either too generic, lack customization, or fail to provide the scalability and security required by growing businesses. Additionally, fragmented tools often force organizations to use multiple applications for client data, project tracking, and invoicing, which results in inefficiencies, data duplication, and poor user experience.

Client Vault aims to resolve these challenges by providing a unified solution where businesses can securely manage clients, projects, and financial workflows in one place, while ensuring scalability, data integrity, and ease of use.

Sub-Problems

A. Client Data Management and Usability

The first sub-problem is the absence of a smooth and user-friendly interface for storing and managing client data. Businesses often struggle with scattered records across different tools, leading to inefficiency. The system needs to allow easy onboarding of clients, quick search and filtering, and intuitive dashboards that present client details, ongoing projects, and financial records in a single view.

B. Project Tracking and Collaboration

The second sub-problem is the need for real-time project monitoring and updates. Existing tools often lack proper tracking mechanisms or integration between tasks, milestones, and client communications. Businesses require a solution where team members can collaborate, update project statuses, and clients can transparently view progress without delays or miscommunication.

C. Secure Document and File Management

The third sub-problem is the lack of a secure and structured document management system. Businesses deal with sensitive client files such as contracts, invoices, and proposals that need to be stored safely and accessed conveniently. The platform must provide features like version control, secure file sharing, and role-based access to ensure privacy and integrity of documents.

D. Scalability and Performance

The fourth sub-problem focuses on scalability and performance. As businesses expand and onboard more clients and projects, the system should handle larger volumes of data without compromising performance. The platform must be capable of supporting hundreds of concurrent users while maintaining a fast and reliable experience.

E. Secure Financial Transactions

Another critical sub-problem is the integration of secure and flexible financial management features. Businesses require the ability to generate invoices, track payments, and support multiple payment gateways. The system must ensure data security during transactions while preventing fraud and unauthorized access.

F. Multi -Platform Accessibility

The final sub-problem is ensuring multi-platform support, where both businesses and clients can access the system seamlessly through web and mobile applications. Whether a manager is tracking invoices on a desktop or a client is checking project updates via mobile, the experience should remain consistent and intuitive across devices.

3.2 Requirement Specification

The requirements for the Client Vault system focus on the features and functionalities necessary to provide an efficient, secure, and user-friendly platform for managing clients, projects, and financial operations, independent of how they will be implemented. These requirements ensure that the system meets business needs and addresses key operational challenges.

Functional Requirements

A. User Registration and Authentication

Users, including business administrators, employees, and clients, must be able to register and log in securely. The system should support email-based authentication as well as social logins for ease of access. Role-based access control must be implemented to ensure that sensitive information is only accessible to authorized users.

B. Client and Project Management

Businesses must be able to onboard new clients, manage client profiles, and track associated projects. The system should allow assigning tasks, setting milestones, and monitoring progress, ensuring smooth collaboration between teams and clients.

C. Document and File Management

The platform must provide secure storage for important client documents, such as contracts, invoices, and proposals. Features should include version control, secure sharing, and role-based access permissions to maintain confidentiality and data integrity.

D. Communication and Notifications

Both businesses and clients should be able to receive timely notifications about project updates, deadlines, invoices, and other important events. The system should also support internal communication tools or integrations to streamline collaboration and reduce miscommunication.

E. Financial Management and Payment Processing

The system must support creating and sending invoices, tracking payments, and managing financial records. It should integrate with multiple secure payment gateways and ensure that all transactions are encrypted to protect sensitive financial data.

F. Admin Management

The system should feature an admin panel that allows administrators to manage users, oversee project activities, monitor performance metrics, and resolve disputes or technical issues efficiently. The admin should also have control over access rights and data security policies.

3.3 Planning and Scheduling

Planning and Scheduling are critical components of software development, as they ensure that tasks are organized, resources are allocated, and the project proceeds smoothly towards completion. In this project, planning involves breaking down the major tasks and milestones required to complete the Client Vault platform, considering dependencies and constraints. Scheduling focuses on ensuring that resources, such as time, team members, and tools, are available to meet deadlines and deliver the project as planned.

Planning

Planning is about identifying all the small, manageable tasks that need to be completed to achieve the project's goal. Each phase of the project is broken down into individual tasks or subtasks. Additionally, planning must account for constraints like task dependencies, resource availability, and deadlines, which control when specific tasks can start and finish.

Key Phases

1. **Requirement Gathering**: Define the functional and non-functional requirements for client, project, and financial management features.
2. **System Design**: Outline the architecture, database schema, and create UI/UX mockups for the web dashboard and client portal.
3. **Development**: Break down backend and frontend development tasks, including authentication, client management, document storage, and invoicing modules.
4. **Testing**: Conduct unit testing, integration testing, and user acceptance testing to ensure functionality, security, and usability.
5. **Deployment**: Plan the release of the Client Vault web application on a cloud environment and configure backup and security policies.

Scheduling

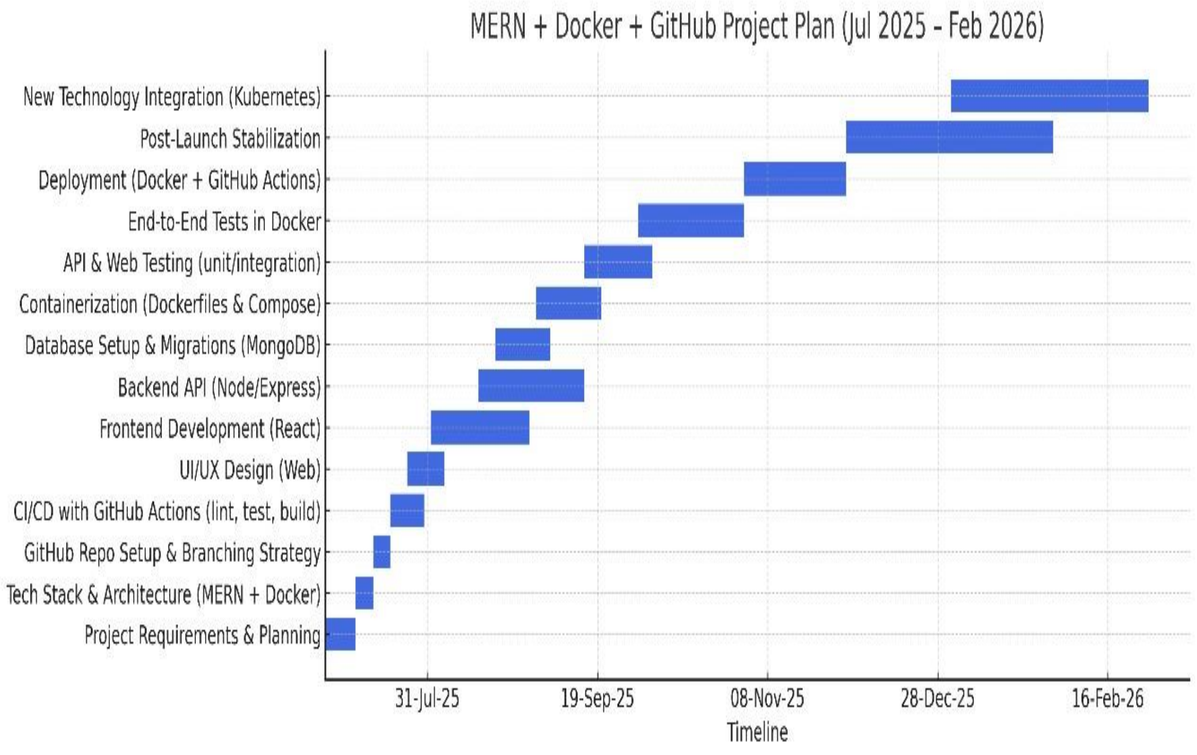
Once the tasks are planned, scheduling involves assigning start and end dates, estimating durations, and determining whether the available resources are sufficient to complete each task. Scheduling helps allocate personnel, tools, and other resources to ensure that tasks are completed efficiently.

Key Elements

1. **Task Sequencing**: Arranging tasks in a logical order, such as requirement gathering → system design → development → testing → deployment.
2. **Time Estimation**: Providing realistic time frames for each task based on its complexity, such as database design taking longer than UI prototyping.
3. **Milestones**: Identifying major checkpoints (e.g., completion of client management module, integration of payment processing, deployment on the cloud).
4. **Monitoring Progress**: Regularly tracking progress against the project schedule and adjusting plans in response to risks, delays, or scope changes.

Gantt Chart

A Gantt chart is a visual representation of the project schedule, showing tasks and their durations along a timeline. It includes start and end dates for tasks, task dependencies, and milestones, providing a clear view of the project's progress. For the Client Vault project, the Gantt chart helps in planning and tracking activities such as requirement gathering, system design, development, testing, and deployment. It also highlights dependencies between tasks, ensuring that delays in one stage do not impact the overall timeline.



3.4 Software and Hardware Requirements

Hardware Requirements

Processor	Dual Core (Intel i5)
Ram	8 GB
Disk Capacity	256 GB SSD
Peripherals	Mouse, Keyboard
Graphics	Integrated Graphics

Software Requirements

Operating System	Window 11
Development Tools	Visual Studio Code

Backend Development	Node.js, Express.js, MongoDB
Frontend Development	React

3.5 Preliminary Product Description

Requirements and Objectives

The development of Client Vault, a secure web-based client and document management system, aims to streamline the way businesses handle client data, store documents, and manage interactions. The primary objectives of this system are to ensure data security**,** efficient client management**,** and a user-friendly interface for administrators, employees, and clients. The main requirements and objectives include:

- **Secure Data Storage and Access Control:** Client Vault must ensure that all client documents and sensitive information are stored securely using encryption and access control mechanisms. Only authorized users should be able to access or modify data.
- **Centralized Client Management:** The system should provide businesses with a centralized platform to maintain client profiles, contact details, documents, and history of interactions.
- **Role-Based Access Permissions:** Administrators should be able to assign different access levels (e.g., admin, staff, client) to ensure security and data confidentiality.
- **Document Upload and Sharing:** Users must be able to securely upload, categorize, and share documents with clients and staff. Version control and activity tracking should also be available.
- **Task and Communication Management:** The system should allow users to assign tasks, set deadlines, and communicate through an integrated messaging or notification system.

