THOMAS MORE

INTERNSHIP

Realization Document

Zeeshan Mehmood
Applied Computer Science

2025 – 2026

Table of Contents

GLOSSARY

Abbreviation	Meaning
AD	Active Directory
API	Application Programming Interface
ARM	Azure Resource Manager
ASP.NET Core	Microsoft web application framework used to build web applications
Azure AD	Azure Active Directory, now Microsoft Entra ID
Bicep	Azure Infrastructure as Code language
CDN	Content Delivery Network
CI/CD	Continuous Integration and Continuous Deployment
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
DB	Database
Dev	Development
EF Core	Entity Framework Core
Entra ID	Microsoft Entra ID, Microsoft's cloud identity platform
Env	Environment
ERD	Entity Relationship Diagram
HTML	Hyper Text Markup Language
IaC	Infrastructure as Code
KPI	Key Performance Indicator
LIMS	Laboratory Information Management System
LINQ	Language-Integrated Query
MoSCoW	Must Have, Should Have, Could Have, and Won't Have
MS	Microsoft
MVC	Model-View-Controller
MVP	Minimum Viable Product
OIDC	OpenID Connect
PRD	Production
QA	Quality Assurance
R&D	Research and Development
RBAC	Role-Based Access Control

Abbreviation	Meaning
SignalR	ASP.NET Core library for real-time communication
SMTP	Simple Mail Transfer Protocol
SPA	Single Page Application
SQL	Structured Query Language
UI	User Interface
UX	User Experience

1. Introduction

This realization document explains how the Visit Request Portal was designed and developed as an internal web application during the 2026 internship at Soudal NV. The document builds on the Internship Project Plan and describes how the project evolved from requirement analysis and solution design to implementation, cloud deployment and final evaluation. Each phase of the development process is documented to provide a clear overview of the realized system and the technical and architectural decisions that were made.

1.1. Company context

Soudal NV is a Belgian family-owned multinational company headquartered in Turnhout and recognized as one of the world's leading manufacturers of sealants, adhesives, and polyurethane foams. The company operates production sites and sales branches in many countries and invests continuously in innovation, sustainability, and digitalization to support its global operations. The Visit Request Portal contributes to this digital transformation by replacing an informal, email-based visitor planning process with a structured, centralized web application.

1.2. Personal Role and Responsibilities

The objective of the internship was to apply theoretical knowledge from the Bachelor of Applied Computer Science in a real industrial environment and to deliver a practical software solution for a concrete business problem.

The internship assignment covered the complete realization of the Visit Request Portal application, from requirements analysis and design to implementation, testing, deployment, and documentation. The responsibilities covered multiple stages of the software development lifecycle:

- Requirement analysis with business stakeholders and end users.
- Creation of use case diagram and detailed use case descriptions for requestors, admins, operations and managers.
- Design of Figma UI prototypes for role-specific workflows.
- Database modelling and creation of an Entity Relationship Diagram (ERD) for core entities
- Implementation of the ASP.NET Core 8 Razor Pages application, including business logic, validation rules, and real-time updates with SignalR.
- Integration with Azure Active Directory for authentication and role-based authorization.
- Implementation of email notifications using MS Graph.
- Setup of Infrastructure as Code using Azure Bicep and configuration of Azure DevOps CI/CD pipelines for Dev/QA/Prd environments.
- Execution of functional and non-functional testing, bug fixing, performance tuning, and preparation for production deployment.
- Creation of technical documentation and this realization document.

- In addition to the main assignment, an MVP application for the R&D department was developed and deployed. This application was not part of the original internship assignment but was completed as additional work during the internship.

1.3. Project Background and Objective

Within Soudal, visitor planning and scheduling were traditionally handled through informal methods such as email communication and personal calendars. While this approach works for small volumes of visits, it becomes inefficient when the number of visitors increases and multiple departments are involved.

The absence of a centralized system created several operational challenges. First, there was no shared overview of planned visits, making it difficult to coordinate schedules between departments. Second, administrators had limited control over visitor capacity and could not easily block certain time periods when visits should not be allowed. Third, management had limited visibility into visitor activity and could not easily consult historical records or analyze visit statistics.

To address these issues, Soudal required an internal web-based application that centralizes visitor scheduling and introduces a structured approval workflow.

The goal of this project was therefore to design and develop a secure internal application called the Visit Request Portal. The system allows employees to request visits through a calendar interface, while administrators review and approve or reject these requests.

The application is integrated with Microsoft Azure Active Directory for authentication and is designed to operate within the company's internal Microsoft ecosystem.

1.4. Document Structure

This realization document is organized into five main chapters, followed by a reference list and attachments.

- **Chapter 1 – Introduction**
  Describes the company context, personal role and responsibilities, the project background, and the objectives of the Visit Request Portal.
- **Chapter 2 – Analysis**
  Explains the analysis of the existing situation, requirements, and constraints, compares possible solution approaches and technologies, and motivates the chosen architecture and tool stack.
- **Chapter 3 – Realization**
  Presents the implemented solution, system architecture, database model, core functionalities, user workflows, infrastructure as code setup, CI/CD pipeline, authentication, notification system, and deployment. It also briefly discusses important design patterns and technical decisions.

- **Chapter 4 – Additional Work**
  Describes the additional MVP application developed for the R&D department after completion of the main internship assignment. This chapter explains the objective, architecture, implemented features, current MVP status, and future improvements of the R&D Formulation Database.
- **Chapter 5 – Conclusion**
  Evaluates the result of the main Visit Request Portal project, reflects on the additional R&D application, summarizes key learnings, and formulates recommendations for future development.

The reference list contains the written and spoken sources consulted during the project, while the attachments include supporting materials such as project plan and user manuals.

2. Analysis

This chapter explains how the problem domain was analyzed, which requirements were collected, which architectural and technical options were considered, and why the final choices were selected.

2.1. Requirements Analysis

The project originated from the assignment “Event Tool Soudal”, which defines the need for an internal visitor planning and management system with a calendar-based booking solution. Through multiple meetings with internal stakeholders, the problem was further refined and detailed requirements were gathered.

Based on these requirements, a comprehensive use case diagram was created to capture all functionalities of the application. Subsequently, Figma prototypes were developed for each use case to visualize the system. These prototypes were reviewed with the client, and the necessary improvements were implemented based on the feedback received.

2.1.1. Functional Requirements

The functional requirements were analyzed and translated into a use case diagram covering the main system workflows.

2.1.1.1. Use Case Diagram

A use case diagram is like a simple picture that shows who uses a system and what they can do with it. Imagine drawing a stick figure (the user) connected by lines to bubbles (the tasks), all inside a box (the system) — that's essentially it.

A use case is a single task or action a user can perform, like "Login", "Place an Order", or "Generate a Report." It describes what the system should do, not how it does it technically. Together, they help teams — developers, managers, and clients — agree on what the software needs to do before writing a single line of code, like a blueprint before building a house.

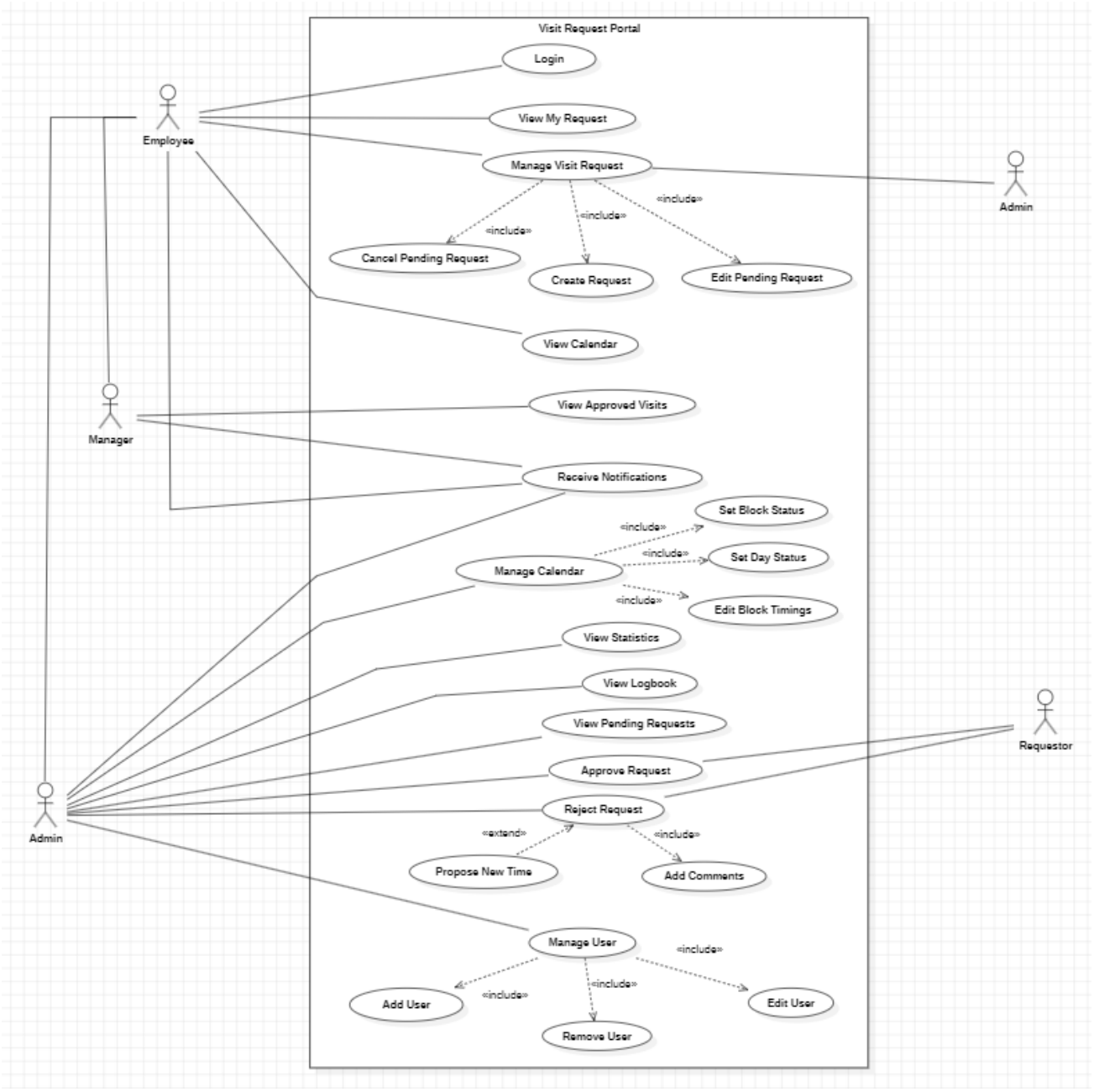


Figure 1: Use Case Diagram

2.1.1.2. Use Case Description

- **Login**
  As an internal employee of the company, I can log in to the application using my Microsoft Azure AD account via the company SharePoint portal.

- **View My Request**
  As an internal employee of the company, I can view all the visit requests that I created. Each request must display its current status and all related details.
- **Manage Visit Request**
  As an internal employee of the company, I can manage my visit requests, including creating, editing, or cancelling them when applicable.
- **Create Request**
  As an internal employee of the company, I can create a new visit request by providing all required details such as visitor information, number of people, date, time, and responsible person.
- **Edit Pending Request**
  As an internal employee of the company, I can edit a visit request only when its status is pending approval.
- **Cancel Pending Request**
  As an internal employee of the company, I can cancel a visit request while it is still pending, or it has been approved.
- **View Calendar**
  As an internal employee of the company, I can view the calendar to check availability and number of existing visit schedules per block of each day.
- **View Approved Visits**
  As a manager, I can view approved visit requests to understand who is visiting and when. Each approved request must display all related details.
- **Receive Notifications**
  As a user of the system, I can receive notifications regarding request creation, updates, approvals, rejections, or cancellations.
- **Manage Calendar**
  As an admin, I can manage the calendar by controlling availability and scheduling rules.
- **Set Block Status**
  As an admin, I can block specific time slots in the calendar to prevent new visit requests. I must be able to set any block status to available, busy or blocked and can provide comments for it.
- **Set Day Status**
  As an admin, I can mark an entire day as unavailable for visit requests.
- **Edit Block Timings**
  As an admin, I can change the timings of morning/afternoon/evening blocks
- **View Statistics**
  As an admin, I can view statistics such as number of visits, peak times, and visitor trends per year.
- **View Logbook**
  As an admin, I can view historical records of all visits for tracking and auditing purposes.
- **View Pending Requests**
  As an admin, I can view all pending visit requests that require approval or rejection.
- **Approve Request**
  As an admin, I can approve a visit request, allowing it to be scheduled in the calendar.

- **Reject Request**
  As an admin, I can reject a visit request by providing a reason for rejection.
- **Add Comments**
  As an admin, I can add comments when rejecting a request to inform the requestor.
- **Propose New Time**
  As an admin, I can propose alternative timings when rejecting a request.
- **Manage User**
  As an admin, I can manage users within the system, including assigning roles and permissions.
- **Add User**
  As an admin, I can assign role of manager or admin to any employee of the company.
- **Edit User**
  As an admin, I can edit role of any employee of the company.
- **Remove User**
  As an admin, I can remove the user account form the application.

2.1.2. Non-Functional Requirements

Important non-functional requirements:

- Secure authentication using Azure Active Directory (Microsoft Entra ID).
- Role-based authorization in the application.
- Clear and intuitive user interface in English.
- Reliable data persistence in Azure SQL database.
- Scalability with separate Dev, QA, and Production environments.
- Automated deployment via CI/CD pipeline in Azure DevOps.
- Use of Infrastructure as code to define and deploy Azure resources.

2.2. Feature Prioritization

To ensure efficient use of time and resources, the system requirements were prioritized using the MoSCoW method. This method classifies features into Must Have, Should Have, Could Have, and Won't Have categories. This prioritization helped define the Minimum Viable Product (MVP) and ensured that critical functionality was delivered within the internship timeframe.

Priority	Features
Must Have	User authentication, create visit request, edit pending requests, cancel request, admin approval/rejection workflow, calendar view, role-based access (Employee/Admin/Manager), admin calendar management (block days/time slots) and Managers to view all approved visits.
Should Have	Reporting & statistics, email notifications, user dashboard view, proposed alternative timing for rejected requests and option to fill checklist after approval of visit request.
Could Have	Real-time-in-app notifications, advanced reporting visuals, country-based reporting, enhanced filtering and search.
Won't Have	Public visitor portal, mobile application, excel template tracking/upload.

Table 1: MoSCoW

The "Must Have" features define the core functionality required for the system to operate effectively. "Should Have" and "Could Have" features enhance usability and user experience but were prioritized based on available time. "Won't Have" features were intentionally excluded from the project scope.

2.3. Figma Prototypes

To validate the system design and user flows, interactive Figma prototypes were created. These prototypes enabled early feedback from stakeholders and helped refine the application requirements before implementation.

Prototype link: **Figma Prototype Link** (For demo purposes, use any email and password of minimum 6 characters to login)

2.4. Technology Stack

Several implementation approaches and technology stacks were considered.

