University Course Management System (UniManage)

Application Development Coursework 2 Report

Module: CS6004ES Application Development

Student Name: Yasas Pasindu Fernando

Student ID: 25026764

Submission Type: Approved individual submission

Declaration

Module: CS6004ES **Deadline: 05/15/2026**

Module Leader: Mr. Migara Alawatta **Student ID: 25026764**

PLAGIARISM

You are reminded that there exist regulations concerning plagiarism. Extracts from these regulations are printed below. Please sign below to say that you have read and understand these extracts:

(signature:)__yasaspasindufernando@gmail.com_____ Date: 05/14/2026

This header sheet should be attached to the work you submit. No work will be accepted without it.

Extracts from University *Regulations* on Cheating, Plagiarism and Collusion

Section 2.3: "The following broad types of offence can be identified and are provided as indicative examples…

(i) Cheating: including taking unauthorised material into an examination; consulting unauthorised material outside the examination hall during the examination; obtaining an unseen examination paper in advance of the examination; copying from another examinee; using an unauthorised calculator during the examination or storing unauthorised material in the memory of a programmable calculator which is taken into the examination; copying coursework.

(ii) Falsifying data in experimental results.

(iii) Personation, where a substitute takes an examination or test on behalf of the candidate. Both candidate and substitute may be guilty of an offence under these Regulations.

(iv) Bribery or attempted bribery of a person thought to have some influence on the candidate's assessment.

(v) Collusion to present joint work as the work solely of one individual.

(vi) Plagiarism, where the work or ideas of another are presented as the candidate's own.

(vii) Other conduct calculated to secure an advantage on assessment.

(viii) Assisting in any of the above.

Some notes on what this means for students:

1. Copying another student's work is an offence, whether from a copy on paper or from a computer file, and in whatever form the intellectual property being copied takes, including text, mathematical notation and computer programs.

2. Taking extracts from published sources *without attribution* is an offence. To quote ideas, sometimes using extracts, is generally to be encouraged. Quoting ideas is achieved by stating an author's argument and attributing it, perhaps by quoting, immediately in the text, his or her name and year of publication, e.g. "$e = mc^2$ (Einstein 1905)". A *references* section at the end of your work should then list all such references in alphabetical order of authors' surnames. (There are variations on this referencing system which your tutors may prefer you to use.) If you wish to quote a paragraph or so from published work then indent the quotation on both left and right margins, using an italic font where practicable, and introduce the quotation with an attribution.

Acknowledgement

I thank the lecturer and module team for their guidance during CS6004ES Application Development and for approving the individual completion route. I also used official documentation for ASP.NET Core, Entity Framework Core, Pomelo MySQL, QuestPDF, Bootstrap, and related tools when checking technical decisions.

I also acknowledge ESOFT Metro Campus and London Metropolitan University for the module structure and academic guidance around the submission.

Abstract

UniManage is a University Course Management System developed for CS6004ES Application Development Coursework 2 as an ASP.NET Core MVC web application targeting .NET 8. The system supports common academic administration work through registration and login, role-based dashboards, course management, enrolment, assignments, submissions, grading, reports, messages, meetings, audit logs, and practical security controls.

Source-code review confirmed the main MVC structure, Identity authentication, EF Core data access, reporting helpers, audit logging, and security configuration. The project includes thirteen controllers, twelve domain models, eighteen view models, thirteen infrastructure components, fifty Razor views, four Entity Framework Core migrations, an ApplicationDbContext that inherits from IdentityDbContext, and a Program.cs configuration for Identity, EF Core with Pomelo MySQL, rate limiting, anti-forgery, exception handling, and HTTPS redirection. The build has been verified locally with dotnet build --no-restore, producing zero warnings and zero errors.

The report explains the academic problem, requirements, design, implementation, testing, evaluation, limitations, and reflection in a way that supports marking and viva discussion. It is written as an approved individual submission.

Table of Contents

List of Figures

List of Tables

Abbreviations and Glossary

AD: Application Development

API: Application Programming Interface

ASP.NET: Active Server Pages .NET

CDN: Content Delivery Network

CRUD: Create, Read, Update, Delete

CSRF: Cross-Site Request Forgery

CSS: Cascading Style Sheets

CSV: Comma-Separated Values

DI: Dependency Injection

DB: Database

DTO: Data Transfer Object

EF Core: Entity Framework Core

ERD: Entity Relationship Diagram

HSTS: HTTP Strict Transport Security

HTML: HyperText Markup Language

HTTPS: HyperText Transfer Protocol Secure

ICS: iCalendar file format used to export meetings

ID: Identifier

JS: JavaScript

JSON: JavaScript Object Notation

MVC: Model View Controller

ORM: Object Relational Mapping

OWASP: Open Web Application Security Project

PDF: Portable Document Format

RBAC: Role-Based Access Control

SDK: Software Development Kit

SQL: Structured Query Language

UCMS: University Course Management System

UI: User Interface

URL: Uniform Resource Locator

UTC: Coordinated Universal Time

UX: User Experience

Glossary terms:

ApplicationUser: the project's user entity that extends IdentityUser and stores additional profile fields such as full name and phone number.

ApplicationDbContext: the EF Core context used by UniManage. It inherits from IdentityDbContext<ApplicationUser> and exposes DbSets for the academic entities.

DbInitializer: the seeding component that applies migrations and creates demonstration roles, users, and sample data.

AppRoles: a static class that holds the role name constants used by the project (Administrator, Lecturer, Student).

ViewModel: a class designed for transferring data between controllers and views, used to keep entity classes separate from presentation concerns.

Razor view: a .cshtml server-side template that combines HTML markup with C# logic to render the user interface.

1. Introduction

1.1 Background and Coursework Context

CS6004ES Application Development at ESOFT Metro Campus and London Metropolitan University focuses on building a working web application, not only a design proposal. Coursework 2 requires the design, implementation, testing, and documentation of a university-style management system that supports realistic academic workflows. The marking items include the user interface, registration and login, role dashboards, course management, enrolment, assignment and grading, reporting, communication, security, validation, diagrams, user manual, class and method descriptions, reflection, and programming style.

UniManage was built by the author to address this brief. The application uses ASP.NET Core MVC, C#, Visual Studio, Entity Framework Core, Pomelo MySQL, ASP.NET Core Identity, Razor views, and Bootstrap. These technologies matched the coursework because they support MVC structure, database persistence, authentication, validation, and a browser-based interface in one project.

The system supports three actors. A Student can register, log in, view a personal dashboard, browse courses, enrol on eligible courses, view and submit assignments, view grades and feedback, send and receive messages with lecturers, and view scheduled meetings. A Lecturer can log in, view a teaching dashboard, manage assignments for owned courses, upload course materials, grade submissions, send and receive messages with enrolled students, and schedule meetings. An Administrator can log in, view a system-wide dashboard, manage courses, manage user accounts, view audit logs, and produce reports.

1.2 Purpose of the Report

This report gives the marker enough evidence to understand what was built, how it works, and how the implementation addresses the coursework brief.

It explains the academic problem that UniManage solves and the way that problem has been broken down into actors, use cases, and software components.

It documents the architecture, database design, and implementation of the system, with reference to actual source files in the project.

It records the validation, exception handling, and security controls that have been implemented in the code.

It records the test plan and the verification activities that have been carried out.

It evaluates the implementation against the marking items in the coursework brief.

It reflects on the author's experience of completing the work as an approved individual submission and identifies future improvements.

The report is a technical record with reflection, rather than only a feature list. Direct source-code references are used to keep the writing honest, and the reflection chapter explains the learning from the approved individual completion route.

1.3 Approved Individual Submission Context

Following approval from the lecturer and module team, the author completed the implementation, testing, documentation, and final preparation as an individual submission. The report keeps this context neutral and evidence based. It does not name other students or compare work between students. The author refers to themselves as "the author" in the body chapters and uses a more personal tone only in the reflection chapter.

This approach keeps the report fair and easy to mark. Completed features are linked to source code, build output, screenshots, diagrams, and supporting appendices.

1.4 Scope of the UniManage Application

The scope of UniManage is defined by the actors that the system supports, by the academic workflows that the system implements, and by the boundaries that the system deliberately does not cross.

The system is in scope when:

A user registers, logs in, logs out, or recovers a password.

A Student browses, enrols, views or submits an assignment, views grades, sends or receives messages, or views meetings.

A Lecturer manages assignments for an owned course, uploads materials, grades submissions, sends or receives messages, or schedules meetings for an owned course.

An Administrator manages the course catalogue, views reports, and reviews audit logs.

The system is out of scope when:

An external timetable system is required to import or export class schedules.

A finance or fees subsystem is required.

A live video conference is required inside the application; UniManage stores meeting URLs and joining links rather than hosting video conferences.

A mobile application is required; UniManage is a responsive web application accessed through a browser.

A complex content management system for marketing pages is required; UniManage focuses on academic operations.

The boundary is enforced both at the user interface and at the controller layer. Each controller action is restricted to the appropriate role through Authorize attributes, and ownership checks are applied where a lecturer must only see or modify their own course or assignment. The system boundary is described clearly, and features outside the final scope are recorded as limitations or future improvements.

1.5 Report Structure

The report is structured into front matter, numbered technical chapters, supporting appendices, an individual contribution report, and an installation/user manual. After the front matter, the report introduces the project, explains the academic problem, defines the aim and scope, records the requirements, explains the architecture, database design, implementation, user interface, class structure, logical solution, validation, security, programming style, testing, challenges, limitations, reflection, future improvements, conclusion, references, and appendices. The supporting sections then provide individual contribution evidence, user guidance, source/build links, final evidence summary, selected unit-test evidence, and selected unit-test code examples.

2. Problem Description and Case Study Analysis

2.1 Academic Administration Problem

A modern university produces and processes a large amount of academic information every day. Students need to know which courses they are enrolled on, what assignments are due, and what feedback they have received. Lecturers need to know which courses they teach, which students are enrolled on those courses, what assignments have been set, and what submissions are pending review. Administrators need to know which courses run in the current academic period, which lecturers are teaching, how many students are enrolled, how many assignments have been set, and whether the system is being used securely.

When this information is held in scattered spreadsheets, paper records, or unconnected tools, errors creep in. Students enrol twice on the same course. Capacity is exceeded because there is no automatic check. Submissions are lost because there is no central record of who submitted what and when. Grades are inconsistent because feedback is held in private files. Communication between lecturer and student is split across email, instant messengers, and printed notices. Reports are produced manually at the end of the academic period and are sometimes inaccurate.

UniManage addresses this academic administration problem by providing a single web application that records the academic data, enforces business rules through the controller layer, and presents role-aware dashboards to each user. The intention is not to replace specialist systems for finance, payroll, or live video conferencing. The intention is to provide a clean, secure, and reliable foundation for the academic operations of a university course management system.

2.2 Stakeholder Analysis

The stakeholders for UniManage are the people who use the system, the people who run the system, and the people who depend on the data that the system produces. Table 1 summarises the main stakeholders, their roles, their main goals, and the way that the system supports each goal.

Stakeholder	Role	Main Goals	How UniManage Supports the Goals
Student	End user	Find courses, enrol on eligible courses, complete assignments, view grades, communicate with lecturers, plan meetings.	Student dashboard with enrolled courses, deadlines, grades, messages, calendar. Course browse with eligibility checks. Submission and grading screens. Inbox with allowed-recipient messaging.
Lecturer	End user	Manage owned courses, set and grade assignments, share materials, communicate with enrolled students, plan meetings.	Lecturer dashboard with teaching summary, pending submissions, recent activity. MyCourses view, assignment management, grading screen, materials upload, scheduling, inbox.
Administrator	End user	Manage the course catalogue, view system-wide reports, oversee security, review audit logs.	Administrator dashboard with system counts and popular courses, course CRUD, reporting suite, audit log review.
Module leader / lecturer team	Marker / overseer	Verify that the application meets the coursework brief and that the documentation is evidence based.	Marking-aligned report, final evidence summary, build verification, source-code references.
ESOFT / London Met administration	Internal stakeholder	Confirm coursework completion in line with academic standards.	Approved individual submission record, declaration, formal report.
Author	Developer	Deliver a complete and verifiable solution as an approved individual submission.	Source code, documentation, screenshots, evidence appendices.

Table 1: Stakeholder Analysis

The marker is treated as a primary reader of the report. The system has therefore been described in a marker-friendly way, with cross-references between requirements, source code,

screenshots, and tests. The report uses source-code references, screenshots, diagrams, build evidence, and tests to keep the evidence clear and traceable.

2.3 Problems Faced by Students

Students typically face several problems when academic information is scattered or inconsistent. They struggle to find a complete list of available courses with their prerequisites and capacity. They often miss deadlines because there is no single calendar of due dates. They cannot easily verify that an enrolment has been recorded. They cannot see grades and feedback in a structured format and instead rely on email threads. They have no clear way to message a lecturer in a record-keeping format that respects ownership and privacy.

UniManage addresses these problems through the Student dashboard, the course browse view, the enrolment workflow with capacity, duplicate, and prerequisite checks, the assignment and submission screens, the grading view, the inbox, and the meetings calendar. Each of these features is accessible from a single signed-in session, which removes the cognitive load of switching between unrelated tools.

2.4 Problems Faced by Lecturers

Lecturers also face common problems. They need to know which students are enrolled on each course they teach. They need to record grades and feedback in a structured way that the student can verify. They need to share teaching materials in a controlled environment that respects copyright and access rules. They need to plan meetings, including online meetings, and provide joining links to the correct students. They need to communicate with students in a way that is respectful of role boundaries.

UniManage addresses these problems through the Lecturer dashboard, MyCourses, the assignment management screens, the grading screen, the course material upload, the meeting scheduler, and the messaging module restricted to allowed recipients. The combination of ownership filters and authorisation attributes keeps each lecturer focused on the data that belongs to them.

2.5 Problems Faced by Administrators

Administrators are responsible for the integrity of the academic catalogue and for visibility over how the system is used. They face problems such as duplicated course codes, courses without lecturers, courses set with impossible enrolment limits, and a lack of audit traceability for security-sensitive actions.

UniManage addresses these problems through the Administrator dashboard with system counts and popular courses, the Admin User Management screens, the course CRUD screens with unique-code validation, the reporting suite (course popularity, student performance, lecturer workload, enrolments, pass and fail, assignment attendance), the audit log review screen, and the export options for CSV and PDF reports.

2.6 Success Criteria for the Solution

The success criteria for UniManage are derived from the coursework brief and from the stakeholder analysis. They are summarised below.

- The application can be built locally with dotnet “build --no-restore” without errors.
- The application starts on https://localhost:7212 or http://localhost:5103 and seeds demonstration roles and users automatically.
- A user can register, log in, and log out.
- A Student can browse courses, enrol on an eligible course, submit an assignment, view grades, message a lecturer linked to an enrolled course, and view meetings.
- A Lecturer can manage assignments for an owned course, grade a submission, upload a course material, schedule a meeting for an owned course, and message a student enrolled on an owned course.
- An Administrator can list, create, edit, and delete courses, view system-wide reports, and review the audit log.
- Validation, anti-forgery, rate limiting, secure cookies, and security headers are present and active.
- Build, screenshot, diagram, repository, and database evidence have been collected for the final submission.

The chapters that follow show how UniManage meets each of these success criteria and how the coursework brief has been addressed.

3. Aim, Objectives and Functional Scope

3.1 Aim

The aim of the project is to design, implement, document, and evaluate UniManage as an ASP.NET Core MVC University Course Management System that supports role-based academic workflows, enforces appropriate security and validation, and is delivered as an approved individual submission for CS6004ES Application Development Coursework 2.

3.2 Objectives

The aim is broken down into the following SMART objectives:

O1. To implement registration and login using ASP.NET Core Identity, with strong password rules, account lockout, anti-forgery, and audit logging on success and failure.

O2. To implement role-based access for Student, Lecturer, and Administrator users, with [Authorize] attributes on controllers and dashboard redirection by role.

O3. To provide a Student dashboard, a Lecturer dashboard, and an Administrator dashboard that present role-relevant data using EF Core queries and view models.

O4. To implement course management, including create, read, update, delete, search, paging, lecturer assignment, prerequisite handling, capacity, and material upload.

O5. To implement an enrolment workflow with capacity, duplicate, and prerequisite checks.

O6. To implement assignment management, submission upload, and grading with feedback, score range validation, and ownership checks.

O7. To implement reporting and analytics including course popularity, student performance, lecturer workload, enrolments, pass and fail, and assignment attendance, with CSV and PDF export support.

O8. To implement an internal communication module with inbox, thread, reply, compose, and allowed-recipient validation, plus meeting scheduling and ICS export.

O9. To implement validation, exception handling, anti-forgery, rate limiting, secure cookies, security headers, and audit logging.

O10. To prepare a complete documentation package including a main report, an individual contribution report, a user manual, final evidence summary, diagrams, screenshots, source/build links, and supporting appendices.

O11. To submit the final coursework as an approved individual submission, with evidence-based wording and neutral individual-submission material.

3.3 Functional Scope

The functional scope is the set of features that are inside the system boundary. Table 2 summarises these features.

ID	Feature	Source-Code Evidence
FS1	User registration, login, logout, password reset	Controllers/AccountController.cs, ViewModels/RegisterViewModel.cs, ViewModels/LoginViewModel.cs, ViewModels/ForgotPasswordViewModel.cs, ViewModels/ResetPasswordViewModel.cs, Views/Account/
FS2	Optional Google login if configured	Program.cs (Google auth registration), Controllers/AccountController.cs
FS3	Role-based access control	Models/AppRoles.cs, [Authorize] attributes in controllers, Data/DbInitializer.cs
FS4	Student dashboard	Controllers/DashboardController.cs, ViewModels/DashboardViewModels.cs, Views/Dashboard/Student.cshtml
FS5	Lecturer dashboard	Controllers/DashboardController.cs, ViewModels/DashboardViewModels.cs, Views/Dashboard/Lecturer.cshtml

ID	Feature	Source-Code Evidence
FS6	Administrator dashboard	Controllers/DashboardController.cs, ViewModels/DashboardViewModels.cs, Views/Dashboard/Administrator.cshtml
FS7	Course management with CRUD, search, paging	Controllers/CoursesController.cs, Models/Course.cs, ViewModels/CourseInputViewModel.cs, Views/Courses/
FS8	Course material upload	Controllers/CoursesController.cs upload action, Models/CourseMaterial.cs, ViewModels/MaterialUploadViewModel.cs, Infrastructure/UploadedFileStore.cs
FS9	Enrolment with capacity, duplicate, and prerequisite checks	Controllers/EnrollmentsController.cs, Models/Enrollment.cs, ViewModels/CourseBrowseRowViewModel.cs, Views/Enrollments/Browse.cshtml
FS10	Assignment management for lecturers	Controllers/AssignmentsController.cs, Models/Assignment.cs, ViewModels/AssignmentInputViewModel.cs, Views/Assignments/
FS11	Submission upload and grading	Controllers/SubmissionsController.cs, Models/Submission.cs, Models/SubmissionStatus.cs, ViewModels/SubmissionSubmitViewModel.cs, ViewModels/GradeSubmissionViewModel.cs, Views/Submissions/
FS12	Reporting and analytics with CSV and PDF	Controllers/ReportsController.cs, Infrastructure/CsvWriter.cs, Infrastructure/PdfReport.cs, Views/Reports/
FS13	Internal messaging module	Controllers/MessagesController.cs, Models/Message.cs, ViewModels/MessageInboxRowViewModel.cs, ViewModels/MessageThreadViewModel.cs, ViewModels/MessageComposeViewModel.cs, Views/Messages/
FS14	Meeting scheduling and ICS export	Controllers/MeetingsController.cs, Models/Meeting.cs, ViewModels/MeetingInputViewModel.cs, Infrastructure/CalendarLink.cs, Infrastructure/MeetLinkGenerator.cs, Views/Meetings/

ID	Feature	Source-Code Evidence
FS15	Profile management and password change	Controllers/ProfileController.cs, ViewModels/ProfileViewModel.cs, Views/Profile/
FS16	Audit logging and audit log review	Models/AuditLog.cs, Infrastructure/AuditLogger.cs, Infrastructure/IAuditLogger.cs, Controllers/AuditLogsController.cs, Views/AuditLogs/Index.cshtml
FS17	Security pipeline	Program.cs (Identity, anti-forgery, rate limiting, HTTPS, HSTS), Infrastructure/SecurityHeadersMiddleware.cs
FS18	Email service for password reset	Infrastructure/IEmailService.cs, Infrastructure/SmtpEmailService.cs, Infrastructure/EmailSettings.cs, Infrastructure/EmailTemplates.cs

Table 2: Functional Scope Inclusions

Features that are deliberately outside the final scope are explained in Section 1.4. Attendance tracking, announcements, formal production hardening, long-term monitoring, and wider integration testing are recorded as limitations or future improvements rather than described as completed features.

3.4 Non-Functional Scope

Table 3 summarises the non-functional quality attributes that the system aims to satisfy.

ID	Quality Attribute	Description	Evidence in Project
NF1	Usability	Role-aware navigation, dashboards, alert partial, validation feedback.	Views/Shared/_Layout.cshtml, Views/Shared/_Alerts.cshtml, role views, Bootstrap.
NF2	Performance	EF Core queries use filtering, paging, and AsNoTracking for read-only views; rate limiting limits abusive traffic.	EF Core queries in controllers, Program.cs rate limiting setup.
NF3	Reliability	Retry on failure for MySQL connection, exception handling middleware, audit logging of failures.	Program.cs EnableRetryOnFailure, UseExceptionHandler, AuditLogger.
NF4	Security	Identity, RBAC, anti-forgery, rate limiting, security headers, secure cookies, file validation, audit logging.	Program.cs, Infrastructure/SecurityHeadersMiddleware.cs, controller checks.
NF5	Maintainability	Layered MVC structure with Controllers, Models, ViewModels, Views, Data, Infrastructure folders, dependency injection, single-responsibility services.	Folder layout, Program.cs service registration.
NF6	Data integrity	EF Core relationships, unique course code,	Data/ApplicationDbContext.cs, model annotations.

ID	Quality Attribute	Description	Evidence in Project
		unique enrolment per student/course, data annotation rules.	
NF7	Accessibility	Skip link, semantic markup, label association in forms, Bootstrap responsive layout. Basic accessibility evidence is included; a formal accessibility audit is listed as a future improvement.	_Layout.cshtml, role views.
NF8	Auditability	Audit logger records security and academic events with user identifier, action, detail, and success flag.	Infrastructure/AuditLogger.cs, Models/AuditLog.cs, Controllers/AuditLogsController.cs.

Table 3: Non-Functional Scope and Quality Attributes

3.5 Assumptions and Constraints

Table 4 lists the assumptions and constraints that have shaped the project.

Type	Description
Assumption	The assessment environment is expected to support Visual Studio 2022 or a compatible version, .NET 8 SDK, and a database configuration suitable for running the application. The connection string can be adjusted during setup if a different local database environment is used.
Assumption	The submitted implementation uses MySQL with the Pomelo EF Core provider. This is documented clearly in the installation guide, and the database setup steps are provided for the submitted configuration.
Assumption	The seeded demonstration users defined in Data/DbInitializer.cs are available for testing, demonstration, and marker review.
Assumption	The approved individual submission route documented in Section 1.3 is accepted as the context for this report.
Constraint	The assessment environment will support Visual Studio 2022 or a compatible version, .NET 8 SDK, and a database configuration suitable for running the application. The database connection string can be adjusted during setup if the marker uses a different local database environment.
Constraint	The author works within a coursework deadline and reports only what has been implemented; no fictional features are described.
Constraint	Sensitive configuration values such as database passwords, SMTP passwords, and Google client secrets are not included in screenshots or in the body of the report.
Constraint	The audit log must not store free-text personal data beyond what is needed for academic traceability.

Table 4: Assumptions and Constraints

4. Research and Background

4.1 ASP.NET Core MVC

ASP.NET Core MVC was suitable for UniManage because the coursework needed a clear separation between data models, request handling, and Razor views. In this project, controllers handle workflows such as courses, enrolments and submissions; models represent database entities; and views present role-specific pages.

UniManage uses the conventional MVC routing pattern {controller}/{action}/{id?} configured in Program.cs. Each controller is registered through AddControllersWithViews, and views are rendered through Razor. The author has used the same MVC structure across all features, so the marker can move from one controller to another without unfamiliar conventions.

4.2 C# and .NET

C# and .NET 8 provide the runtime and language features used across UniManage. The project uses asynchronous controller actions, LINQ queries for EF Core, dependency injection, and strongly typed model and view-model classes. This kept the implementation consistent with Visual Studio and normal ASP.NET Core practice.

C# language features are used throughout UniManage. For example, asynchronous controller actions return Task<IActionResult>, EF Core queries use LINQ, and validation messages use string interpolation. The use of nullable reference types in the project (where present) helps to identify potentially null values during compilation rather than at runtime.

4.3 Razor Views and Tag Helpers

Razor views were used because they fit naturally with ASP.NET Core MVC. In UniManage, Razor Tag Helpers bind form fields to view-model properties, generate anti-forgery tokens, and display validation messages. The shared layout _Layout.cshtml provides the navigation bar, alert partial, main content area, and footer, while role-specific views provide the actual academic screens.

This helped keep the interface code consistent. For example, form views can use the same asp-for and asp-validation-for pattern across registration, courses, assignments, submissions, grades, messages, and meetings.

4.4 Entity Framework Core and MySQL

UniManage connects to a MySQL 8 database through the Pomelo Entity Framework Core MySQL provider. The decision to use MySQL rather than SQL Server reflects the local availability of MySQL on the development environment and the popularity of MySQL in industry. The EF Core configuration in Program.cs enables retry on failure with up to five retries and a thirty second maximum delay, which protects the application from transient connection issues. The Pomelo provider is registered through UseMySql and is targeted at MySQL 8.0.36.

EF Core migrations are stored under Data/Migrations/. The current migrations are:

- 20260408041036_InitialUniManage.cs
- 20260501122117_AddAuditLogAndSecurity.cs
- 20260501125135_AddMeetings.cs
- 20260506142007_AddApplicationUserDateOfBirth.cs

The application calls DbInitializer.SeedAsync at startup, which applies pending migrations and creates demonstration roles, users, and sample data. The marker can therefore start from a clean database and obtain a working environment automatically.

4.5 ASP.NET Core Identity and Role-Based Access Control

ASP.NET Core Identity was used because UniManage needs normal account features such as password hashing, sign-in, sign-out, lockout, external login, and role membership. UniManage registers Identity through AddIdentity<ApplicationUser, IdentityRole> in Program.cs. The configuration enforces a minimum password length of 8 characters, requires at least one digit, one uppercase letter, and one lowercase letter, allows lockout for new users with up to 5 failed attempts, and uses a 15 minute lockout window.

Role-based access control is provided by the Identity role system. Three roles are seeded by DbInitializer: Administrator, Lecturer, and Student. Each controller action that requires a role uses [Authorize(Roles = AppRoles.X)], where AppRoles is a static class that holds the role name constants. The author has used a constants class instead of magic strings so that role names are consistent across the codebase.

Identity also issues an authentication cookie configured through ConfigureApplicationCookie. The cookie is HTTP-only, uses SameSiteMode.Lax, and is marked secure when the request itself is secure. The session length is eight hours with sliding expiration. These settings reduce the chance of cookie theft and session hijacking while still allowing a normal academic session.

4.6 Bootstrap and Web User Interface Design

Bootstrap was used to keep the interface clear and responsive without building a custom design system. UniManage uses Bootstrap classes through the layout file and role views for spacing, tables, cards, alerts, buttons, and the collapsing navigation bar.

The layout file _Layout.cshtml defines the main navigation bar, exposes the Razor sections for content, and includes the alerts partial. A custom CSS file wwwroot/css/site.css adds project-specific styling. A custom JavaScript file wwwroot/js/site.js is also present for any client-side behaviour. Bootstrap utility classes are used to maintain consistent spacing, typography, and colour usage across pages.

4.7 Software Engineering Principles Applied

UniManage applies a small set of software engineering principles that are visible in the codebase.

Separation of concerns. The project separates models (data and validation rules), view models (presentation data), controllers (request flow and business rules), views (HTML rendering), data access (ApplicationDbContext, migrations, seeding), and cross-cutting infrastructure (audit logger, email service, file store, security headers, CSV writer, PDF report). Each folder has a clearly defined responsibility.

Single responsibility. Each controller is focused on one academic area, and each infrastructure service is focused on one concern. For example, AuditLogger is only responsible for writing audit records, CsvWriter is only responsible for writing CSV output, and PdfReport is only responsible for building PDF reports through QuestPDF.

Layered architecture. Requests flow through middleware (security headers, HTTPS redirection, rate limiting), then through Identity, then through the controller, then through EF Core, and finally back through the view to the browser.

Defence in depth. Security is implemented at the network layer (HTTPS, HSTS), the request layer (rate limiting, anti-forgery), the application layer (Identity, role attributes, ownership checks, validation), the data layer (EF Core parameterised queries, unique constraints), and the audit layer (audit log).

Convention over configuration. Where ASP.NET Core MVC provides sensible defaults (routing, model binding, view discovery), the author has used those defaults instead of overriding them. This keeps the codebase smaller and easier to maintain.

Asynchronous I/O. Database, file, and email operations are asynchronous, which keeps the application responsive under load.

4.8 Security and Data Protection Background

UniManage handles academic and account data, so basic web security is important. The main concerns are unauthorised access, weak authentication, CSRF, unsafe file access, SQL injection, XSS, and accidental exposure of configuration values.

UniManage relies on Identity for authentication, on the [Authorize] attribute for Authorisation, on parameterised EF Core queries for SQL injection protection, on the Razor engine's automatic HTML encoding for XSS protection, on anti-forgery tokens for CSRF protection, on HTTPS and HSTS for transport security, on a custom security headers middleware for defence in depth, and on the audit logger for traceability. Sensitive configuration values are supplied through local configuration or environment variables and are excluded from the report. Chapter 13 explains each control in more detail.

5. Requirement Analysis

5.1 Requirement Derivation

Requirements were derived from the coursework brief, the UniManage case study, and a review of the code that is actually present. This was important because the report must show implemented evidence rather than expected features.

Most requirements are supported by source files, configuration, screenshots, diagrams, or testing evidence. Items outside the final scope are recorded as limitations or future improvements rather than described as completed features.

5.2 Functional Requirements

Table 5 lists the functional requirements that the system supports. Each requirement is referenced from later chapters and from the testing chapter.

ID	Functional Requirement	Source-Code Evidence
FR1	The system shall allow a user to register an account with role selection (Student or Lecturer).	Controllers/AccountController.cs (Register), ViewModels/RegisterViewModel.cs, Views/Account/Register.cshtml
FR2	The system shall allow a registered user to log in using a strong password.	Controllers/AccountController.cs (Login), ViewModels/LoginViewModel.cs, Views/Account/Login.cshtml, Program.cs Identity setup
FR3	The system shall lock the account after 5 failed login attempts for 15 minutes.	Program.cs lockout configuration
FR4	The system shall allow a user to log out and invalidate the authentication cookie.	Controllers/AccountController.cs (Logout), Program.cs cookie configuration
FR5	The system shall allow a user to request a password reset and to reset the password using a token.	Controllers/AccountController.cs (ForgotPassword, ResetPassword), ViewModels/ForgotPasswordViewModel.cs, ViewModels/ResetPasswordViewModel.cs
FR6	The system shall allow optional Google login if	Program.cs Google authentication block

ID	Functional Requirement	Source-Code Evidence
	Google client id and secret are configured.	
FR7	The system shall route the signed-in user to the dashboard that matches the assigned role.	Controllers/DashboardController.cs (Index, Student, Lecturer, Administrator)
FR8	The system shall display a Student dashboard summarising enrolled courses, deadlines, grades, messages, calendar, and meetings.	Controllers/DashboardController.cs (Student), ViewModels/DashboardViewModels.cs, Views/Dashboard/Student.cshtml
FR9	The system shall display a Lecturer dashboard summarising teaching courses, pending submissions, recent activity, messages, and meetings.	Controllers/DashboardController.cs (Lecturer), Views/Dashboard/Lecturer.cshtml
FR10	The system shall display an Administrator dashboard summarising system counts, popular courses, and audit activity.	Controllers/DashboardController.cs (Administrator), Views/Dashboard/Administrator.cshtml
FR11	The system shall allow an Administrator to create, read, update, delete, search, and page through courses.	Controllers/CoursesController.cs, Models/Course.cs, ViewModels/CourseInputViewModel.cs, Views/Courses/
FR12	The system shall allow a Lecturer to view and access only the courses they own.	Controllers/CoursesController.cs (MyCourses), ownership filter checks
FR13	The system shall allow a Lecturer to upload course material files for an owned course.	Controllers/CoursesController.cs (UploadMaterial), Models/CourseMaterial.cs, ViewModels/MaterialUploadViewModel.cs, Infrastructure/UploadedFileStore.cs
FR14	The system shall allow a Student to browse courses, see eligibility, and enrol on an open course.	Controllers/EnrollmentsController.cs (Browse, Enroll), ViewModels/CourseBrowseRowViewModel.cs, Views/Enrollments/Browse.cshtml
FR15	The system shall prevent duplicate enrolment,	Controllers/EnrollmentsController.cs business rule checks

ID	Functional Requirement	Source-Code Evidence
	exceeding capacity, and missing prerequisites.	
FR16	The system shall allow a Lecturer to create, edit, and delete assignments for an owned course.	Controllers/AssignmentsController.cs (Create, Edit, Delete, ForCourse, Mine), Models/Assignment.cs, ViewModels/AssignmentInputViewModel.cs
FR17	The system shall allow a Student to submit text and file content for an assignment they are enrolled on.	Controllers/SubmissionsController.cs (Submit), Models/Submission.cs, ViewModels/SubmissionSubmitViewModel.cs, Views/Submissions/Submit.cshtml
FR18	The system shall allow a Lecturer to grade submissions and record feedback.	Controllers/SubmissionsController.cs (Grade), ViewModels/GradeSubmissionViewModel.cs, Views/Submissions/Grade.cshtml
FR19	The system shall allow secure download of submission and material files only by the rightful user.	Controllers/SubmissionsController.cs (CourseMaterialFile, SubmissionFile), Infrastructure/UploadedFileStore.cs
FR20	The system shall produce reports for course popularity, student performance, lecturer workload, enrolments, pass and fail, and assignment attendance.	Controllers/ReportsController.cs, Views/Reports/
FR21	The system shall export selected reports to CSV and PDF.	Infrastructure/CsvWriter.cs, Infrastructure/PdfReport.cs, Controllers/ReportsController.cs
FR22	The system shall allow internal messaging through inbox, thread, reply, and compose actions.	Controllers/MessagesController.cs, Models/Message.cs, Views/Messages/, message view models
FR23	The system shall enforce allowed-recipient rules between Students and Lecturers based on enrolments.	Controllers/MessagesController.cs Compose validation
FR24	The system shall allow Lecturers to schedule meetings for owned courses, including a join URL.	Controllers/MeetingsController.cs, Models/Meeting.cs, ViewModels/MeetingInputViewModel.cs, Views/Meetings/

ID	Functional Requirement	Source-Code Evidence
FR25	The system shall allow ICS export of scheduled meetings.	Controllers/MeetingsController.cs (Ics), Infrastructure/CalendarLink.cs
FR26	The system shall allow a user to view and update their profile and to change their password.	Controllers/ProfileController.cs, ViewModels/ProfileViewModel.cs, Views/Profile/
FR27	The system shall record audit log entries for security-sensitive and academic actions.	Models/AuditLog.cs, Infrastructure/AuditLogger.cs, Controllers/AuditLogsController.cs, Views/AuditLogs/Index.cshtml
FR28	The system shall display a friendly error page if an unhandled exception occurs in production.	Program.cs UseExceptionHandler, Views/Shared/Error.cshtml
FR29	The system shall apply rate limiting to authentication and upload endpoints.	Program.cs AddRateLimiter policies (auth, upload)
FR30	The system shall enforce anti-forgery tokens on POST forms.	Program.cs AddAntiforgery, [ValidateAntiForgeryToken] attributes

Table 5: Functional Requirements

5.3 Non-Functional Requirements

Table 6 lists the non-functional requirements with rationale and source-code evidence.

ID	Non-Functional Requirement	Rationale / Evidence
NFR1	Usability: Users should reach key tasks in a small number of clicks.	Role-aware navigation in _Layout.cshtml, role dashboards, alerts partial, consistent forms.
NFR2	Performance: Dashboard queries should return within a reasonable time on a typical academic dataset.	EF Core filtering and paging, AsNoTracking for read-only queries, retry on failure.
NFR3	Security: The application should resist common web threats such as CSRF, brute force login, and unauthorised access.	Identity, lockout, anti-forgery, rate limiting, security headers, HTTPS, HSTS.

ID	Non-Functional Requirement	Rationale / Evidence
NFR4	Maintainability: New features should fit into the existing structure without rewriting core code.	Layered MVC structure, dependency injection, single-responsibility services.
NFR5	Reliability: Transient database errors should not cause user-visible failures.	EF Core retry on failure, exception handling middleware.
NFR6	Data integrity: Domain rules such as unique course code and unique enrolment per student/course should be enforced at the database level.	EF Core unique indexes, fluent configuration in ApplicationDbContext.
NFR7	Accessibility: Forms should be usable with keyboard, semantic labels, and skip links.	_Layout.cshtml, label association in form views. Basic accessibility evidence is included; a formal accessibility audit is listed as a future improvement.
NFR8	Auditability: Important actions should be recorded with user, action, and detail.	AuditLogger and AuditLog entity.
NFR9	Configurability: Database connection, email settings, and Google credentials should be configurable through ASP.NET Core configuration providers, including environment variables for deployment.	Program.cs, docker-compose.yml, deployment.env.example.
NFR10	Deployability: The project should build with a single dotnet build --no-restore command.	Build verified locally with zero warnings and zero errors.

Table 6: Non-Functional Requirements

5.4 Use Case Summary

Table 7 summarises the main use cases. The diagram in Section 6.2 shows the visual layout.

Use Case	Primary Actor	Goal
Register Account	Visitor	Create a Student or Lecturer account.
Log In	Registered User	Access the role-appropriate dashboard.
Reset Password	Registered User	Recover access using an emailed token.
View Dashboard	Student / Lecturer / Administrator	See role-relevant academic information.
Browse Courses	Student	Find a course to enrol on.
Enrol on Course	Student	Add a course to the personal record subject to capacity, prerequisite, and duplicate checks.
Submit Assignment	Student	Upload submission text and file.
View Grades and Feedback	Student	Read marker feedback.
Send Message	Student / Lecturer	Communicate within an allowed relationship.
Manage Course	Administrator	Maintain the course catalogue.
Manage Assignment	Lecturer	Set or update assignments for an owned course.
Grade Submission	Lecturer	Record marks and feedback.
Upload Course Material	Lecturer	Provide teaching resources.
Schedule Meeting	Lecturer	Organise a class meeting for an owned course.
View Reports	Administrator	Review system-wide academic data.
Review Audit Log	Administrator	Inspect security-sensitive and academic events.
Update Profile	Any User	Maintain personal information.
Change Password	Any User	Update the password under authentication.

Table 7: Use Case Summary

5.5 Requirement Traceability Summary

Table 8 maps the marking items in the brief to the functional requirements that support them and to the implementation evidence.

Marking Item	Supporting Requirements	Source-Code Evidence	Test Cases
Application user interface	NFR1, NFR7	_Layout.cshtml, _Alerts.cshtml, role views, site.css	T1, T8, T9, T10, T11
User Registration and Login	FR1, FR2, FR3, FR4, FR5, FR6, FR30	AccountController, Program.cs	T1, T2, T3, T4, T5
Student Dashboard	FR7, FR8	DashboardController.Student, dashboard view models	T8
Lecturer Dashboard	FR7, FR9	DashboardController.Lecturer	T9
Administrator Dashboard	FR7, FR10	DashboardController.Administrator	T10
Course Management	FR11, FR12, FR13	CoursesController, Course, course views	T12, T13, T14, T15
Enrollment System	FR14, FR15	EnrollmentsController, Enrollment	T16, T17
Assignment and Grading	FR16, FR17, FR18, FR19	AssignmentsController, SubmissionsController	T18, T19, T20, T21
Reporting and Analytics	FR20, FR21	ReportsController, CsvWriter, PdfReport	T22, T23
Communication Module	FR22, FR23, FR24, FR25	MessagesController, MeetingsController	T24, T25
Security and Data Protection	NFR3, FR27, FR29, FR30	Program.cs, SecurityHeadersMiddleware, AuditLogger	T26, T27, T28
Validation and Exception Handling	FR28, NFR6	Data annotations, ModelState, Error.cshtml	T29, T30
Installation Guide and User Manual	NFR4, NFR9	Installation Guide and User Manual section	Manual review
Concise Logical Solution	NFR4, NFR5	Chapter 11 of this report	Manual review
Architecture Diagrams	NFR4	Shared diagram folder and Figures 1 to 6	Manual review

Marking Item	Supporting Requirements	Source-Code Evidence	Test Cases
Detailed Description of Classes	NFR4	Tables 4, 5, 6 in Chapter 10	Manual review
Own Reflection	n/a	Chapter 19	Manual review
Programming Style	NFR4	Folder structure, naming, DI	Manual review
Interface Design and Usability	NFR1, NFR7	Layout, partials, CSS	T11, T31

Table 8: Requirement Traceability Summary

6. System Architecture and Design

6.1 High-Level Architecture

The high-level architecture describes how the browser, the ASP.NET Core MVC application, the infrastructure services, and the database interact. UniManage follows a standard layered architecture for ASP.NET Core MVC applications. A request enters through Kestrel and passes through HTTPS redirection, static file serving, security headers, routing, rate limiting, authentication, and authorisation. After this middleware pipeline, the request reaches the controller, which uses EF Core through ApplicationDbContext to query or update the MySQL database, and which composes a view model that is rendered through Razor.

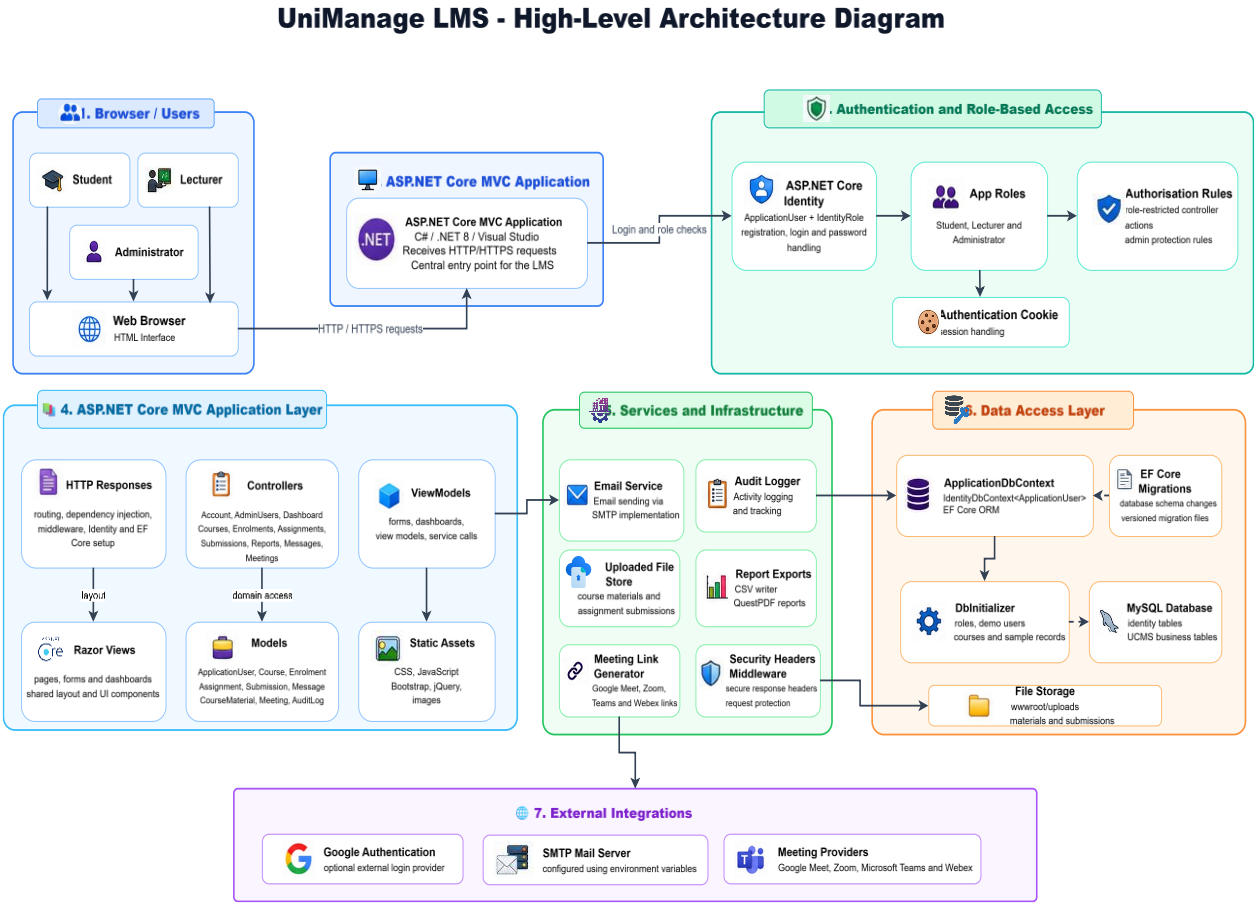


Figure 1: High-Level Architecture Diagram

Editable source available in shared diagram folder:
https://drive.google.com/drive/folders/1PvvXtEf_6VFk632ysTXCwjlybSpcOaef?usp=sharing

This diagram shows the high-level system architecture of UniManage. It traces a request from the browser through Kestrel, the HTTPS redirection middleware, the static file middleware, the security headers middleware, the routing middleware, the rate limiter, authentication, and authorisation, into the controllers under Controllers/, then through Entity Framework Core in Data/ApplicationDbContext.cs into the MySQL database accessed through the Pomelo provider, and back through Razor views to the browser. It also shows the supporting infrastructure services in Infrastructure/ and the static assets served from wwwroot/. The editable diagram source is available in the shared diagram folder.

The architecture consists of the following components:

Browser. The user agent that renders Razor-generated HTML, executes the small amount of JavaScript in wwwroot/js/site.js, and submits forms using anti-forgery tokens.

Kestrel and middleware. The ASP.NET Core hosting layer accepts the request and passes it through the middleware pipeline configured in Program.cs.

Identity. Authenticates the user using a cookie. Ten failed sign-in attempts within a minute are throttled by the rate limiter under the auth policy.

Controllers. The thirteen controllers under Controllers/ handle request flow, validation, and business rules.

EF Core. Translates LINQ queries into MySQL SQL through ApplicationDbContext.

Infrastructure services. A small set of services in Infrastructure/ provide cross-cutting capability such as audit logging, email, file storage, CSV writing, PDF rendering, calendar links, and security headers.

MySQL database. Stores all application data through Pomelo EF Core.

Static assets. Bootstrap, custom CSS, JavaScript, and images served from wwwroot/.

The diagram description explains how each component depends on the next. The author has used this layered architecture to keep request handling, business rules, persistence, and presentation in separate folders, which makes the project easier to maintain and easier to mark.

6.2 Use Case Diagram

Figure 2: Use Case Diagram

Editable draw.io source:

https://drive.google.com/file/d/1H9c4yXOcljag8dTxCizQyQIkxixHglvC/view?usp=sharing

This diagram shows the use case model for UniManage. It identifies the three actors (Student, Lecturer, Administrator) and the use cases listed in Table 7, including Register Account, Log In, Reset Password, View Dashboard, Browse Courses, Enrol on Course, Submit Assignment, View Grades and Feedback, Send Message, Manage Course, Manage Assignment, Grade Submission, Upload Course Material, Schedule Meeting, View Reports, Review Audit Log, Update Profile, and Change Password. The editable draw.io source is linked above.

The use case diagram identifies three actors and the use cases listed in Table 7. The Student actor is associated with the Register, Log In, View Student Dashboard, Browse Courses, Enrol, Submit Assignment, View Grades, Send Message, View Meetings, Update Profile, and Change Password use cases. The Lecturer actor is associated with the Log In, View Lecturer Dashboard, Manage Assignment, Grade Submission, Upload Course Material, Schedule Meeting, Send Message, Update Profile, and Change Password use cases. The Administrator actor is associated with the Log In, View Administrator Dashboard, Manage Course, View Report, Review Audit Log, Update Profile, and Change Password use cases. Common use cases such as Register, Log In, Update Profile, and Change Password are extended by all relevant actors.

6.3 Entity Relationship Diagram

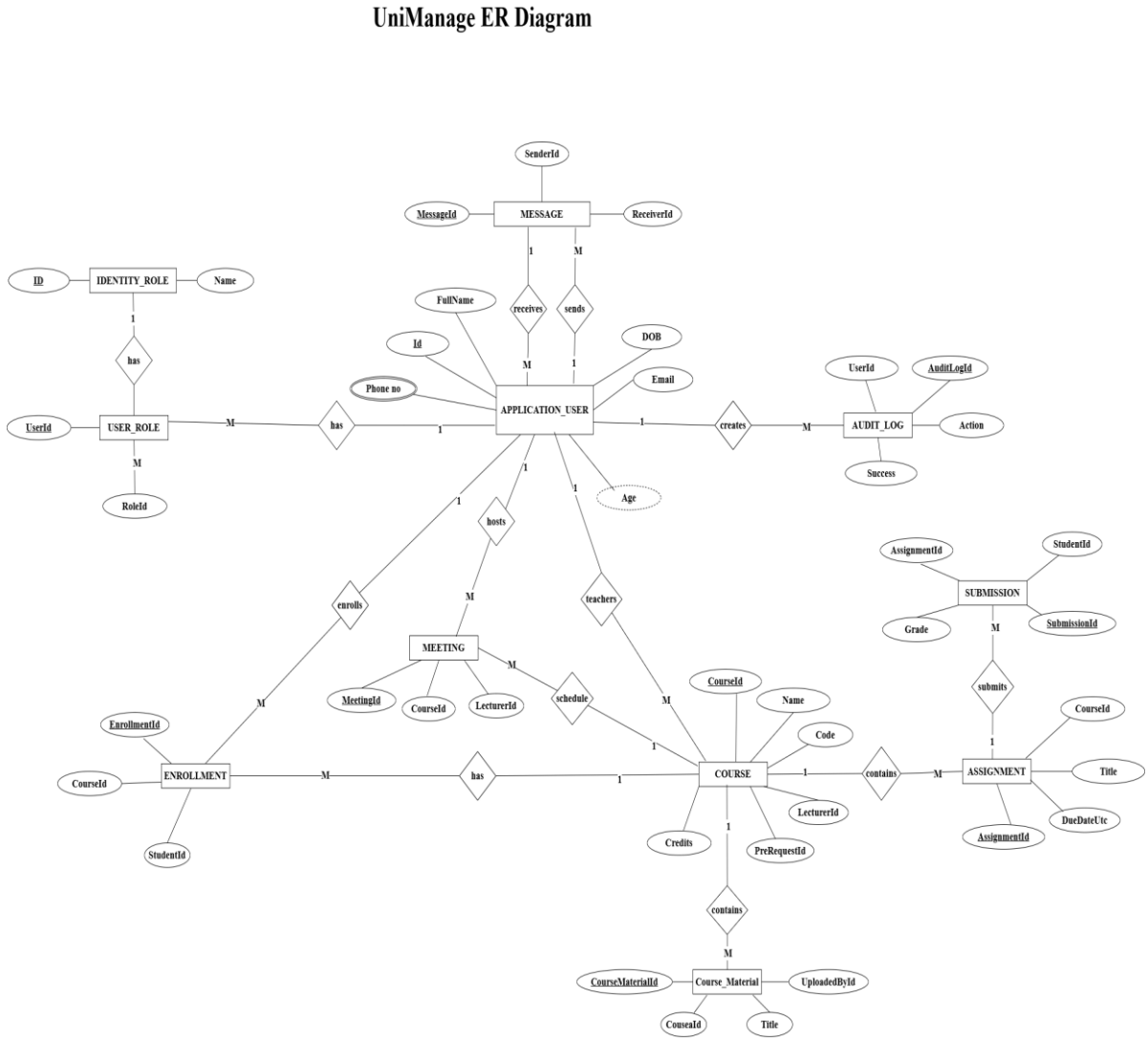


Figure 3: Entity Relationship Diagram

Editable draw.io source: https://drive.google.com/file/d/15IA6BQnpswKNplfang6vebdsNYfXlYU8/view?usp=sharing

This diagram shows the logical data model used by UniManage. It places ApplicationUser at the centre and connects it through one-to-many relationships to Course (lecturer side), Enrollment (student side), Submission (student and grader sides), CourseMaterial (uploader side), Message (sender and receiver sides), Meeting (lecturer side), and AuditLog (optional user side). The diagram also shows the optional self-reference from Course to Course for prerequisites, the unique-pair index on Enrollment(StudentId, CourseId), and the relationship from Assignment to Course and from Submission to Assignment. The editable draw.io source is linked above.