- **Audit Trail and Reporting:** Client Vault must include reporting tools and maintain a detailed audit log of all activities for compliance and accountability purposes.
- **Seamless User Experience:** The platform must be intuitive, responsive, and easy to navigate, ensuring that users can quickly find information and perform tasks without unnecessary complexity.

Function and Operation of the System

Client Vault will operate as a web application that enables businesses to manage client relationships, securely handle documents, and track all interactions in a single platform. The following is a breakdown of the key system functions:

- **User Registration and Authentication:** Users including administrators, staff, and clients will register for the system using secure methods such as email and password. Strong authentication mechanisms will be implemented, including encrypted password storage and optional two-factor authentication.
- **Client Profile Management:** Businesses can create and manage client profiles, including contact details, associated documents, tasks, and communication history.
- **Document Management:** The platform will allow secure upload, categorization, and retrieval of documents. Access rights will be applied based on user roles, and document updates will be tracked with version history.
- **Task Assignment and Tracking:** Users will be able to create and assign tasks, set due dates, and track progress. Automated reminders and notifications will help ensure timely task completion.
- **Communication and Collaboration:** Client Vault will include internal messaging or notification features to facilitate communication between staff and clients, ensuring streamlined collaboration.

• **Administration and Monitoring:** Administrators will have access to an admin panel to manage users, monitor system activity, generate reports, and ensure compliance with security policies.

3.6 Conceptual Models

In development, conceptual models are essential for understanding the problem domain and creating a visual representation of the system’s structure, components, and functionality.

3.6.1 Data Flow Diagram (DFDs)

A Data flow diagram is a graphical view of how data is processed in a system in term of input and output.

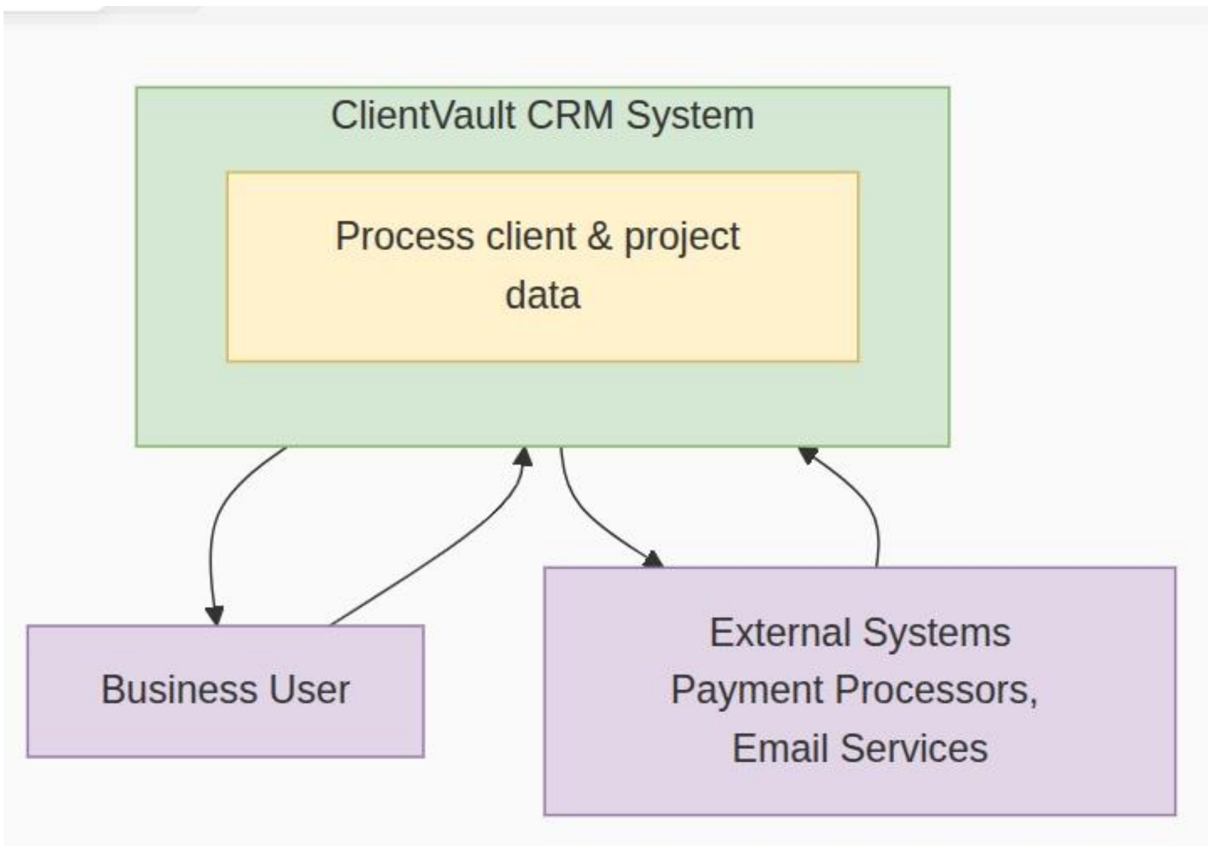
Data Flow Diagram Symbol

Symbol	**Description**
→	**Data Flow:** Data Flow are pipelines through the packets of information flow.
⬭	**Process:** A Process or task performed by the system.

▭	**Entity:** Entity are object of the system. A source or destination data of a system.
═	**Store Data:** A place where data to be stored.

Level 0 DFD:

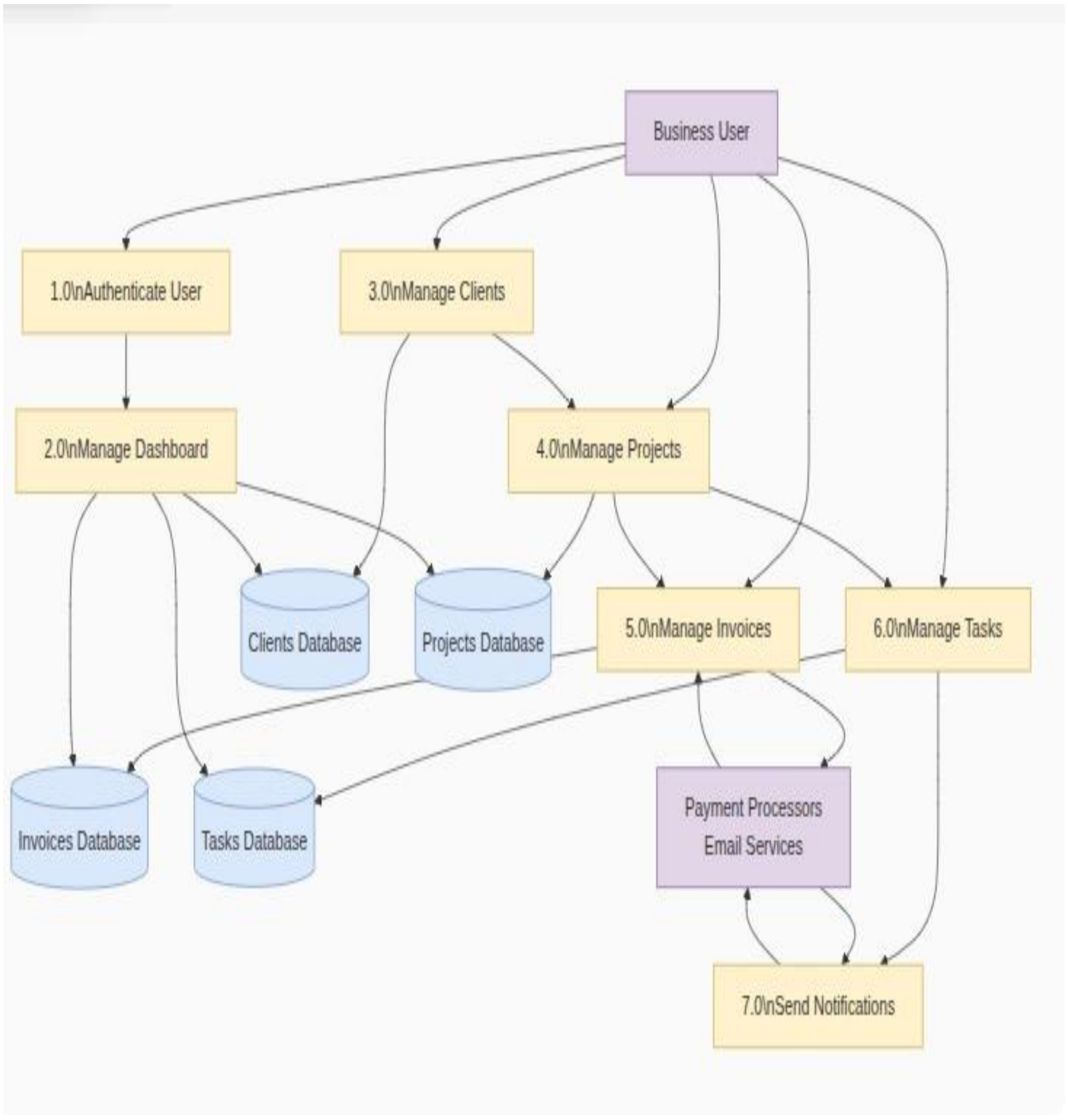
- **Client Vault System:** This represents the central system, which acts as the hub for managing client data, documents, tasks, and communication.
- **Administrator:** Interacts with the system to manage users, assign roles, monitor activities, generate reports, and ensure data security.
- **Staff/User (Employee):** Uses the system to manage client profiles, upload and share documents, assign tasks, and collaborate with clients.
- **Client:** Interacts with the system to securely access shared documents, review tasks or updates, and communicate with staff.
- **Document Repository:** Stores all uploaded documents, ensuring version control, secure access, and retrieval.
- **Audit & Reports:** Maintains logs of activities, tracks system usage, and provides reports for compliance and monitoring


ClientVault CRM System
Process client & project data
Business User
External Systems Payment Processors, Email Services

Level 1 DFD (User Process):

- **User Authentication**: Clients securely log in using credentials or multi-factor authentication to access the vault.
- **Upload Documents**: Clients can upload confidential documents to the vault, ensuring secure storage and encryption.
- **Access & Download Documents**: Clients can browse and download their uploaded or shared documents anytime.
- **Share Documents**: Clients can securely share documents with staff or other authorized users, with permissions control.

