An-Najah National University

Faculty of Engineering & Information Technology

Computer Engineering Department

Fall 2025/2026

SmartOps

Integrated Operations Management Platform for SMEs

Presented By:

Yousef Hanna (12112193)

Mahmoud Asmar (12114182)

Supervised by: Dr. Amjad AbuHassan

Presented in partial fulfilment of the requirements for the Bachelor degree in Computer Engineering.

Dedication

To our families, whose love, patience, and unwavering support carried us through every challenge of this journey.

To our supervisor, Dr. Amjad AbuHassan, for his guidance and belief in our work.

And to every SME striving to work smarter — this platform was built with you in mind.

Acknowledgment

We would like to extend our sincere gratitude to our supervisor, Dr. Amjad AbuHassan, whose guidance, support, and valuable insights were instrumental to the success of this project. His expertise in software engineering and system design helped us navigate technical challenges and maintain focus on delivering a platform that truly serves the operational needs of modern businesses.

Our appreciation also goes to the Department of Computer Engineering at An-Najah National University, whose comprehensive curriculum and supportive learning environment provided us with the technical foundation necessary to undertake this project.

To our families, we offer our deepest thanks for their unwavering support, patience, and encouragement throughout this journey. Your belief in our abilities gave us the strength to persevere through challenges and late nights of development.

We are also grateful to our colleagues and friends who provided valuable feedback, testing support, and constructive suggestions that helped shape SmartOps into a more robust and user-friendly platform.

Disclaimer

This report was written by students at the Computer Engineering Department, Faculty of Engineering, An-Najah National University. It has not been altered or corrected, other than editorial corrections, as a result of assessment and it may contain language as well as content errors. The views expressed in it together with any outcomes and recommendations are solely those of the student(s). An-Najah National University accepts no responsibility or liability for the consequences of this report being used for a purpose other than the purpose for which it was commissioned.

Abstract

SmartOps is an integrated operations management platform designed to support small and medium-sized enterprises (SMEs) in organizing and optimizing daily business activities. The system provides a centralized digital environment for managing users, projects, tasks, and analytical reports through both a web application and a mobile application that share a unified backend infrastructure.

Many SMEs still rely on fragmented tools, manual tracking methods, and unstructured communication, which reduces workflow visibility, causes delays, and limits performance monitoring. SmartOps addresses these challenges by offering a unified system with real-time updates, improving task coordination, transparency, and operational control across teams. The platform follows modern software engineering principles and a RESTful architecture, enabling scalable and maintainable development.

Role-Based Access Control (RBAC) is implemented to ensure secure and organized access for different user roles, including administrators, employees, and clients. In addition, SmartOps also includes an AI Health Analysis module that evaluates project progress, deadline pressure, task completion, pending work, overdue tasks, and high-priority risks. The module calculates a project health score, classifies risk level, predicts possible delay conditions, and presents a smart summary to support proactive project monitoring.

SmartOps is designed with a modular and extensible architecture to allow future enhancements such as advanced analytics, cloud deployment, and integration with external enterprise systems. The proposed solution aims to provide a practical, flexible, and future-ready platform that accelerates digital transformation within modern business environments.

Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
CORS	Cross-Origin Resource Sharing
CSS	Cascading Style Sheets
DTO	Data Transfer Object
ER	Entity-Relationship
HTTP	Hypertext Transfer Protocol
JWT	JSON Web Token
ML	Machine Learning
OTP	One-Time Password
RBAC	Role-Based Access Control
REST	Representational State Transfer
SME	Small and Medium-sized Enterprise
SQL	Structured Query Language
UI	User Interface
UX	User Experience

1 Introduction

1.1 General Background

The rapid growth of digital transformation has created a strong demand for integrated software solutions that unify project coordination, team management, and business reporting into a single platform. Small and medium-sized enterprises (SMEs) represent the backbone of many economies, yet they often lack access to enterprise-grade management tools that are both affordable and easy to deploy.

Modern businesses increasingly depend on software to coordinate operations across distributed teams, manage client relationships, and deliver projects on time. Despite this, many SMEs continue to rely on a fragmented collection of spreadsheets, messaging apps, and standalone tools that do not communicate with each other. This leads to duplicated effort, poor visibility into project status, and difficulty in monitoring employee performance or client satisfaction.

SmartOps was conceived to address this gap by providing a unified, web-based operations management platform tailored to the needs of software-focused SMEs. The platform connects administrators, employees, and clients in a single digital workspace and provides each role with the tools and information relevant to their responsibilities.

1.2 Problem Statement

SMEs involved in software development and service delivery face a set of recurring operational challenges that stem from the absence of a centralized management system. Specifically:

- Project progress and task assignments are tracked manually or across disconnected tools, making it difficult for managers to maintain real-time oversight.
- Clients have no structured mechanism to submit project requests, monitor progress, or receive updates, reducing transparency and trust.
- Employees lack a unified interface for viewing their assigned tasks, updating their status, and receiving notifications about changes.
- There is no systematic way to prioritize work intelligently based on deadlines, workload, or historical performance.

- The absence of a project template catalog forces teams to restart the planning process from scratch for every new engagement.

These problems collectively reduce operational efficiency, increase the risk of missed deadlines, and make it difficult to scale the business.

1.3 Objectives of the Project

The main objectives of SmartOps are:

1. To develop a centralized web platform that supports project creation, assignment, and tracking for administrators, employees, and clients.
2. To implement a Role-Based Access Control (RBAC) system that enforces secure, role-appropriate access to all system features.
3. To provide a client-facing portal where clients can submit project requests, track project status, and monitor task progress.
4. To build a project templates catalog that enables faster project initiation using predefined software project types.
5. To integrate a real-time notification system that keeps all users informed about task assignments, project updates, and request status changes.
6. To implement an AI-based project analysis module capable of evaluating project health, predicting delay risks, detecting bottlenecks, and generating managerial recommendations.
7. To deliver a responsive user interface accessible on both desktop and mobile devices, built with React.js and Tailwind CSS.

1.4 Significance of the Project

SmartOps addresses a real and underserved need in the SME software sector. By bringing project management, client communication, task tracking, and intelligent analysis together into one platform, it eliminates the overhead of juggling multiple tools and reduces the risk of miscommunication.

For administrators, SmartOps provides complete visibility over all ongoing projects, team workload, and client requests. For employees, it delivers a focused workspace showing only their relevant tasks and updates. For clients, it offers a transparent window into the work being done on their behalf.

The inclusion of a project templates catalog also has significant business value: it standardizes common project types (such as web applications, mobile apps, and management systems) and allows clients to initiate requests from proven starting points, reducing the time from request to project kickoff. The AI analysis module further enhances project oversight by providing risk assessment, health scoring, delay prediction, and automated recommendations for project managers.

1.5 Organization of the Report

The remainder of this report is organized as follows:

Chapter 2 presents the theoretical background, related work, and a comparison of existing project management platforms.

Chapter 3 describes the requirements analysis, system architecture, database design, technology stack, implementation methodology, and user interface design of SmartOps.

Chapter 4 presents the implementation results, including the developed web and mobile application interfaces, AI health analysis module, cloud deployment, and system evaluation.

Chapter 5 discusses the achievement of project objectives, the strengths and limitations of the system, and lessons learned during development.

Chapter 6 concludes the report and provides recommendations for future enhancements and research directions.

2 Theoretical Background and Previous Work

2.1 Operations Management and SMEs

Operations management is the discipline concerned with designing, overseeing, and controlling the processes that produce goods or deliver services. For SMEs in the software sector, effective operations management translates directly into the ability to deliver projects on time, within budget, and to client specifications.

A central challenge for SMEs is that enterprise-grade operations management tools such as SAP or Oracle ERP are prohibitively expensive and complex to configure. This has created demand for lighter, purpose-built platforms that retain core functionality while remaining accessible and affordable.

2.2 Role-Based Access Control (RBAC)

Role-Based Access Control is a security model in which system permissions are assigned to roles rather than to individual users. Users are then assigned to one or more roles, inheriting the associated permissions. RBAC is widely adopted in multi-user applications because it simplifies permission management, reduces the risk of privilege escalation, and makes it straightforward to enforce the principle of least privilege.

In SmartOps, RBAC is implemented with three primary roles: Admin, Employee, and Client. Each role has a distinct set of accessible pages, allowable operations, and visible data. This ensures that sensitive management functions are restricted to administrators while clients and employees interact only with the parts of the system relevant to them.

2.3 RESTful API Architecture

Representational State Transfer (REST) is an architectural style for designing networked applications. A RESTful API communicates over HTTP using standard methods (GET, POST, PUT, PATCH, DELETE) and represents resources as URLs. REST APIs are stateless, meaning each request from the client contains all information necessary for the server to fulfill it.

SmartOps uses a RESTful backend built with Express.js. Each functional domain (authentication, users, projects, tasks, notifications, project requests, and templates) is exposed as a set of versioned API endpoints. JWT-based authentication is used to secure all protected routes.

2.4 JSON Web Tokens (JWT)

JSON Web Tokens are a compact, URL-safe means of representing claims between two parties. In web application authentication, a JWT is issued by the server upon successful login and subsequently sent by the client with each request. The server validates the token using a secret key, eliminating the need to store session state server-side.

SmartOps uses JWT for stateless authentication. Upon login, the server signs a token containing the user's ID and role. The middleware on protected routes decodes and verifies this token before granting access.

2.5 Related Work and Existing Platforms

Several platforms offer overlapping functionality with SmartOps. Table 2.1 summarises how SmartOps compares with four widely-used tools.

Table 2.1: Comparison of SmartOps with Existing Platforms

Feature	**Trello**	**Jira**	**Asana**	**Monday**	**SmartOps**
RBAC (Admin/Emp/Client)	×	Partial	Partial	Partial	✓
Client project requests	×	×	×	×	✓
Project templates catalog	×	✓	Partial	Partial	✓
Real-time notifications	Partial	✓	✓	✓	✓
AI project Health analysis	×	×	×	×	✓
SME-focused affordability	✓	×	×	×	✓

The key differentiator of SmartOps is the combination of a client-facing request portal, a project templates catalog, and a planned AI module within a single, cohesive platform targeted at software SMEs.

3 Methodology

3.1 Preface

This chapter describes the development approach, technology decisions, system architecture, database design, and user interface of SmartOps. The system was built following modern software engineering practices, with a clear separation between frontend, backend, and data layers.

3.1.1 Tools, Methods, and Programming Languages

The SmartOps technology stack was selected to balance developer productivity, performance, and long-term maintainability. Table 3.1 lists the primary tools and their roles.

Table 3.1: SmartOps Technology Stack

Layer	Technology	Purpose
Frontend	React.js 19	Component-based UI framework
Frontend	Vite 8	Fast development build tool
Frontend	Tailwind CSS 3	Utility-first responsive styling
Frontend	React Router DOM 7	Client-side routing and protected routes
Frontend	Axios	HTTP client for API communication
Frontend	React Hook Form + Zod	Form management and schema validation
Frontend	Socket.IO Client	Real-time notification delivery
Mobile	Flutter (Dart)	Cross-platform iOS & Android mobile app
Mobile	Dio / http	HTTP client for REST API calls in Flutter
Mobile	flutter_secure_storage	Secure JWT token storage on device
Backend	Node.js	JavaScript runtime environment
Backend	Express.js 5	RESTful API framework
Backend	MySQL 2	Relational database driver
Backend	JSON Web Token (JWT)	Stateless user authentication
Backend	bcryptjs	Password hashing and verification
Backend	Zod	Server-side input validation (DTOs)
Backend	Brevo / Nodemailer	Transactional email (OTP delivery)
Database	MySQL	Relational data persistence
Cloud	Railway	Backend and database hosting/deployment
Design	Figma	UI/UX mockups and prototyping
Version Ctrl	Git / GitHub	Source control and collaboration
Testing	Postman	API development and endpoint testing

3.1.2 Frontend Architecture

The SmartOps frontend is a modern, single-page application (SPA) built with React.js 19 and bundled using Vite. The codebase follows a modular, feature-based architecture with a clear directory structure that separates concerns across dedicated folders:

- `src/assets/` — Static assets such as images and icons.
- `src/components/` — Reusable UI components organised by feature domain:
  - `auth/` — Login, register, OTP, and password reset components.
  - `common/` — Shared elements (buttons, modals, loaders).

  - `dashboard/` — Dashboard overview widgets and cards.
  - `dashboard-content/` — Role-specific dashboard content panels.
  - `notifications/` — Notification bell and notification list items.
  - `project-request/` — Client request form and request list components.
  - `project-templates/` — Template browsing and selection components.
  - `projects/` — Project cards, project creation form, project list.
  - `tasks/` — Task cards, task assignment form, status update controls.
  - `landing/` — Landing page sections and hero components.
  - `layout/` — Sidebar, navbar, and page wrapper components.
- `src/pages/` — Top-level page components, each mapped to a route (e.g. `DashboardPage`, `ProjectsPage`, `TasksPage`).
- `src/routes/` — Route definitions, `<ProtectedRoute>`, and `<RoleRoute>` wrappers that enforce authentication and role-based access control at the navigation level.
- `src/services/` — Axios-based API service modules, one per domain (auth, projects, tasks, notifications, requests, templates).
- `src/context/` — React Context providers for global state (authentication token, current user, notification count).
- `src/hooks/` — Custom React hooks that encapsulate reusable logic (e.g. `useAuth`, `useNotifications`).
- `src/utils/` — Utility functions (date formatting, status colour mapping, role helpers).

The entry point is `src/main.jsx`, which mounts the React app into the DOM. `src/App.jsx` wraps the application with context providers and renders the `AppRoutes` component. Routing is handled by React Router DOM v7. After successful login, the `RoleRoute` wrapper reads the user's role from context and redirects to the appropriate interface.

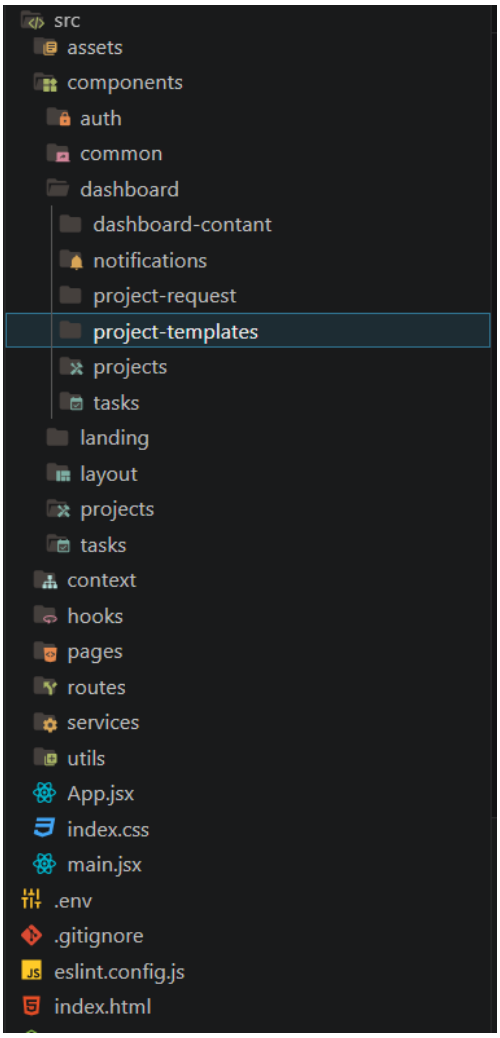


Figure 3.1: SmartOps – Frontend Source Directory Structure

3.1.3 Backend Architecture

The SmartOps backend is a RESTful API built with Node.js and Express.js 5. It follows a layered architecture that separates routing, business logic, data access, validation, real-time communication, and intelligent project analysis. The backend serves both the React.js web application and the Flutter mobile application through a unified API layer.

- **Routes** (`src/routes/`) — Define HTTP endpoints and attach middleware chains. Route modules cover authentication, users, projects, tasks, project requests, project templates, notifications, and AI analysis.
- **Controllers** (`src/controllers/`) — Contain the business logic for each endpoint. Controllers interact with the MySQL database through the `mysql2/promise` connection pool and coordinate validation, notifications, and project workflows.
- **Middleware** (`src/middleware/`) — Middleware modules handle JWT authentica-

tion (`authMiddleware`), role enforcement (`roleMiddleware`), and DTO validation (`validateMiddleware`).

- **DTOs** (`src/dtos/`) — Zod schemas that validate and sanitize incoming request bodies for authentication, users, projects, tasks, project requests, project templates, and AI-related requests.
- **AI Analysis Module** (`src/services/aiAnalyzer.js`) — Provides intelligent project evaluation by analyzing project progress, task completion rates, overdue tasks, priority levels, deadlines, and project status. The module calculates project health scores, classifies risk levels, predicts delay risks, detects bottlenecks, generates recommendations, and can trigger automated notifications when high-risk conditions are detected.
- **Real-Time Communication** (`Socket.IO`) — Enables instant notification delivery between the server and connected clients. Notifications are pushed to users in real time without requiring page refreshes.
- **Utilities** (`src/utils/`) — Helper functions including OTP generation, email delivery through Brevo/Nodemailer, date handling, and notification helpers. “‘

Table 3.2 summarises the main API endpoint groups exposed by the backend.