2.4.1. Weighted Ranking Method

Criteria	Weights	ASP.NET Core Razor Pages	ASP.NET MVC	ASP.NET Web API + Angular
Required by client	10	8	5	5
Implementation simplicity	5	4	3	2
Scalability	7	3	4	6
Security	8	7	6	5
Total	30	177	141	142

Table 2: Weighted Ranking Method

Three main directions were examined:

- A traditional ASP.NET MVC web application (server-rendered views).

- A Single Page Application with a fully decoupled Angular frontend and ASP.NET Web API backend.
- ASP.NET Core Razor Pages with server-side rendering, enhanced by small JavaScript components and SignalR for real-time features.

Given the scope, timeline, and existing expertise at Soudal, the Razor Pages approach was chosen:

- It fits well with form-heavy, workflow-oriented business applications.
- It reduces complexity compared to a full Single Page Application (SPA) and simplifies server-side validation and security.
- It integrates naturally with ASP.NET Core authentication, authorization, and Entity Framework Core.

2.5. Infrastructure

For infrastructure, the main options were manual portal-based provisioning, Azure Resource Manager (ARM) JSON templates or Bicep definitions.

Manual provisioning was rejected because it is hard to reproduce across environments and error-prone. ARM JSON templates are powerful but verbose and less readable. Bicep was chosen because it provides a concise, type-safe domain-specific language for Azure resources and supports modularization and parameterization.

The final infrastructure architecture includes:

- Azure App Service Plan and Web App for hosting the portal.
- Azure SQL Server and Database for application data.
- Azure Key Vault for secrets (connection strings, env variable, client secrets).
- Application Insights and Log Analytics for monitoring.
- System-assigned managed identity for secure access to Key Vault.

2.6. Deployment

For deployment, several options were considered, including GitHub Actions, GitLab CI/CD and Azure DevOps pipelines.

GitHub Actions and GitLab CI/CD are widely used and provide flexible automation capabilities. However, they were not selected due to the existing infrastructure and practices within the company. Soudal primarily uses Azure DevOps for project management, version control, and deployment workflows, making it the most suitable choice.

Azure DevOps was chosen because it integrates seamlessly with Azure services, provides built-in support for CI/CD pipelines, and aligns with the company's established development processes. It also enables centralized management of pipelines, environments, and access control, which is important for an internal enterprise application.

Additionally, a multi-environment deployment strategy was considered to support proper development and release workflows. The application is deployed across three environments:

• Development (Dev) – for active development and testing.
• Quality Assurance (QA) – for validation and pre-production testing.
• Production (Prd) – for the live application used by end users.

This approach ensures better quality control, safer releases, and easier troubleshooting. The deployment approach includes:

• Automated build and deployment pipelines using Azure DevOps.
• Integration with Azure resources such as App Service and Azure SQL.
• Secure handling of secrets through Azure Key Vault integration.
• Continuous integration and continuous deployment (CI/CD) to ensure consistent and reliable releases.

2.7. Notifications and Email

For handling notifications, multiple approaches were evaluated, including Microsoft Graph API, SMTP-based email solutions, and real-time in-application notifications.

An SMTP-based solution using MailKit was initially considered for email communication due to its simplicity and straightforward implementation. However, this approach presented several limitations including lack of integration with Azure Active Directory, limited monitoring capabilities, and dependency on traditional SMTP infrastructure that may not align with modern cloud-native architectures.

As an alternative, Microsoft Graph API was evaluated and ultimately selected. While Microsoft Graph requires additional configuration such as application registration, permission management, and administrative consent, it provides significant advantages including:

- Seamless integration with Azure Active Directory for centralized identity and access management
- Modern API-based capabilities with better security controls
- Enhanced monitoring and audit logging through Azure
- Support for rich email features and better error handling
- Alignment with Microsoft 365 ecosystem and organizational standards

Despite the initial setup complexity, Microsoft Graph offers a more robust, secure, and maintainable solution for email communication within the enterprise environment. The additional configuration overhead is justified by the long-term benefits of using a modern, cloud-native approach that integrates naturally with the organization's existing Microsoft infrastructure.

In addition to email notifications, real-time in-application notifications were considered as a valuable enhancement to improve user experience. Technologies such as SignalR enable

instant updates within the application without requiring page refreshes. This approach was identified as a suitable solution for delivering live updates on request status changes and system events.

Therefore, the final design combines a reliable email notification mechanism with the option for real-time in-app notifications, ensuring both immediate user feedback and compatibility with existing enterprise systems.

2.8. Limitations

The scope of this project was clearly defined to focus on the core functionality of an internal visitor management system. As a result, the following aspects were intentionally excluded:

- A public-facing portal for external visitors.
- A dedicated mobile application.
- Uploading or tracking of the Excel template sent after approval.
- Advanced capacity optimization algorithms or automated hard limits per time block.

In addition, due to time constraints, certain non-essential features were identified but were considered to implemented after successful delivery of Minimum Viable Product (MVP). These include enhancements such as advanced reporting visualizations and real-time in-application notifications.

2.9. Summary and Chosen Direction

Based on the analysis and evaluation of different approaches, the following technology stack and architectural decisions were selected for the implementation of the application:

- **Backend:** ASP.NET Core 8 with Razor Pages, using C# and Entity Framework Core 8 for data access.
- **Database:** SQL Server / Azure SQL Database, designed through an ERD and implemented using EF Core migrations.
- **Frontend:** Tailwind CSS (via CDN), Bootstrap Icons, and vanilla JavaScript for lightweight UI enhancements.
- **Real-time communication:** ASP.NET Core SignalR to enable live updates for dashboards and calendar interactions.
- **Authentication & Authorization:** Azure Active Directory (OpenID Connect), with role-based access managed within the application.
- **Infrastructure as Code:** Azure Bicep modules (App Service, Database, Key Vault, Monitoring) orchestrated through a central deployment file.
- **DevOps & Deployment:** Azure DevOps repositories and YAML pipelines, supporting a multi-environment setup (Development, QA, Production) with structured branch-based deployments.
- **Email Notifications:** MS Graph based email handling aligned with the company's internal infrastructure, using a template-based approach.
- **Design & Modelling:** Figma for UI prototyping and StarUML for use case diagrams and ERD design.

This combination of technologies was selected to ensure alignment with company standards, reduce implementation complexity, and provide a scalable and maintainable solution within the project timeline.

3. Realization

This chapter describes the implementation of the Visit Request Portal. It explains how the analyzed requirements were translated into a functional system, covering the architecture, database implementation, core modules, authentication, notifications, infrastructure, and deployment strategy. The goal of this chapter is not to describe every line of code, but to present the most important technical decisions, system components, and implemented functionalities.

3.1. System Architecture

The Visit Request Portal was implemented as a web-based enterprise application using ASP.NET Core 8 Razor Pages. The system follows a layered architecture, separating concerns between presentation, business logic, data access, and infrastructure.
The architecture consists of the following layers:

- **Presentation Layer**
  Implemented using Razor Pages, Tailwind CSS, and JavaScript. This layer handles UI rendering, form interactions, and user experience.
- **Application Layer**
  Contains business logic such as validation rules, request lifecycle management, approval workflows, and notification handling.
- **Data Access Layer**
  Implemented using Entity Framework Core, responsible for database communication, entity mapping, and query execution.
- **Infrastructure Layer**
  Includes Azure services such as App Service, Azure SQL Database, Key Vault, and monitoring tools, provisioned using Infrastructure as Code.

This architecture ensures maintainability, scalability, and clear separation of responsibilities, which is essential for enterprise applications.

3.2. Database Design and Implementation

The database was designed based on the ERD created during the analysis phase and implemented using Azure SQL Database with Entity Framework Core migrations.

3.2.1. Data Model

The core entities include:

- **User**: Stores user information such as email, name, role (Admin, Manager, Employee), and notification preferences.
- **VisitRequest**: Represents a visit request and contains all required information such as visitor name, type, number of people, start and end time, status (Pending, Approved, Rejected, Cancelled), and audit fields.
- **CalendarDay**: Represents a specific day and defines whether the day is open or closed for visits.
- **Block**: Represents time blocks (Morning, Afternoon, Evening) within a day, including their status (Available, Busy, Blocked).
- **TimeBlockSetting**: Stores configurable time ranges for each block.
- **Notification**: Stores in-application notifications for users.
- **VisitRequestChecklist**: Stores set of questions to be filled in by requestor after the approval of visit request.

3.2.2. ERD Diagram

An ERD (Entity Relationship Diagram) is like a blueprint for a database. It shows what kind of information the system stores and how different pieces of information are connected to each other.

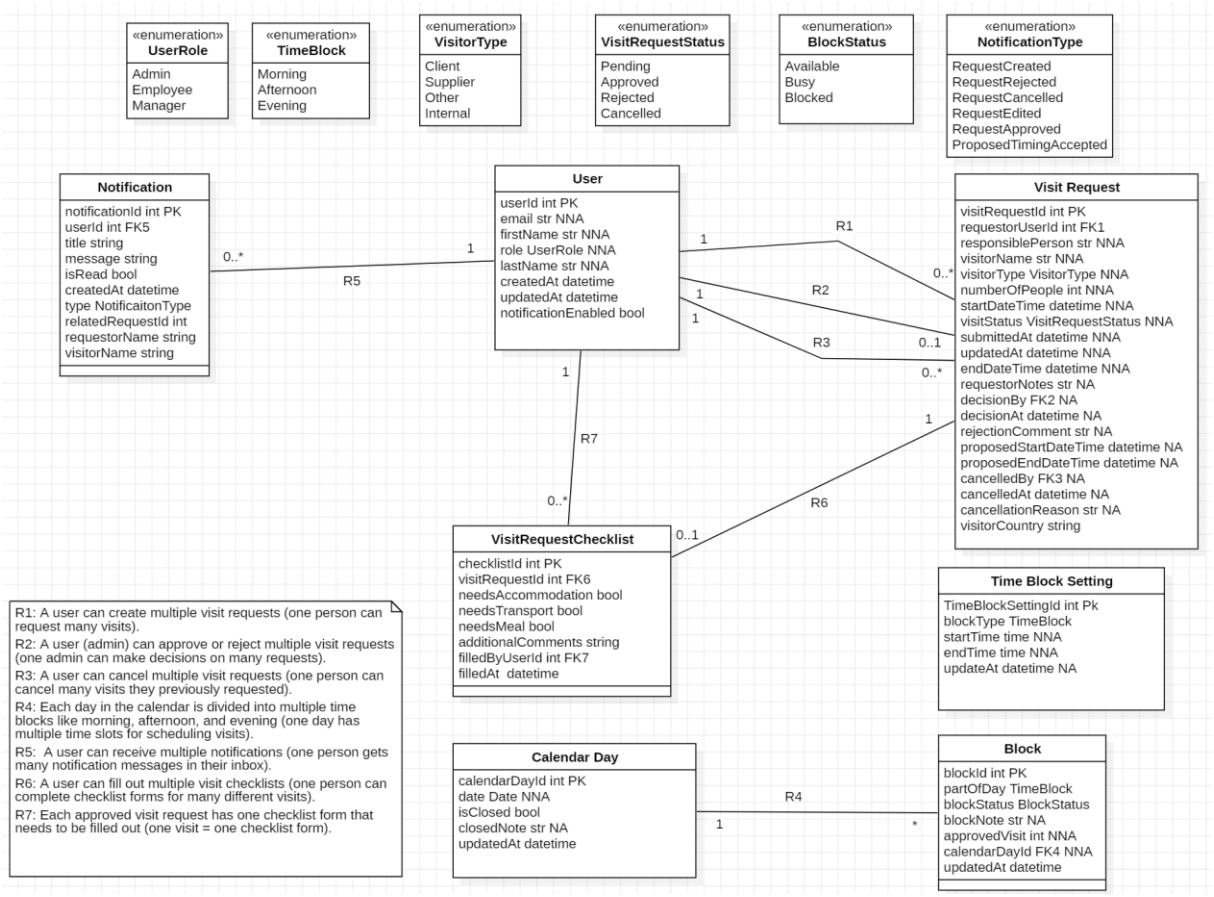


Figure 2: ERD Diagram

Relationships are defined as follows:

- R1: A user can create multiple visit requests (one person can request many visits).
- R2: A user (admin) can approve or reject multiple visit requests (one admin can make decisions on many requests).
- R3: A user can cancel multiple visit requests (one person can cancel many visits they previously requested).
- R4: Each day in the calendar is divided into multiple time blocks like morning, afternoon, and evening (one day has multiple time slots for scheduling visits).
- R5: A user can receive multiple notifications (one person gets many notification messages).
- R6: A user can fill out multiple visit checklists (one person can complete checklist forms for many different visits).
- R7: Each approved visit request has one checklist form that needs to be filled out (one visit = one checklist form).

This database design ensures data consistency, auditability of actions and support for business rules such as approval workflows and cancellation policies.

3.3. Core Functional Modules

The application was implemented as a set of functional modules, each responsible for a specific part of the system. Some of the key features are mentioned below but detail of all features can be found in the user manuals provided in the attachments.

3.3.1. Role-Based Access Control

The system implements role-based authorization with four main roles:

- **Employee (Requestor):** Can view calendar, create and manage own requests
- **Admin:** Can register a visit request, approve/reject requests, manage calendar, manage users, view statistics and logbook
- **Manager:** Can register a visit request and have read-only access to approved visits
- **Operation:** Operations users can create and manage their own visit requests like employees but also have access to an Operations calendar. This calendar shows approved visits for which room checklist information has been completed, allowing the operations team to prepare the required rooms and facilities.

Authorization is enforced at:

- Page level (Razor Pages authorization)
- Business logic level (validation of ownership and permissions)

This ensures data security and controlled access across the application.