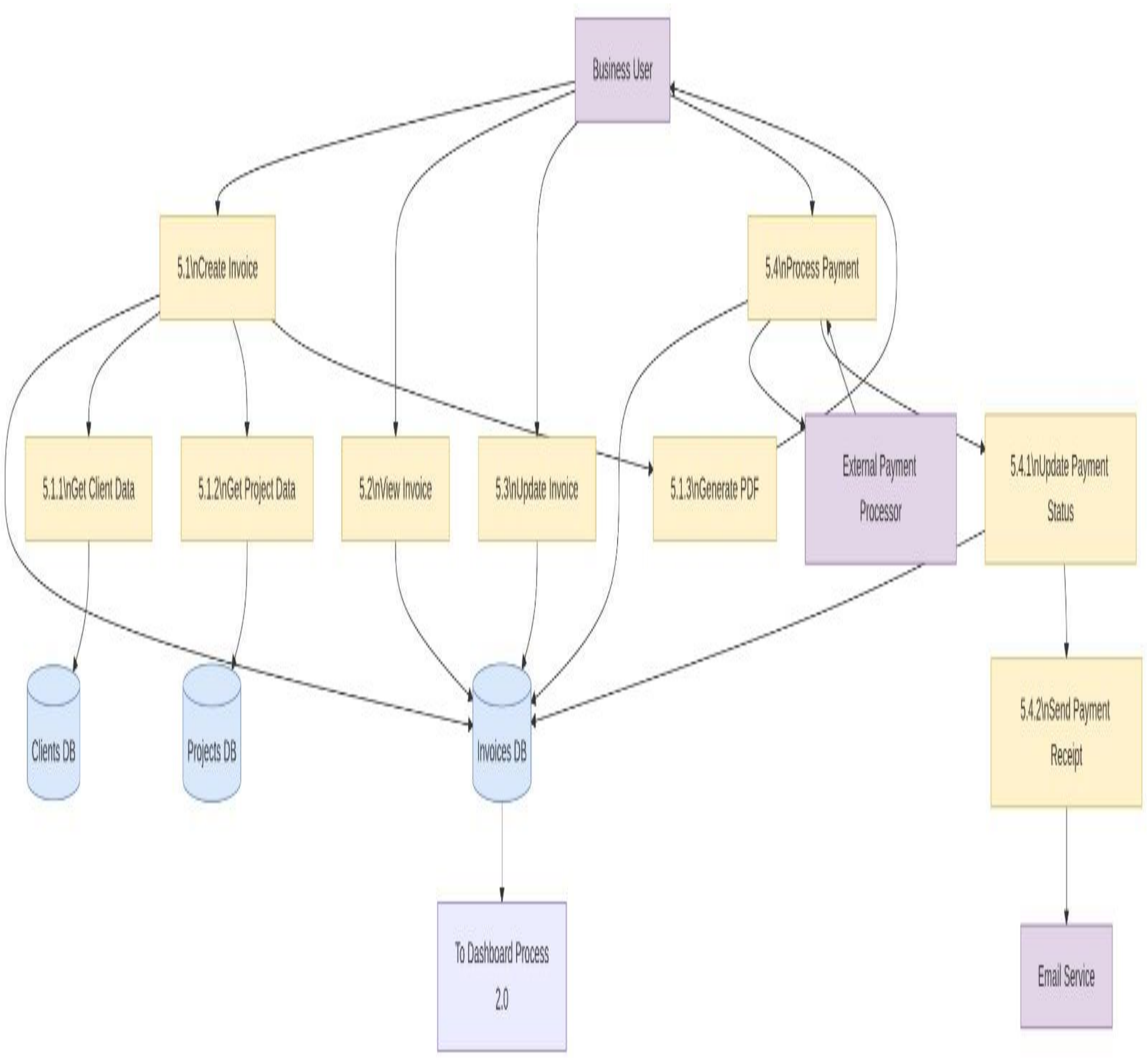
- **Provide Feedback**: Clients can submit feedback about system usability, document access, or support services.



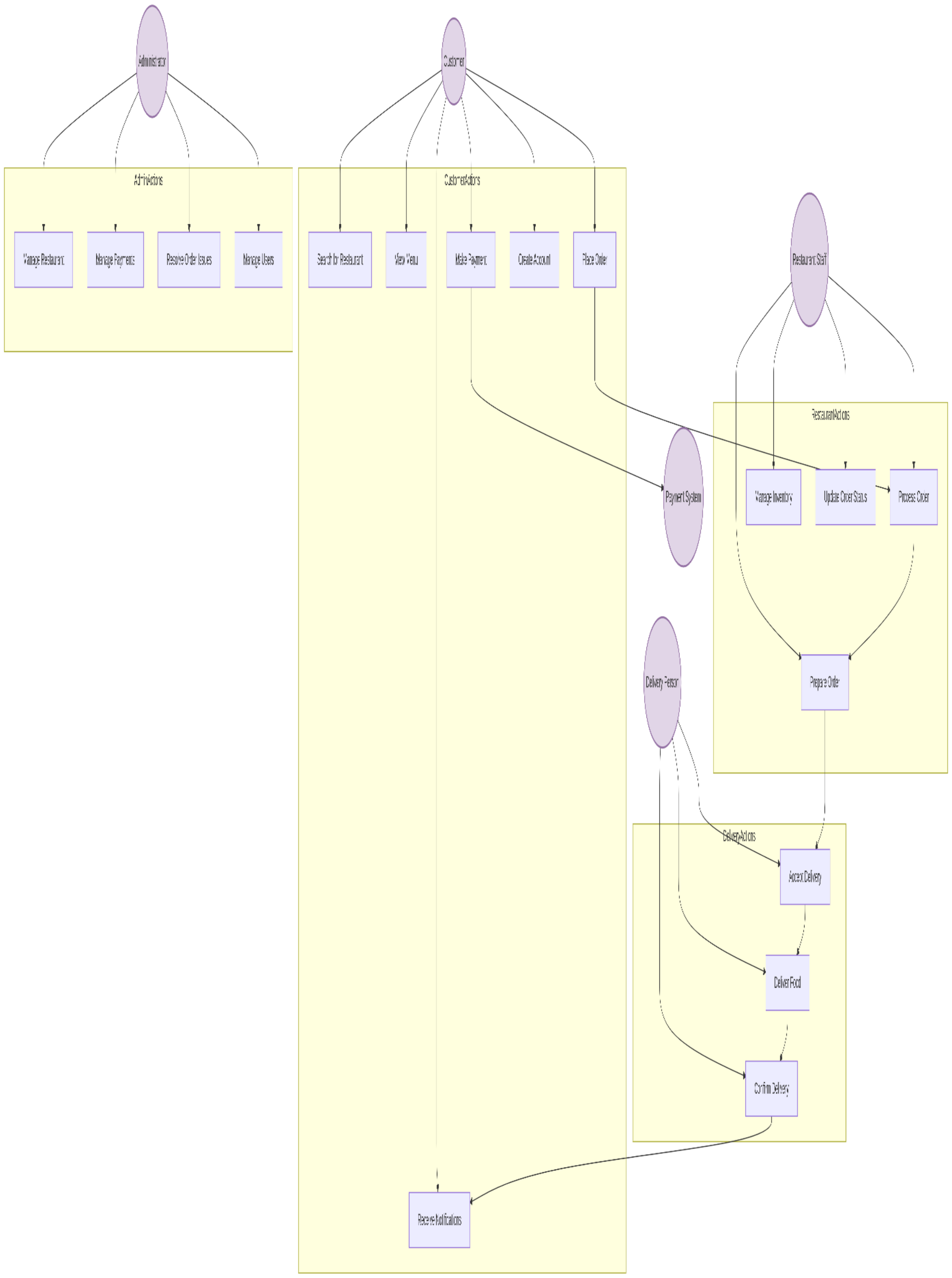
Level 2 DFD (Admin/Staff Process):

- **Manage Client Records**: Admins and staff can create, update, or remove client profiles and associated information.

- **Receive & Review Documents**: Staff can access documents uploaded by clients, review them, and take necessary actions.
- **Update Document Status**: Staff update the status of documents (e.g., Received, Under Review, Approved, Archived*)* to keep clients informed.
- **Maintain Data Security & Permissions**: Admins ensure proper access control, update user permissions, and manage encryption policies to secure sensitive files.