The ER diagram shows the relationships between the application's main entities. ApplicationUser is the central entity. A Course is taught by exactly one ApplicationUser (the lecturer) and may have a Course as a prerequisite. An Enrollment links one ApplicationUser (the student) to one Course and is unique on the combination of student and course. An Assignment belongs to one Course. A Submission belongs to one Assignment, one ApplicationUser (the student), and optionally one ApplicationUser (the grader). A CourseMaterial belongs to one Course and is uploaded by one ApplicationUser. A Message is sent from one ApplicationUser to another. A Meeting belongs to one Course and is owned by one ApplicationUser (the lecturer). An AuditLog records the user identifier where available.

The relationships are configured in Data/ApplicationDbContext.cs through the EF Core fluent API, including foreign keys, delete behaviour, and unique constraints such as the unique course code and the unique student-course enrolment.

6.4 UML Class Diagram

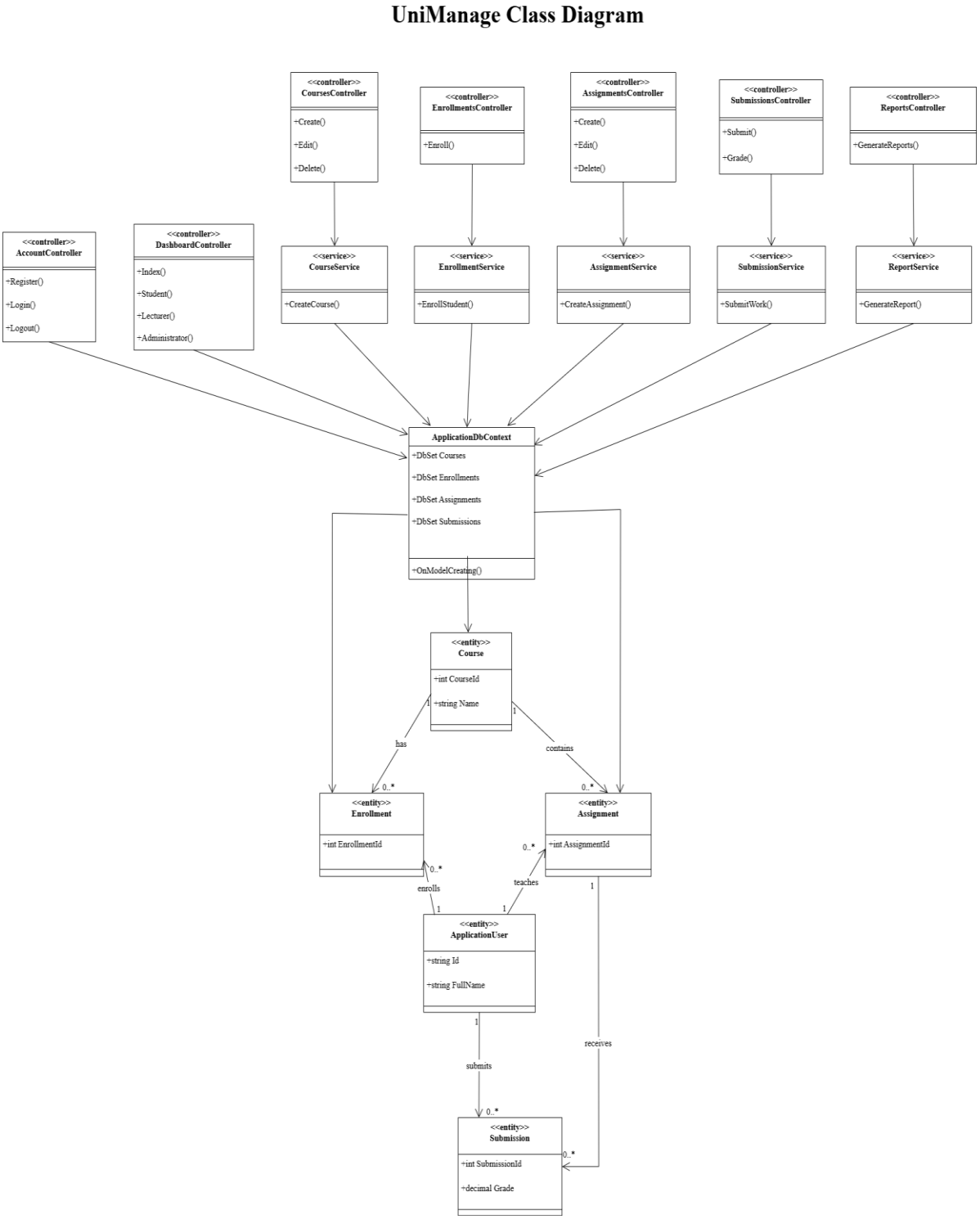


Figure 4: UML Class Diagram

Editable draw.io source:

https://drive.google.com/file/d/189YrFogTsnGSPoH0Se6xBBerum1HaiOl/view?usp=sharing

This diagram shows the main classes used by UniManage and their dependencies. It includes the thirteen controllers (AccountController, AdminUsersController, DashboardController, CoursesController, EnrollmentsController, AssignmentsController, SubmissionsController, ReportsController, MessagesController, MeetingsController, ProfileController, AuditLogsController, HomeController), the domain models, the view models, the ApplicationDbContext, and the infrastructure services such as IAuditLogger, IEmailService, UploadedFileStore, CsvWriter, PdfReport, and SecurityHeadersMiddleware. The editable draw.io source is linked above.

The class diagram summarises the controllers, models, view models, services, and the ApplicationDbContext. The diagram shows that controllers depend on ApplicationDbContext and on infrastructure interfaces such as IAuditLogger and IEmailService. Models are plain C# classes with data annotation attributes. View models are also plain C# classes that are used only between controllers and views, and are not persisted by EF Core. The diagram does not include role-specific controller classes because the project uses a single DashboardController with separate actions for each role.

6.5 Application Flowchart

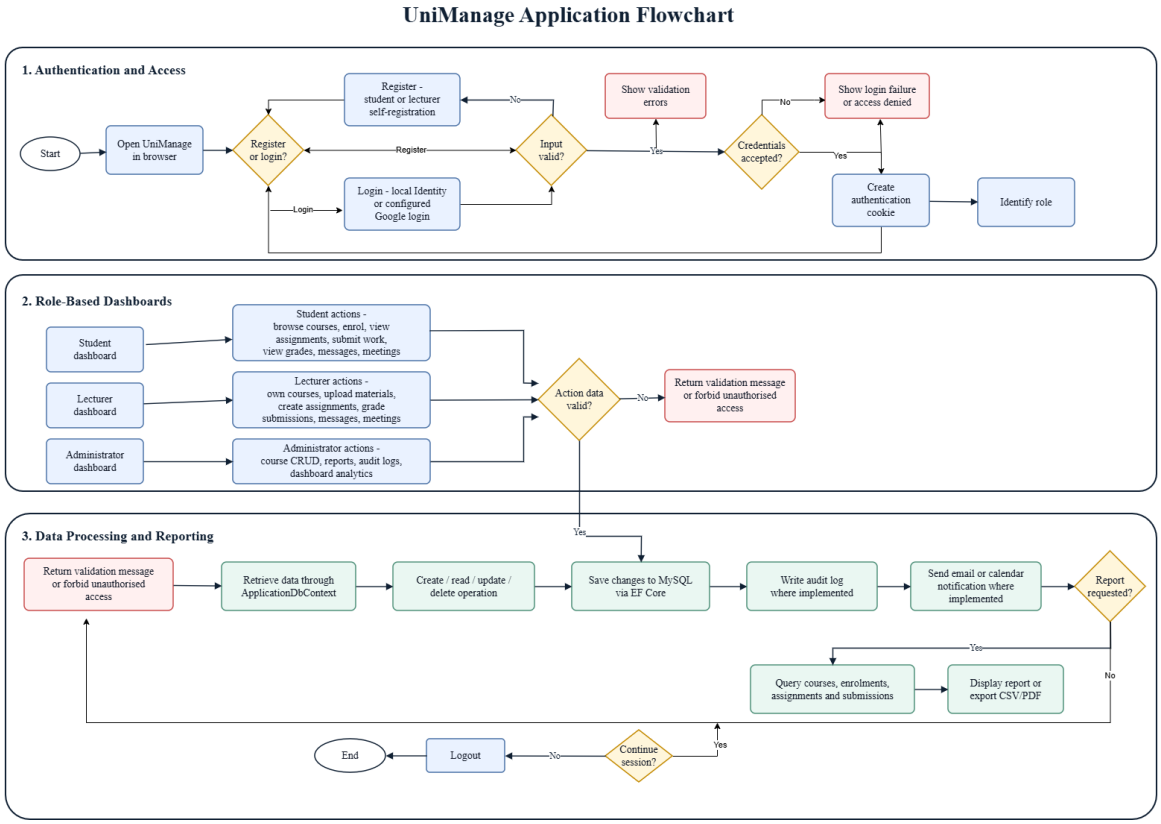


Figure 5: Application Flowchart

Editable source available in shared diagram folder: https://drive.google.com/drive/folders/1PvvXtEf_6VFk632ysTXCwjlybSpcOaef?usp=sharing

This diagram shows the application flow from the home page through registration or login, role-aware redirection to the appropriate dashboard, the controller validation step, the EF Core data access step, the response back to the user, and the logout path. It also shows the friendly error page that is reached when an unhandled exception is raised in production. The editable diagram source is available in the shared diagram folder.

The flowchart starts at the home page, where an unauthenticated user is offered Register or Log In. After successful login, the user is redirected to the role-appropriate dashboard. From the dashboard, the user navigates to a feature, performs an action, and the controller validates the input, calls EF Core, and responds with either a success view or a validation message. The flowchart also covers the logout action and the friendly error page.

6.6 Authentication and Role-Based Access Flow

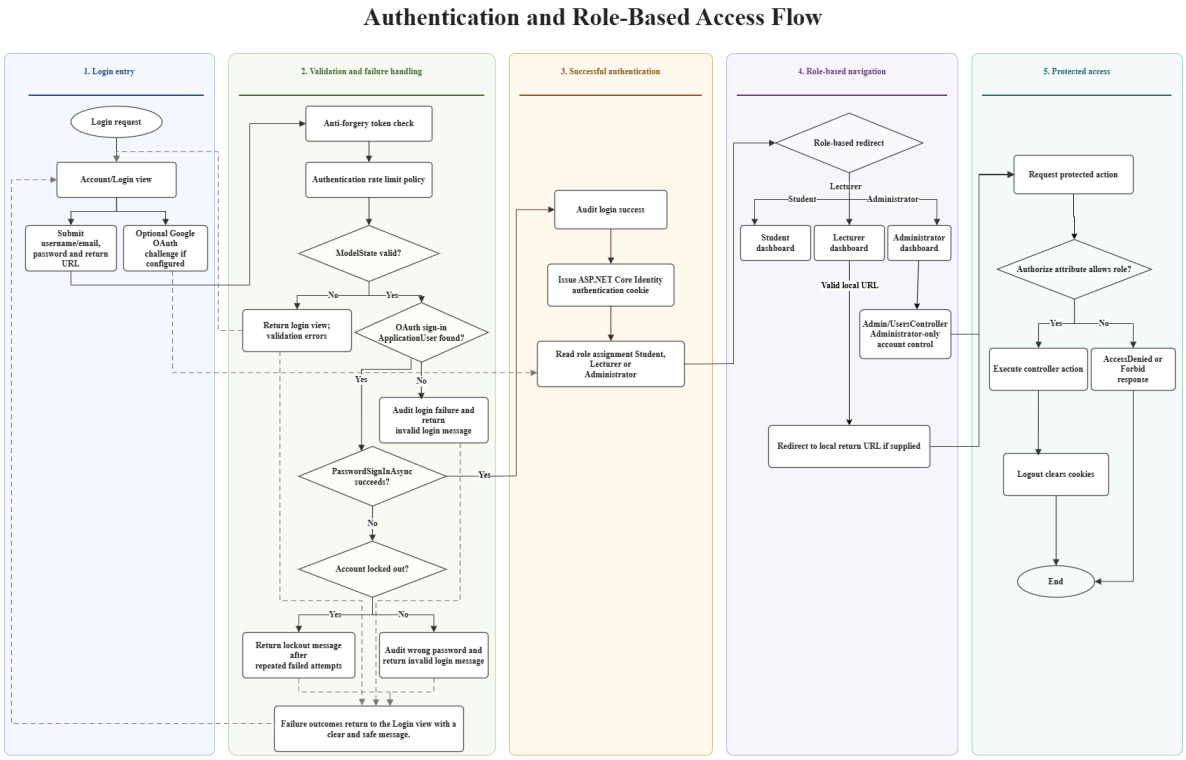


Figure 6: Authentication and Role-Based Access Flow

Editable source available in shared diagram folder:
https://drive.google.com/drive/folders/1PvvXtEf_6VFk632ysTXCwjlybSpcOaef?usp=sharing

This diagram shows the authentication and role-based access flow. It traces the path from the login form to AccountController.Login, through the Identity sign-in API, the lockout policy and the auth rate-limiting policy, to the issued authentication cookie, and on to Dashboard/Index, which inspects the role claim and forwards the request to the Student, Lecturer, or Administrator action. It also shows the access denied page that is returned when an unauthorised role attempts to open a restricted action. The editable diagram source is available in the shared diagram folder.

The authentication flow shows the path from the login form to the authentication cookie. The user posts the form to AccountController.Login, which validates the model state, calls Identity to verify the credentials, and either signs the user in or returns a validation error. After sign-in,

the controller redirects the user to Dashboard/Index, which inspects the role claim and forwards the request to Student, Lecturer, or Administrator. If the user opens an action that requires a different role, the [Authorize] attribute returns a 403 response and the access denied page is shown.

6.7 Design Decisions and Rationale

The author has taken several design decisions during the project. The most important decisions are summarised below.

MVC over Razor Pages. The brief refers to ASP.NET MVC. ASP.NET Core MVC is the modern equivalent and is therefore the natural choice. The MVC structure also matches the marking criteria for separation of concerns.

EF Core with Pomelo MySQL over SQL Server / LocalDB. The local environment includes MySQL 8 and the Pomelo provider is mature. The EF Core layer abstracts the underlying engine, so the design remains portable.

Single DashboardController over three role-specific controllers. A single controller with three role-restricted actions reduces duplication, makes routing simpler, and keeps role-specific logic in one place. The pattern matches the way the brief talks about "dashboards" rather than "dashboard controllers".

Submission stores grade fields directly. The brief mentions assignments and grading. A separate Grade entity would add tables and complexity without changing the workflow. The Submission entity therefore stores the grade fields directly, which keeps the data model small and keeps the grading view simple.

Custom security headers middleware. Although ASP.NET Core ships with several security features, security headers such as Content Security Policy, X-Content-Type-Options, and Referrer Policy are added through a small middleware in Infrastructure/SecurityHeadersMiddleware.cs. This makes the controls visible and reviewable.

QuestPDF over a separate reporting tool. QuestPDF is open source under the Community licence, integrates cleanly with C#, and produces professional PDF output. The licence type is set to Community in Program.cs.

7. Database Design

7.1 Database Technology

UniManage uses Entity Framework Core 8 with the Pomelo MySQL provider against a MySQL 8 server. The connection string is read from the DefaultConnection entry of appsettings.json (or appsettings.Development.json during development). The configuration in Program.cs registers the ApplicationDbContext and enables retry on failure.

```
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseMySql(connectionString, new MySqlServerVersion(new Version(8,
0, 36)), mySql =>
        mySql.EnableRetryOnFailure(maxRetryCount: 5, maxRetryDelay:
TimeSpan.FromSeconds(30), errorNumbersToAdd: null)));
```

EF Core retry on failure is important because MySQL connections can fail transiently when the server is starting or when a network blip occurs. The retry policy hides these transient errors from the user and reduces the chance of false-negative test results.

The brief refers to SQL Server / LocalDB as a possible technology. UniManage uses MySQL through Pomelo, which is recorded honestly in the documentation. If the marker requires a SQL Server / LocalDB build, the approach would be to replace UseMySql with UseSqlServer and update the connection string. The migrations would need to be regenerated to reflect SQL Server data types. This option is recorded as a future conversion route because no SQL Server scripts are present in the source.

7.2 ApplicationDbContext

ApplicationDbContext inherits from IdentityDbContext<ApplicationUser>. This means the context exposes the standard Identity tables (Users, Roles, UserRoles, UserClaims, UserLogins, UserTokens, RoleClaims) and, in addition, the academic tables described in the next section. The constructor accepts DbContextOptions<ApplicationDbContext> so that the options can be supplied through dependency injection.

The fluent API in OnModelCreating is used to configure relationships, delete behaviour, and unique indexes. The most important configurations include the unique course code, the unique student-course enrolment, the lecturer-course foreign key with restrict-on-delete behaviour, the prerequisite course self-reference, the assignment-course relationship, the submission-assignment-student relationship, the message sender and receiver self-references, the meeting-course-lecturer relationships, and the audit log user reference.

7.3 Main Entities

Table 9 lists the main academic entities. The Identity entities (AspNetUsers, AspNetRoles, etc.) are not duplicated in the table because they are managed by the framework.

Entity	Primary Key	Main Foreign Keys	Purpose
ApplicationUser	Id (string)	Identity relationships	Stores user profile data and Identity user fields.
Course	CourseId	LecturerId (ApplicationUser), PrerequisiteId (Course, optional)	Stores course code, name, description, credits, enrolment limit, lecturer assignment, optional prerequisite.
Enrollment	EnrollmentId	StudentId (ApplicationUser), CourseId (Course)	Links a student to a course; unique on the pair.
Assignment	AssignmentId	CourseId (Course)	Records coursework tasks set by a lecturer for a course.
Submission	SubmissionId	AssignmentId, StudentId, GradedById (optional)	Records student submissions, grade, feedback, status, file metadata.
CourseMaterial	CourseMaterialId	CourseId, UploadedById	Records uploaded teaching material metadata.
Message	MessageId	SenderId, ReceiverId	Records direct messages between users.
Meeting	MeetingId	CourseId, LecturerId	Records scheduled course meetings, including a meeting URL.
AuditLog	AuditLogId	UserId (optional)	Records security-sensitive and academic events.

Table 9: Core Database Entities

7.4 Entity Relationships

- The relationships between the entities are summarised below.
- ApplicationUser to Course (lecturer side). One-to-many. A lecturer may teach many courses; a course has one lecturer.
- Course to Course (prerequisite). Optional self-reference. A course may require another course as a prerequisite, or none at all.
- ApplicationUser to Enrollment (student side). One-to-many. A student may have many enrolments.
- Course to Enrollment. One-to-many. A course may have many enrolments.
- Enrollment uniqueness. Unique index on the pair (StudentId, CourseId) prevents duplicate enrolments.
- Course to Assignment. One-to-many. A course may have many assignments.
- Assignment to Submission. One-to-many. An assignment may have many submissions.
- ApplicationUser to Submission (student side). One-to-many.
- ApplicationUser to Submission (grader side). Optional one-to-many.
- Course to CourseMaterial. One-to-many.
- ApplicationUser to CourseMaterial (uploader side). One-to-many.
- ApplicationUser to Message (sender and receiver sides). Two one-to-many self-references.
- Course to Meeting. One-to-many.
- ApplicationUser to Meeting (lecturer side). One-to-many.
- ApplicationUser to AuditLog (optional). One-to-many; the audit log may record events even when the user is anonymous, in which case the UserId is null.

7.5 Keys, Constraints and Validation Rules

The database enforces the following constraints, in addition to the field-level data annotation rules:

Unique course code on Course.Code.

- Unique enrolment per student and course on Enrollment(StudentId, CourseId).
- Maximum lengths on string fields, set through data annotations such as [StringLength].
- Required fields enforced through [Required] annotations.
- Range constraints on numeric fields such as MaxPoints on Assignment and the grade on Submission.
- Optional foreign keys for PrerequisiteId and GradedById.
- Default values set in the model constructor or through migrations, for example EnrolledAtUtc = DateTime.UtcNow.

The combination of database constraints and application-level validation provides defence in depth. A bad request that bypasses the controller would still be caught by the database; a bad input that satisfies the database would still be caught by ModelState before reaching the database.

7.6 Migrations and Seed Data

EF Core migrations describe the schema changes required to bring the database up to date. UniManage has four migrations:

20260408041036_InitialUniManage.cs:- creates the Identity tables and the initial academic tables (Courses, Enrollments, Assignments, Submissions, CourseMaterials, Messages).

20260501122117_AddAuditLogAndSecurity.cs:- adds the AuditLogs table and supporting columns for security tracking.

20260501125135_AddMeetings.cs:- adds the Meetings table and the meeting URL column.

20260506142007_AddApplicationUserDateOfBirth.cs:- adds the optional date-of-birth field to ApplicationUser.

DbInitializer.SeedAsync is called from Program.cs at startup. It applies pending migrations, creates the three roles (Administrator, Lecturer, Student), and creates the confirmed

demonstration users admin@gmail.com, lecturer@gmail.com, and lms@gmail.com. The seeder is idempotent: it checks for existing users, roles, courses, enrolments, assignments, submissions, and meetings before creating new records. This prevents duplicate seed data when the application is restarted.

The confirmed seeded modules are Advanced Software Engineering, Application Development, Artificial Intelligence, Project Analysis and Practice, and Individual Project. The seeder also creates sample enrolments, assignments, selected submissions or grades, and course meetings for demonstration. This data helps the marker test the application quickly during installation, viva, and browser review. The seeded credentials are listed in the user manual and are intended for coursework marking only, not for production use.

7.7 Database and Migration Evidence

This section provides database and migration evidence for the submitted UniManage implementation.

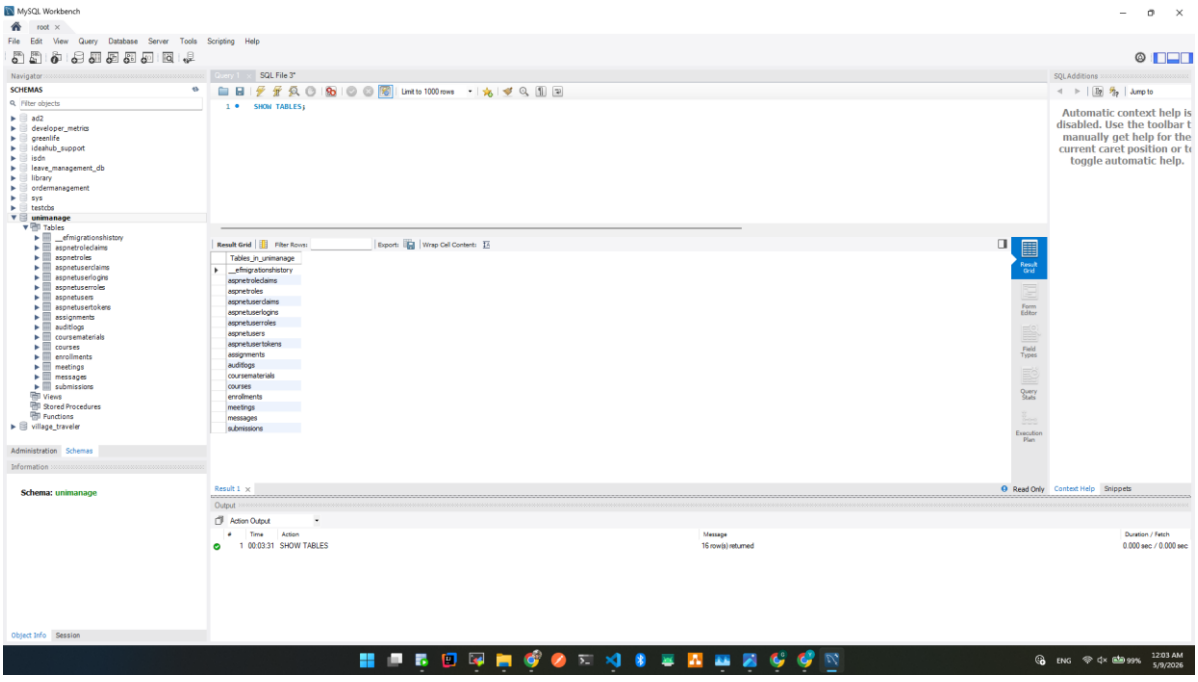


Figure 7: Database and Seed Data Evidence

This screenshot shows a MySQL client connected to the seeded UniManage database, with the Courses, Enrollments, or Submissions table populated after a few actions in the application. Personal data is excluded or redacted in the submitted report.

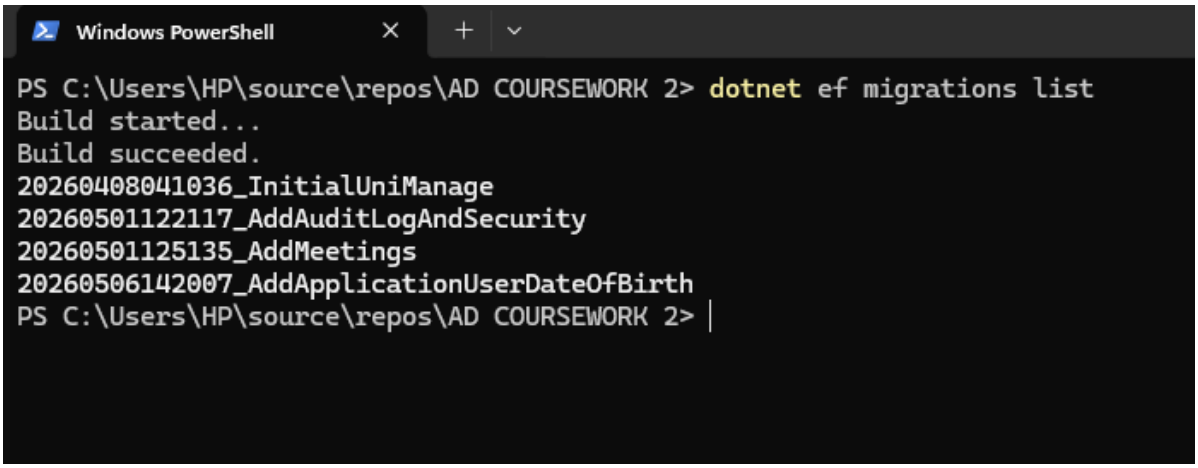


Figure 7b: EF Core Migration Evidence

This screenshot shows the console output of dotnet ef migrations list from the project root, listing the four EF Core migrations.

The following database items are not submitted as part of this final report: a raw SQL backup script, a SQL Server / LocalDB equivalent script, and a database backup file (.bak or .sql dump).

8. Implementation

This chapter explains the main implementation work feature by feature. The focus is on the source files involved, request flow, validation and business rules, view models, and evidence still needed for the final submission.

8.1 Project Structure

The project follows the standard ASP.NET Core MVC layout with the following top-level folders:

Controllers/ (thirteen files): AccountController.cs, AdminUsersController.cs, AssignmentsController.cs, AuditLogsController.cs, CoursesController.cs, DashboardController.cs, EnrollmentsController.cs, HomeController.cs, MeetingsController.cs, MessagesController.cs, ProfileController.cs, ReportsController.cs, SubmissionsController.cs.

Models/ (twelve files): AppRoles.cs, ApplicationUser.cs, Assignment.cs, AuditLog.cs, Course.cs, CourseMaterial.cs, Enrollment.cs, ErrorViewModel.cs, Meeting.cs, Message.cs, Submission.cs, SubmissionStatus.cs.

ViewModels/ (eighteen files):- AdminUserManagementViewModels.cs, AssignmentInputViewModel.cs, CourseBrowseRowViewModel.cs, CourseInputViewModel.cs, DashboardViewModels.cs, DateOfBirthRules.cs, ForgotPasswordViewModel.cs, GradeSubmissionViewModel.cs, LoginViewModel.cs, MaterialUploadViewModel.cs, MeetingInputViewModel.cs, MessageComposeViewModel.cs, MessageInboxRowViewModel.cs, MessageThreadViewModel.cs, ProfileViewModel.cs, RegisterViewModel.cs, ResetPasswordViewModel.cs, SubmissionSubmitViewModel.cs.

Data/:- ApplicationDbContext.cs, ApplicationDbContextFactory.cs, DbInitializer.cs, Data/Migrations/ (four migrations and the model snapshot).

Infrastructure/ (thirteen files):- AuditLogger.cs, CalendarLink.cs, CsvWriter.cs, EmailSettings.cs, EmailTemplates.cs, GradeBandHelper.cs, IAuditLogger.cs, IEmailService.cs, MeetLinkGenerator.cs, PdfReport.cs, SecurityHeadersMiddleware.cs, SmtpEmailService.cs, UploadedFileStore.cs.

Views/ (fifty Razor views):- Account/, AdminUsers/, AuditLogs/, Assignments/, Courses/, Dashboard/, Enrollments/, Home/, Meetings/, Messages/, Profile/, Reports/, Submissions/, Shared/, plus _ViewStart.cshtml and _ViewImports.cshtml.

wwwroot/: static assets including css/site.css, js/site.js, lib/, and assets/.

Configuration:- appsettings.json, appsettings.Development.json, Properties/launchSettings.json.

This layout follows the conventional ASP.NET Core MVC structure, so the marker can move from a controller to the related model, view model, view, and database evidence without learning a custom folder style.

8.2 Program.cs and Startup Configuration

Program.cs is the composition root. It is responsible for service registration and middleware configuration. The file does the following work in order:

Sets the QuestPDF licence to Community to comply with the QuestPDF licensing terms. Reads the DefaultConnection connection string from configuration. If the connection string is missing, the application throws an InvalidOperationException at startup so that the misconfiguration is visible immediately.

Registers ApplicationDbContext with the Pomelo MySQL provider and enables retry on failure.

Registers ASP.NET Core Identity with ApplicationUser and IdentityRole. The Identity options enforce strong password rules, lockout, and unique email.

Registers the cookie authentication scheme as the default and conditionally registers Google authentication if a client id and client secret are configured.

Configures the application cookie with login, logout, and access denied paths, sliding expiration of eight hours, HTTP-only flag, SameSiteMode.Lax, and CookieSecurePolicy.SameAsRequest.

Binds the email settings, registers SmtpEmailService for IEmailService, registers IHttpContextAccessor, and registers AuditLogger for IAuditLogger.

Configures the rate limiter with auth and upload policies.

Configures anti-forgery cookies as HTTP-only with SameSiteMode.Strict.

Registers MVC controllers with views.

Builds the application.

Runs DbInitializer.SeedAsync inside a service scope.

Applies the production exception handler and HSTS in production, or the developer exception page in development.

Applies HTTPS redirection, static files, the security headers middleware, routing, the rate limiter, authentication, and authorisation, then maps the default route and runs the application.

Sensitive deployment settings are read through the ASP.NET Core configuration system rather than being written into the report. In the Docker workflow, docker-compose.yml maps server-side environment variables into ConnectionStrings_DefaultConnection, Authentication_Google_ClientId, Authentication_Google_ClientSecret, and the Email_... SMTP settings. The deployment.env.example file is a safe template only; real database passwords, SMTP passwords, Google OAuth secrets, and server credentials are not included in the public documentation.

Listing 8.1 shows the security-related part of Program.cs.

```
builder.Services.AddRateLimiter(options =>
{
    options.RejectionStatusCode = StatusCodes.Status429TooManyRequests;
    options.AddPolicy("auth", httpContext =>
        RateLimitPartition.GetFixedWindowLimiter(
            partitionKey: httpContext.Connection.RemoteIpAddress?.ToString()
?? "anon",
            factory: _ => new FixedWindowRateLimiterOptions
            {
                PermitLimit = 10,
                Window = TimeSpan.FromMinutes(1),
                QueueLimit = 0,
                AutoReplenishment = true
            }));
    options.AddPolicy("upload", ...);
});
builder.Services.AddAntiforgery(options =>
{
    options.Cookie.HttpOnly = true;
    options.Cookie.SameSite = SameSiteMode.Strict;
    options.Cookie.SecurePolicy = CookieSecurePolicy.SameAsRequest;
});
```

8.3 User Registration and Login

Registration and login are implemented in Controllers/AccountController.cs. The Register GET action returns the registration form. The Register POST action validates RegisterViewModel, creates a new ApplicationUser, calls UserManager.CreateAsync, assigns the chosen role (Student or Lecturer), and signs the user in. Audit log entries are written for both successful and failed registrations.

The Login GET action returns the login form. The Login POST action validates LoginViewModel, accepts either username or email as the login identifier, calls SignInManager.PasswordSignInAsync with lockoutOnFailure: true, and either signs the user in or returns a validation error. Failed sign-in attempts are recorded in the audit log. After successful login, the user is redirected to Dashboard/Index, which routes the user to the role dashboard.

The Logout action calls SignInManager.SignOutAsync and redirects to the home page. The ForgotPassword and ResetPassword actions provide the standard Identity password recovery flow with email tokens.

Google OAuth login is included as an optional sign-in route. Program.cs registers the Google authentication handler only when Authentication:Google:ClientId and Authentication:Google:ClientSecret are configured. AccountController.LoginWithGoogle checks those values before starting the OAuth challenge, and GoogleCallback creates a Student account from the Google profile if the email is not already registered. This makes sign-in easier where OAuth is configured, but it depends on the correct Google Cloud authorised redirect URI for the local or hosted URL. If Docker or a reverse proxy is used, the redirect URI, HTTPS state, forwarded host, and protocol handling must be checked before hosted Google sign-in is treated as fully evidenced.

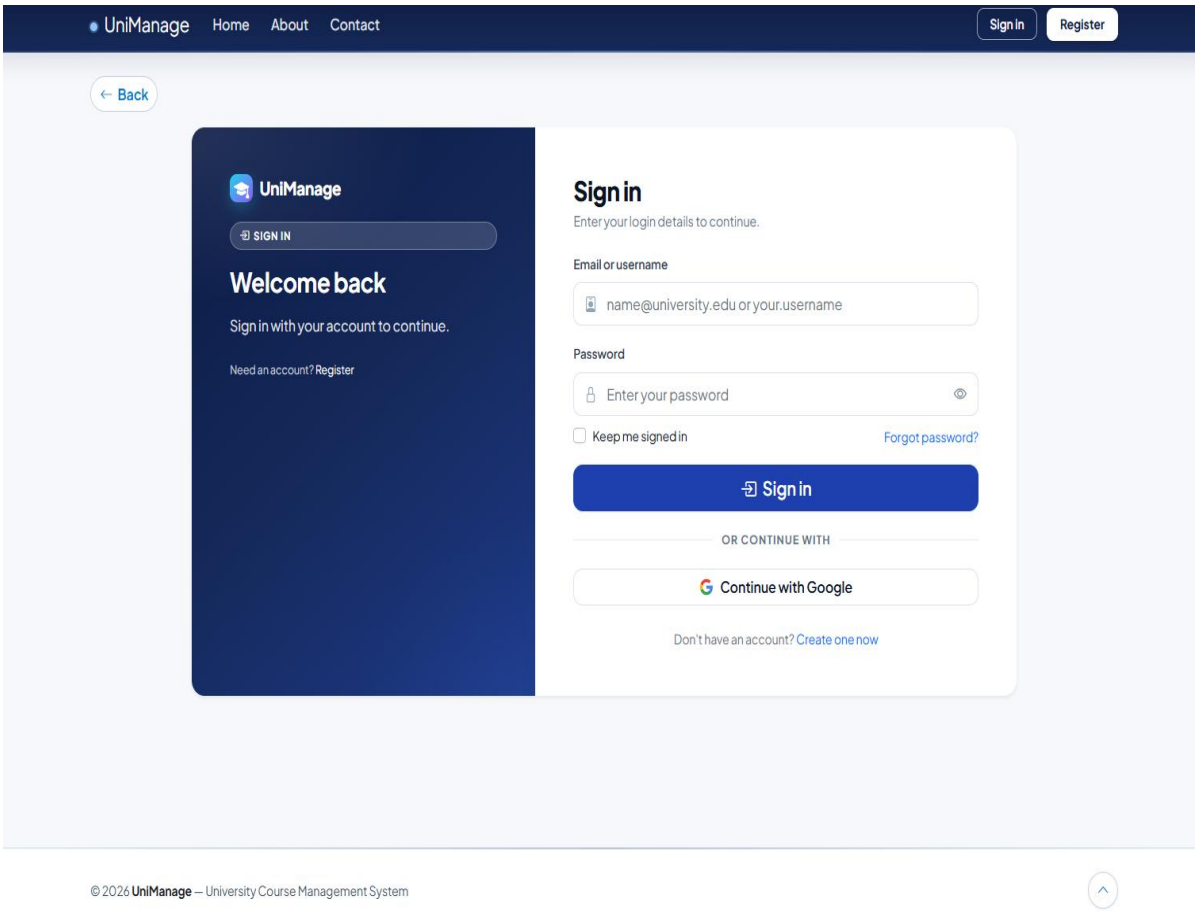


Figure 8: Login Page

This screenshot shows the login page after running the application.

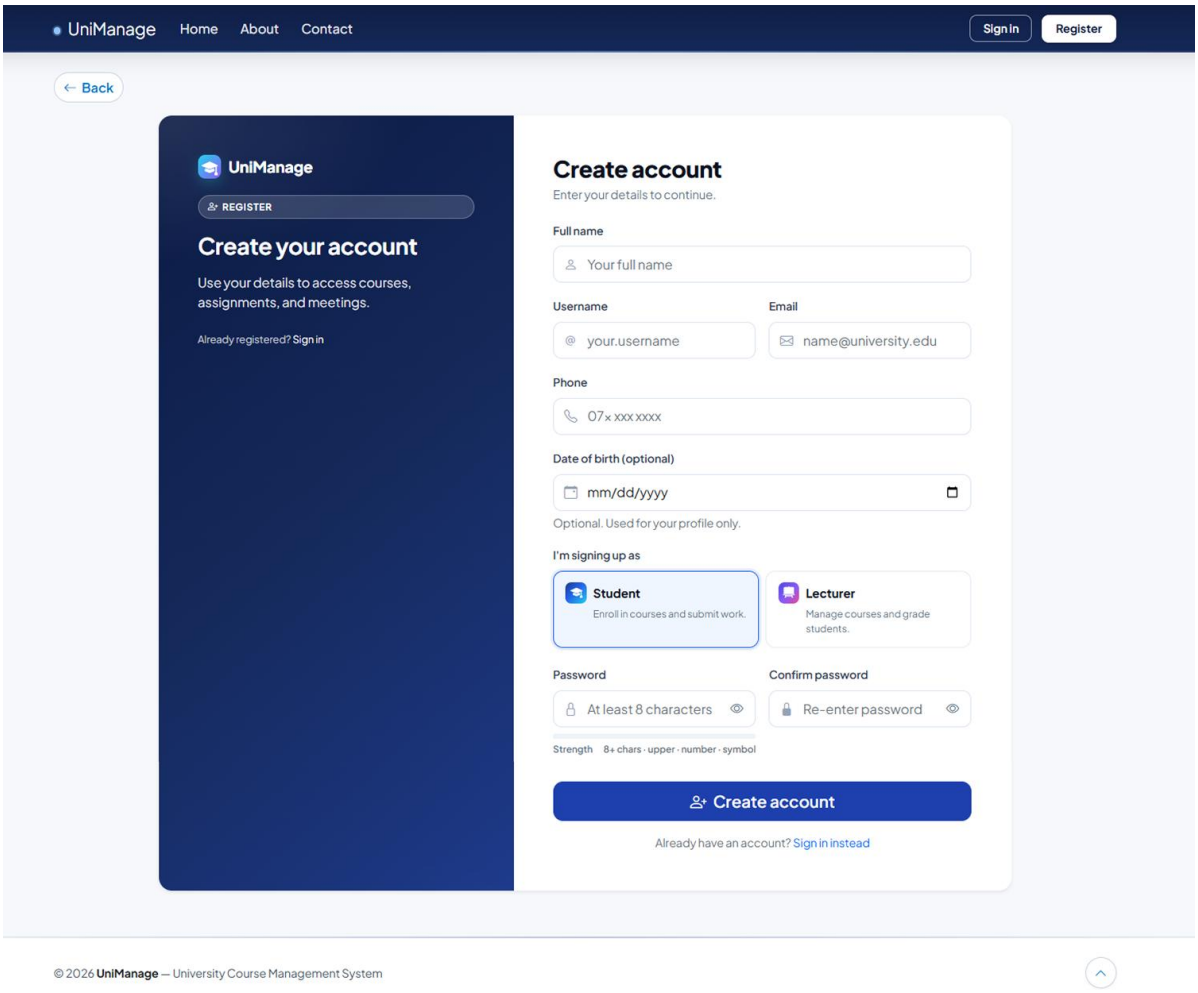


Figure 8b: User Registration Page

This screenshot shows the registration page with the role selector visible.

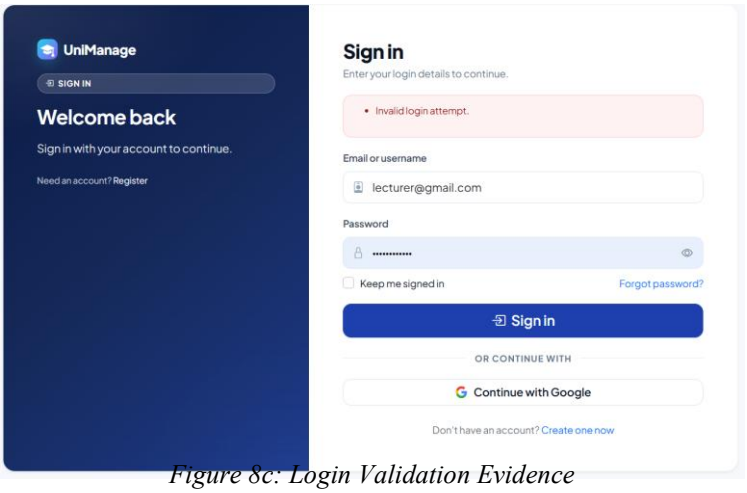


Figure 8c: Login Validation Evidence

This screenshot shows the login form after invalid credentials are submitted. It records the validation evidence for the failed login case.

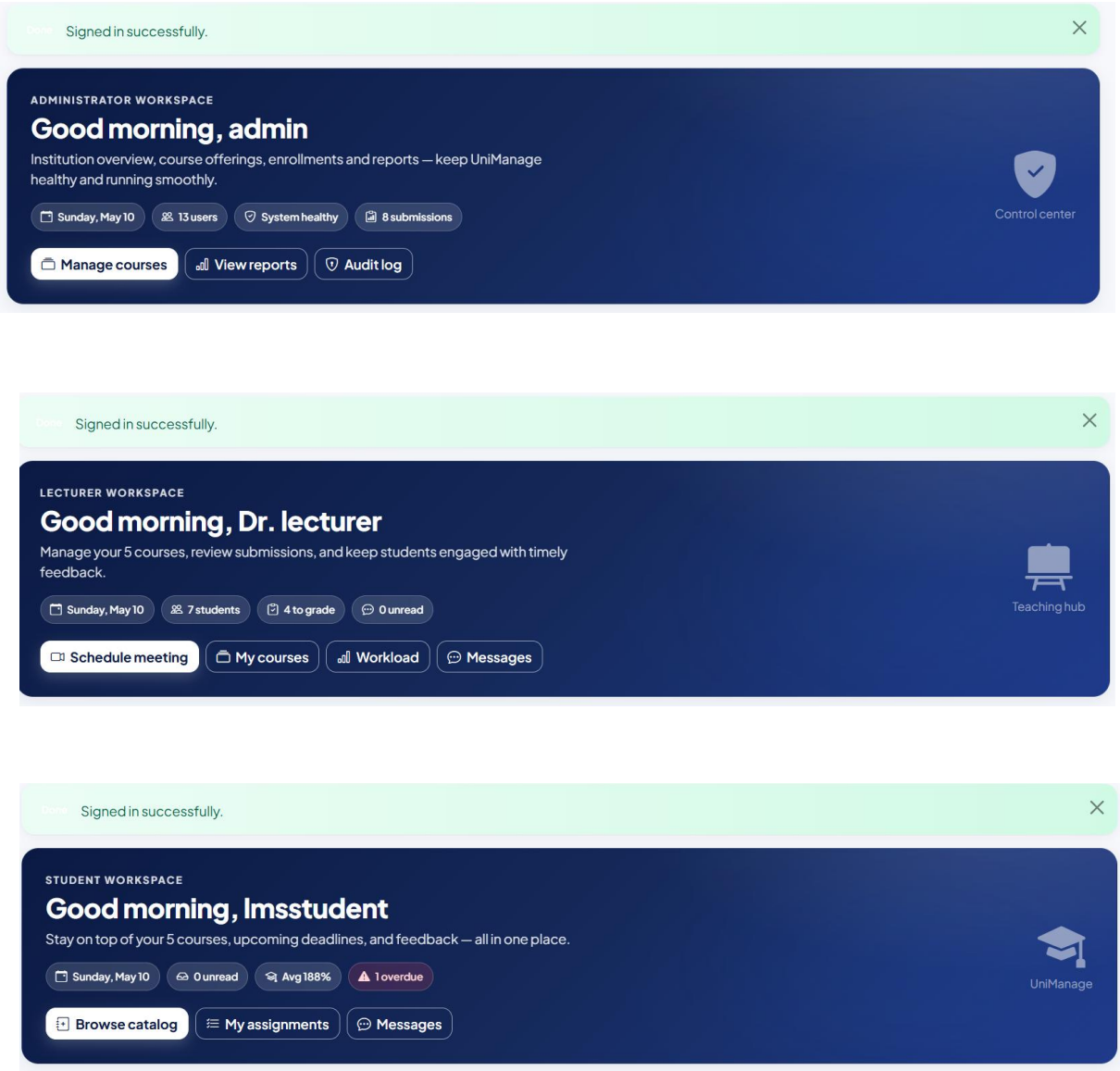


Figure 8d: Successful Login Redirect

This screenshot shows the browser URL after a successful Student login, with the user redirected to the role-appropriate dashboard.

8.4 Role-Based Access Control

Role-based access control is implemented through three layers. First, AppRoles defines the role name constants. Second, DbInitializer seeds the three roles and the three demonstration users. Third, each controller action uses [Authorize(Roles = AppRoles.X)] to restrict access. Where a Lecturer must only see their own courses or assignments, an additional ownership filter is applied inside the action.

The three roles have different permissions. Students access enrolled courses, assignments, submissions, grades, relevant meetings, and messages. Lecturers manage course-related assignments, submissions, grading, materials, meetings, and communication with enrolled students. Administrators manage courses, reports, audit views, dashboards, and user accounts where implemented. The shared navigation in _Layout.cshtml changes according to the signed-in role, which makes the interface clearer and reduces access-denied navigation paths.

The role names are held in AppRoles rather than repeated as plain text. This keeps role checks consistent across controllers and views, and it reduces the chance of a spelling mistake causing an incorrect access decision.

8.5 Student Dashboard

The Student dashboard is implemented by DashboardController.Student and Views/Dashboard/Student.cshtml. The action queries the Enrollments, Assignments, Submissions, Messages, and Meetings tables to build a StudentDashboardViewModel (defined inside ViewModels/DashboardViewModels.cs). The view model holds enrolled courses, upcoming deadlines, recent grades, recent messages, calendar events, and meetings. EF Core read queries use AsNoTracking where possible to reduce the change tracker overhead.

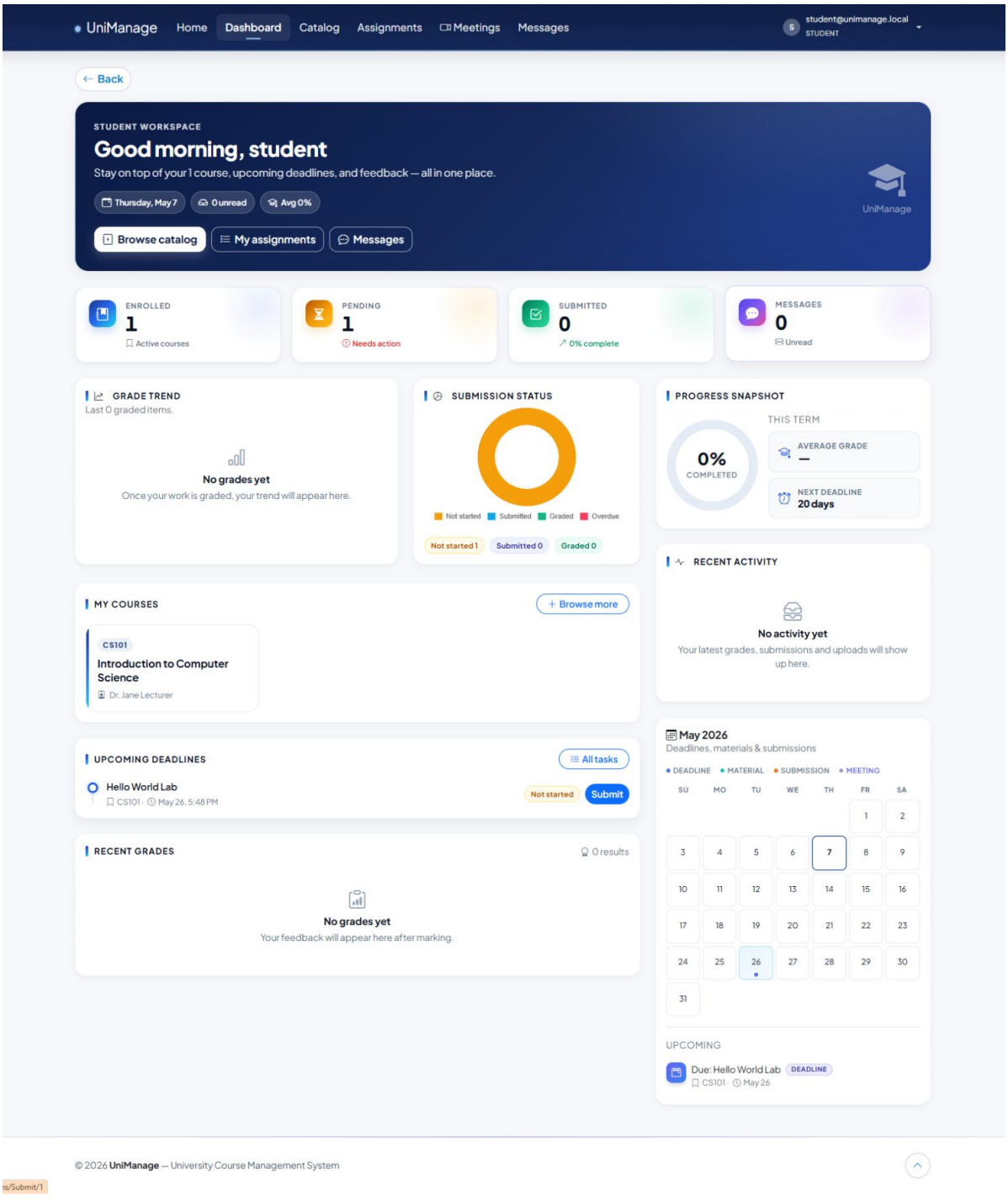


Figure 9: Student Dashboard

This screenshot shows the Student dashboard after login. It brings together the student's enrolled courses, assignments, messages, and meeting information.