Table 3.2: SmartOps API Endpoint Groups

Base Path	Methods	Description
`/api/auth`	POST, GET	Register, login, OTP verify, reset password, get current user
`/api/users`	GET, POST, PUT, DELETE	Admin user management (CRUD)
`/api/projects`	GET, POST, PUT, DELETE	Project management (Admin only for write)
`/api/tasks`	GET, POST, PUT, PATCH, DELETE	Task management; employees can update status
`/api/project-requests`	GET, POST, PUT	Client requests; Admin reviews and approves/rejects
`/api/project-templates`	GET, POST, PUT, DELETE	Template catalog management
`/api/notifications`	GET, PATCH	Retrieve and mark notifications as read
`/api/ai-analysis`	GET, POST	Project health analysis, risk assessment, delay prediction, bottleneck detection, and recommendations

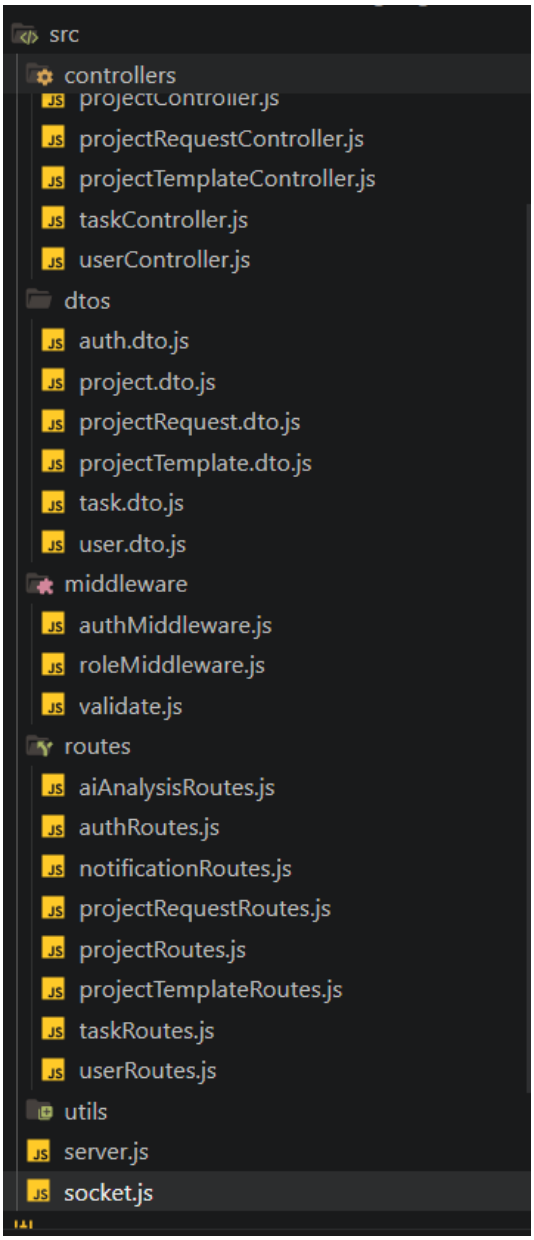


Figure 3.2: SmartOps – Backend Architecture

3.1.4 API Testing with Postman

All REST API endpoints were tested using Postman before integration with the React web application and Flutter mobile application. Postman collections were used to verify authentication, authorization, input validation, CRUD operations, project request workflows, notification generation, and role-based access control.

Authentication-protected endpoints were tested using JWT Bearer tokens obtained from the login endpoint. Test scenarios included successful requests, invalid input validation, unauthorized access attempts, and role-restricted operations.

The testing process helped identify integration issues early and ensured that all backend

endpoints behaved correctly before being consumed by the frontend and mobile clients.

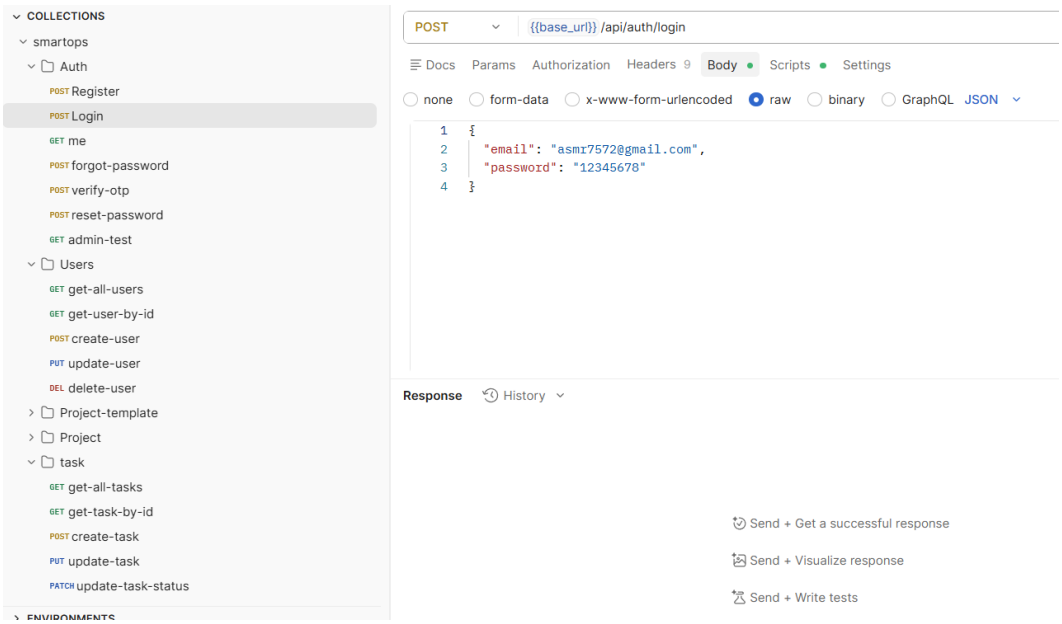


Figure 3.3: SmartOps API Testing Using Postman

3.1.5 Database Design

SmartOps uses MySQL as its relational database management system. The schema comprises seven tables that model the core entities of the platform:

- **users** — Stores all system users with fields: `user_id`, `name`, `email`, `password`, `role` (`enum`: admin, employee, client), `created_at`.
- **projects** — Stores projects with client association, priority, status (`pending`, `in_progress`, `completed`, `cancelled`), deadlines, and template reference.
- **tasks** — Stores tasks linked to a project, assigned to an employee, with priority (`low`, `medium`, `high`) and status.
- **project_requests** — Stores client-submitted project requests with review workflow fields: `status` (`pending`, `approved`, `rejected`), `rejection_reason`, `reviewed_by`.
- **project_templates** — A catalog of predefined software project types (e.g. web application, mobile app, management system) with estimated durations.
- **notifications** — Stores system notifications for users with `is_read` boolean and `type` classification.
- **ai_analysis** — Stores AI-generated analysis per project: `health_score` and `predicted_delay`.

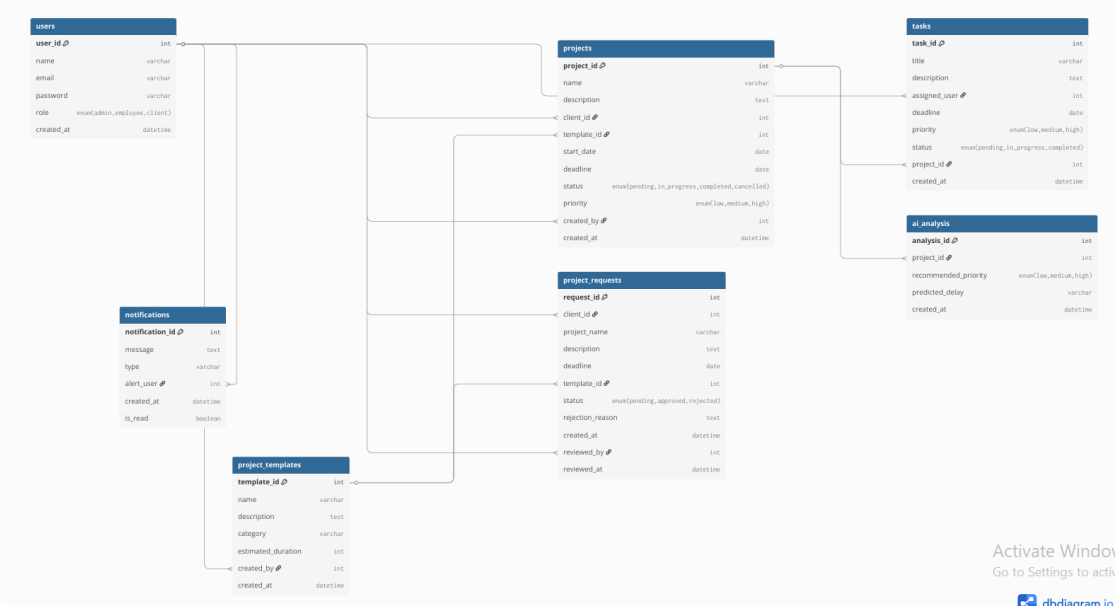


Figure 3.4: SmartOps – Entity Relationship Diagram

3.1.6 System Architecture Overview

Figure 3.5 shows the full system architecture, illustrating how the React.js frontend communicates with the Express.js backend over REST, how the backend connects to the MySQL database, and how real-time notifications are delivered via Socket.IO.

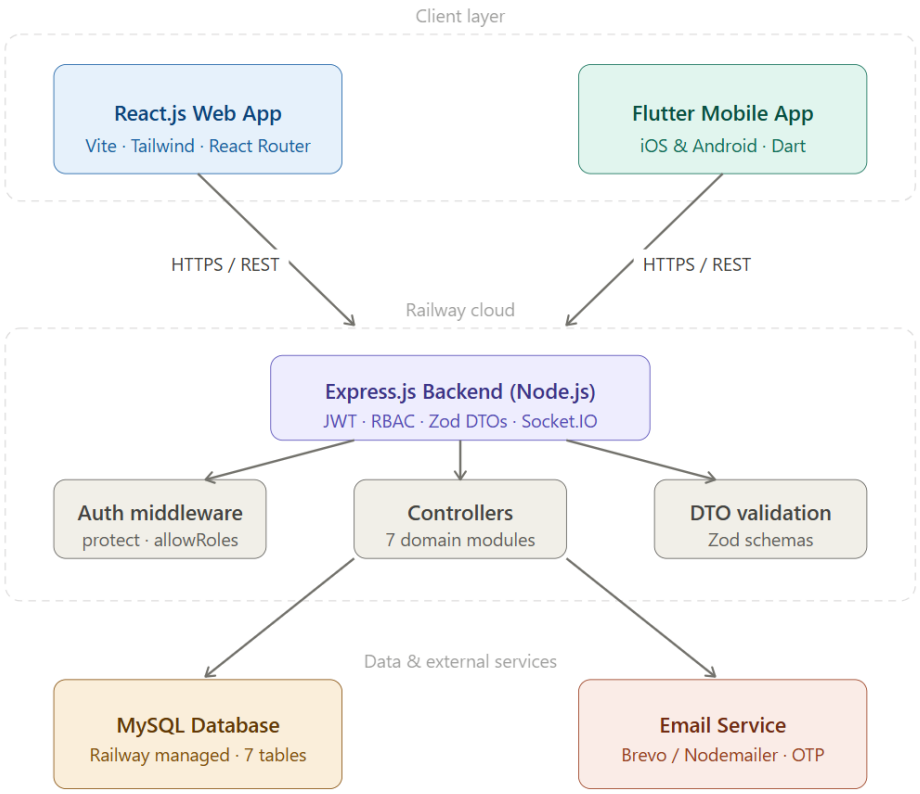


Figure 3.5: SmartOps – Full System Architecture

3.2 Standards and Specifications

SmartOps was designed and implemented in accordance with a set of widely recognised software engineering standards and industry best practices. The following standards were applied throughout the development process.

REST Architectural Style (Fielding, 2000). The SmartOps backend strictly follows the REST constraints: stateless communication, uniform resource identification via URLs, and standard HTTP methods (GET, POST, PUT, PATCH, DELETE). All API responses use JSON and standard HTTP status codes (200, 201, 400, 401, 403, 404, 422, 500). This ensures interoperability between the web frontend, the Flutter mobile app, and any future third-party integrations.

JSON Web Token Standard (RFC 7519). Authentication is implemented using JSON Web Tokens as specified in RFC 7519. Tokens are signed using the HMAC-SHA256 algorithm (`HS256`), contain a standard set of claims (`sub`, `iat`, `exp`), and are transmitted via the HTTP `Authorization: Bearer` header. The 7-day expiry window and server-side signature verification follow the standard's recommendations for web application sessions.

OWASP Security Guidelines. The project follows relevant OWASP Top 10 mitigations:

- **Broken Access Control (A01):** Enforced via dual-layer RBAC (frontend route guards + backend `roleMiddleware`).
- **Injection (A03):** All SQL queries use parameterised statements through the `mysql2` prepared-statement API, preventing SQL injection.
- **Security Misconfiguration (A05):** CORS is configured on the Express server to restrict cross-origin requests to trusted origins only.
- **Identification and Authentication Failures (A07):** Passwords are hashed with bcryptjs (salt rounds = 10); JWT secrets are stored in environment variables, never in source code.

Material Design (Flutter). The SmartOps Flutter mobile application follows Google's Material Design 3 guidelines for layout, typography, colour system, and interactive components. This ensures a consistent, accessible, and familiar user experience on both Android and iOS devices.

Semantic Versioning. All project dependencies follow Semantic Versioning (SemVer) conventions. Dependency versions are pinned in `package.json` (Node.js) and `pubspec.yaml` (Flutter) to guarantee reproducible builds.

IEEE 830 — Software Requirements Specification. The SmartOps system requirements (Chapter 3, Section 3.5) were structured following the IEEE 830 SRS standard: each requirement is uniquely identified (URn), clearly stated, testable, and traceable to a specific feature in the implementation.

3.3 Design Constraints

The design and development of SmartOps were shaped by several realistic constraints across economic, technical, societal, ethical, and sustainability dimensions.

Economic Constraints. The project was developed as a university graduation project with no commercial budget. All technologies selected are open-source and free for development use (React.js, Flutter, Express.js, MySQL). Cloud deployment was achieved using Railway's free/starter tier, which imposes limits on compute resources (shared CPU, 512 MB RAM) and monthly usage hours. This constrained the system's ability to handle very high concurrent load in the current deployment and was a primary reason for selecting a lightweight, connection-pooled architecture (`connectionLimit: 10`) on the backend.

Technical Constraints.

- **Stateless backend:** The use of JWT authentication required the frontend to manage token storage and renewal, since the server maintains no session state.
- **Single-database architecture:** Both the web frontend and the Flutter mobile app share a single MySQL instance hosted on Railway. This simplifies data consistency but means the database is a single point of failure in the current architecture.
- **AI module deferral:** Machine-learning-based prediction was deferred due to insufficient historical project data. Instead, a rule-based AI Health Analysis module was implemented to evaluate project health, risk levels, delay conditions, bottlenecks, and recommendations.
- **Railway deployment constraints:** Railway's free tier does not support persistent file storage, which required all media handling to remain URL-based or deferred to a later phase.

Societal and Privacy Constraints. SmartOps stores personally identifiable information (PII) including names, email addresses, and business project data. All passwords are stored exclusively as bcrypt hashes. No plaintext credentials are ever logged or returned in API responses. The system's RBAC model ensures employees cannot access client data and clients cannot access other clients' projects, protecting user privacy.

Ethical Constraints. The project does not collect data beyond what is necessary for

its operational purpose. No tracking, advertising, or analytics data is shared with third parties. All third-party libraries used are open-source with permissive licences (MIT, Apache 2.0), ensuring no intellectual property violations.

Sustainability Constraints. The modular codebase (separated frontend, backend, and mobile app repositories) is designed to support long-term maintenance and future enhancement. New features can be added as independent modules without restructuring the existing system. The use of widely-adopted technologies (React, Flutter, Node.js, MySQL) ensures that the project can be maintained by future developers who are familiar with the standard ecosystem.

SmartOps includes a project templates catalog that covers the most common categories of software projects that SMEs develop for their clients. Administrators can define and manage templates in the following categories:

- Web Applications
- Mobile Applications
- E-commerce Systems
- Management Systems (HR, Inventory, Hospital, School, etc.)
- AI-based Systems
- IoT / Embedded Systems
- Network and Cybersecurity Solutions
- Database Systems
- UI/UX Redesign Projects
- Automation Systems
- Cloud-based Solutions

Clients can browse these templates and select one as the basis for a project request, pre-populating standard project details and estimated durations.

3.4 System Requirements

3.4.1 Functional Requirements

Users (Admin, Employee, Client)

UR1: The system shall enable the Admin to register new user accounts.

- SR1: The system shall require the Admin to select the user role (Admin, Employee, Client) during account creation.
- SR2: The system shall provide a registration form with mandatory fields: name, email, and password.
- SR3: The system shall ensure that each email is unique.
- SR4: The system shall securely store passwords using encryption.

UR2: The system shall allow users to log in securely.

- SR1: The system shall validate user credentials (email and password).
- SR2: The system shall generate a secure authentication token (JWT) upon successful login.
- SR3: The system shall restrict access to protected endpoints without a valid token.

Project Management

UR3: The system shall enable the Admin to create and manage projects.

- SR1: The system shall allow the Admin to create a project with fields: name, description, start date, deadline, status, and priority.
- SR2: The system shall store the creator (Admin) of the project.
- SR3: The system shall allow retrieving all projects created by the Admin.
- SR4: The system shall associate a project with the corresponding Client when created from an approved request.

UR4: The system shall allow users to view projects.

- SR1: The system shall allow Employees and Clients to view project details.
- SR2: The system shall display project status and deadlines.

Task Management

UR5: The system shall enable the Admin to create tasks within a project.

- SR1: The system shall allow the Admin to assign a task to an Employee.
- SR2: The system shall require task fields: title, description, assigned user, deadline, priority, and project ID.
- SR3: The system shall set the default task status to “pending”.

UR6: The system shall allow Employees to view their assigned tasks.

- SR1: The system shall retrieve tasks based on the logged-in user ID.
- SR2: The system shall display task details including priority, deadline, and status.

UR7: The system shall allow Employees to update task status.

- SR1: The system shall allow updating task status (pending, in_progress, completed).
- SR2: The system shall ensure only the assigned user can update their task.

Notification System

UR8: The system shall notify users about important actions.

- SR1: The system shall create a notification when a task is assigned.
- SR2: The system shall allow users to retrieve their notifications.
- SR3: The system shall allow users to mark notifications as read.

Software Templates Catalog

UR9: The system shall manage project templates.

- SR1: The system shall allow the Admin to add project templates.
- SR2: The system shall store template name, description, category, and estimated duration.
- SR3: The system shall allow Clients to browse available templates.
- SR4: The system shall allow templates to be used in project requests.

Security & Validation

UR10: The system shall ensure data security and integrity.

- SR1: The system shall validate all input data before processing.
- SR2: The system shall protect routes using authentication middleware.
- SR3: The system shall prevent unauthorized access to resources.

Client Functionalities

UR11: The system shall allow Clients to view their assigned projects.

- SR1: The system shall retrieve projects associated with the Client.
- SR2: The system shall display project details including name, status, and deadlines.

UR12: The system shall allow Clients to monitor task progress.

- SR1: The system shall allow Clients to view tasks within a project.
- SR2: The system shall display task status (pending, in_progress, completed).
- SR3: The system shall ensure Clients have read-only access to tasks.

UR13: The system shall notify Clients about project updates.

- SR1: The system shall notify Clients when project status changes.
- SR2: The system shall notify Clients when a request is approved or rejected.
- SR3: The system shall allow Clients to view their notifications.