3.6.2 Use case


Administrator
Customer
AdminActions
Manage Restaurant
Manage Payments
Resolve Order Issues
Manage Users
CustomerActions
Search for Restaurant
View Menu
Make Payment
Create Account
Place Order
Restaurant Staff
RestaurantActions
Manage Inventory
Update Order Status
Process Order
Payment System
Prepare Order
Delivery Person
DeliveryActions
Accept Delivery
Deliver Food
Confirm Delivery
Receive Notifications

Chapter 4
System Design

The System Design phase focuses on breaking down the overall problem into smaller, manageable modules and detailing the desired features and operations for each part of the system.

4.1 Basic Module

a. User Management Module

This module manages all aspects of client and staff accounts. Features include:

- Secure user registration and login.
- Profile management (edit personal details, contact information, organization details).
- Role-based access control (differentiating between clients, staff, and administrators).
- Authentication and authorization to ensure data security.

---

b. Document Management Module

This module enables clients and staff to manage documents securely. Features include:

- Secure upload and download of documents.
- Categorization and tagging of documents for easy retrieval.
- Version control to track updates and revisions.
- Real-time status tracking of documents.

---

c. Access & Permission Control Module

This module ensures that documents are accessed only by authorized users. Features include:

- Role-based access to files and folders.
- Granular permission settings (view, edit, share, download).
- Audit trails to monitor access history.
- Secure sharing with expiration and access revocation options.

---

d. Communication & Notification Module

This module provides seamless communication between clients and staff. Features include:

- In-app secure messaging.
- Automated email and SMS notifications for updates.
- Alerts for document uploads, approvals, or requests.
- Task reminders and deadline notifications.

---

e. Payment & Subscription Module

This module handles subscription and billing processes. Features include:

- Integration with multiple payment gateways (credit/debit cards, digital wallets).
- Secure encryption of financial data.
- Invoice and receipt generation for transactions.
- Subscription management (upgrades, downgrades, renewals).

---

f. Security & Compliance Module

This module ensures the platform adheres to industry security and compliance standards. Features include:

- End-to-end encryption of stored and transferred data.
- Multi-factor authentication for account access.
- Compliance with data protection regulations (e.g., GDPR).
- Regular system security monitoring and threat detection.

---

g. Admin Management Module

This module gives administrators full control over the system. Features include:

- User management (add, update, and delete accounts).
- System monitoring for performance, errors, or unusual activities.
- Handling client queries and disputes.
- Generating analytics and reports on system usage.

Integration of Modules

Once each module has been fully developed and tested, they will be integrated into a unified system. The integration will follow a **modular and incremental approach**, ensuring that individual components work seamlessly together.

4.2 Data Design

Data Design is essential to ensuring that the Client Vault system can efficiently organize, manage, and manipulate data. The design focuses on the structure of the database, how the data is stored and retrieved, and the enforcement of integrity through constraints and validation.

4.2.1 Schema Design

The project uses **MongoDB**, a NoSQL database, which allows for flexible and scalable schema design suitable for document storage, access control, and secure transactions.

1. User Schema

Field Name	**Data Type**	**Description**	**Constraints**
_id	ObjectId	Unique user ID	Primary Key
name	String	User’s full name	Not Null
email	String	User’s email address	Not Null, Unique
password	String(hashed)	Securely hashed password	Not Null
role	String	Defines user role (client, admin, auditor)	Not Null
createdAt	Date	Account creation date	Default: current timestamp
lastLogin	Date	Last login activity	Optional
subscriptionPlan	String	Type of subscription (Free, Premium, etc.)	Optional

2. Document Schema

Field Name	**Data Type**	**Description**	**Constraints**
_id	ObjectId	Unique document ID	Primary Key

ownerId	ObjectId	Reference to the user who uploaded	Foreign Key → User._id
fileName	String	Name of the document	Not Null
fileType	String	Document type (PDF, DOCX, XLSX, etc.)	Not Null
fileSize	Number	Size of the file in MB/KB	Not Null
uploadDate	Date	Date when document was uploaded	Default: current timestamp
versionHistory	Array<Object>	Stores past versions of the file	Optional
accessLevel	String	Defines access (private, shared, public)	Not Null
encryptionKey	String	Encrypted key for secure file storage	Not Null

3. Access Control Schema

Field Name	**Data Type**	**Description**	**Constraints**
_id	ObjectId	Unique access record ID	Primary Key
documentId	ObjectId	Reference to document	Foreign Key → Document._id
userId	ObjectId	Reference to user with access	Foreign Key → User._id

permissionType	String	Permission level (view, edit, share)	Not Null
grantedAt	Date	Date when access was granted	Default: current timestamp
grantedBy	ObjectId	Reference to admin/user who granted access	Foreign Key → User._id

4. Audit Logs Schema

Field Name	**Data Type**	**Description**	**Constraints**
_id	ObjectId	Unique log entry ID	Primary Key
userId	ObjectId	Reference to the user who performed action	Foreign Key → User._id
actionType	String	Type of action (upload, download, delete, share, login, logout)	Not Null
documentId	ObjectId	Reference to affected document (if any)	Foreign Key → Document._id, Optional
timestamp	Date	Time when the action occurred	Default: current timestamp

ipAddress	String	IP address of the user	Optional
deviceInfo	String	Device/browser details	Optional
status	String	Action status (success, failed, unauthorized)	Not Null

5. Backup & Recovery Schema

Field Name	**Data Type**	**Description**	**Constraints**
_id	ObjectId	Unique backup entry ID	Primary Key
documentId	ObjectId	Reference to the original document	Foreign Key → Document._id
backupVersion	Number	Version number of the backup	Auto-increment per document
backupFileUrl	String	Encrypted backup file storage path/URL	Not Null
createdAt	Date	Timestamp when backup was created	Default: current timestamp
createdBy	ObjectId	Reference to user/admin who triggered backup	Foreign Key → User._id

recoveryStatus	String	Status of backup (active, restored, expired)	Default: active
expiryDate	Date	Expiration date for backup (if retention policy applied)	Optional
checksum	String	Hash value for file integrity verification	Not Null

6. Notification Schema

This schema manages alerts and reminders for users regarding file uploads, access requests, security warnings, and backups.

Field Name	**Data Type**	**Description**	**Constraints**
_id	ObjectId	Unique notification ID	Primary Key
userId	ObjectId	Reference to user receiving the notification	Foreign Key → User._id
message	String	Notification content/message	Not Null
type	String	Type of notification (info, warning, alert, reminder)	Not Null
isRead	Boolean	Marks if notification has been read	Default: false

createdAt	Date	Timestamp when notification was generated	Default: current timestamp

7. Session Schema

This schema ensures secure login sessions and activity tracking for auditing and monitoring

Field Name	Data Type	Description	Constraints
_id	ObjectId	Unique session ID	Primary Key
userId	ObjectId	Reference to logged-in user	Foreign Key → User._id
token	String	Encrypted session token	Not Null, Unique
deviceInfo	String	Device/browser details of the session	Optional
ipAddress	String	IP address of the user during session	Not Null
createdAt	Date	Session start time	Default: current timestamp
expiresAt	Date	Session expiry time	Not Null
isActive	Boolean	Marks if session is currently active	Default: true

4.2.2 Data Integrity and Constraints

To maintain data integrity across the Client Vault system, several constraints and validation checks are applied to ensure correctness, consistency, and security of stored information.

---

a. Primary Key Constraints

Each collection in the database (such as Users, Documents, Access Control, Sessions) has a **Primary Key** that uniquely identifies each record, preventing duplication. For instance:

- **User Collection**: The `_id` (ObjectId) serves as the primary key, ensuring each user has a unique identifier.
- **Document Collection**: The `_id` uniquely identifies each document, preventing duplication of file entries.
- **Session Collection**: Each session record is uniquely identified by `_id`, ensuring that no two sessions share the same identifier.

---

b. Foreign Key Constraints

Foreign key–like relationships (using ObjectId references in MongoDB) maintain referential integrity between collections. This ensures that references to other collections remain valid. Examples:

- In the **Document Collection**, the `ownerId` references a valid user in the **User Collection**, ensuring a document cannot exist without an associated owner.
- In the **AccessControl Collection**, `userId` and `documentId` reference valid records in **User** and **Document**, ensuring access permissions are tied to existing users and files.

- In the **AuditLogs Collection**, `performedBy` links to a valid user, ensuring accountability of every action.

---

c. Not Null Constraints

Certain critical fields are marked **Not Null** to prevent incomplete or invalid entries:

- **User Collection**: `email`, `password`, and `role` must be provided during registration.
- **Document Collection**: `title`, `ownerId`, and `filePath` must always be specified when uploading a file.
- **AccessControl Collection**: `permissions` and `userId` cannot be null when granting access rights.

---

d. Unique Constraints

Unique constraints are enforced to avoid duplication where not allowed:

- **User Email**: Each user must register with a unique email address, preventing multiple accounts with the same email.
- **Session Token**: Each session's `token` must be unique, ensuring session hijacking is prevented.
- **Document Versioning (if applied)**: A unique combination of `documentId + versionNumber` ensures no two records overlap for the same document version.

---

e. Validation Rules

To complement database-level constraints, **application-level validation rules** are enforced before data is committed:

- **Email Validation**: Ensures a valid email format during registration.
- **Password Strength**: Requires passwords to meet complexity rules (length, uppercase, special characters) for stronger security.
- **File Validation**: Validates file size, type, and metadata before upload to prevent malicious files.
- **Access Permissions**: Verifies that permission levels (read, write, admin) conform to predefined rules before granting access.
- **Audit Logs Validation**: Ensures every action logged contains a valid timestamp and reference to the user who performed it.

4.3 Procedural Design

Procedural design in the Client Vault system serves as the backbone of how the application's functionalities are structured, executed, and maintained. It systematically outlines the steps, workflows, and procedures required to perform the core operations of the system such as user authentication, document upload, secure storage, controlled access, sharing, and audit logging. The primary goal of this design is to break down high-level system requirements into smaller, logical, and executable processes that can be implemented through code.