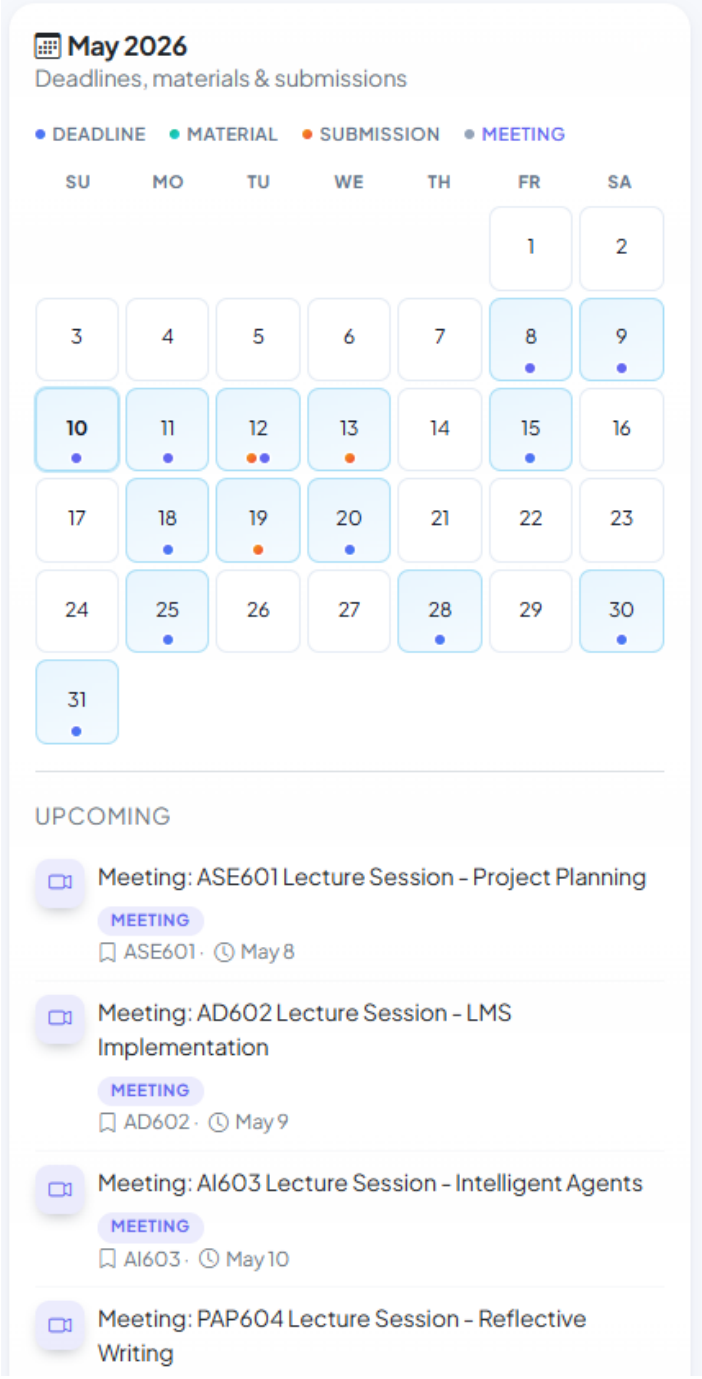


Figure 9b: Student Calendar and Deadlines Panel

This screenshot shows the student calendar and deadlines panel.

8.6 Lecturer Dashboard

The Lecturer dashboard is implemented by DashboardController.Lecturer and Views/Dashboard/Lecturer.cshtml. The action filters courses, assignments, and submissions by the signed-in lecturer's user identifier. The view model holds teaching courses, enrolment counts, assignment counts, pending submissions, recently graded submissions, recent messages, recent meetings, and recent activity.

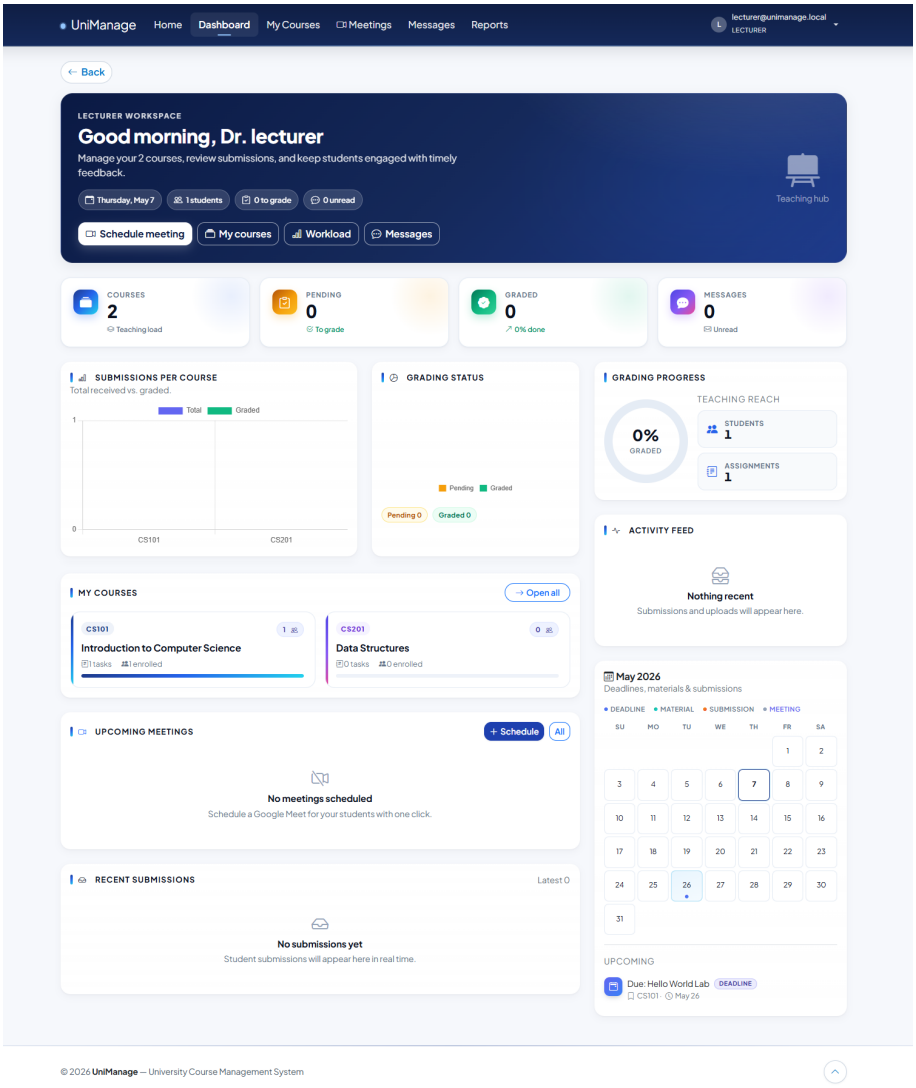
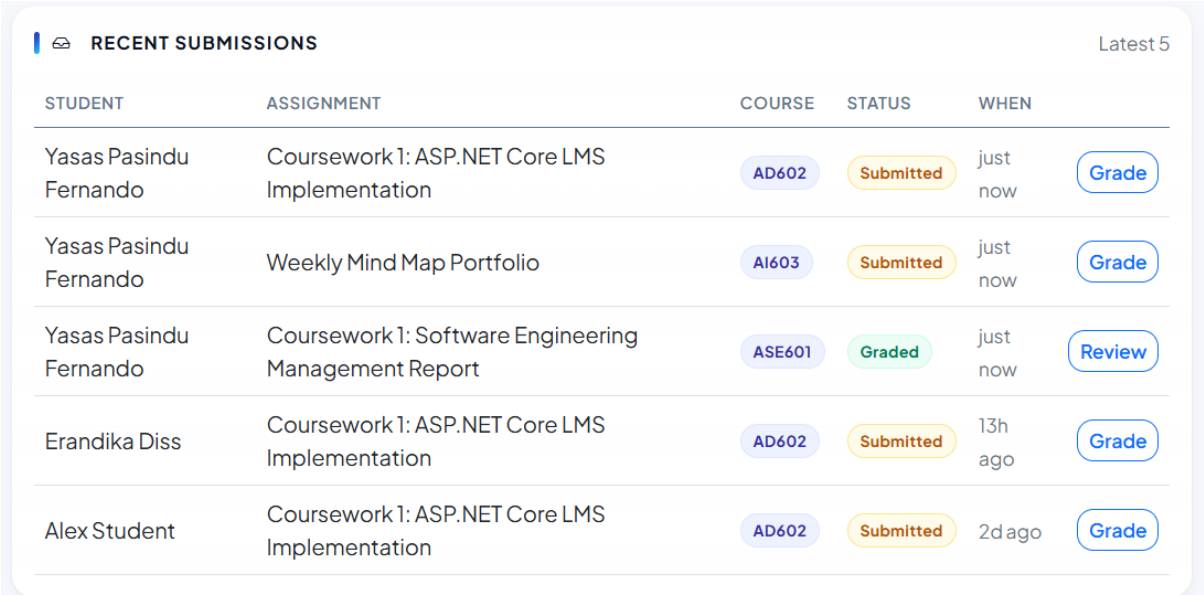


Figure 10: Lecturer Dashboard

This screenshot shows the lecturer dashboard.



RECENT SUBMISSIONS — Latest 5

STUDENT	ASSIGNMENT	COURSE	STATUS	WHEN	
Yasas Pasindu Fernando	Coursework 1: ASP.NET Core LMS Implementation	AD602	Submitted	just now	Grade
Yasas Pasindu Fernando	Weekly Mind Map Portfolio	AI603	Submitted	just now	Grade
Yasas Pasindu Fernando	Coursework 1: Software Engineering Management Report	ASE601	Graded	just now	Review
Erandika Diss	Coursework 1: ASP.NET Core LMS Implementation	AD602	Submitted	13h ago	Grade
Alex Student	Coursework 1: ASP.NET Core LMS Implementation	AD602	Submitted	2d ago	Grade

Figure 10b: Lecturer Pending Submissions

This screenshot shows the Lecturer pending submissions view.

8.7 Administrator Dashboard and User Management

The Administrator dashboard is implemented by DashboardController.Administrator and Views/Dashboard/Administrator.cshtml. The action queries the database to count users, courses, enrolments, assignments, submissions, and messages, identify popular courses by enrolment count, and load recent audit log entries.

Admin User Management was refined as a practical account-control feature. It is implemented through Controllers/AdminUsersController.cs, ViewModels/AdminUserManagementViewModels.cs, and Views/AdminUsers/. The Administrator can review registered users, create accounts, edit account details and roles, and lock or unlock accounts. The list shows practical account details such as username, full name, email, phone, role, lock status, and email confirmation status. The create and edit screens use server-side validation before Identity records are changed.

The controller also contains safety checks. An Administrator cannot lock their own account. Locking another administrator is blocked unless at least two administrators exist. Editing the only remaining administrator is blocked, and removing the last administrator role is also blocked. Deletion is limited to non-administrator accounts; administrator accounts and the

signed-in administrator's own account cannot be deleted. This keeps the feature useful for coursework administration without creating an obvious risk of losing administrator access.

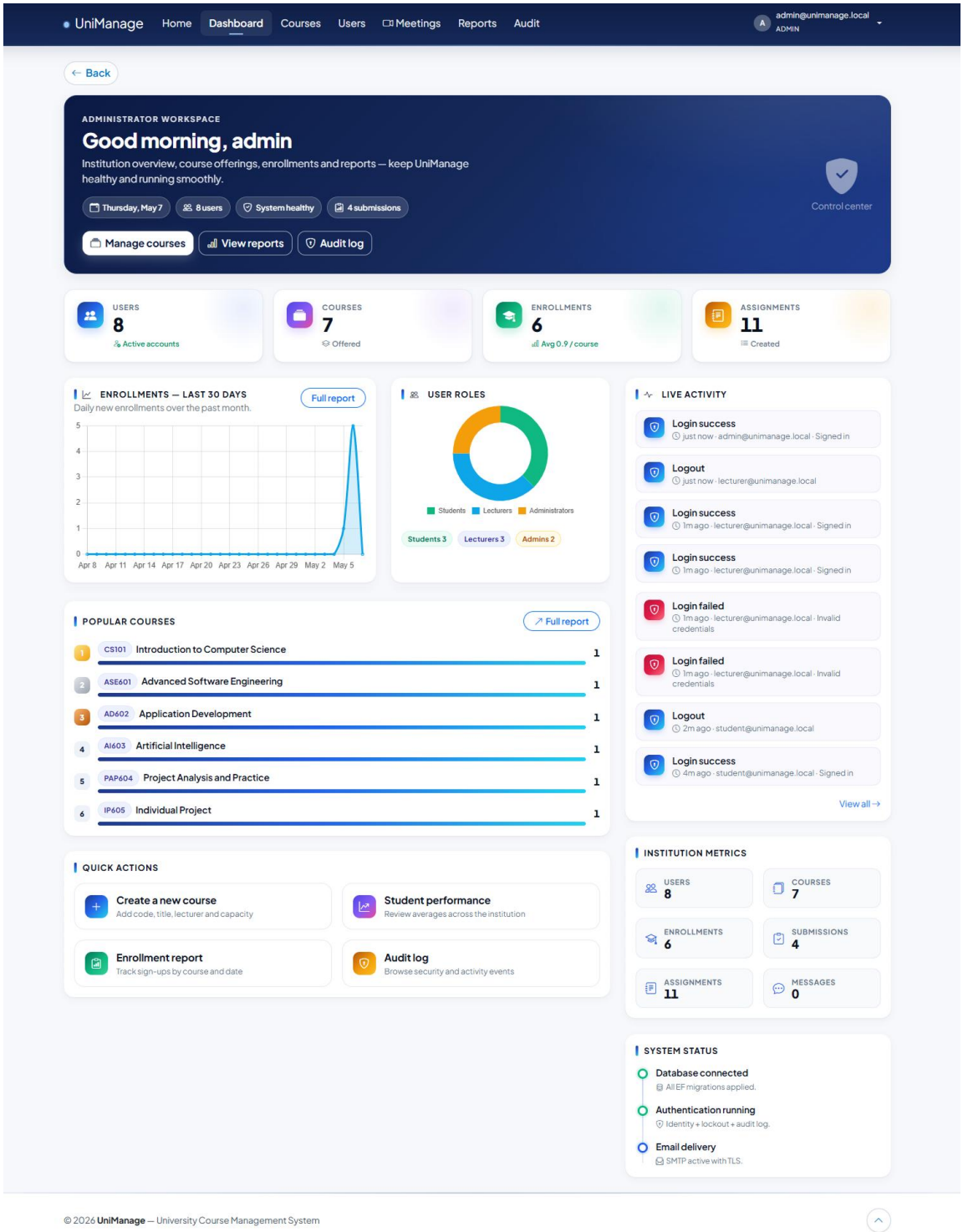


Figure 11: Administrator Dashboard

This screenshot shows the Administrator dashboard.

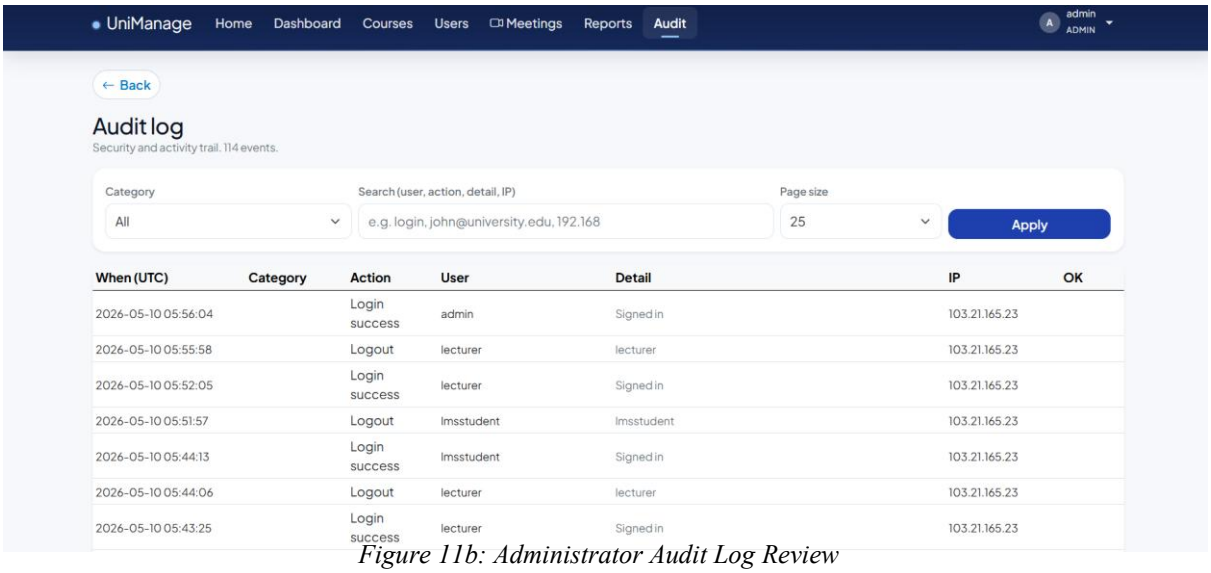


Figure 11b: Administrator Audit Log Review

This screenshot shows the Administrator audit log review.

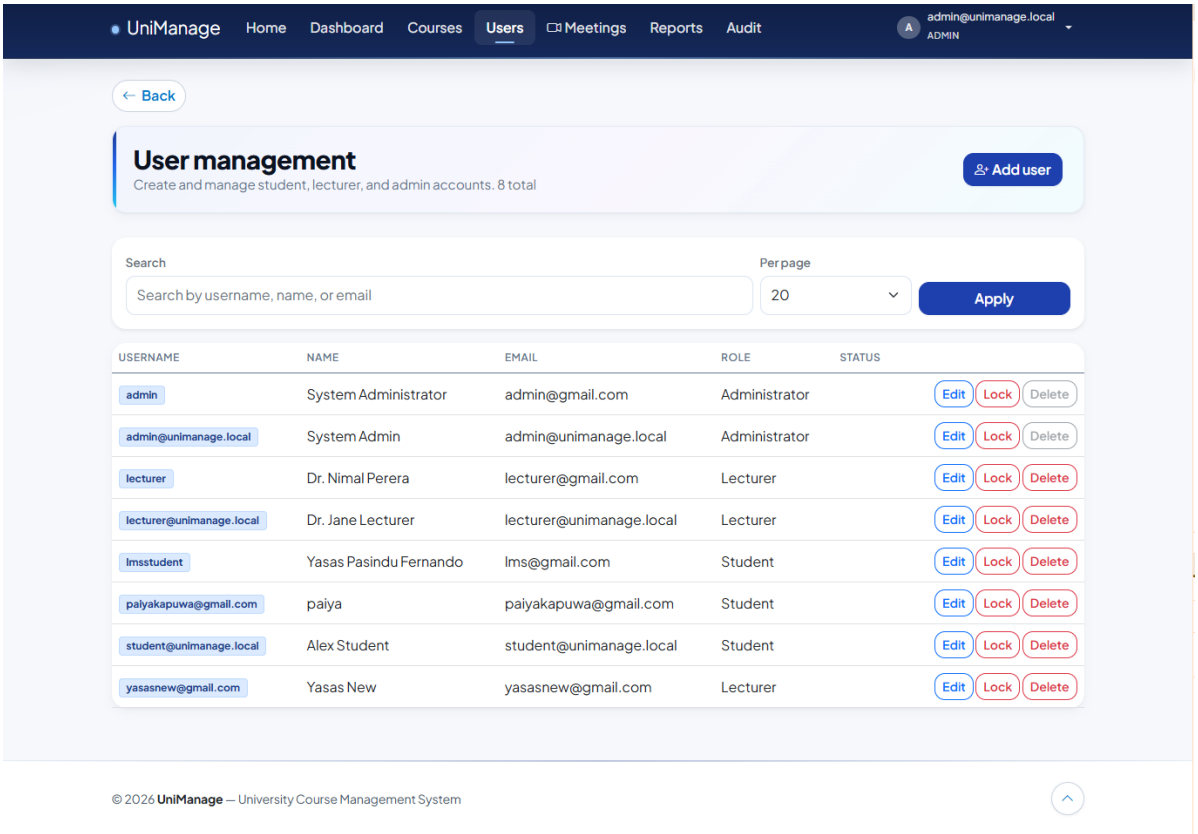


Figure 11c: Admin User Management Screen

This screenshot shows the Admin User Management list with user roles and lock status visible.

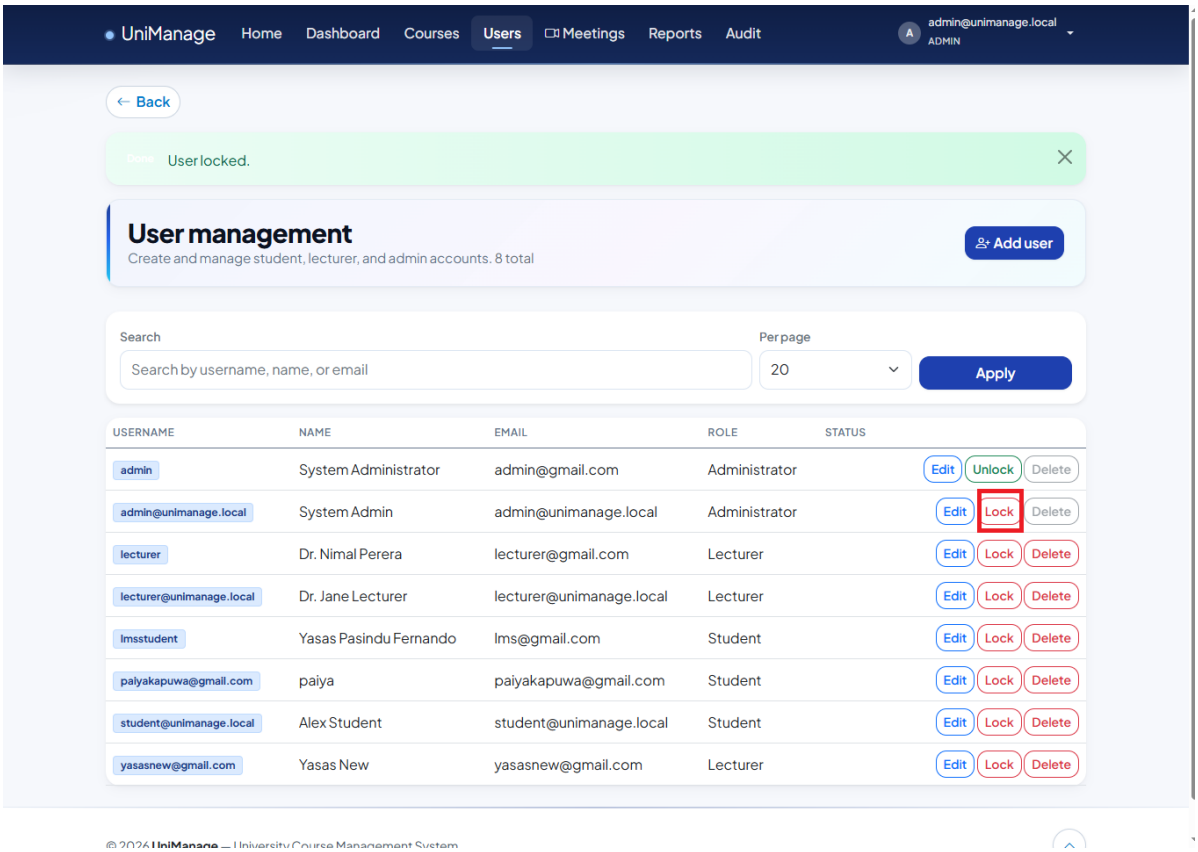


Figure 11d: User Lock and Unlock Evidence

This screenshot shows a lock or unlock action message from the Admin User Management screen.

8.8 Course Management

Course management is implemented in Controllers/CoursesController.cs and the Views/Courses/ folder. The actions are:

Index: lists courses with search and paging support.

Details: shows a single course.

Create (GET and POST): allows an Administrator to create a new course using CourseInputViewModel for validation. The unique course code is enforced through a database unique index.

Edit (GET and POST): allows an Administrator to update an existing course.

Delete (GET and POST): allows an Administrator to delete a course, with appropriate cascade or restrict behaviour.

MyCourses: allows a Lecturer to see only their own courses.

UploadMaterial (POST): allows a Lecturer to upload a course material file for an owned course. The action validates the file extension, the file size, the owner, and applies the upload rate-limiting policy.

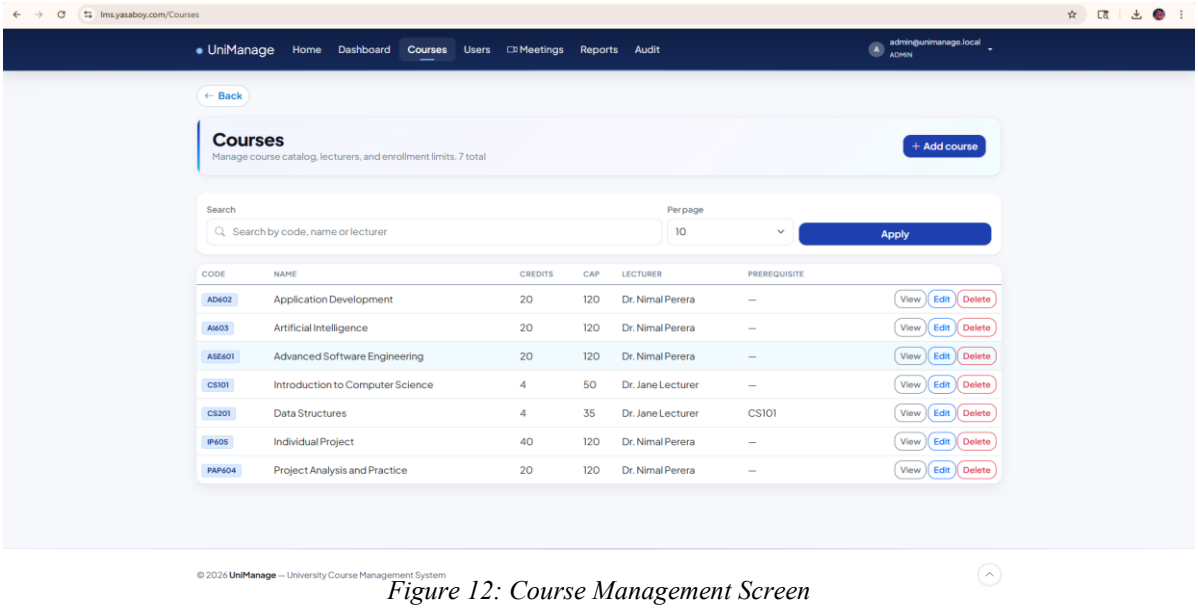


Figure 12: Course Management Screen

This screenshot shows the course list (administrator).

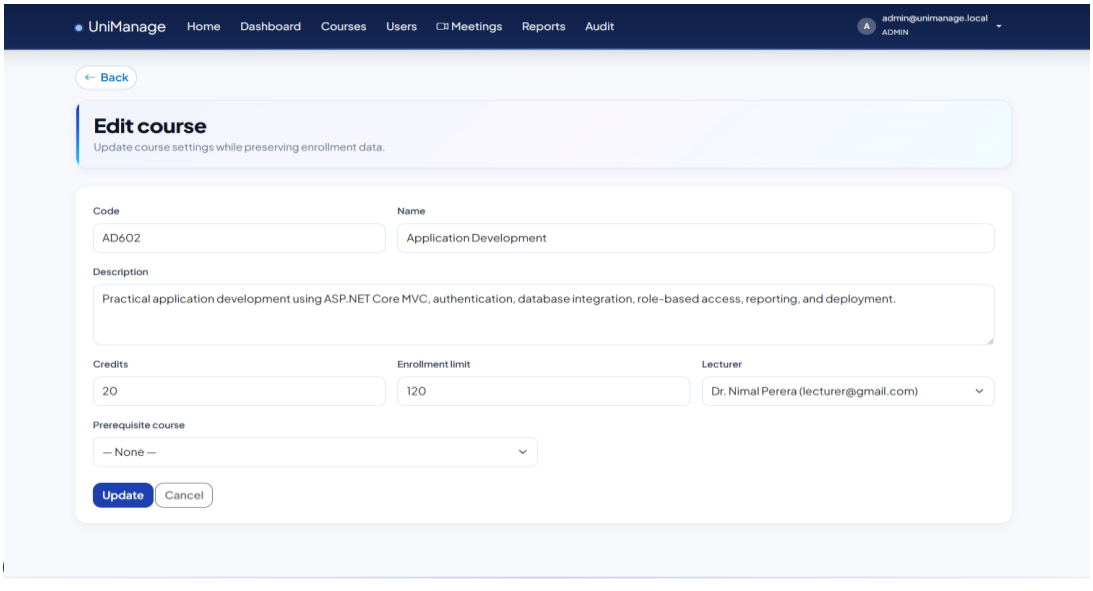


Figure 12b: Course Create or Edit Form

This screenshot shows the course create form.

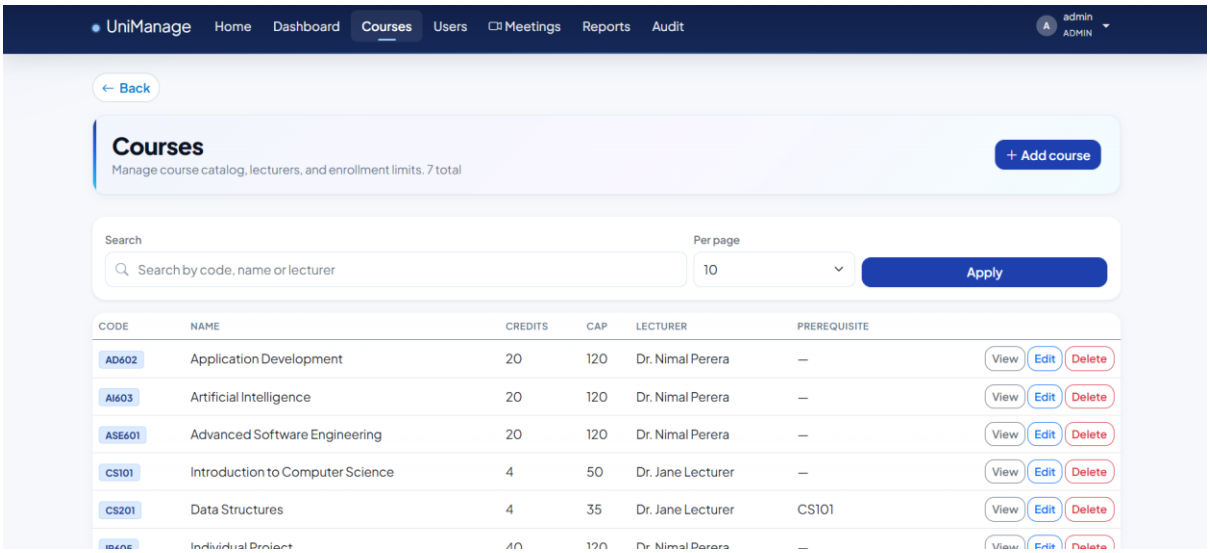


Figure 12c: Course Edit Form

This screenshot shows the course edit form.

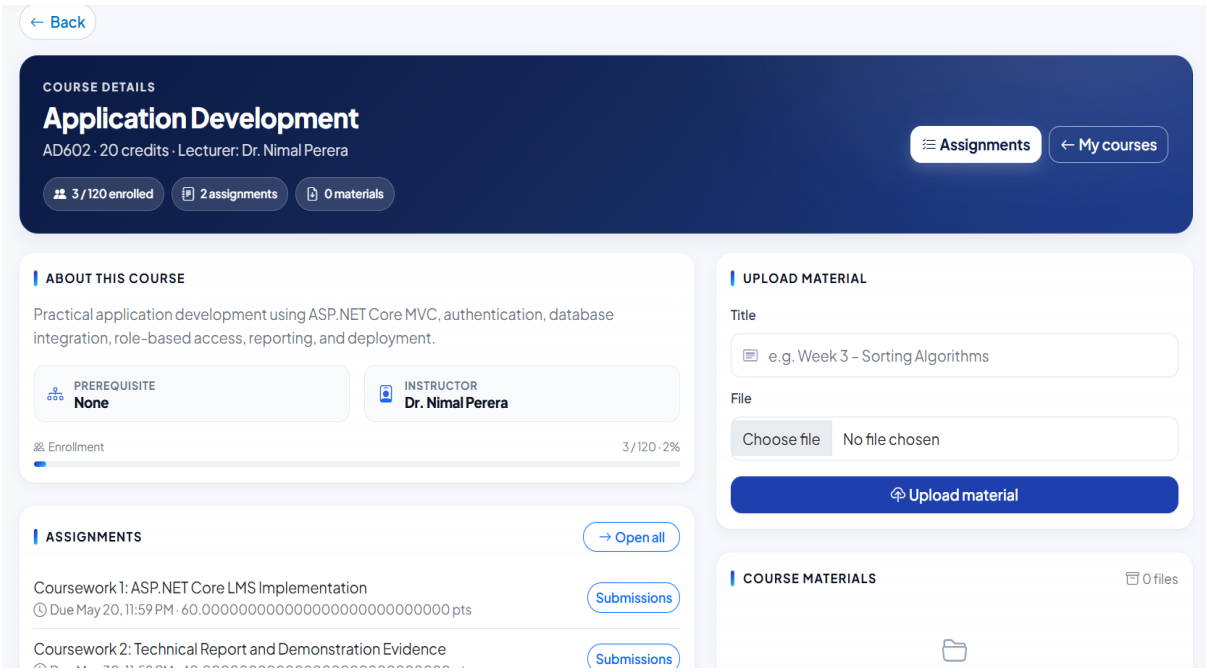


Figure 12d: Course Material Upload

This screenshot shows the course material upload.

8.9 Enrollment System

The enrolment system is implemented in Controllers/EnrollmentsController.cs. The Browse action lists eligible courses for the signed-in Student, joining the courses table with the student's enrolments to compute eligibility. The Enroll POST action checks that the course exists, that the course is open (i.e. capacity has not been reached), that the student has not already enrolled (uniqueness check), and that any prerequisite has been satisfied. If all checks pass, the action creates an Enrollment record and writes an audit log entry.

The eligibility check uses a single composed LINQ query rather than separate queries to keep the database round-trip count low. The capacity check uses the count of existing enrolments for the course compared with the EnrollmentLimit field.

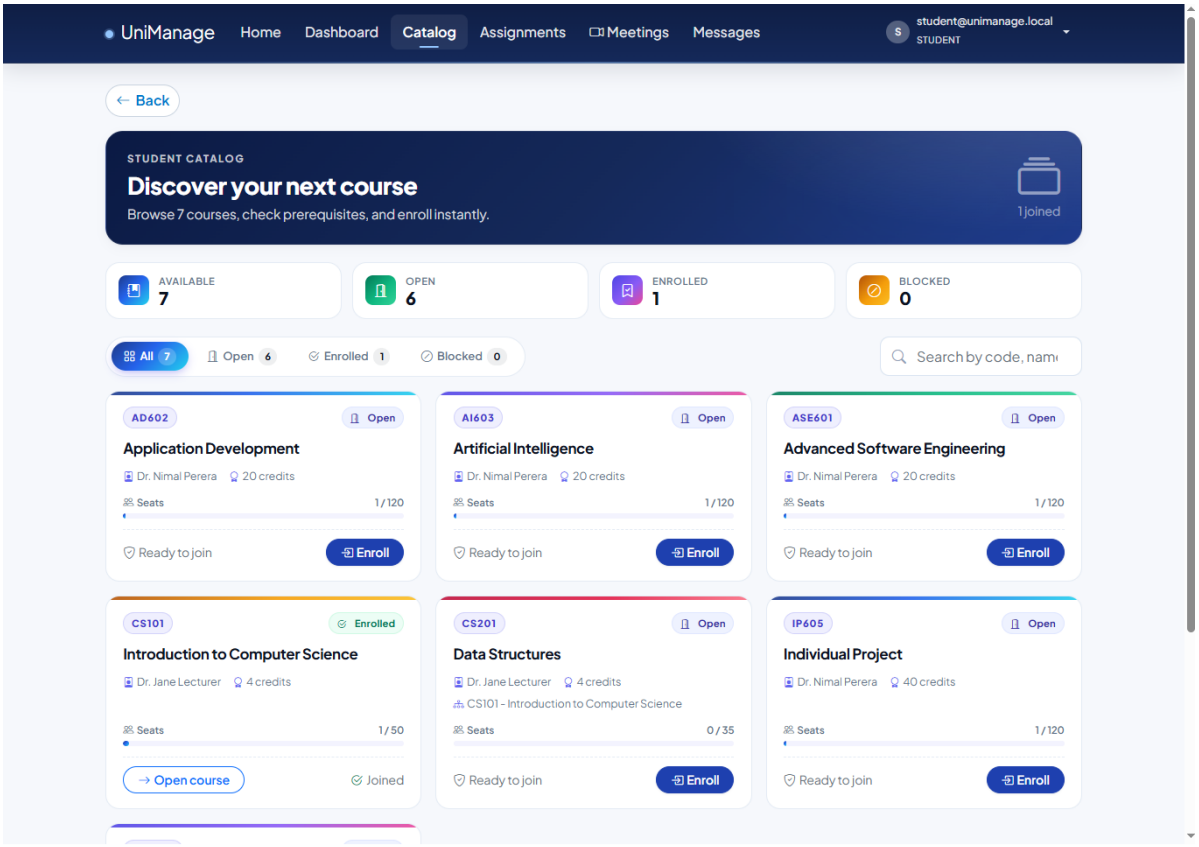


Figure 13: Student Course Enrolment Page

This screenshot shows the course browse (student).

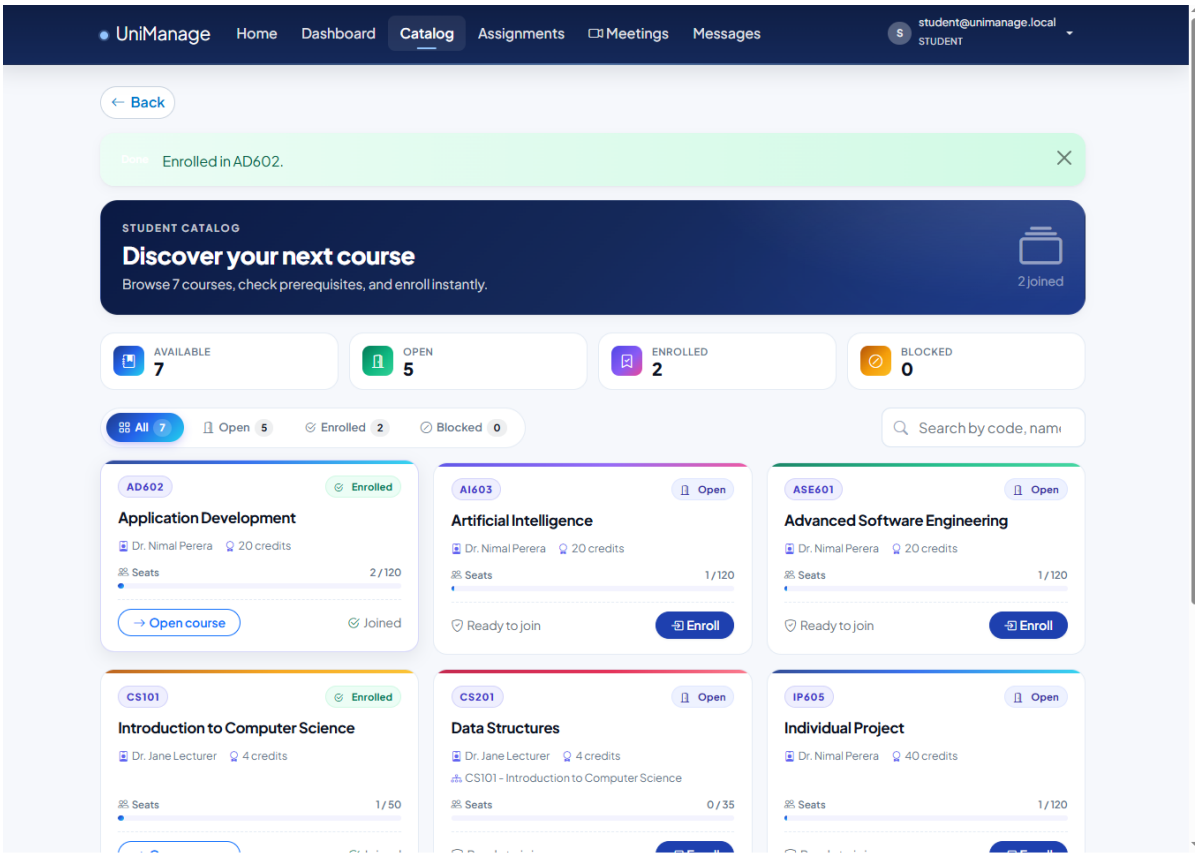


Figure 13b: Successful Course Enrolment Evidence

This screenshot shows the successful enrolment confirmation.

8.10 Assignment Management

Assignment management is implemented in Controllers/AssignmentsController.cs and the Views/Assignments/ folder. The actions are:

ForCourse: lists assignments for a given course.

Mine: lists assignments owned by the signed-in lecturer.

Create (GET and POST): allows a Lecturer to create an assignment for an owned course using AssignmentInputViewModel.

Edit (GET and POST): allows a Lecturer to update their own assignment.

Delete (GET and POST): allows a Lecturer to delete their own assignment.

The validation enforced through AssignmentInputViewModel includes a required title, a maximum length on the description, a future or near-future due date, and a sensible MaxPoints range.

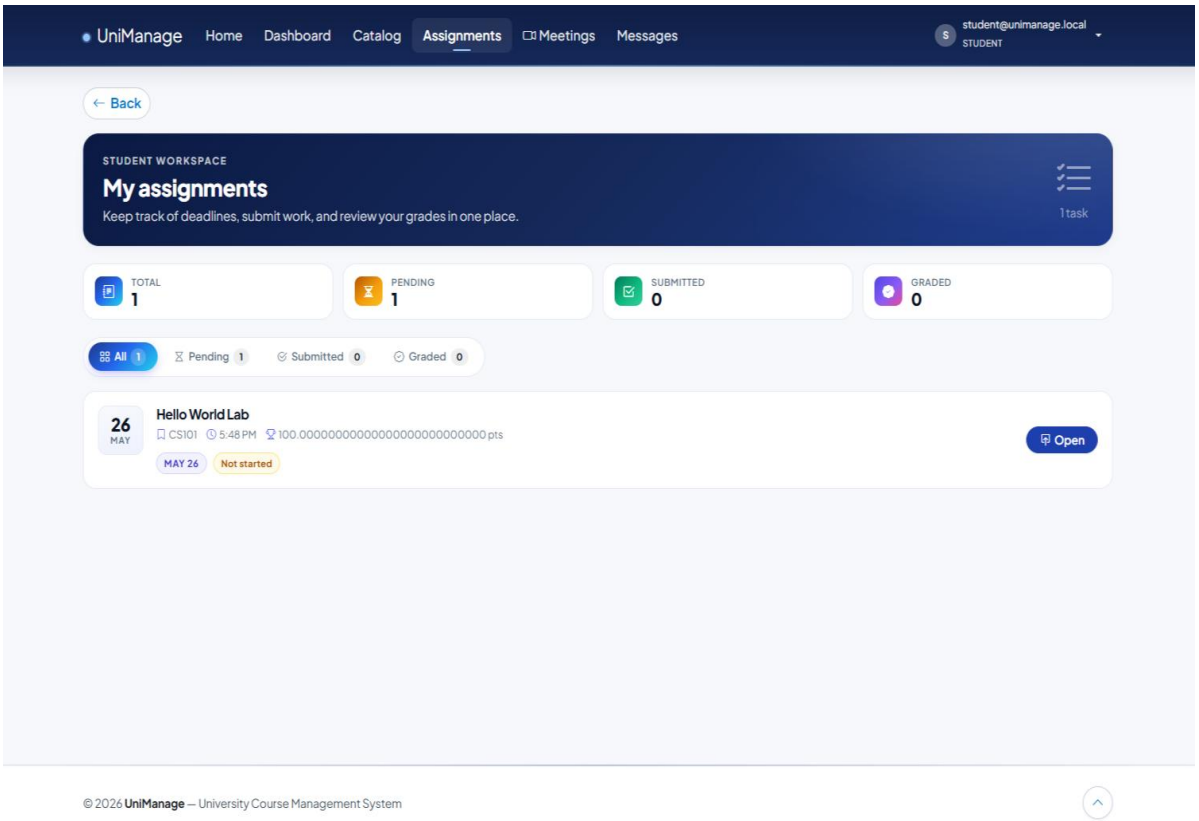


Figure 14: Assignment Management Screen

This screenshot shows the Lecturer assignment list.

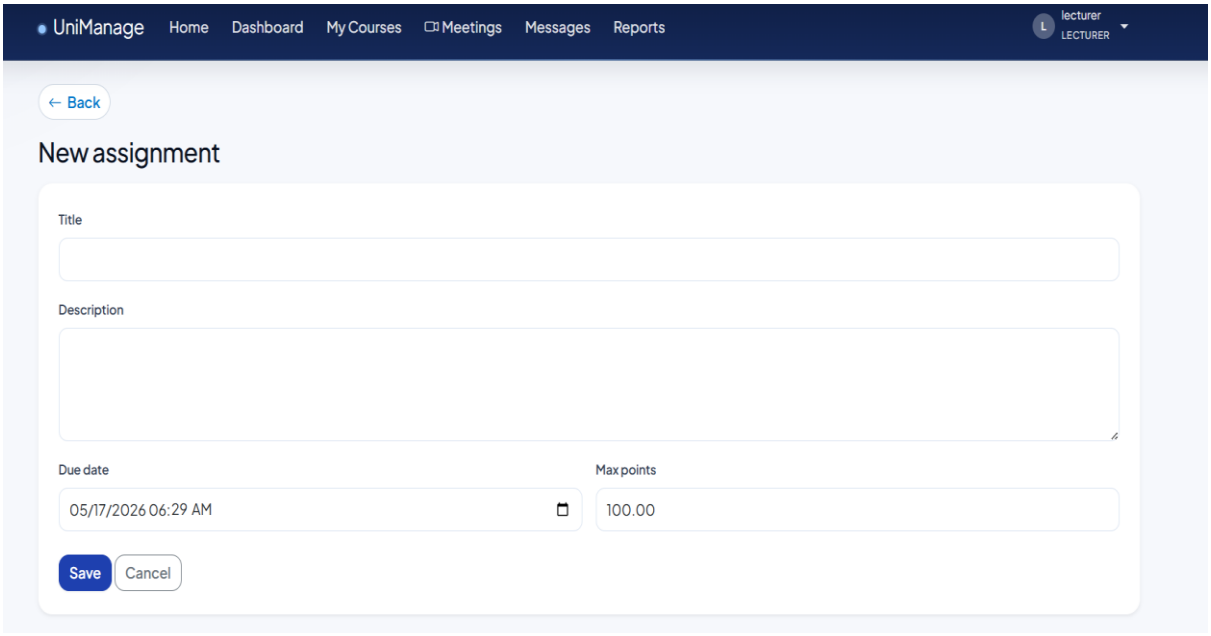


Figure 14b: Assignment Create Form

This screenshot shows the assignment creation form.

8.11 Submission and Grading

Submission and grading are implemented in Controllers/SubmissionsController.cs and the Views/Submissions/ folder. The Student-facing action Submit accepts text content and a file, validates that the student is enrolled on the course, validates the file using UploadedFileStore, and creates or updates a Submission record. If the submission has already been graded, the submission is locked and further changes are not allowed.

The Lecturer-facing action Grade checks that the lecturer owns the assignment's course, displays the submission, and records a grade and feedback through GradeSubmissionViewModel. The grade is constrained to a sensible range and the feedback is constrained to a maximum length.

The action ForAssignment allows a Lecturer to see all submissions for an assignment. The actions CourseMaterialFile and SubmissionFile provide secure file download paths that check role and ownership before streaming the file.

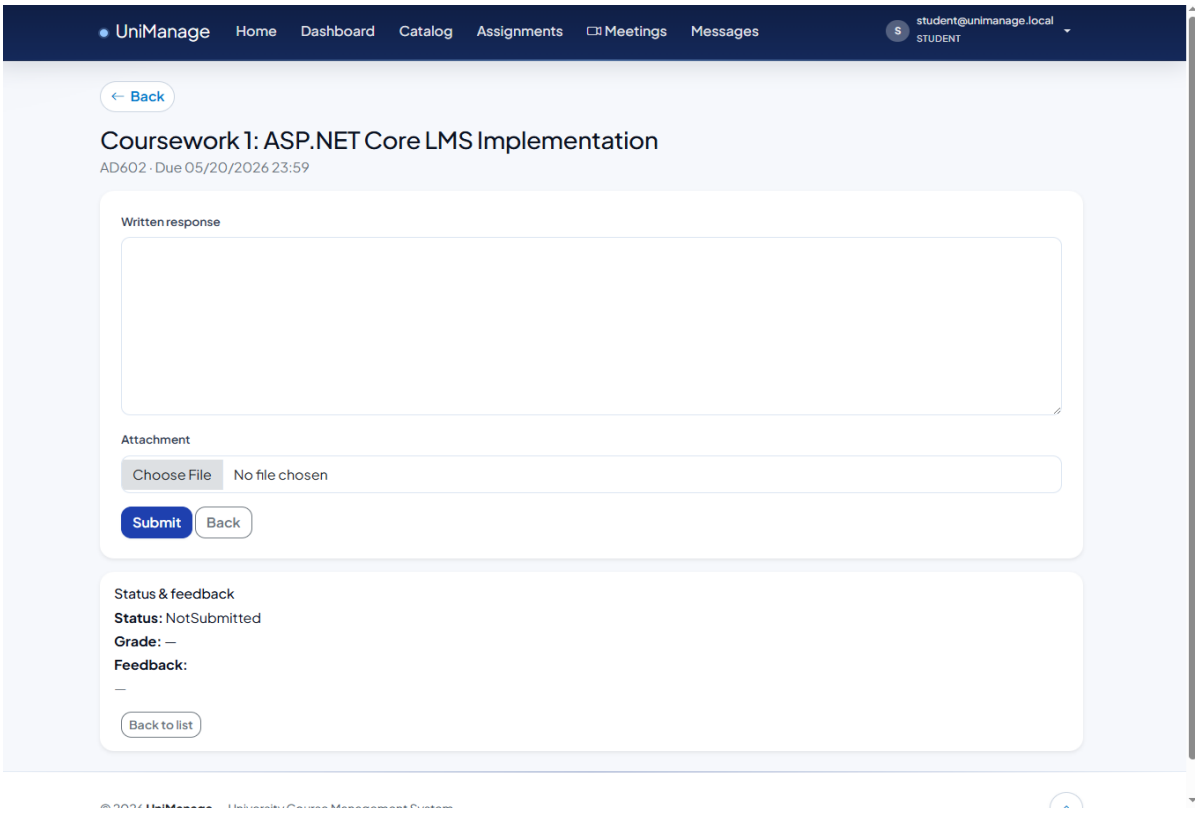


Figure 14c: Student Assignment Submission Page

This screenshot shows the Student assignment submission form.

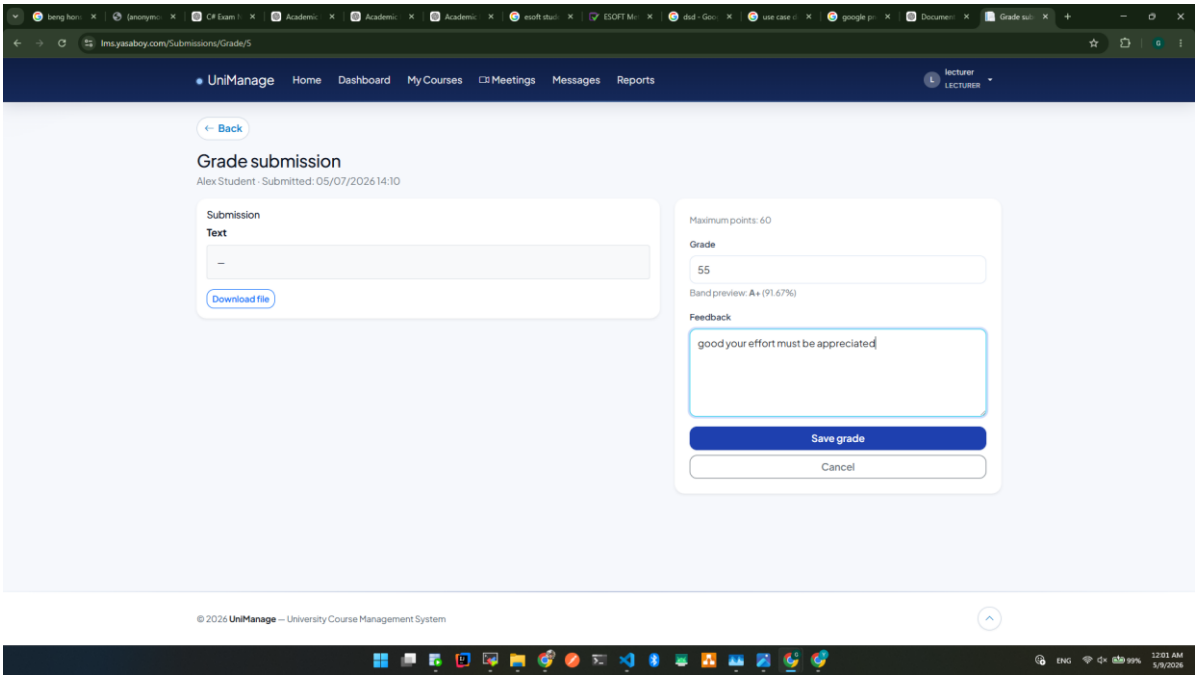


Figure 14d: Submission Grading Page

This screenshot shows the lecturer grading page for a submitted assignment. It provides evidence for the assignment feedback and grading workflow.