Client Project Requests

UR15: The system shall allow Clients to request new projects.

- SR1: The system shall allow the Client to submit a project request with project name, deadline, and description.
- SR2: The system shall associate the request with the Client ID.
- SR3: The system shall set the request status to "pending" by default.

UR16: The system shall allow Admin to review project requests.

- SR1: The system shall allow the Admin to view all pending requests.
- SR2: The system shall allow the Admin to approve or reject a request.
- SR3: The system shall allow the Admin to provide a rejection reason.
- SR4: Upon approval, the system shall create a new project based on the request.

UR17: The system shall notify Clients about request status updates.

- SR1: The system shall notify the Client when the request is approved or rejected.
- SR2: The system shall allow the Client to view request status.

Template-Based Project Requests

UR18: The system shall allow Clients to request projects based on existing templates.

- SR1: The system shall allow Clients to browse and select from available project templates before submitting a request.
- SR2: The system shall store the selected template reference in the request.

- SR3: The system shall allow Clients to optionally modify project details before submission.

UR19: The system shall create a new project from a template upon approval.

- SR1: The system shall duplicate the selected template into a new project.
- SR2: The system shall assign the new project to the Admin.
- SR3: The system shall maintain a reference to the original template for tracking.

AI Health Analysis

UR20: The system shall provide AI-based project health analysis.

- SR1: The system shall analyze project task statistics, including total tasks, completed tasks, pending tasks, in-progress tasks, overdue tasks, and high-priority tasks.
- SR2: The system shall calculate a project health score from 0 to 100.
- SR3: The system shall classify the project risk level as low, medium, high, or critical based on the calculated health score.
- SR4: The system shall generate a delay prediction message based on project progress, task status, and deadline pressure.
- SR5: The system shall generate a smart project summary describing the current project condition.
- SR6: The system shall detect bottlenecks such as overdue tasks, high-priority tasks not started, or a large number of pending tasks.
- SR7: The system shall provide recommendations to help administrators improve project progress and reduce delay risk.
- SR8: The system shall display the AI analysis result inside the Project Details page.

3.4.2 AI Health Analysis Module

SmartOps includes an AI Health Analysis module integrated inside the Project Details page. This module analyzes the current state of a selected project and provides an intelligent health overview based on task progress, pending tasks, overdue tasks, high-priority tasks, project deadline, and project status.

The module calculates a health score from 0 to 100 and classifies the project risk level as low, medium, high, or critical. It also generates a short smart summary and a delay prediction message that helps administrators understand whether the project is progressing normally or may require attention.

The analysis is generated when the user clicks the *Generate AI Analysis* button. The result is displayed as a visual card inside the Project Details page, showing the health score, risk level, prediction, completion percentage, pending tasks, overdue tasks, and high-priority task indicators.

Although the module is described as AI-based, the current implementation uses a rule-based intelligent scoring engine rather than a trained machine learning model. This approach was selected because the system does not yet contain enough historical project data to train a reliable predictive model. Note: The AI module (UR20) is planned for a future development phase. The database schema includes the `ai_analysis` table to support this feature, but the machine learning model has not yet been implemented.

3.4.3 Non-Functional Requirements

1. **Performance:** API responses for standard queries shall be returned within 500 milliseconds under normal load.
2. **Security:** All passwords shall be hashed using bcryptjs before storage. All API routes shall be protected by JWT authentication middleware except public endpoints (login, register).
3. **Scalability:** The backend shall use a MySQL connection pool (`connectionLimit: 10`) to support concurrent users without blocking.
4. **Usability:** The user interface shall be responsive and usable on desktop and mobile screen sizes.
5. **Maintainability:** The codebase shall follow a modular folder structure with clear separation between routes, controllers, DTOs, and middleware.
6. **Validation:** All API inputs shall be validated against Zod/Joi schemas (DTOs) before reaching business logic.

3.5 UI/UX Design

The SmartOps user interface was designed in Figma before implementation. The design follows a dark, professional theme with a navy-blue sidebar, clean card layouts, and colour-coded status indicators. The interface was built to be fully responsive, supporting both desktop and mobile viewports.

The application covers the following screens: Landing Page, Register, Login, Forgot Password, Reset Password, Verify OTP, Dashboard, Projects, Project Details, Tasks, Assign Task, Requests, Project Templates, Notifications, and Export Report.

3.5.1 Landing Page

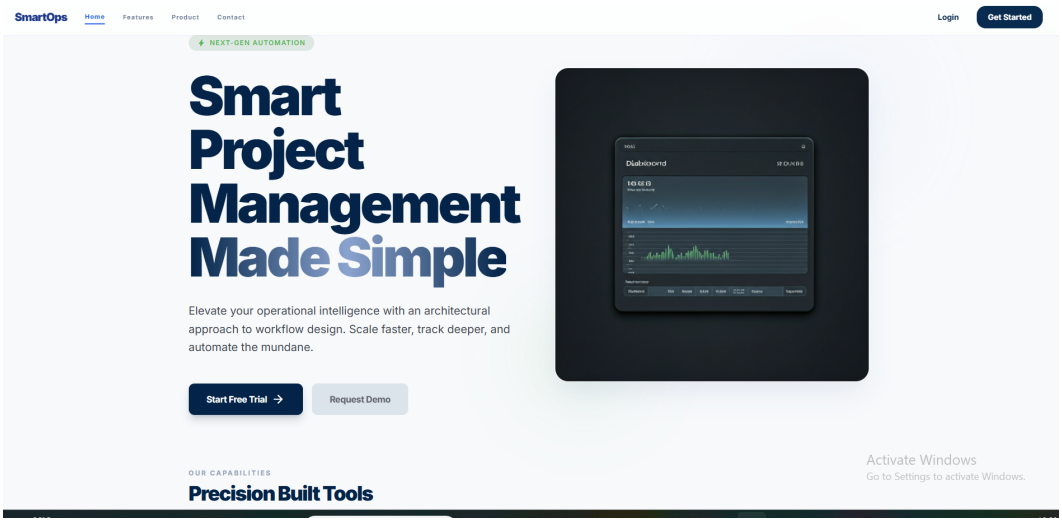


Figure 3.6: SmartOps – Landing Page

3.5.2 Authentication Pages

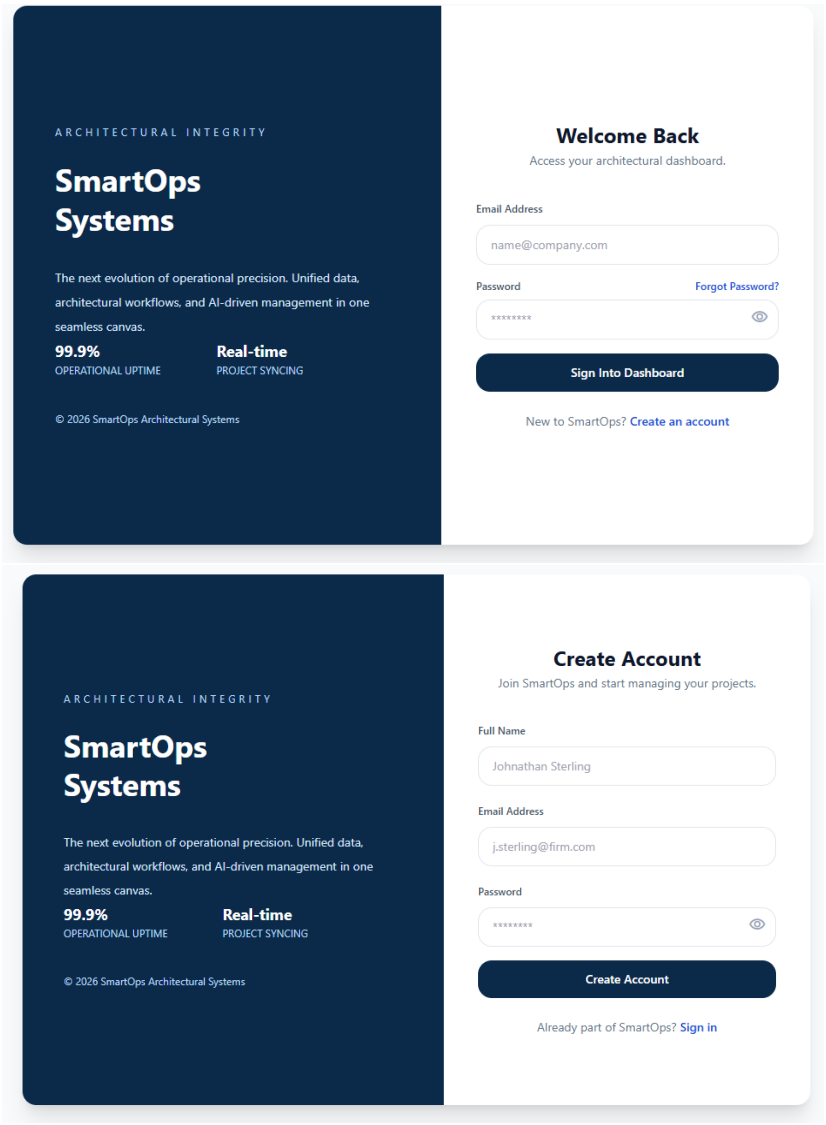


Figure 3.7: SmartOps – Login and Register Pages

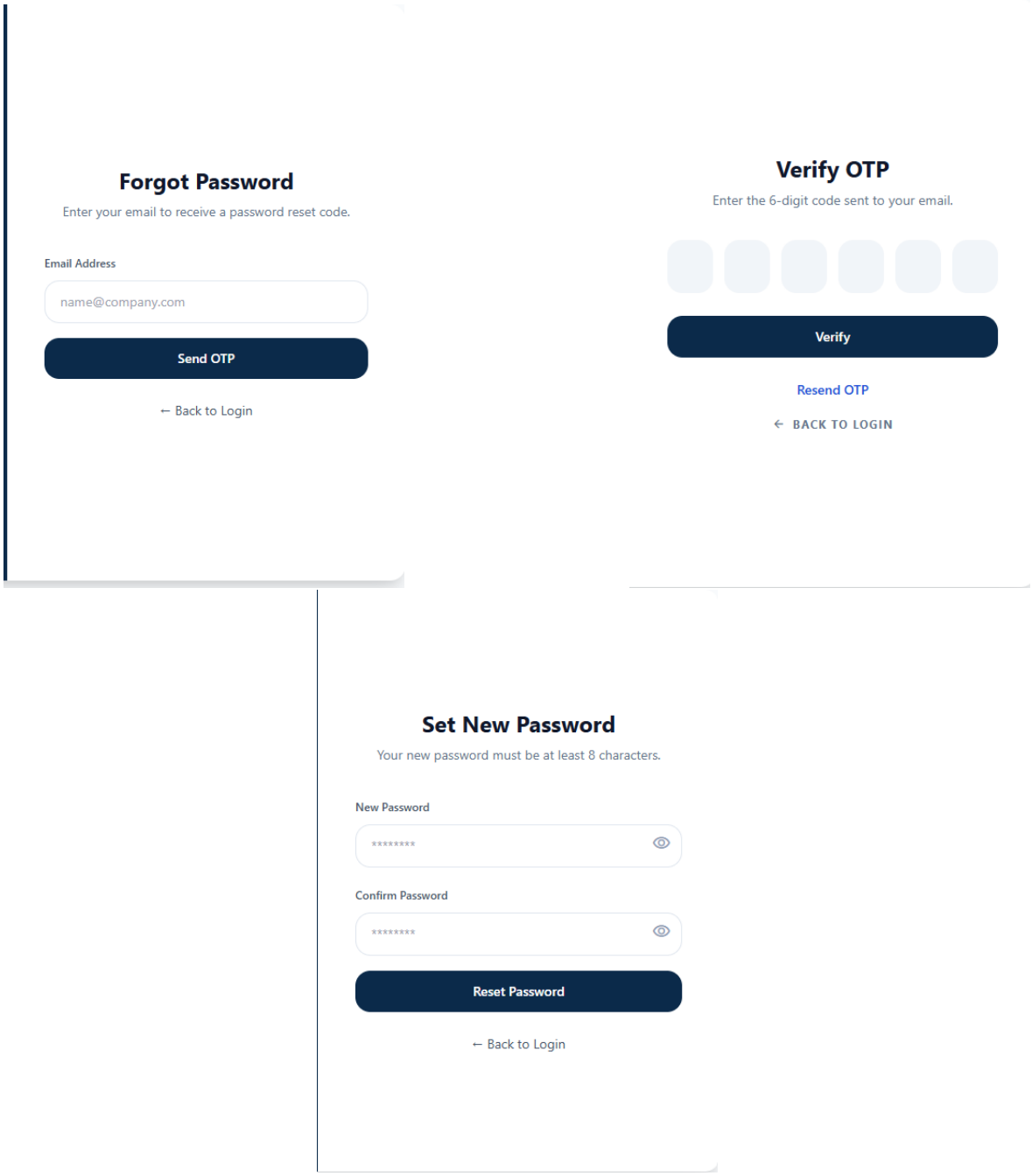
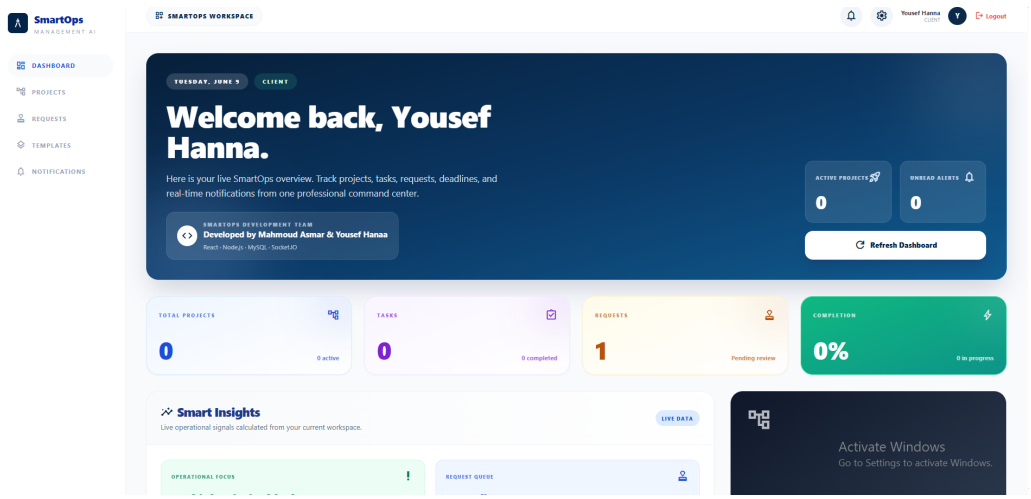
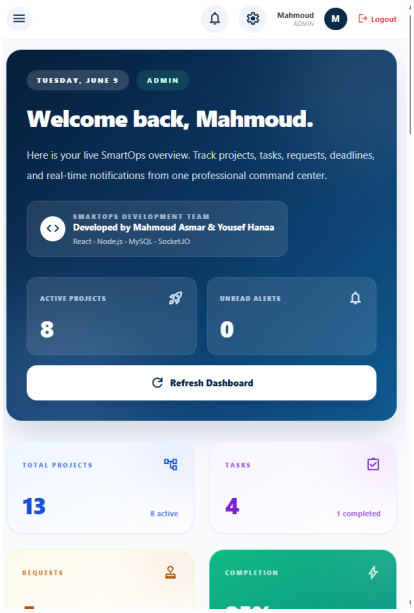