3.3.2. Visit Request Management

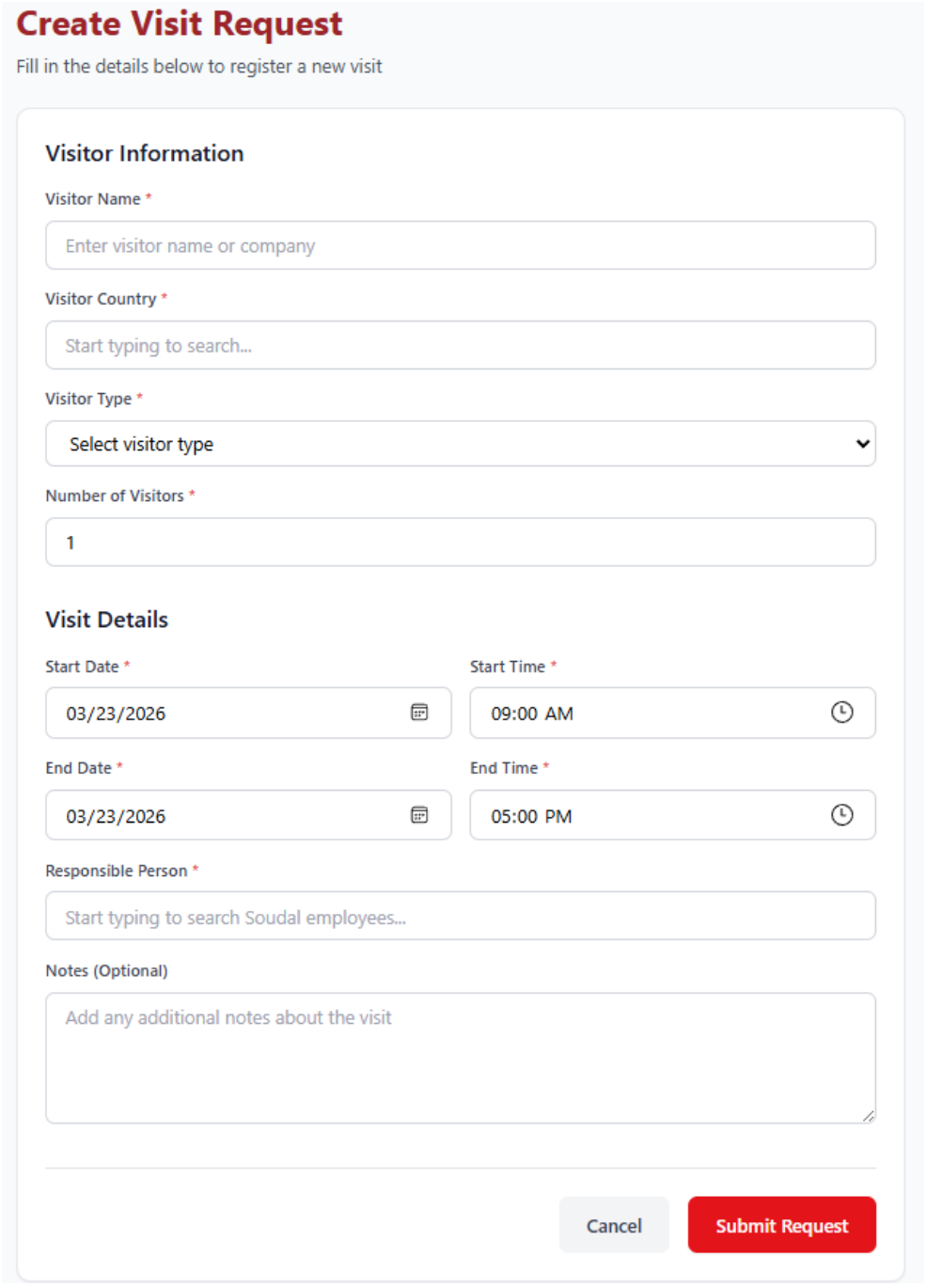


Figure 3: Create Visit Request

This module allows employees to create, edit, cancel, and view their visit requests. Key features:

- Creation of requests with validation (date, time, required fields)
- Editing allowed only for pending or rejected requests
- Cancellation with mandatory reason
- Automatic status handling (Pending, Approved, Rejected, Cancelled)

When a visit request gets created/edited, admins get email and the in-app notification. They can approve requests, reject requests with mandatory comments and can also propose alternative timings, requestors are notified by email and in-app notification about it. Requestors can edit and propose new timings of the rejected request, or they can also accept the proposed timings.

The system enforces business rules such as:

- No modification of past visits
- Reason required if request is rejected or cancelled
- Requests cannot be created for closed days and past days
- Requests created by admins are automatically approved

3.3.3. Calendar Management

The calendar module provides a visual overview of visit availability. It includes a monthly calendar interface that organizes each day into three predefined time blocks: Morning, Afternoon, and Evening. Each time block shows number of approved visits for it and is color-coded to indicate its current status:

- Green: Available
- Orange: Busy
- Red: Too busy
- Grey: Closed

Administrators have the capability to manage availability at both the day and block levels. Specifically, they can change the status of each block, close entire day, or reopen previously closed days to modify availability as required. When admin changes the status of block to busy or close a day, then they must enter the reason so that other users can see it.

To reduce repetitive administrative work, the calendar management interface allows admins to update multiple blocks in a single action. An admin can change only the selected block, apply the same status to the whole day, or select specific blocks such as Morning and Afternoon. The selected blocks are updated together, and the changes are immediately reflected in the calendar views.

The system distinguishes between busy and blocked states as informational indicators rather than hard restrictions. These states signal limited capacity or increased approval difficulty but do not prevent users from submitting requests within those time periods.

All approved visits and administrative actions are immediately reflected in the calendar interface through dynamic synchronization, ensuring real-time accuracy of scheduling data presented to users.

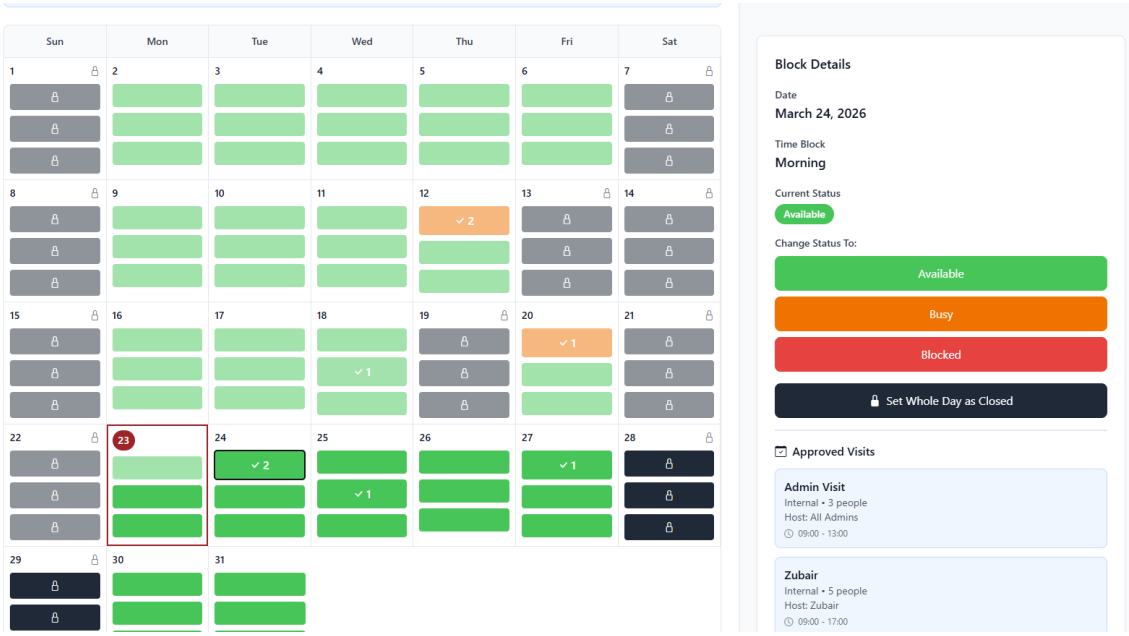


Figure 4: Admin Calendar View

If an admin closes a day that contains existing approved visits, all approved visits scheduled for that day are automatically cancelled. The requestors of those visits are notified of the cancellation via email.

The calendar also supports visits that span multiple time blocks or multiple days. When an approved visit overlaps several blocks, the visit count is shown on every affected block. For example, a visit starting on Friday afternoon and ending on Monday afternoon is reflected in all overlapping blocks between those dates. This ensures that the calendar gives an accurate overview of facility usage, even for longer visits.

3.3.4. Room Checklist Management

The room checklist module allows admins to specify which rooms are required for approved visits. From the admin dashboard, an admin can open a room checklist modal for an approved upcoming visit and select one or more required rooms.

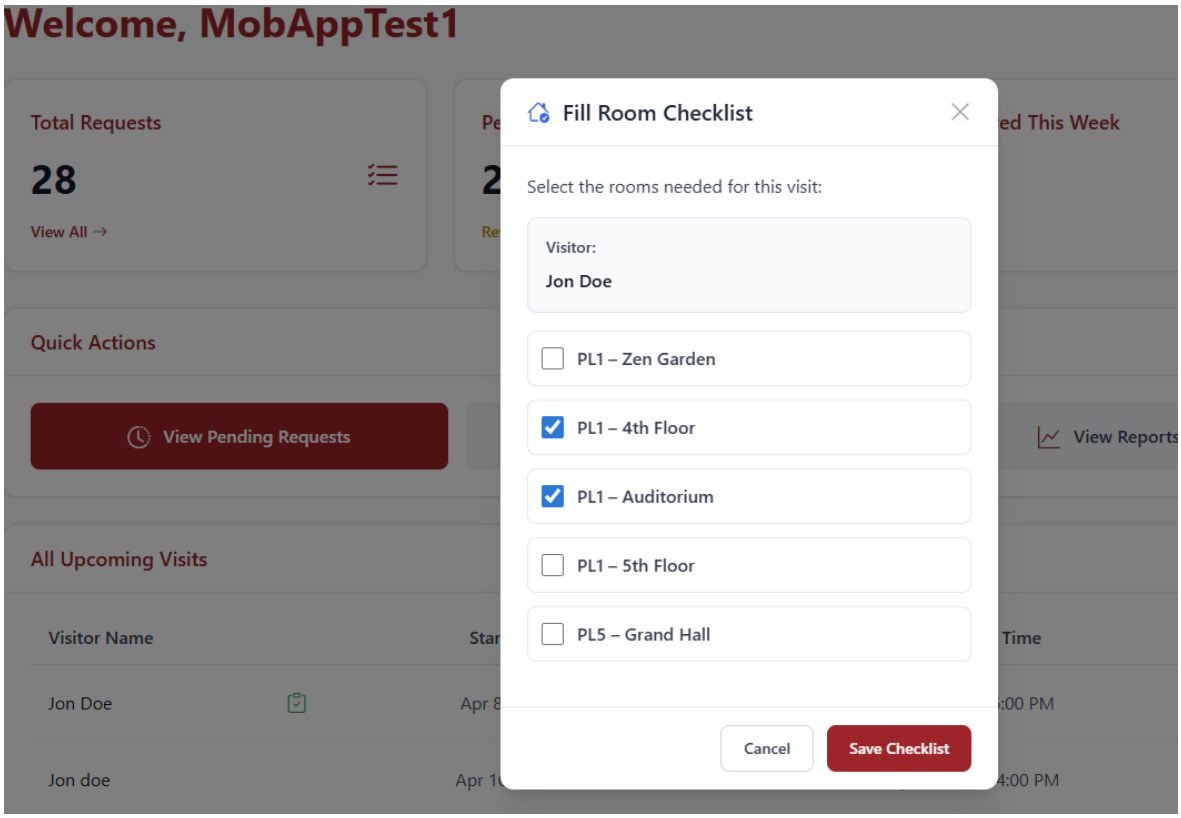


Figure 5: Checklist

The system validates that at least one room is selected before saving. Once the room checklist is submitted or updated, Operations users receive in-application notifications and, when enabled, email notifications. This module supports operational preparation by ensuring that room requirements are visible before the visit takes place

3.3.5. Operations Calendar

The Operations role includes a dedicated calendar view that only displays approved visits with completed room checklists. Unlike the general calendar, this view is focused on operational preparation rather than request creation or approval. Each time block shows the number of visits requiring room preparation. When an Operations user selects a block, the sidebar displays visit details together with the required rooms. This allows the operations team to prepare facilities based on confirmed visit and room information.

3.3.6. Notification System

The application implements a comprehensive notification mechanism designed to ensure that users are continuously informed about all relevant events within the visit request workflow. This mechanism is composed of two complementary subsystems:

- Real-time in-application notification system
- An asynchronous email notification system

These subsystems operate independently but are triggered together in response to key business events, thereby providing both immediate feedback within the user interface and external communication through email channels.

In-application notification system is responsible for delivering real-time updates directly within the application interface. It is implemented using ASP.NET Core SignalR, which enables bidirectional communication between the server and connected clients. Notifications are generated for significant events such as the creation, approval, rejection, editing, and cancellation of visit requests. These notifications are persisted in a dedicated Notification table, ensuring that users can access historical records in addition to receiving real-time updates. Each notification is associated with a specific user and is delivered through SignalR by targeting user-specific groups, typically identified by a pattern such as user-{userId}. This allows precise and efficient delivery of notifications to the intended recipients. The user interface reflects these updates instantly through notification badges, toast messages, and a centralized notifications page, eliminating the need for manual page refreshes. Notably, this system operates independently of user preferences, ensuring that critical system feedback is always visible within the application.

The real-time nature of this system extends beyond simple notifications and includes dynamic updates to key interface elements such as dashboard metrics, request status indicators, and calendar availability. For example, when an administrator approves or rejects a request, the corresponding status badge on the requestor's dashboard is updated immediately, and relevant stakeholders are notified without delay. Similarly, changes in calendar availability, such as blocking or reopening time slots, are propagated instantly to all connected users, maintaining consistency across the system.

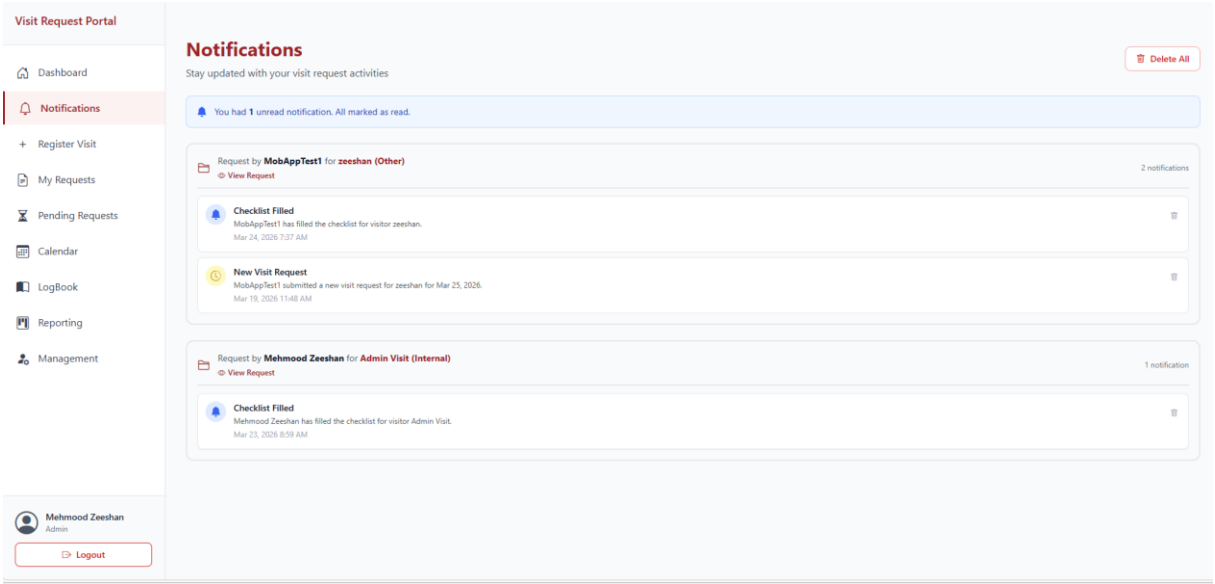


Figure 6: In-App Notifications

Complementing the in-application system, the email notification subsystem is designed to provide external communication through robust and scalable architecture. This subsystem is implemented using the Microsoft Graph API, which enables secure and enterprise-grade email

delivery through Azure Active Directory authentication. Emails are sent from a dedicated organizational mailbox and are processed asynchronously to avoid blocking user operations. When an event occurs, the system enqueues an email notification using a channel-based queue, and a background hosted service processes this queue independently of the main request pipeline. This design ensures that user interactions remain responsive while maintaining reliable delivery of email notifications.