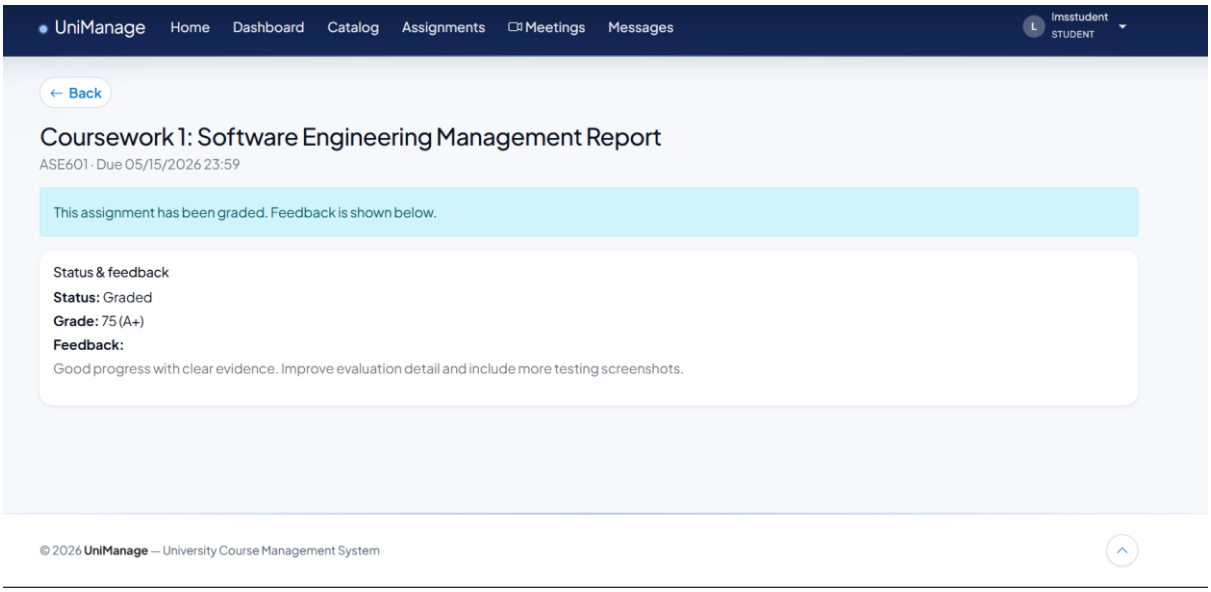


Figure 14e: Student Graded Submission View

This screenshot shows the graded submission view (student).

8.12 Reporting and Analytics

The reporting suite is implemented in Controllers/ReportsController.cs and the Views/Reports/ folder. The reports include:

CoursePopularity: groups enrolments by course and orders by enrolment count.
StudentPerformance: aggregates grades per student.
LecturerWorkload: aggregates assignments and pending submissions per lecturer.
Enrollments: lists enrolments with filters such as course or date.
PassFail: groups submissions into pass and fail buckets using a configurable threshold.
AssignmentAttendance: shows the proportion of enrolled students who submitted each assignment.

CSV export is provided through Infrastructure/CsvWriter.cs, which writes properly escaped CSV content. PDF export is provided through Infrastructure/PdfReport.cs, which uses QuestPDF to render the report layout. The QuestPDF licence is set to Community in Program.cs.

The report actions are role-aware. Administrators can access the full report set. Lecturers can access course popularity, student performance, and lecturer workload views with lecturer-scoped data where the controller applies the signed-in lecturer filter. This supports academic decision-making without exposing unrelated course information to a lecturer.

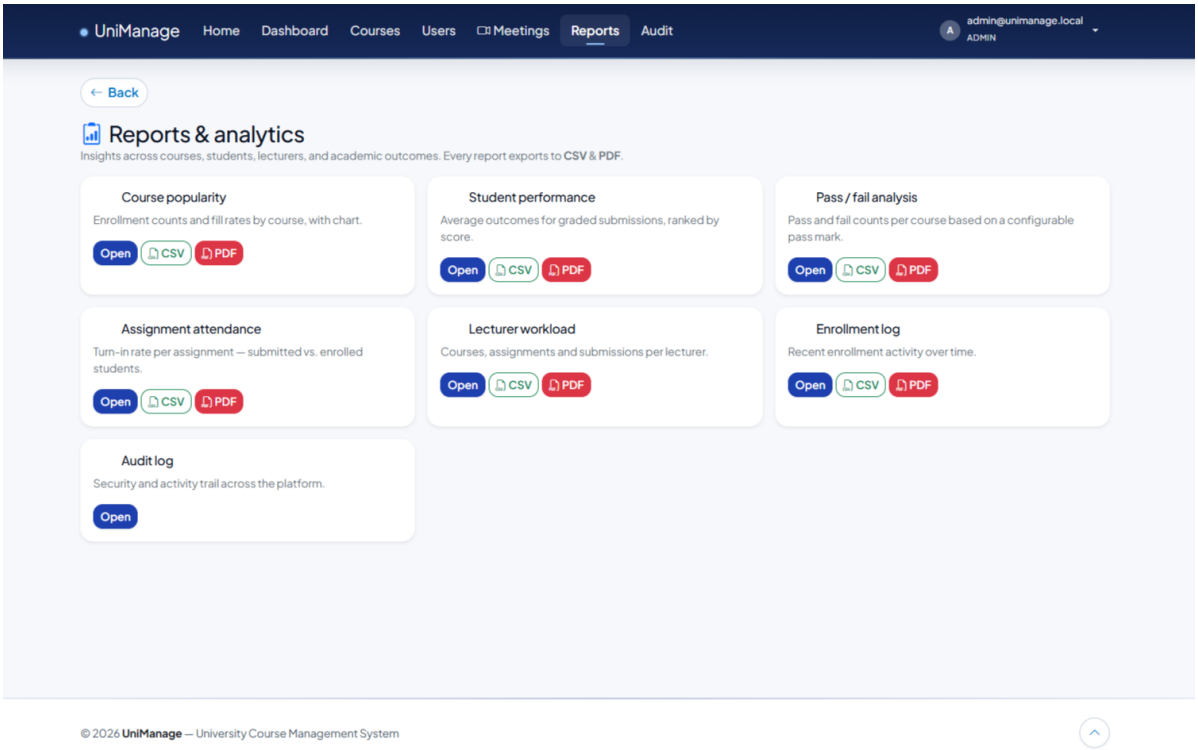


Figure 15: Reporting and Analytics Screen

This screenshot shows the reports landing page.

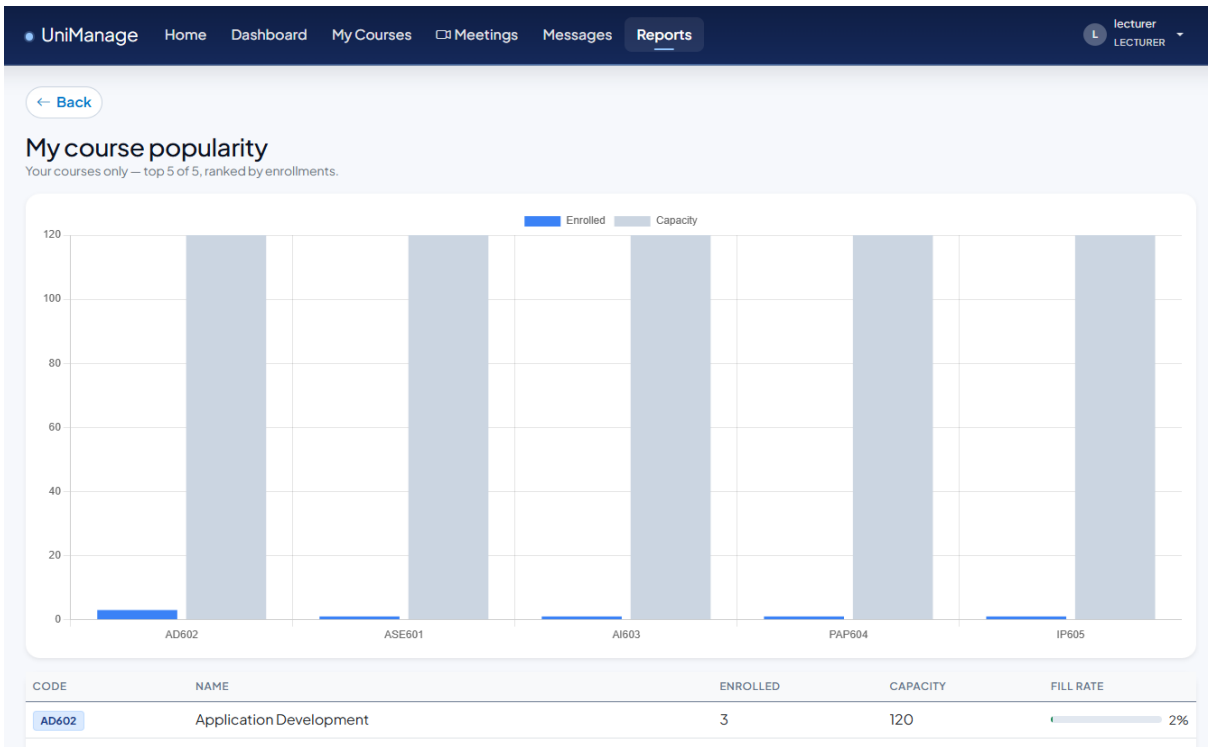


Figure 15b: Sample Report Output

This screenshot shows the sample report output.

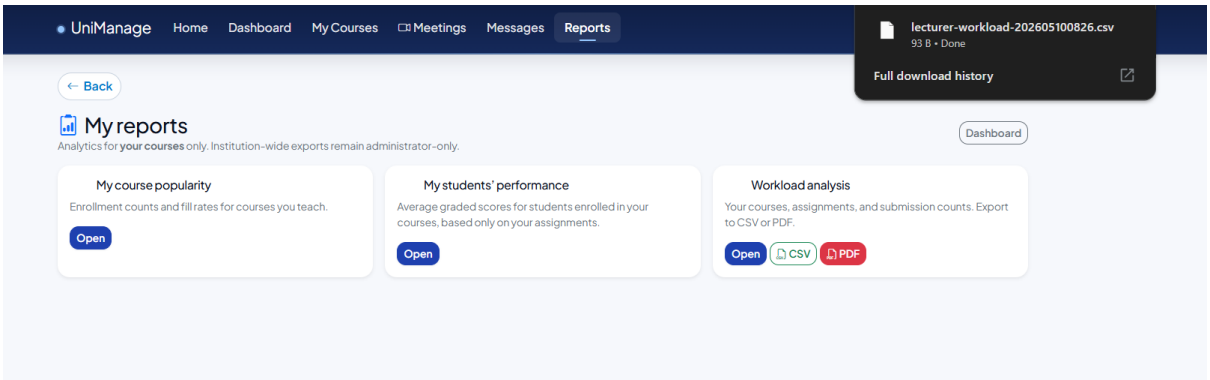


Figure 15c: CSV Export Evidence

This screenshot shows the CSV export evidence.

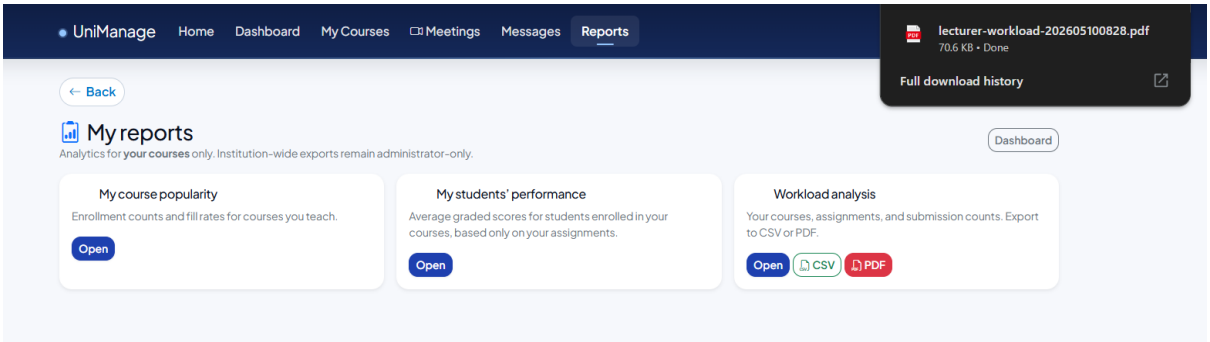


Figure 15d: PDF Export Evidence

This screenshot shows the pdf export evidence.

8.13 Communication Module

The communication module is implemented in Controllers/MessagesController.cs and the Views/Messages/ folder. The actions are:

Inbox: lists messages for the signed-in user using MessageInboxRowViewModel.

Thread: shows a conversation between two users using MessageThreadViewModel.

Reply (POST): adds a reply message to a thread.

Compose (GET and POST): allows the user to compose a message to an allowed recipient using MessageComposeViewModel. The validation rule is that a Student may message a

Lecturer who teaches a course on which the student is enrolled, and a Lecturer may message a Student who is enrolled on a course owned by the lecturer.

MarkAllRead: marks all messages in the inbox as read.

The Reply action also supports optional photo attachment in a message thread. The controller restricts the accepted file types to JPG, PNG, GIF, and WEBP, limits the size to 5 MB, and stores accepted images under wwwroot/uploads/messages. This is treated as a supporting communication refinement, not as a separate broadcast feature.

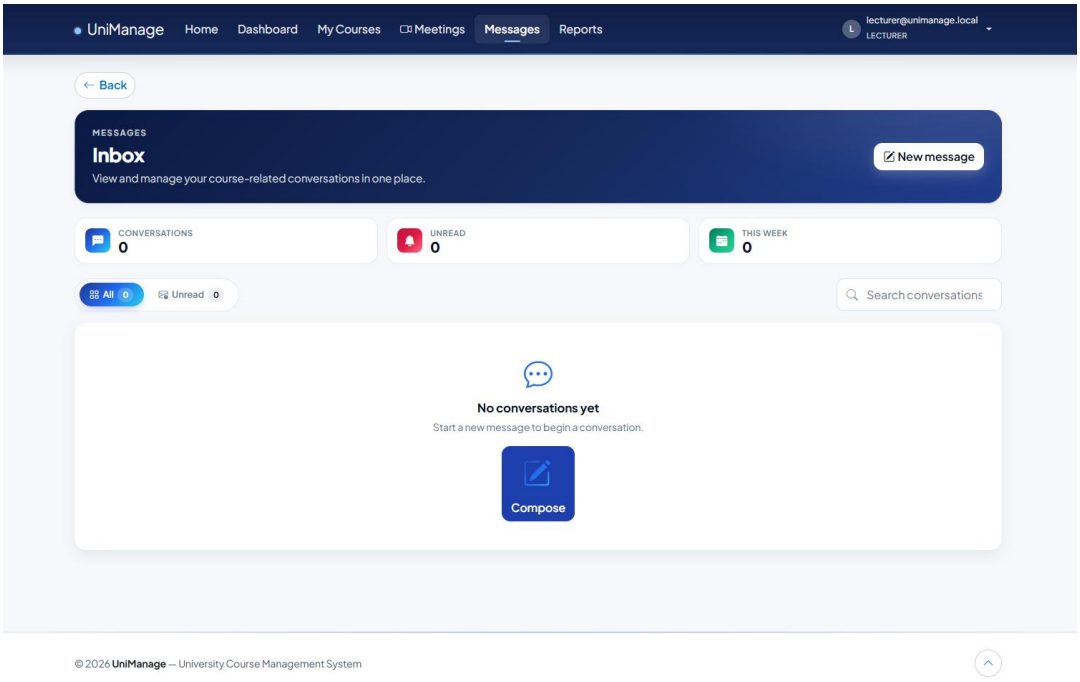


Figure 16: Messaging Page

This screenshot shows the inbox view.

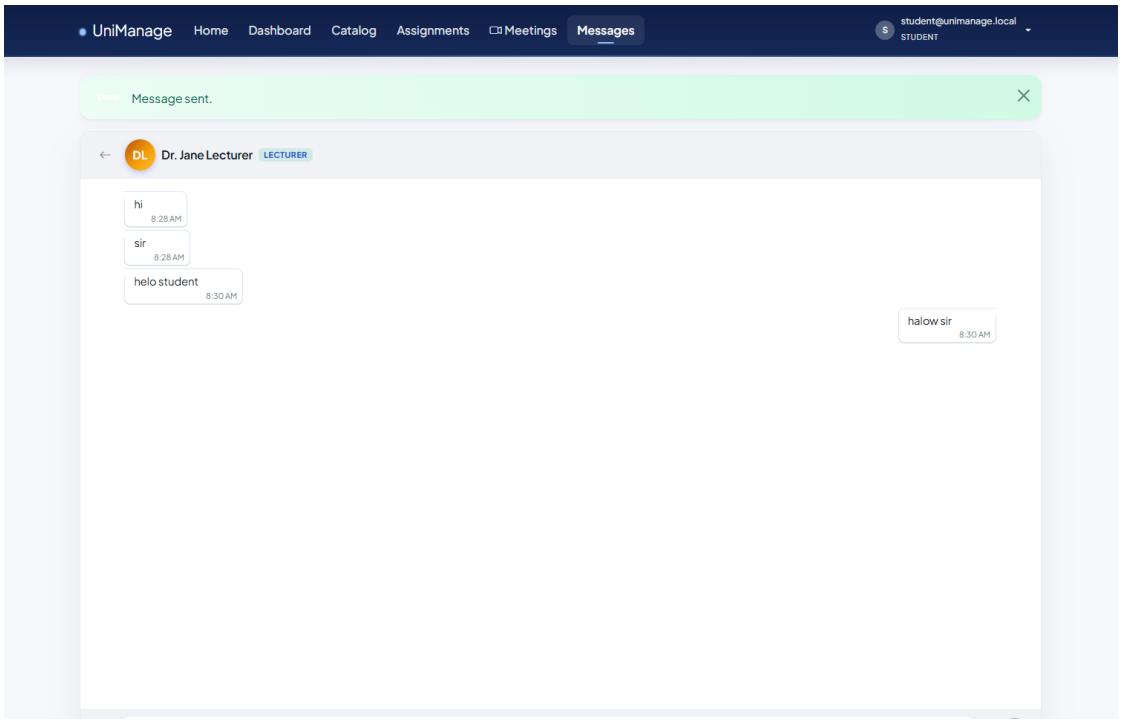


Figure 16b: Chat / Message Thread

This screenshot shows the message thread view.

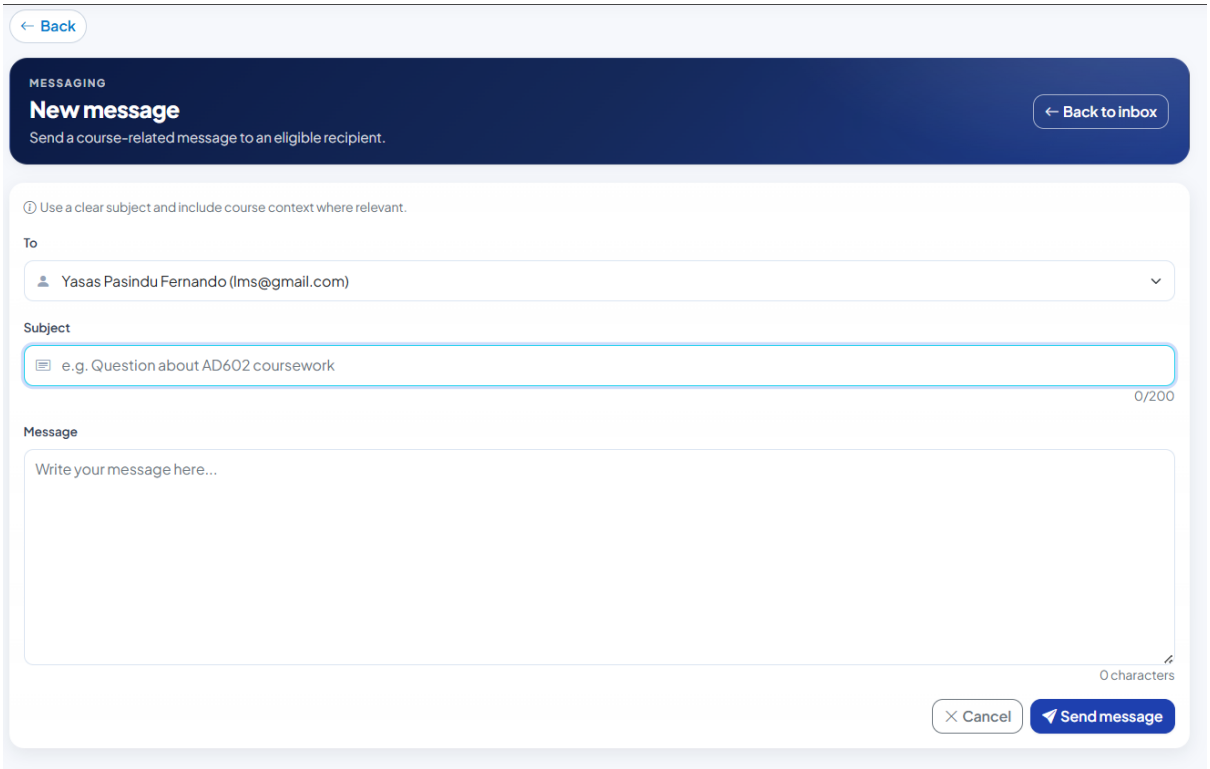


Figure 16c: Compose Message Form

This screenshot shows the compose message form.

8.14 Meeting Scheduling

Meeting scheduling is implemented in Controllers/MeetingsController.cs and the Views/Meetings/ folder. The actions are:

Index: lists meetings for the signed-in user.
Create (GET and POST): allows a Lecturer to schedule a meeting for an owned course using MeetingInputViewModel. The form captures the title, the description, the scheduled date and time, the duration, and an optional meeting URL.

Edit (GET and POST): allows a Lecturer to update an owned meeting.

Delete (GET and POST): allows a Lecturer to delete an owned meeting.

Join: redirects the user to the meeting URL after authorisation checks.

Ics: produces an ICS calendar file using Infrastructure/CalendarLink.cs, which can be imported into a calendar application.

Infrastructure/MeetLinkGenerator.cs can generate a Google Meet style URL when required, and the meeting form can also store a URL provided by the lecturer. Students see meetings for enrolled courses. Lecturers manage meetings for their own courses. Administrators can review meeting records and join links through the authorised meeting list, but the system does not claim attendance tracking or a separate broadcast feature.

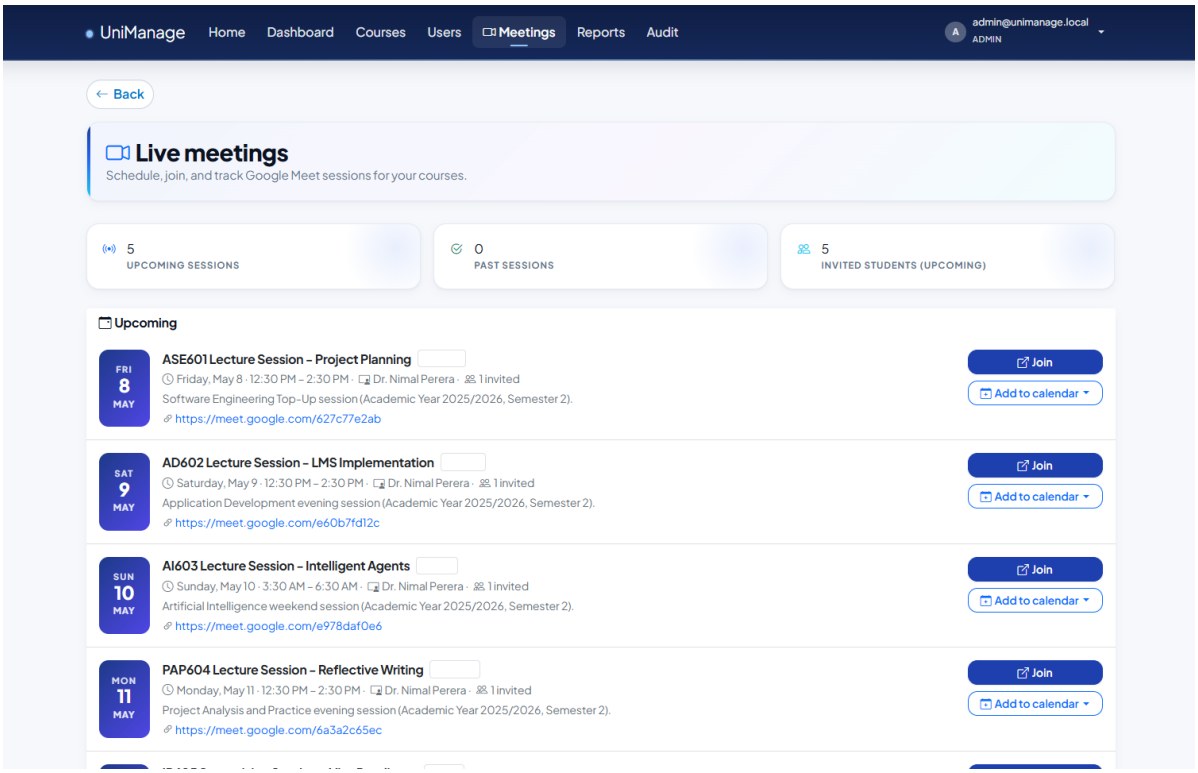


Figure 16d: Meeting Scheduling Screen

This screenshot shows the meetings list.

8.15 Course Materials and File Uploads

Course materials are implemented through the CourseMaterial entity, the MaterialUploadViewModel, and the UploadMaterial action of CoursesController. Uploaded files are stored on disk through Infrastructure/UploadedFileStore.cs, which validates the file extension and the file size, generates a stored file name to avoid collisions, and writes the metadata into the database.

Downloads are served through controlled actions that confirm role and ownership before streaming the file. The action does not expose direct URLs into the file system, which prevents path traversal and unauthorised access to stored files.

8.16 Profile and Password Management

The profile and password actions are implemented in Controllers/ProfileController.cs and the Views/Profile/ folder. The Index action allows the signed-in user to view and update their profile fields through ProfileViewModel. The ChangePassword action allows the user to change their password under authentication, using the standard Identity password change API.

8.17 Audit Logging

Audit logging is implemented through Models/AuditLog.cs, Infrastructure/IAuditLogger.cs, Infrastructure/AuditLogger.cs, and the AuditLogsController. The audit logger writes an AuditLog record with the following fields: category (for example Account, Course, Enrollment, Submission, Security), action (for example Login, Register, Create, Update, Delete, Grade, Upload, Download), detail (a short, human-readable description), user identifier (where available), and a success flag. The log is reviewed through Views/AuditLogs/Index.cshtml, which displays paged log entries for an Administrator.

The audit logger is wrapped in a try/catch block inside the implementation so that a failure to write an audit entry does not crash the calling action. This is appropriate because the audit log is a supplementary record rather than the primary data store. If a log entry cannot be written, the action continues, and the failure is reported through the standard exception handler to the application logs.

8.18 Infrastructure Services

The Infrastructure/ folder contains services that support the controllers without holding business state. The full list is:

- AuditLogger.cs and IAuditLogger.cs: audit logger described in Section 8.17.
- CalendarLink.cs: builds ICS calendar content for meetings.
- CsvWriter.cs: writes CSV content with escaping.
- EmailSettings.cs: strongly typed settings for SMTP.
- EmailTemplates.cs: holds the email templates used by the password reset and registration welcome email flows. It also formats user-facing date and time values, including converting the registration joined time from UTC to Sri Lanka local time before display.
- IEmailService.cs and SmtpEmailService.cs: email service contract and SMTP implementation.
  MeetLinkGenerator.cs: helper used during meeting creation.
- PdfReport.cs: PDF rendering using QuestPDF.
- SecurityHeadersMiddleware.cs: custom middleware that sets security headers on every response.

- UploadedFileStore.cs: manages uploaded file validation and storage.

Each service is registered through dependency injection in Program.cs where appropriate. Controllers depend on the interface (for example IAuditLogger, IEmailService) rather than on the concrete implementation, which keeps the controllers easy to test and replace.

8.19 UI Layout and Static Assets

The UI layout is centralised in Views/Shared/_Layout.cshtml. The layout defines the document head, the navigation bar, the alert partial, the main content section, and the footer. The alerts partial Views/Shared/_Alerts.cshtml displays success, information, warning, and error messages using TempData. The Views/Shared/_DashboardCalendar.cshtml partial is reused by the dashboards to render a small calendar block.

Static assets live under wwwroot/. The lib/ folder holds Bootstrap and any other client libraries; the css/site.css file holds project-specific styles; the js/site.js file holds custom client-side behaviour; the assets/ folder holds images and icons used by the UI. The author has kept the custom CSS and JavaScript small to avoid duplicating the framework features and to keep the code reviewable.

8.20 Hosting and Docker Deployment Evidence

The application was also hosted on a Google Cloud virtual machine using Docker as supporting demonstration evidence. The hosted version is available through the custom domain and was used to support review, browser testing and viva preparation. This hosted deployment does not replace the local Visual Studio build route required for assessment, but it shows that the application can run outside the development machine.

Docker was used because it gives the application a more consistent runtime between the development machine and the cloud host. The Dockerfile describes the .NET 8 base image, build steps, and entry point for the published application. After the image is built, it can be transferred to the Google Cloud virtual machine and started through Docker. This helped reduce environment differences during testing. The Dockerfile itself remains in the source repository rather than being copied into the report body.

Cloudflare DNS and Nginx/server reachability evidence are included as supporting deployment evidence. These screenshots are used to show the hosting setup path, while full CDN/security

hardening is not claimed as a completed production feature. Figure UM19 is used as hosted UniManage application evidence, and Figures UM21 and UM22 support the DNS and server setup evidence.

The hosted version was used mainly to make testing and review easier. It helped the author check the application outside the local Visual Studio environment and share the system through a browser. This made configuration, login, navigation, and role-based access issues easier to notice. It is included as supporting evidence only; the local Visual Studio build remains the main assessment route.

The hosted version is included as supporting coursework evidence to demonstrate that the application can run in a hosted environment and to support easier review and testing. It is not presented as a live production service.

The custom domain was configured for the hosted demonstration version. HTTPS/domain access was used where available during final testing. Full production hardening, long-term monitoring, automated backups and formal security testing remain future improvements. Connection strings, SMTP passwords, Google client secrets, SSH private keys, DNS or provider credentials, and cloud service account keys are kept outside the report and outside screenshots.

The hosting evidence summary is shown in Table 10 and repeated in Appendix C.

Evidence Item	Details	Status	Notes
Hosted application on custom domain (UniManage UI)	https://lms.yasaboy.com/	Completed as hosted UI evidence	Figure UM19 shows the UniManage homepage; Figure UM22 remains server reachability evidence only
Previous IP-based endpoint	http://34.171.6.234:8080	Completed / Backup evidence	Earlier Google Cloud hosted endpoint, retained as fallback evidence
Hosting platform	Google Cloud	Completed	VM-hosted deployment evidence
Containerisation	Docker	Completed	Application packaged for deployment
HTTPS / TLS on custom domain	HTTPS/domain access used where available during final testing	Supporting demonstration evidence	Full production hardening, long-term monitoring, automated

Evidence Item	Details	Status	Notes
(browser session evidence)			backups and formal security testing remain future improvements.
Cloudflare DNS record	lms A record points to 34.171.6.234	Supporting deployment evidence	DNS evidence included; full CDN/security hardening is not claimed.
Nginx server reachability	Default Nginx page loads on lms.yasaboy.com	Supporting server evidence	Server reachability evidence only; full reverse proxy hardening is not claimed.
Cloudflare DNS/security hardening	DNS evidence included	Future improvement	Full CDN/security hardening is not claimed as a completed production feature.
Nginx application routing/hardening	Supporting server evidence only	Future improvement	Full reverse proxy hardening is not claimed as a completed production feature.
Visual Studio local assessment	Required	Completed for build; local route remains primary	Coursework assessment route as per the brief
Secrets and credentials	Not included in report	Safe	Kept outside appsettings.json screenshots and outside the body of the report

Table 10: Hosting and Deployment Evidence

Cloudflare DNS and Nginx/server reachability evidence are included as supporting deployment evidence. These screenshots are used to show the hosting setup path, while full CDN/security hardening is not claimed as a completed production feature.

8.21 Environment-Based Configuration and Secrets

Sensitive configuration is not committed directly to the repository. Google OAuth Client ID and Client Secret are supplied through Authentication_Google_ClientId and Authentication_Google_ClientSecret where Docker or server-level environment variables are used. SMTP settings are supplied through the Email_... variables, and the database connection string is supplied through ConnectionStrings_DefaultConnection for Docker deployment or local configuration during development.

deployment.env.example is a safe template only. It names the required settings but does not contain real secrets. Real database passwords, OAuth client secrets, SMTP passwords, SSH keys, cloud service account keys, and DNS provider credentials must remain outside the public repository and outside screenshots.

Google OAuth depends on the authorised redirect URIs configured in Google Cloud. Local and hosted environments can require different redirect URI values, especially when HTTPS, custom domains, Docker, or reverse proxy routing are involved. Forwarded-header handling is only claimed where the final submitted Program.cs and Docker configuration include it; otherwise it remains a deployment consideration to verify before relying on hosted Google sign-in.

9. User Interface and Usability

9.1 Design Language

UniManage uses a Bootstrap-based interface with limited custom styling. The design is kept clear rather than decorative. Headings use the framework's heading classes, primary actions use solid buttons, secondary actions use outline buttons, and destructive actions use the btn-danger style.

The interface avoids decorative effects that distract from the academic tasks. Tables are compact, dashboard summaries use cards, and alerts show feedback after form submissions. This gives the application a simple, professional appearance suitable for a university course management system.

9.2 Layout Structure

The layout structure is defined in _Layout.cshtml. The page is composed of the following parts:

A top navigation bar with the brand link, role-aware items, profile menu, and a sign-in or sign-out action.

A main content area where each Razor view contributes its content.
A footer with the project name and the academic context.
A skip-to-main-content link near the top of the page to support keyboard navigation.
The _Alerts.cshtml partial is included in the layout so that any TempData-based feedback message is shown automatically after a form submission. The _DashboardCalendar.cshtml partial renders a calendar block when included by a dashboard view.

9.3 Role-Aware Navigation

The navigation bar adapts to the signed-in user's role. A Student sees links for the Student dashboard, course browsing, assignments, submissions, messages, and meetings. A Lecturer sees links for the Lecturer dashboard, owned courses, assignments, submissions, materials, messages, and meetings. An Administrator sees links for the Administrator dashboard, the course catalogue, user management, reports, audit logs, and meetings.

The role-aware navigation is implemented using Razor conditions inside the layout. The view checks User.IsInRole(AppRoles.X) before rendering each role-specific link. Anonymous users see only the home, login, and register links. This pattern reduces navigation noise and ensures that no link points to a forbidden action.

9.4 Forms and Validation Feedback

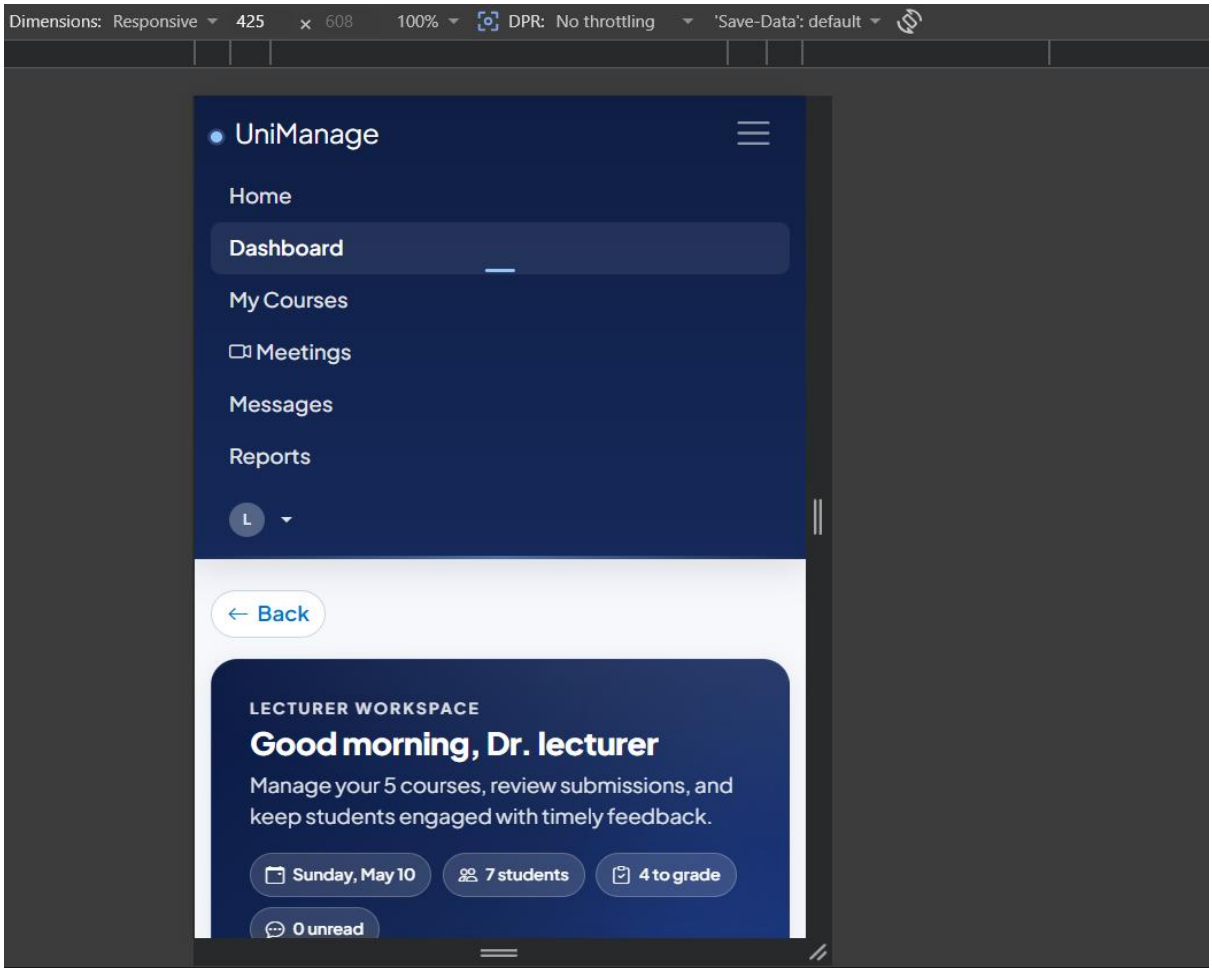
Forms in UniManage use the standard ASP.NET Core MVC form pattern. Each form binds to a view model rather than to an entity. The form fields use Tag Helpers such as asp-for, asp-validation-for, and asp-validation-summary. The view models declare validation rules through data annotations such as [Required], [StringLength], [Range], and [EmailAddress]. When the form is posted with invalid data, the controller returns the same view with the populated model and the validation errors are displayed next to the offending field.

The _ValidationScriptsPartial.cshtml partial loads the client-side jQuery validation scripts for client-side validation. Server-side validation always runs in addition to client-side validation, so a malicious client cannot bypass the rules.

9.5 Responsive Design

The Bootstrap grid provides responsive layout support. Forms, tables, and dashboards adjust their column widths according to the viewport. The author has avoided fixed pixel widths in custom CSS so that the layout remains usable on smaller screens. The navigation bar uses the framework's collapse component to convert into a hamburger menu on narrow viewports.

Figure 17: Responsive Design Evidence



This screenshot shows a dashboard at a narrower viewport as evidence of responsive design.

9.6 Accessibility Considerations

The author has applied several accessibility considerations:

A skip link near the top of the layout allows keyboard users to bypass the navigation and jump to the main content.

Form labels are associated with their input fields through Tag Helpers, which improves screen-reader behaviour.

Headings follow a logical structure (h1 for the page title, h2 for major sections).

Colour is not used as the only way to communicate state; alerts use both colour and icon or text.

Buttons have meaningful text rather than relying on icons alone.

A full accessibility audit (for example WCAG 2.1 AA compliance) has not been carried out, so it is recorded as a future improvement in the limitations chapter.

9.7 Error and Confirmation Messages

Error and confirmation messages are shown through three mechanisms:

Inline validation messages next to form fields, generated through Razor Tag Helpers.
A page-level validation summary at the top of forms, generated through asp-validation-summary.
TempData-based alerts shown by the _Alerts.cshtml partial after a redirect.
For deeper errors that are not caused by user input, the application displays the friendly error page Views/Shared/Error.cshtml in production. The friendly page does not reveal stack traces or internal server detail. In development, the developer exception page is shown instead.

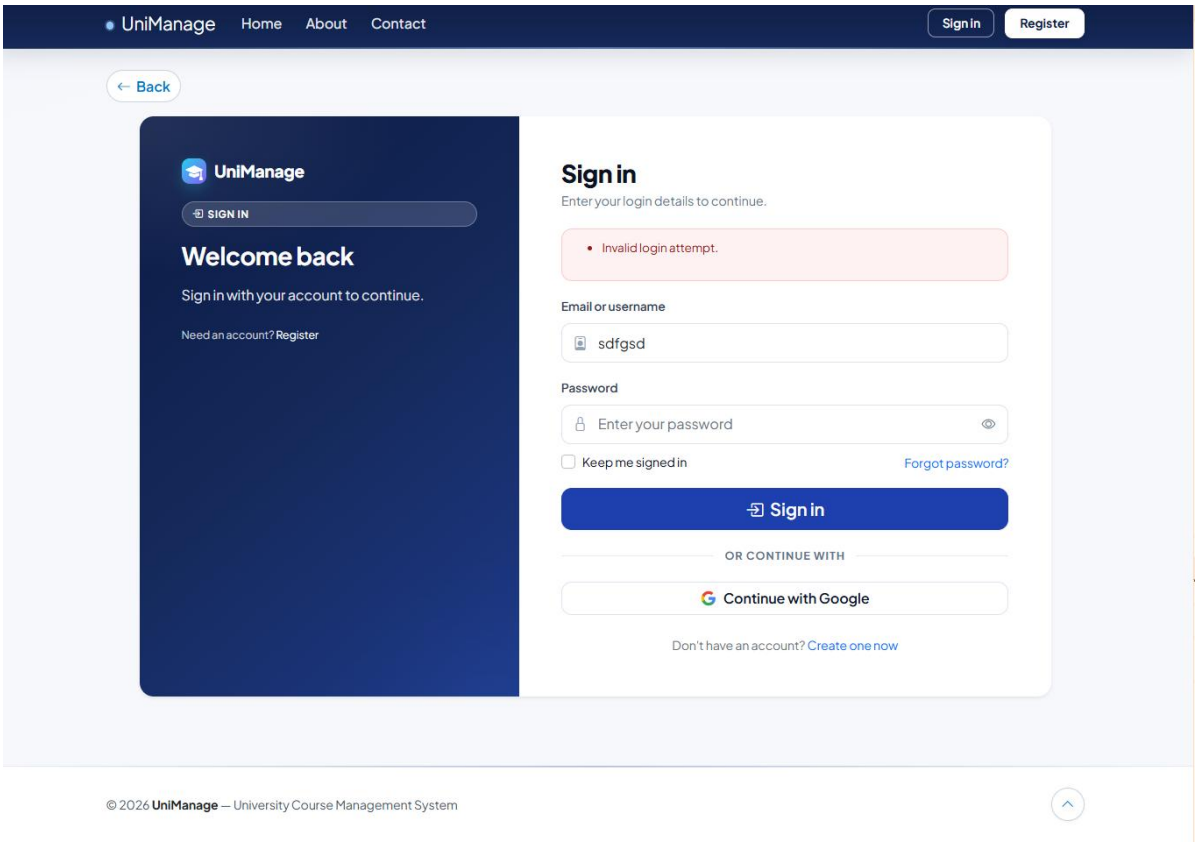


Figure 18: Validation and Error Handling Evidence

This screenshot shows the validation error example.

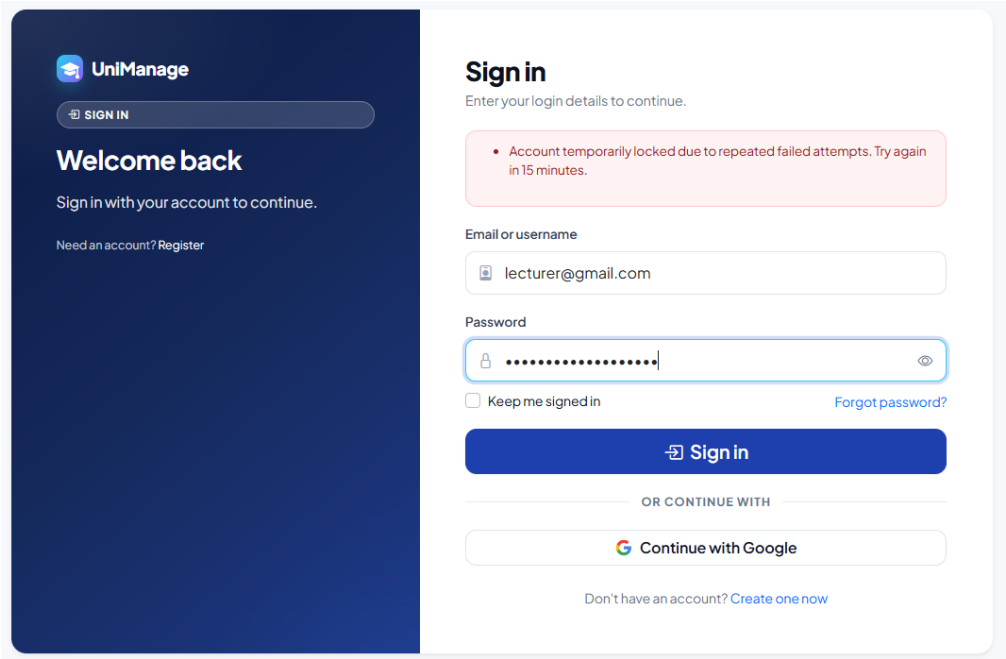


Figure 18b: Temporary Account Lockout Evidence

This screenshot shows the temporary account lockout message after repeated failed login attempts.

9.8 Responsive Layout Check

The responsive check was done using browser mobile viewports. UniManage uses a Bootstrap-based layout, so the login page, navigation, dashboards, forms, reports, and course or assignment pages adjust to smaller screen widths without needing a separate mobile application. This supports the interface design and usability evidence, but it is not presented as a full accessibility audit.

During testing, the author checked whether important controls remained visible, forms were still usable, and navigation did not block the main content on a smaller screen. The responsive screenshots are included as supporting evidence. Any remaining visual issues are treated as future UI improvements rather than core functional failures.

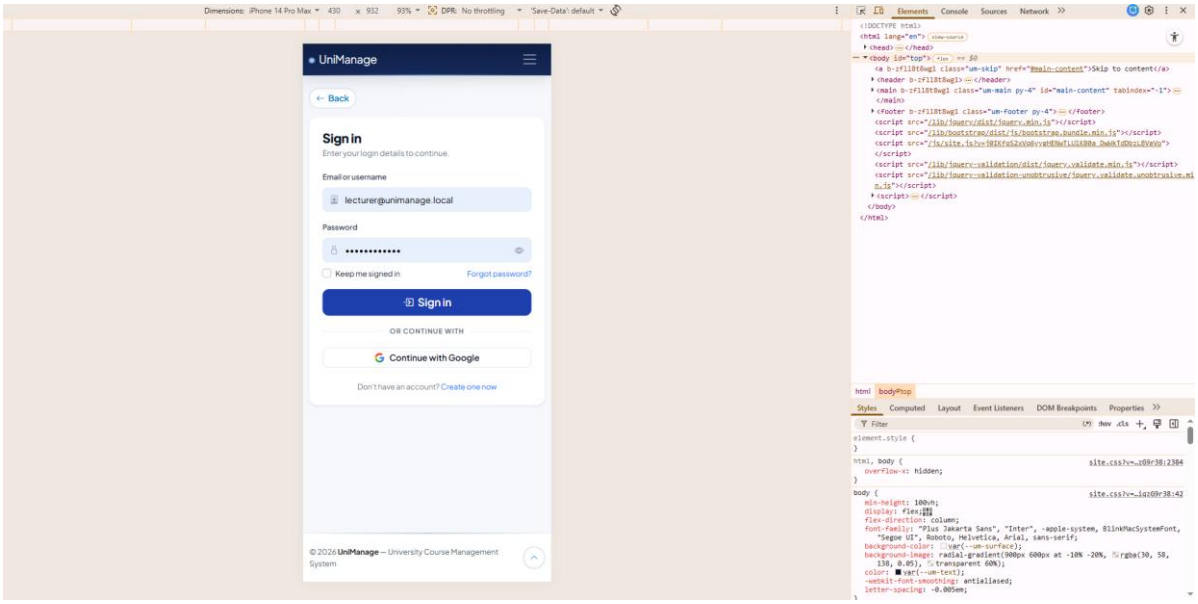


Figure M1: Mobile View of Login Page

This screenshot shows the login page in a mobile browser viewport.

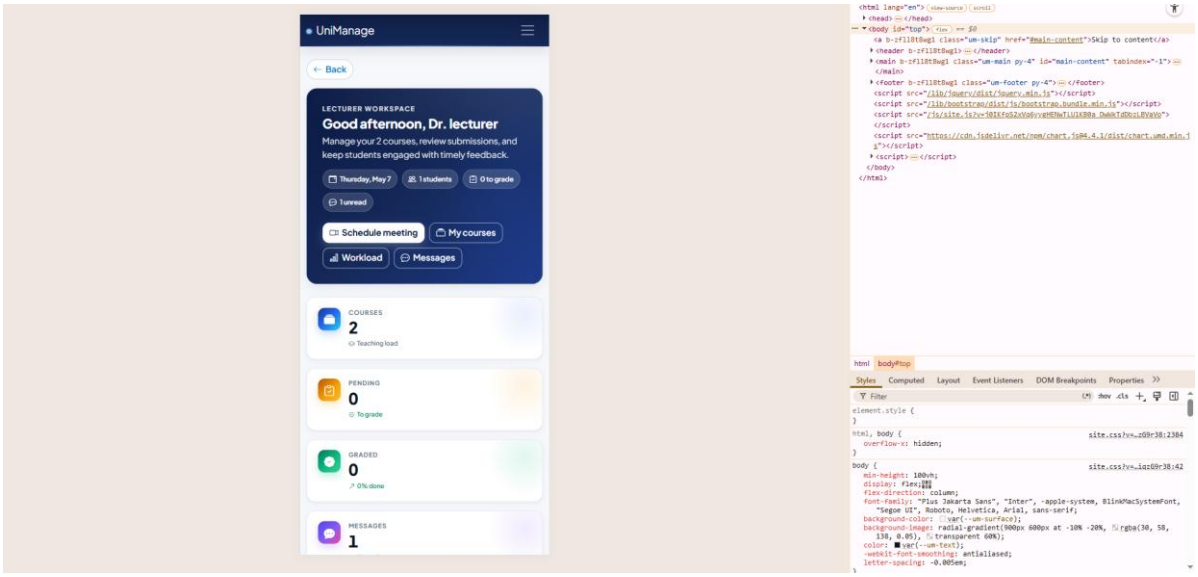


Figure M2: Mobile View of Dashboard

This screenshot shows a dashboard in a mobile browser viewport.