Figure 3.8: SmartOps – Password Recovery Flow

3.5.3 Dashboard Page



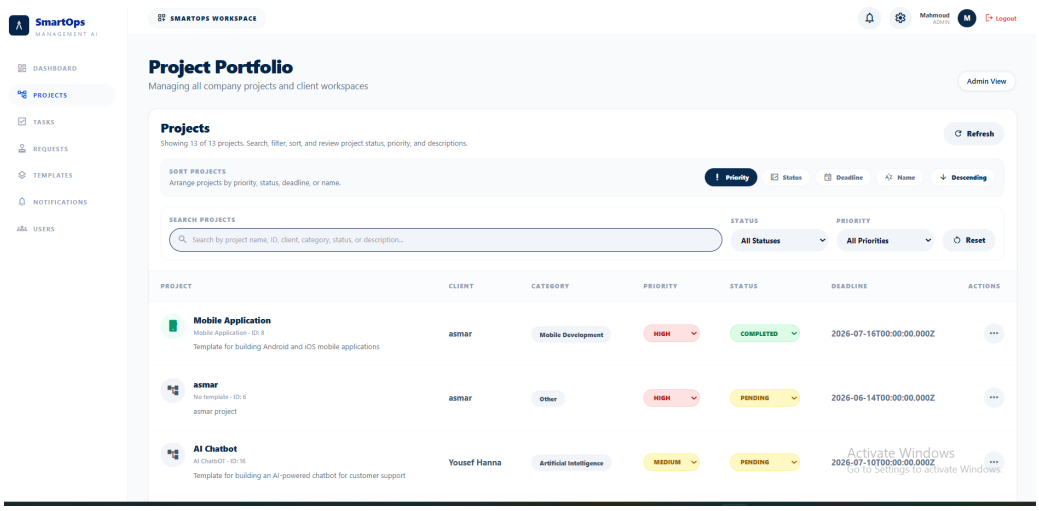
(a) Desktop View



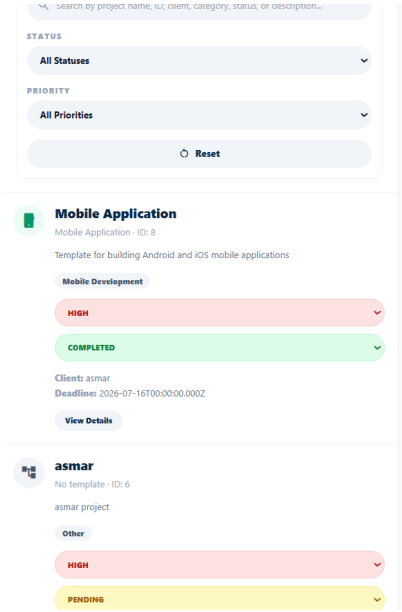
(b) Mobile View

Figure 3.9: SmartOps – Dashboard Page

3.5.4 Projects Page



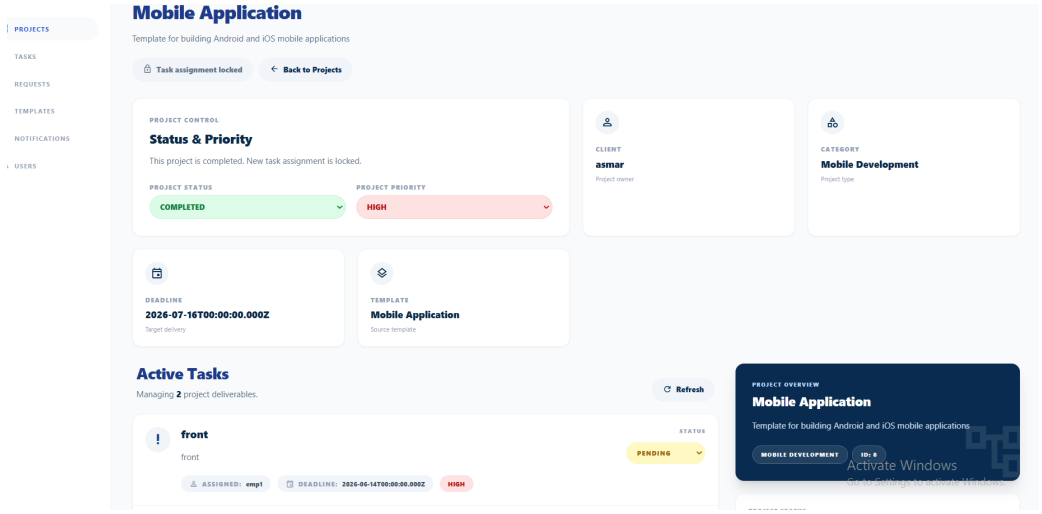
(a) Desktop View



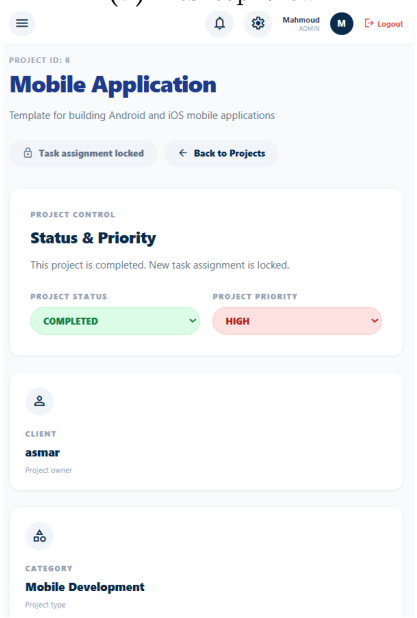
(b) Mobile View

Figure 3.10: SmartOps – Projects Page

3.5.5 Project Details Page



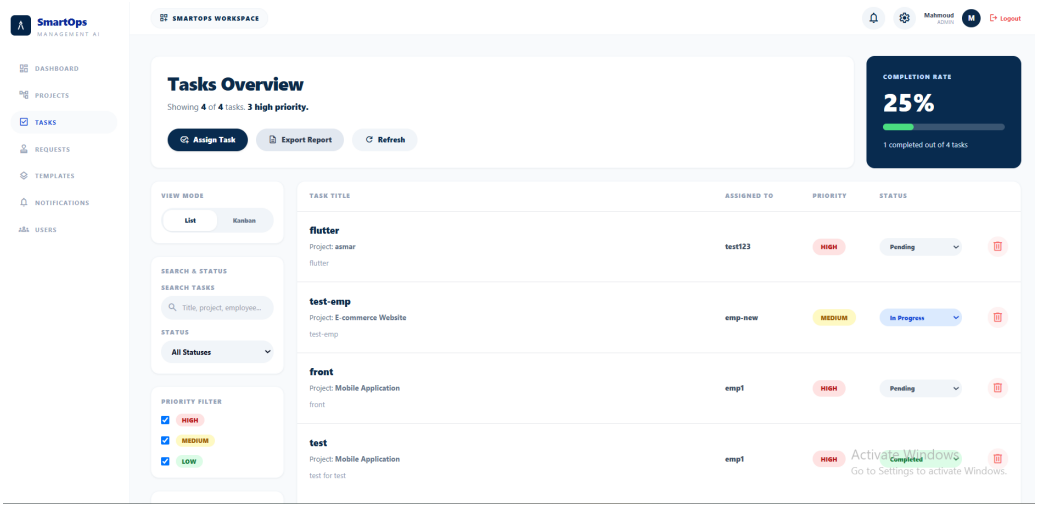
(a) Desktop View



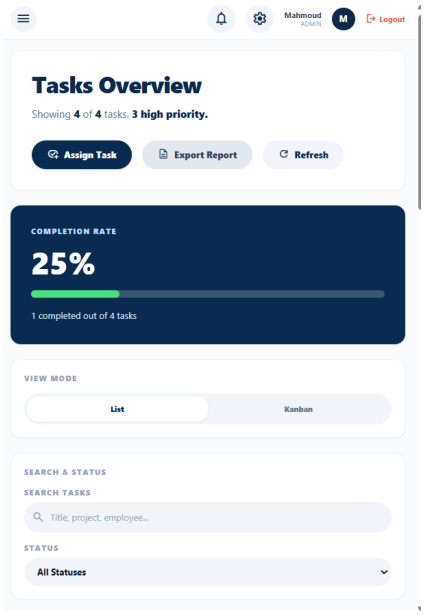
(b) Mobile View

Figure 3.11: SmartOps – Projects Details Page

3.5.6 Tasks Page



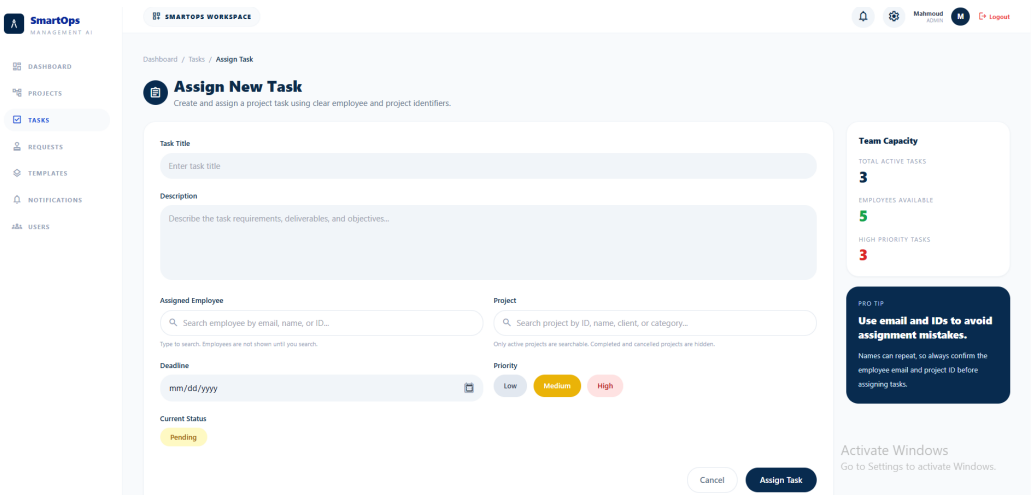
(a) Desktop View



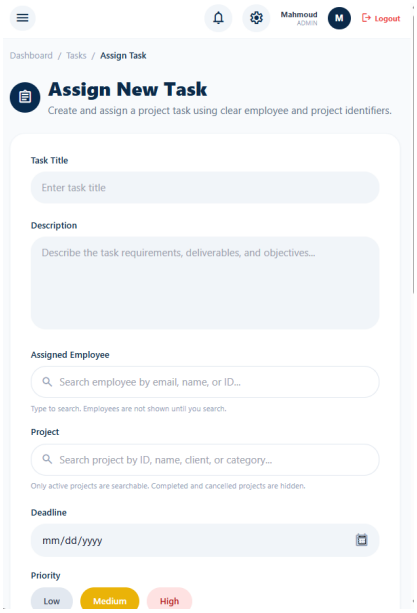
(b) Mobile View

Figure 3.12: SmartOps – Tasks Page

3.5.7 Assign Task Page



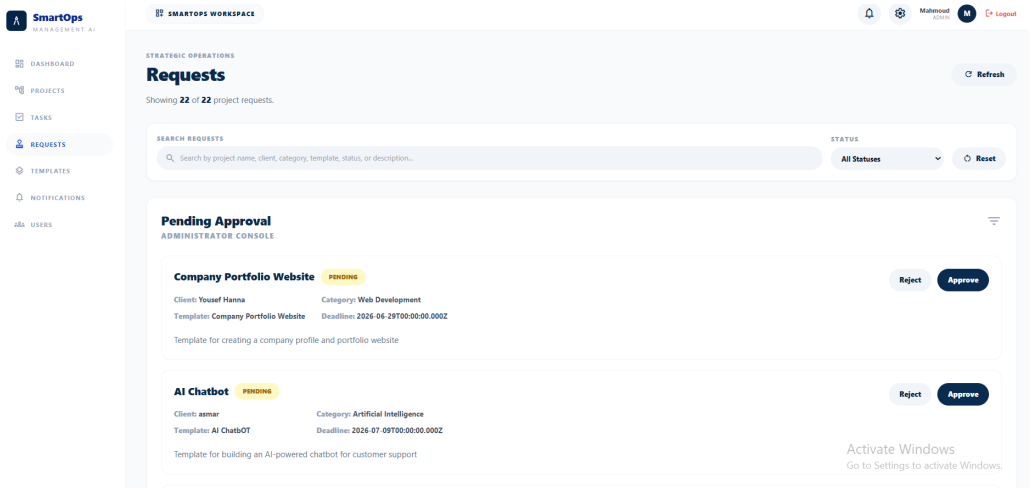
(a) Desktop View



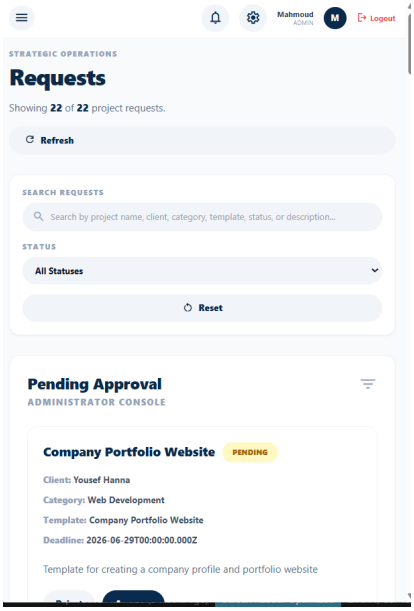
(b) Mobile View

Figure 3.13: SmartOps – Assign Task Page

3.5.8 Requests Page



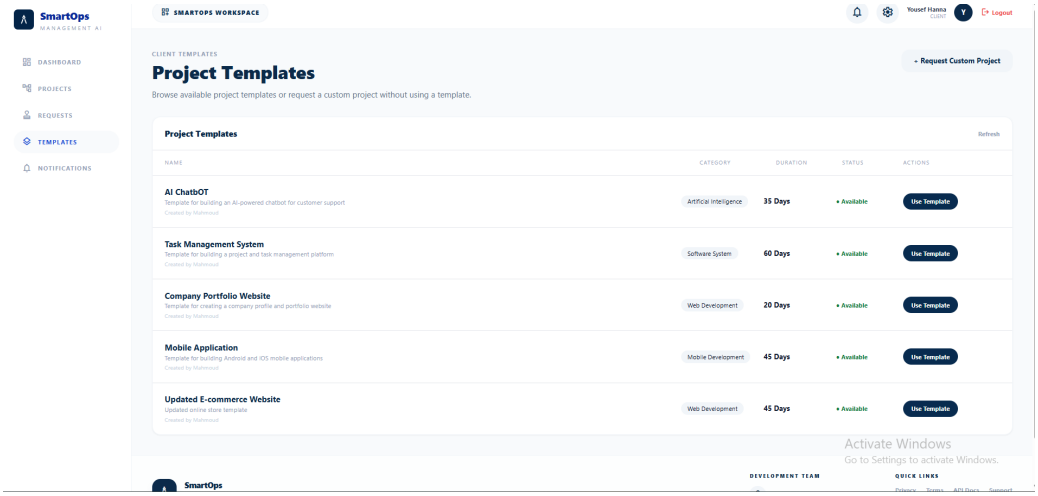
(a) Desktop View



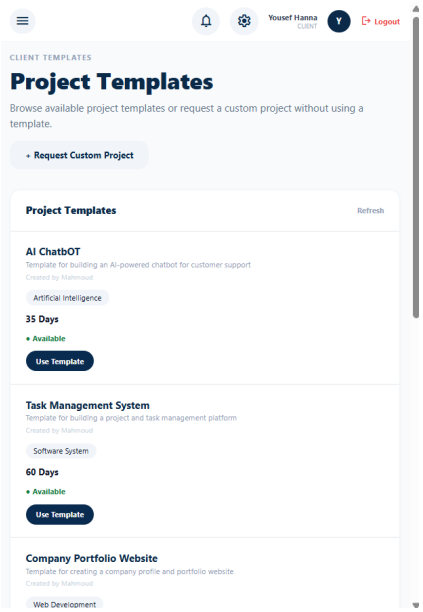
(b) Mobile View

Figure 3.14: SmartOps – Requests Page

3.5.9 Project Templates Page



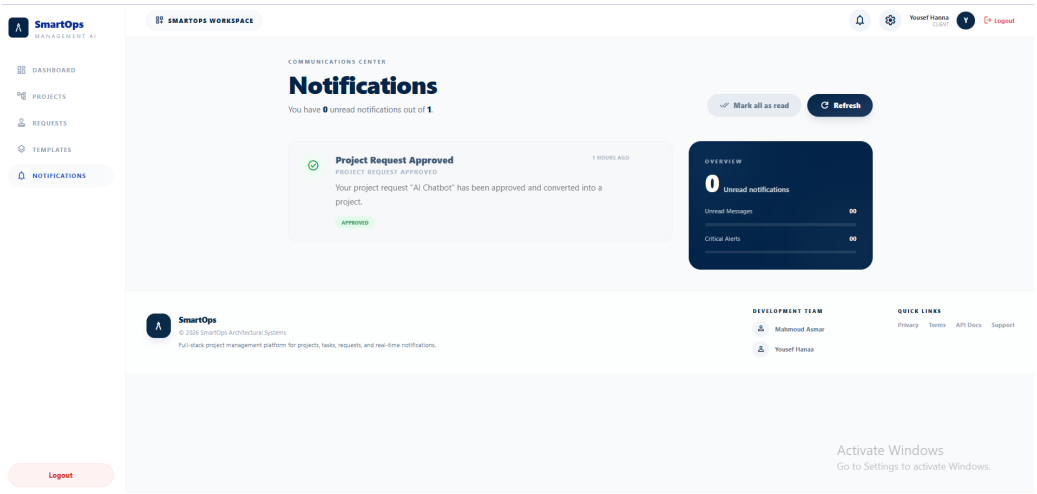
(a) Desktop View



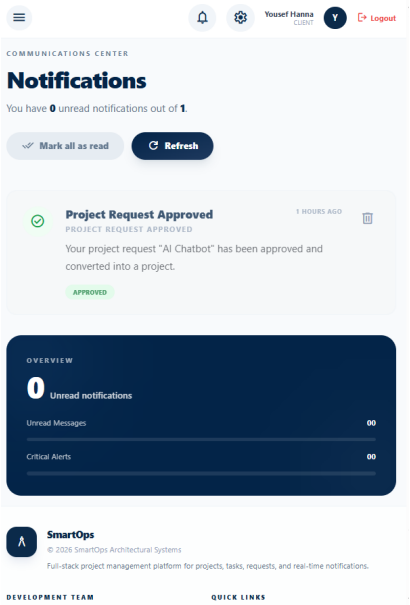
(b) Mobile View

Figure 3.15: SmartOps – Templates Page

3.5.10 Notifications Page



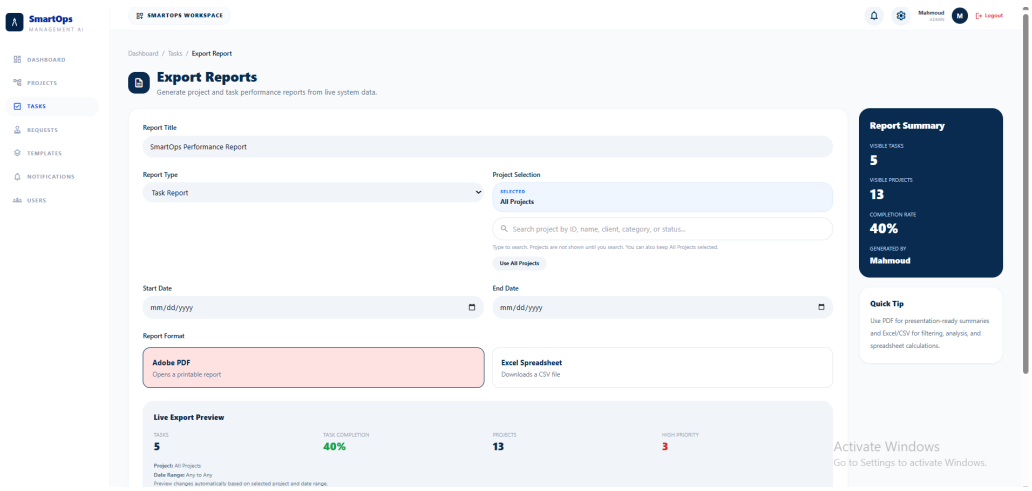
(a) Desktop View



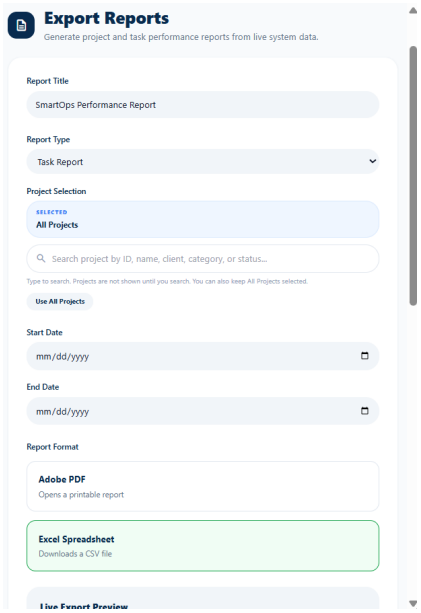
(b) Mobile View

Figure 3.16: SmartOps – Notifications Page

3.5.11 Export Report Page



(a) Desktop View



(b) Mobile View

Figure 3.17: SmartOps – Export Report Page

3.6 How the System Works

This section explains the internal logic of each major SmartOps workflow from the perspective of both the user and the code, tracing the path from a user action in the browser through the API layer down to the database, and back.