The email system follows a template-based approach, where predefined HTML templates are dynamically populated with relevant data such as visitor details, requestor information, and timestamps. The selection of recipients is governed by business rules and user preferences. Specifically, email notifications are only sent to users who have enabled notifications through a configurable flag, ensuring compliance with user preferences.

Different event types trigger notifications to different roles; for instance, administrators are notified when a new request is created or edited, while requestors receive notifications when their requests are approved, rejected, or cancelled. Managers, in contrast, are treated similarly to regular employees and receive notifications only for their own requests rather than system-wide events, reflecting a deliberate design decision to reduce unnecessary information overload.

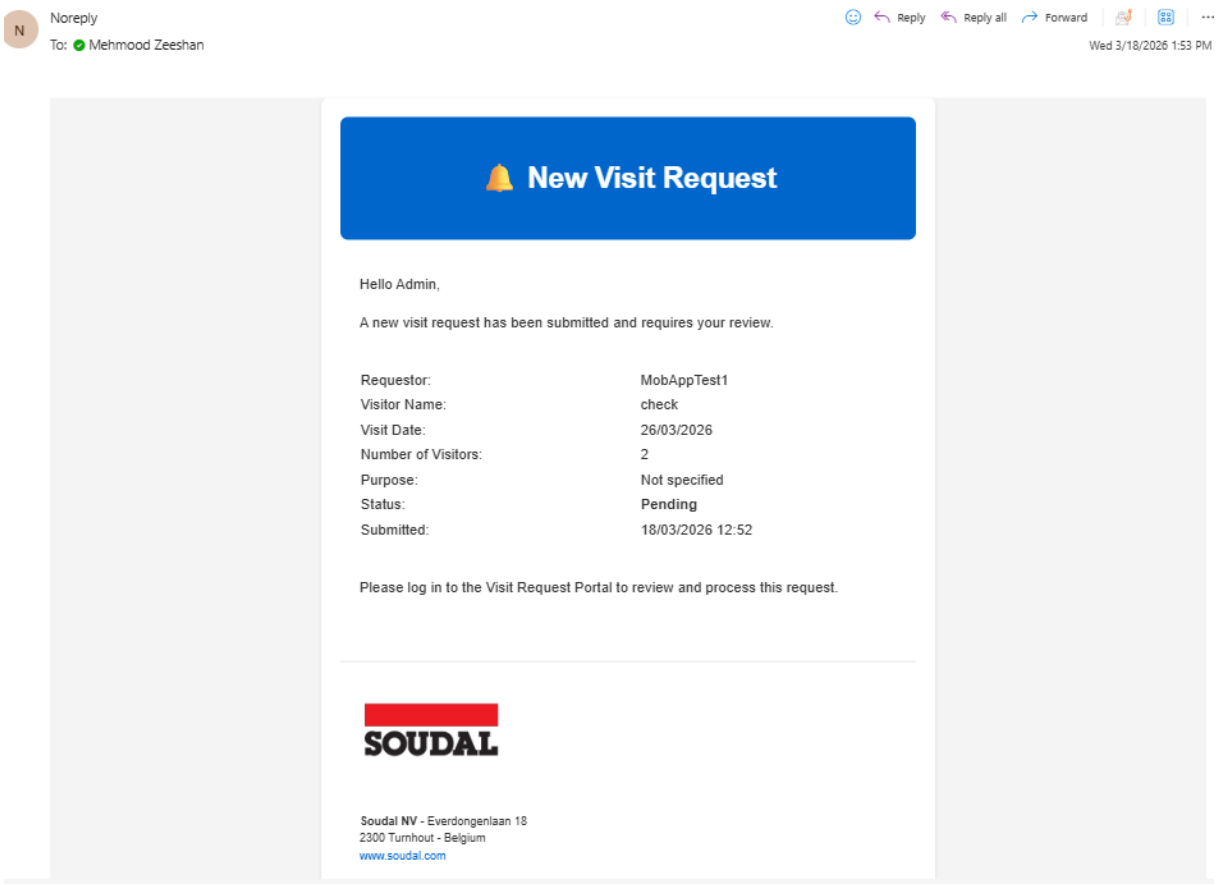


Figure 7: Email Notification

The integration between the two notification systems is achieved at the application layer, where both in-application and email notifications are triggered simultaneously in response to domain events. A typical implementation involves invoking separate services responsible for each notification type, ensuring a clear separation of concerns. For example, after a visit request is created or updated, the application executes logic similar to the following:

```
1. // In-app notification (always triggered)
2. await
_inAppNotificationService.NotifyAdminsNewRequestAsync(visitRequest,
requestorName);
3.
4. // Email notification (only for users with NotificationEnabled = true)
5. var requestor = await
_context.Users.FindAsync(visitRequest.RequestorUserId);
6. await
_emailNotificationService.NotifyVisitRequestCreatedAsync(visitRequest,
requestor);
```

This approach guarantees that all users receive immediate feedback within the application while respecting individual preferences for email communication. The separation of notification responsibilities into distinct services, such as the in-app notification service and the email notification service, enhances maintainability and allows each subsystem to evolve independently.

From a design perspective, several key decisions contribute to the effectiveness of this notification architecture. The separation between real-time and email notifications ensures fault isolation and improves system reliability. The use of asynchronous processing for email delivery prevents performance bottlenecks and enhances scalability. Role-based notification logic ensures that users receive only relevant information, while persistent storage of notifications supports traceability and audit requirements. Collectively, these design choices result in a notification system that is responsive, scalable, and aligned with enterprise application standards, significantly enhancing the overall user experience and operational efficiency of the Visit Request Portal.

3.3.7. Logbook

The logbook includes comprehensive tracking and filtering capabilities to support efficient record management. Admin can filter log entries by date, status, or visitor type to quickly locate relevant information. In addition, the system maintains a complete audit trail of all actions performed, ensuring transparency, accountability, and traceability of changes within the logbook.

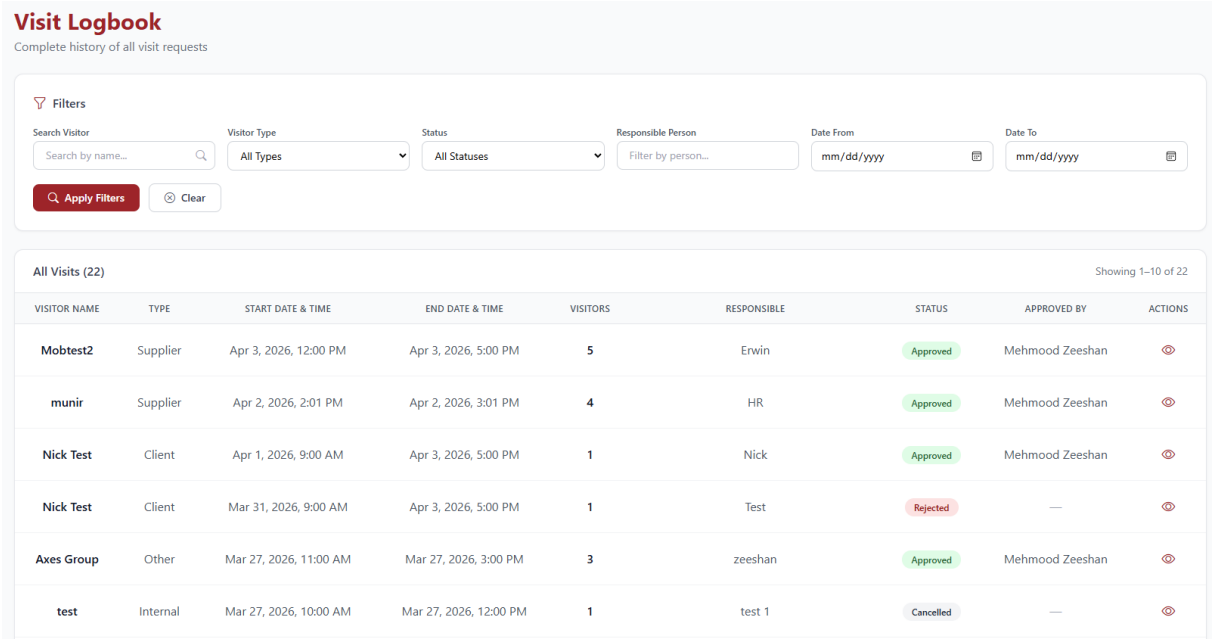


Figure 8: Admin Logbook

3.3.8. Reporting

The reporting module delivers detailed insights into visitor activity and system performance. It includes key performance indicators such as total visits, approved requests, and the number of visitors per month, providing a clear overview of trends and operational efficiency. Interactive charts visualize data on monthly visits, visitor types, and peak days, enabling quick identification of patterns.

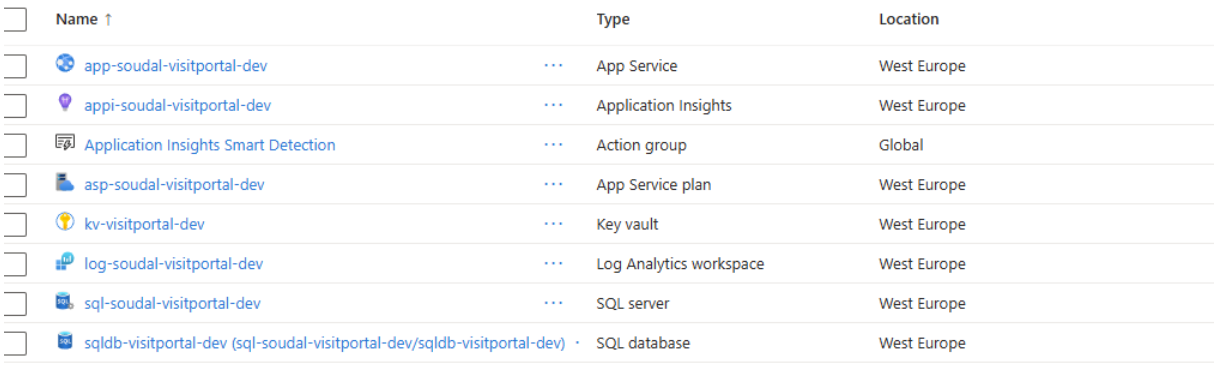
Figure 9: Admin reporting

3.3.9. Infrastructure and Deployment

Application is deployed on Microsoft Azure through a fully automated and scalable infrastructure setup. Using Infrastructure as Code (IaC) principles, the entire environment is defined with Azure Bicep, enabling reproducible deployments, version-controlled configurations, and seamless management across multiple environments.

Primary Azure components include:

- Azure App Service for hosting
- Azure SQL Database for data management
- Azure Key Vault for secure storage of sensitive information
- Application Insights for real-time monitoring and performance tracking.



Name ↑	Type	Location
app-soudal-visitportal-dev	App Service	West Europe
appi-soudal-visitportal-dev	Application Insights	West Europe
Application Insights Smart Detection	Action group	Global
asp-soudal-visitportal-dev	App Service plan	West Europe
kv-visitportal-dev	Key vault	West Europe
log-soudal-visitportal-dev	Log Analytics workspace	West Europe
sql-soudal-visitportal-dev	SQL server	West Europe
sqldb-visitportal-dev (sql-soudal-visitportal-dev/sqldb-visitportal-dev)	SQL database	West Europe

Figure 10: Azure Deployment

Deployment operations are managed through a CI/CD pipeline implemented in Azure DevOps, ensuring consistent and reliable delivery processes. The pipeline is composed of sequential stages that include building the application, running validations, deploying infrastructure via Bicep templates, deploying the application itself, and configuring environment-specific settings.

Database schema changes were managed through Entity Framework Core migrations. The CI/CD pipeline deployed the application and infrastructure, while database migrations were executed separately for each environment during initial setup. This reduced deployment risk and allowed explicit control over schema updates.

By maintaining dedicated Development, QA, and Production environments, the system supports continuous integration and delivery practices that result in reliable deployments, reduced manual errors, and faster release cycles, improving both development efficiency and operational stability.

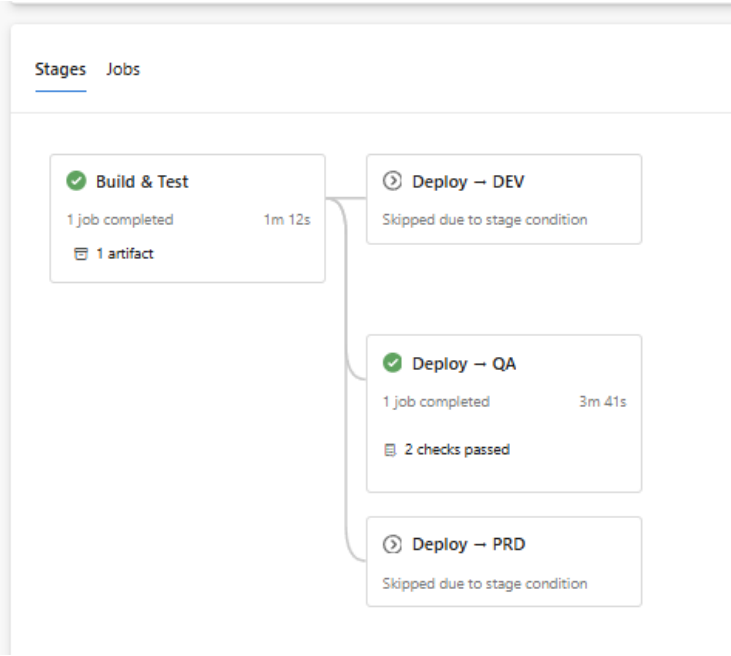


Figure 11: CI/CD Pipeline

3.3.10. Authentication and Security

Authentication within the system is managed through Microsoft Azure Active Directory (Entra ID), ensuring secure and centralized user access. Users authenticate using their company accounts via OpenID Connect, providing a reliable and standardized login process. When users sign in for the first time, their information is automatically registered in the system database, streamlining onboarding and access management. All users are assigned the role of normal employee and admin can later change their role to admin/manager.

Authorization mechanisms are based on role-based access control (RBAC) combined with claims-based identity, granting users permission according to their assigned roles and verified credentials. This approach ensures that each user interacts with the system according to their specific access level and responsibilities.

To maintain a strong security posture, the system adheres to best practices such as avoiding hardcoded secrets, storing sensitive data securely in Azure Key Vault, enforcing HTTPS-only communication, and utilizing managed identities for secure access to services and resources.

3.4. Development Process and Sprint Planning

Development of the application was carried out using an iterative approach based on agile principles, where the work was structured into sprints of two weeks. Azure DevOps was used as the primary tool to manage the development process, enabling clear organization of requirements through product backlog items and associated tasks. The system was divided into major backlog areas including project setup, user authentication, employee workflow, admin workflow, manager workflow, cloud deployment, and documentation. Each backlog item represented a functional or technical module of the application and was further broken down

into smaller implementation tasks to ensure a structured and manageable development process.