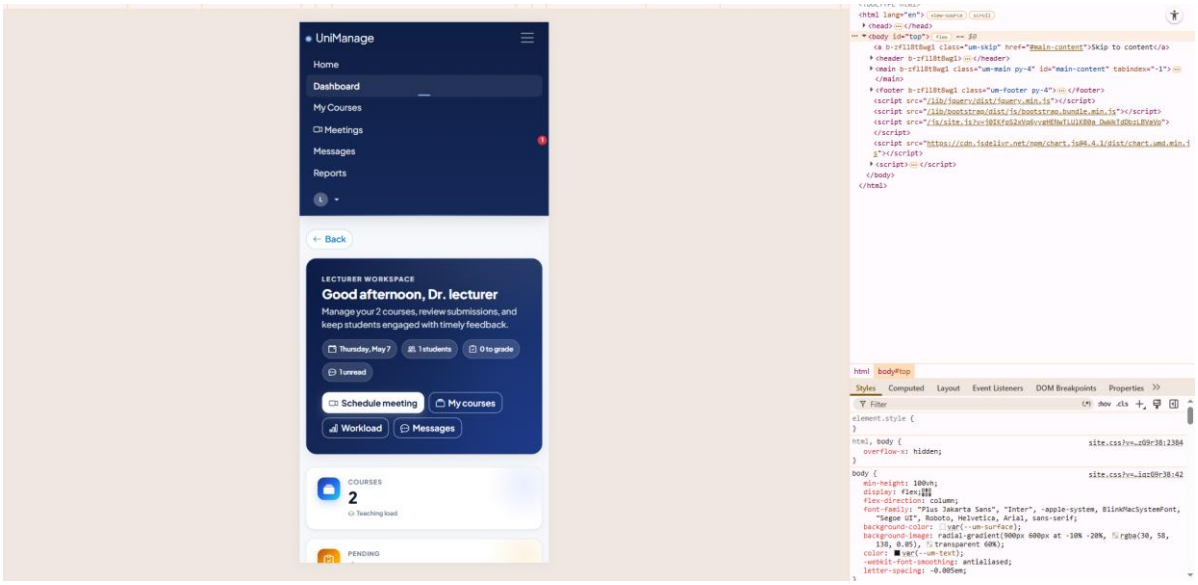


Figure M3: Mobile Responsive Navigation

This screenshot shows the collapsed navigation menu in a mobile browser viewport.

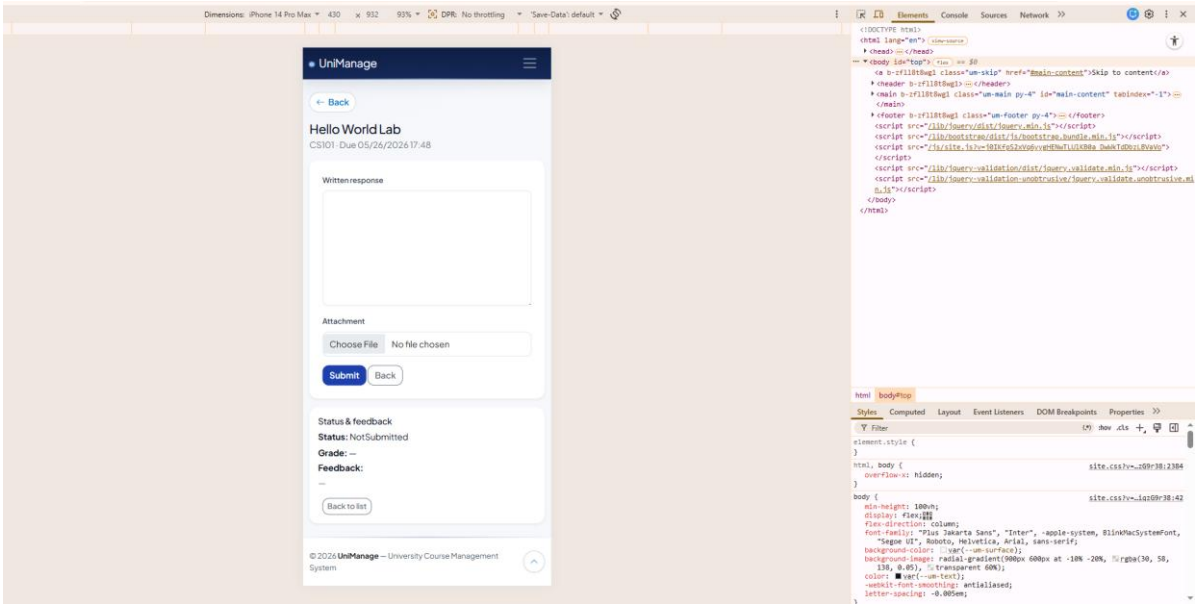


Figure M4: Mobile View of Course or Assignment Form

This screenshot shows an assignment submission form in a mobile browser viewport.

10. Detailed Description of Classes, Properties and Methods

10.1 Main Controllers and Responsibilities

Table 11 summarises the controllers and their responsibilities, key actions, related views, and the coursework area that they serve.

Controller	Main Responsibility	Key Actions / Methods	Related Views	Coursework Area
AccountController	Authentication and account flows	Register, Login, Logout, ForgotPassword, ResetPassword, GoogleCallback	Views/Account/	Registration / Login
DashboardController	Role dashboard routing and dashboard data	Index, Student, Lecturer, Administrator	Views/Dashboard/	Dashboards
AdminUsersController	Administrator user account control	Index, Create, Edit, ToggleLock, Delete	Views/AdminUsers/	Administrator / Security

Controller	Main Responsibility	Key Actions / Methods	Related Views	Coursework Area
CoursesController	Course CRUD, lecturer courses, materials	Index, MyCourses, Details, Create, Edit, Delete, UploadMaterial	Views/Courses/	Course Management
EnrollmentsController	Student course browsing and enrolment	Browse, Enroll	Views/Enrollments/Browse.cshtml	Enrollment
AssignmentsController	Assignment CRUD and listing	ForCourse, Mine, Create, Edit, Delete	Views/Assignments/	Assignment Management
SubmissionsController	Submission, grading, file access	Submit, Grade, ForAssignment, CourseMaterialFile, SubmissionFile	Views/Submissions/	Submission and Grading
ReportsController	Reporting and exports	CoursePopularity, StudentPerformance, LecturerWorkload, Enrollments, PassFail, AssignmentAttendance	Views/Reports/	Reporting
MessagesController	Messaging workflow	Inbox, Thread, Reply, Compose, MarkAllRead	Views/Messages/	Communication
MeetingsController	Meeting scheduling and ICS export	Index, Create, Edit, Delete, Join, Ics	Views/Meetings/	Communication
ProfileController	Profile and password change	Index, ChangePassword	Views/Profile/	Account Management
AuditLogsController	Audit log review	Index	Views/AuditLogs/Index.cshtml	Security Evidence
HomeController	Public landing	Index, Privacy (if present), Error	Views/Home/, Views/Shared/Error.cshtml	Application Foundation

Controller	Main Responsibility	Key Actions / Methods	Related Views	Coursework Area
	and error page			

Table 11: Main Controllers and Responsibilities

10.2 Main Models and Properties

Table 12 summarises the main domain models and their properties.

Model	Key Properties	Purpose	Database Relationship
ApplicationUser	Id, FullName, Email, UserName, PhoneNumber, DateOfBirth, lockout fields (plus inherited Identity fields)	Extends Identity user with profile and account-control fields	Related to Course (lecturer), Enrollment (student), Submission (student/grader), Message (sender/receiver), CourseMaterial (uploader), Meeting (lecturer), AuditLog (user)
Course	CourseId, Code (unique), Name, Description, Credits, EnrollmentLimit, LecturerId, PrerequisiteId	Course record	One lecturer, optional prerequisite, many enrolments, assignments, materials, meetings
Enrollment	EnrollmentId, StudentId, CourseId, EnrolledAtUtc	Student-course link	Many-to-one with student and course; unique on the pair
Assignment	AssignmentId, CourseId, Title, Description, DueDateUtc, MaxPoints	Coursework task	Many assignments per course

Model	Key Properties	Purpose	Database Relationship
Submission	SubmissionId, AssignmentId, StudentId, Status, Grade, Feedback, SubmittedAtUtc, GradedById, GradedAtUtc, file metadata	Student submission and grading record	Many submissions per assignment
SubmissionStatus	Submitted, Graded (enum)	Submission status	Used by Submission
Message	MessageId, SenderId, ReceiverId, Subject, Content, IsRead, SentAtUtc	Communication record	Two foreign keys to ApplicationUser
CourseMaterial	CourseMaterialId, CourseId, Title, StoredFileName, OriginalFileName, UploadedById, UploadedAtUtc	Uploaded teaching material metadata	Many materials per course
Meeting	MeetingId, CourseId, LecturerId, Title, Description, ScheduledAtUtc, DurationMinutes, MeetingUrl	Meeting record	Many meetings per course
AuditLog	AuditLogId, Category, Action, Detail, UserId, Success, CreatedAtUtc	Audit trail	Optional foreign key to ApplicationUser
AppRoles	Administrator, Lecturer, Student (constants)	Role name constants	Used by Identity roles and [Authorize]
AdminUserManagementViewModels	AdminUserRowViewModel, AdminUserCreateViewModel, AdminUserEditViewModel with username, email, role, lock status and validation fields	Administrator user-management forms and list rows	Not persisted directly

Model	Key Properties	Purpose	Database Relationship
ErrorViewModel	RequestId, ShowRequestId	Used by error view	Not persisted

Table 12: Main Models and Properties

10.3 Main Methods and Purpose

Table 13 lists the main methods that drive the application.

Class / File	Method	Purpose	Input / Output	Error Handling / Validation
Program.cs	service configuration	Registers DbContext, Identity, email, audit, rate limiting, anti-forgery and MVC	App configuration to service container	Throws if connection string missing
Program.cs	middleware configuration	Configures HTTPS, security headers, routing, rate limiter, authentication, authorisation	Request pipeline	Production exception handler vs developer exception page
DbInitializer	SeedAsync	Applies migrations and creates demo roles, users, and sample data	Service provider to seeded database	Throws on failed user creation
AccountController	Register	Creates Student or Lecturer user	Register form to signed-in user	Role check, password validation, audit logging
AccountController	Login	Authenticates user with lockout	Login form to dashboard redirect	ModelState, lockout, audit failure

Class / File	Method	Purpose	Input / Output	Error Handling / Validation
AccountController	ForgotPassword	Sends password reset email	Email address to email send	Token generation, generic response to avoid user enumeration
AccountController	ResetPassword	Resets password using token	Token and new password to updated user	Token validation, password rules
AdminUsersController	Index	Lists users for administrator review	Search and paging to user list	Administrator-only role attribute
AdminUsersController	Create / Edit	Creates or updates user account details and roles	User form to Identity user	Role list, username format, duplicate username/email, date validation
AdminUsersController	ToggleLock	Locks or unlocks a user account	User ID to lock state change	Blocks self-lock and unsafe administrator lock
AdminUsersController	Delete	Deletes a non-administrator user account	User ID to Identity delete	Blocks self-delete and administrator delete
DashboardController	Index	Routes by role	Role claim to role action	Returns 403 if unauthorised
DashboardController	Student / Lecturer / Administrator	Builds role-specific dashboard view models	EF Core queries to Razor view	Role attribute
CoursesController	Create	Creates course	Course form to	Lecturer and prerequisite

Class / File	Method	Purpose	Input / Output	Error Handling / Validation
			database record	validation, unique code
CoursesController	Edit	Updates course	Course form to database record	Concurrency and validation
CoursesController	UploadMaterial	Uploads course material	File and title to stored metadata	File type, size, owner, rate limit
EnrollmentsController	Browse	Lists eligible courses	Student claim to course list	Role attribute
EnrollmentsController	Enroll	Enrols student on course	Course ID to enrolment record	Duplicate, capacity, prerequisite checks, audit
AssignmentsController	Create	Creates assignment	Assignment form to database record	Course ownership and model validation
AssignmentsController	Edit / Delete	Updates or removes assignment	Assignment ID and form	Course ownership
SubmissionsController	Submit	Saves student submission	Text/file to submission record	Enrolment, file validation, graded lock
SubmissionsController	Grade	Records grade and feedback	Grade form to updated submission	Lecturer ownership and score range
SubmissionsController	CourseMaterialFile / SubmissionFile	Streams a stored file	File ID to file content	Role and ownership
ReportsController	report actions	Builds analytics rows and exports	Query output to view, CSV, or PDF	Role attribute
MessagesController	Compose	Sends message	Recipient, subject, content	Allowed-recipient validation

Class / File	Method	Purpose	Input / Output	Error Handling / Validation
MessagesController	Inbox / Thread / Reply	Lists or extends a conversation	View models to Razor views	Role attribute
MeetingsController	Create / Edit / Delete	Manages meetings	Meeting form to meeting record	Course ownership, URL and time validation
MeetingsController	Join / Ics	Joins meeting or downloads ICS	Meeting ID to redirect or file	Role and ownership
ProfileController	Index / ChangePassword	Profile and password update	Profile form, password form	Authentication and validation
AuditLogsController	Index	Lists audit log entries	View model to Razor view	Administrator role
AuditLogger	LogAsync	Saves audit record	Category, action, detail, user, success	Internal try/catch logging
UploadedFileStore	SaveAsync	Validates and stores uploaded file	File stream and metadata	File type, size, naming
CsvWriter	Build	Writes CSV content	Rows to byte array	Escapes commas, quotes, newlines
PdfReport	Build	Builds PDF using QuestPDF	Data to byte array	Layout exceptions handled by middleware
SmtpEmailService	SendAsync	Sends email using SMTP	Subject, body, recipient	Connection and auth errors
SecurityHeadersMiddleware	InvokeAsync	Adds security headers to every response	HttpContext to response headers	Headers added before next middleware

Table 13: Main Methods and Purpose

10.4 Controller Layer Explanation

The controllers in UniManage are used to handle the main user actions, such as logging in, managing courses, submitting assignments, grading work, sending messages, and viewing reports. Most controller actions first check whether the submitted form data is valid. After that, the action checks the user role or ownership where needed, reads or updates the database through ApplicationDbContext, and then returns the correct view or redirects the user to the next page.

Role protection is applied using the [Authorize] attribute. For example, administrator-only, lecturer-only, and student-only actions are restricted so that users cannot access pages outside their role. Where extra user information is needed, the controllers use ASP.NET Core Identity services such as UserManager and SignInManager. Supporting services such as audit logging, email handling, and file storage are used where the feature requires them.

Author kept repeated logic out of the controller actions where possible. For example, common tasks such as audit logging, file storage, email generation, CSV export, PDF report creation, and meeting link generation are handled through separate infrastructure classes. This made the controllers easier to follow and helped keep the MVC structure clearer.

10.5 Model and ViewModel Layer Explanation

Models are plain C# classes with primary key, foreign key, and field properties. They use data annotations for length, range, and required validation. They are mapped to database tables through ApplicationDbContext and the EF Core conventions.

View models are also plain C# classes, but they are not persisted. They carry data between controllers and views. The author has used view models for every form so that user input cannot bind directly to entity properties that should not be exposed (a defence against over-posting). For example, CourseInputViewModel exposes only the course fields that are supposed to be set during create or edit.

10.6 Infrastructure and Service Layer Explanation

The Infrastructure layer encapsulates services that are not strictly part of the domain model but are needed by multiple controllers. The layer contains:

The audit logger (AuditLogger, IAuditLogger), which saves audit records and is used by every controller that performs a security-sensitive action.

The email service (SmtpEmailService, IEmailService, EmailSettings, EmailTemplates), which is used by the password reset flow.

The file store (UploadedFileStore), which is used by submissions and course materials.
The CSV writer (CsvWriter), which is used by the reports controller.
The PDF report builder (PdfReport), which is used by the reports controller.
The calendar link helper (CalendarLink) and the meeting link generator (MeetLinkGenerator), which are used by the meetings controller.

The custom security headers middleware (SecurityHeadersMiddleware), which is registered in Program.cs and runs for every response.

Each service uses dependency injection in the constructor, which makes the service replaceable in tests or in different deployments.

11. Logical Solution Explanation

11.1 Overall Request-to-Response Flow

The end-to-end request flow can be summarised as follows:

The browser sends a request to the application.
Kestrel accepts the connection and passes it to the middleware pipeline.
HTTPS redirection ensures the request is on the secure scheme.
The static file middleware serves any request that targets a file in wwwroot.
The custom security headers middleware adds standard headers to the response.
Routing matches the request to a controller and action.
Rate limiting checks the policy attached to the action and either allows the request or returns 429.

Authentication identifies the user from the cookie.

Authorisation checks the [Authorize] attribute and the role.

The controller action runs. It validates the model, performs business rules, queries or updates EF Core, and writes audit log entries.

EF Core translates the LINQ queries into MySQL SQL and executes them through the Pomelo provider.

The action returns either a redirect or a view.
The Razor engine renders the view using the view model.
The response flows back through the pipeline to the browser.
The browser renders the HTML and any small amount of JavaScript runs.
This flow is the same for every feature. The differences between features are the validation rules, the EF Core queries, and the resulting view. Keeping the flow consistent is one of the strengths of MVC and makes the project easier to maintain and to mark.

11.2 Registration and Login Logic

The registration logic accepts a RegisterViewModel, validates it through ModelState, builds an ApplicationUser with the provided full name, email, and phone number, and calls UserManager.CreateAsync. If the operation succeeds, the user is added to the chosen role (Student or Lecturer) through UserManager.AddToRoleAsync, signed in through SignInManager.SignInAsync, and recorded in the audit log. If the operation fails, the validation errors from IdentityResult are added to ModelState and the form is returned with the errors.

The login logic accepts a LoginViewModel, validates it, and calls SignInManager.PasswordSignInAsync with lockoutOnFailure: true. The result indicates success, lockout, requires-two-factor, or failure. The action handles each case explicitly, writes an audit log entry, and either redirects to Dashboard/Index or returns the form with an error.

11.3 Dashboard Logic

DashboardController.Index inspects the role claim of the signed-in user and forwards to the role-appropriate action. Each role action runs a series of EF Core queries to fill the role-specific view model and returns the matching view. The author has consciously kept role-specific business rules out of Index so that the role mapping remains a single, easy-to-read switch.

11.4 Course Management Logic

The course management logic for an Administrator follows the standard CRUD pattern. Index lists courses, optionally filtered by a search term and paged. Create shows the form (GET) and saves the course (POST), enforcing the unique course code constraint. Edit and Delete follow the same pattern.

For a Lecturer, MyCourses filters the courses by LecturerId == userId, and UploadMaterial enforces ownership before saving the uploaded file.

11.5 Enrollment Validation Logic

The enrolment workflow contains several rules that demonstrate validation working together:

The course must exist.
The student must not already be enrolled on the course.
The course must have spare capacity, computed as EnrollmentLimit - count(existing enrolments).

The course's prerequisite, if any, must be satisfied by the student.

The student must not have a soft business reason to be blocked, for example a temporarily disabled account.

If all rules pass, the action creates the Enrollment record, writes an audit log entry, and redirects to a confirmation page. If any rule fails, the action returns the browse view with a friendly error message. The combination of unique index (database) and rule check (application) provides defence in depth.

11.6 Assignment, Submission and Grading Logic

The assignment, submission, and grading workflow connects three controllers. AssignmentsController.Create lets a Lecturer set an assignment for an owned course. SubmissionsController.Submit lets a Student upload text and a file for an assignment they are enrolled on. SubmissionsController.Grade lets a Lecturer record a grade and feedback for a submission on an owned course.

The grading workflow is the most rule-heavy. The action verifies that the submission belongs to an assignment whose course is owned by the lecturer, displays the submission, accepts a grade and feedback through GradeSubmissionViewModel, validates the grade range, updates the Submission entity, sets the grader and grading time, and writes an audit log entry. After grading, the submission is locked so that the student can no longer modify it.

11.7 Reporting Logic

The reports controller runs an EF Core query for each report. Most queries use GroupBy and aggregate functions, and they project into a report view model. The view model is rendered through Razor for HTML output, written through CsvWriter for CSV output, or rendered through PdfReport for PDF output. The reports are gated by the Administrator role.

11.8 Communication Logic

Messages and meetings represent the communication features. The messaging logic enforces an allowed-recipient rule: a student can message a lecturer who teaches a course on which the student is enrolled, and vice versa. This rule is implemented as a database join during the recipient look-up, so the front end never offers an unauthorised recipient.

The meeting logic restricts scheduling and editing to the lecturer who owns the course. Students can see meetings that belong to courses on which they are enrolled. The ICS export is provided for users who want to import the meeting into a personal calendar.

11.9 Security and Validation Logic

Security and validation logic runs across every feature. Each form posts an anti-forgery token; each input is validated through data annotations and ModelState; each action is gated by [Authorize] and ownership checks; each controlled file download checks the requester's role

and ownership before streaming the file; each security-sensitive action writes an audit log entry. The combination of these controls is described in Chapter 13.

12. Validation, Exception Handling and Error Management

12.1 Data Annotation Validation

Data annotation validation is the first line of defence. Each view model declares attributes such as [Required], [StringLength], [Range], [EmailAddress], [Compare], and [DataType]. The MVC framework runs these attributes during model binding and populates ModelState. Table 14 shows representative validation rules for selected forms.

Form / View Model	Selected Validation Rules
RegisterViewModel	[Required] on full name, email, password; [EmailAddress] on email; [StringLength] on full name; password requirements enforced by Identity (8 chars, digit, upper, lower)
LoginViewModel	[Required] on login identifier and password; login identifier accepts email or username
ForgotPasswordViewModel	[Required], [EmailAddress] on email
ResetPasswordViewModel	[Required] on token, password, confirm password; [Compare] on confirm password
CourseInputViewModel	[Required] on code and name; [StringLength] on code, name, description; [Range] on credits and enrolment limit
AssignmentInputViewModel	[Required] on title, due date; [StringLength] on title and description; [Range] on max points
SubmissionSubmitViewModel	[StringLength] on text content; file validation in controller
GradeSubmissionViewModel	[Range] on grade; [StringLength] on feedback
MessageComposeViewModel	[Required] on subject, content, recipient; allowed-recipient check in controller
MeetingInputViewModel	[Required] on title, scheduled time; [Url] on meeting URL where present; [Range] on duration
MaterialUploadViewModel	[Required] on title; file validation in controller
ProfileViewModel	[Required] on full name and email; [EmailAddress] on email; [Phone] on phone number where present

Table 14: Validation Rules by Form

12.2 ModelState Handling

Every POST action that accepts user input begins with a ModelState.IsValid check. If the model state is invalid, the action returns the same view with the populated view model so that the user can see and correct the validation errors. The author has avoided silent failures and has avoided redirecting to a generic error page when the user can recover by correcting the form.

12.3 Server-Side Ownership Checks

Some validation rules cannot be expressed as data annotations. For example, "a Lecturer can only edit assignments for courses they own" requires a database lookup. These checks are implemented inside the action after the ModelState check. If the check fails, the action returns either a 403 Forbidden response or a redirect to the appropriate index view with a TempData warning. The combination of role attributes and ownership checks prevents both unauthorised role access and unauthorised data access.

12.4 File Upload Validation

UploadedFileStore validates the uploaded file before storing it. The validation includes:

File size: rejects files larger than the configured maximum.
File extension: rejects files whose extension is not in the allowed list.
File name: stores the file under a generated stored file name to avoid collisions and to prevent path traversal attacks.

Content type: optional check against the allowed MIME types.
If the file fails validation, the action adds a model error and returns the form with an explanation. The rate-limiting policy upload further restricts the number of uploads per minute per user to prevent abuse.

12.5 Anti-Forgery Protection

Anti-forgery is configured globally in Program.cs through AddAntiforgery. Every Razor form generated through Tag Helpers includes a hidden anti-forgery token. POST actions that accept form data are decorated with [ValidateAntiForgeryToken]. The combination prevents cross-site request forgery attacks.

The anti-forgery cookie is configured with HttpOnly = true, SameSite = Strict, and SecurePolicy = SameAsRequest, which provides additional defence against cookie theft.

12.6 Exception Handling Middleware

In production, Program.cs configures app.UseExceptionHandler("/Home/Error") so that any unhandled exception is caught, logged, and replaced with the friendly error view. The view does not include any stack trace, source file, or server detail. In development, app.UseDeveloperExceptionPage() provides a detailed exception page that helps the author find the source of the problem.

The friendly error view uses ErrorViewModel to optionally show a request identifier, which can be cross-referenced with the application logs.

12.7 Friendly Error Pages

Apart from the global exception handler, individual actions use the standard MVC return types BadRequest, NotFound, and Forbid to indicate specific error conditions. The framework renders these as the appropriate status code page, which can be customised through app.UseStatusCodePagesWithReExecute if required. The author has retained the standard behaviour and instead emphasised friendly TempData messages on success and on most foreseeable failures.

12.8 Audit and Error Evidence

Each security-sensitive failure is recorded in the audit log with success = false. This means that a marker reviewing the audit log can see not only successful actions but also blocked actions, such as failed sign-ins, rejected enrolments, and unauthorised access attempts. The audit log therefore acts as a forensic record as well as a behavioural record.

This evidence is inserted in Section 9.7.

13. Security and Data Protection

Security is a cross-cutting concern in UniManage. The system is designed with defence in depth: each request passes through a chain of controls so that a failure of one control does not lead to a compromise.

13.1 Authentication

Authentication is provided by ASP.NET Core Identity. The ApplicationUser class extends IdentityUser and adds profile fields such as FullName and PhoneNumber. The Identity options enforce a minimum password length of eight characters, require a digit, require an uppercase letter, and require a lowercase letter. A non-alphanumeric character is not strictly required, which keeps the password rule consistent with the seeded credentials.

Account lockout is enabled. After five failed attempts, the account is locked for fifteen minutes. The lockout policy reduces the effectiveness of brute-force attacks. The login form is also protected by the auth rate-limiting policy, which allows ten requests per IP address per minute. Combined, the lockout and rate-limiting policies make brute-force attacks impractical against a single account or against the system as a whole.

Optional Google login is supported. Where the Authentication:Google:ClientId and Authentication:Google:ClientSecret configuration values are present, the Google authentication handler is registered in Program.cs, including a OnRemoteFailure handler that redirects to the login page with the failure reason. In deployment, these values are supplied through environment variables rather than being written into the report. Google login is not required for marking, and the demonstration users use plain Identity credentials.

13.2 Authorisation

Authorisation is enforced through [Authorize] attributes on controllers and actions, with Roles set to AppRoles.Administrator, AppRoles.Lecturer, or AppRoles.Student as appropriate. Where multiple roles are allowed, the attribute lists them separated by commas. The base controller class is decorated with [Authorize] so that anonymous users cannot reach any action by default; only the home page and the account flows are explicitly anonymous.

Ownership checks are added inside actions where role membership is not enough. For example, a Lecturer must only see their own courses, materials, assignments, submissions, and meetings.

The ownership check is implemented as a database filter (LecturerId == userId) in the EF Core query, so the resulting view never contains rows that the user is not authorised to see.

13.3 Password Handling

Passwords are never stored in plain text. ASP.NET Core Identity uses PBKDF2 with HMAC-SHA-512 by default, with a per-user salt and configurable iteration count. Password reset uses Identity's tokenised flow. The password reset token is generated by UserManager.GeneratePasswordResetTokenAsync, sent through the email service, and validated by UserManager.ResetPasswordAsync. The reset endpoint is rate limited under the auth policy.

A future version could add "have I been pwned" checks or pre-generated weak password lists. The current password rules are suitable for the academic deployment.

13.4 Role-Based Access Control

Role-based access control is the third major control. Three roles are seeded by DbInitializer: Administrator, Lecturer, Student. Each role is assigned to the corresponding demonstration user. New registrants choose either Student or Lecturer. The Administrator role is not available through self-registration; only the seeded administrator can use it. This is consistent with the principle that high-privilege roles should not be granted by self-service.

Admin User Management strengthens account control for the Administrator role. The feature allows user review, create/edit, lock/unlock, and controlled deletion of non-administrator users. The controller blocks self-lock, self-delete, administrator deletion, and unsafe changes that would leave the system without administrator access. These controls support the Security and Data Protection requirement in a practical coursework scope.

13.5 Protection Against Common Web Risks

Table 15 maps the security controls in UniManage to common web risks.

Risk	Control in UniManage	Source-Code Evidence
Injection (SQL, command)	EF Core parameterised queries	All controller queries through DbSet and LINQ
Broken authentication	Identity with strong password rules, lockout, secure cookies	Program.cs Identity options, ConfigureApplicationCookie
Sensitive data exposure	HTTPS redirection, HSTS, secure cookie policy, redacted screenshots	Program.cs, Installation Guide and User Manual
Broken access control	[Authorize] attributes, role checks, ownership filters	All controllers
Security misconfiguration	Security headers, HSTS, secure cookie policy	Infrastructure/SecurityHeadersMiddleware.cs, Program.cs
Cross-site scripting (XSS)	Razor automatic HTML encoding, content-type checks	All views, model binding
Cross-site request forgery (CSRF)	Anti-forgery cookie and token	Program.cs, [ValidateAntiForgeryToken] attributes
Insecure deserialization	Strongly typed view models, no untrusted serialiser	View models throughout
Components with known vulnerabilities	Pinned package versions, latest patch level when developed	AD COURSEWORK 2.csproj
Insufficient logging and monitoring	Audit logger with IAuditLogger and AuditLog entity	Infrastructure/AuditLogger.cs, Models/AuditLog.cs
Brute-force attacks	Account lockout, rate limiter on auth policy	Program.cs lockout, AddRateLimiter
Path traversal in file storage	UploadedFileStore generates stored file names	Infrastructure/UploadedFileStore.cs

Risk	Control in UniManage	Source-Code Evidence
Excess upload load	Rate limiter on upload policy	Program.cs upload policy

Table 15: Security Controls Mapped to Common Web Risks

13.6 File Security

File security combines validation, controlled storage, and controlled retrieval. Files are validated by UploadedFileStore and stored under a generated file name in a controlled directory. Files are retrieved through controller actions (CourseMaterialFile, SubmissionFile) that check role and ownership before streaming the file. Direct URL access into the upload directory is not exposed to the browser, which prevents direct file enumeration.

13.7 Audit Logging and Traceability

Audit logging is provided through IAuditLogger and AuditLogger. The audit log records the category of the event, the action, a short human-readable detail, the user identifier (where available), and a success flag. Categories include Account, Course, Enrollment, Assignment, Submission, Material, Message, Meeting, Report, and Security. Table 16 shows representative categories and actions.

Category	Actions Recorded
Account	Register, Login, LoginFailed, Logout, PasswordReset
Course	Create, Update, Delete
Enrollment	Enroll, EnrollFailed
Assignment	Create, Update, Delete
Submission	Submit, Grade
Material	Upload, Download
Message	Send, Reply
Meeting	Create, Update, Delete, Join
Report	Generate, Export
Security	AccessDenied, RateLimited, AntiForgeryFailure (where reported)

Table 16: Audit Logger Categories

Audit log entries are reviewed through AuditLogsController.Index, which is restricted to Administrators.

13.8 Data Protection Considerations

The author has applied several data protection considerations:

Sensitive configuration (database password, SMTP password, Google client secret) is supplied through local configuration or environment variables and is excluded from screenshots and the report.

Personal data shown in screenshots is excluded or redacted in the submitted report.

The audit log records identifiers and actions but does not record passwords or sensitive payloads.

HTTPS and HSTS protect data in transit.

Cookies are HTTP-only and SameSite protected.

The Docker workflow uses environment-variable mapping for database, Google OAuth, and SMTP settings. deployment.env.example remains a template only and must not contain real secrets.

The retention of audit log data is not formally documented; this is recorded as a limitation in the limitations chapter.

14. Programming Style and Maintainability

Table 17 summarises the main programming style practices applied in the project.

Practice	Description	Evidence
Naming conventions	PascalCase for classes, methods, and properties; camelCase for parameters and local variables.	All controllers, models, services
MVC folder structure	Controllers, Models, ViewModels, Views, Data, Infrastructure	Project root
Single responsibility	Each service and controller addresses one area	Infrastructure folder
Dependency injection	Services are injected through constructors	All controllers
Asynchronous programming	EF Core, Identity, file, and email operations use async/await	All controllers and services

Practice	Description	Evidence
Strongly typed view models	Forms bind to view models, not entities	ViewModels folder
Routing conventions	Default {controller}/{action}/{id?} pattern	Program.cs
Configuration through ASP.NET Core configuration	Connection strings, Google OAuth values, and email settings can come from local configuration or environment variables	Program.cs, docker-compose.yml, deployment.env.example
Migrations under source control	Data/Migrations is committed	Git history
Centralised security headers	Custom middleware sets headers consistently	Infrastructure/SecurityHeadersMiddleware.cs
Centralised audit logging	Single IAuditLogger service	Infrastructure/AuditLogger.cs
Build verification	dotnet build --no-restore returns 0 errors and 0 warnings	Build evidence

Table 17: Programming Style Practices

14.1 Naming and Readability

Names in the project are descriptive. Controller names end in Controller. Model names match the academic concept (Course, Enrollment, Submission). View models have a ViewModel suffix. Methods are named after the action they perform (Submit, Grade, Enroll). Local variables are named after their purpose. Acronyms such as Url, Id, and Pdf follow the .NET naming style.

14.2 MVC Separation of Concerns

Separation of concerns is enforced through the folder structure and through the use of dependency injection. Models contain data and validation rules. Controllers contain request flow and business rules. Views contain presentation. Services contain cross-cutting concerns. The data access layer is encapsulated behind ApplicationDbContext. This makes it possible to update one layer without touching the others.

14.3 View Models and Entities

View models and entities are kept separate. View models hold only the fields that the form expects. Entities hold the fields that are persisted. This protects against over-posting attacks where a malicious user adds extra fields to a form to update properties they should not be able to set.

14.4 Dependency Injection

All services are registered through Program.cs and consumed through constructor injection. The author has not used a service locator pattern. Lifetimes are chosen carefully: scoped for ApplicationDbContext and most services, singleton for stateless utility classes where used, and transient for short-lived services where necessary.

14.5 Asynchronous Programming

EF Core operations, Identity calls, file operations, and email calls are asynchronous. The asynchronous pattern keeps the application responsive and reduces thread pool starvation. Controller actions are also asynchronous and use await rather than .Result or .Wait(), which prevents deadlocks.

14.6 Comments and Code Clarity

The author has used comments sparingly. Self-documenting names, small methods, and conventional structure reduce the need for comments. Where a comment is added, it explains intent or trade-offs rather than restating the code. The author has avoided narrating obvious operations (such as "increment the counter") and has not added emojis or decorative comments.

14.7 Maintainability Improvements

The current codebase is maintainable, but there is always room for improvement. The reflection chapter (Chapter 19) and the future improvements chapter (Chapter 21) include items such as automated tests, continuous integration, structured logging, and caching of read-heavy queries. These items would further improve maintainability without requiring a full rewrite.

15. Testing

15.1 Testing Strategy

Testing was carried out in a practical coursework-focused way. The strategy combines manual functional testing, role-based testing, security checks, build verification, and a small set of selected unit tests. Full automated integration and end to end UI suites are not included; that scope remains a limitation and is noted in Chapter 18. Manual testing was used because the marking criteria require visible feature evidence and screenshots.

Selected xUnit test evidence is documented in Section 15.10 as supporting evidence for isolated logic such as role constants, view-model data annotations, grading bounds, CSV export bytes, calendar link generation, password-reset email HTML, and security response headers. This evidence does not replace manual functional testing, the test plan in Section 15.9, or the Visual Studio demonstration expected by the brief. The final submission should include the test source files or a current dotnet test screenshot before automated testing is treated as fully evidenced.

The strategy is simple: test each important feature with a normal case and at least one rejection case, record the result through screenshots or audit evidence where possible, and link each test back to a requirement and marking item.

15.2 Test Environment

The test environment is the author's local machine running Windows, Visual Studio 2022, .NET 8 SDK, MySQL 8, and a modern browser. The application is started with dotnet run or through Visual Studio. The seeded demonstration users are used for sign-in. The browser developer tools are used to verify HTTPS, cookie attributes, and request/response details where relevant.

15.3 Functional Testing

Functional testing covers the path through every major feature. Each functional test starts from a known state (for example a clean database after seeding) and ends with a verified outcome. The test plan in Section 15.9 lists the individual cases.

15.4 Validation Testing

Validation testing covers data annotations, ModelState behaviour, ownership checks, and file validation. Each form is tested with valid input and with input that violates one or more rules. The expected result is a friendly validation message and no database change.

15.5 Role-Based Access Testing

Role-based access testing checks whether each role can reach the correct pages and whether restricted pages stay protected. For example, a Student can be signed in and then directed to /Courses/Create; the expected result is access denied. Similar checks are repeated for Lecturer and Administrator routes.

15.6 Database Testing

Database testing confirms that records are created, updated, and deleted correctly. The verification is done by signing in to MySQL with a database tool and inspecting the table contents after each action. The unique constraints and foreign keys are also tested by attempting to violate them through the UI.

15.7 UI and Usability Testing

UI and usability testing covers navigation, layout, alerts, and responsive design. The expectation is that each page is reachable through the role-aware menu, that each form provides appropriate feedback, and that the layout adapts to a narrow viewport.

15.8 Build Verification

Build verification is performed with dotnet build --no-restore from the project root. The build has been verified to succeed with zero warnings and zero errors. The build evidence is available as a console capture. After the registration email timestamp adjustment, the project was rebuilt successfully with zero errors.

15.9 Test Plan and Results

Table 18 lists the test plan with at least twenty-five test cases.

Test ID	Test Area	Test Scenario	Input / Setup	Expected Result	Actual Result	Status
T1	Registration	Register a new Student	Valid form	User created, signed in, audit log entry written	Manual test case listed; selected evidence attached elsewhere	Manual review
T2	Registration	Register a new Lecturer	Valid form	User created, signed in	Manual test case listed; selected evidence attached elsewhere	Manual review
T3	Registration validation	Submit register form with weak password	Password "abc"	Validation message, no user created	Manual test case listed; selected evidence attached elsewhere	Manual review
T4	Login	Log in as seeded Student	Demo credentials	Redirect to Student dashboard	Manual test case listed; selected evidence attached elsewhere	Manual review
T5	Login lockout	Submit five wrong passwords	Wrong password	Account locked, audit log entry	Manual test case listed; selected evidence attached elsewhere	Manual review
T6	Logout	Sign out	Authenticated session	Redirect to home, cookie cleared	Manual test case listed; selected evidence attached elsewhere	Manual review
T7	Forgot password	Request reset for known email	Valid email	Email sent (or token logged), generic confirmation	Manual test case listed; selected evidence attached elsewhere	Manual review

Test ID	Test Area	Test Scenario	Input / Setup	Expected Result	Actual Result	Status
T8	Student dashboard	Student opens dashboard	Authenticated student	Dashboard shows enrolled courses and deadlines	Manual test case listed; selected evidence attached elsewhere	Manual review
T9	Lecturer dashboard	Lecturer opens dashboard	Authenticated lecturer	Dashboard shows teaching summary and pending submissions	Manual test case listed; selected evidence attached elsewhere	Manual review
T10	Administrator dashboard	Administrator opens dashboard	Authenticated administrator	Dashboard shows counts and popular courses	Manual test case listed; selected evidence attached elsewhere	Manual review
T11	UI navigation	Navigate using role-aware menu	Each role	Each role sees only relevant items	Manual test case listed; selected evidence attached elsewhere	Manual review
T12	Course list	Administrator views course list	Administrator	List displayed with search and paging	Manual test case listed; selected evidence attached elsewhere	Manual review
T13	Course create	Administrator creates a course	Valid form	Course saved, redirect, audit log entry	Manual test case listed; selected evidence attached elsewhere	Manual review
T14	Course code uniqueness	Administrator creates two courses with the same code	Duplicate code	Second create rejected	Manual test case listed; selected evidence attached elsewhere	Manual review

Test ID	Test Area	Test Scenario	Input / Setup	Expected Result	Actual Result	Status
T15	Course delete	Administrator deletes a course	Course id	Course removed, related records handled	Manual test case listed; selected evidence attached elsewhere	Manual review
T16	Enrolment success	Student enrols on an open course	Course with capacity	Enrolment saved, confirmation displayed	Manual test case listed; selected evidence attached elsewhere	Manual review
T17	Enrolment rejection	Student attempts duplicate enrolment	Already enrolled	Rejection message, no new record	Manual test case listed; selected evidence attached elsewhere	Manual review
T18	Assignment create	Lecturer creates an assignment for an owned course	Valid form	Assignment saved	Manual test case listed; selected evidence attached elsewhere	Manual review
T19	Assignment ownership	Lecturer attempts to edit an assignment for another lecturer	Other lecturer's assignment	Forbidden	Manual test case listed; selected evidence attached elsewhere	Manual review
T20	Submission upload	Student submits text and file	Enrolled student	Submission saved, file stored	Manual test case listed; selected evidence attached elsewhere	Manual review
T21	Grading	Lecturer grades a submission with valid grade	Valid grade and feedback	Grade and feedback saved, submission locked	Manual test case listed; selected evidence attached elsewhere	Manual review
T22	Reporting	Administrator opens course popularity report	Administrator	Report rows displayed	Manual test case listed; selected evidence attached elsewhere	Manual review

Test ID	Test Area	Test Scenario	Input / Setup	Expected Result	Actual Result	Status
T23	CSV export	Administrator exports a report as CSV	Administrator	CSV file downloaded	Manual test case listed; selected evidence attached elsewhere	Manual review
T24	Messaging compose	Student messages an allowed lecturer	Valid recipient	Message saved, thread updated	Manual test case listed; selected evidence attached elsewhere	Manual review
T25	Messaging restriction	Student attempts to message an unrelated lecturer	Disallowed recipient	Message rejected	Manual test case listed; selected evidence attached elsewhere	Manual review
T26	Anti-forgery	Submit POST without token using developer tools	No token	Request rejected	Manual test case listed; selected evidence attached elsewhere	Manual review
T27	Rate limit	Submit eleven login requests within a minute	Eleven requests	Eleventh receives 429	Manual test case listed; selected evidence attached elsewhere	Manual review
T28	Access denied	Student opens /Courses/Create	Authenticated student	Forbidden, audit log entry	Manual test case listed; selected evidence attached elsewhere	Manual review
T29	Validation message	Submit invalid course form	Missing required fields	Inline validation messages	Manual test case listed; selected evidence attached elsewhere	Manual review
T30	Friendly error	Trigger an error in production	Production environment	Error.cshtml page displayed	Manual test case listed; selected evidence attached elsewhere	Manual review

Test ID	Test Area	Test Scenario	Input / Setup	Expected Result	Actual Result	Status
T31	Responsive layout	Open dashboard on a narrow viewport	Browser resize	Layout adapts, navigation collapses	Manual test case listed; selected evidence attached elsewhere	Manual review
T32	Build verification	Run dotnet build --no-restore	Project root	Zero warnings and zero errors	Confirmed (build succeeded)	Confirmed
T33	Google OAuth	Open Google sign-in when OAuth values are configured	Configured OAuth client	OAuth challenge starts and returns to the application	Manual test case listed; selected evidence attached elsewhere	Manual review
T34	Admin User Management	Administrator opens user-management list	Administrator account	User list shows username, email, role and lock status	Manual test case listed; selected evidence attached elsewhere	Manual review
T35	Admin lock/unlock	Administrator locks or unlocks a non-self account	Target user account	Status changes and audit entry is written	Manual test case listed; selected evidence attached elsewhere	Manual review
T36	Admin safety rule	Administrator attempts unsafe self-lock or last-admin action	Current administrator account	Action is blocked with a friendly message	Manual test case listed; selected evidence attached elsewhere	Manual review
T37	Mobile viewport	Open login, dashboard, navigation and form in mobile viewport	Browser responsive tools	Controls remain visible and usable	Manual test case listed; selected evidence attached elsewhere	Manual review

Table 18: Test Plan and Results

15.10 Selected Unit Testing Evidence

Selected automated unit tests are treated as supporting testing evidence only. They do not replace manual functional testing, screenshot evidence, local Visual Studio build verification, or the viva demonstration.

The tests focus on role constants, view-model data-annotation validation, and small infrastructure utilities. External dependencies such as live MySQL, SMTP, Google OAuth, and Google Cloud hosting were deliberately excluded from the automated test scope.

The test project is named UniManage.Tests, targets .NET 8, and uses xUnit with FluentAssertions where included in the final package. A shared ValidationTestHelper.ValidateModel wrapper around Validator.TryValidateObject is used so that each test exercises the same data-annotation validation that the controller would see during model binding. The view-model checks cover LoginViewModel, RegisterViewModel, CourseInputViewModel, AssignmentInputViewModel, and MessageComposeViewModel. The utility checks cover CsvWriter, CalendarLink, MeetLinkGenerator, EmailTemplates.BuildPasswordResetEmail, SecurityHeadersMiddleware, and grading bounds. Figure 19 shows the selected unit-test output used as supporting evidence.

Test Area	Example Test	Purpose	Evidence	Status
Role constants	AppRolesTests.AppRoles_ShouldContainExpectedRoleNames	Confirms RBAC role string constants are stable	UniManage.Tests	Source evidence and selected unit-test output included
Login validation	ViewModelValidationTests.LoginViewModel_WithMissingEmailAndPassword_ShouldBeInvalid	Confirms required login fields are enforced	UniManage.Tests	Source evidence and selected unit-test output included
Register validation	ViewModelValidationTests.RegisterViewModel_WithValidInput_ShouldBeValid and the matching negative tests	Confirms registration data annotations	UniManage.Tests	Source evidence and selected unit-test output included
Course validation	ViewModelValidationTests.CourseInputViewModel_WithMissingCodeAndName_ShouldBeInvalid	Confirms course form rules	UniManage.Tests	Source evidence and selected unit-test output included
Assignment validation	ViewModelValidationTests.AssignmentInputViewModel_WithMissingTitle_ShouldBeInvalid	Confirms assignment form rules	UniManage.Tests	Source evidence and selected unit-test output included
Messaging validation	ViewModelValidationTests.MessageComposeViewModel_WithMissingRecipientSubjectAndContent_ShouldBeInvalid	Confirms required messaging fields	UniManage.Tests	Source evidence and selected unit-test output included
Grading bounds	InfrastructureUtilityTests.SubmissionGradingRules_IsValidGrade_ShouldRespectInclusiveZeroToMax	Confirms inclusive grade bounds used with MaxPoints	UniManage.Tests	Source evidence and selected unit-test output included
CSV utility	InfrastructureUtilityTests.CsvWriter_Build_ShouldIncludeHeadersAndUtf8Bom	Confirms UTF-8 BOM and CSV header output	UniManage.Tests	Source evidence and selected unit-test output included

Test Area	Example Test	Purpose	Evidence	Status
Calendar utility	InfrastructureUtilityTests.CalendarLink_BuildIcs_ShouldContainTitleStartAndUrl	Confirms .ics content and Google Calendar URL shape	UniManage.Tests	Source evidence and selected unit-test output included
Meeting link generator	InfrastructureUtilityTests.MeetLinkGenerator_GenerateGoogleMeetUrl_ShouldReturnNonEmptyMeetUrl	Confirms generated meeting URL and validator	UniManage.Tests	Source evidence and selected unit-test output included
Password reset email	InfrastructureUtilityTests.EmailTemplates_BuildPasswordResetEmail_ShouldContainResetGuidanceAndEncodedLink	Confirms reset email content and encoded link	UniManage.Tests	Source evidence and selected unit-test output included
Security headers	InfrastructureUtilityTests.SecurityHeadersMiddleware_ShouldAddExpectedResponseHeaders	Confirms security headers added to responses	UniManage.Tests	Source evidence and selected unit-test output included

Table 19: Selected Unit Test Evidence

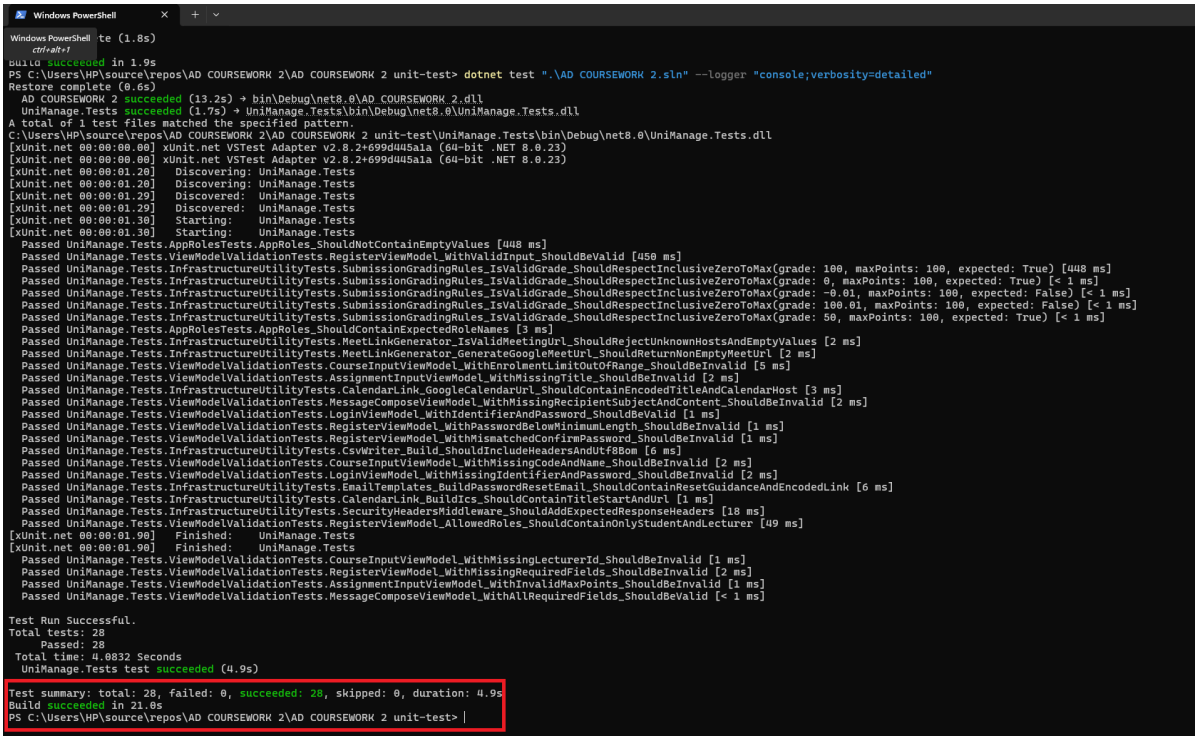


Figure 19: Selected Unit Test Output

This screenshot shows the selected unit-test output. The test run completed successfully with 28 tests passed and 0 failed tests.

16. Challenges Faced

16.1 Technical Challenges

The main technical challenges were Identity integration with Pomelo MySQL, EF Core retry configuration, the two rate-limiting policies, QuestPDF Community licence setup, and the audit logger. These were handled by checking the official documentation, running the application, and using the audit log to confirm that important events were captured.

One challenge was keeping the final changes useful without making the system unstable close to submission. The author refined Admin User Management, environment-based configuration, seed data, and hosted testing, but avoided adding optional modules that were not central to the marking scheme. This kept the focus on authentication, dashboards, course management, enrolment, assignment and grading, reporting, communication, and security.

Another challenge was moving from local development to a hosted environment. The application worked in Visual Studio, but Docker, database configuration, domain access, HTTPS or reverse proxy behaviour, and Google OAuth redirect settings still had to be checked carefully. I handled this step by step and retested login and role navigation after configuration changes.

The system was tested using seeded demo accounts for Administrator, Lecturer, and Student roles. Functional testing covered registration, login, Google sign-in where configured, course management, enrolment, assignment creation, student submission, grading, reports, meetings, messaging, and Admin User Management. Role-based testing checked that restricted pages could not be opened by unauthorised users. The seed data made it easier to repeat the same tests after rebuilding or restarting the application.

16.2 Documentation Challenges

The main documentation challenge was writing a long report without inventing features. I handled this by checking the source code chapter by chapter and keeping uncertain items as limitations or future improvements rather than presenting them as completed evidence. Content from unrelated coursework reports was avoided because it would not match the submitted UniManage application.

During the final export stage, I had to check the PDF manually because captions, screenshots and tables can shift when a Word document is exported.

16.3 Evidence Preparation Challenges

Capturing useful evidence took separate planning. Screenshots need the right window size, readable text, and redacted personal data. Diagrams need to be exported clearly enough for Word and PDF. Database and deployment screenshots must not reveal connection strings or credentials. The final evidence summary and screenshot evidence were prepared to keep the report consistent.

16.4 Individual Completion Challenges

The approved individual completion route required me to manage the full workload, including planning, testing, documentation, evidence capture, and final submission. It also required careful wording so that the report stayed neutral and focused on the submitted work. I managed

this through weekly planning, evidence tracking, and wording checks before the final version was prepared.

17. Limitations

17.1 Functional Limitations

The current build includes the required academic workflows and a focused Admin User Management feature. It does not include a full automated unit and integration test suite for every controller and UI flow, advanced reporting filters, attendance tracking, announcements, or live video conferencing provisioning. Selected xUnit testing evidence is useful where included in the final package, but it does not replace broader automated coverage. These items are recorded as limitations or future improvements rather than completed features.

17.2 Testing Limitations

Selected automated test source files and unit-test output are included in the final evidence. Manual testing has also been used for functional, validation, and security behaviour. The test plan in Chapter 15 lists thirty-seven test cases; build verification is confirmed, and the remaining manual cases are listed for review rather than separately evidenced with screenshots.