3.6.1 Authentication and Password Recovery Flow

Registration. When a new user visits the Register page and submits the form, the frontend calls `POST /api/auth/register`. The request body is validated by

the `registerDto` Zod schema (minimum 2-character name, valid email, minimum 8-character password). The `authController.register` function checks for email uniqueness in the `users` table, hashes the password with `bcryptjs` (salt rounds = 10), and inserts a new row with the role `client` (the public registration endpoint always creates clients; admins and employees are created by the Admin through the Users management page).

Login. On the Login page, after the user submits their credentials, the frontend calls `POST /api/auth/login`. The controller queries the `users` table by email, verifies the submitted password against the stored hash using `bcrypt.compare`, and — on success — signs a JWT containing `user_id`, `email`, and `role` with a 7-day expiry. The token and user object are returned to the frontend, where they are stored in `localStorage`. The `AuthContext` provider (React Context) exposes `user`, `token`, and `isAuthenticated` to every component. After login, the `AppRoutes` component reads the user's role and redirects to the appropriate section of the application via the `RoleRoute` wrapper.

Token-based request authentication. Every subsequent API request from the frontend passes through the Axios request interceptor defined in `src/services/api.js`, which reads the JWT from `localStorage` and attaches it as a `Bearer` token in the `Authorization` header. On the backend, the `authMiddleware.protect` function decodes the token using `jwt.verify`, queries the user from the database, and attaches the user object to `req.user`. If the token is missing, expired, or invalid, a 401 response is returned immediately.

Password Recovery (Forgot Password → OTP → Reset). The password recovery flow is a three-step process:

1. The user enters their email on the Forgot Password page. The frontend calls `POST /api/auth/forgot-password`. The controller generates a 6-digit numeric OTP (`Math.random`-based), sets an expiry of 10 minutes in the `reset_otp_expires` column, and sends the OTP to the user's email via the `sendEmail` utility (Brevo/Nodemailer).

2. The user enters the OTP on the Verify OTP page. The frontend calls `POST /api/auth/verify-otp`. The controller checks the OTP value, ensures it has not expired, and sets `is_otp_verified = TRUE` in the database. The email is passed through session storage between pages.

3. On the Reset Password page, the user enters a new password. The frontend calls `POST /api/auth/reset-password`. The controller verifies that `is_otp_verified` is `TRUE` and the OTP window has not expired, hashes the new password, updates the `password` column, and clears all OTP-related fields (`reset_otp`, `reset_otp_expires`,

`is_otp_verified`).

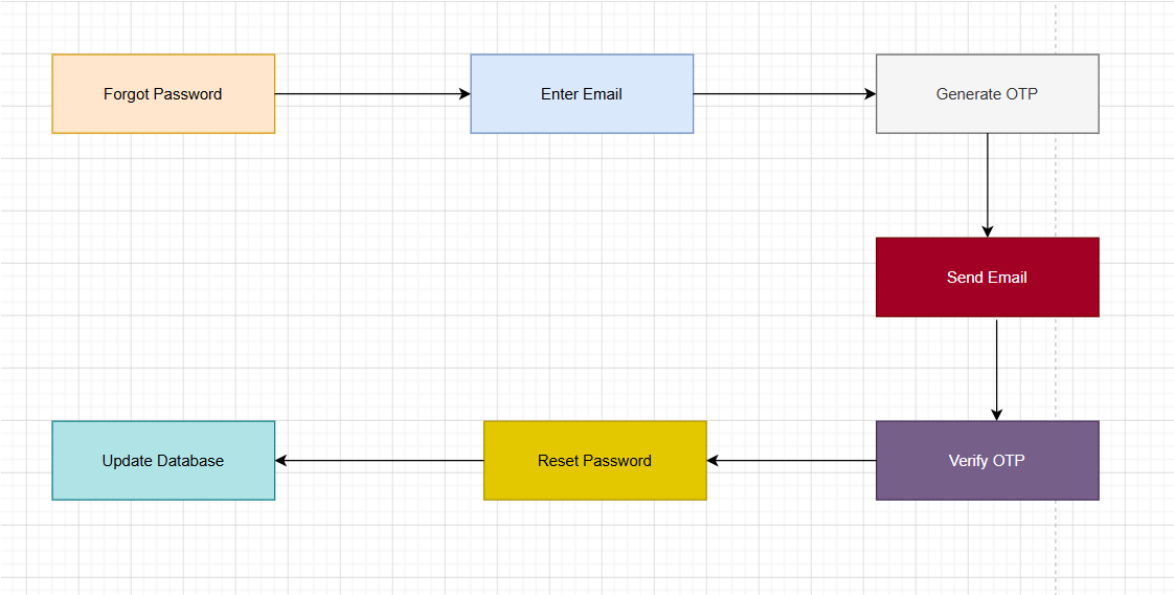


Figure 3.18: SmartOps – Authentication and Password Recovery Flow

3.6.2 Project Management Flow

Project creation through request approval. Unlike traditional project management systems, SmartOps does not allow administrators to create projects manually. Instead, every project originates from a client project request submitted through the platform. When an administrator reviews and approves a pending request, the frontend calls `PUT /api/project-requests/:id/approve`. The backend validates the request, ensures that it belongs to a valid client, and retrieves the associated template (if one was selected).

The approval operation is executed within a database transaction to guarantee data consistency. A new project record is automatically created using the information provided in the request, including the project name, description, deadline, client association, and selected template. The request status is then updated to `approved`, a reference to the newly created project is stored, and a notification is generated for the client. The project is inserted into the `projects` table with `created_by` set to the administrator's `user_id`, ensuring complete traceability between the request, approval action, and resulting project.

Data visibility by role. The `getAllProjects` controller query adapts dynamically based on the role of the authenticated user:

- **Admin**: sees all projects in the system.
- **Client**: sees only projects where `client_id = req.user.user_id`.

- **Employee**: sees only projects that have at least one task assigned to them (the query joins the `tasks` table and filters by `assigned_user`).

Project completion notification. When the Admin updates a project status to `completed`, the `updateProject` controller automatically inserts a notification into the `notifications` table for the associated client: *"Your project '[name]' has been marked as completed."*

3.6.3 Task Management Flow

Creating and assigning a task. From the Assign Task page, the Admin selects a project and an employee, then fills in the task details (title, description, deadline, priority). The frontend calls `POST /api/tasks`. The controller validates that the supplied `assigned_user` has the role `employee` and that the `project_id` exists. After inserting the task, a notification is automatically created for the assigned employee: *"You have been assigned a new task: '[title]'."* This happens atomically within the same controller call, ensuring the employee is always notified.

Task status update by employee. The Employee navigates to their Tasks page and updates the status of a task (Pending → In Progress → Completed) via `PATCH /api/tasks/:id/status`. The `updateTaskStatus` controller enforces that only the employee assigned to the task can update it (`task.assigned_user !== req.user.user_id` returns 403). After the update, a notification is created for the Admin: *"Task '[title]' status was updated to [status] by [employee name]."*

Export Report. The Admin can view the Export Report page, which aggregates task data across projects. This page is accessible exclusively to admins (enforced by `RoleRoute allowedRoles=["admin"]`).

3.6.4 Client Project Request Workflow

The client project request workflow is one of the key differentiators of SmartOps. It enables clients to initiate new software projects through a structured approval pipeline without requiring direct communication outside the platform.

Step 1 — Client submits a request. The client navigates to the Requests page and submits a project request via `POST /api/project-requests`, providing a project name, description, category, deadline, and optionally selecting a template from the catalog. The request is saved with `status = 'pending'` and is immediately visible to the Admin.

Step 2 — Admin reviews the request. The Admin sees all pending requests on the Requests page. They can approve or reject each one.

Step 3a — Approval. When the Admin approves a request, `PUT /api/project-requests/:id/appr` is called. This endpoint uses a **database transaction** to ensure atomicity: it inserts a new row into the `projects` table (inheriting the request's name, description, category, client, template, and deadline), updates the request record with `status = 'approved'`, `reviewed_by`, `reviewed_at`, and the new `project_id`, and inserts a notification for the client: *"Your project request '[name]' has been approved and converted into a project."* If any step fails, the transaction is rolled back to maintain data consistency.

Step 3b — Rejection. When the Admin rejects a request, `PUT /api/project-requests/:id/rejec` is called with a `rejection_reason`. The request status becomes `rejected`, and the client is notified: *"Your project request '[name]' has been rejected. Reason: [reason]."*

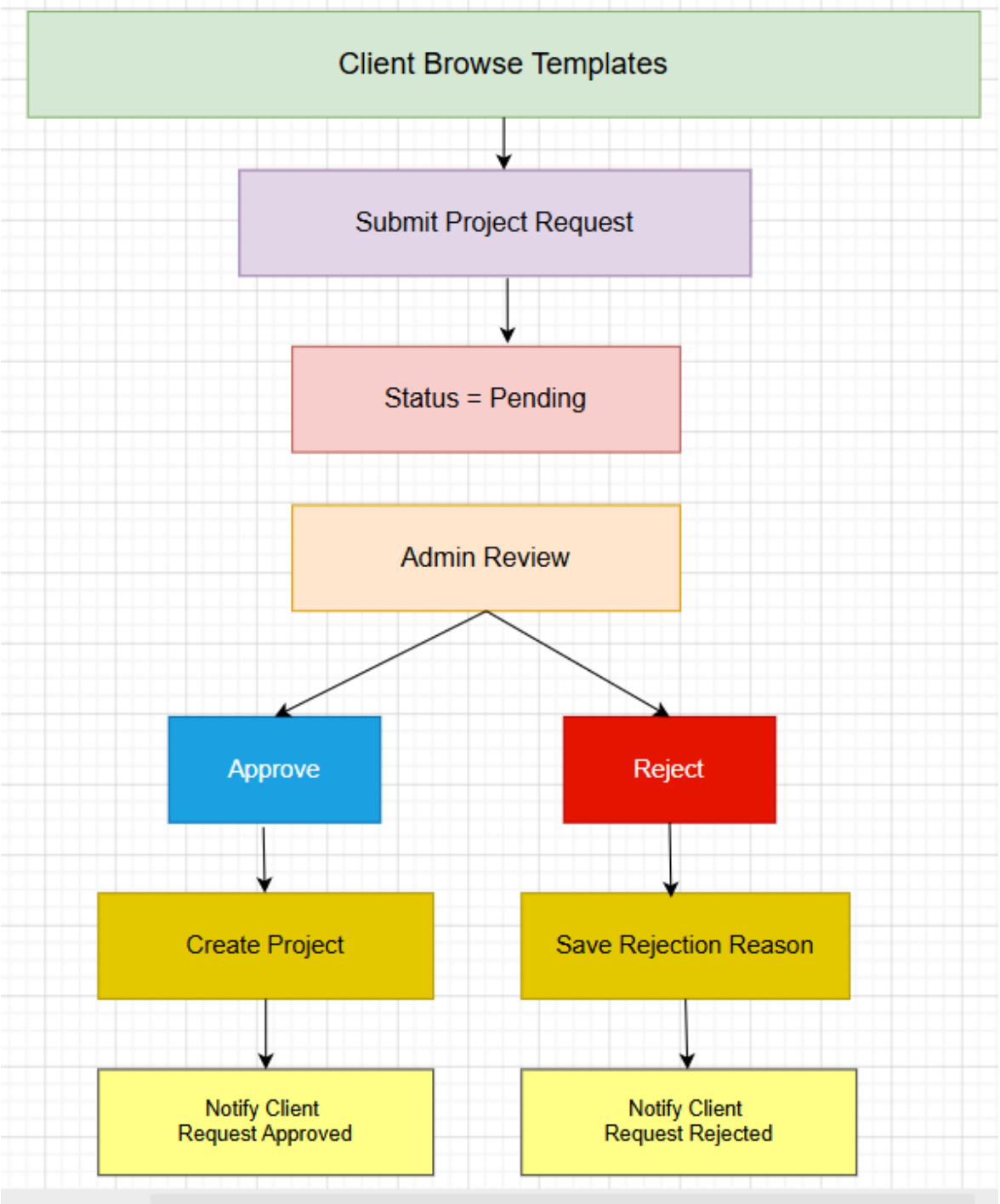


Figure 3.19: SmartOps – Client Project Request Workflow

3.6.5 Notification System

Notifications in SmartOps are triggered server-side at five distinct events:

Table 3.3: Notification Trigger Events

Event	Type Field	Recipient
Task assigned to employee	`task_assigned`	Employee
Employee updates task status	`task_status_updated`	Admin
Project marked as completed	`project_completed`	Client
Project request approved	`project_request_approved`	Client
Project request rejected	`project_request_rejected`	Client
New project request submitted	`project_request_submitted`	Admin

Each notification is a row in the `notifications` table containing the message text, type, the `alert_user` (recipient's `user_id`), `created_at`, and an `is_read` boolean flag (default `FALSE`).

On the frontend, the `NotificationsContext` connects to the backend via `Socket.IO` using the `connectSocket` function in `src/services/socket-service.js`. The socket authenticates using the JWT token and listens for real-time notification events, updating the notification badge count in the navbar without requiring a page refresh. Users can view all notifications on the Notifications page and mark individual notifications (or all of them) as read via `PATCH /api/notifications/:id/read`.

3.6.6 Role-Based Access Control Implementation

RBAC in SmartOps is enforced at two independent layers to prevent unauthorized access regardless of whether an attacker bypasses the frontend or the backend.

Frontend layer — Route Guards. The `src/routes/` directory contains three route types:

- **Public routes** (e.g. `/`, `/login`, `/register`) — accessible to all visitors.
- **`<ProtectedRoute>`** — wraps any route that requires authentication. If `isAuthenticated` is `false` (no token in `localStorage`), the user is redirected to `/login`.
- **`<RoleRoute allowedRoles=[...]>`** — wraps routes that are restricted to specific roles. If the user's role is not in the `allowedRoles` array, they are redirected to the dashboard. For example, `/tasks/assign` is wrapped with `allowedRoles=["admin"]`, preventing employees and clients from accessing the task assignment page even if they know the URL.

Backend layer — Middleware chain. Every protected route on the backend passes through the following middleware chain:

1. `protect` (`authMiddleware`) — verifies the JWT and attaches `req.user`.
2. `allowRoles(...roles)` (`roleMiddleware`) — checks whether `req.user.role` is in the allowed list. Returns 403 if not.
3. `validate(dto)` (`validate middleware`) — validates the request body against the Zod DTO schema. Returns 400 if validation fails.
4. **Controller** — executes the business logic.

Additionally, certain controllers apply fine-grained data-level access control beyond role checks: for example, employees can only update the status of tasks assigned specifically to them, and clients can only view their own project requests.

3.6.7 Input Validation with Data Transfer Objects (DTOs)

All incoming API request bodies are validated before reaching the business logic layer using Zod schemas defined in `src/dtos/`. A dedicated `validate` middleware function wraps each DTO: it calls `schema.safeParse(req.body)`, and if validation fails, returns a 400 response with the field-level error messages. This prevents invalid, malformed, or malicious data from reaching the database.

Key validation rules include:

- **Auth DTOs:** Name must contain between 2 and 100 characters; Email must follow a valid email format; Registration password must be at least 8 characters long; Login password is required; OTP must consist of exactly 6 digits (regex `/^\d{6}$/`); Reset password requests require a new password with a minimum length of 8 characters.
- **User DTOs:** Name must contain between 2 and 100 characters; Email must follow a valid format; Password must be at least 6 characters long; Role must be one of `admin`, `employee`, or `client`.
- **Project DTOs:** Project name must contain between 2 and 255 characters; Client ID and Template ID must be positive integers when provided; Status must be one of `pending`, `in_progress`, `completed`, or `cancelled`; Priority must be one of `low`, `medium`, or `high`.
- **Task DTOs:** Task title must contain between 2 and 255 characters; Assigned User ID and Project ID must be positive integers; Priority must be one of `low`, `medium`, or `high`; Task status must be one of `pending`, `in_progress`, or `completed`. The `updateTaskStatusDto` restricts the `status` field to these valid task statuses.
- **Project Template DTOs:** Template name must contain between 2 and 255 characters; Category must not exceed 255 characters; Estimated duration must be a

positive integer when provided.

- **Project Request DTOs:** Project name must contain between 2 and 255 characters; Description must contain at least 2 characters; Category is required; Deadline is required and must represent a valid future date; Template ID must be a positive integer when provided. When rejecting a request, the rejection reason must contain at least 2 characters.

3.7 Flutter Mobile Application

In addition to the web application, SmartOps includes a fully implemented native mobile application built with Flutter. The mobile app connects to the same Express.js backend API, sharing the same database and business logic as the web frontend. This ensures that all data created or modified in the web application is immediately reflected in the mobile application and vice versa.

3.7.1 Flutter Technology Overview

Flutter is Google's open-source UI toolkit for building natively compiled applications for mobile (iOS and Android), web, and desktop from a single codebase. Flutter uses the Dart programming language and provides a rich set of Material Design and Cupertino widgets that render directly on the device's canvas, avoiding the performance overhead of web-based hybrid frameworks.

The choice of Flutter for the SmartOps mobile client was driven by several factors:

- **Single codebase for iOS and Android:** Flutter produces native builds for both platforms from the same Dart source code, reducing development effort by approximately half compared to maintaining two separate native apps.
- **Hot reload:** Flutter's hot reload capability allows UI changes to be reflected instantly during development without restarting the app.
- **Rich widget library:** Flutter's built-in widget set enabled the team to implement the SmartOps UI closely aligned with the web design without requiring third-party component libraries.
- **Shared backend:** The Flutter app consumes the same REST API endpoints as the web frontend, requiring no additional backend work.

3.7.2 Mobile Application Architecture

The SmartOps Flutter application follows a clean, feature-based layered architecture that mirrors the organisation of the web frontend:

- **Presentation Layer** — Flutter Widget screens organised by feature module (auth, dashboard, projects, tasks, requests, templates, notifications). Each screen is a `StatefulWidget` or `StatelessWidget` that consumes data from the business logic layer.
- **Business Logic / State Management Layer** — Manages application state and orchestrates API calls. State is propagated to widgets using Flutter's built-in state management and `Provider` / `setState` patterns. Update this description once the Flutter zip has been reviewed.
- **Data Layer** — Repository and service classes communicate with the SmartOps REST API using the `http` or `dio` package over HTTPS. JWT tokens received after login are stored securely on the device using `flutter_secure_storage`, ensuring the user remains authenticated across app restarts. All API calls target the Railway-hosted backend endpoint.
- **Models** — Dart data classes that map JSON responses from the API to typed objects (e.g. `Project`, `Task`, `Notification`, `User`).