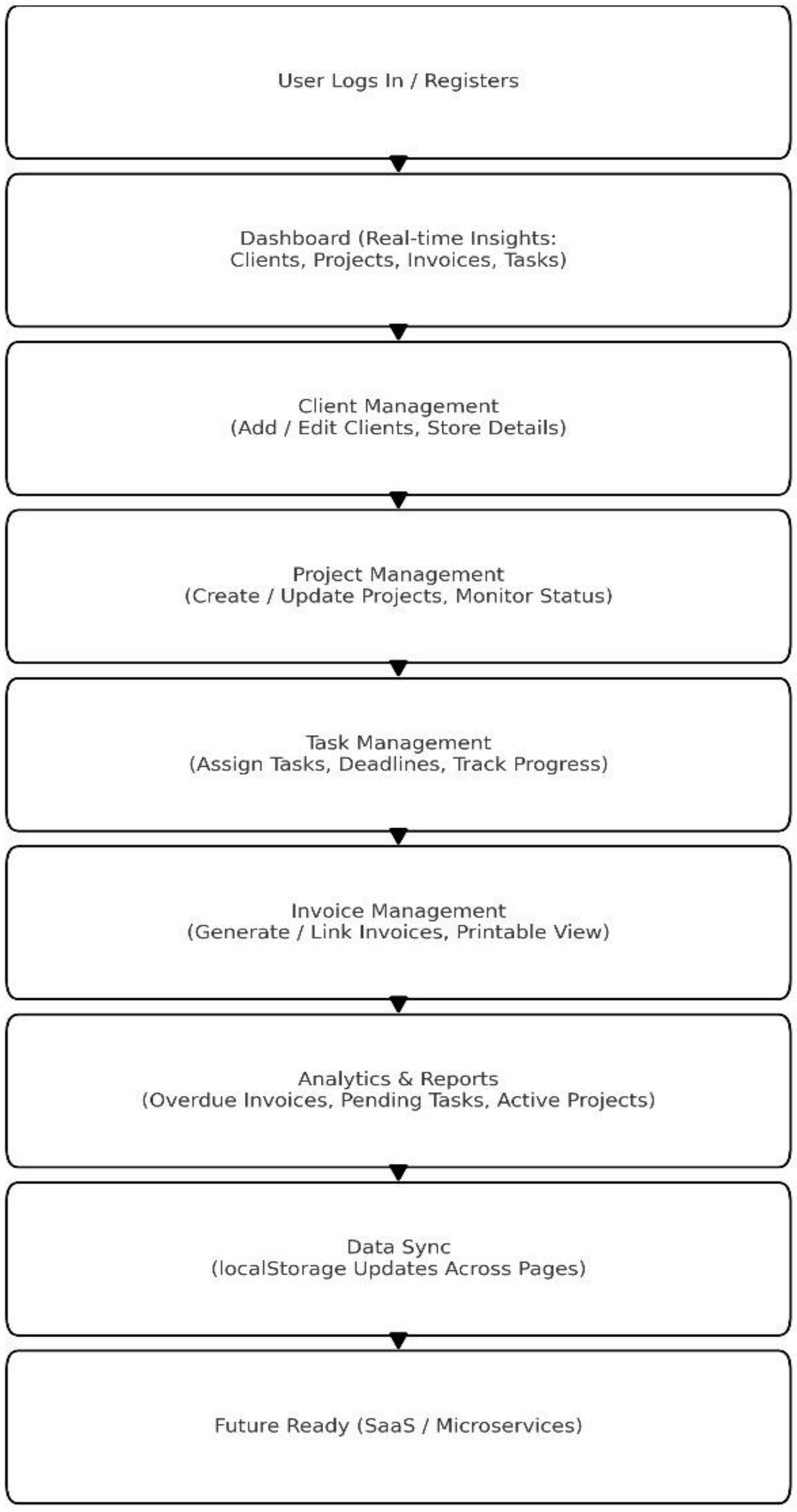
In the context of Client Vault, procedural design ensures that every interaction — from a user logging into the platform, to uploading a sensitive client document, to generating compliance audit reports — follows a clear, repeatable, and secure flow. Logical diagrams are used to visually represent these flows, making it easier to understand system behavior and identify potential bottlenecks or risks. Similarly, data structures define how entities like users, documents, and access permissions are organized in memory, ensuring optimal performance and maintainability.

Additionally, algorithm design specifies the step-by-step logic for key processes such as authentication, document encryption, access validation, and audit log generation. This guarantees that every function is executed consistently, securely, and in alignment with system objectives. By clearly defining these processes, procedural design provides a roadmap that reduces ambiguity during development, ensures data integrity, and supports future scalability.

Furthermore, procedural design enhances **system** reliability by promoting modular development, where each process is independently designed, tested, and then integrated into the larger system. This modularity not only reduces development errors but also simplifies debugging and maintenance in the long run. By incorporating error-handling mechanisms, validation rules, and exception management at the procedural level, the system can gracefully recover from unexpected issues while preserving critical client data. Ultimately, procedural design ensures that Client Vault operates with efficiency, accuracy, and security, meeting the expectations of both end users and regulatory requirement.

4.3.1 Logic Diagrams

Control Flow Chart (Order Placement Process)


User Logs In / Registers
Dashboard (Real-time Insights: Clients, Projects, Invoices, Tasks)
Client Management (Add / Edit Clients, Store Details)
Project Management (Create / Update Projects, Monitor Status)
Task Management (Assign Tasks, Deadlines, Track Progress)
Invoice Management (Generate / Link Invoices, Printable View)
Analytics & Reports (Overdue Invoices, Pending Tasks, Active Projects)
Data Sync (localStorage Updates Across Pages)
Future Ready (SaaS / Microservices)

4.3.2 Data Structures

Data Structures form the backbone of the Client Vault application, defining how sensitive client documents, users, and permissions are organized, stored, and manipulated. In Client Vault, entities such as users, documents, and access permissions are central to the system's functionality.

1.Client Schema

```
{
    id: 1,
    name: String,
    email: String,
    phone: String,
    company: String,
    address: String,
    notes: String,
    createdAt: Date
}
```

2. Project schema

```
{
  id: String,
  clientId: String,
  title: String,
  description: String,
  status: String,
  startDate: Date,
  dueDate: Date,
  budget: Number,
  createdAt: Date
}
```

3. Invoice schema

```
{
  id: String,
  clientId: String,
  projectId: String,
  invoiceNumber: String,
  dateIssued: Date,
  dueDate: Date,
  status: String,
  items: [
    {
      description: String,
      quantity: Number,
      rate: Number,
      amount: Number
    }
  ],
  total: Number,
  notes: String
}
```

4. Task schema

```
{
  id: String,
  projectId: String,
  title: String,
  description: String,
  assignedTo: String,
  status: String,
  dueDate: Date,
  createdAt: Date
}
```

5.Dashboard Schema

```
{
  totalClients: Number,
  activeProjects: Number,
  overdueInvoices: Number,
  tasksDueToday: Number,
  revenueThisMonth: Number
}
```

4.3.3 Algorithms Design

1. **User Login and Authentication**

   Input Data:

- User email and password

  Output Data:

- Authentication success/failure
- User role and dashboard access

Logic of Processes:

1. Retrieve email and password from user input.
2. Verify if the email exists in the database.
3. Compare the entered password with the hashed password in the database.
4. If valid, generate an authentication token and fetch the user role.
5. Redirect user to their respective dashboard (Admin, Client, or Manager).
6. If invalid, return an authentication error.

---

2. **Document Upload and Storage**

   Input Data:

- Document file (PDF, Word, Excel, Image, etc.)
- Metadata (document name, type, tags, client ID, uploader ID)

  Output Data:

- Upload confirmation
- Document ID generated

Logic of Processes:

1. Retrieve document file and metadata from user input.

2. Validate file format and size.
3. Generate a unique Document ID.
4. Encrypt the document before storage for security.
5. Store the document in the secure storage system.
6. Save metadata (with uploader ID and timestamp) in the database.
7. Return confirmation with Document ID.

---

3. **Document Access Request**

   Input Data:

- User ID
- Document ID

  Output Data:

- Access granted or denied
- Document retrieved (if granted)

Logic of Processes:

1. Retrieve User ID and Document ID from request.
2. Verify if the user has permission to access the document (via role/ACL).
3. If access is valid, decrypt and fetch the document.
4. Log the access attempt in access logs (user, timestamp, document ID).
5. Return the document to the user.
6. If invalid, deny access and notify the user.

---

4. **Sharing Document with Client/Team**

   Input Data:

- Document ID

- Recipient User ID(s)
- Access type (View/Edit/Download)

  Output Data:
- Share confirmation
- Updated permissions

Logic of Processes:

1. Retrieve document and recipient IDs.
2. Verify the current user’s permission to share the document.
3. Update access control list (ACL) with new recipient and permissions.
4. Notify the recipient(s) about document sharing.
5. Return share confirmation to the sender.

---

5. **Audit Log Generation**

   Input Data:

- User actions (upload, view, share delete, etc.)