17.3 Deployment Limitations

The application has been verified to run locally on https://localhost:7212 and http://localhost:5103. A supporting hosted demonstration has been deployed to Google Cloud using a Docker container, with the intended public address https://lms.yasaboy.com/ and an earlier IP-based endpoint http://34.171.6.234:8080 retained as backup evidence. Section 8.20 and Appendix C distinguish between DNS and server-reachability screenshots on the one hand, and application or HTTPS session evidence on the other.

The Cloudflare DNS evidence shows DNS-only routing rather than Cloudflare proxy or security services. The Nginx screenshot confirms that the server responded with the default Nginx page, but it does not by itself prove that Nginx was configured as a reverse proxy to the UniManage Docker container. Reverse proxy hardening, Cloudflare proxy configuration, and documented HTTPS/TLS configuration remain areas for final verification or future improvement unless additional evidence is inserted.

The hosted demonstration is still presented as coursework demonstration evidence rather than a production deployment. For production use, the author would need to document and harden the complete deployment pipeline, including reverse proxy configuration if used, DNS configuration, secret management, firewall rules, monitoring, database backup procedures, and CI/CD automation.

The hosted URL can support review when the cloud virtual machine and Docker container are running. If the hosted route is temporarily unavailable, the local Visual Studio assessment route documented in the user manual remains supported.

17.4 Accessibility Limitations

The author has applied basic accessibility considerations such as a skip link, label association, and semantic markup. A formal WCAG 2.1 AA audit has not been carried out and is therefore noted as a limitation.

17.5 Evidence Limitations

The final report includes screenshot, diagram, repository, build, database, hosted demonstration, responsive layout, and selected unit-test evidence. Additional items are treated as optional supporting evidence or future improvements, rather than as required evidence.

18. Own Reflection on Experience

18.1 Technical Learning

I learned that building the feature is only one part of the coursework. The work also had to be proved through screenshots, build output, diagrams, test evidence, and clear documentation. Technically, UniManage required ASP.NET Core MVC routing, Razor views, EF Core, Pomelo MySQL, Identity, anti-forgery, rate limiting, custom middleware, and dependency injection to work together in one project. A key lesson was that security and data integrity need several controls. For example, the unique constraint on Enrollment(StudentId, CourseId) and the application-level duplicate check work together to prevent duplicate enrolments.

I also learned to rely more carefully on official documentation. Microsoft Learn, EF Core documentation, Pomelo MySQL documentation, QuestPDF documentation, and Bootstrap

documentation helped shape the implementation. Where ASP.NET Core already provided a feature, such as anti-forgery, I used the framework feature instead of rebuilding it.

A further technical learning point is the importance of asynchronous programming. Database, file, and email operations all benefit from async/await, which keeps the application responsive and avoids blocking the thread pool. Working through the project has reinforced the discipline of writing asynchronous code from the outset rather than retrofitting it later.

18.2 Professional Learning

I learned that documentation is not a separate last-minute task. It has to explain the code, prove the features, and stay aligned with the marking scheme. The final evidence summary helped track screenshots and diagrams, while the marking alignment report helped keep the writing focused on the brief.

Preparing the evidence pack taught me that a working application is not enough; the implementation, screenshots, build output and documentation must also match clearly.

I also learned how to balance breadth and depth. A long report can become repetitive if every section says the same thing in different words. I therefore tried to give each chapter a clear focus and to avoid unnecessary overlap. Where overlap is unavoidable, I used cross-references rather than restating the same content.

18.3 Problem-Solving Experience

Several problems required deliberate problem-solving during the project. Three examples are given.

First, the integration of EF Core with the Pomelo MySQL provider initially produced transient connection errors during startup because the local MySQL service was slow to start. The author resolved this by enabling retry on failure with up to five retries and a thirty second maximum delay, which hid the transient failures from the user.

Second, the rate limiter was first configured with a single policy that applied to every endpoint, which caused unnecessary 429 responses on dashboard pages. The author resolved this by splitting the rate limiter into two named policies (auth and upload) and applying them only to the relevant actions.

Third, the audit logger initially failed silently when the audit table did not exist (during the first migration). The author resolved this by ordering the migrations carefully, ensuring that the audit table is created before any audit calls run, and by wrapping the audit write in a try/catch block so that an audit failure cannot crash the calling action.

18.4 Impact of Approved Individual Submission

The approved individual completion route affected the planning, workload, and documentation. The author had to treat the project as a single-person delivery, with a clear scope and weekly schedule. Every chapter, screenshot, and diagram had to be prepared and checked by the author, so prioritisation became important. The individual route also gave clarity because the final scope and evidence list were easier to track.

The individual route did not reduce the academic standard. I still had to produce a working codebase, a full documentation package, and enough evidence for marking and viva discussion. My response to the approval was to focus on clear evidence rather than making unsupported claims.

18.5 Ethical and Academic Integrity Awareness

Academic integrity has been a central concern. The author has not copied text from unrelated coursework reports. Where official documentation has informed a decision, the documentation is acknowledged in the references chapter. Where a feature could not be confirmed by source code, the author has recorded it as a limitation or future improvement rather than embellishing the report. The author has also kept sensitive configuration values out of the report and has redacted personal data in screenshots.

Data protection has informed several decisions. The audit log records identifiers and actions but does not record passwords or sensitive payloads. Personal data shown in screenshots is redacted. The retention period for audit log data is not formally documented, which is recorded as a limitation.

18.6 Future Learning Plan

The author plans to continue the learning from this coursework in three areas.

First, the author intends to learn automated testing in depth, including unit testing with xUnit, integration testing with the ASP.NET Core test host, and UI testing with Playwright. This will allow the author to ship code with regression tests rather than relying solely on manual testing.

Second, the author intends to learn continuous integration and deployment using GitHub Actions, which will support a hosted demonstration environment and consistent build verification.

Third, the author intends to expand the security and data protection awareness gained during this project by studying OWASP guidance, the .NET security documentation, and accessibility standards in more depth. These topics combine technical and professional learning, which is a useful foundation for further academic work and for industry practice.

19. Individual Contribution

The full individual contribution report is attached as a supporting document. The supporting document explains the author's approved individual completion context and summarises the implementation, testing, documentation, and submission preparation work.

For convenience, the contribution is summarised below in a single paragraph: the author selected and completed UniManage as an approved individual submission, has produced the source code, the documentation package, the diagrams, the user manual, the evidence summary, and this main report. The author has reviewed the source code chapter by chapter, has prepared a marker-friendly mapping between requirements and evidence, and has kept unverifiable claims as limitations or optional supporting evidence. The contribution is therefore the entire delivery, prepared and submitted in accordance with the academic integrity rules of CS6004ES Application Development.

20. Future Improvements

Table 20 lists the future improvements that the author has identified.

Improvement	Justification
Automated unit and integration tests	Provide regression coverage and reduce reliance on manual testing.
Continuous integration and deployment	Provide consistent build verification and a hosted demonstration environment.
Accessibility audit (WCAG 2.1 AA)	Confirm that the user interface is accessible to all users.
Further Admin User Management refinements	Add clearer filtering, export, and audit views for administrator account control.
Advanced report filters and chart visualisations	Improve the academic insight provided by the reporting suite.
Email and in-app notifications	Notify students of new assignments and grades, and lecturers of new submissions.
Two-factor authentication	Strengthen authentication for high-privilege accounts.
Structured logging with Serilog or similar	Improve observability and integrate with log management tools.
Caching of read-heavy queries	Improve dashboard performance for large datasets.
Internationalisation and localisation	Support multiple languages where required.
Mobile-friendly enhancements beyond Bootstrap defaults	Improve the experience on small viewports.
Database backup automation	Provide a routine backup script and a restore process.
Maintain HTTPS/TLS certificate configuration and document renewal steps	Once a secure HTTPS session on the custom domain is evidenced for UniManage, document the certificate authority used, the renewal cadence, and any automation so that TLS remains valid over time.
Enable and document Cloudflare proxy/CDN protection if it is required for the final hosted demonstration	Figure UM21 shows DNS only; proxied Cloudflare mode is not claimed. Enable and capture updated evidence only if required, without exposing secrets.
Complete and document Nginx reverse proxy configuration from the public domain to the UniManage Docker container	Figure UM22 shows the default Nginx page only; capture redacted configuration or routing evidence before claiming full reverse proxy completion.
Keep hosted application screenshot evidence current	Keep Figure UM19 current for future demonstrations and review so the hosted

Improvement	Justification
	application view remains aligned with the infrastructure evidence.
Document and harden the reverse proxy configuration if Nginx is used	If the deployment uses Nginx in front of the Docker container, capture the relevant configuration (with secrets redacted) and harden the proxy with appropriate timeouts, headers, and rate limits.
Document DNS and Cloudflare configuration if used	Figure UM21 provides partial DNS evidence; extend with any further redacted records needed for assessment, including proxy status if that mode is enabled later.
Extend secret handling with a managed secret store	Continue keeping connection strings, SMTP credentials, Google client secrets, and DNS or provider credentials out of committed files; consider Google Secret Manager or a similar managed store for a more formal deployment.
Add CI/CD deployment automation for Docker images	Automate the build, image push, and deployment of the Docker container so that updates to the source repository can be pushed to the hosted demonstration without manual steps.
Add uptime monitoring and database backup procedures	Add a basic uptime monitor for the hosted endpoint and a routine MySQL backup so that the demonstration link remains reliable across the assessment window.

Table 20: Future Improvements and Justification

21. Conclusion

UniManage has been delivered as an approved individual submission for CS6004ES Application Development Coursework 2. The project consists of a complete ASP.NET Core MVC application with thirteen controllers, twelve domain models, eighteen view models, thirteen infrastructure components, fifty Razor views, four EF Core migrations, an Identity configuration with strong password rules and lockout, a rate limiter with two policies, anti-forgery, security headers, HTTPS, HSTS, and a custom audit logger.

The system addresses the academic management problem set by the brief, supports Student, Lecturer, and Administrator roles through role-based dashboards, implements course management, enrolment, assignment management, submission upload, grading, reporting and analytics with CSV and PDF export, internal messaging, meeting scheduling, profile and password management, and an audit log review screen. The system applies validation through data annotations, ModelState handling, ownership checks, file validation, anti-forgery, and exception handling.

The documentation package supports the submission with a main report, an individual contribution report, a user manual, a final evidence summary, diagram sources, source/build links, screenshot evidence, and supporting appendices. The build has been verified locally with zero warnings and zero errors. Items outside the final scope are treated as limitations, optional supporting evidence, or future improvements rather than core evidence.

The author has reflected on the technical, professional, and academic learning gained during the project and has identified a clear set of future improvements. The submission is therefore complete, evidence based, and ready to support the marker's assessment.

22. References

Bootstrap (2026) Bootstrap 5 documentation. Available at: https://getbootstrap.com/docs/5.3/ (Accessed: 5 May 2026).

Microsoft (2026a) ASP.NET Core MVC documentation. Microsoft Learn. Available at: https://learn.microsoft.com/aspnet/core/mvc (Accessed: 5 May 2026).

Microsoft (2026b) C# documentation. Microsoft Learn. Available at: https://learn.microsoft.com/dotnet/csharp (Accessed: 5 May 2026).

Microsoft (2026c) ASP.NET Core Identity documentation. Microsoft Learn. Available at: https://learn.microsoft.com/aspnet/core/security/authentication/identity (Accessed: 5 May 2026).

Microsoft (2026d) Entity Framework Core documentation. Microsoft Learn. Available at: https://learn.microsoft.com/ef/core (Accessed: 5 May 2026).

Microsoft (2026e) ASP.NET Core security documentation. Microsoft Learn. Available at: https://learn.microsoft.com/aspnet/core/security (Accessed: 5 May 2026).

Microsoft (2026f) ASP.NET Core rate limiting middleware. Microsoft Learn. Available at: https://learn.microsoft.com/aspnet/core/performance/rate-limit (Accessed: 5 May 2026).

Open Web Application Security Project (2026) OWASP Top 10. Available at: https://owasp.org/www-project-top-ten/ (Accessed: 5 May 2026).

Pomelo Foundation (2026) Pomelo Entity Framework Core MySQL provider. GitHub. Available at: https://github.com/PomeloFoundation/Pomelo.EntityFrameworkCore.MySql (Accessed: 5 May 2026).

Pressman, R.S. and Maxim, B.R. (2014) Software Engineering: A Practitioner's Approach. 8th edn. New York: McGraw-Hill.

QuestPDF (2026) QuestPDF documentation. Available at: https://www.questpdf.com/documentation/getting-started.html (Accessed: 5 May 2026).

Sommerville, I. (2015) Software Engineering. 10th edn. Harlow: Pearson Education.

Docker (2026) Docker documentation. Available at: https://docs.docker.com/ (Accessed: 5 May 2026).

Google Cloud (2026) Compute Engine documentation. Available at: https://cloud.google.com/compute/docs (Accessed: 5 May 2026).

Cloudflare (2026) Cloudflare DNS documentation. Available at: https://developers.cloudflare.com/dns/ (Accessed: 5 May 2026).

Nginx (2026) Nginx documentation. Available at: https://nginx.org/en/docs/ (Accessed: 5 May 2026).

Oracle (2026) MySQL 8.0 reference manual. Available at: https://dev.mysql.com/doc/refman/8.0/en/ (Accessed: 5 May 2026).

Mozilla Developer Network (2026) HTTP security headers. Available at: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers (Accessed: 5 May 2026).

23. Appendices

Appendix A: Selected Source Code Evidence

This appendix collects short, illustrative code snippets that support the body of the report. Each snippet identifies the file path so the marker can locate the full source.

A1. Identity configuration in Program.cs (excerpt):

```
builder.Services.AddIdentity<ApplicationUser, IdentityRole>(options =>
{
    options.Password.RequiredLength = 8;
    options.Password.RequireDigit = true;
    options.Password.RequireUppercase = true;
    options.Password.RequireLowercase = true;
    options.Password.RequireNonAlphanumeric = false;
    options.User.RequireUniqueEmail = true;
    options.Lockout.AllowedForNewUsers = true;
    options.Lockout.MaxFailedAccessAttempts = 5;
    options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(15);
})
    .AddEntityFrameworkStores<ApplicationDbContext>()
    .AddDefaultTokenProviders();
```

A2. EF Core registration in Program.cs:

```
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseMySql(connectionString, new MySqlServerVersion(new Version(8,
0, 36)), mySql =>
        mySql.EnableRetryOnFailure(maxRetryCount: 5, maxRetryDelay:
TimeSpan.FromSeconds(30), errorNumbersToAdd: null)));
```

A3. Rate limiter registration in Program.cs (excerpt) is shown in Listing 8.1.

A4. Cookie configuration in Program.cs:

```
builder.Services.ConfigureApplicationCookie(options =>
{
    options.LoginPath = "/Account/Login";
    options.LogoutPath = "/Account/Logout";
    options.AccessDeniedPath = "/Account/AccessDenied";
    options.SlidingExpiration = true;
    options.ExpireTimeSpan = TimeSpan.FromHours(8);
```

```
    options.Cookie.HttpOnly = true;
    options.Cookie.SameSite = SameSiteMode.Lax;
    options.Cookie.SecurePolicy = CookieSecurePolicy.SameAsRequest;
});
```

A5. Other illustrative listings should be added during final Word formatting from the following files: Controllers/AccountController.cs, Controllers/EnrollmentsController.cs, Controllers/SubmissionsController.cs, Infrastructure/AuditLogger.cs, Infrastructure/UploadedFileStore.cs, Data/ApplicationDbContext.cs, Data/DbInitializer.cs. Keep each excerpt to ten to thirty lines.

Appendix B: Diagram Sources

Editable diagram sources are stored in the shared Google Drive diagram folder: https://drive.google.com/drive/folders/1PvvXtEf_6VFk632ysTXCwjlybSpcOaef?usp=sharing. Individual draw.io links for the use case, ER and UML class diagrams are listed below. PNG exports are inserted in the report for marker readability.

Figure	Diagram	Editable Source Link
Figure 1	High-Level Architecture Diagram	Shared diagram folder: https://drive.google.com/drive/folders/1PvvXtEf_6VFk632ysTXCwjlybSpcOaef?usp=sharing
Figure 2	Use Case Diagram	Editable draw.io source: https://drive.google.com/file/d/1H9c4yXOcljag8dTxCizQyQIkxixHglvC/view?usp=sharing
Figure 3	Entity Relationship Diagram	Editable draw.io source: https://drive.google.com/file/d/15IA6BQnpswKNplfang6vebdsNYfXlYU8/view?usp=sharing
Figure 4	UML Class Diagram	Editable draw.io source: https://drive.google.com/file/d/189YrFogTsnGSPoH0Se6xBBerum1HaiOl/view?usp=sharing
Figure 5	Application Flowchart	Shared diagram folder: https://drive.google.com/drive/folders/1PvvXtEf_6VFk632ysTXCwjlybSpcOaef?usp=sharing
Figure 6	Authentication and RBAC Flow	Shared diagram folder: https://drive.google.com/drive/folders/1PvvXtEf_6VFk632ysTXCwjlybSpcOaef?usp=sharing

Appendix B.1: Source Repository, Database and Build Evidence

The repository link, packaged build link, and database evidence are included in the source/build links appendix. The current repository remote is https://github.com/YasasPasinduFernando/AD-COURSEWORK-2.git. The current detected migrations are listed in Section 7.6. The build was verified locally with dotnet build --no-restore, returning zero warnings and zero errors.

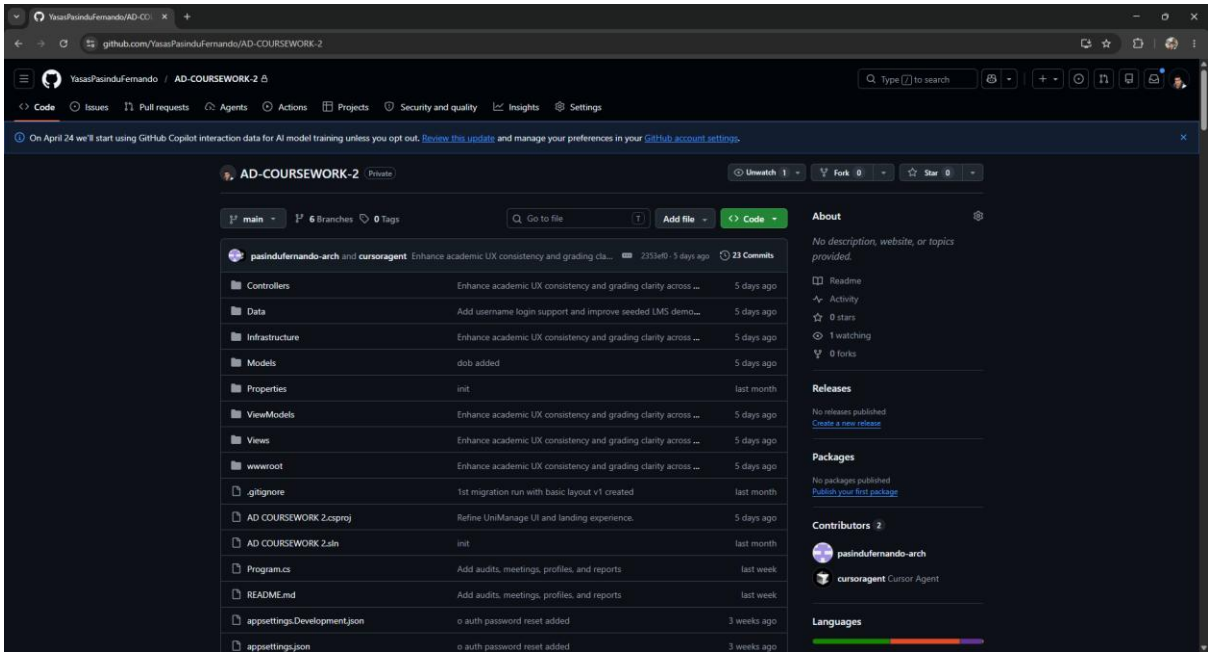


Figure A1: Source Repository Evidence

This screenshot shows the public landing page of the project's source repository on GitHub.

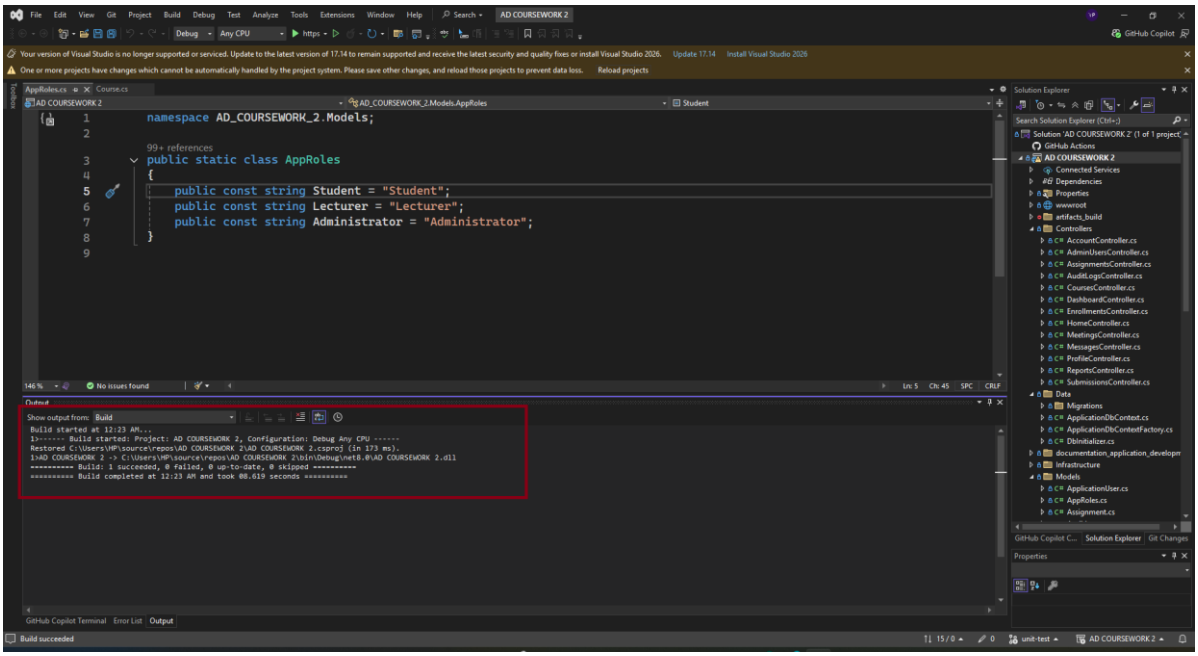


Figure A2: Successful Build Output

This screenshot shows the Visual Studio build output for the submitted UniManage project, with the build completed successfully and no failed projects.

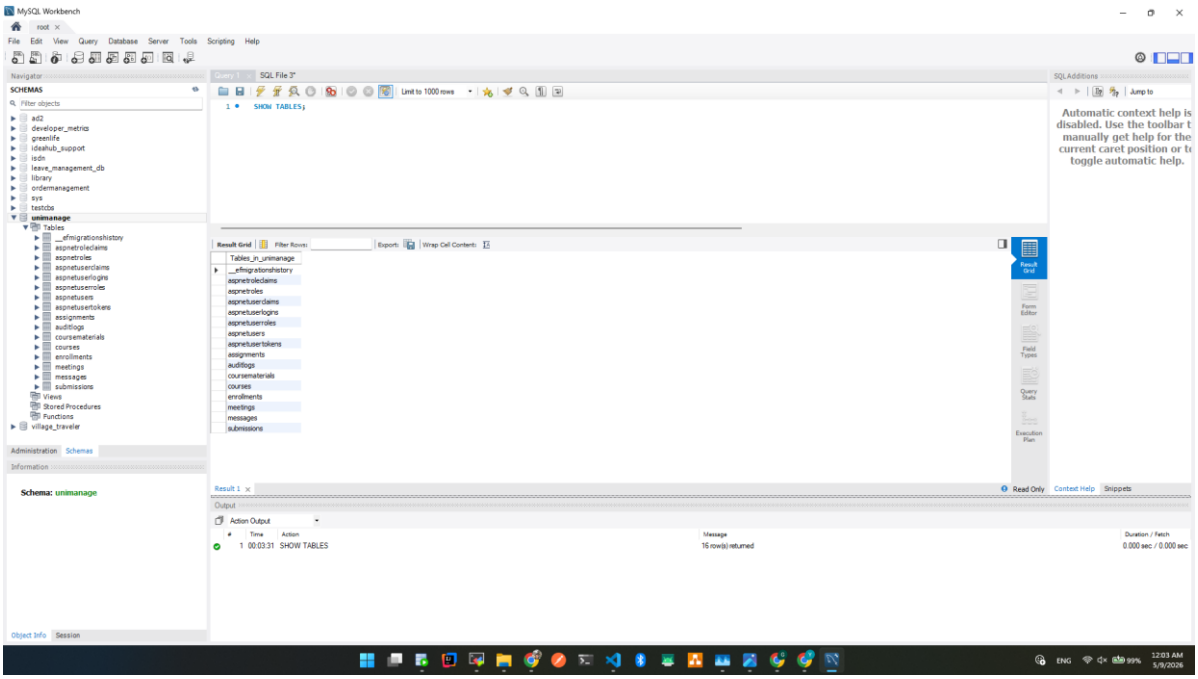


Figure A3: Database and Seed Data Evidence

This screenshot shows a database client connected to the seeded UniManage database, displaying the schema and a representative table such as Courses, Enrollments, and Submissions.

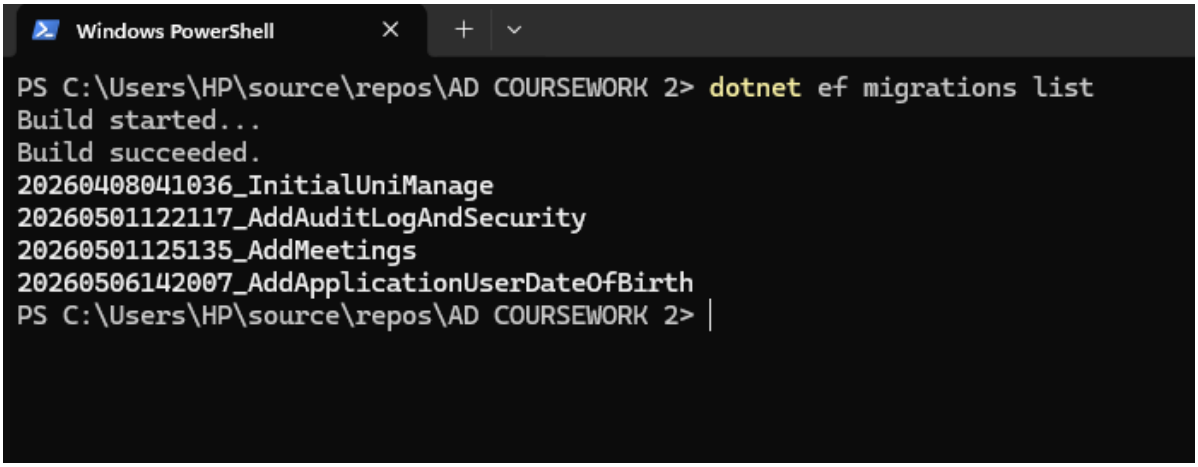


Figure A4: EF Core Migration Evidence

This screenshot shows the console output of dotnet ef migrations list, confirming the four EF Core migrations.

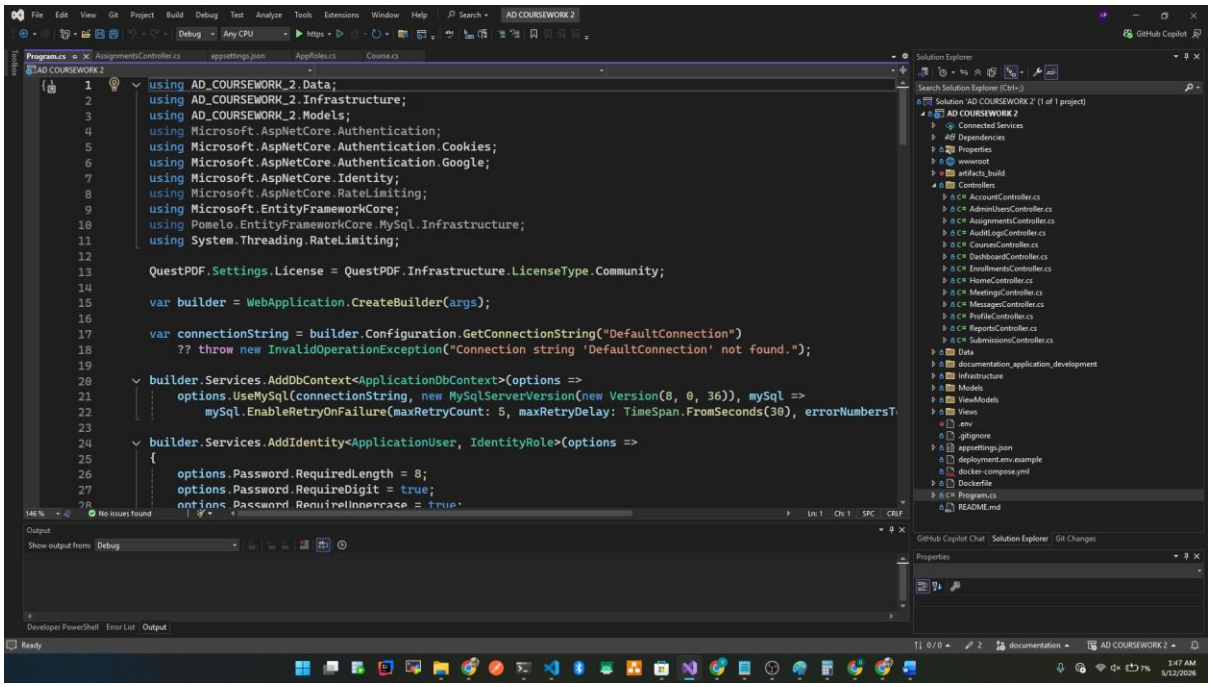


Figure A5: Visual Studio Source Code Evidence

This screenshot shows the Visual Studio 2022 solution open with the AD COURSEWORK 2 project loaded in Solution Explorer.

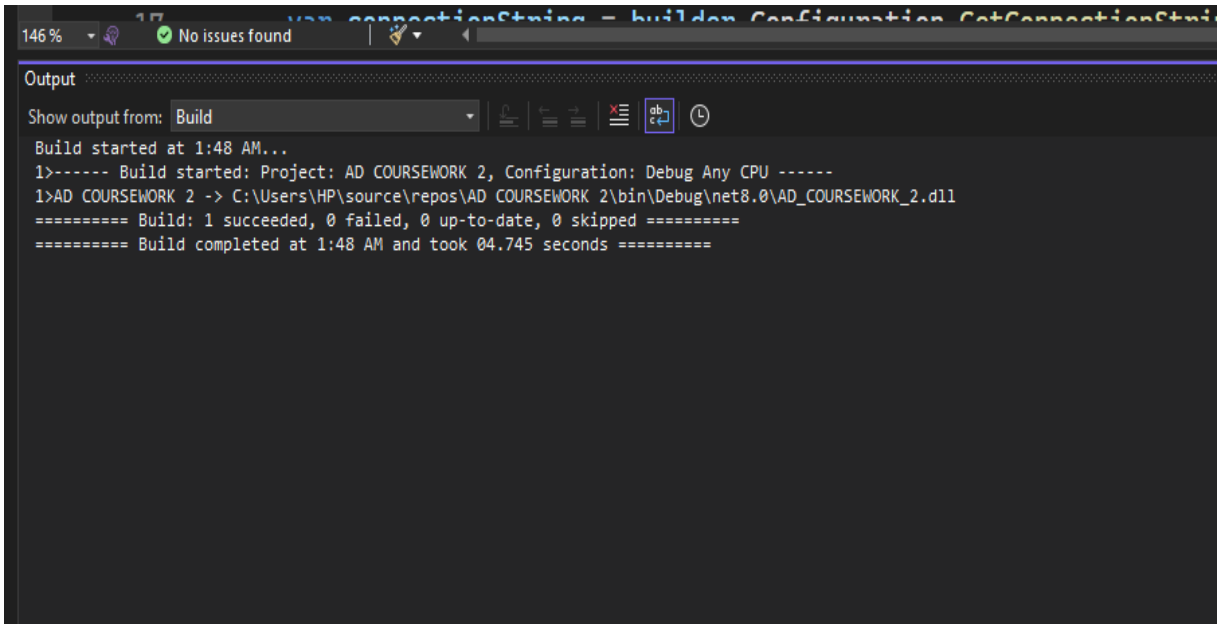


Figure A6: Visual Studio Build Output Evidence

This screenshot shows the console output of dotnet build --no-restore reporting zero warnings and zero errors.

Appendix C: Hosted Deployment Evidence

In addition to the local Visual Studio build, the application has been deployed to a Google Cloud virtual machine using Docker as supporting demonstration evidence. The hosted application URL, backup IP-based endpoint and infrastructure evidence figures are listed below.

Item	Value	Notes
Hosted application on custom domain (UniManage UI)	https://lms.yasaboy.com/	Figure UM19 shows the UniManage homepage on the hosted domain.
Previous IP-based endpoint	http://34.171.6.234:8080	Earlier Google Cloud hosted endpoint, retained as backup evidence.
Hosting platform	Google Cloud	Hosted on a Compute Engine virtual machine.
Containerisation	Docker	The application is packaged as a Docker image.
HTTPS / TLS on custom domain (browser session evidence)	HTTPS/domain access used where available during final testing	Full production hardening, long-term monitoring, automated backups and formal security testing

Item	Value	Notes
		remain future improvements.
Cloudflare DNS record	lms A record to 34.171.6.234	Supporting DNS evidence in Figure UM21; full CDN/security hardening is not claimed.
Nginx server reachability	Default Nginx page on lms.yasaboy.com	Completed as server reachability evidence in Figure UM22.
Cloudflare DNS/security hardening	DNS evidence included	Future improvement; full CDN/security hardening is not claimed.
Nginx application routing/hardening	Supporting server evidence only	Future improvement; full reverse proxy hardening is not claimed.
Deployment purpose	Supporting viva and demonstration evidence	Not a replacement for the local Visual Studio assessment route.
Visual Studio local assessment	Required	The coursework brief assesses the application using Visual Studio 2017 or higher.

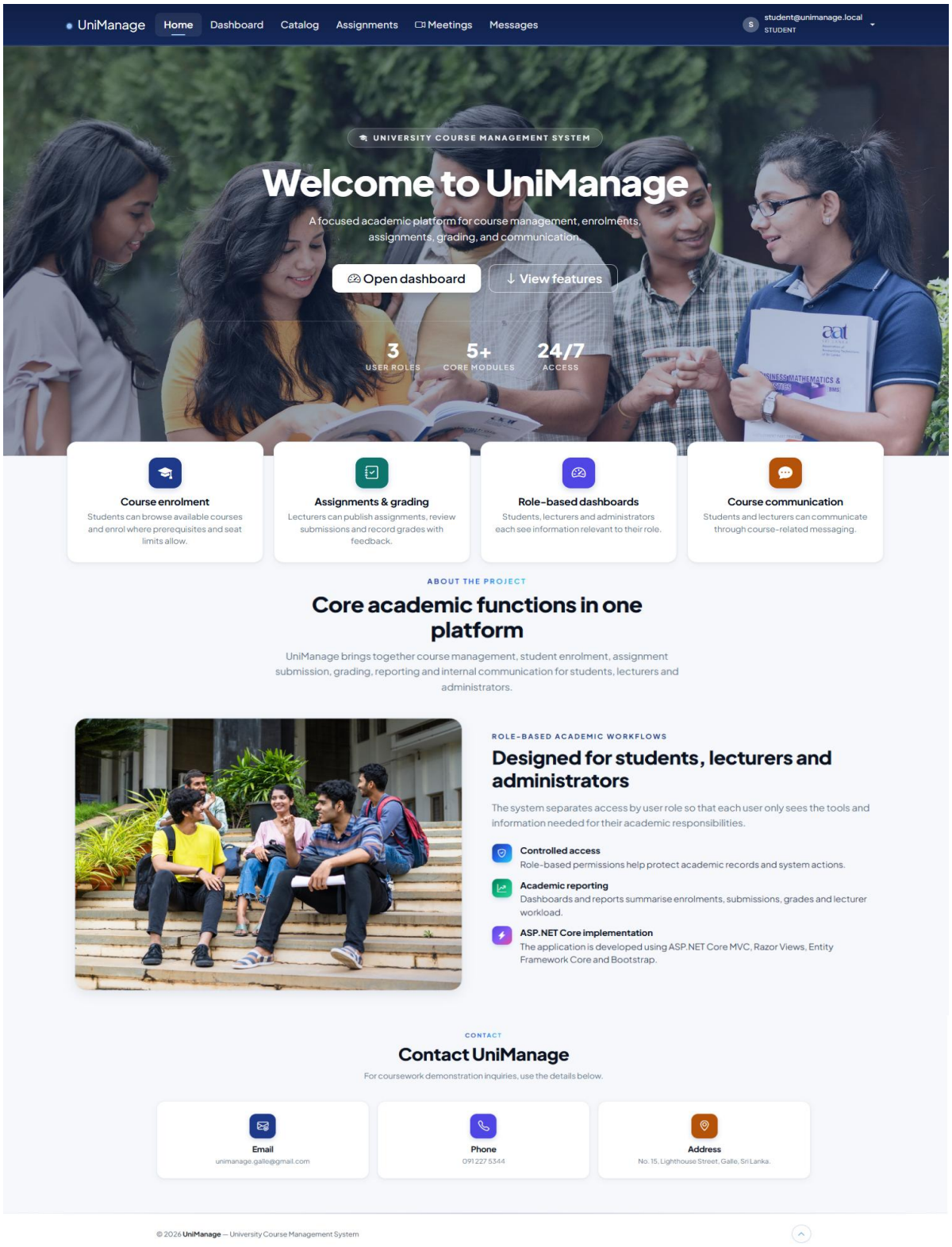


Figure A7: Hosted UniManage Application

This screenshot shows the UniManage application running through the hosted custom domain.

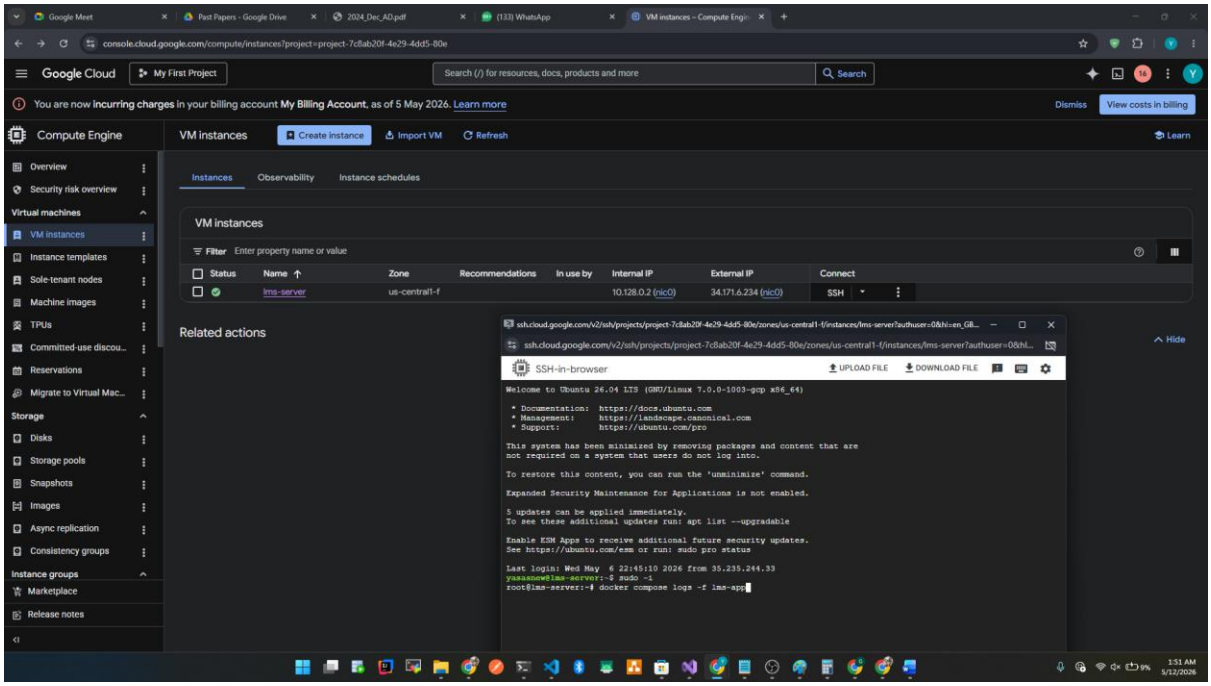


Figure A8: Google Cloud VM and Docker Deployment Evidence

This screenshot shows the Google Cloud deployment evidence used to support the hosted demonstration.

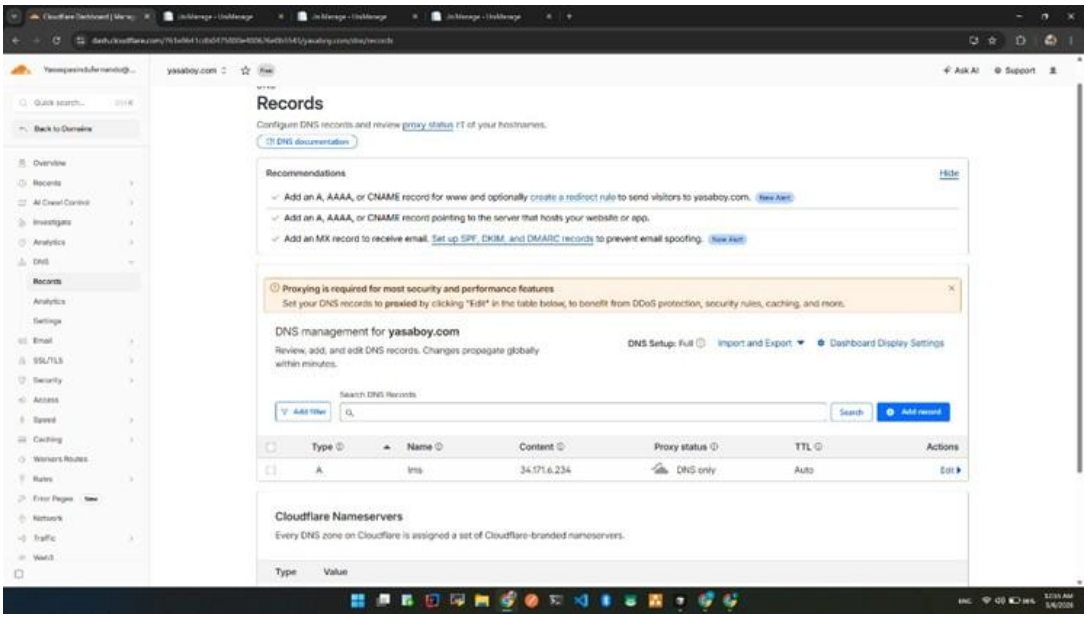


Figure A9: Cloudflare DNS Record Evidence

This screenshot shows the Cloudflare DNS A record used to point the lms.yasaboy.com subdomain to the Google Cloud public IP address.

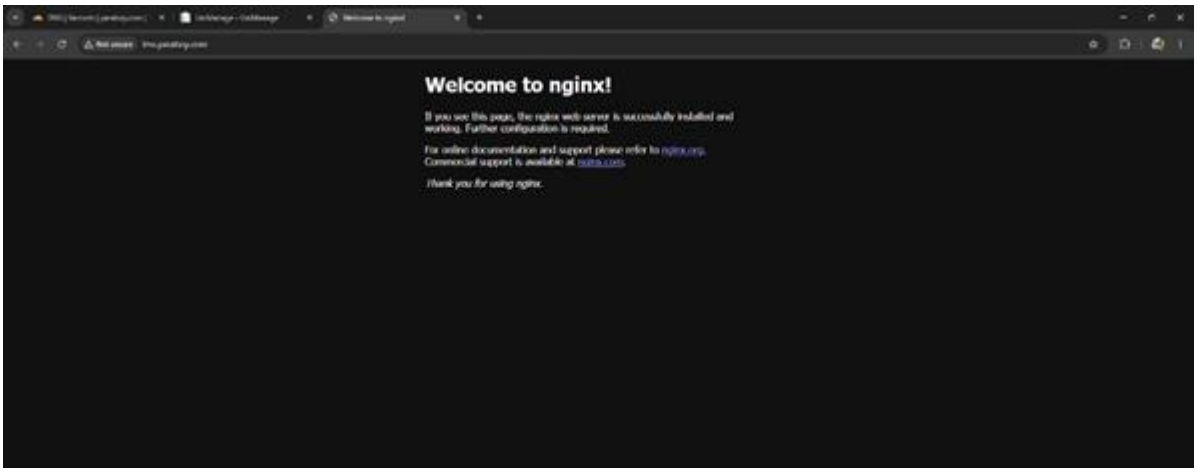


Figure A10: Nginx Server Reachability Evidence

This screenshot shows the default Nginx page loading from the lms.yasaboy.com host, confirming that Nginx was installed and reachable on the server at the time of evidence capture.

Notes for the marker:

The custom domain was configured for the hosted demonstration version. HTTPS/domain access was used where available during final testing. Full production hardening, long-term monitoring, automated backups and formal security testing remain future improvements.

The local Visual Studio build remains the primary assessment route. The local build has been verified with dotnet build --no-restore returning zero warnings and zero errors.

The author has confirmed that no database password, SMTP password, Google client secret, SSH key, DNS or provider credential, or cloud service account key is included in the body of the report or in the hosted screenshots.

Appendix D: User Manual Reference

The complete user manual is included in the final report. The user manual covers system requirements, project package contents, installation, login, user roles, troubleshooting, reporting, and viva demonstration notes.

Appendix E: Test Logs and Audit Evidence

The test plan in Chapter 15 lists the test cases. Evidence is included through screenshots, audit log examples, build output, and supporting appendices where relevant. The build console output is included as supporting build evidence.

Appendix F: Validation Message Evidence

The validation rules used by each form are listed in Table 14. Selected examples of validation messages are included as screenshot evidence where available, including:

Required field missing on registration.
Password too weak on registration.
Wrong credentials on login.
Course code already in use.
Enrolment limit reached.
File too large or wrong type on submission upload.
The validation messages above are covered through inserted evidence where available and through the validation and testing sections.

Appendix G: Configuration Redaction Notes

The author has redacted sensitive configuration values from the report. The following keys must not appear in screenshots, source code listings, or appendices:

ConnectionStrings:DefaultConnection (database password and host).
Authentication:Google:ClientSecret.
Email:Password (SMTP password).
If a screenshot accidentally captures one of these values, the screenshot must be retaken or redacted before being used as evidence. The author has avoided pasting appsettings.json or appsettings.Development.json into the report for this reason.

Appendix H: Viva Demonstration Script

This appendix provides a short demonstration guide for the viva. The steps are intended to help the author present the implemented system in a clear and logical order.

1. Open Visual Studio and load the AD COURSEWORK 2.sln solution. Briefly highlight the main folder structure, including Controllers, Models, ViewModels, Views, Data, Infrastructure, and wwwroot.

2. Run dotnet build --no-restore from the terminal and show the successful build result with zero errors.

3. Start the application using Visual Studio or dotnet run. Show the local application URL or the hosted demonstration URL, depending on the demonstration environment.

4. Sign in using the seeded Administrator account provided in the private setup notes. Demonstrate the Administrator dashboard, Admin User Management, course management, reports, and audit log.

5. Sign out and sign in using the seeded Lecturer account. Demonstrate the Lecturer dashboard, MyCourses, assignment management, grading, course materials, messaging, and meeting scheduling.

6. Sign out and sign in using the seeded Student account. Demonstrate the Student dashboard, course browsing, enrolment, assignment submission, grade view, inbox, and meetings.

7. Trigger a validation error, such as submitting a required form without completing all required fields, and show the validation message.

8. Show an audit log entry that relates to a recent action, such as login, enrolment, course update, or grading.

9. Briefly show the EF Core migrations folder and explain how migrations and seeded data support the database setup.

10. Briefly walk through Program.cs and explain the main configuration areas: ASP.NET Core Identity, EF Core and MySQL, rate limiting, anti-forgery protection, security headers, HTTPS redirection, and MVC routing.

11. Mention genuine limitations and future improvements, such as wider accessibility testing, more automated tests, production monitoring, and further deployment hardening.

Supporting Document: Individual Contribution Report

Student Name: Yasas Pasindu Fernando

Student ID: 25026764

Module: CS6004ES Application Development

Assessment: Coursework 2

Project: UniManage - University Course Management System (UCMS)

Centre: ESOFT Metro Campus / London Metropolitan University

Submission Type: Approved individual submission

Companion Document: Main Application Development Report

1. Introduction

This supporting document explains the author's individual contribution to the final UniManage coursework submission. It sits alongside the main technical report and focuses on the work completed, the decisions made, the evidence prepared, and the learning gained during the project. The wording is neutral and evidence-based because the submission was approved as an individual completion route.

The report is organised so that the marker can move quickly from one academic area to the next. Each numbered section addresses a specific part of the project. Table IC1 and Table IC2 give a quicker summary of the contribution and the supporting evidence.

The report focuses on the work that the author produced and how that work is supported through source code, build evidence, and prepared screenshots.

2. Project Context

UniManage is a University Course Management System implemented as an ASP.NET Core MVC web application targeting .NET 8. It uses C# for application logic, Razor views and Bootstrap for the interface, Entity Framework Core with Pomelo MySQL for data access, and ASP.NET Core Identity for authentication and roles. The system supports Student, Lecturer, and Administrator workflows, including dashboards, course management, enrolment, assignments, submissions, grading, reports, messages, meetings, profile management, and audit logging.

The CS6004ES Application Development brief sets clear marking items, including the interface, registration and login, role dashboards, course management, enrolment, assignment and grading, reports, communication, security, validation, programming style, installation guide, diagrams, class and method descriptions, and reflection. The implementation and documentation are aligned with these items so that the marker can trace each area to evidence.

The technical baseline is recorded in the main report. Thirteen controllers, twelve domain models, eighteen view models, thirteen infrastructure components, fifty Razor views, four Entity Framework Core migrations, and a Program.cs startup file have been confirmed by source-code review. The build has been verified locally with dotnet build --no-restore, returning zero warnings and zero errors.

3. Approved Individual Submission Context

Following approval to complete the coursework individually, the author took responsibility for the final implementation, testing, documentation, and submission preparation. The final package is supported by screenshots, diagrams, source/build links, database evidence, hosted demonstration evidence, and selected unit-test evidence.

The approved individual submission context shaped this contribution report. Each academic area is described as work completed by the author, and the report separates source-confirmed work from optional supporting evidence and future improvements.

4. Personal Motivation and Project Selection

The author chose to complete UniManage as the coursework project for several reasons. First, a university course management system is a realistic academic problem that combines authentication, role-based access, structured database relationships, validation, reporting, and user interface design in a single application. Second, the problem aligns naturally with the marking items in the brief, which makes it easier to demonstrate coverage of each marking area through source-code evidence. Third, the technologies selected for the project, namely ASP.NET Core MVC, C#, Razor, Entity Framework Core, MySQL, and ASP.NET Core Identity, are widely used in industry and are therefore valuable to the author's professional development beyond the coursework.

Personal motivation also played a part. The author has been interested in modern .NET web development since the start of the module and chose UniManage because it allowed the author to apply that interest to a focused academic problem. The approved individual completion route reinforced the author's commitment to deliver a complete, evidence-based submission rather than a partial implementation.

5. Summary of Main Contribution

The author's contribution covers the full project life cycle: requirements, architecture, database design, implementation, validation, security, interface work, testing, evidence preparation, and documentation. Table IC1 summarises the contribution by area, with figure references and a status flag for marker convenience.