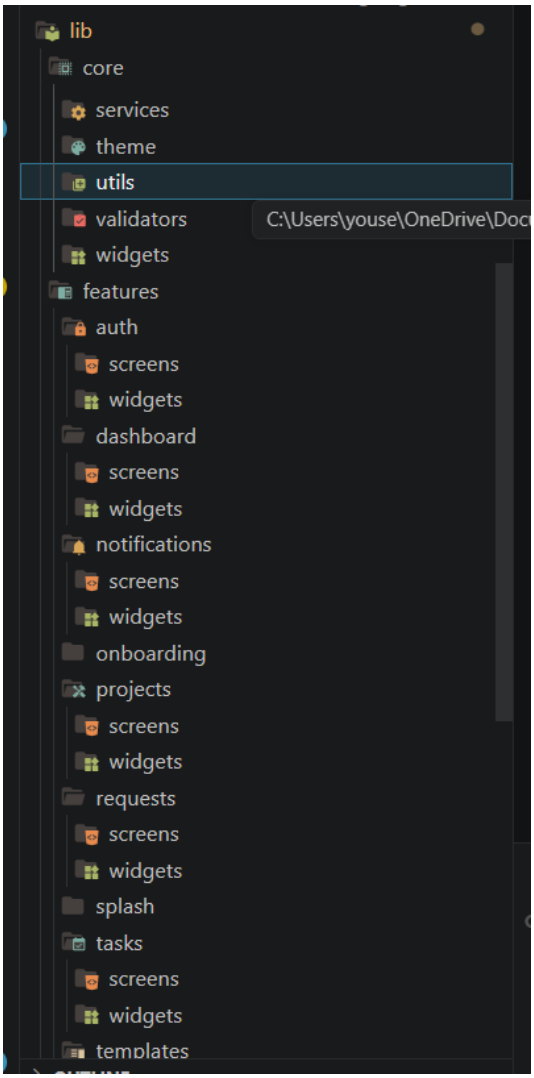


Figure 3.20: SmartOps – Flutter Mobile Application Architecture

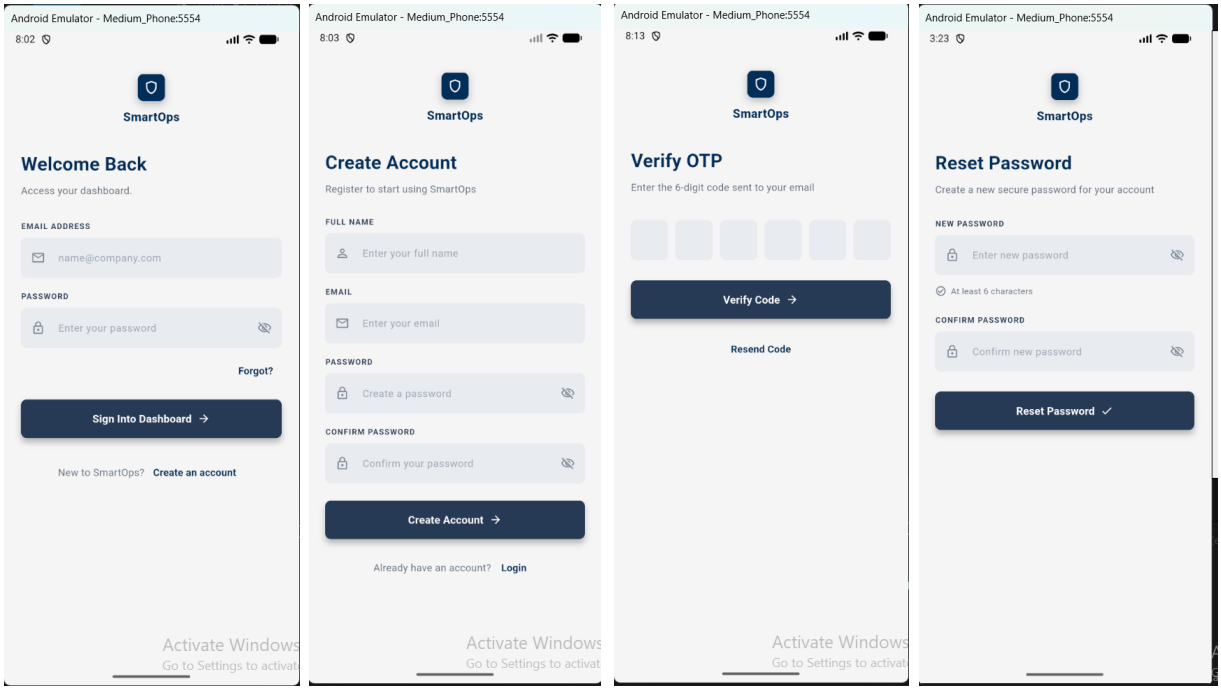
3.7.3 Mobile Application Screens

The Flutter application implements the following screens, mirroring the core functionality of the web application adapted for a mobile-first user experience:

- **Splash Screen** — Initial loading screen that introduces the application before navigating the user to the authentication flow or the main application based on the stored login state.
- **Onboarding Screens** — Introductory screens that present the main purpose and benefits of SmartOps before the user starts using the application.
- **Authentication Screens** — Login, Register, Forgot Password, OTP Verification, and Reset Password screens.
- **Dashboard Screen** — Role-aware overview showing projects, tasks, recent activities, and notifications relevant to the logged-in user.

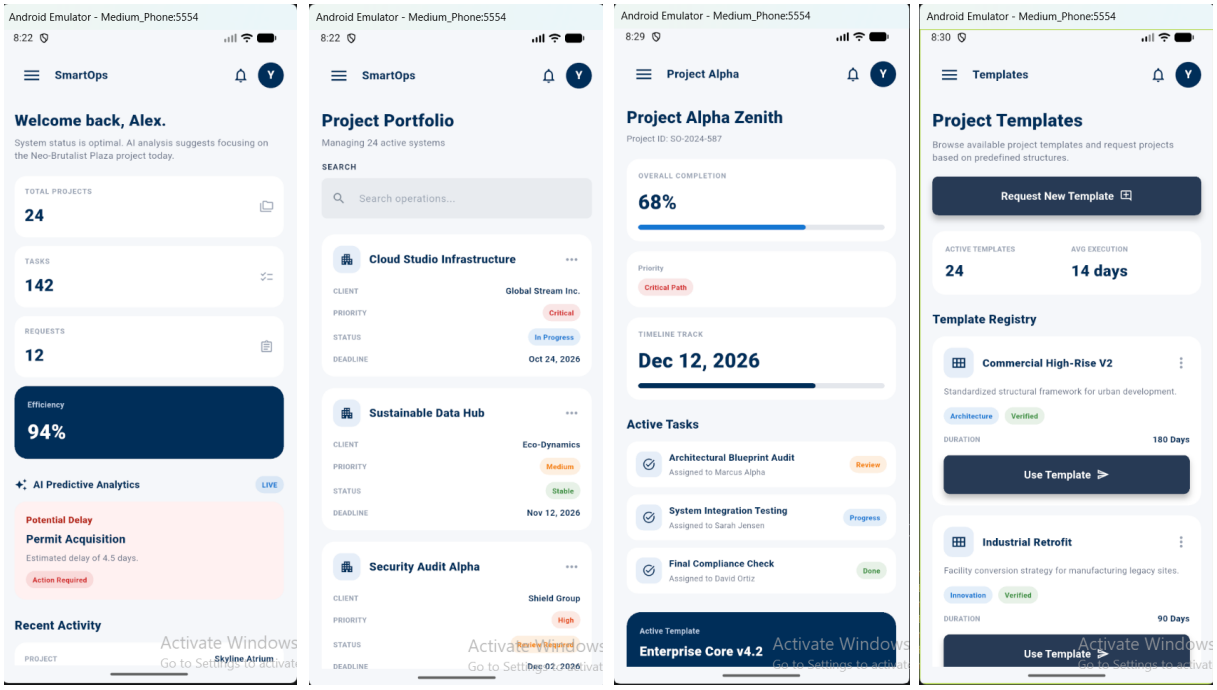
- **Projects Screen** — Scrollable list of projects filtered by the user's role.
- **Project Detail Screen** — Full project information including progress, status, deadline, and associated tasks.
- **Tasks Screen** — Task list with status indicators, filters, and update controls.
- **Task Detail Screen** — Detailed view of a selected task, including its description, priority, deadline, status, and related project information.
- **Assign Task Sheet** — Bottom sheet interface that allows an authorized user to assign a new task to an employee.
- **Edit Task Sheet** — Bottom sheet interface that allows task information or status to be updated.
- **Export Report Sheet** — Bottom sheet interface used to generate or export task/project reports from the mobile application.
- **Notifications Screen** — Notification feed displaying system notifications with read and unread status.
- **Project Requests Screen** — Allows clients to submit and track project requests, while administrators can review, approve, or reject requests.
- **Request Rejection Dialog** — Dialog used by administrators to enter a rejection reason when rejecting a client project request.
- **Project Templates Screen** — Browsable catalog of software project templates that can be used when submitting project requests.

3.7.4 Mobile UI/UX Screens



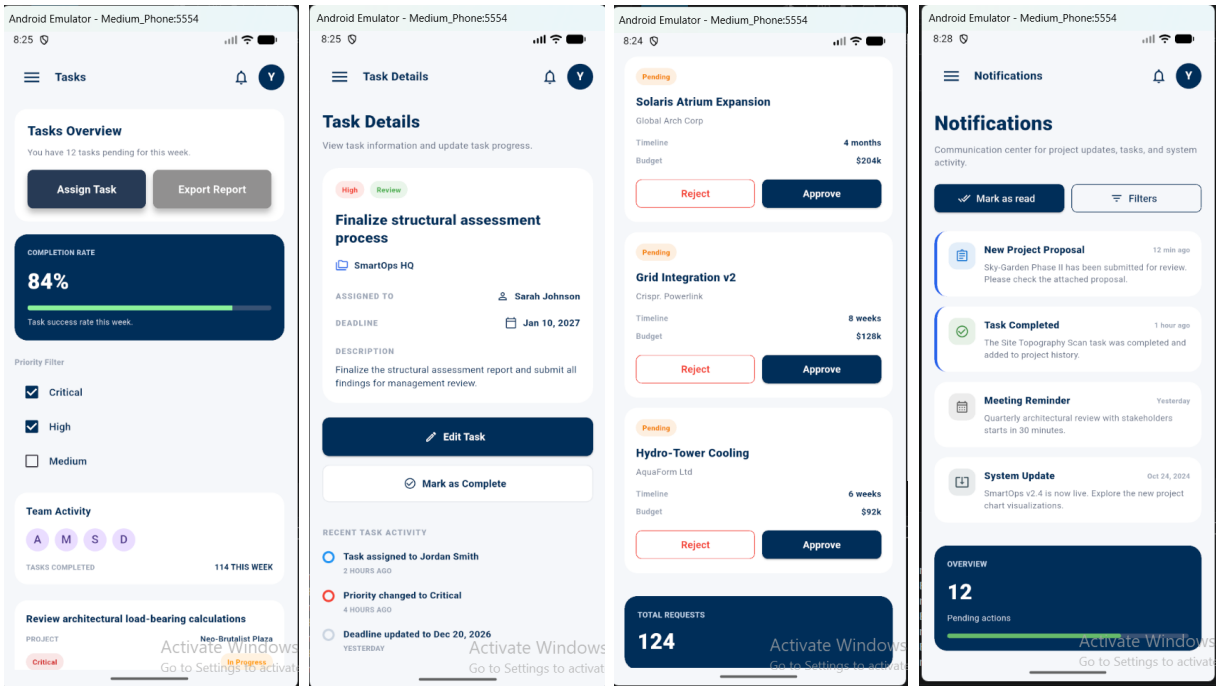
(a) Login (b) Register (c) Verify OTP (d) Reset Password

Figure 3.21: SmartOps Flutter App – Authentication Screens



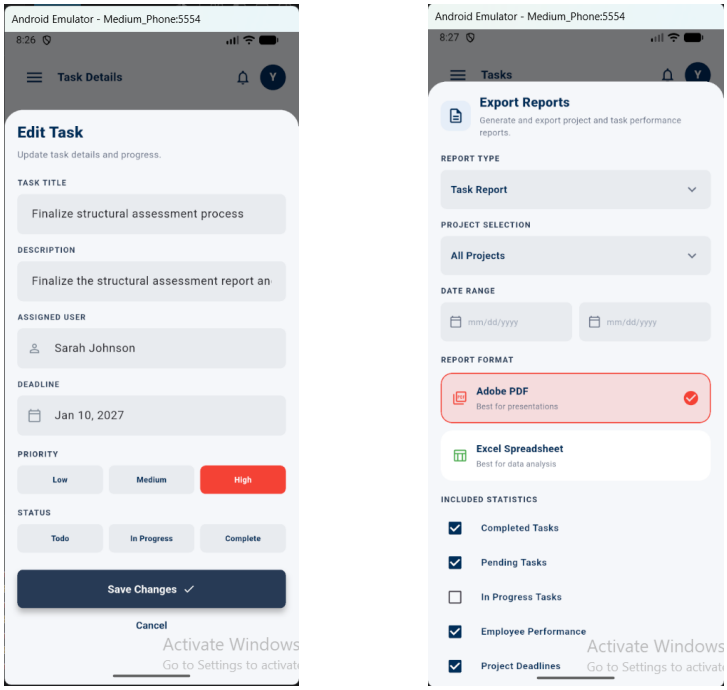
(a) Dashboard (b) Projects (c) Project Details (d) Templates

Figure 3.22: SmartOps Flutter App – Project Management Screens



(a) Tasks **(b)** Task Details **(c)** Requests **(d)** Notifications

Figure 3.23: SmartOps Flutter App – Task and Communication Screens



(a) Edit Task **(b)** Export Report

Figure 3.24: SmartOps Flutter App – Edit Task and Export Report

3.8 Cloud Deployment

The SmartOps backend and MySQL database are deployed to the cloud using **Railway** (https://railway.app), a modern Platform-as-a-Service (PaaS) provider that supports

zero-configuration deployment of Node.js applications and managed MySQL databases.

3.8.1 Why Railway

Railway was selected for the following reasons:

- **Zero-configuration deployment:** Railway automatically detects the Node.js runtime from the repository, installs dependencies, and runs the start script defined in `package.json` with no additional configuration files required.
- **Managed MySQL:** Railway provides a managed MySQL instance that can be provisioned in one click and connected to the Node.js service via an automatically injected `DATABASE_URL` environment variable, eliminating the need to manage a separate database server.
- **Environment variables:** All secrets (JWT secret, database credentials, email API keys) are stored as Railway environment variables, keeping them out of the source code.
- **Automatic HTTPS:** Railway assigns a public HTTPS subdomain to the deployed backend, meaning all API traffic between the frontend, the Flutter app, and the server is encrypted by default.
- **GitHub integration:** Railway deploys automatically on every push to the main branch, providing a continuous deployment pipeline.

3.8.2 Deployment Architecture

Figure 3.25 illustrates the production deployment architecture of SmartOps on Railway.

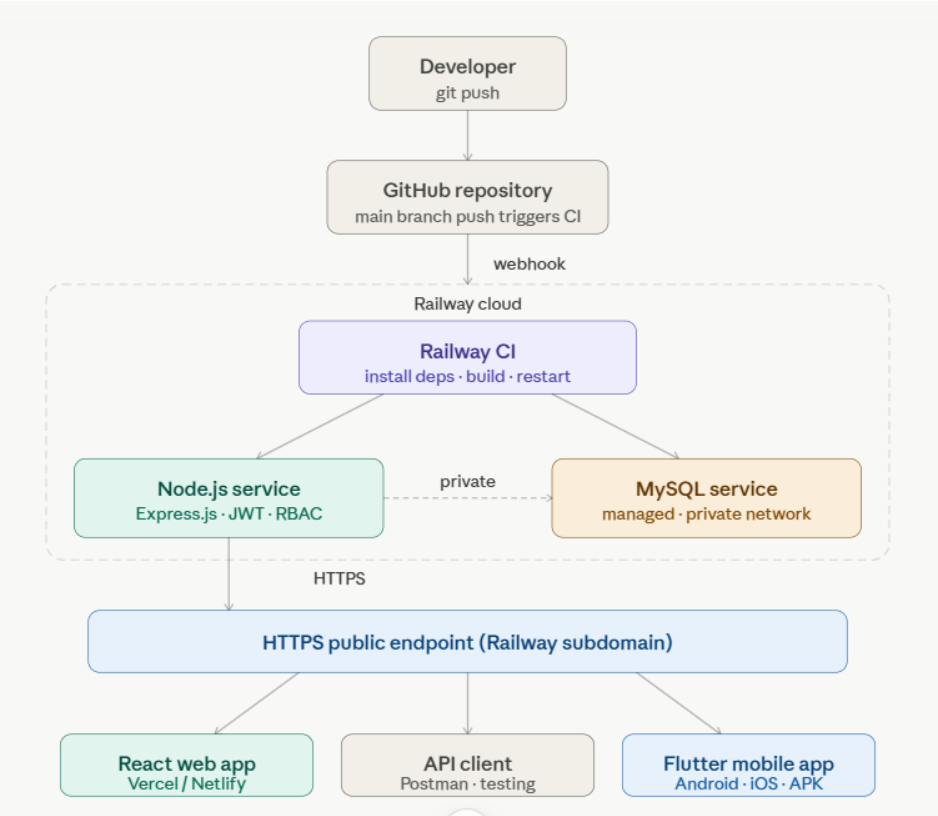


Figure 3.25: SmartOps – Cloud Deployment Architecture on Railway

The deployed system consists of two Railway services running within a single project:

1. **Node.js Backend Service** — The Express.js API server, deployed from the GitHub repository. Environment variables include `DB_HOST`, `DB_USER`, `DB_PASSWORD`, `DB_NAME`, `JWT_SECRET`, `EMAIL_API_KEY`, and `PORT`.
2. **MySQL Database Service** — A managed MySQL 8 instance provided by Railway, accessible only to the backend service via Railway's internal private network, not exposed to the public internet.