  Output Data:
- Structured audit log entries
- Report generation

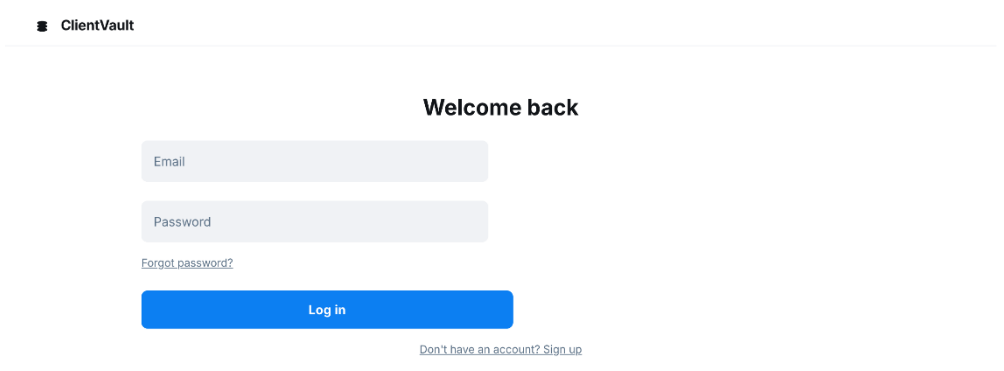
Logic of Processes:

1. Capture user action along with User ID, Document ID, timestamp, and action type.
2. Store this information in the audit log database.
3. Generate audit reports on demand (daily, weekly, monthly).
4. Allow admins to query logs for security or compliance checks.
5. Return audit report in requested format (PDF/Excel).

.

4.4 User Interface Design

Login page



ClientVault

Welcome back

Email

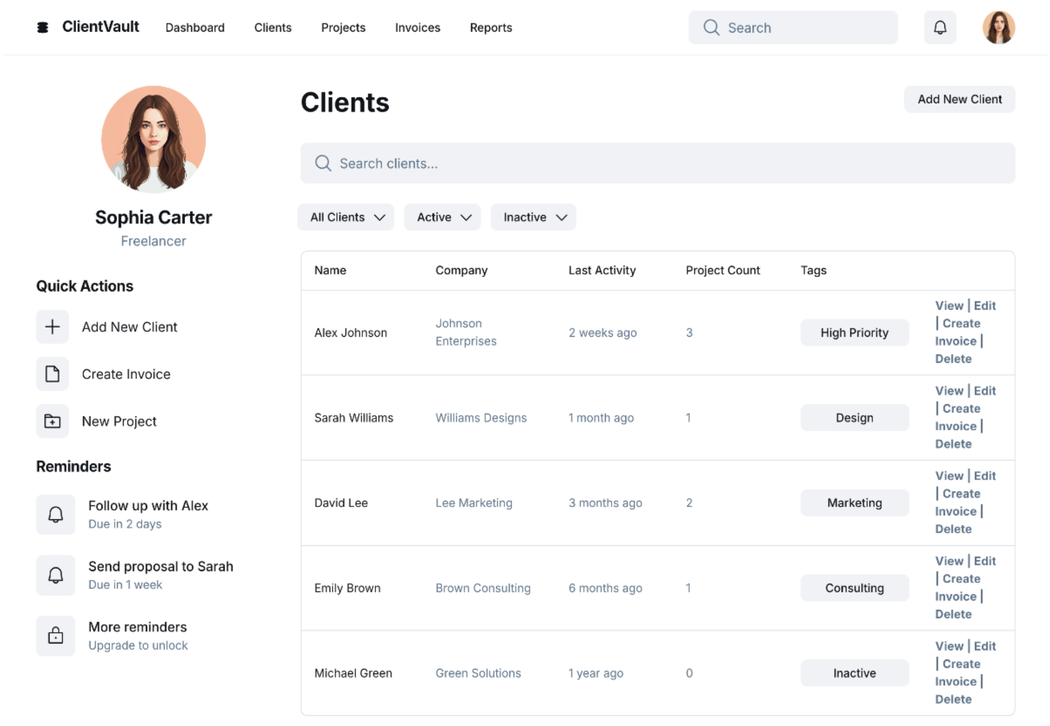
Password

Forgot password?

Log in

Don't have an account? Sign up

Clients page



ClientVault Dashboard Clients Projects Invoices Reports Search

Sophia Carter
Freelancer

Quick Actions
- Add New Client
- Create Invoice
- New Project

Reminders
- Follow up with Alex
  Due in 2 days
- Send proposal to Sarah
  Due in 1 week
- More reminders
  Upgrade to unlock

Clients

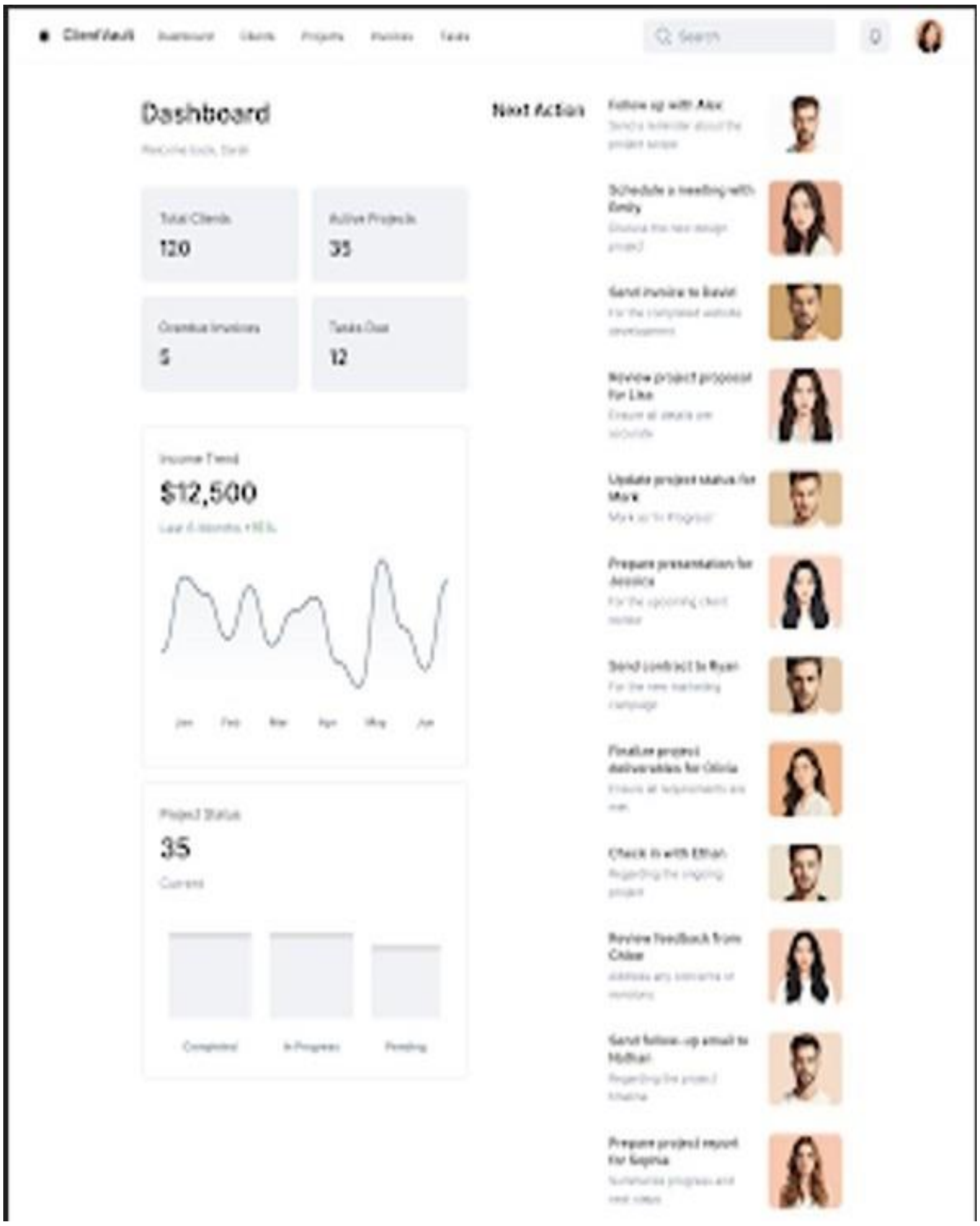
Add New Client

Search clients...

All Clients | Active | Inactive

Name	Company	Last Activity	Project Count	Tags	
Alex Johnson	Johnson Enterprises	2 weeks ago	3	High Priority	View | Edit | Create Invoice | Delete
Sarah Williams	Williams Designs	1 month ago	1	Design	View | Edit | Create Invoice | Delete
David Lee	Lee Marketing	3 months ago	2	Marketing	View | Edit | Create Invoice | Delete
Emily Brown	Brown Consulting	6 months ago	1	Consulting	View | Edit | Create Invoice | Delete
Michael Green	Green Solutions	1 year ago	0	Inactive	View | Edit | Create Invoice | Delete

Dashboard page



ClientVault Dashboard Clients Projects Invoices Tasks Search

Dashboard

Welcome back, Sarah

Total Clients 120	Active Projects 35
Overdue Invoices 5	Tasks Due 12

Income Trend

$12,500

Last 6 months +15%

Project Status

35

Current

Next Action

Follow up with Alex
Send a reminder about the project scope

Schedule a meeting with Emily
Discuss the new design project

Send invoice to David
For the completed website development

Review project proposal for Lisa
Ensure all details are accurate

Update project status for Mark
Mark as 'In Progress'