For example, the project setup backlog included tasks such as configuring the project repository, defining the database structure, setting up branching strategies, and implementing initial application components. The authentication backlog covered the integration of Microsoft Entra ID and the configuration of secure access control. Functional backlogs such as employee and admin workflows were decomposed into tasks for developing dashboards, calendar views, visit request management, checklist functionality, reporting, logbook, and real-time notification features. Additionally, infrastructure-related work such as CI/CD pipeline setup, Infrastructure as Code implementation, and Azure deployment was organized under a dedicated backlog, ensuring a clear separation between application development and deployment concerns.

Throughout each sprint, tasks were tracked and updated within Azure DevOps, allowing continuous monitoring of progress and ensuring that completed work aligned with the defined requirements. This structured approach improved traceability, as each implemented feature could be directly linked to a backlog item and its corresponding tasks. It also enabled incremental delivery of functionality, where each sprint resulted in a working part of the system that could be tested and refined. Overall, the use of backlog-driven development and sprint-based execution contributed to a well-organized workflow, efficient development process, and successful implementation of the application.

Product Backlo...	Employee Workflow		New	Business
Task	Create Dashboard Page for employee		Done	
Task	Create Calendar Page for Employee		Done	
Task	Create Register visit request functionality		Done	
Task	add checklist to fill when request is approved		Done	
Product Backlo...	Admin workflow		New	Business
Task	Create Admin Dashboard Page		Done	
Task	Create Visit Request Page For admin		Done	
Task	Create Calendar management for admin		Done	
Task	Create LogBook Page for Admin		Done	
Task	Create Reporting Page For Admin		Done	
Task	Create Management Page for Admin		Done	
Task	create real time notifications		Done	

Figure 12: Product Backlog

3.5. Summary of Implementation

The Visit Request Portal was successfully implemented as a full-stack enterprise web application that:

- Centralizes visitor scheduling
- Implements structured approval workflows
- Provides real-time updates and notifications

- Integrates with Azure Active Directory
- Uses cloud-native infrastructure and CI/CD pipelines

The system meets all core requirements defined in the analysis phase and provides a scalable foundation for future enhancements.

4. Additional Work

The main assignment of the internship was the design, development, and deployment of the Visit Request Portal for Soudal. After the core scope of this project was completed within the planned timeframe, additional time remained during the internship. In consultation with the company, this time was used to develop an additional application for the R&D department.

This additional application was not part of the original internship assignment. It was realized as extra work to provide further value to the organization and to support the digitalization of formulation-related processes within the R&D environment. The result was an MVP version of the R&D Formulation Database, a web-based application for creating, managing, searching, and testing chemical formulations.

The purpose of this chapter is to document the additional work separately from the main Visit Request Portal project. This keeps the scope of the original assignment clear, while still showing the additional technical realization completed during the internship.

4.1. Objective of the R&D Formulation Database

The R&D Formulation Database was designed as an internal web application to support R&D and laboratory users in managing formulation data in a structured way. In an R&D context, formulations contain different administrative details, ingredient compositions, production instructions, and test results. Managing this information manually or through disconnected files can make it difficult to search historical formulations, compare results, maintain traceability, and reuse existing work efficiently.

The objective of the application was therefore to create a centralized formulation management system where users can create new formulations, copy existing formulations, define ingredient compositions, register production instructions, assign tests, enter test results, and search formulation records using multiple criteria.

The application was developed as an MVP. This means that the focus was placed on delivering the core workflows first, so that the R&D department could already use and evaluate the application. Additional refinements, such as advanced export functionality, editing existing formulations, pagination, and further UI improvements, were identified as future work.

4.2. Analysis and Design

Before implementing the R&D Formulation Database, the main workflows and data structure were analyzed. Because the application contains several connected processes, such as formulation creation, ingredient composition, production instructions, test registration, bulk result entry, and user permission management, visual design and modelling were used to clarify the intended functionality.

The analysis and design phase produced three main artefacts: a Figma prototype, a use case diagram, and an entity relationship diagram. The Figma prototype was used to validate the user interface and workflow structure. The use case diagram described the interaction between users and the system. The ERD defined the main data entities and relationships required for implementation.

4.2.1. Figma Prototype

Figma prototype was created to visualize the main screens and user flows of the R&D Formulation Database before and during implementation. The prototype helped clarify the layout of the search page, formulation creation wizard, formulation detail page, bulk operation modals, test management page, and user management page.

The prototype was especially useful for the four-step formulation wizard. It provided a visual overview of how users move from administrative data to ingredient composition, production instructions, and test results. This reduced ambiguity before implementation and helped ensure that the application followed a structured workflow.

Prototype link: Figma Link

4.2.2. Use Case Diagram

A use case diagram was created to identify the main actors and interactions within the R&D Formulation Database. The system includes several user types: Administrator, users with view permission only, users with edit and delete permissions and user that can also add test in the system. Each actor has access to different functionalities based on their permissions.

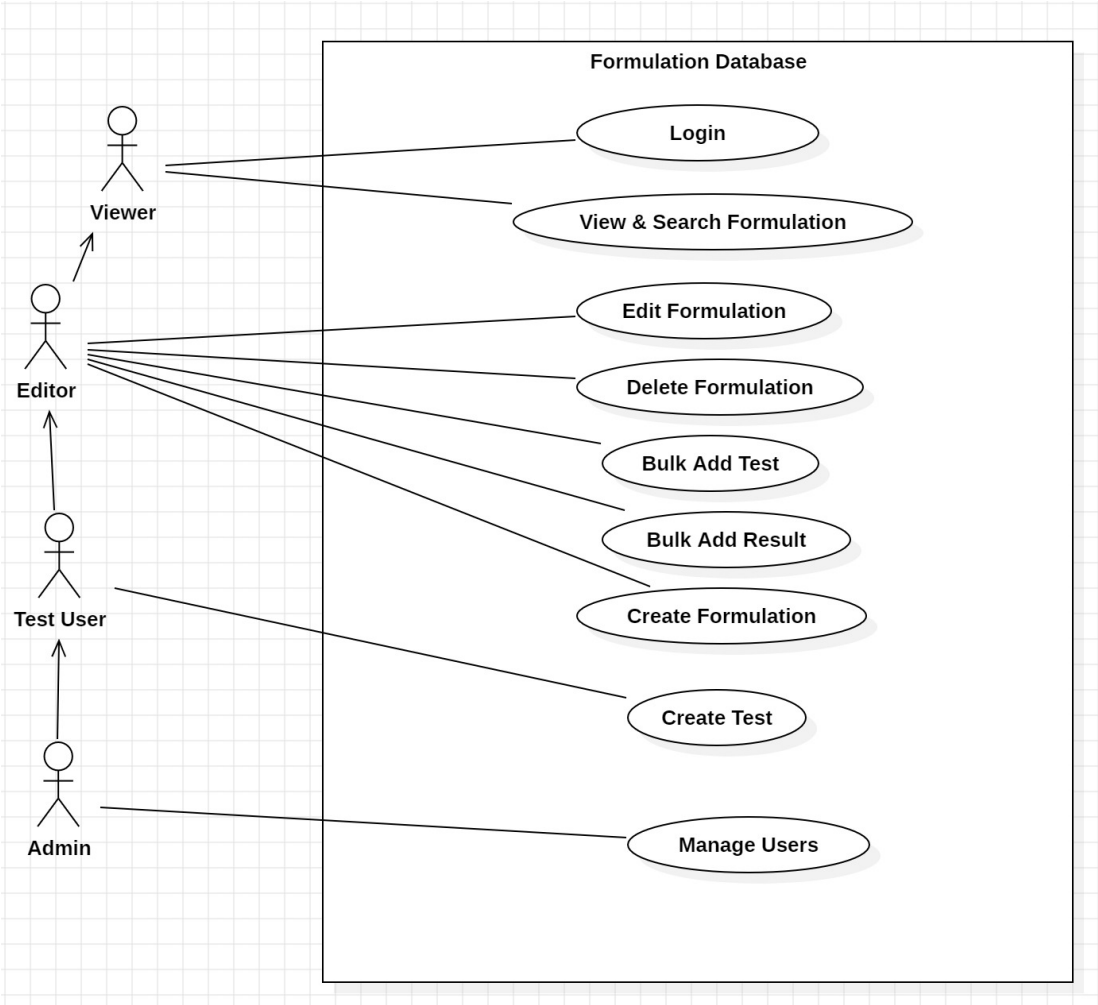


Figure 13: UseCase Diagram - Formulation Database

Regular users mainly search and view formulations, while users with edit and delete can create and copy formulations depending on their permissions. Test personnel mainly interact with test result workflows. Administrators have full access and can manage users, permissions, and test definitions.

Actor	**Main use cases**
Administrator	Manage users, manage permissions, manage test definitions, create formulations, copy formulations, delete formulations, search formulations, view details, bulk add tests, bulk add results.
Editor	Search formulations, view formulation details, create formulations, copy formulations, define ingredients, add production instructions, assign tests.
Test Personnel	Search formulations, view formulation details, enter test results, bulk add results and create tests.
Viewer	Search formulations, view formulation details.

Table 3: Primary Actors

4.2.3. Entity Relationship Diagram

Entity Relationship Diagram (ERD) was created to model the database structure of the R&D Formulation Database. The ERD was used as the basis for implementing the Entity Framework Core models, relationships, foreign keys, and migrations. Because the free version of StarUML was no longer available, dbdiagram.io was used as an alternative tool to create the ERD.

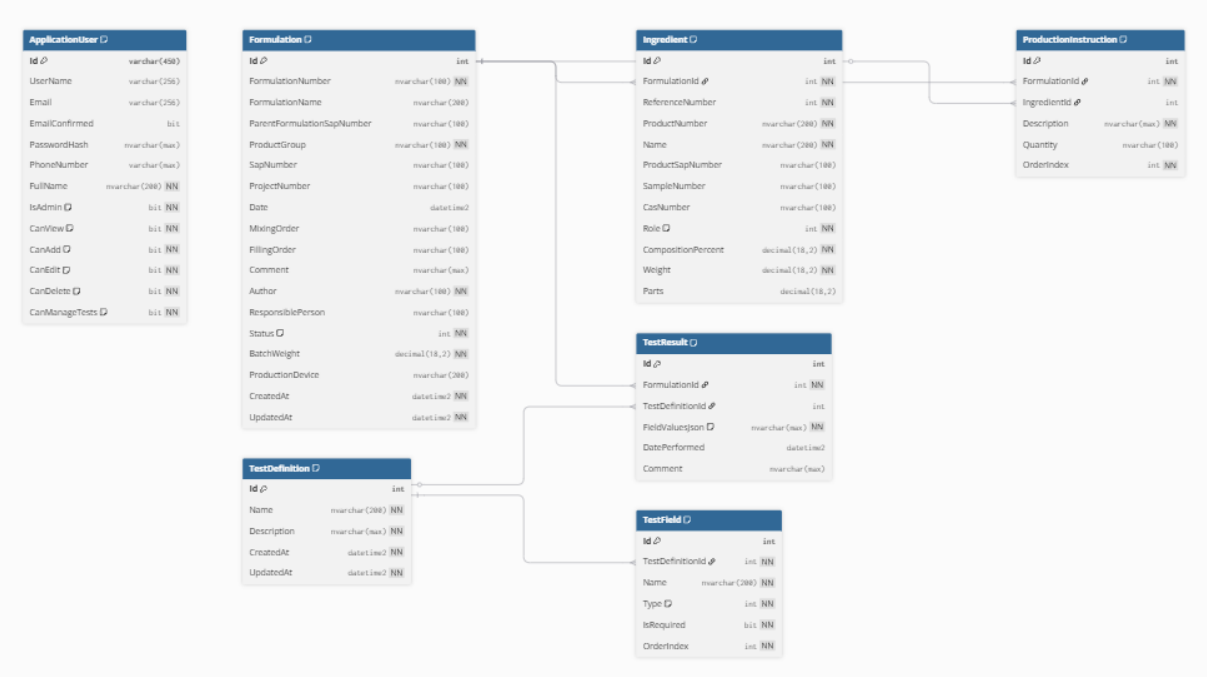


Figure 14: ERD Diagram - Formulation Database

The ERD shows the relationship between formulations, ingredients, production instructions, test definitions, test fields, test results, and application users. A formulation can contain multiple ingredients, multiple production instructions, and multiple test results. Test results are linked to test definitions, while test definitions contain multiple dynamic test fields. This structure allows the system to support different types of laboratory tests without hardcoding every possible test field into the formulation entity.

Relationship	Description
ApplicationUser → Formulation	One user can create multiple formulations.
Formulation → Ingredient	One formulation can contain multiple ingredients.
Formulation → ProductionInstruction	One formulation can contain multiple production instructions.
ProductionInstruction → Ingredient	A production instruction can optionally refer to one ingredient.
Formulation → TestResult	One formulation can contain multiple test results.
TestDefinition → TestField	One test definition can contain multiple dynamic fields.

Relationship	Description
TestDefinition → TestResult	One test definition can be used in multiple formulation test results.
ApplicationUser → Permissions	Each user has permission settings that determine allowed actions.

Table 4: Relationship descriptions

4.3. Target Users and Access Control

The application was designed for several types of users within the R&D and laboratory environment.

Administrators have full access to the system. They can manage users, manage permissions, create and delete formulations, manage test definitions, and access all formulation data.

Users with edit permission can create and work with formulations. They can use the formulation wizard, define ingredients, add production instructions, assign tests, and consult formulation details.

Test personnel can enter or update test results based on their permission level. This supports the workflow where formulation development and test execution may be handled by different users.

Regular users can view and search formulation data if they have the required view permission.

The authorization model was implemented using a permission-based approach. Instead of only using broad user roles, the system supports specific permissions: View, Add, Edit, Delete, and Tests. These permissions determine which actions a user is allowed to perform. This makes the system more flexible because access can be adjusted per user instead of assigning every user the same level of access.

4.4. Technical Architecture

The R&D Formulation Database was implemented as a web application using ASP.NET Core Razor Pages. This approach was chosen because it fits well with form-heavy business applications and allows reuse of knowledge and implementation patterns from the Visit Request Portal.

The application uses server-rendered architecture where Razor Pages handle the user interface, form processing, validation, and navigation. Entity Framework Core is used for database access and object-relational mapping. The database stores formulation data, ingredients, production instructions, test definitions, test results, users, and permission settings.

Authentication was integrated with Microsoft Azure Active Directory through OpenID Connect. Users authenticate using their company account, and user information is synchronized into the application on login. ASP.NET Core Identity and custom authorization policies were used to support the permission-based access model.

The user interface was styled with Tailwind CSS and designed to be clean, practical, and suitable for internal business use. JavaScript was used for client-side enhancements such as dynamic ingredient rows, real-time calculations, composition validation, bulk selection handling, and interactive wizard behavior.

The main technical components of the application are:

- Backend: ASP.NET Core Razor Pages
- Database access: Entity Framework Core
- Authentication: Microsoft Azure Active Directory / Microsoft Entra ID
- Authorization: ASP.NET Core Identity with custom permission policies
- Frontend styling: Tailwind CSS
- Client-side logic: JavaScript
- Database: SQL Server / Azure SQL compatible structure
- Deployment preparation: Azure App Service, Azure SQL Database, Azure Key Vault, and Bicep infrastructure templates

4.5. Database Design and Data Model

The database model was designed around the formulation lifecycle. The central entity is Formulation, which stores administrative information such as formulation number, formulation name, product group, SAP number, project number, production date, author, responsible person, status, and comments.