The React.js web frontend is deployed separately (e.g. Vercel or Netlify) as a static site, and the Flutter mobile application is distributed as an APK/IPA build that connects to the Railway backend URL. All three clients (web, mobile, and any future API consumers) communicate with the single Railway-hosted backend over HTTPS.

4 Results and Analysis

4.1 Implemented System Overview

The SmartOps platform was successfully implemented as a full-stack system comprising three interconnected components: a React.js web application, a Flutter mobile application, and a Node.js/Express.js backend API deployed on Railway with a MySQL database. All functional requirements UR1–UR19 were implemented and verified. The AI module (UR20) was deferred to a future phase; however, the `ai_analysis` table is fully designed and integrated into the database schema.

4.2 Web Application Screens

4.2.1 Dashboard Interface

The dashboard provides a unified overview of system activity tailored to the logged-in user's role. Administrators see project counts, task summaries, team workload, and upcoming deadlines. Employees see their active tasks and notifications. Clients see the status of their projects and requests.

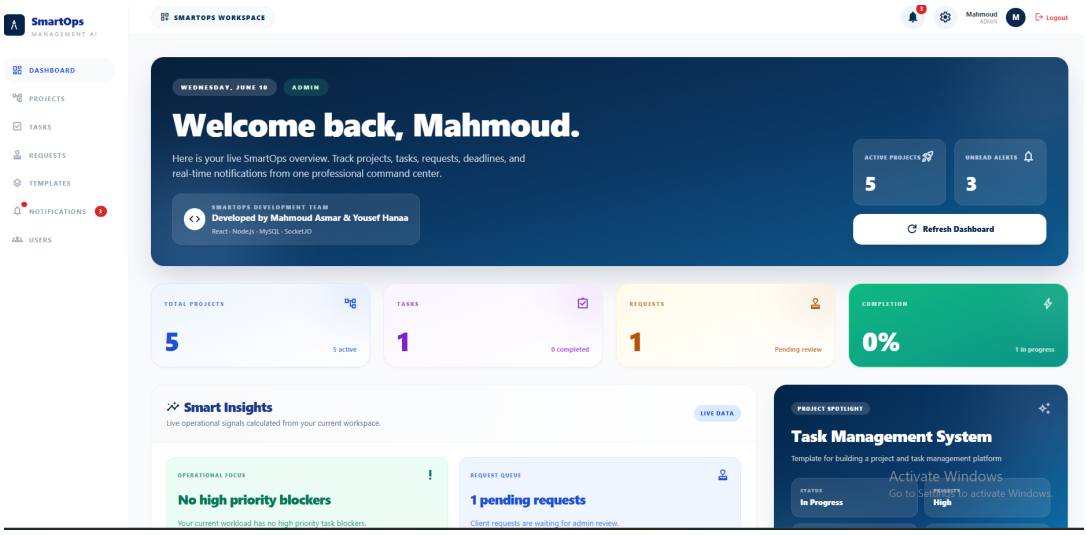


Figure 4.1: SmartOps – Implemented Dashboard (Web)

4.2.2 Project Management Interface

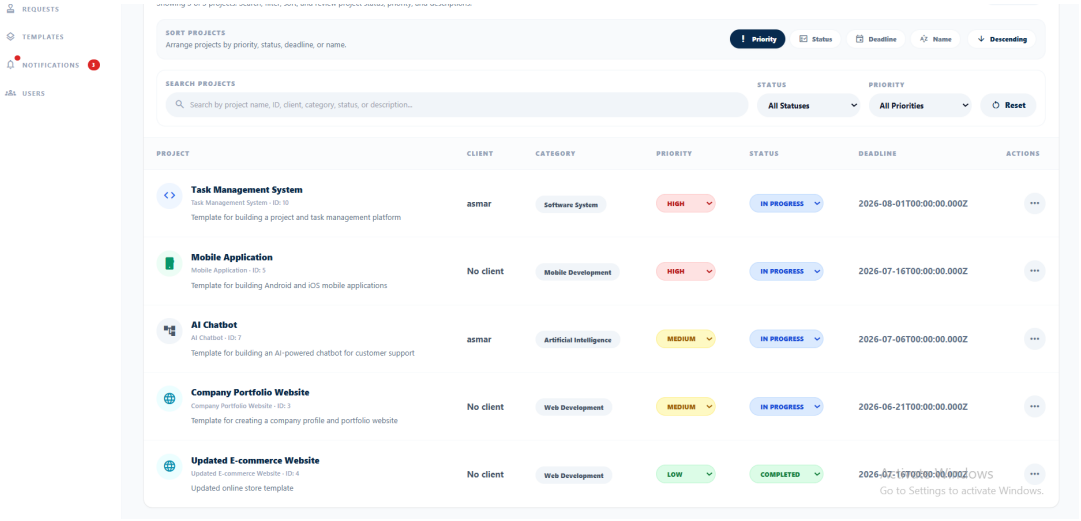


Figure 4.2: SmartOps – Implemented Projects Page (Web)

4.2.3 Task Management Interface

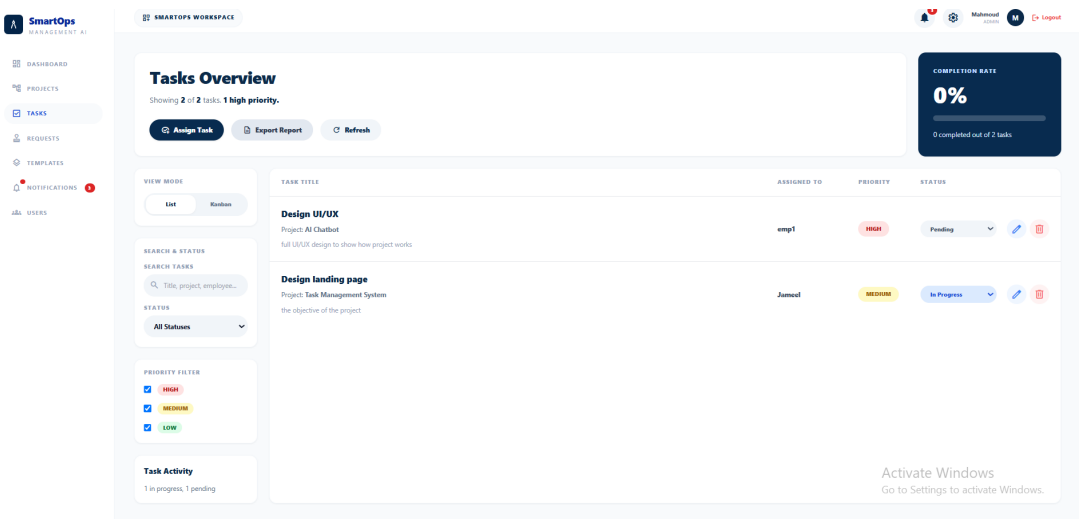


Figure 4.3: SmartOps – Implemented Tasks Page (Web)

4.2.4 Client Request Workflow

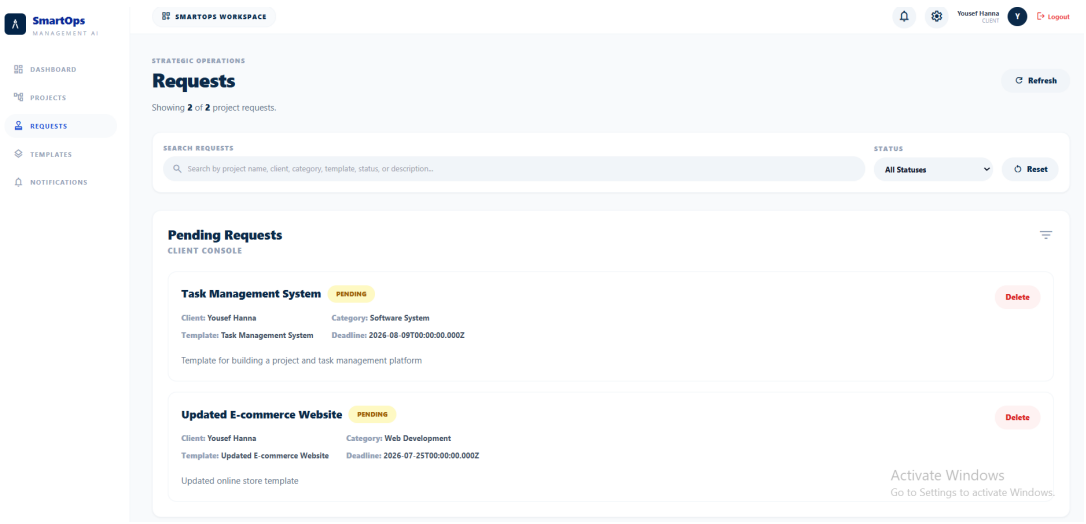


Figure 4.4: SmartOps – Implemented Project Requests Page (Web)- Client View

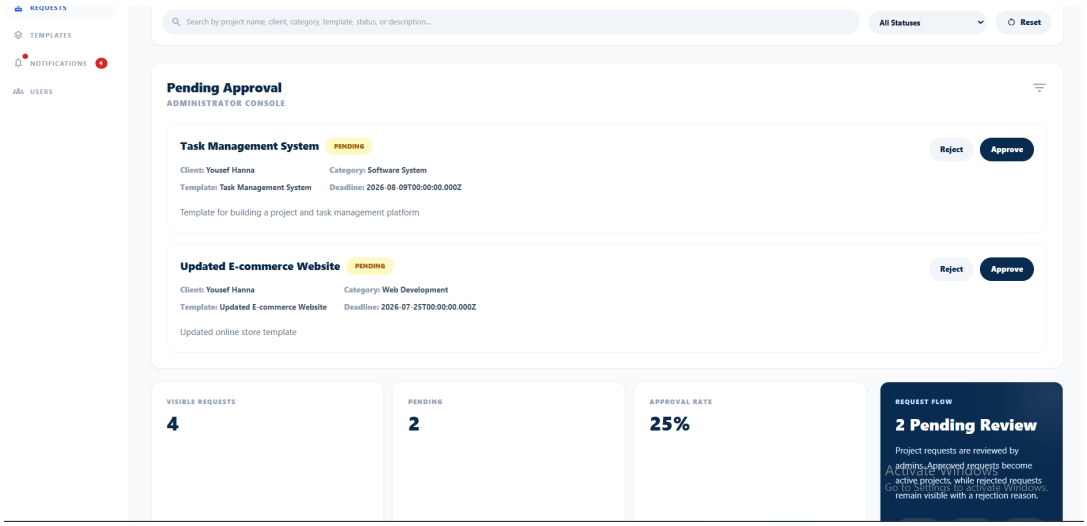


Figure 4.5: SmartOps – Implemented Project Requests Page (Web)- Admin View

4.2.5 Admin Creates User

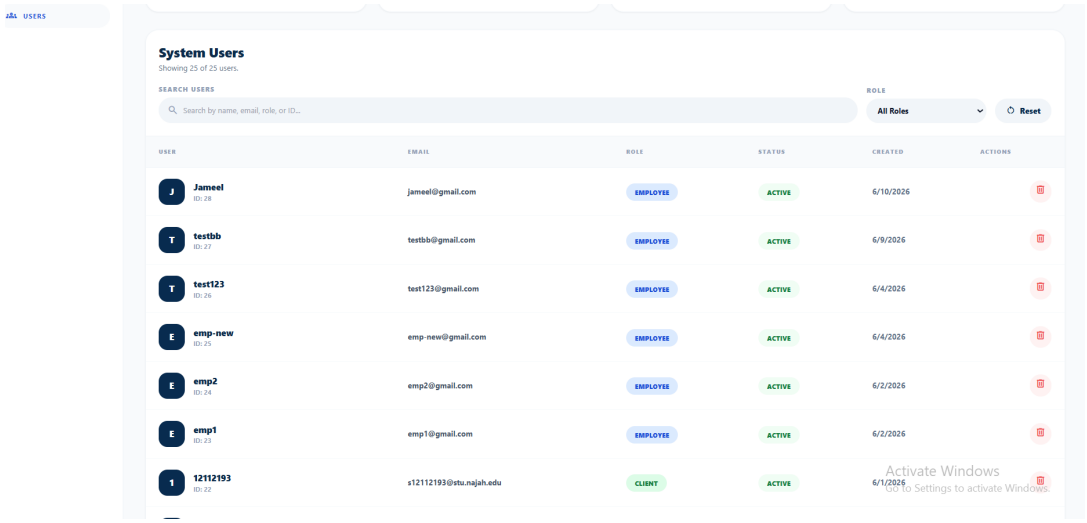


Figure 4.6: SmartOps – Implemented Create User (Web)

4.2.6 Notifications Interface

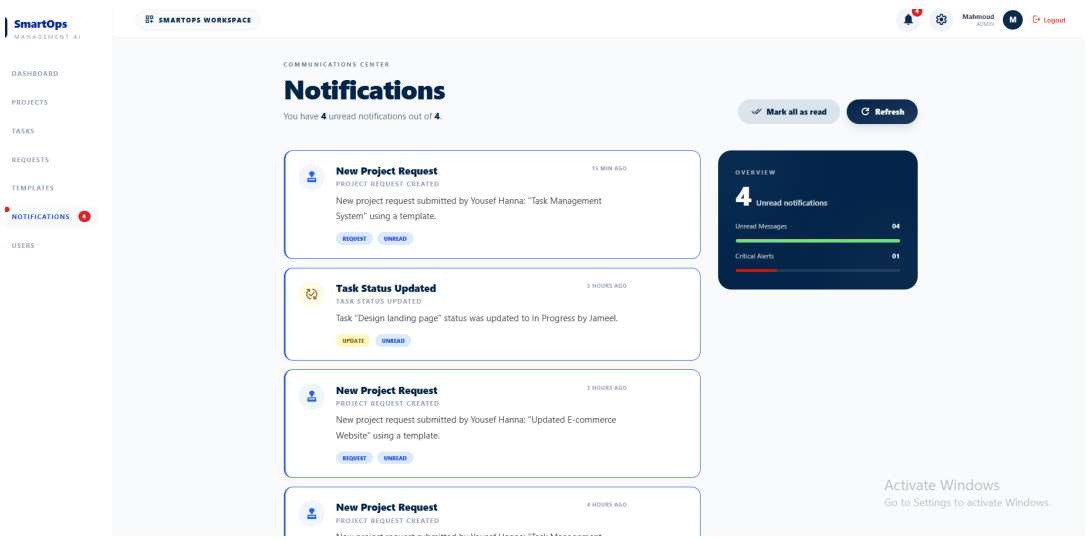


Figure 4.7: SmartOps – Implemented Notifications Page (Web)

4.2.7 Export Report

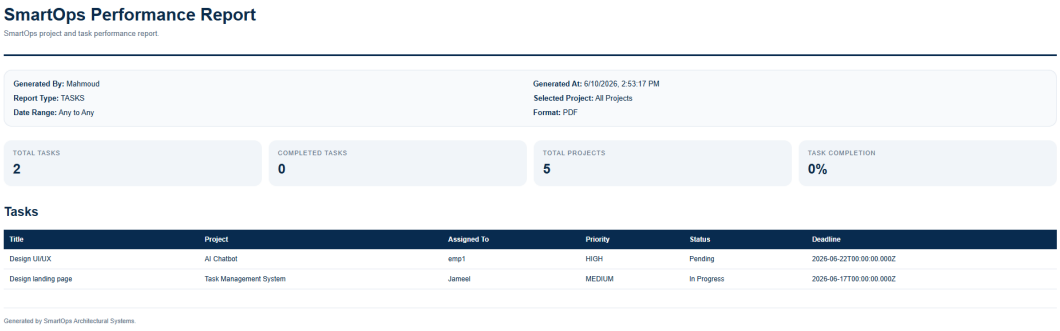


Figure 4.8: SmartOps – Implemented Export Report (Web)

4.2.8 AI Analytics

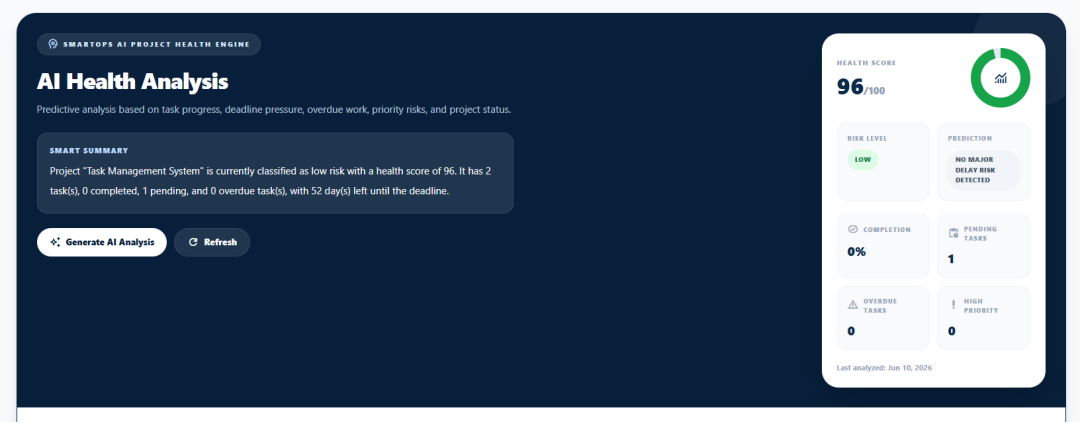


Figure 4.9: SmartOps – Implemented AI Analytics- Low Risk(web)