Prepare presentation for Jessica
For the upcoming client review

Send contract to Ryan
For the new marketing campaign

Finalize project deliverables for Olivia
Ensure all requirements are met

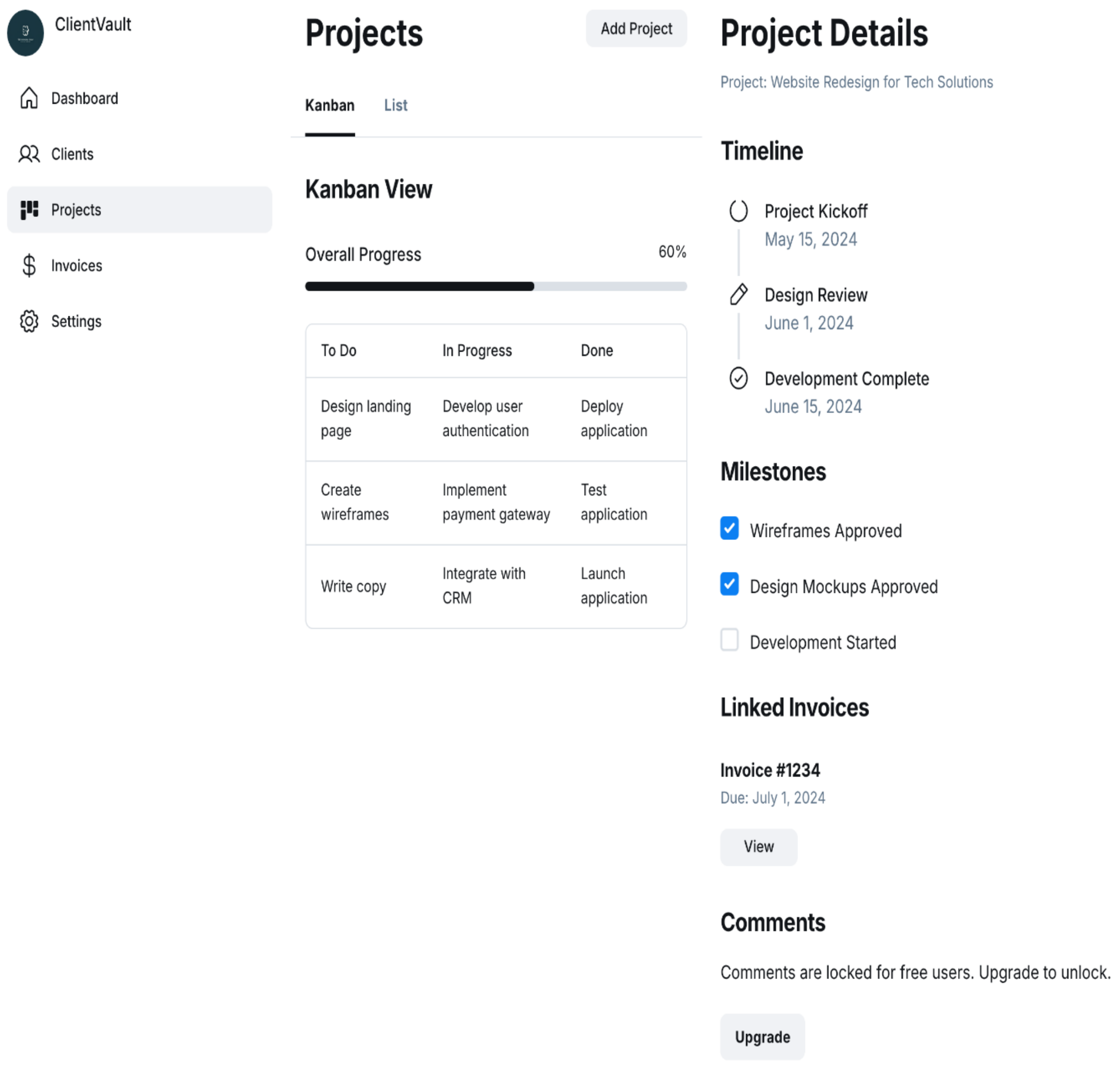
Check in with Ethan
Regarding the ongoing project

Review feedback from Chloe
Address any concerns or revisions

Send follow-up email to Nathan
Regarding the project timeline

Prepare project report for Sophia
Summarize progress and next steps

Project Details



Projects

Add Project

Kanban   List

Kanban View

Overall Progress 60%

To Do	In Progress	Done
Design landing page	Develop user authentication	Deploy application
Create wireframes	Implement payment gateway	Test application
Write copy	Integrate with CRM	Launch application

Project Details

Project: Website Redesign for Tech Solutions

Timeline

Project Kickoff
May 15, 2024

Design Review
June 1, 2024

Development Complete
June 15, 2024

Milestones

- [x] Wireframes Approved
- [x] Design Mockups Approved
- [ ] Development Started

Linked Invoices

Invoice #1234
Due: July 1, 2024

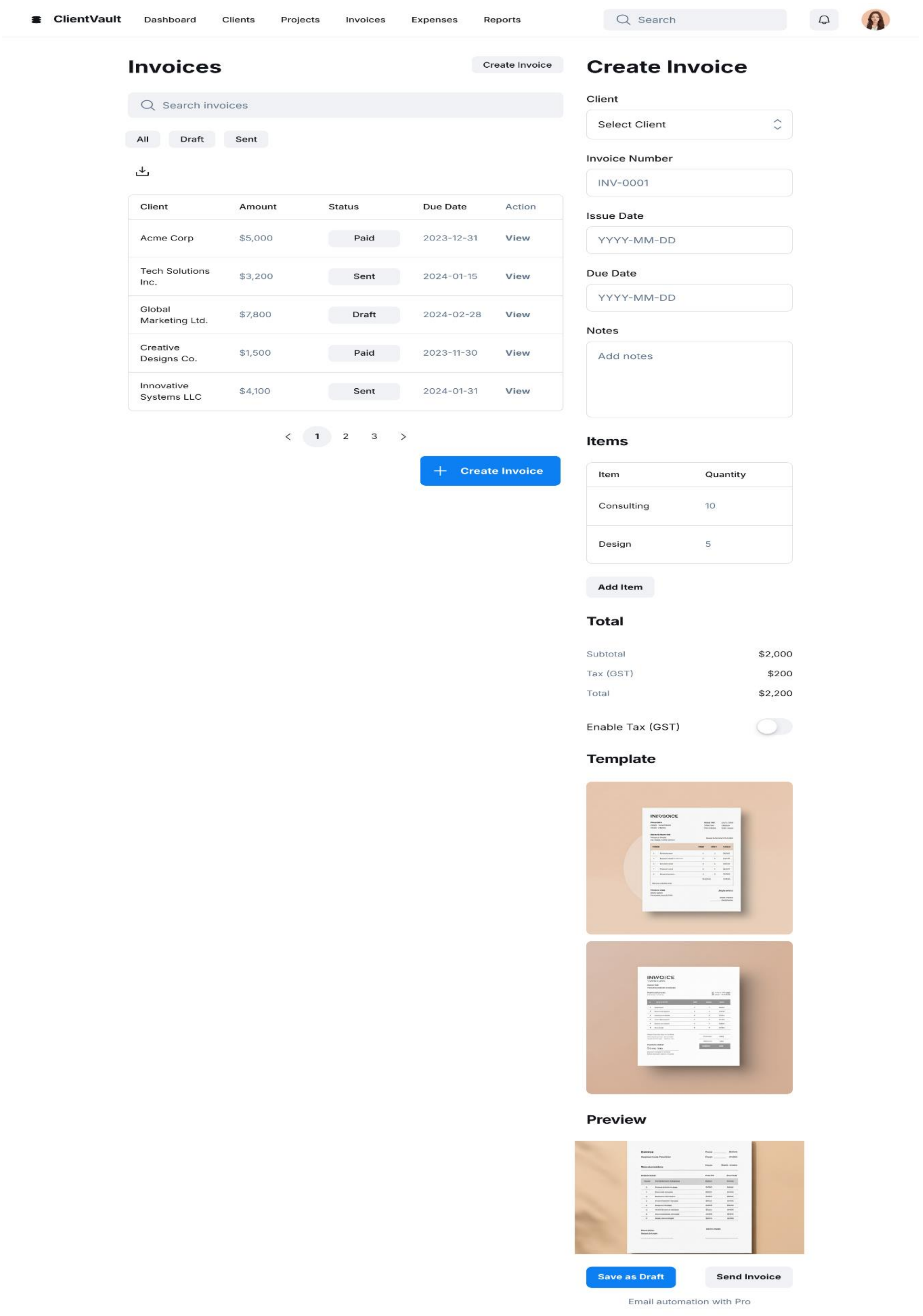
View

Comments

Comments are locked for free users. Upgrade to unlock.

Upgrade

Invoice Page



Invoices

Client	Amount	Status	Due Date	Action
Acme Corp	$5,000	Paid	2023-12-31	View
Tech Solutions Inc.	$3,200	Sent	2024-01-15	View
Global Marketing Ltd.	$7,800	Draft	2024-02-28	View
Creative Designs Co.	$1,500	Paid	2023-11-30	View
Innovative Systems LLC	$4,100	Sent	2024-01-31	View

Create Invoice

Client: Select Client

Invoice Number: INV-0001

Issue Date: YYYY-MM-DD

Due Date: YYYY-MM-DD

Notes: Add notes

Items

Item	Quantity
Consulting	10
Design	5

Total

Subtotal	$2,000
Tax (GST)	$200
Total	$2,200

Enable Tax (GST)

Template

Preview

4.5 Security Issues

In Client Vault, security and real-time operations are critical due to the sensitive nature of client, project, invoice, and task data, as well as multi-user interactions between admins, employees, and clients.

Real-Time Considerations

1. **Real-Time Data Handling**
   - **Challenge:** The system must provide real-time updates across clients, projects, invoices, and tasks to ensure accurate tracking and management.
   - **Solution:** Implement efficient, low-latency communication using WebSockets**, Socket.IO**, or real-time databases (like **Firebase**) to synchronize updates across all users instantaneously.
2. **High Availability and Scalability**
   - **Challenge:** The system must handle multiple users simultaneously, especially during peak usage (e.g., multiple projects and invoices being updated concurrently).
   - **Solution:** Design the infrastructure for horizontal scalability using cloud services (AWS, Azure, or MongoDB Atlas) to ensure dynamic handling of traffic and prevent downtime.
3. **Real-Time Updates on Projects and Tasks**
   - **Challenge:** Users expect immediate feedback on project status, task completion, and invoice generation. Delays can disrupt workflow and reporting.
   - **Solution:** Implement asynchronous communication**,** push notifications, and live dashboards to provide instant updates on changes.