Area	Contribution Summary	Evidence / Figure Reference
Requirement analysis	Mapped 30 functional requirements (FR1 to FR30) and 10 non-functional requirements (NFR1 to NFR10) to source-code evidence and to marking items.	Tables 1, 2, 7, 11, 14, 15 in the main report
Architecture and system design	Prepared six diagrams (architecture, use case, ERD, UML class, application flowchart, authentication flow) as Mermaid sources and documented design decisions.	Figures 1 to 6, Section 6 of the main report, Appendix C

Area	Contribution Summary	Evidence / Figure Reference
Database design	Documented EF Core configuration, academic entities, relationships, unique indexes, four migrations, and seeded demonstration data.	Figure 3, Table 3, Section 7 of the main report
User registration and login	Documented AccountController with Register, Login, Logout, ForgotPassword, ResetPassword, and optional Google login support.	Figures 7, 7b, 7c, 7d, Section 8.3 of the main report
Role-based access control	Documented AppRoles, role seeding through DbInitializer, [Authorise] attributes, and ownership filters.	Section 8.4 of the main report, Table 17
Student dashboard	Documented DashboardController.Student, StudentDashboardViewModel, and Views/Dashboard/Student.cshtml.	Figures 8, 8b, Section 8.5 of the main report
Lecturer dashboard	Documented DashboardController.Lecturer, LecturerDashboardViewModel, and Views/Dashboard/Lecturer.cshtml.	Figures 9, 9b, Section 8.6 of the main report
Administrator dashboard and user management	Documented DashboardController.Administrator, AdminUsersController, admin view models, dashboard view, user-management list, create/edit, lock/unlock, safety rules, and controlled delete behaviour.	Figures 10, 10b, 10c, 10d, Section 8.7 of the main report
Course management	Documented CoursesController with Index, MyCourses, Create, Edit, Delete, UploadMaterial, search, paging, and ownership filters.	Figures 11, 11b, 11c, 11d, Section 8.8 of the main report
Enrolment system	Documented EnrollmentsController with Browse and Enroll actions, capacity, duplicate, and prerequisite checks.	Figures 12, 12b, 12c, Section 8.9 of the main report

Area	Contribution Summary	Evidence / Figure Reference
Assignment, submission and grading	Documented AssignmentsController, SubmissionsController, ownership checks, file validation, grade range, feedback, and graded lock.	Figures 13, 13b, 13c, 13d, 13e, Sections 8.10 and 8.11 of the main report
Reporting and analytics	Documented ReportsController with six reports plus CSV and PDF export through QuestPDF.	Figures 14, 14b, 14c, 14d, Section 8.12 of the main report
Communication and meetings	Documented MessagesController, allowed-recipient validation, MeetingsController, ICS export.	Figures 15, 15b, 15c, 15d, Sections 8.13 and 8.14 of the main report
Security, validation and error handling	Documented Identity options, lockout, anti-forgery, rate limiter, security headers middleware, environment-based configuration, Admin User Management safety rules, exception handling, validation rules.	Figures 16, 16b, Sections 12 and 13 of the main report, Tables 16, 17
UI/UX and usability	Documented layout, alerts partial, role-aware navigation, validation feedback, responsive design, mobile viewport checks, accessibility considerations.	Section 9 of the main report
Testing and debugging	Prepared test plan with 37 cases (T1 to T37), build verification, manual functional, validation, role, security, hosted, admin account-control and responsive testing.	Section 15 of the main report, Table 8
Documentation and user manual	Prepared the main report (24,000+ words), this contribution report, the user manual, final evidence summary, source/build links, diagram sources, screenshot evidence, and supporting appendices.	documentation_application_development/ folder
Evidence preparation and viva readiness	Prepared screenshot, diagram, source, build, and database evidence for the final report; prepared a viva demonstration script.	Appendix B, Appendix H of the main report

Table IC1: Summary of Individual Contribution

6. Contribution to Requirement Analysis

The author derived the requirements from the coursework brief, the source-code review, and the UniManage case-study problem. This kept the requirement list close to the marking scheme and also close to what the code actually implements.

The author produced thirty functional requirements (FR1 to FR30) covering registration, login, lockout, role-based redirection, dashboards, course management with paging and search, lecturer ownership, course material upload, enrolment with capacity, duplicate, and prerequisite checks, assignment management, submission upload, grading with feedback, secure file download, six reports, CSV and PDF export, internal messaging, allowed-recipient validation, meeting scheduling, ICS export, profile and password management, audit logging, friendly error pages, rate limiting, and anti-forgery protection. The author also produced ten non-functional requirements (NFR1 to NFR10) covering usability, performance, security, maintainability, reliability, data integrity, accessibility, auditability, configurability, and deployability.

The requirements were then mapped to the marking items through Table 7 and Table 8 in the main report. This makes it easier to trace a marking item to requirements, source files, and test cases.

7. Contribution to Architecture and System Design

The author prepared the system architecture diagram, use case diagram, ERD, UML class diagram, application flowchart, and authentication and role-based access flow. Editable sources are linked through the shared diagram folder and the individual diagram links where listed. The diagrams were prepared from the actual implementation rather than from an abstract plan.

The architectural narrative explains how the request flows through the middleware pipeline (HTTPS redirection, static files, security headers, routing, rate limiting, authentication, authorisation), through the controller, through Entity Framework Core, and finally to the MySQL database. The narrative also identifies cross-cutting concerns such as audit logging, validation, and exception handling. Design decisions are documented in Section 6.7 of the main report, including the choice of MVC over Razor Pages, the choice of Pomelo MySQL over

SQL Server / LocalDB, the choice of a single DashboardController rather than three role-specific controllers, the decision to store grade fields directly in Submission rather than in a separate Grade entity, the decision to add a custom security headers middleware, and the choice of QuestPDF for PDF reports.

8. Contribution to Database Design

The author documented the database design from Data/ApplicationDbContext.cs, the model classes in Models/, and the four EF Core migrations under Data/Migrations/. The contribution includes a clear list of academic entities (Table 9), a relationship narrative covering the lecturer-course link, the optional self-reference for prerequisites, the unique student-course enrolment, the assignment-course relationship, the submission-assignment-student-grader relationship, the message sender and receiver relationships, the meeting-course-lecturer relationships, and the optional audit log user reference.

The author also documented the constraints used in the database, including the unique index on Course.Code, the unique index on Enrollment(StudentId, CourseId), the maximum-length attributes on string fields, the required attributes on key fields, and the range constraints on numeric fields such as MaxPoints on assignments and the grade on submissions. The seeding logic in DbInitializer is documented because it provides the reproducible demonstration data needed for the viva.

The author has clearly recorded that UniManage uses MySQL through the Pomelo provider rather than SQL Server / LocalDB. Where the brief refers to SQL Server / LocalDB, the author has documented the alternative approach in honest terms instead of overclaiming.

9. Contribution to User Registration and Login

The author documented the registration, login, logout, password reset, and optional Google login flows implemented in Controllers/AccountController.cs. The contribution links each action to its view, view model, and Identity API. The author has also documented the security configuration in Program.cs that supports the flow, including the strong password rules (minimum eight characters, at least one digit, at least one uppercase letter, at least one lowercase letter), the unique email constraint, the lockout policy (five failed attempts, fifteen minute lockout), and the application cookie configuration (HTTP-only, SameSiteMode.Lax, secure when the request is secure, sliding expiration of eight hours).

The author has identified the audit log entries written by the registration and login flows. Successful registrations write a positive audit entry, failed registrations record the validation failure, successful logins write a positive audit entry, and failed logins write a negative audit entry. This combination supports the marker's review by providing a concrete record of who registered or signed in, when, and whether the action succeeded.

10. Contribution to Role-Based Access Control

The author documented the three-layer approach to role-based access control. First, Models/AppRoles.cs defines the role name constants (Administrator, Lecturer, Student). Second, Data/DbInitializer.cs seeds the three roles and creates the three demonstration users. Third, controller actions use [Authorize(Roles = AppRoles.X)] to restrict access to the appropriate role. Where the role check is not enough, controller actions add an ownership filter through LecturerId == userId or StudentId == userId so that a lecturer cannot see another lecturer's data and a student cannot see another student's data.

The author has also documented the Views/Account/AccessDenied.cshtml page, which presents a friendly response when a user tries to open an action they are not authorised to use. The combination of role attributes, ownership filters, and the access denied page provides defence in depth at the application layer.

11. Contribution to Student Dashboard

The author documented the student dashboard implementation through DashboardController.Student, the StudentDashboardViewModel defined in ViewModels/DashboardViewModels.cs, and Views/Dashboard/Student.cshtml. The dashboard view model exposes enrolled courses, upcoming deadlines, recent grades, recent messages, calendar entries, and meetings. EF Core read queries use AsNoTracking where possible to reduce the change tracker overhead, and queries are scoped to the signed-in student's user identifier so that no cross-account data is exposed.

The author has identified the supporting partial Views/Shared/_DashboardCalendar.cshtml, which renders a small calendar block reused by the dashboards. This partial is one example of the maintainability practices documented in Section 14 of the main report.

12. Contribution to Lecturer Dashboard

The author documented the lecturer dashboard implementation through DashboardController.Lecturer, the LecturerDashboardViewModel, and Views/Dashboard/Lecturer.cshtml. The lecturer dashboard view model exposes the courses owned by the signed-in lecturer, enrolment counts, assignment counts, pending submissions, recently graded submissions, recent messages, recent meetings, and recent audit activity relevant to the lecturer's work. The author has identified that the dashboard uses ownership filters to ensure that the lecturer only sees their own data and is therefore consistent with the role-based access control approach documented in Section 10 of this contribution report.

13. Contribution to Administrator Dashboard and User Management

The author documented the administrator dashboard implementation through DashboardController.Administrator, the AdminDashboardViewModel, and Views/Dashboard/Administrator.cshtml. The administrator dashboard exposes counts for users, courses, enrolments, assignments, submissions, and messages, identifies popular courses by enrolment count, and presents recent audit log entries.

The author also covered the Admin User Management feature implemented through AdminUsersController, AdminUserManagementViewModels, and Views/AdminUsers/. The Administrator can review registered users, create accounts, edit account details and roles, lock or unlock accounts, and delete non-administrator users where allowed by the controller. The controller blocks unsafe actions such as self-lock, self-delete, administrator deletion, last-admin changes, and unsafe administrator locking. This strengthens both the Administrator Dashboard and Security and Data Protection marking items.

14. Contribution to Course Management

The author documented the course management implementation through Controllers/CoursesController.cs, the Course entity, the CourseInputViewModel, the MaterialUploadViewModel, and the Views/Courses/ folder. The actions documented include Index with search and paging, Details, Create, Edit, Delete, MyCourses, and UploadMaterial. The author has documented the unique course code constraint at the database level, the prerequisite handling, the lecturer assignment, the capacity limit, and the file upload pipeline through Infrastructure/UploadedFileStore.cs.

The author has also documented that the upload action is gated by the upload rate-limiting policy configured in Program.cs, which permits up to thirty uploads per minute per user. This protects the file store against abuse without preventing reasonable academic use.

15. Contribution to Enrolment System

The author documented the enrolment workflow in Controllers/EnrollmentsController.cs, including the Browse action that lists eligible courses for the signed-in student and the Enroll action that creates an enrolment record after the business rules have been checked. The business rules are: the course exists, the course is open (capacity has not been reached), the student has

not already enrolled, and any prerequisite has been satisfied. The combination of the unique index on Enrollment(StudentId, CourseId) and the application-level duplicate check prevents duplicates by defence in depth.

The author has produced eligibility logic that uses a single composed LINQ query rather than multiple queries to keep the database round-trip count low. The author has also documented the audit log entries that record successful and unsuccessful enrolment attempts.

16. Contribution to Assignment, Submission and Grading

The author documented the assignment lifecycle in Controllers/AssignmentsController.cs, including ForCourse, Mine, Create, Edit, and Delete, with ownership checks that ensure a lecturer can only manage assignments for their own courses. The submission lifecycle is documented in Controllers/SubmissionsController.cs, including Submit, Grade, ForAssignment, CourseMaterialFile, and SubmissionFile. The author has documented the file validation through Infrastructure/UploadedFileStore.cs, including extension allow-listing, size limits, and stored file naming to prevent path traversal.

The grading workflow is documented through GradeSubmissionViewModel, the score range validation, the feedback length validation, and the graded lock that prevents a student from modifying a submission after it has been graded. Each grading action writes an audit log entry that records the lecturer, the submission, and the grade.

17. Contribution to Reporting and Analytics

The author documented the reporting suite in Controllers/ReportsController.cs, including six reports: course popularity, student performance, lecturer workload, enrolments, pass and fail, and assignment attendance. Each report uses an EF Core query with GroupBy and aggregate functions and projects into a report view model. CSV export is provided through Infrastructure/CsvWriter.cs, which writes properly escaped CSV content. PDF export is provided through Infrastructure/PdfReport.cs, which uses QuestPDF to render the report layout. The author has identified that the QuestPDF licence type is set to Community in Program.cs, which keeps the project within the QuestPDF licensing terms.

18. Contribution to Communication and Meetings

The author documented the messaging implementation in Controllers/MessagesController.cs, including Inbox, Thread, Reply, Compose, and MarkAllRead. The allowed-recipient rule is documented: a Student may message a Lecturer who teaches a course on which the Student is enrolled, and a Lecturer may message a Student enrolled on an owned course. The author has identified that this rule is implemented at the database level through a join during the recipient look-up, so the user interface does not offer an unauthorised recipient.

The author has also documented the meeting implementation in Controllers/MeetingsController.cs, including Index, Create, Edit, Delete, Join, and Ics. The ICS export is provided through Infrastructure/CalendarLink.cs, which produces an iCalendar file for import into a personal calendar tool. The meeting URL is stored as part of the Meeting entity rather than being provisioned by the application; this is consistent with the documented scope.

19. Contribution to Security, Validation and Error Handling

The author documented the security pipeline in detail. Identity uses strong password rules and lockout. Anti-forgery is enabled globally through AddAntiforgery in Program.cs, with the cookie configured as HTTP-only, SameSiteMode.Strict, and secure when the request is secure. Rate limiting is configured with two named policies, auth (ten requests per minute per IP) and upload (thirty requests per minute per user). The custom Infrastructure/SecurityHeadersMiddleware.cs adds standard headers to every response. HTTPS redirection and HSTS are enabled. The application cookie is HTTP-only with SameSiteMode.Lax and secure when the request is secure. The author also documented environment-based configuration for database, Google OAuth and SMTP settings, with real secrets kept out of the public repository and screenshots.

Validation is documented through data annotations on view models, ModelState checks in controller actions, server-side ownership checks where role membership is not enough, and file upload validation through UploadedFileStore. Error handling is documented through the production exception handler app.UseExceptionHandler("/Home/Error"), the developer exception page in development, and the friendly Views/Shared/Error.cshtml page. The author has also documented the audit logger as a key part of the error and event-tracking story.

20. Contribution to UI/UX and Usability

The author documented the user interface design through Views/Shared/_Layout.cshtml, Views/Shared/_Alerts.cshtml, Views/Shared/_DashboardCalendar.cshtml, the role-specific dashboard views, the form views, and the static assets in wwwroot/css/site.css and wwwroot/js/site.js. The author has documented the role-aware navigation that adapts the menu items to the signed-in user's role, the alert partial that displays TempData feedback, the responsive Bootstrap grid that supports narrower viewports, the mobile viewport check, the skip link near the top of the layout that helps keyboard users, and the label association used throughout the form views to support assistive technology. A formal WCAG 2.1 AA audit has not been carried out and is treated as a future accessibility improvement.

21. Contribution to Testing and Debugging

The author prepared a manual testing strategy with thirty-seven test cases (T1 to T37) in Section 15.9 of the main report. The cases cover the main coursework workflows, including authentication, dashboards, courses, enrolment, assignments, submissions, grading, reports, messages, meetings, Admin User Management, validation, security checks, responsive layout, and build verification. T32 is confirmed by dotnet build --no-restore returning zero warnings and zero errors. The other cases still need final screenshots or manual result evidence before upload.

Debugging during development included resolving transient MySQL connection errors at startup by enabling EF Core retry on failure, splitting a single rate limiter policy into two named policies (auth and upload) to avoid spurious 429 responses on dashboard pages, ordering the migrations so that the audit table exists before any audit calls run, and wrapping the audit write in a try/catch block so that an audit failure cannot crash the calling action. Each problem and its resolution is documented in Section 17.1 of the main report.

22. Contribution to Documentation and User Manual

The author prepared the documentation package at documentation_application_development/. The package contains the main report, contribution report, user manual, final evidence summary, source/build links, diagram sources, screenshot evidence, supporting appendices, and final Word/PDF export. These files support the report, appendices, and viva preparation.

The documents use UK English, neutral individual-submission wording, no em dashes, and no unnecessary symbols. The body of the technical report mainly uses "the author", while the reflection chapter is allowed to sound more personal.

23. Contribution to Evidence Preparation and Viva Readiness

The evidence package was prepared by listing the screenshots, diagram exports, source-code listings, database evidence, build evidence, and viva preparation items. Appendix B lists the screenshot evidence, Appendix C lists the diagram exports, and Appendix T summarises the final evidence package.

The viva demonstration script in Appendix H starts with the solution open in Visual Studio, includes a build verification, demonstrates each role with seeded users, triggers a validation error, shows an audit log entry, and ends with a short summary of items that still need final evidence.

The seeded demonstration users that support the viva are admin@gmail.com (Administrator), lecturer@gmail.com (Lecturer), and lms@gmail.com (Student). The seeded passwords are documented in Data/DbInitializer.cs and in the user manual.

24. Challenges, Learning and Final Preparation

The author faced a small set of repeated challenges. One was aligning the coursework wording with the actual project because UniManage uses Pomelo MySQL rather than SQL Server / LocalDB. This was handled by explaining the supported database approach honestly. Another challenge was evidence preparation, because screenshots need to be redacted, readable, and named consistently. Appendix B and the screenshot evidence helped keep that work organised. A further challenge was keeping the writing consistent across a long set of documents without making it sound mechanical.

The project gave the author practical experience with MVC routing, Razor views, EF Core with Pomelo, Identity, anti-forgery, rate limiting, custom middleware, dependency injection, asynchronous programming, and PDF generation through QuestPDF. It also gave the author experience in evidence-based academic writing and in marking unclear claims instead of overstating them. Completing the work individually also helped the author build confidence in managing a substantial academic deadline.

The final preparation stage focuses on capturing screenshots, including Admin User Management and mobile responsive evidence, exporting Mermaid diagrams, checking title page details and references, confirming links, and converting the report into Word and PDF for upload.

25. Supporting Evidence

Table IC2 lists the evidence prepared for assessment, the purpose of each item, the location, and the final evidence status.

Evidence Type	Purpose	Location	Final Evidence Status
Source code	Confirms implemented features and supports source-code traceability	Project root, including Controllers/, Models/, ViewModels/, Views/, Data/, Infrastructure/, wwwroot/, Program.cs, AD COURSEWORK 2.sln, AD COURSEWORK 2.csproj	Source/build link included
Build verification	Confirms that the project compiles cleanly	Build output figure and supporting evidence	Evidence attached in final report
Architecture and design diagrams	Supports Section 6 of the main report	Shared diagram folder and Figures 1 to 6	Included in final report
Screenshots	Supports figures referenced from the main report	Inserted screenshots in the main report and user manual	Evidence attached in final report

Evidence Type	Purpose	Location	Final Evidence Status
	and the user manual		
Final evidence summary	Summarises the status of required evidence items	Final evidence summary appendix	Included in supporting appendices
Source and build links	Records repository, build, and database evidence URLs	Source/build links appendix	Source/build link included
User manual	Supports the installation guide and user manual marking item	Installation Guide and User Manual	Included in final report
Final evidence summary for figures	Tracks figure-by-figure capture	Final List of Figures and Appendix B	Included where available
Database evidence	Supports the database design marking item	Database evidence figures and migration evidence	Evidence attached in final report
Audit log evidence	Supports the security and audit traceability story	Audit log review screen and security/audit sections	Included where available
Validation message evidence	Supports the validation and exception handling marking item	Validation and testing sections	Evidence attached in final report

Evidence Type	Purpose	Location	Final Evidence Status
Individual approval evidence	Supports the approved individual submission context	Approved individual submission context	Included where available; manual review only if required
Repository link	Supports submission packaging	Source/build links appendix	Source/build link included
Packaged build / hosted deployment	Supports submission packaging	Source/build links appendix and hosted demonstration section	Source/build and hosted demonstration links included
Final Word and PDF report	Supports the formal report submission	Final DOCX and PDF submission files	Final Word/PDF exported

Table IC2: Evidence Prepared for Assessment

26. Summary

The author completed UniManage as an approved individual submission for CS6004ES Application Development Coursework 2. The contribution covers requirements, architecture, database design, implementation, validation, security, interface work, testing, debugging, documentation, and evidence preparation. The technical baseline is a working ASP.NET Core MVC 8 application with thirteen controllers, twelve domain models, eighteen view models, thirteen infrastructure components, fifty Razor views, four EF Core migrations, and a Program.cs startup configuration that enables Identity, EF Core retry on failure, rate limiting, anti-forgery, security headers, HTTPS redirection, and HSTS.

Following approval to complete the coursework individually, the author took responsibility for the final implementation, testing, documentation, and submission preparation. The final package is supported by screenshots, diagrams, source/build links, database evidence, hosted demonstration evidence, and selected unit-test evidence.

End of individual contribution report.

Installation Guide and User Manual

Project: UniManage - University Course Management System (UCMS)

Module: CS6004ES Application Development

Assessment: Coursework 2

Student: Yasas Pasindu Fernando

Student ID: 25026764

Submission Type: Approved individual submission

Companion Documents: Main Application Development Report and Individual Contribution Report

1. Introduction

This document is the installation guide and user manual for UniManage, the University Course Management System developed for CS6004ES Application Development Coursework 2. It is written for the marker, external moderator, and any reader who needs to install, run, demonstrate, or operate the application. The steps are based on the actual project files and the local development setup.

The manual is organised into fourteen sections. Sections 1 to 5 cover requirements, package contents, installation, and first-time setup. Sections 6 to 9 explain login and the Student, Lecturer, and Administrator workflows. The later sections cover validation messages, reports, security notes, troubleshooting, responsive use, and the viva demonstration order.

Following approval to complete the coursework individually, the author completed the implementation, testing, documentation, and final preparation as an individual submission. The manual uses UK English and a practical step-by-step style.

2. System Requirements

The following items are required to install and run UniManage on a development workstation.

Item	Requirement	Notes
Operating system	Windows 10 or later	The development environment is Windows; macOS and Linux can run the application using the .NET 8 SDK and Pomelo MySQL provider, but Visual Studio integration is best on Windows.
Development environment	Visual Studio 2022 (Community edition is sufficient)	Visual Studio 2022 or a compatible version is recommended because the project targets .NET 8.
.NET runtime	.NET 8 SDK	Required for dotnet build, dotnet run, and the Visual Studio project.
Database engine	MySQL Server 8.x	The project uses the Pomelo Entity Framework Core MySQL provider.
Database tool	Optional MySQL Workbench, HeidiSQL, or DBeaver	Used to inspect tables and capture database evidence.
Browser	Microsoft Edge, Google Chrome, or Mozilla Firefox (latest stable)	Used to access the running application.
Internet connection	Required for the first run	Used to restore NuGet packages, optional Google login, optional SMTP email, and the small number of CDN-hosted assets.
Disk space	About 1 GB free	Project source, NuGet cache, and database storage.

If the marker uses SQL Server / LocalDB instead of MySQL, the connection string and the EF Core provider must be replaced. The current implementation uses MySQL through Pomelo and is therefore the supported configuration. SQL Server / LocalDB compatibility is treated as a future compatibility improvement.

3. Project Package Contents

The project package contains the following items.

Item	Path	Purpose
Solution file	AD COURSEWORK 2.sln	Visual Studio solution file.
Project file	AD COURSEWORK 2.csproj	Web project file with package references.
Application entry point	Program.cs	Service registration, middleware pipeline, and seeding.
Controllers	Controllers/	Thirteen controllers handling the request flow.
Domain models	Models/	Twelve model classes for academic entities.
View models	ViewModels/	Eighteen view models for forms and dashboards.
Razor views	Views/	Fifty .cshtml views including shared layouts and partials.
Data access	Data/	ApplicationDbContext, factory, seeder, and EF Core migrations.
Infrastructure	Infrastructure/	Thirteen infrastructure files: audit logger, email service, file store, security headers middleware, CSV writer, PDF report builder, calendar link helper, meeting link helper, grading helper, and supporting interfaces.
Static assets	wwwroot/	Bootstrap, custom CSS, custom JavaScript, library folder, and images.
Configuration	appsettings.json, appsettings.Development.json	Connection string, email settings, optional Google credentials.
Launch settings	Properties/launchSettings.json	HTTP and HTTPS launch profiles for development.
Documentation	documentation_application_development/	Main report, individual contribution report, user manual, final evidence summary, source/build

Item	Path	Purpose
		links, diagrams, screenshots, and supporting appendices.
Project README	README.md	Top-level project description.

4. Installation Guide

The installation guide assumes that the marker has Windows, Visual Studio 2022, the .NET 8 SDK, and a running MySQL 8 server. The steps are kept short so that the application can be opened, configured, built, and run without unnecessary detours.

4.1 Extracting the Project

If the project has been delivered as a ZIP archive, extract the archive into a short local path such as C:\src\unimanage or C:\workspace\AD COURSEWORK 2. Avoid placing the project under a deep path with spaces and special characters because some EF Core tooling commands behave better with shorter paths.

If the project has been delivered as a Git repository, clone the repository into a similar local path:

git clone https://github.com/YasasPasinduFernando/AD-COURSEWORK-2.git
After extraction or cloning, the local folder should contain AD COURSEWORK 2.sln, AD COURSEWORK 2.csproj, Program.cs, the source folders listed in Section 3, and the documentation_application_development folder.

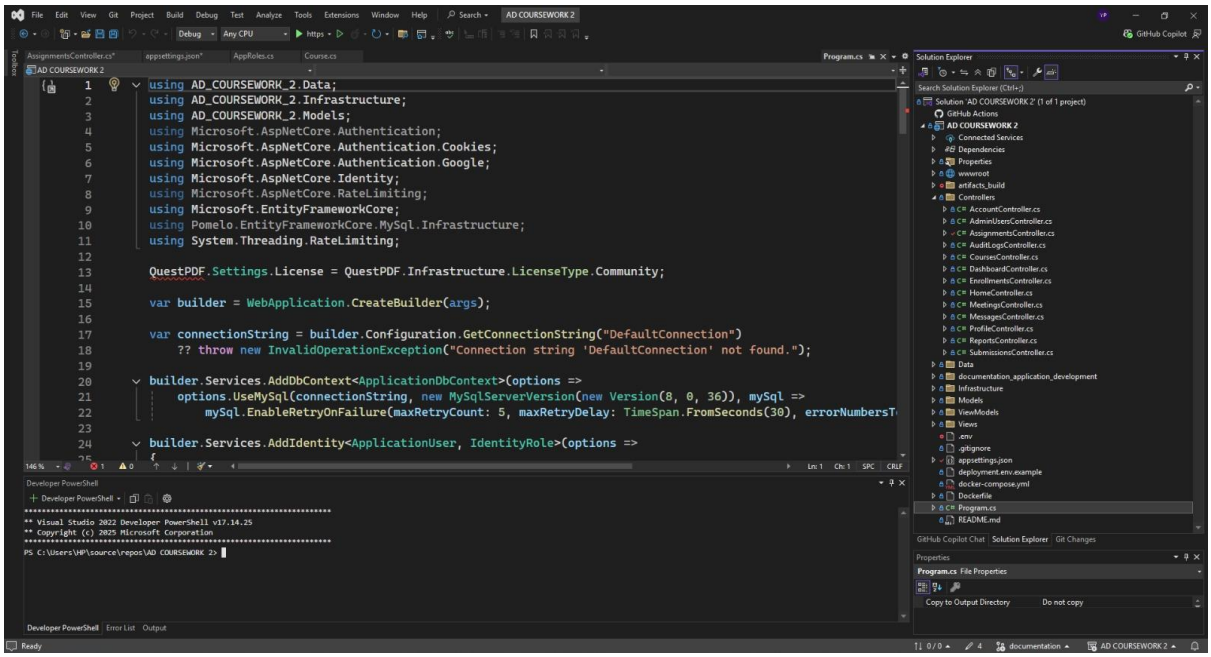


Figure UM1: The solution loaded

This screenshot shows the solution loaded in Visual Studio with the source folders visible in Solution Explorer.

4.2 Opening the Solution in Visual Studio

Double-click AD COURSEWORK 2.sln to open the solution in Visual Studio 2022. Confirm that the project loads without missing project errors. The status bar should report that the solution is ready and the Solution Explorer should display the AD COURSEWORK 2 web project together with the documentation_application_development folder if it is included in the solution.

If Visual Studio prompts to install missing components, accept the prompts so that the .NET 8 SDK and the ASP.NET Core workload are present. If the prompt suggests SQL Server LocalDB, that component is not required because the project uses MySQL.

4.3 Restoring NuGet Packages

NuGet packages are restored automatically the first time Visual Studio opens the solution. To restore manually from the command line, open a PowerShell window in the project folder and run:

```
dotnet restore
```

The main packages used by the project include:

Microsoft.AspNetCore.Identity.EntityFrameworkCore

Microsoft.EntityFrameworkCore and Microsoft.EntityFrameworkCore.Design

Pomelo.EntityFrameworkCore.MySql

Microsoft.AspNetCore.Authentication.Google

QuestPDF

If a restore failure occurs, confirm that NuGet.org is reachable and that the workstation has internet access. The exact pinned versions are recorded in AD COURSEWORK 2.csproj.

4.4 Checking appsettings.json

Open appsettings.json in the project root. Confirm the following sections.

ConnectionStrings:DefaultConnection contains the MySQL server, port, database name, user, and password to use during development.

Email contains the SMTP host, port, sender name, sender email, username, and password used by the password reset flow. If email is not required for the marker's environment, the application still runs because email is only sent during password reset.

Authentication:Google contains the optional Google client identifier and client secret. If both values are present, Google login is enabled in Program.cs. If either value is empty, Google login is disabled and the standard Identity login is used.

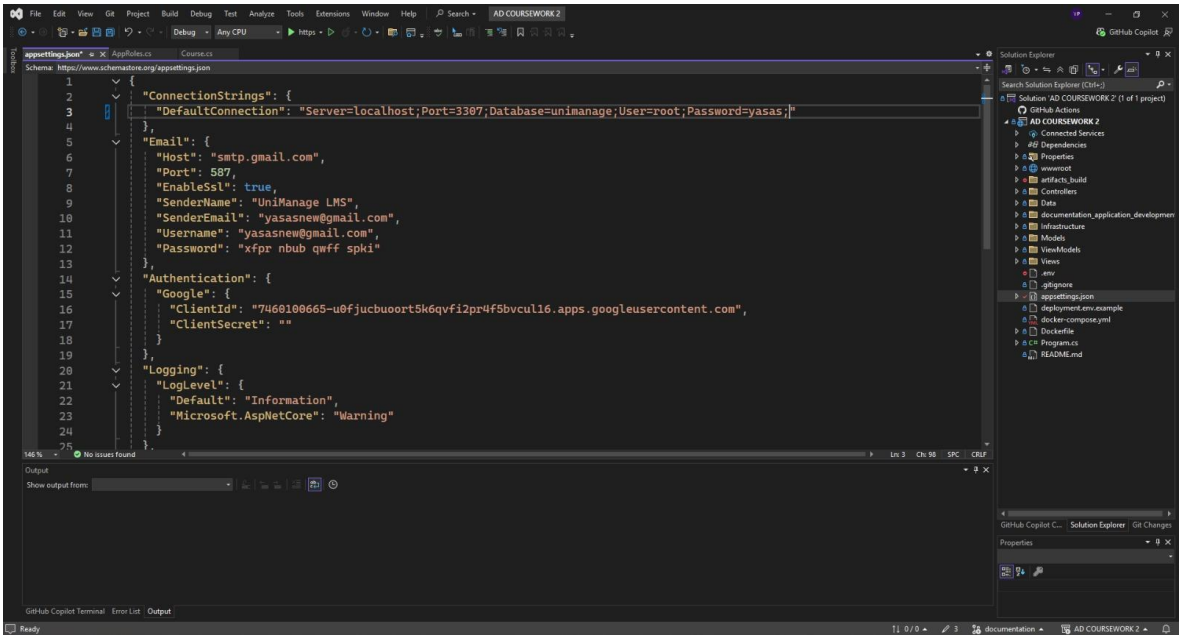


Figure UM2: The configuration file

This screenshot shows the configuration file structure used by the application.

Sensitive credentials such as database, SMTP and Google OAuth values are excluded from the report and shared screenshots. Configuration evidence uses redacted or non-sensitive values.

4.5 Configuring the Database Connection

The default connection string targets a local MySQL server on a non-standard port. The format is:

```
Server=localhost;Port=3307;Database=unimanage;User=root;Password=YOUR_PASSWORD
;TreatTinyAsBoolean=true;CharSet=utf8mb4;
```

To adapt the connection string to the marker's machine, edit the Server, Port, Database, User, and Password parts as appropriate. If the local MySQL server uses the default port, change 3307 to 3306. If a different database name is preferred, change unimanage to the chosen name; the application will create the schema during the first run.

The marker can also place the connection string into appsettings.Development.json so that the value is only used during development. Sensitive credentials must remain outside of source control and outside of the report.

4.6 Applying Migrations or Creating the Database

The project applies pending migrations automatically at startup through Data/DbInitializer.cs. As soon as the application runs, the schema is created and the demonstration roles, users, and sample data are seeded.

If the marker prefers to apply migrations manually before the first run, open a PowerShell window in the project folder and run:

```
dotnet ef database update
```

The current migrations are:

20260408041036_InitialUniManage
20260501122117_AddAuditLogAndSecurity
20260501125135_AddMeetings

20260506142007_AddApplicationUserDateOfBirth

To list the migrations from the command line, run:

```
dotnet ef migrations list
```

The output is the evidence captured for Figure 7b in the main report. Raw SQL backup scripts and SQL Server / LocalDB scripts are not included in the project; the EF Core migrations are the supported approach.

4.7 Building the Project

To build the project from the command line, run the following from the project folder:

```
dotnet build --no-restore
```

The expected output is:

```
Build succeeded.
0 Warning(s)
0 Error(s)
```

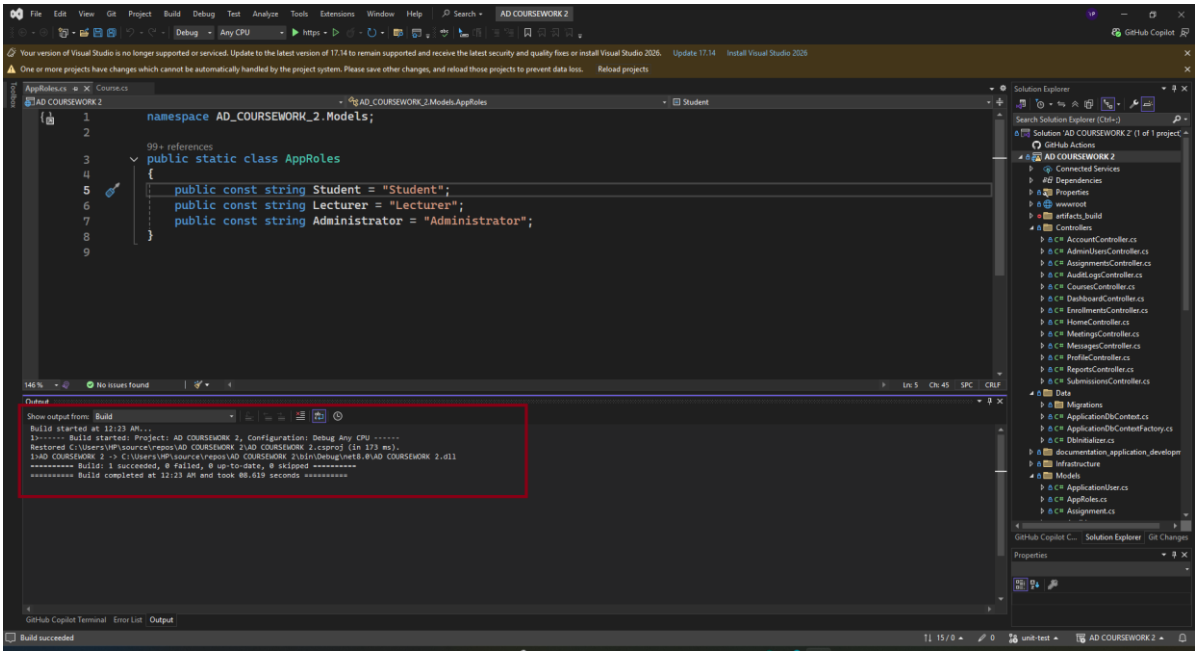


Figure UM3: Successful Build Output

This screenshot shows the Visual Studio build output with the build completed successfully and no failed projects.

To build inside Visual Studio, select Build, then Build Solution from the top menu. The Output window displays the same zero warning and zero error result.

4.8 Running the Application

To run the application from the command line, use:

```
dotnet run
```

To run inside Visual Studio, press F5 or click the Start button on the main toolbar. The Visual Studio launch profile uses the HTTPS profile by default.

The detected launch URLs are:

https://localhost:7212

http://localhost:5103

The first run applies the migrations, seeds the demonstration roles and users, and starts listening on the launch URLs. Open a browser and navigate to the HTTPS URL. The home page is displayed. From there, the user can log in or register.

4.9 Troubleshooting Build Errors

Error	Likely cause	Suggested fix
The .NET 8 SDK is missing	Visual Studio components are out of date	Open Visual Studio Installer, modify the workload, and add the .NET 8 SDK.
NuGet restore fails	No internet access or NuGet feed not reachable	Confirm internet access, run dotnet nuget list source, and add https://api.nuget.org/v3/index.json if missing.
EF Core tools not found	Global tool not installed	Run dotnet tool install --global dotnet-ef.
Database connection refused	MySQL not running or wrong port	Start the MySQL service and confirm the port matches the connection string.
Access denied for the database user	Wrong user or password	Recreate the user, grant rights to the unimanage database, and update the connection string.
Migration fails	Database user does not have CREATE privilege	Grant the user the necessary privileges or use a higher-privilege user for the first run.
dotnet build fails	Stale obj/bin folders or wrong .NET SDK	Run dotnet clean and dotnet build --no-restore again.
Port already in use	Another process is using 5103 or 7212	Stop the other process or change the port in Properties/launchSettings.json.

Error	Likely cause	Suggested fix
Static files not served	wwwroot missing or build skipped	Confirm that the wwwroot folder is present and that the build completed.
HTTPS certificate trust prompt	Development certificate not trusted	Run dotnet dev-certs https --trust once on the workstation.

5. First-Time Setup

The first time the application runs, DbInitializer creates three roles (Administrator, Lecturer, Student) and three demonstration users. Sample courses, enrolments, and assignments may also be created where the seeder has been configured to do so. The marker can sign in straight away using the demonstration credentials listed in Section 6.

If the seeded users are not visible after the first run, restart the application or check the logs for an error during seeding. The seeder is idempotent, so restarting the application should not create duplicate users, modules, enrolments, assignments, submissions, or meetings.

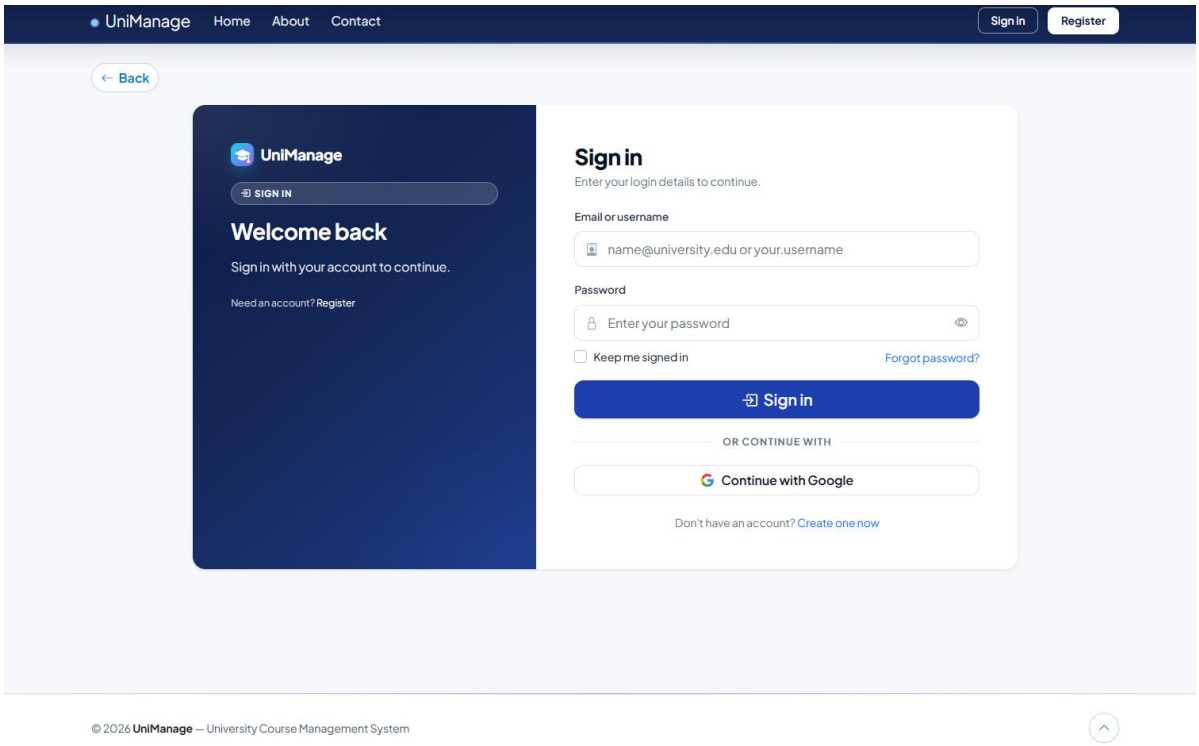


Figure UM4: Login Page

This screenshot shows the login page after the first run.

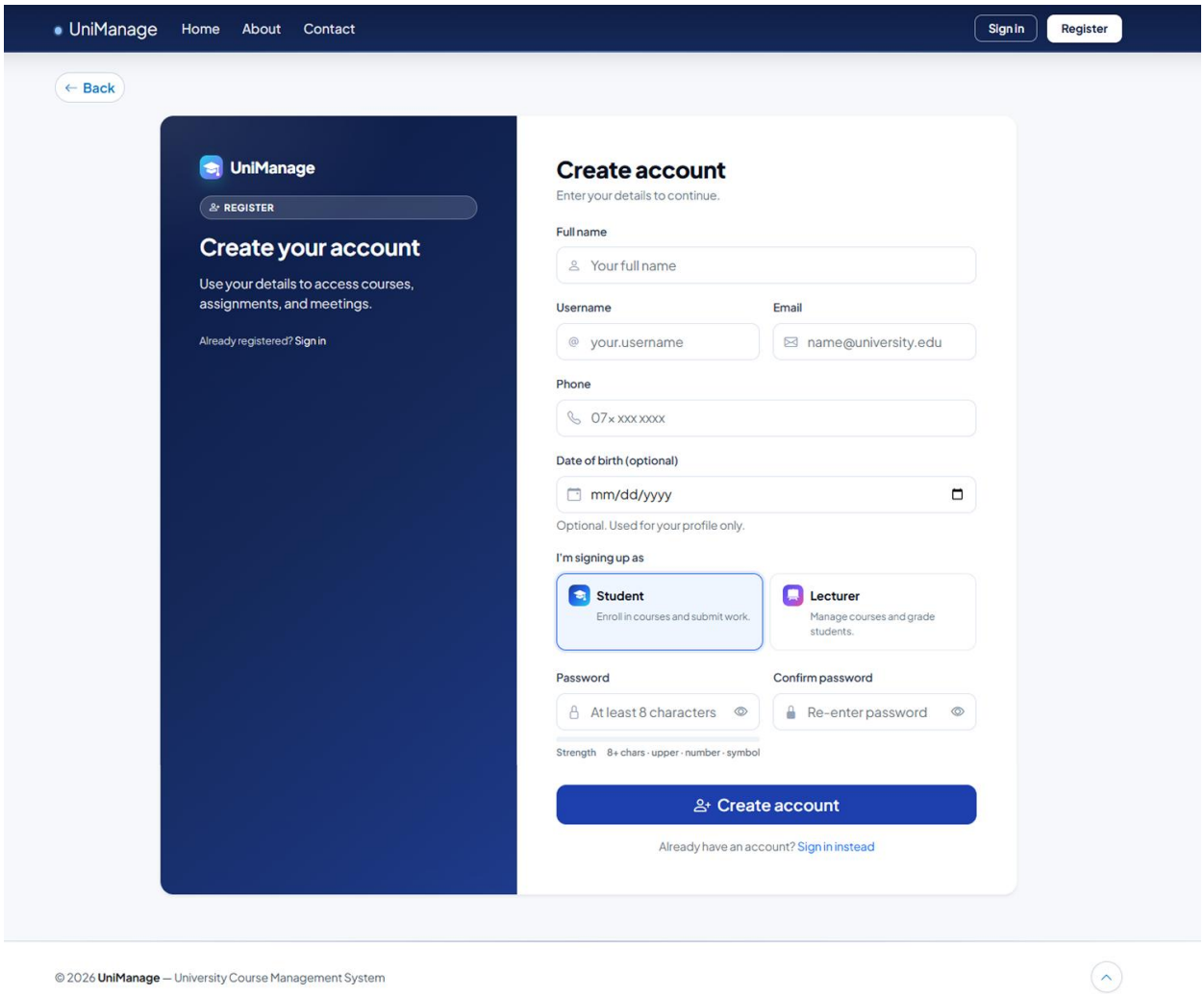


Figure UM5: User Registration Page

This screenshot shows the registration page with the role selector visible.

5A. Hosted Demonstration Access

In addition to the local installation route, a supporting hosted demonstration was prepared on Google Cloud. The custom domain is https://lms.yasaboy.com/, with hosted application evidence in Figure UM19 and DNS/server evidence in Figures UM21 and UM22. The hosted route can support viva and marker review, but it is supporting evidence only. The local Visual Studio route remains the primary assessment path.

The hosted URL is provided to make review and testing easier. If the hosted service is unavailable, the local Visual Studio installation and run method remains the primary assessment route.

5A.1 Hosted URL

The primary hosted demonstration URL is:

https://lms.yasaboy.com/

The earlier IP-based endpoint, which was used during the initial deployment phase and is retained as a backup in case the custom-domain endpoint is temporarily unreachable, is:

http://34.171.6.234:8080

The custom domain was configured for the hosted demonstration version. HTTPS/domain access was used where available during final testing. The earlier IP-based URL is kept only as fallback evidence, and the local Visual Studio route remains the main assessment route.

5A.2 Browser Access Steps

Open a modern browser such as Microsoft Edge, Google Chrome, or Mozilla Firefox.
Go to https://lms.yasaboy.com/.
Confirm that the UniManage homepage loads. If the page is correct, the address bar reflects the HTTPS or HTTP state for that deployment route.

Use Sign in or Register depending on the demonstration task. The seeded demonstration credentials listed in Section 6 of this manual are sufficient.

If the hosted URL is unavailable, use the local Visual Studio run method described in Section 4 of this manual. The earlier IP-based endpoint http://34.171.6.234:8080 may also be tried as a temporary backup.

5A.3 Expected Landing Page

The expected landing page is the UniManage public home page rendered by Views/Home/Index.cshtml. The page offers two primary actions: "Create your account" linking to /Account/Register and "I already have an account" linking to /Account/Login. The footer shows the project name and the copyright line.

5A.4 Login on the Hosted Version

After the home page loads, the marker can sign in using the seeded demonstration users defined in Data/DbInitializer.cs. The credentials are listed in Section 6 of this manual. The hosted version uses the same role-based redirection as the local version, so signing in as the seeded administrator, lecturer, or student opens the appropriate dashboard.

No additional credentials are exposed beyond the seeded demonstration users. Real student data, real lecturer data, and production-style credentials are not used on the hosted demonstration.

5A.5 Availability Warning

The hosted demonstration depends on the Google Cloud virtual machine and Docker container being in a running state. Cloud instances can be stopped, restarted, or reconfigured between the time of writing and the marking session. Before relying on the hosted URL, the marker should open it in a browser. If the custom-domain URL does not open, the earlier IP endpoint may be tried as a fallback. If neither endpoint is reachable, use the local installation route in Section 4.

5A.6 Security Notes for the Hosted Demonstration

The intended custom-domain endpoint is HTTPS, but the wider deployment is still a coursework demonstration rather than a production service. The marker is asked not to enter real personal data on the demonstration site.

Database passwords, SMTP passwords, Google client secrets, SSH keys, DNS or provider credentials, and cloud service account keys are not included in this manual or in the body of the report. These values should be supplied only through local configuration or environment variables when running the application.

If the marker chooses to demonstrate the hosted login on screen, the seeded demonstration credentials are sufficient. There is no need to expose any other account information.

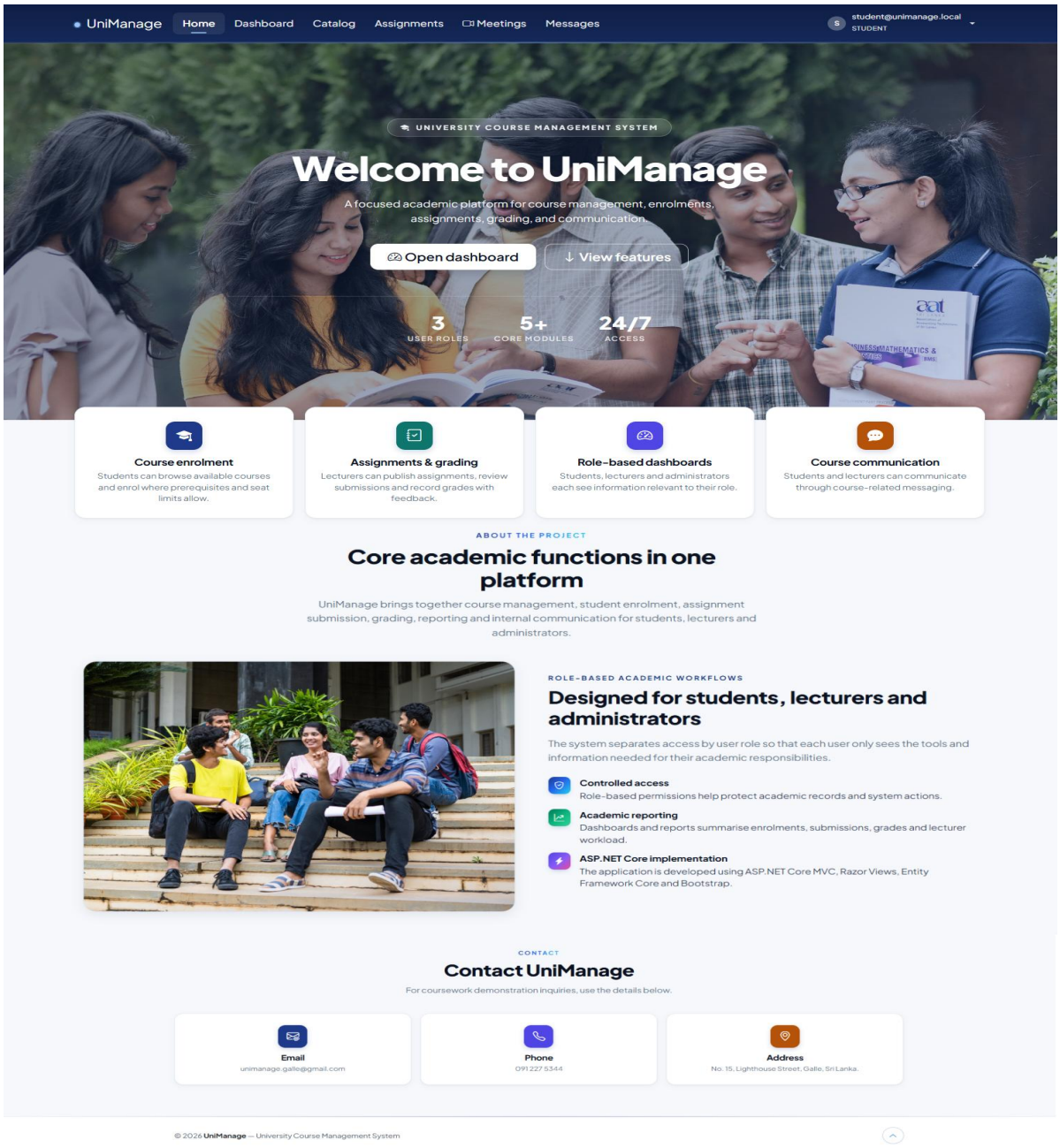


Figure UM19: Hosted UniManage Application

This screenshot shows the UniManage application running through the hosted custom domain.