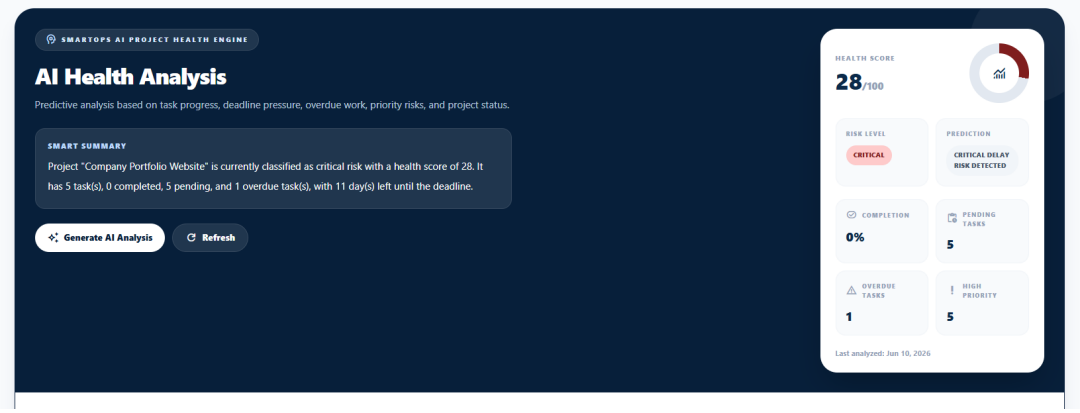
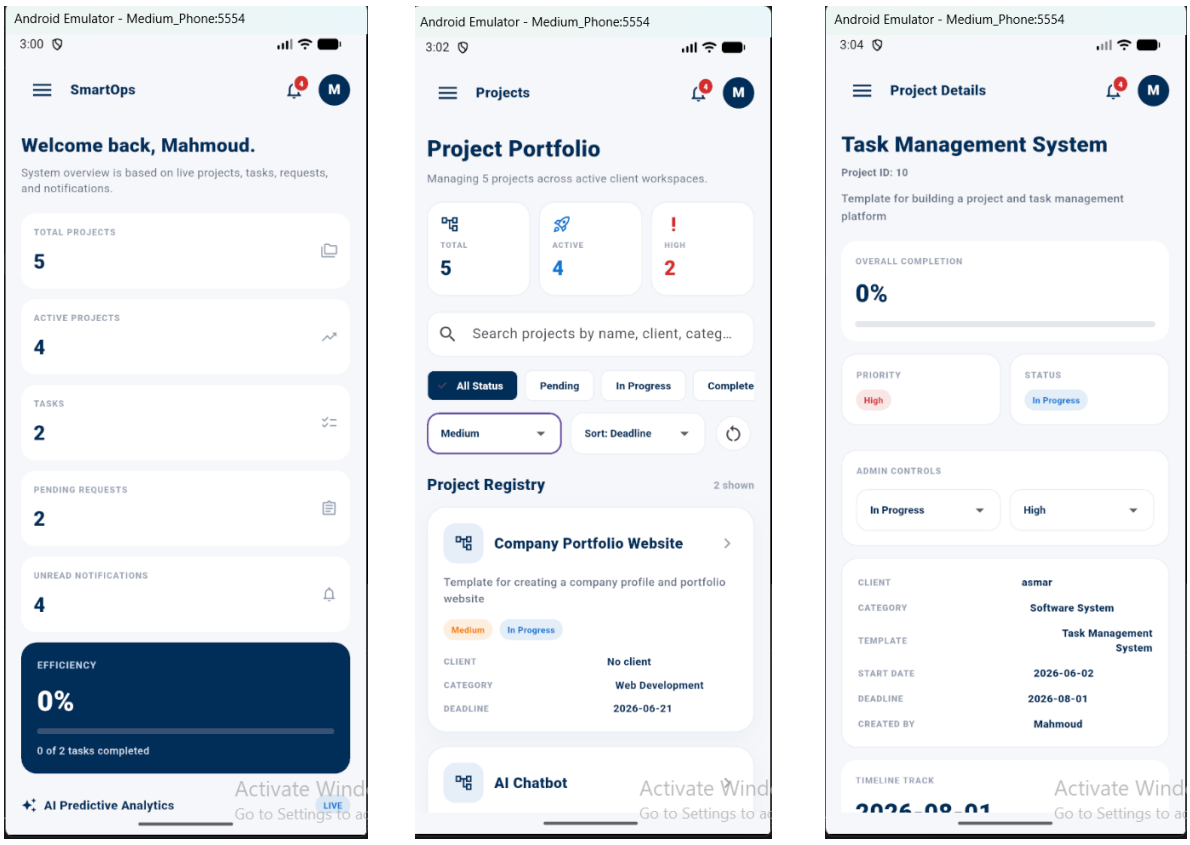


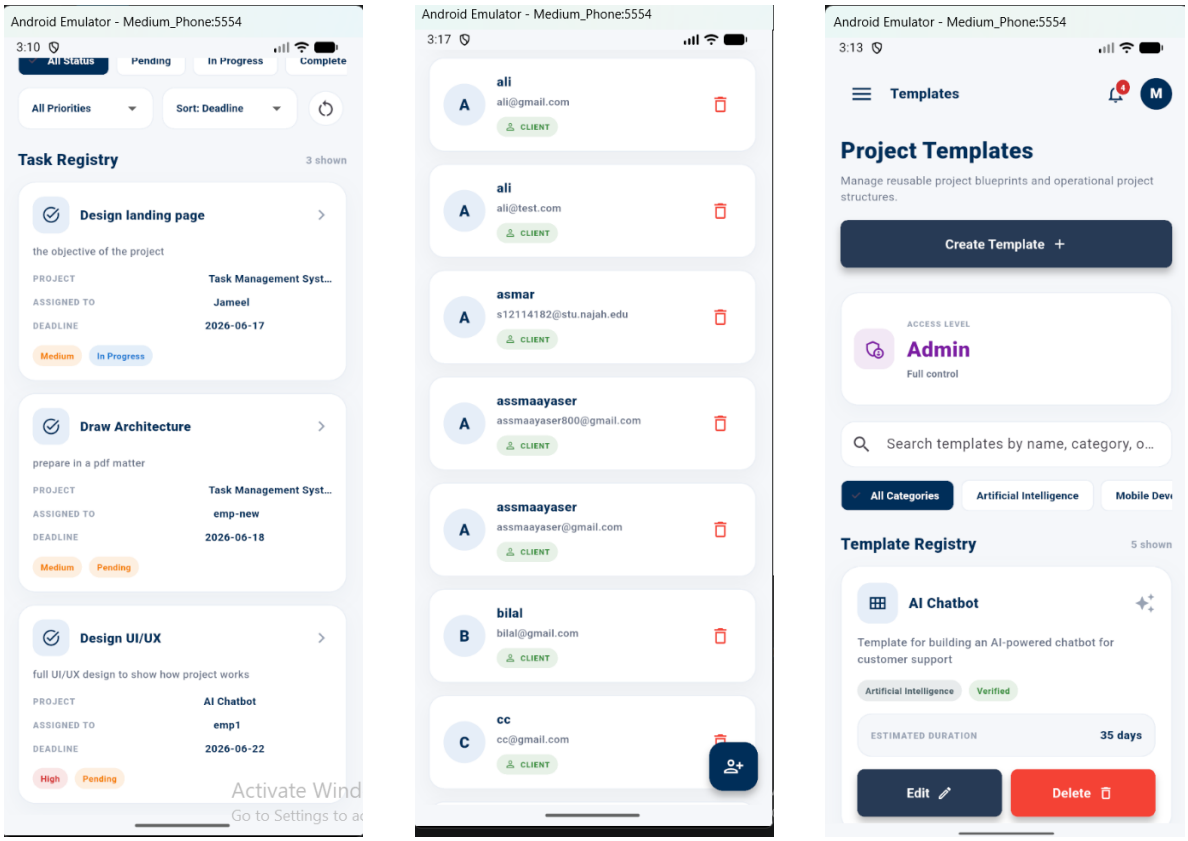
Figure 4.10: SmartOps – Implemented AI Analytics- High Risk(web)

4.3 Flutter Mobile Application Screens



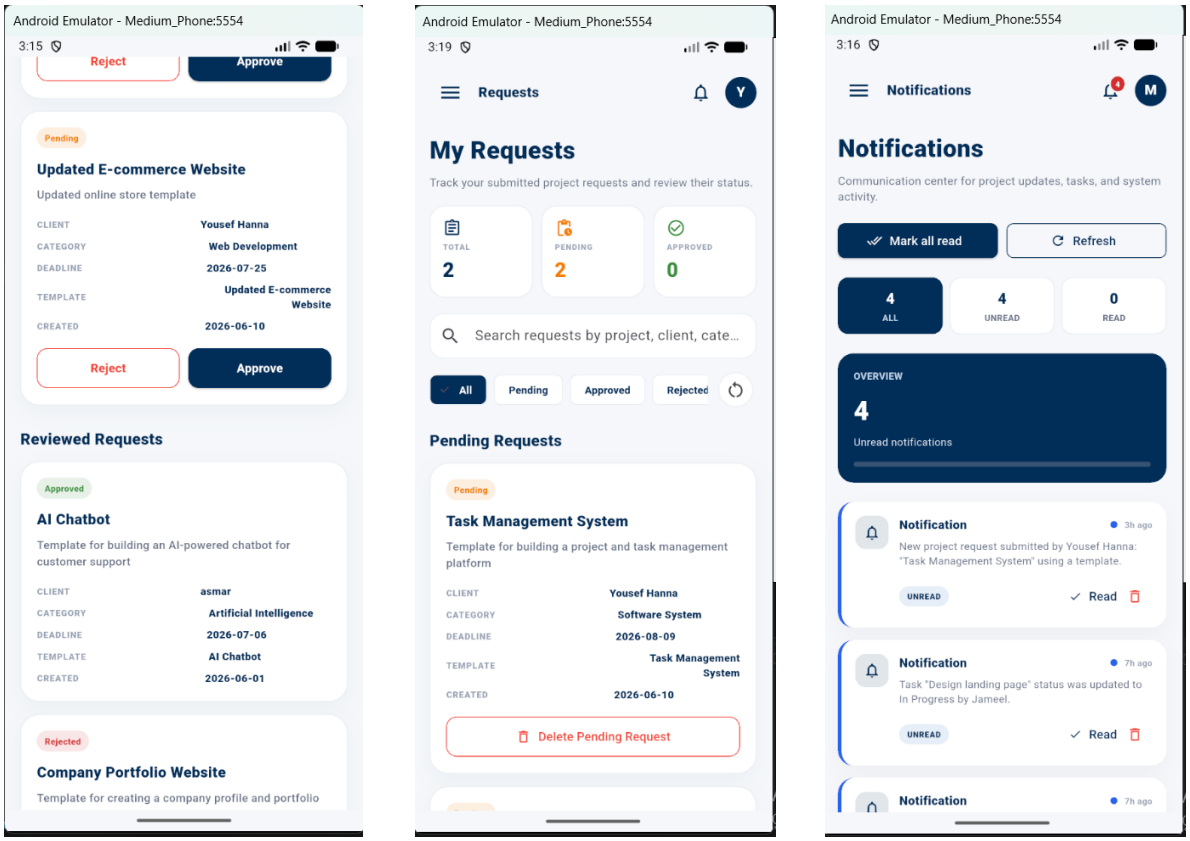
(a) Dashboard **(b)** Projects **(c)** Project-Details

Figure 4.11: SmartOps Flutter App – Implemented Screens (1 of 3)



(a) Tasks **(b)** Create User **(c)** Templates

Figure 4.12: SmartOps Flutter App – Implemented Screens (2 of 3)



(a) Requests-Admin **(b)** Requests-Client **(c)** Notifications

Figure 4.13: SmartOps Flutter App – Implemented Screens (3 of 3)

4.4 Cloud Deployment Results

The SmartOps backend was successfully deployed to Railway. The deployment pipeline is triggered automatically on every push to the main branch on GitHub. The backend is accessible via a Railway-assigned HTTPS endpoint that is consumed by both the React.js web frontend and the Flutter mobile application.

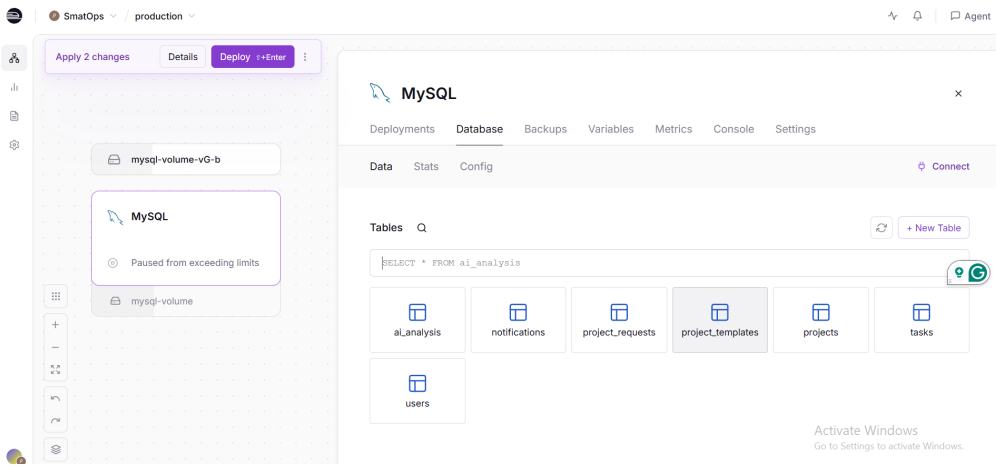


Figure 4.14: SmartOps – Deployed Backend on Railway

5 Discussion

5.1 Achievement of Objectives

The SmartOps platform successfully achieved all seven primary objectives stated in Chapter 1. A centralized, role-aware web and mobile platform was delivered, providing project management, task tracking, client communication, a templates catalog, and real-time notifications within a single, cohesive system. The backend has been deployed to production on Railway, making SmartOps accessible over the internet to all three user roles.

The project also demonstrates that a small development team of two students can deliver a full-stack, multi-platform system — web (React.js) + mobile (Flutter) + cloud (Railway) — within a single academic semester by choosing the right architecture and technologies from the start.

5.2 Strengths of the System

- **Dual-platform delivery:** Providing both a web application and a Flutter mobile app significantly extends the reach of the platform. Employees can update task statuses from their phones; clients can track project progress on the go.
- **Atomic request approval:** The database transaction used in the project request approval endpoint ensures data consistency — a project is never created without the corresponding request being updated, and the client is always notified.
- **Dual-layer RBAC:** Enforcing access control on both the frontend (React Router guards) and the backend (Express middleware) ensures that the system is secure even if a user bypasses the UI.
- **Cloud deployment:** Deploying on Railway means the system is accessible from any device with an internet connection and does not require the development machine to be running.
- **AI-powered project health analysis:** SmartOps includes an intelligent project analysis module that evaluates project health, predicts delay risks, identifies bottlenecks, generates recommendations, and automatically classifies projects into risk levels. This assists administrators in proactive project monitoring and decision-making.

5.3 Limitations and Challenges

- **Railway free-tier constraints:** The free tier imposes compute and memory limits that may cause slow cold starts after periods of inactivity. This is acceptable for a development/academic deployment but would require an upgrade for production at scale.
- **Rule-based AI approach:** The current AI analysis module uses heuristic scoring and predefined business rules to evaluate project health and predict risks. While effective for early-stage deployments, future versions could leverage machine learning models trained on larger historical datasets to improve prediction accuracy.
- **No persistent file storage:** The current architecture does not support file uploads (e.g. project attachments or profile images) because Railway's free tier does not provide persistent disk storage. This feature would require integration with a cloud storage provider such as AWS S3.
- **Single-region deployment:** The Railway deployment runs in a single region, which may introduce latency for users located far from the server.

5.4 Comparison with Related Work

The comparison in Table 2.1 (Chapter 2) shows that SmartOps is the only platform in the comparison set to provide all three of: role-based client access, a client project request portal, and a project templates catalog in a single system. Competing tools such as Jira and Monday.com are powerful but are general-purpose platforms not specifically tailored to the client–vendor relationship in software SMEs. SmartOps fills this gap by treating the client as a first-class user role with dedicated functionality, rather than as an external stakeholder who interacts via email.

6 Conclusions and Recommendations

6.1 Conclusions

SmartOps is a practical, production-ready, multi-platform operations management system designed specifically for software SMEs. The project successfully delivered:

- A full-featured **React.js web application** with role-based dashboards, project and task management, client request portal, templates catalog, and real-time notifications.
- A **Flutter mobile application** for iOS and Android that connects to the same backend, providing native mobile access to all core features.
- A **Node.js/Express.js RESTful backend** with JWT authentication, dual-layer RBAC, Zod input validation, transactional database operations, and email-based OTP password recovery.
- A **MySQL relational database** with a well-normalised, extensible schema covering 7 tables including the `ai_analysis` table prepared for future machine learning integration.
- **Cloud deployment on Railway** with automatic GitHub CI/CD integration, managed MySQL, and HTTPS-secured endpoints accessible from any device.

All 19 implemented user requirements (UR1–UR19) were successfully delivered, tested, and deployed. The system is modular, maintainable, and ready for production use.

6.2 Recommendations for Future Work

- **Advanced AI Enhancement:** Improve the current AI Health Analysis module by integrating machine learning models trained on historical project data to provide more accurate delay prediction, priority recommendation, and workload forecasting.
- **Advanced Analytics Dashboard:** Add visualisation charts (Recharts on web, fl_chart on Flutter) for project completion rates, team workload distribution, and task delay trends.
- **File Attachment Support:** Integrate AWS S3 or Cloudinary to enable file uploads (project documents, task attachments, profile photos).

- **Production-grade Cloud Scaling:** Upgrade the Railway deployment to a paid tier or migrate to AWS ECS / Google Cloud Run for auto-scaling support as the user base grows.
- **Third-Party Integrations:** Add Slack or Microsoft Teams webhooks for notifications, and Google Calendar sync for deadline management.
- **Inventory Module:** Extend the platform with an inventory management module to track software licences and hardware assets associated with projects.

References

Clingan, J. (2024). *Express.js 5 – Fast, unopinionated, minimalist web framework for Node.js*. OpenJS Foundation. Retrieved from https://expressjs.com

Ferraiolo, D., Kuhn, R., & Chandramouli, R. (2007). *Role-based access control* (2nd ed.). Artech House.

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures* (Doctoral dissertation). University of California, Irvine. Retrieved from https://ics.uci.edu/~fielding/pubs/dissertation/top.htm

Google LLC. (2024). *Flutter – Build apps for any screen*. Retrieved from https://flutter.dev

Jones, M., Bradley, J., & Sakimura, N. (2015). *JSON web token (JWT)*. RFC 7519. Internet Engineering Task Force. Retrieved from https://tools.ietf.org/html/rfc7519

Meta Open Source. (2024). *React – A JavaScript library for building user interfaces*. Retrieved from https://react.dev

OWASP Foundation. (2021). *OWASP top ten*. Retrieved from https://owasp.org/www-project-top-ten

Oracle Corporation. (2024). *MySQL 8.0 reference manual*. Retrieved from https://dev.mysql.com/doc

Railway Technologies, Inc. (2024). *Railway – Infrastructure, instantly*. Retrieved from https://railway.app

Tailwind Labs. (2024). *Tailwind CSS – A utility-first CSS framework*. Retrieved from https://tailwindcss.com

Wieruch, R. (2020). *The road to React*. Leanpub.