Each formulation can contain multiple ingredients. The ingredient entity stores data such as reference number, product name, product SAP number, sample number, CAS number, ingredient role, composition percentage, calculated weight, and optional parts value used during ratio calculations.

Production instructions are stored separately and linked to formulation. A production instruction can optionally refer to a specific ingredient, allowing the system to describe how ingredients should be used during the production process.

The test system is based on test definitions and test results. A TestDefinition defines a type of test, such as viscosity, pH, stability, or ISO film properties. Each test definition can contain multiple dynamic fields. These fields can have different data types, such as text, number, percentage, sign, or date. TestResult records store the actual values entered for a specific formulation and test.

The user model contains account information and permission fields. These permissions define whether a user can view, add, edit, delete, or manage test definitions.

The main implemented entities are:

- ApplicationUser: Stores user information and permissions.
- Formulation: Stores the main administrative and formulation metadata.
- Ingredient: Stores ingredient composition and calculation data.
- ProductionInstruction: Stores production steps linked to formulation.
- TestDefinition: Defines available tests and their structure.
- TestField: Defines the fields belonging to a test definition.
- TestResult: Stores test values for formulation.
- FormulationStatus: Defines the state of a formulation, such as Planned, In Testing or Finished.
- IngredientRole: Defines the role of an ingredient, such as Polymer, Oil, Crosslinker, Catalyst, or Additive.
- FieldType: Defines the data type of a test field.

This database structure supports traceability because formulation records include creation and modification timestamps, and related formulation data is stored in structured tables rather than as unstructured documents.

4.6. Formulation Creation Workflow

One of the core modules of the application is the formulation creation workflow. This workflow was implemented as a four-step wizard to guide users through the complete process of creating a formulation.

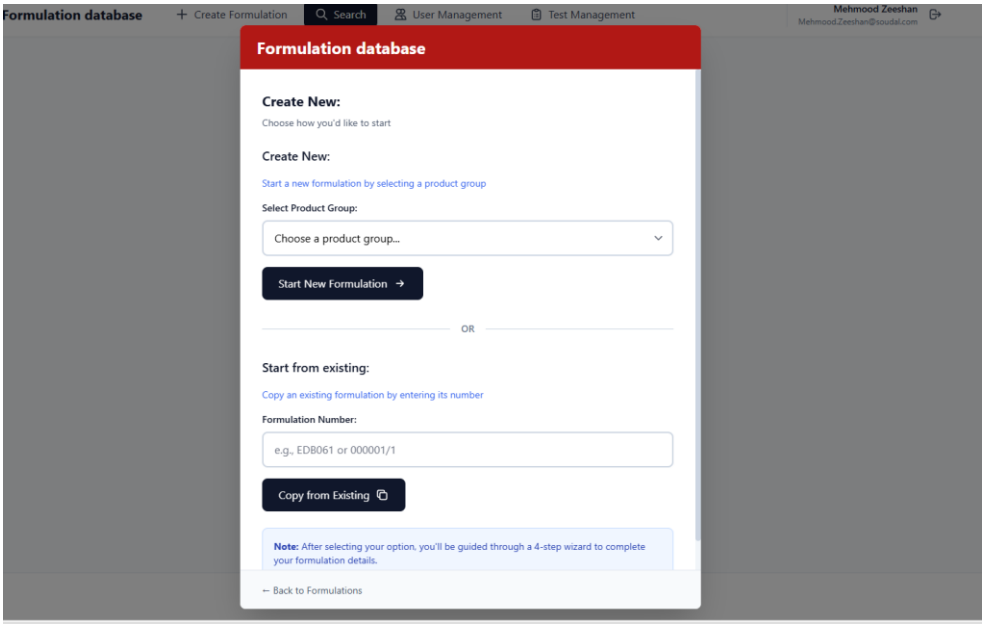


Figure 15: Creating a New Formulation

The first step captures administrative data. Required fields include the formulation number, product group, and author. Optional but searchable fields include formulation name, parent formulation SAP number, production date, mixing order, filling order, project number, status, responsible person, and comments. Required fields are visually distinguished from optional fields, and validation prevents the user from continuing when mandatory information is missing.

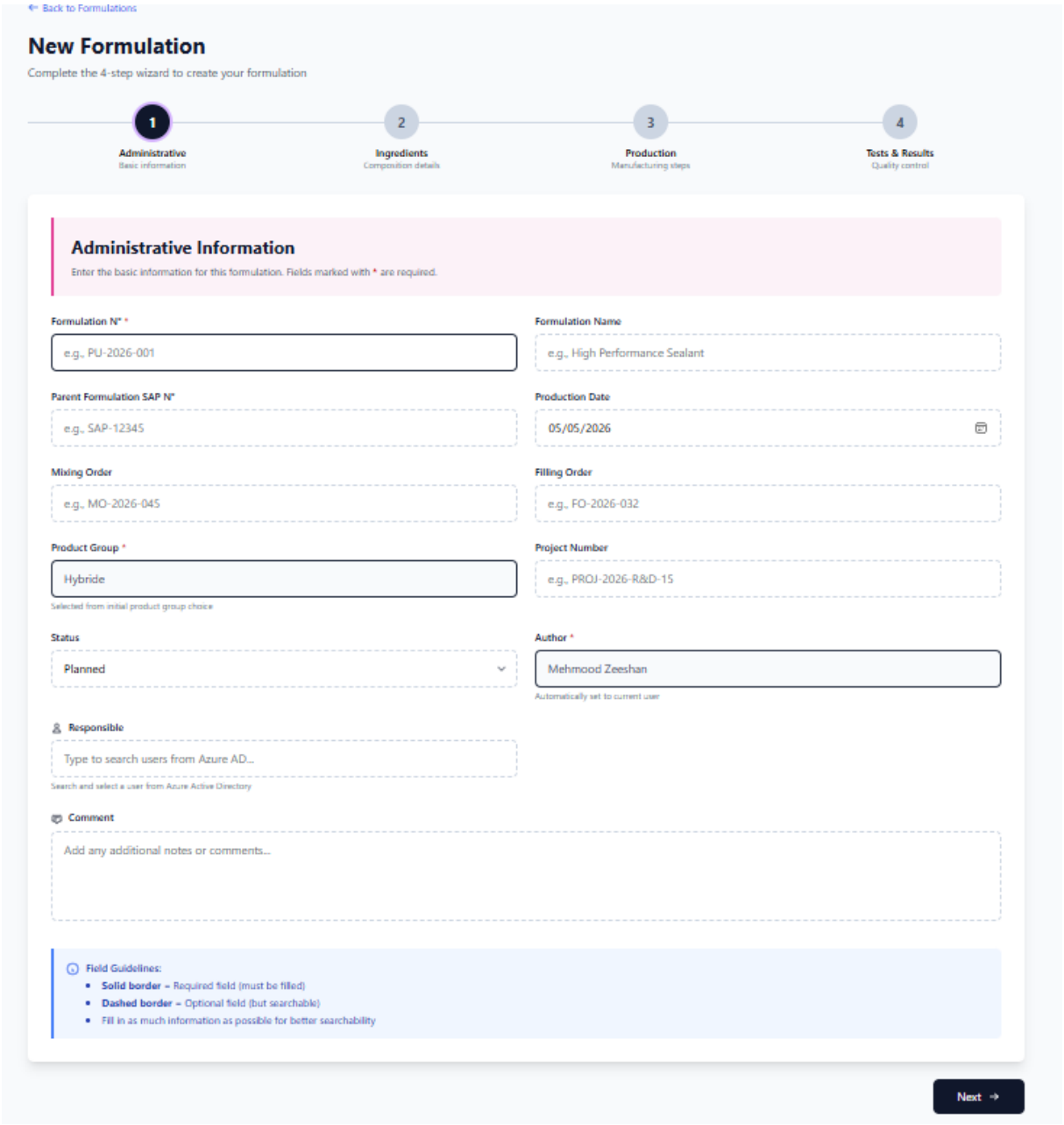


Figure 16: Step 1 - Administrative Information

The second step handles ingredient composition. The user enters the total batch weight and then adds ingredient rows. Each ingredient contains a product name, product SAP number, sample number, CAS number, role, composition percentage, calculated weight, and optional

parts value. The system automatically calculates ingredient weight using the composition percentage and batch weight.

The formula used for weight calculation is:

Weight = Composition Percentage / 100 × Batch Weight

For example, if the batch weight is 1000 g and an ingredient has a composition percentage of 45%, the calculated weight is 450 g.

The system also validates that the total composition percentage equals 100%. A visual indicator shows whether the composition is valid. This reduces manual calculation errors and helps the user detect incomplete or incorrect ingredient compositions before saving.

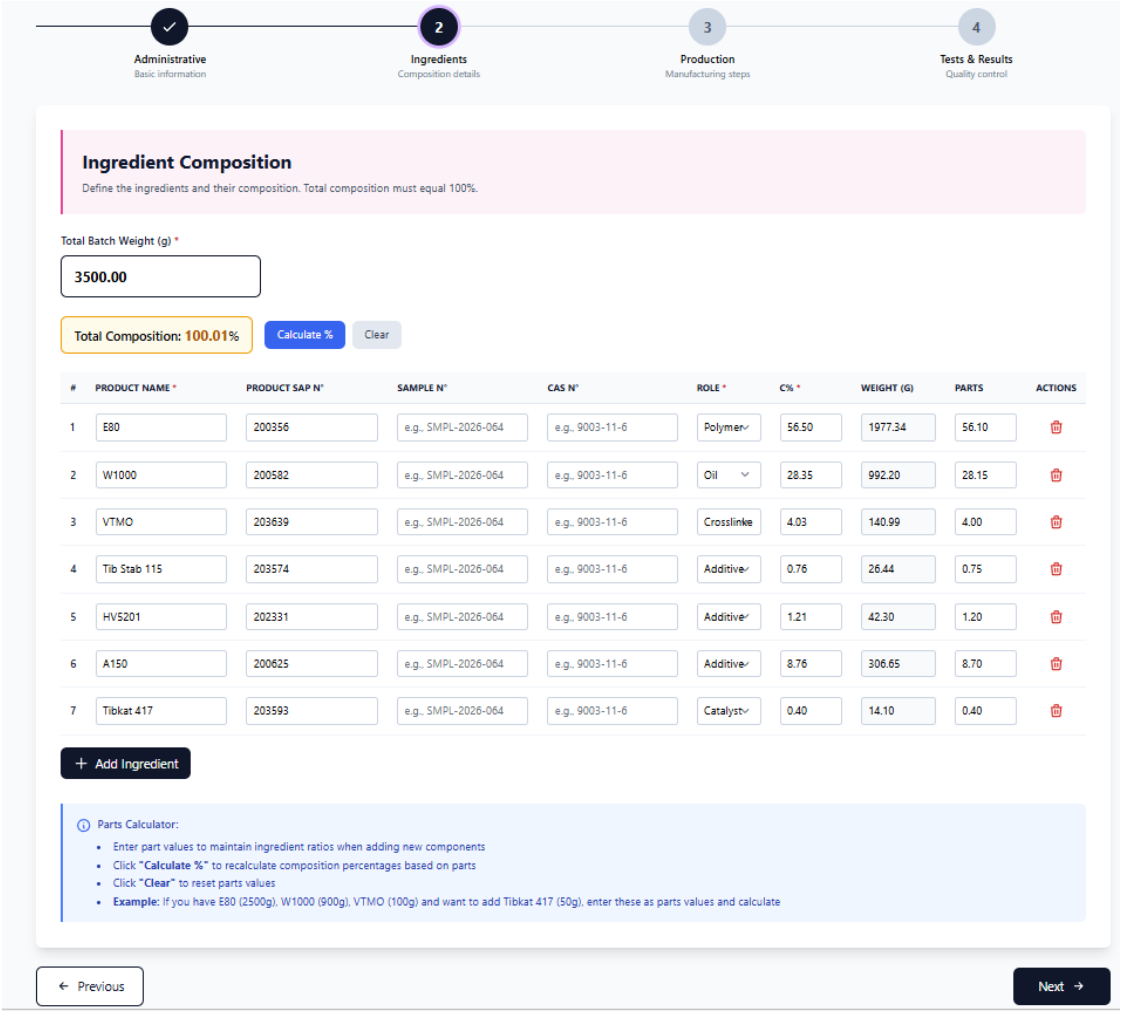


Figure 17: Step 2 - Ingredient Composition

The third step captures production instructions. The user can specify the device or equipment used and add multiple production instruction lines. Each instruction can be linked to a specific ingredient or written as a general production step. This allows the formulation to contain both composition data and practical production guidance.

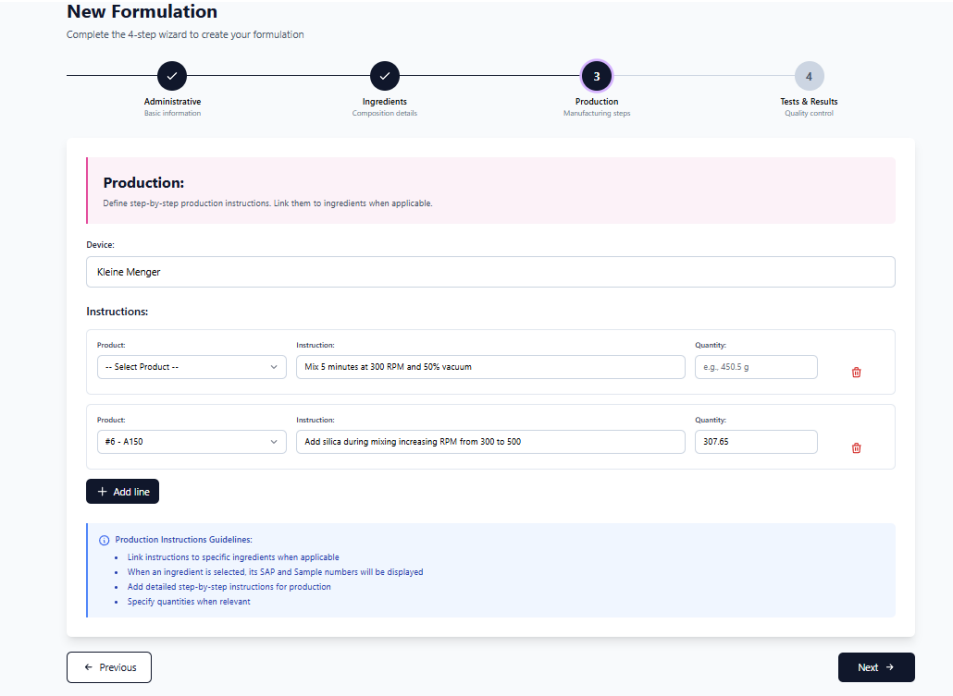


Figure 18: Step 3 - Production Instructions

The fourth step handles tests and results. Users can assign test definitions to the formulation and enter test result values based on the fields defined for each test. Because test definitions are dynamic, different tests can have different fields and field types. This makes the system flexible for different R&D and laboratory test procedures.