Security Issues

1. **Data Privacy and Security**
   - **Issue:** ClientVault stores sensitive information such as client details, project files, and invoice data, making it a target for data breaches.
   - **Mitigation:** Encrypt all data in transit (**HTTPS/SSL**) and at rest. Restrict sensitive data access only to authorized users. Implement database-level access control.
2. **Authentication and Authorization**
   - **Issue:** Weak or poorly implemented authentication can allow unauthorized account access.
   - **Mitigation:** Enforce **multi-factor authentication (MFA)**, **OAuth 2.0** for third-party logins, and secure password storage with **bcrypt** hashing and salting. Use role-based access control (RBAC) to manage user permissions.
3. **Invoice and Payment Security**
   - **Issue:** If ClientVault handles online payments or stores financial records, these become targets for fraud.
   - **Mitigation:** Integrate **PCI-DSS compliant payment gateways**, tokenize sensitive payment data, and secure all transactions via **SSL/TLS**. Avoid storing raw credit card information.
4. **Real-Time Data Access Security**
   - **Issue:** Unauthorized access to live project, task, or client data could compromise confidentiality.
   - **Mitigation:** Encrypt all sensitive real-time data and limit visibility to only users involved with specific clients, projects, or tasks. Implement session management and activity logging.
5. **API Security**
   - **Issue:** ClientVault's APIs could be targeted for attacks such as DDoS, injection, or man-in-the-middle attacks.

- **Mitigation:** Protect APIs using **OAuth 2.0**, **input validation**, **rate limiting**, and monitor logs for suspicious activity. Apply secure coding practices to prevent injection attacks.

Security Policy Plans and Architecture

1. **Encryption Policy**
    - **Plan:** Encrypt all sensitive data, including client details, project information, invoice data, and communication between users and the server. Use industry-standard encryption protocols such as AE**S** for data at rest and TLS/HTTPS for data in transit.
    - **Architecture:** All interactions between the client, server, and third-party services (e.g., cloud storage or email notifications) will occur over secure, encrypted connections. Passwords will be hashed and salted using strong algorithms (e.g., bcrypt) before storage.
2. **Access Control Policy**
    - **Plan:** Implement Role-Based Access Control (RBAC) to restrict access based on user roles. For example:
        - Admins can manage all clients, projects, invoices, and tasks.
        - Project managers can manage projects, assign tasks, and view associated invoices.
        - Employees can access tasks assigned to them and update project progress.
        - Clients can view only their own project status and invoices.
    - **Architecture:** The system will enforce granular access controls at both the API level and the **UI** level**,** ensuring that each role has specific permissions, preventing unauthorized access to sensitive features and data.

3. **Secure Development Practices**
   - **Plan:** Follow secure coding guidelines by validating and sanitizing all user inputs, avoiding exposure of sensitive data in logs or error messages, and preventing common security vulnerabilities like injection attacks.
   - **Architecture:** Utilize secure frameworks and libraries (e.g., **Zod** for schema validation in TypeScript, or Mongoose validation for MongoDB) to enforce strict data validation. Conduct **regular security audits, code reviews, and penetration testing** to detect and mitigate vulnerabilities proactively.

4.6 Test Cases Design

1. Registration

Test Case No.	**Test Case**	**Pre-condition**	**Expected Result**
1	Check that a user can register on ClientVault by entering valid information such as name, email, password, and role (Admin/Employee/Client).	All registration fields are visible and the registration page is functional.	User is registered successfully and redirected to the login page.
2	Check that the system displays appropriate error messages when a user tries to register with missing or incorrect information.	User is on the registration page and attempts to submit incomplete or invalid details.	Error message like "All fields are required" or "Invalid email format" is displayed.

3	Check that the system detects duplicate email addresses during registration.	Database contains an existing email address for another registered user.	Error message "Email already exists" is displayed.
4	Check that users receive a confirmation email after successful registration.	User has completed registration with a valid email.	A confirmation email is sent to the user's registered email address.
5	Check that passwords are stored securely using encryption in the database.	User has entered a password during registration.	Password is **hashed and salted** (e.g., using bcrypt) and stored securely in the database, ensuring it is not visible in plain text.

2. Login

Test Case No.	**Test Case**	**Pre-condition**	**Expected Result**
1	Check that a registered user can log in using valid email and password.	User has already registered and confirmed their email.	User is successfully logged in and redirected to the dashboard.

2	Check that the system displays an error when the user enters incorrect email or password.	User tries to log in with incorrect credentials.	Error message "Invalid email or password" is displayed.
3	Check that the system provides "Forgot Password" functionality.	User has forgotten their password and clicks the "Forgot Password" link.	User receives an email with instructions to reset their password.
4	Check that social media login (Google, Facebook) works correctly.	Social login APIs are integrated with the application.	User is authenticated using social media credentials and redirected to the dashboard.

3.Dashboard and Navigation

Test Case No.	**Test Case**	**Pre-condition**	**Expected Result**

1	Check that the user is redirected to their respective dashboard after successful login.	User has logged in successfully.	Admins are taken to the admin dashboard, project managers to the project overview page, employees to their assigned tasks, and clients to their project/invoice overview.
2	Check that the dashboard displays a personalized welcome message and account details.	User is logged in and on their dashboard.	A welcome message with the user's name and role is displayed along with relevant account details (e.g., total clients, active projects, pending tasks).
3	Check that project managers can view and search projects by client, status, or deadline.	Project manager is on the dashboard.	User can filter and view projects based on client, project status, or deadline, with results displayed in real-time.
4	Check that employees can view and manage their assigned tasks from the dashboard.	Employee is on the dashboard.	The employee can see their tasks, update progress, mark completion,

			and receive real-time status updates.
5	Check that clients can view their projects and invoices from the dashboard.	Client is logged in and on the dashboard.	Client can see their associated projects, task progress, and invoices, with the ability to download invoices if required.

4. **Project Management**

Test Case No.	**Test Case**	**Pre-condition**	**Expected Result**
1	Check that a project manager can update project details (name, description, deadline, status).	Project manager is logged in and has access to the project.	Project details are updated successfully and changes are reflected in real-time on the dashboard.
2	Check that a project manager can add, update, or delete tasks within a project.	Project exists and task management system is active.	Tasks are successfully added, updated, or deleted with real-time updates visible to assigned employees.

3	Check that notifications are sent when task deadlines are approaching or overdue.	Tasks have defined deadlines and responsible employees are assigned.	Notification alerts are sent to assigned employees and project managers when deadlines are near or overdue.
4	Check that project status changes are reflected across client and employee dashboards.	Project status is updated by the project manager.	Clients and employees see the updated project status immediately on their dashboards.
5	Check that project history/log is maintained for all updates and changes.	Project has previous updates recorded.	All changes to projects and tasks are logged with timestamps, user details, and visible in project history.

5. Invoice Management and Payment

Test Case No.	**Test Case**	**Pre-condition**	**Expected Result**
1	Check that project managers can generate invoices for completed projects or tasks.	Project has been completed and all billing details are entered.	Invoice is generated successfully with correct client details, project details, and total amount.

2	Check that invoices can be customized with additional notes or discounts.	Project manager is generating an invoice.	Invoice reflects custom notes, discounts, or special instructions as entered.
3	Check that clients can view, download, or print invoices from their dashboard.	Invoice has been generated and client is logged in.	Client can view the invoice, download it as PDF, or print it successfully.
4	Check that invoices reflect scheduled or milestone-based billing.	Project has scheduled milestones or partial billing setup.	Invoice accurately reflects milestone or scheduled billing amounts.
5	Check that payment status is updated automatically after client payment.	Client makes payment via integrated payment gateway.	Invoice status updates to "Paid" or "Pending" automatically and is visible to both client and project manager.

6.Task Management and Real-Time Updates

Test Case No.	Test Case	Pre-condition	Expected Result
1	Check that employees receive real-time updates on assigned tasks.	Task has been assigned to an employee.	Task updates (new assignment, status changes, deadlines) are displayed to the employee in real-time on their dashboard.

2	Check that employees can update task status in real-time.	Employee is logged in and working on an assigned task.	Task status (e.g., "In Progress", "Completed") is updated immediately and visible to the project manager and client if applicable.
3	Check that clients and project managers can track task progress in real-time.	Tasks are updated by employees.	Task progress is reflected in real-time on dashboards of project managers and clients.
4	Check that notifications are sent for task updates or approaching deadlines.	Task has defined deadlines or critical updates.	Notifications are sent automatically to assigned employees and project managers when tasks are updated or deadlines approach.
5	Check that task history/logs are maintained for all updates.	Tasks have previous updates recorded.	All changes to tasks are logged with timestamps, user details, and displayed in task history for auditing and reference.

7.Payment and Invoice Processing

Test Case No.	Test Case	Pre-condition	Expected Result
1	Check that clients can make payments for invoices using credit card, debit card, or digital wallets.	Payment gateway integrated and functional.	Payment processed successfully; invoice status updates to "Paid"; confirmation displayed.

2	Check that payments fail when invalid or expired card details are entered.	Client on payment page with invalid card details.	Error message displayed; payment does not proceed.
3	Check that partial or milestone payments are handled correctly.	Invoice has partial/milestone payments setup.	Payment processed correctly; invoice reflects updated balance.
4	Check that payment transactions are securely logged.	Payment completed.	Transactions logged securely without storing sensitive card information.
5	Check that failed transactions prompt retry or alternative payment options.	Payment attempt fails.	User prompted to retry or choose alternative method.