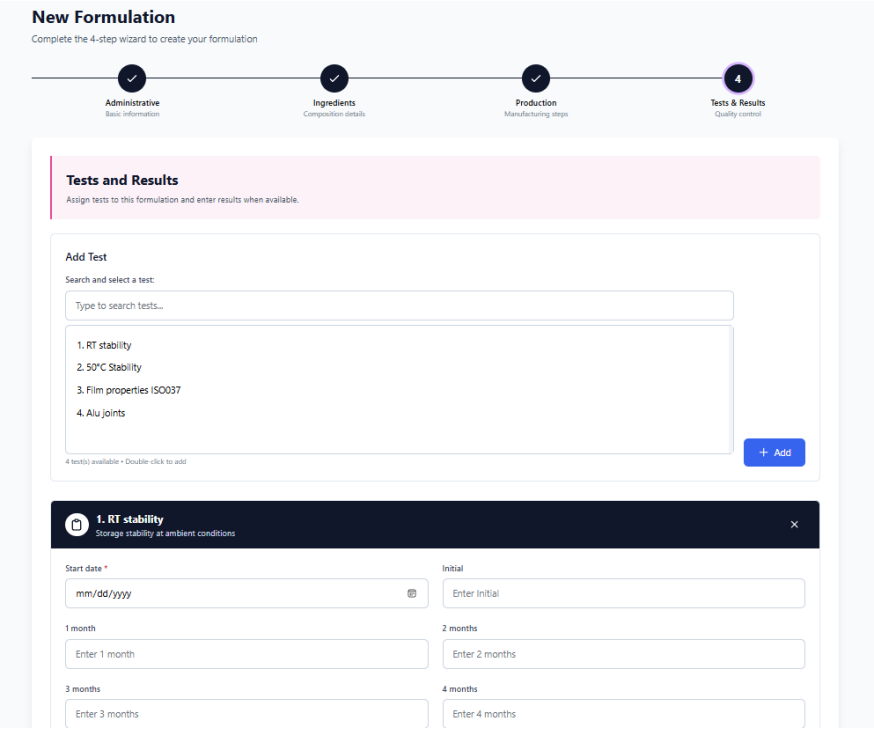


Figure 19: Step 4 - Adding Tests

After completing the wizard, the formulation is saved to the database together with its ingredients, production instructions, and test results.

4.7. Copy Existing Formulation

The application also supports copying an existing formulation. This feature was included because R&D work often builds on previous formulations. Instead of creating a new formulation from scratch, users can duplicate an existing formulation and modify it.

When a formulation is copied, the system duplicates the administrative data, ingredients, production instructions, and test structure. The formulation number is cleared so that the user must enter a new unique number. Creation and update timestamps are also reset so that the copied formulation becomes a new record rather than modifying the original one.

This workflow improves efficiency because users can reuse previous formulation structures while preserving the original formulation as historical data.

4.8. Advanced Search and Formulation List

The search page acts as the main dashboard of the application. It allows users to search formulation records using multiple criteria.

The implemented search fields include formulation number, product group, project number, status, ingredient name, and date range. Search operators such as equals, not equals, and contains were implemented for relevant fields. The search logic is built using LINQ-based dynamic query construction.

Search results are displayed in a table with columns for formulation number, product group, project number, author, date, status, and actions. Users can select individual rows or use a select-all checkbox for bulk operations. The selected count is updated dynamically using JavaScript. The search module provides the basis for several other workflows, including viewing formulation details, deleting formulations, bulk adding tests, and bulk adding results.

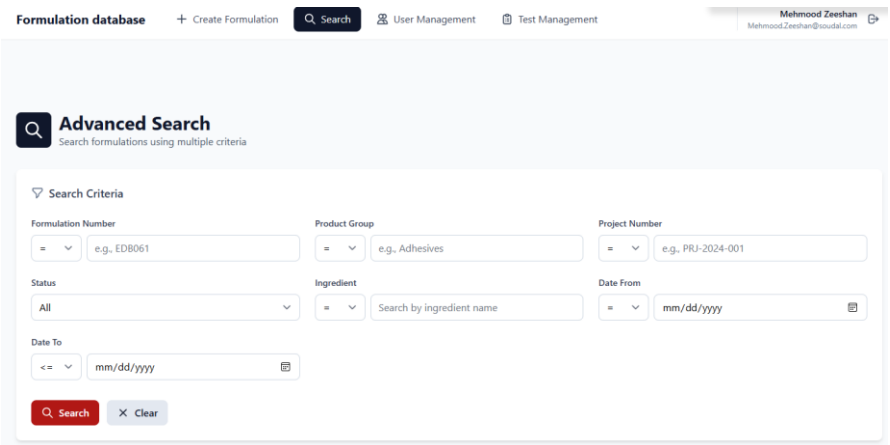


Figure 20: Search Page

4.9. Formulation Details Page

The formulation details page provides a read-only overview of all information related to a formulation. It displays the formulation number, product group, status, administrative information, ingredient composition, production instructions, and test results.

The administrative section shows fields such as SAP number, project number, date, author, responsible person, and status. The ingredient section displays the batch weight and a complete ingredient table, including composition percentages and calculated weights. The production section shows the device used and the ordered production instructions. The test results section displays each assigned test with its corresponding values, date performed, and comments.

This page provides users with a complete overview of a formulation without requiring them to open multiple files or screens.

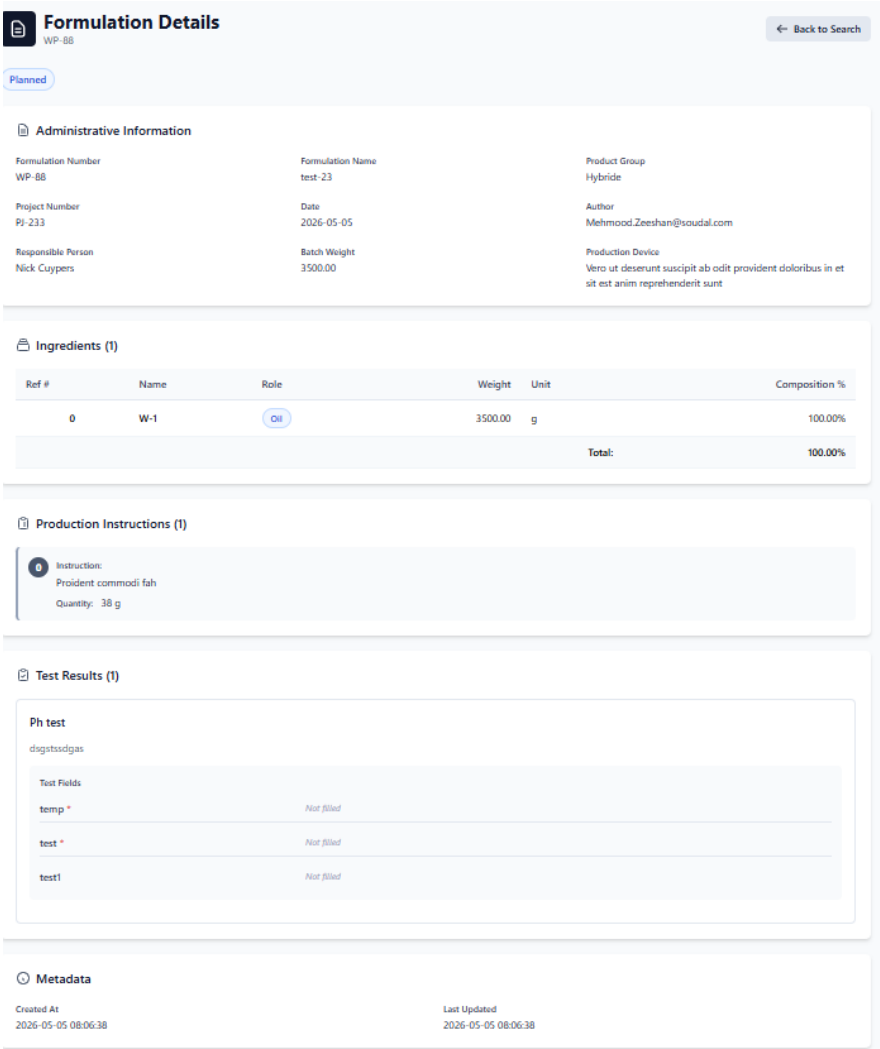


Figure 21: Formulation Detail Page

4.10. Test Definition Management

The test management module allows authorized users to create, edit, and delete test definitions. Test definition describes the structure of a test that can be assigned to formulations.

Each test definition contains a name, description, and one or more fields. Field types include string, number, percentage, signs, and date. Fields can also be marked as required. This design allows admins or authorized users to define test templates without changing the application code.

For example, an ISO film properties test can contain fields such as Modulus 100%, F max, and E at F max. A viscosity test can contain different fields, such as viscosity value, temperature, and unit.

The test management module makes the application adaptable to different laboratory procedures and avoids hardcoding every possible test structure into the application.

4.11. Bulk Operations

To support efficient work with multiple formulations, bulk operations were implemented.

The Bulk Add Tests feature allows users to select multiple formulations from the search results and assign one or more tests to all selected formulations. The system checks duplicates and skips tests that are already assigned. This prevents unnecessary duplicate test records.

The Bulk Add Results feature allows users to select multiple formulations and enter test results in a grid interface. The selected test must be common to all selected formulations. The grid displays formulation numbers as columns and test fields as rows. Users can then enter result values for several formulations at once. This is useful when laboratory results are available for multiple formulations and need to be entered efficiently.

The bulk results grid also supports temporary local saving in the browser, reducing the risk of losing entered data before final confirmation.

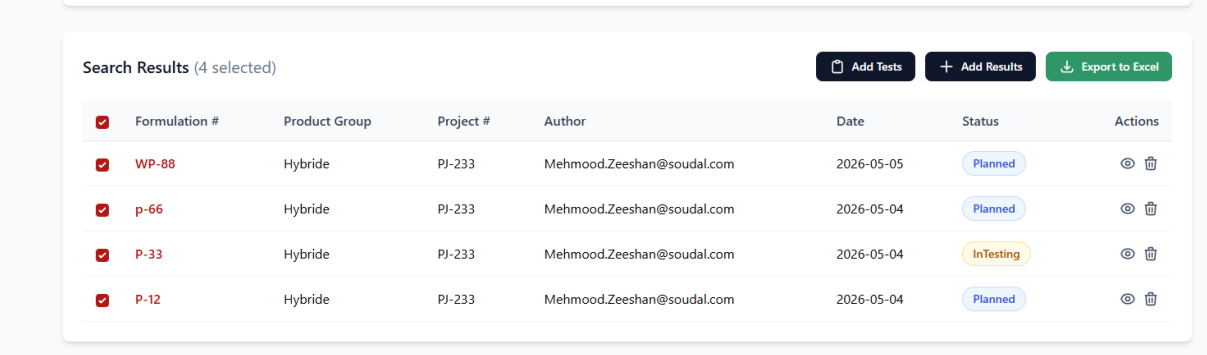


Figure 22: Bulk Operation

4.12. User Management and Permissions

The user management module allows administrators to manage users and their permissions. Users can be searched and added from Azure Active Directory. After a user is added to the application, the administrator can assign permissions using toggles.

The five implemented permissions are:

- View: Allows the user to view and search formulations.
- Add: Allows the user to create new formulations.
- Edit: Allows the user to modify formulation data where this functionality is available.
- Delete: Allows the user to delete formulations.
- Tests: Allows the user to manage test definitions.

Permission enforcement was implemented at both the user interface level and the server level. UI-level enforcement hides or shows actions depending on the current user's permissions. Server-level enforcement uses authorization policies and permission checks to prevent unauthorized access even if a user attempts to access a page or action directly.

This design provides a more granular security model than a simple admin/user distinction.

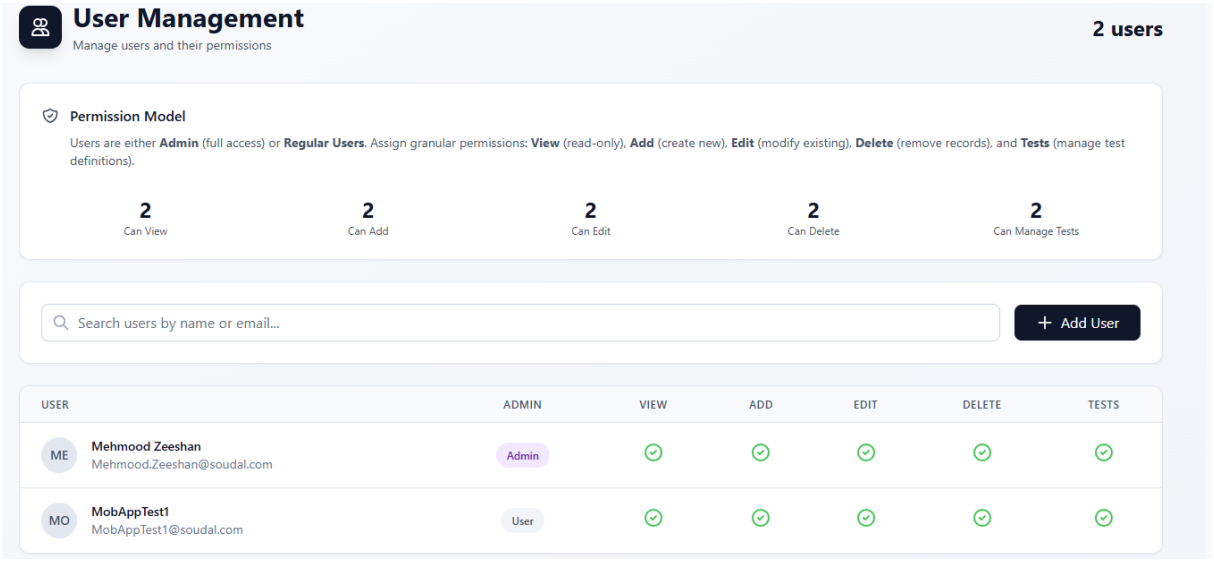


Figure 23: User Management

4.13. Authentication and Security

The application uses Azure Active Directory authentication. When a user accesses the application, they are redirected to the Microsoft login flow and authenticate using their company account. After login, the application reads user claims and synchronizes the user into the local application database.

Security was implemented using a combination of authentication, permission-based authorization, server-side validation, and Entity Framework Core database access. Entity Framework Core protects database queries through parameterized queries, reducing the risk of SQL injection. Sensitive configuration values, such as connection strings and Azure AD secrets, were prepared for storage through secure configuration mechanisms such as Azure Key Vault.

The application also includes an access denied page and protected routes. This ensures that users without the required permission receive clear feedback instead of gaining unauthorized access.

4.14. Infrastructure and Deployment

The R&D Formulation Database was prepared for deployment using Azure-based infrastructure. The deployment configuration includes Azure App Service for hosting, Azure SQL Database for persistent storage, Azure Key Vault for secure configuration, and Bicep templates for Infrastructure as Code.

Bicep templates were created to define the infrastructure in a reproducible way. This approach makes deployment more consistent and reduces the risk of manual configuration errors. Environment-specific configuration files were also prepared for development, staging, and production settings.

The application uses Entity Framework Core migrations to create and update the database schema. Multiple migrations were created and applied during development. Database indexes were also implemented for search performance, especially because the search page is one of the central features of the system.

At MVP level, the deployment setup was sufficient for basic use and validation. Some production-readiness aspects, such as complete backup strategy, performance optimization, automated testing, and large-scale pagination, remain future improvements.

4.15. Current MVP Status

The R&D Formulation Database reached an MVP state with the main functional workflows operational.

The implemented parts include:

- User authentication through Azure Active Directory.
- Permission-based authorization.
- Formulation creation through a four-step wizard.
- Copying existing formulations.
- Viewing complete formulation details.
- Deleting formulations.
- Advanced search with multiple criteria and operators.

- Ingredient composition management with automatic calculations.
- Parts calculator for maintaining ingredient ratios.
- Production instruction management.
- Test definition management.
- Test result registration.
- Bulk adding tests to multiple formulations.
- Bulk adding results through a grid interface.
- User management with permission configuration.
- Database models, relationships, migrations, and indexes.
- Azure deployment preparation with infrastructure templates.

This MVP already demonstrates the core value of the system: structured formulation registration, searchable historical data, reusable formulation information, and centralized test management.

4.16. Limitations and Future Improvements

Although the MVP is functional, some features were not fully completed within the available time and future work should be considered.

The first important limitation is direct editing of existing formulations. At MVP stage, users can copy an existing formulation and create a new version, but a separate edit page for modifying an existing formulation directly was not yet implemented. Future development should add an edit workflow that loads the existing formulation into the wizard, preserves the original creation timestamp, updates the modification timestamp, and applies the same validation rules as the creation workflow.

The second limitation is data export. The user interface contains preparation for export functionality, but the backend export process was not fully implemented in the MVP. Future development should add field selection, CSV or Excel generation, and file download functionality. This would allow R&D users to export selected formulation data for analysis, reporting, or sharing.

The third limitation is scalability for large datasets. Search functionality works for the current MVP scope, but pagination and sortable table headers should be added before larger production use. This would improve usability and performance when the database contains many formulations.

Further improvements include a centralized toast notification system, enhanced mobile navigation, better accessibility support, print-friendly formulation detail pages, automated tests, caching strategies, and a formal backup and recovery plan.

4.17. Summary of Additional Work

The R&D Formulation Database was developed as additional work after completion of the main Visit Request Portal assignment. The application provides a structured MVP for managing

formulation data, ingredient compositions, production instructions, test definitions, and test results.

The project reused several enterprise application principles also applied in the Visit Request Portal, including Azure AD authentication, permission-based authorization, Entity Framework Core persistence, server-side validation, and Azure-oriented deployment preparation. At the same time, the R&D application introduced different technical challenges, especially dynamic formulation composition, real-time calculation logic, test definition flexibility, and bulk result entry.

This additional project delivered further value to Soudal by providing the R&D department with a working foundation for a centralized formulation management system. Although some features remain future work, the MVP demonstrates the feasibility of replacing fragmented formulation tracking with a structured internal web application.

5. Conclusion

5.1. Recap of the Project

The main objective of this internship was to design, develop, and deploy an internal Visit Request Portal for Soudal. The project started with a practical business problem: visitor planning was handled through informal communication, personal calendars, and email exchanges. This made it difficult to maintain a shared overview of planned visits, avoid scheduling conflicts, track approvals, and provide management with reliable reporting.

The project plan defined the Visit Request Portal as a secure internal web application with Azure AD authentication, role-based access, calendar-based booking, approval and rejection workflows, email notifications, logbook functionality, reporting, audit logging, and deployment to Microsoft Azure. These objectives formed the basis for the analysis, design, implementation, testing, and deployment phases of the internship.

The final application replaces the informal visitor planning process with a centralized digital workflow. Employees can create and manage their own visit requests, admins can approve or reject requests and manage calendar availability, managers can consult approved visits, and operations users can prepare rooms based on completed room checklists. The system also provides real-time in-application notifications, email notifications through Microsoft Graph, reporting, logbook functionality, and secure access through Microsoft Entra ID.

The application was implemented as an ASP.NET Core 8 Razor Pages application using Entity Framework Core and Azure SQL Database. Infrastructure was defined using Azure Bicep, and deployment was automated through Azure DevOps CI/CD pipelines across Development, QA, and Production environments. This resulted in a maintainable and scalable internal enterprise application aligned with Soudal's Microsoft ecosystem.

In addition to the main assignment, an additional MVP application was developed for the R&D department: the R&D Formulation Database. This was not part of the original internship assignment but was completed after the core Visit Request Portal scope was delivered within the planned timeframe. The R&D Formulation Database provides a structured system for managing formulations, ingredient compositions, production instructions, test definitions, test results, user permissions, and bulk result entry.

Together, both applications contributed to the digitalization of internal processes at Soudal. The Visit Request Portal addressed the original visitor planning problem, while the R&D Formulation Database delivered additional value by providing a working foundation for centralized formulation management.

5.2. Key Learnings and Outcomes

5.2.1. Major Achievements

The most important achievement of the internship was the successful realization of the Visit Request Portal as a full-stack internal web application. The system centralizes visitor scheduling, introduces a structured approval workflow, improves visibility for management, and provides admins with tools to manage availability and reporting.

First, I had to understand the client's business requirements clearly and translate them into practical application features. This included analyzing the manual process, identifying the required user roles, and defining the correct approval workflow. A major achievement was converting these requirements into a fully functional application that supports real business use inside the company.

Second major achievement was the integration of the application into the company's Microsoft environment. Authentication was implemented using Microsoft Entra ID, email notifications were sent through Microsoft Graph and cloud deployment was handled through Azure App Service, Azure SQL Database, Azure Key Vault, Application Insights, and Azure DevOps.

Third achievement was the implementation of real-time functionality using SignalR. Notifications, dashboard updates, request status changes, and calendar availability changes are reflected immediately in the user interface. This improves the user experience and reduces the need for manual page refreshes.

Another important achievement was the extension of the system beyond the initially planned roles. The original project scope focused on requestors, admins, and management users. During realization, the system was expanded with an Operations role and room checklist functionality. This made the application more useful for practical facility preparation and not only for request approval.

The final major achievement was the development of the additional R&D Formulation Database MVP. This second application shows that the remaining internship time was used

productively and that the technical patterns learned during the Visit Request Portal project could be reused in another business domain.

5.2.2. Against the Project Plan

The project plan divided the internship into four main phases: requirements and design, core development, cloud deployment, and documentation and stabilization. The result follows this structure closely.

Project Plan Phase	Planned Work	Delivered Result	Evaluation
Requirements and Design	Gather requirements, create use cases, Figma prototypes, ERD, backlog, and architecture plan	Requirements were analyzed, use case diagrams and descriptions were created, Figma prototypes were prepared, and the database model was designed	Completed
Core Development	Implement authentication, calendar, request workflow, admin interface, notifications, logbook, and reporting	Core modules were implemented, including visit request management, calendar management, admin workflows, notifications, reporting, logbook, checklist flows, and role-based access	Completed and extended
Cloud Deployment	Configure Azure infrastructure, deploy the application, and set up CI/CD	Azure infrastructure was defined with Bicep and deployed through Azure DevOps pipelines across Dev, QA, and Prd environments	Completed
Documentation and Stabilization	Test, fix bugs, optimize, document, and prepare handover	Functional testing, bug fixing, technical documentation, developer guidelines, and realization documentation were prepared	Completed
Additional Work	Not part of original scope	R&D Formulation Database MVP was developed and deployed/prepared as additional work	Extra delivery

Table 5: Internship Implementation

5.2.3. Technical and Professional Skills Gained

The internship provided practical experience with the complete software development lifecycle. This included requirement analysis, stakeholder communication, UI prototyping, database modelling, implementation, testing, deployment, documentation, and handover.

From a technical perspective, the internship strengthened my skills in ASP.NET Core Razor Pages, C#, Entity Framework Core, SQL Server, Azure SQL Database, Microsoft Entra ID authentication, role-based authorization, Microsoft Graph email integration, SignalR, Azure Bicep, Azure Key Vault, Application Insights, and Azure DevOps pipelines.

The internship also improved understanding of enterprise application architecture. Both applications required clear separation between presentation logic, business logic, data access, and infrastructure. This helped me understand why maintainability, security, and deployment consistency are important in business-critical internal applications.

The project also improved ability to translate business rules into technical implementation. Examples include request approval rules, rejected request workflows, proposed alternative timings, admin calendar overrides, closed-day validation, automatic cancellation when closing days, notification routing, room checklist handling, and permission-based access.

On a professional level, the internship provided experience in communicating with stakeholders, validating requirements, adjusting functionality based on feedback, documenting technical decisions, and planning work through Azure DevOps backlog items and sprints. These skills were important because the project was not only about writing code, but also about delivering a usable and maintainable solution for a real company environment.

5.2.4. Reflection on Challenges and Solutions

One of the main challenges was translating an informal visitor planning process into a structured digital workflow. In the original process, decisions could be handled through email or personal communication. In the application, these decisions had to be converted into clear statuses, validation rules, user permissions, and notification flows. This required careful analysis of how requests are created, approved, rejected, cancelled, edited, and displayed in the calendar.

Second challenge was calendar availability logic. The calendar needed to support time blocks, busy and blocked states, closed days, approved visit counts, admin overrides, and visits spanning multiple blocks or multiple days. The solution was to model days and blocks separately, calculate approved visit counts dynamically, and synchronize changes through SignalR so that users always see up-to-date availability.

Third challenge was notification handling. The system needed both immediate in-application feedback and reliable email communication. To solve this, the notification system was split into two parts: real-time in-app notifications using SignalR and asynchronous email notifications

using Microsoft Graph. This separation improved maintainability and prevented email processing from blocking user actions.

Fourth challenge was to secure integration with the Microsoft ecosystem. Authentication, authorization, email sending, secret management, and deployment all had to fit within Soudal's existing infrastructure. This was solved by using Microsoft Entra ID, Microsoft Graph, Azure Key Vault, managed identities, and Azure DevOps pipelines.

Fifth challenge was deployment consistency. Manual deployment would have increased the risk of configuration mistakes between environments. To reduce this risk, the infrastructure was described using Azure Bicep and deployed through automated pipelines. This made the environment more reproducible and easier to maintain.

For the R&D Formulation Database, the main challenge was modelling a flexible formulation structure. Formulations contain administrative data, ingredients, calculated weights, production instructions, dynamic test definitions, and test results. The solution was to design a structured data model with separate entities for formulations, ingredients, production instructions, test definitions, test fields, and test results. This made the application flexible enough to support different test types without hardcoding every possible test field.

5.2.5. Recommendations and Future Work

For the Visit Request Portal, the first recommendation is to collect feedback after internal use. End users, admins, managers, and operations users may identify workflow improvements that are difficult to predict during development. This feedback can guide future refinements.

Second recommendation is to expand reporting. The current reporting module already provides useful insight into visitor activity, but future versions could add more filters, export options, department-level reporting, and comparison between periods.

Third recommendation is to further improve audit logging. The system already stores important status changes and historical information, but a more detailed audit trail could help with compliance, troubleshooting, and operational analysis.

Another recommendation is to continue improving the Operations workflow. Room checklist functionality and the Operations calendar provide a strong basis, but future versions could add preparation status, task assignment, room availability checks, or notifications when room preparation is completed.

For the R&D Formulation Database, the most important future improvement is direct editing of existing formulations. At MVP stage, users can copy an existing formulation and create a new version, but a dedicated edit workflow would make the application more complete.

The second important improvement for the R&D application is export functionality. Export to Excel or CSV would allow users to analyze formulation data outside the system and share selected data when needed.

The third improvement is scalability for larger datasets. Pagination, sortable columns, optimized queries, and caching should be added before broader production use.

Other future improvements include automated testing, improved accessibility, centralized toast notifications, print-friendly pages, backup and recovery planning, and additional production monitoring.

5.2.6. Closing Remarks

This internship resulted in the successful realization of the Visit Request Portal as a secure, cloud-based internal application for Soudal. The application addresses the original business problem by centralizing visitor scheduling, introducing structured approval workflows, improving visibility, supporting operational preparation, and providing reporting and historical tracking.

The project also provided valuable practical experience in enterprise software development. It required technical implementation, architectural decision-making, stakeholder communication, testing, deployment automation, and documentation. The result is not only a working application, but also a maintainable foundation that can be extended in the future.

The additional R&D Formulation Database MVP further strengthened the internship outcome. Although it was outside the original assignment, it provided another useful internal tool and demonstrated that the technical knowledge gained during the main project could be applied to a different business process.

Overall, the internship achieved its main objectives and delivered more than the original project scope. The Visit Request Portal is ready as a scalable foundation for internal visitor management, and the R&D Formulation Database provides a practical starting point for structured formulation management within the R&D department.

Use OF AI Tools

During the documentation and development process, AI tools were used as supporting tools to improve efficiency, clarity, and structure. ChatGPT was used to review and improve the wording of documentation, restructure text into a clearer professional format, and support the correction of grammar, sentence flow, and explanation quality. The generated suggestions were not used blindly; they were reviewed, adapted, and aligned with the actual project scope, requirements, and implementation.

GitHub Copilot was also used in Visual Studio Code during development, in accordance with the company's policy on AI-assisted coding. Copilot supported the writing of code snippets, repetitive logic, and implementation suggestions. However, all AI-generated code was manually reviewed, adjusted where necessary, and tested before being accepted into the project. The final responsibility for the correctness, security, maintainability, and suitability of the code remained with me.

The AI tools were therefore used only as assistance tools, not as replacements for analysis, decision-making, or validation. All final documentation and code were checked against the actual project requirements and project implementation before being included in the final deliverables.

Reference list

Chart.js. (2025). *Chart.js*. Retrieved from Chart.js: https://www.chartjs.org/docs/latest/

Microsoft. (2026). *App Service documentation*. Retrieved from Microsoft Learn: https://learn.microsoft.com/en-us/azure/app-service/

Microsoft. (2026). *Azure Key Vault*. Retrieved from Microsoft Learn: https://learn.microsoft.com/en-us/azure/key-vault/

Microsoft. (2026). *Azure Pipelines documentation*. Retrieved from Microsoft Learn: https://learn.microsoft.com/en-us/azure/devops/pipelines/?view=azure-devops

Microsoft. (2026). *Bicep documentation*. Retrieved from Microsoft Learn: https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/

Microsoft. (2026). *Entity Framework Core*. Retrieved from Microsoft Learn: https://learn.microsoft.com/en-us/ef/core/

Microsoft. (2026). *Microsoft Graph documentation*. Retrieved from Microsoft Learn: https://learn.microsoft.com/en-us/graph/

Microsoft. (2026). *Microsoft identity platform documentation*. Retrieved from Microsoft Learn: https://learn.microsoft.com/en-us/entra/identity-platform/

Microsoft. (2026). *Overview of ASP.NET Core SignalR*. Retrieved from Microsoft Learn: https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-10.0

Microsoft. (2026). *Razor Pages architecture and concepts in ASP.NET Core*. Retrieved from Microsoft Learn: https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-10.0&tabs=visual-studio

List Of Figures

List Of Tables

Attachments

1. Project plan.docx
2. Admin user manual visit request portal.docx
3. All users manual visit request portal.docx
4. Manager user manual visit request portal.docx
5. Operations user manual visit request portal.docx