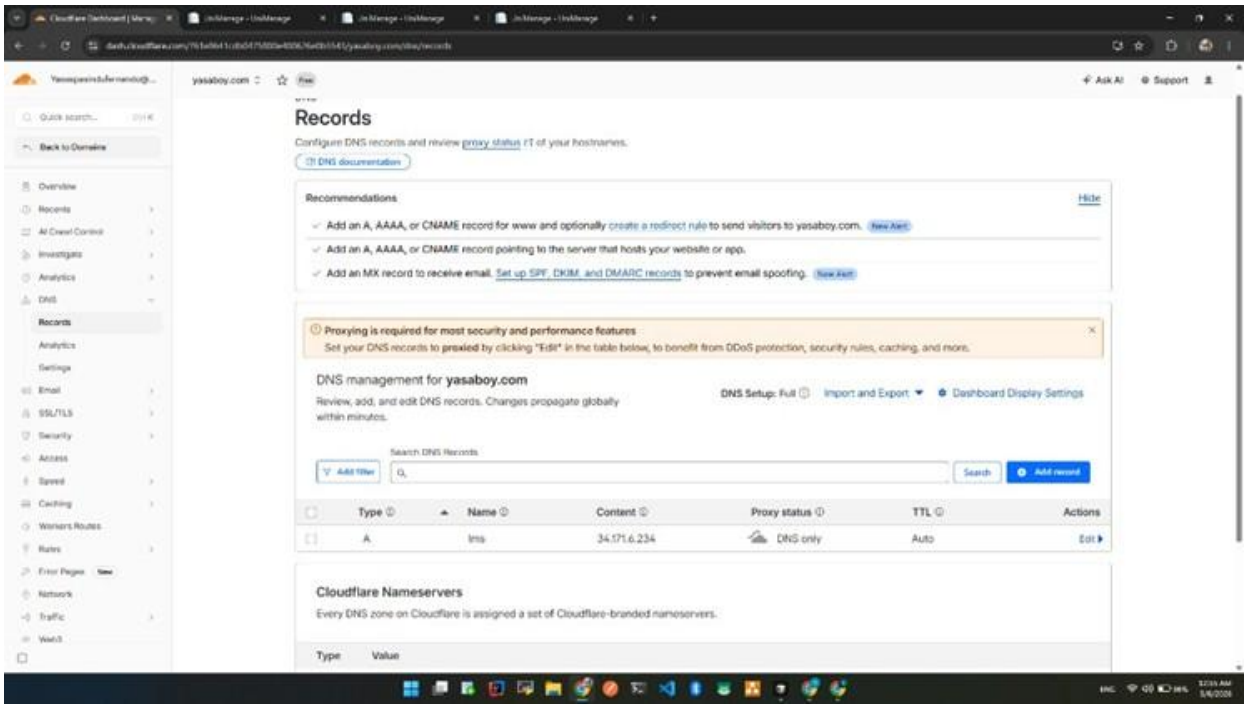


Figure UM21: Cloudflare DNS Record Evidence

This screenshot shows the Cloudflare DNS A record used to point the lms.yasaboy.com subdomain to the Google Cloud public IP address.

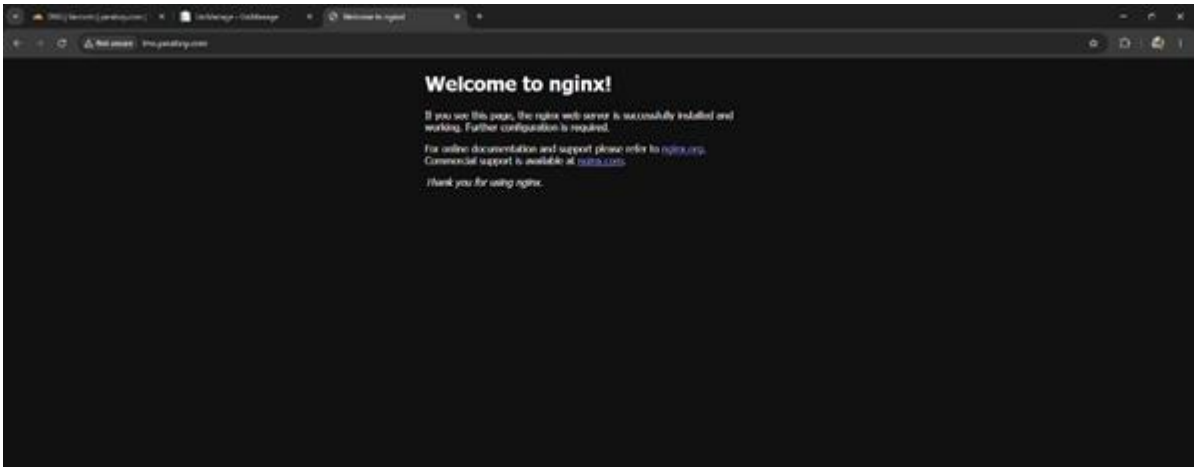


Figure UM22: Nginx Server Reachability Evidence

This screenshot shows the default Nginx page loading from the lms.yasaboy.com host, confirming that Nginx was installed and reachable on the server at the time of evidence capture.

6. Login and User Roles

The seeded demonstration accounts are created by Data/DbInitializer.cs. The credentials are intended for the viva and for marking, not for production use.

Role	Email	Password
Administrator	admin@gmail.com	Admin123!
Lecturer	lecturer@gmail.com	Lecturer123!
Student	lms@gmail.com	Student123!

To log in, open the application, navigate to the login page, enter one of the credentials above, and click Sign in. The system applies the password rules, the lockout policy (five failed attempts, fifteen minute lockout), and the rate limiter on the authentication endpoint (ten requests per minute per IP).

After a successful login, the user is redirected to the dashboard appropriate to their role. The Administrator goes to the Administrator dashboard, the Lecturer goes to the Lecturer dashboard, and the Student goes to the Student dashboard.

To register a new account, click the Register link on the login page. The user is asked for a full name, an email address, a phone number, a password, and a role (Student or Lecturer). The Administrator role is not available through self-registration. After a successful registration, the new user is signed in automatically and redirected to the role dashboard.

If Google OAuth is configured, the login page also allows Google sign-in. This option depends on the Google OAuth Client ID, Client Secret, and authorised redirect URI being configured for the current local or hosted URL. If these values are missing, Google sign-in is disabled by design and the normal seeded Identity credentials remain available.

7. Student User Guide

After logging in, the Student is redirected to the Student dashboard. The dashboard shows enrolled courses, upcoming deadlines, recent grades, recent messages, the calendar, and meetings.

The Student can perform the following actions.

View enrolled courses. The dashboard lists the Student's current enrolments. Each course title links to a detailed view.

Browse the course catalogue. Use the navigation menu to open the course browse page. The page lists eligible courses with their code, name, credits, lecturer, prerequisite, and current enrolment count.

Enrol on a course. Click the Enrol button next to an open course. The system checks that the course is not full, that the Student has not already enrolled, and that any prerequisite has been satisfied. If the checks pass, the enrolment is saved and a confirmation message is displayed. If a check fails, an error message explains the reason.

View assignments. Open the Assignments link in the navigation menu. The page lists assignments for the Student's enrolled courses with the due date and status.

Submit an assignment. Open an assignment, enter the submission text, optionally attach a file, and click Submit. The system validates the file extension, the file size, and the Student's enrolment, then stores the submission.

View grades and feedback. Open the assignment after grading. The Student can read the grade and the lecturer's feedback. After grading, the submission is locked and cannot be edited.

Send a message to a lecturer. Open the Messages page, click Compose, choose a Lecturer who teaches a course on which the Student is enrolled, enter a subject and content, and click Send. The system enforces the allowed-recipient rule.

View meetings. Open the Meetings page. Meetings linked to the Student's enrolled courses are listed with the scheduled time and the join link.

Update profile. Open the Profile page to view and update the full name, email, and phone number.

Change password. Open the Change Password page from the Profile menu, enter the current and new passwords, and click Save.

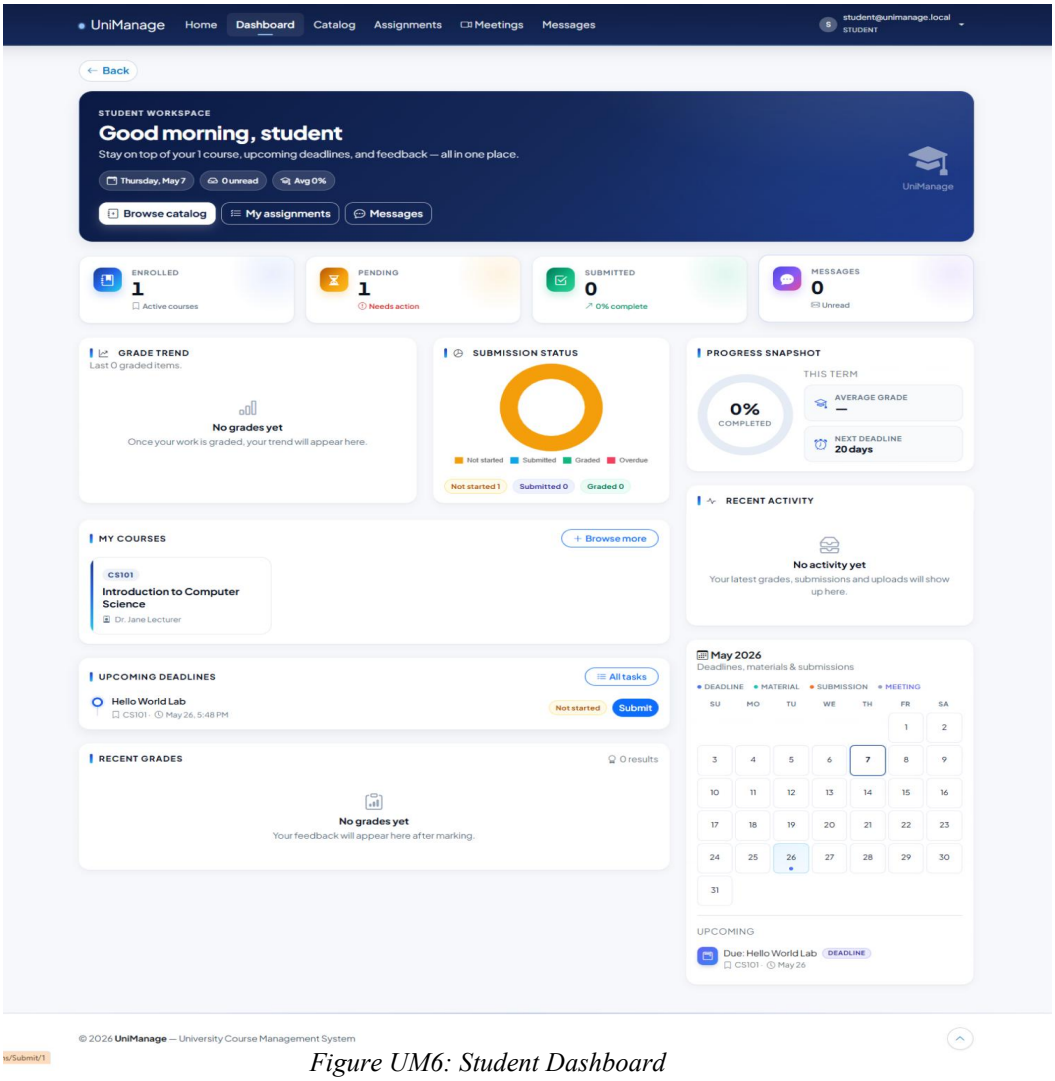


Figure UM6: Student Dashboard

This screenshot shows the Student dashboard after a successful login.

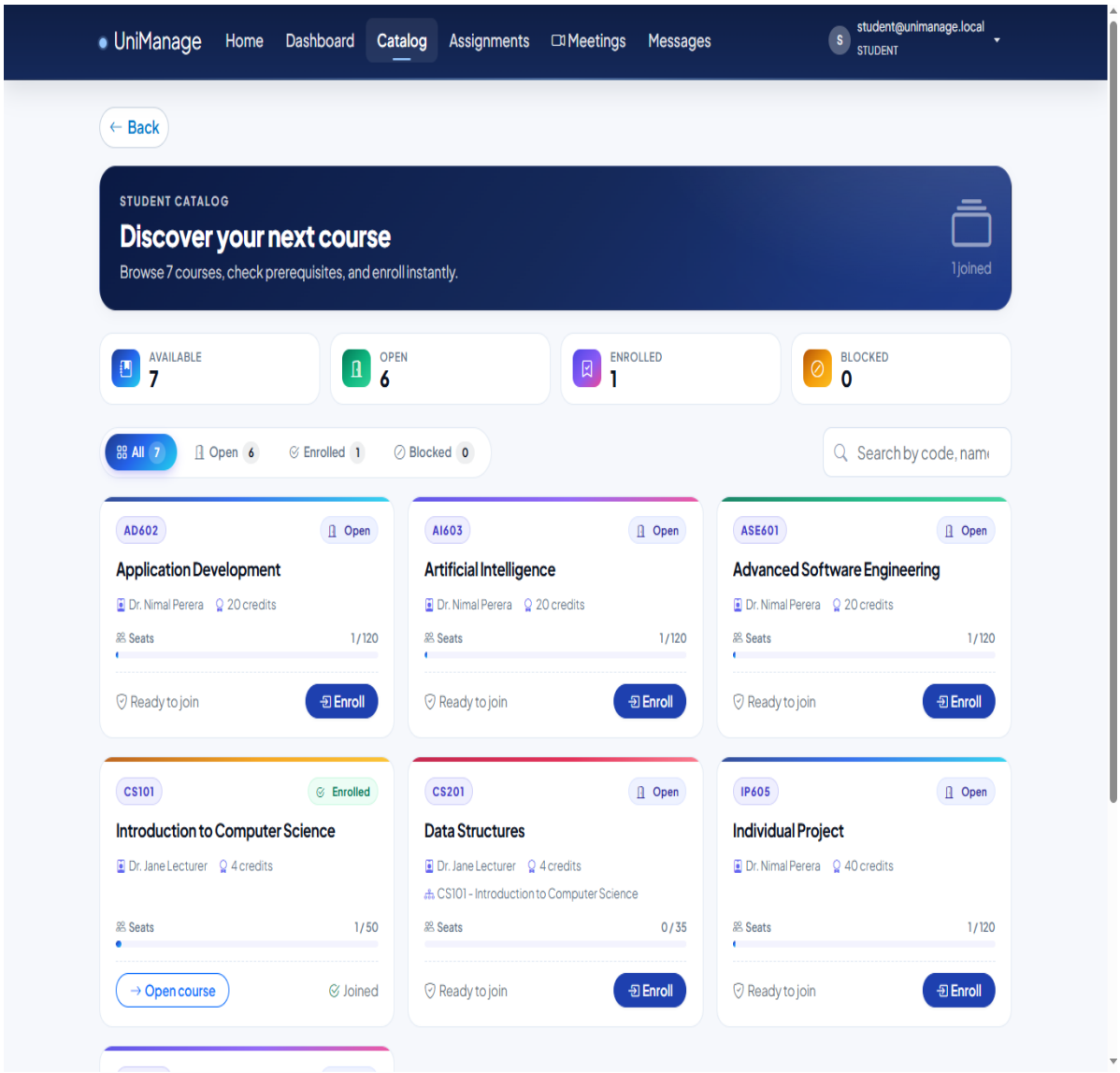


Figure UM10: Student Course Enrolment Page

This screenshot shows the course browse page with the Enroll button visible.

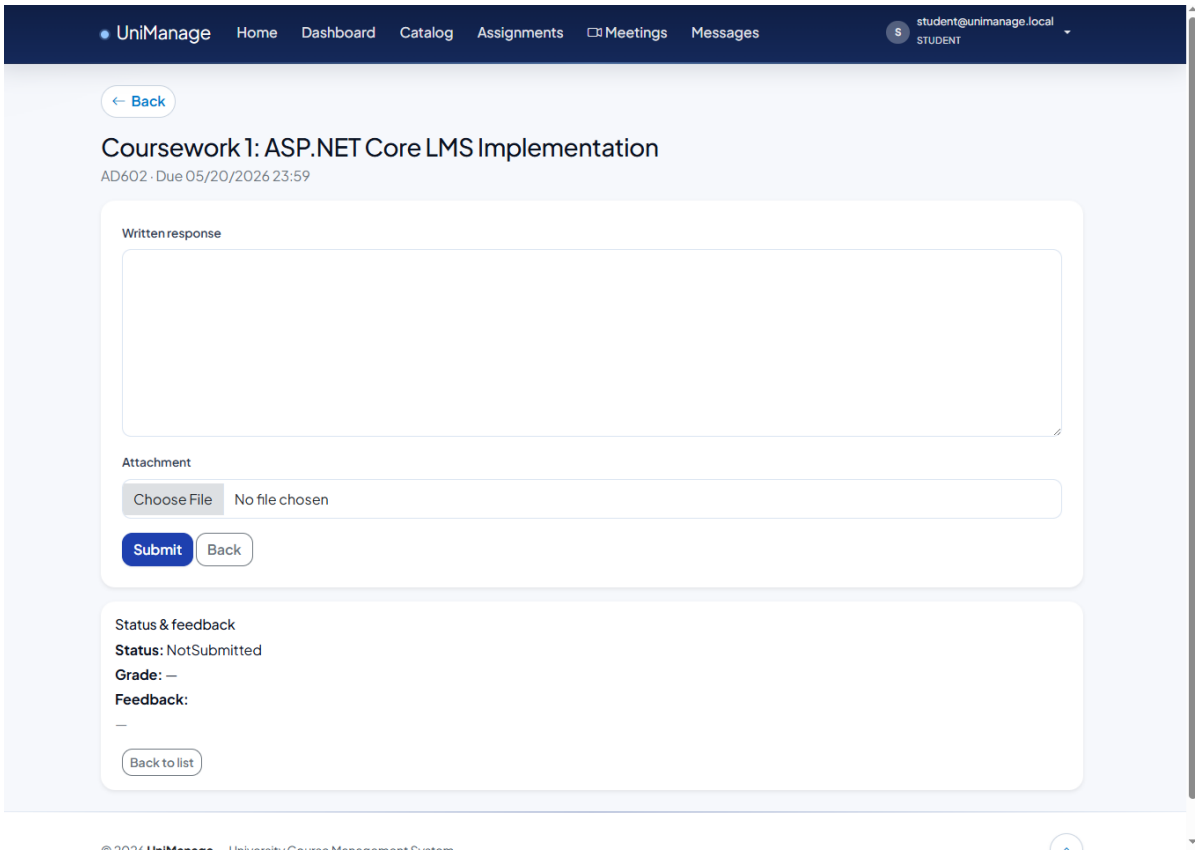


Figure UM12: Student Assignment Submission Page

This screenshot shows the assignment submission form with text and file fields.

8. Lecturer User Guide

After logging in, the Lecturer is redirected to the Lecturer dashboard. The dashboard shows the Lecturer's teaching courses, enrolment counts, assignment counts, pending submissions, recently graded submissions, recent messages, recent meetings, and recent activity.

The Lecturer can perform the following actions.

View taught courses. Open the My Courses page. The page lists only the courses owned by the signed-in Lecturer.

Open course details. Click a course to open its details, including the description, credits, enrolment limit, lecturer, prerequisite, enrolled students, assignments, materials, and meetings.

Upload course materials. From the course details page, click Upload Material, enter a title, select a file, and click Save. The system validates the file extension and size and stores the metadata in the database. The actual file is stored on disk through the upload service.

Create an assignment. Open the Assignments page for an owned course, click Create, and complete the form. The form requires a title, description, due date, and maximum points. After submission, the assignment is saved and listed.

Edit or delete an assignment. From the assignments list, click Edit or Delete next to an assignment. The system checks that the assignment belongs to a course owned by the Lecturer.

View submissions. Click an assignment to view its submissions. Each submission shows the Student, the submitted time, the file name where applicable, and the status.

Grade a submission. Click Grade next to a submission, enter a grade and feedback, and click Save. The system validates the grade range and the feedback length. After grading, the submission is locked and the Student can see the grade and feedback.

Send a message to a Student. Open the Messages page, click Compose, choose a Student enrolled on a course owned by the Lecturer, and send the message.

Schedule a meeting. Open the Meetings page, click Create, select a course owned by the Lecturer, enter a title, scheduled time, duration, and meeting URL, then click Save. The system validates the URL format and the schedule.

Edit or delete a meeting. From the meeting list, click Edit or Delete next to a meeting. The Lecturer can only manage meetings for owned courses.

Update profile and change password. Use the Profile menu in the same way as the Student.

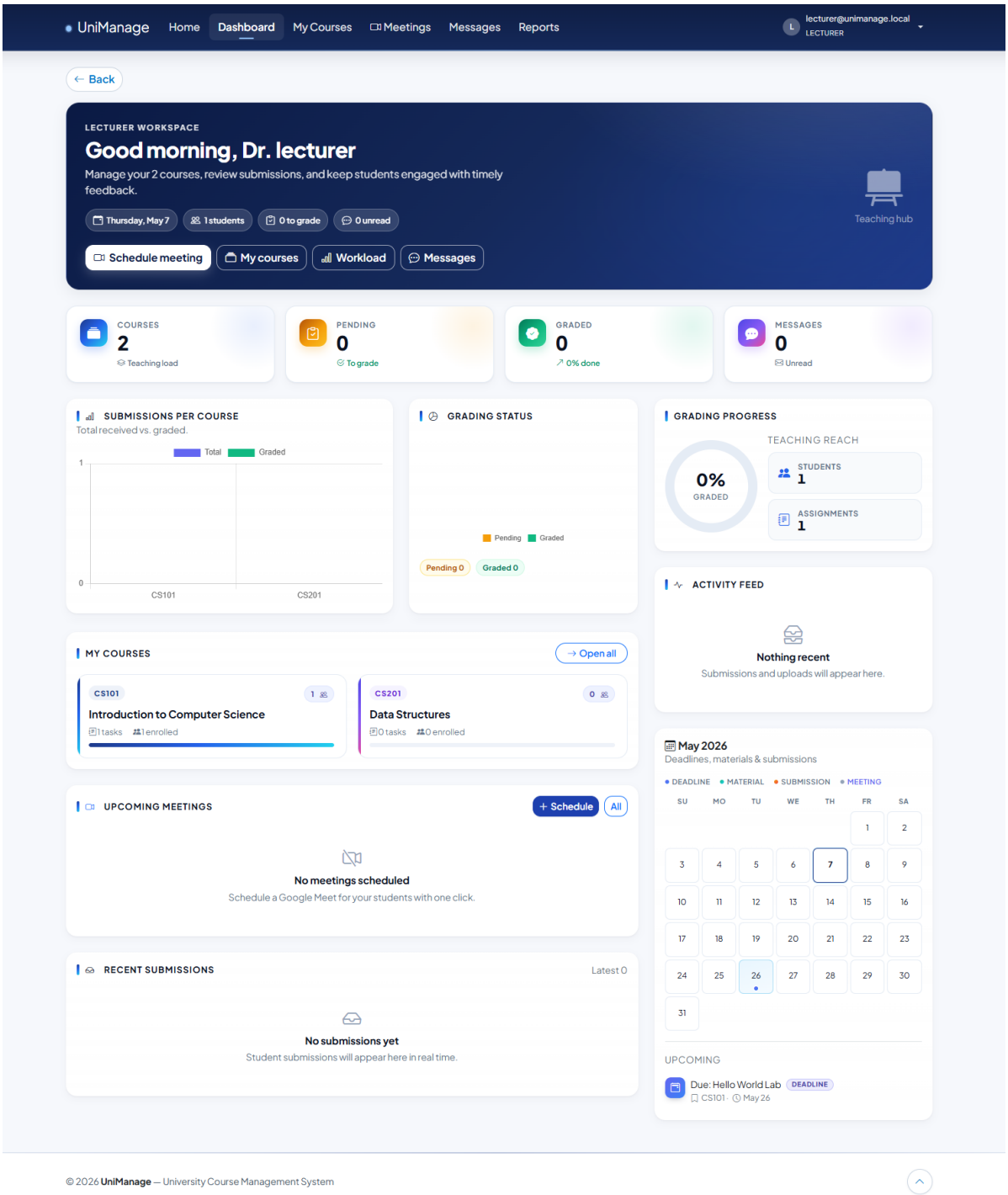


Figure UM7: Lecturer Dashboard

This screenshot shows the Lecturer dashboard with teaching summary and pending submissions.

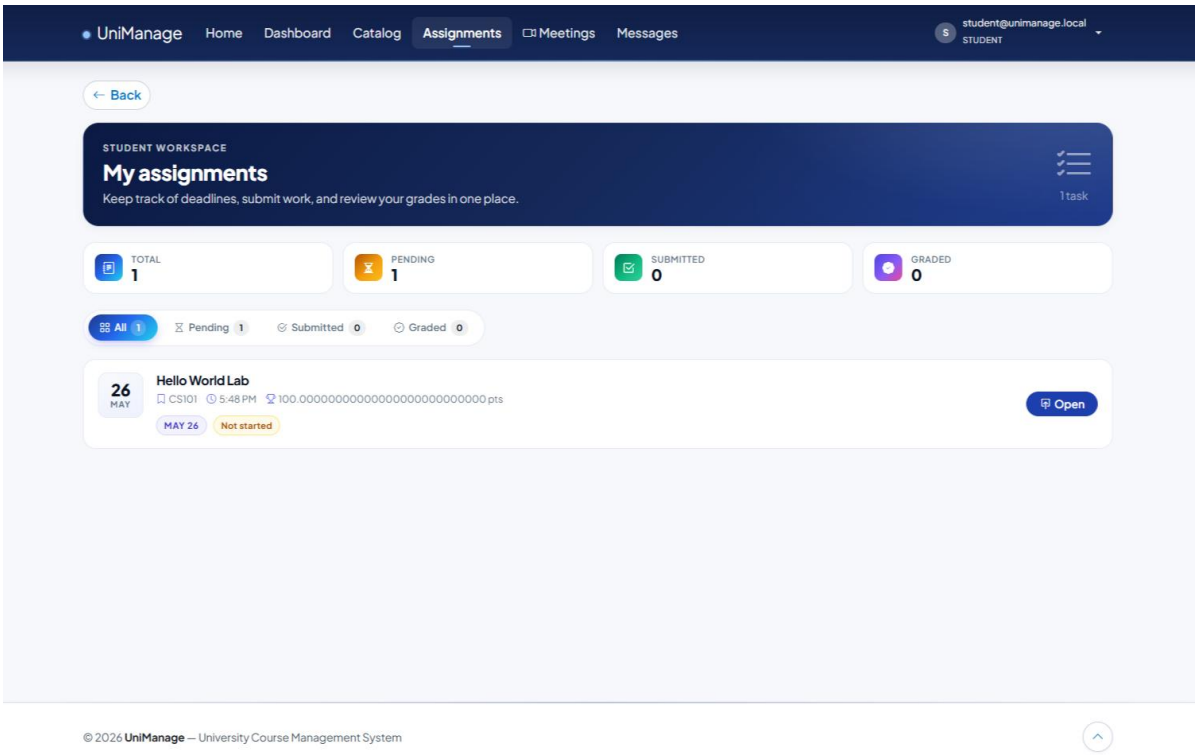


Figure UM11: Assignment Management Screen

This screenshot shows the assignment creation form.

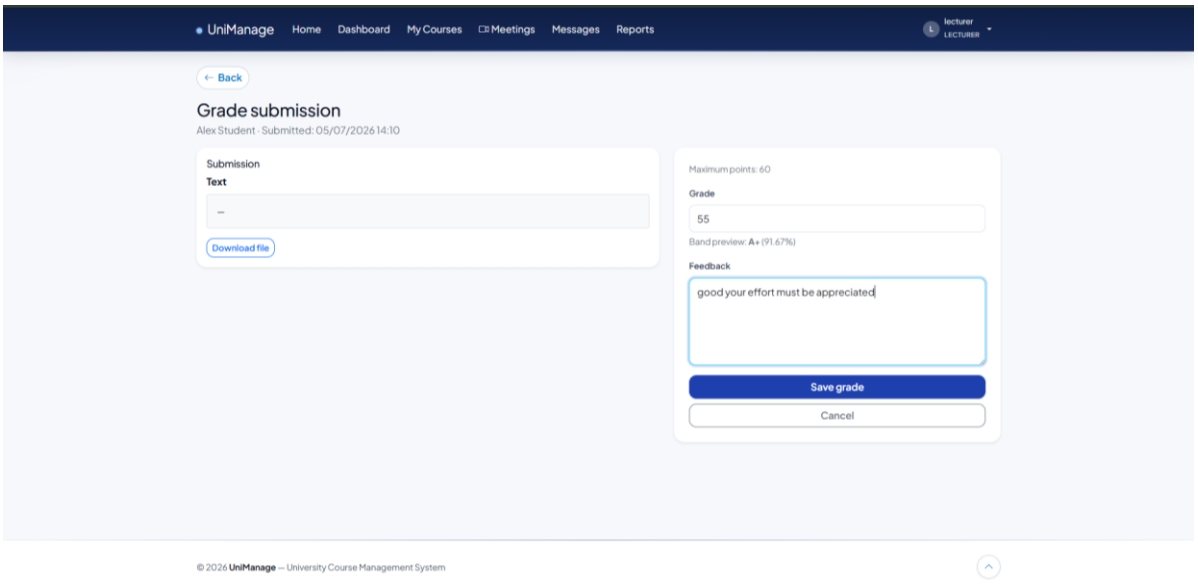


Figure UM13: Submission Grading Page

This screenshot shows the grading form with grade and feedback fields.

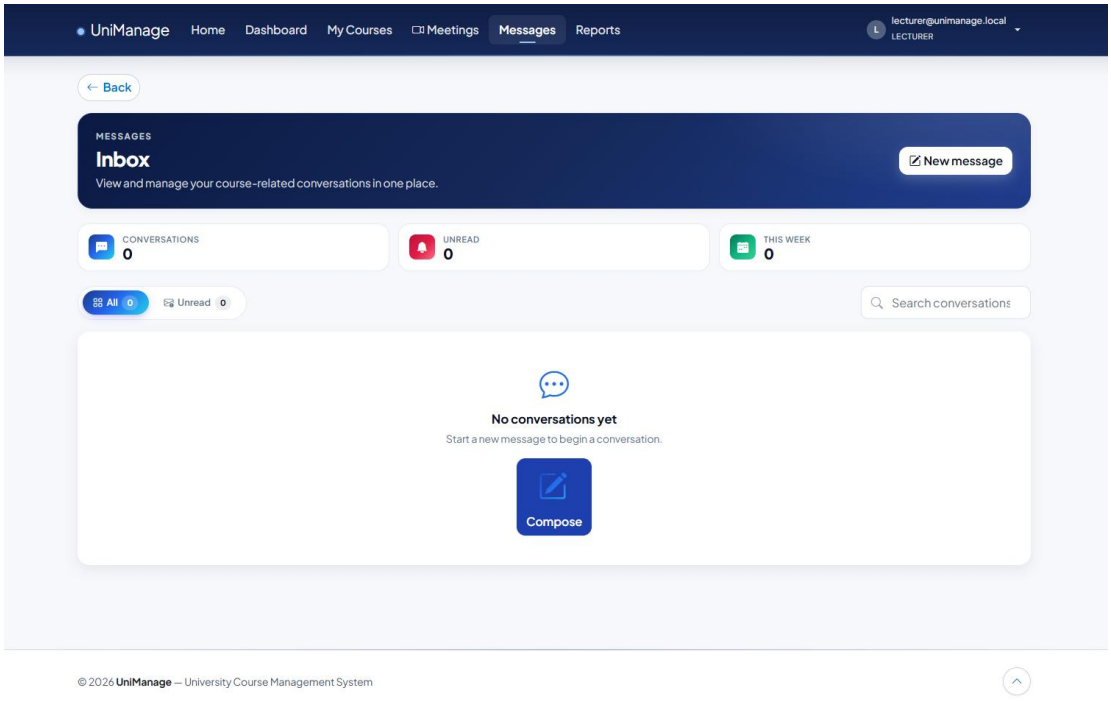


Figure UM15: Messaging Page

This screenshot shows the inbox or compose page.

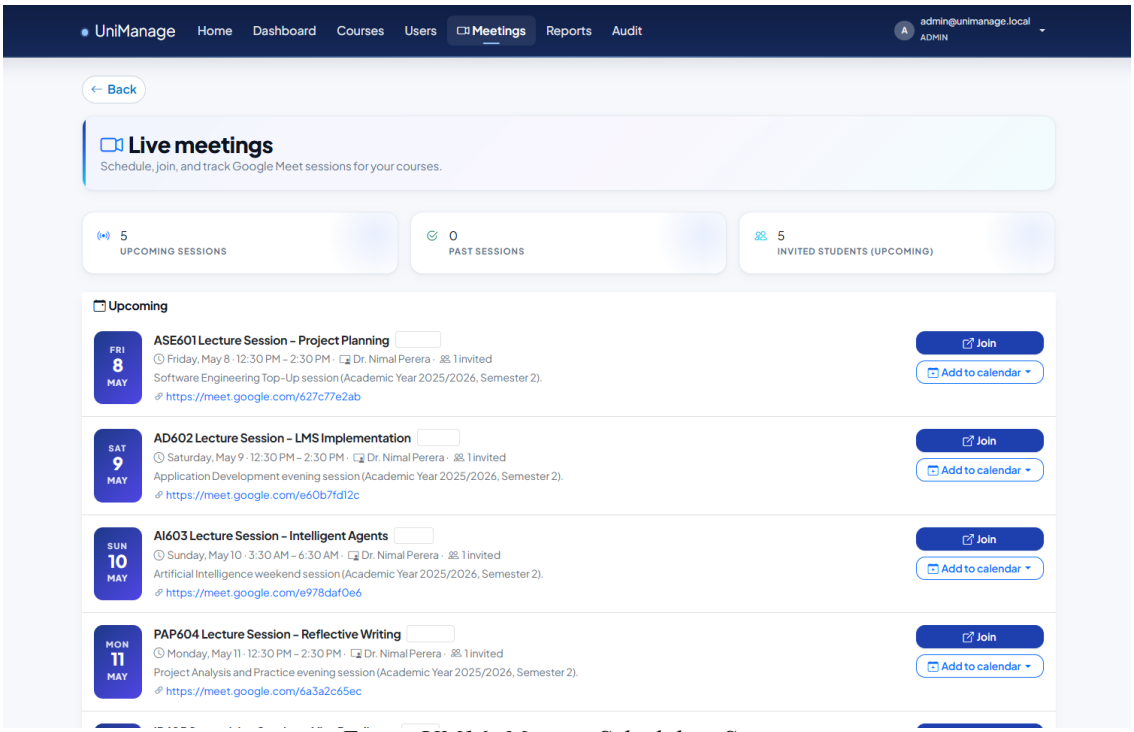


Figure UM16: Meeting Scheduling Screen

This screenshot shows the meeting create form.

9. Administrator User Guide

After logging in, the Administrator is redirected to the Administrator dashboard. The dashboard shows system-wide counts (users, courses, enrolments, assignments, submissions, messages), popular courses by enrolment count, and recent audit log activity.

The Administrator can perform the following actions.

View the dashboard. The dashboard provides a quick overview of the system.
Manage courses. Open the Courses page to list, search, page, view, create, edit, or delete courses. The course form requires the code (unique), name, description, credits, enrolment limit, lecturer, and optional prerequisite.

Manage users. Open the Admin User Management page to review registered users. The list shows username, full name, email, phone, role, lock status, and email confirmation status. Use Create to add an account, Edit to update details or role, and Lock/Unlock to control account access.

Use account safety controls. The system blocks unsafe actions such as locking the signed-in administrator account, deleting the signed-in administrator account, deleting administrator accounts, or changing the only remaining administrator in a way that could remove administrator access.

View reports. Open the Reports page to access the six reports: course popularity, student performance, lecturer workload, enrolments, pass and fail, and assignment attendance. Each report can be exported to CSV. Selected reports can be exported to PDF through QuestPDF.

Review the audit log. Open the Audit Logs page to inspect recent events. The log records the category, action, detail, user identifier (where available), success flag, and timestamp.

Manage meetings (where authorised). Administrators can review meetings as part of the academic oversight responsibility.

Update profile and change password. Use the Profile menu in the same way as the other roles.

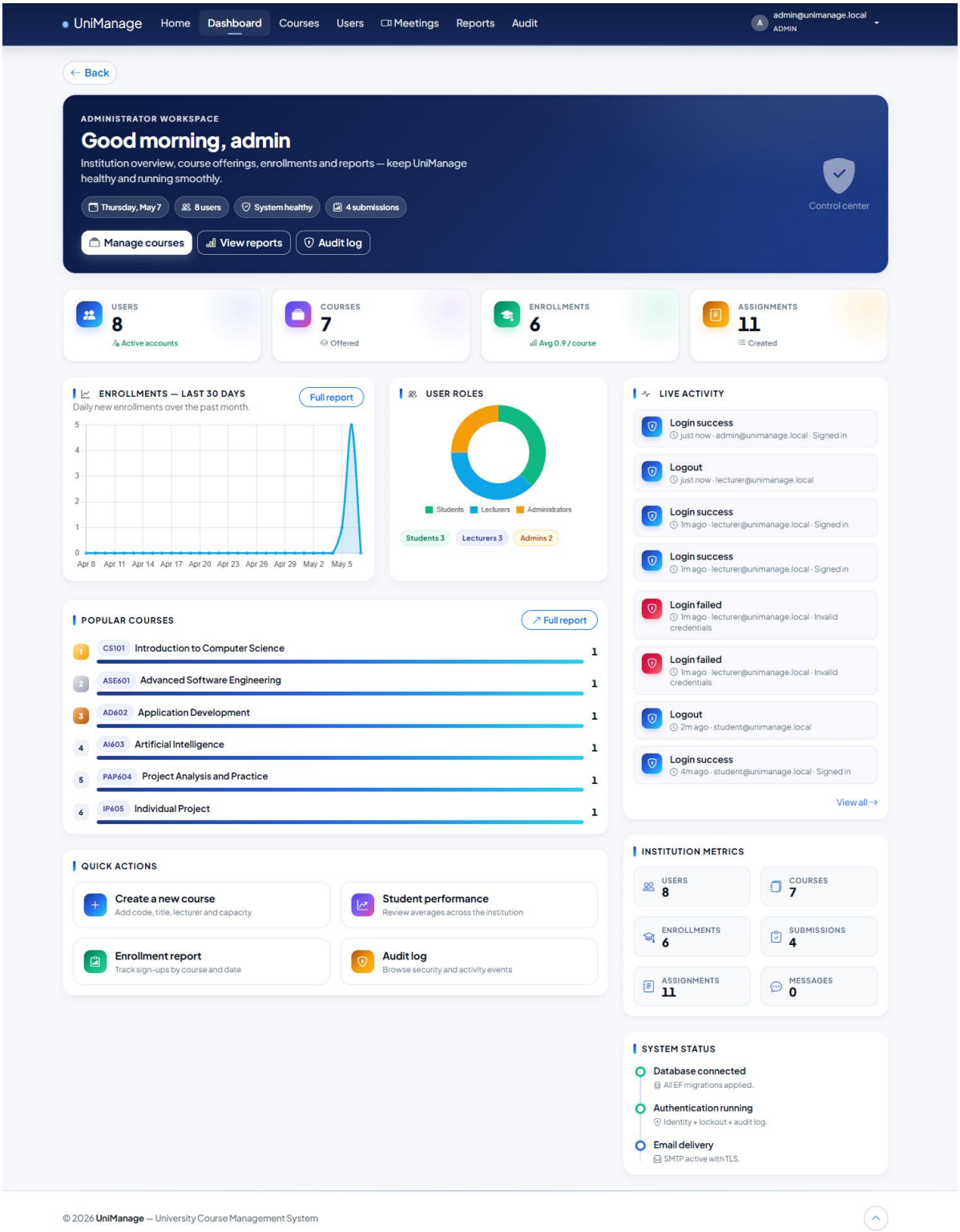


Figure UM8: Administrator Dashboard

This screenshot shows the Administrator dashboard with counts and popular courses.

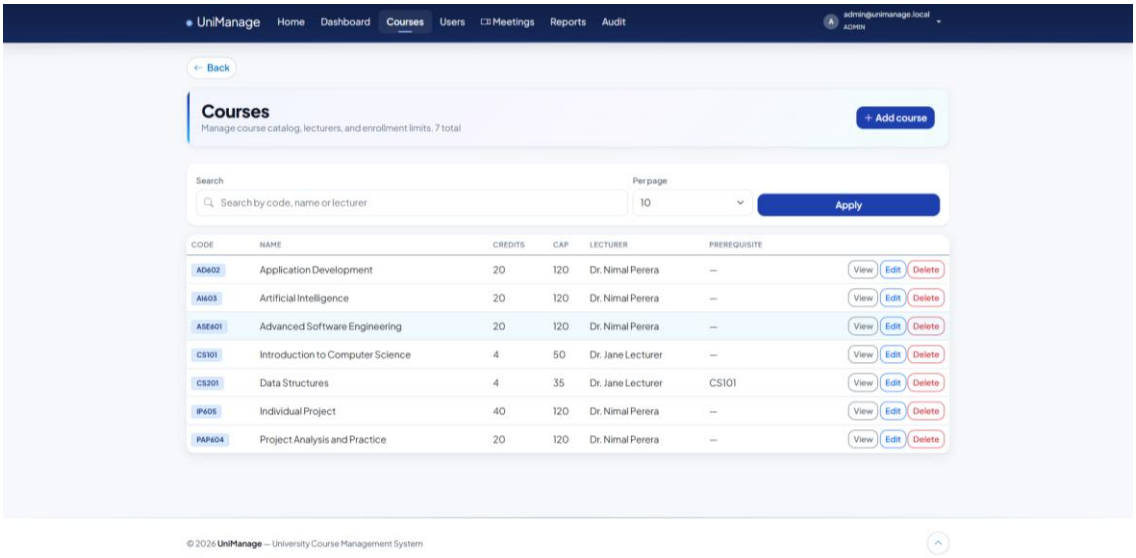


Figure UM9: Course Management Screen

This screenshot shows the course list with search and paging visible.

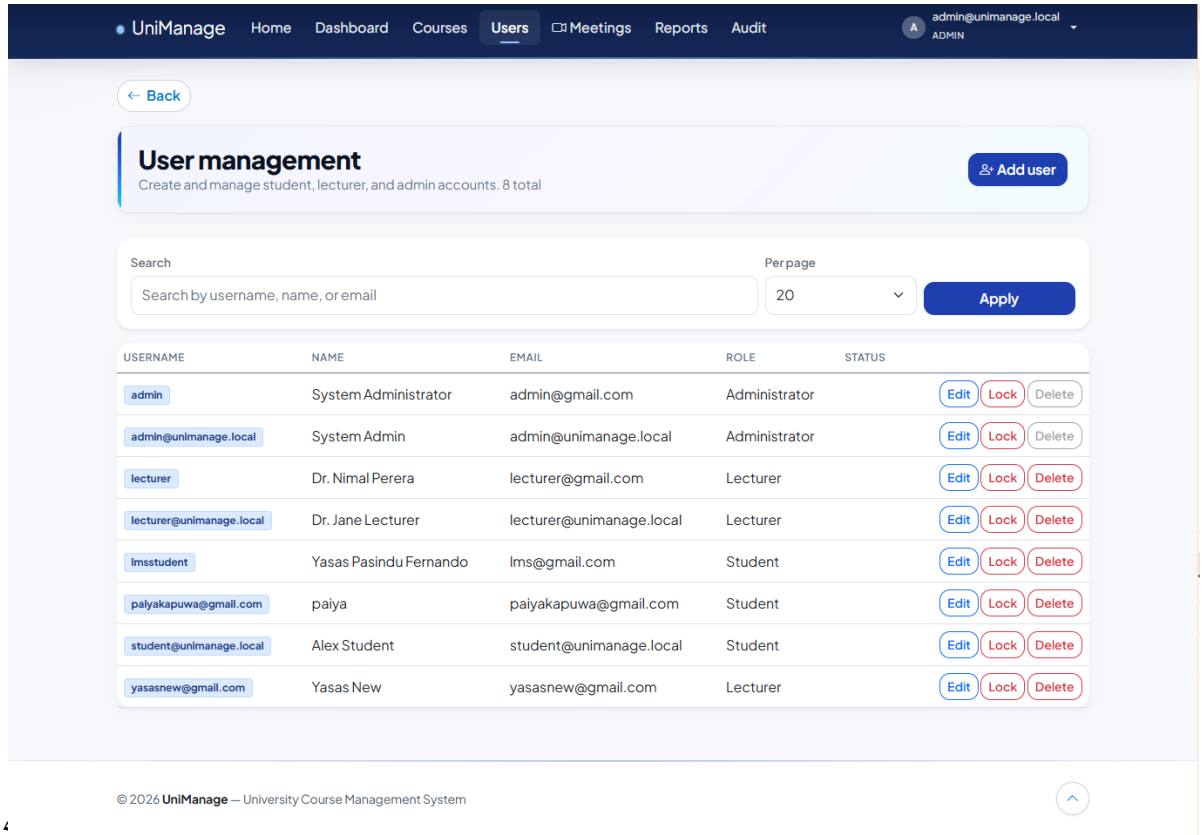


Figure UM9b: Admin User Management Screen

This screenshot shows the Admin User Management list with lock status and role information.

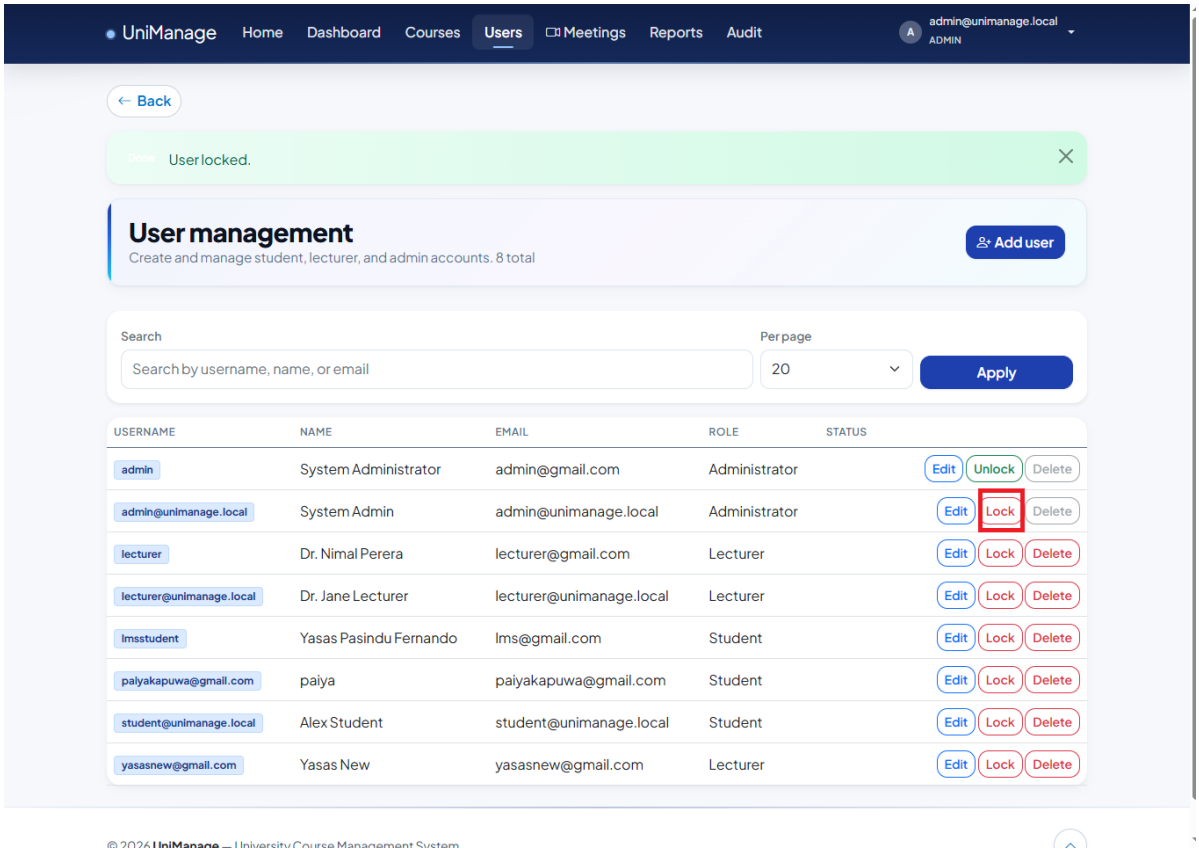


Figure UM9c: User Lock and Unlock Evidence

This screenshot shows the lock or unlock confirmation message for a managed account.

10. Validation and Error Messages

The application uses data annotations on view models and additional checks in controller actions. The following validation behaviour is expected during normal operation.

Scenario	Expected Validation Message
Required field missing	The field is required.
Invalid email format	The Email field is not a valid email address.

Scenario	Expected Validation Message
Weak password	The password must be at least 8 characters and include an uppercase letter, a lowercase letter, and a digit.
Confirm password does not match	The new password and confirmation password do not match.
Course code already exists	A course with this code already exists.
Enrolment full	This course has reached its enrolment limit.
Already enrolled	You are already enrolled on this course.
Prerequisite not satisfied	You must complete the prerequisite course before you can enrol.
Wrong file extension	The file type is not allowed.
File too large	The file is larger than the allowed size.
Disallowed message recipient	You are not allowed to message this user.
Invalid grade	The grade must be within the allowed range.
Anti-forgery token missing	The form has expired. Reload the page and try again.
Rate limit hit	Too many requests. Please wait and try again.
Access denied	You do not have permission to view this page.

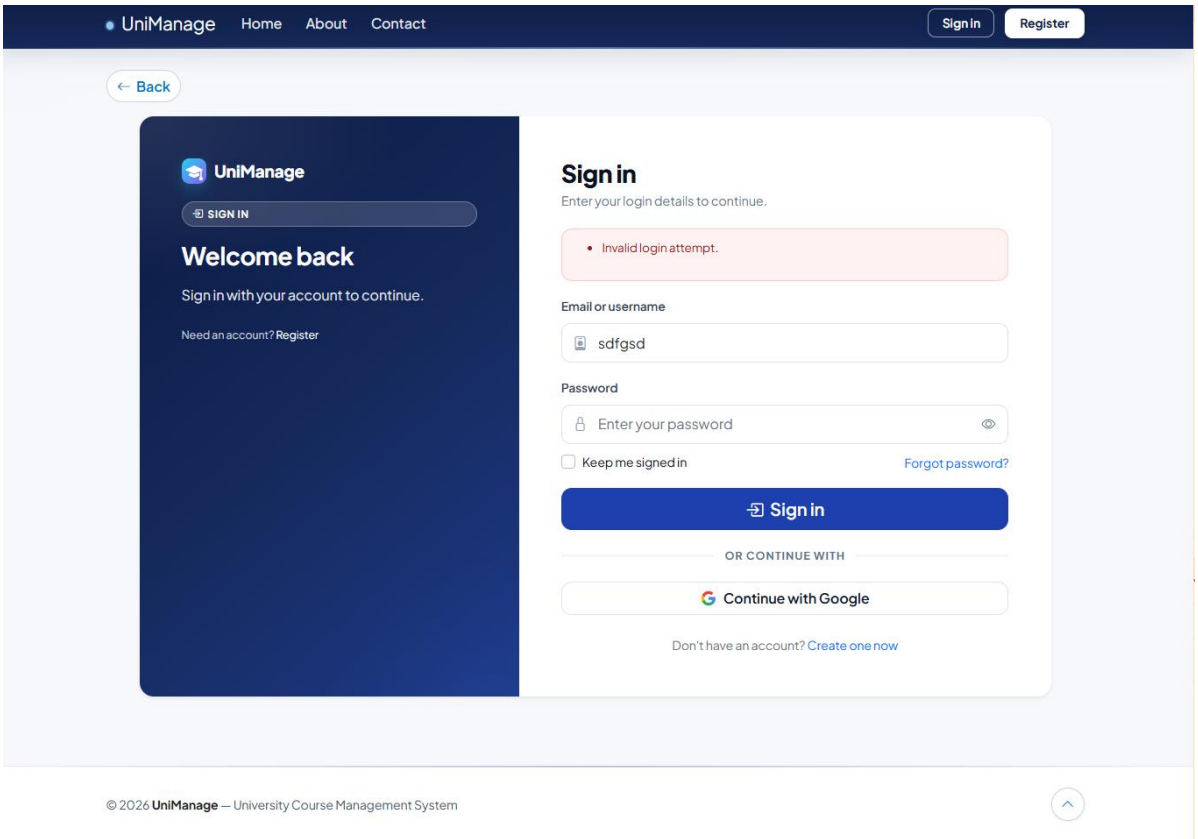


Figure UM17: Validation and Error Handling Evidence

This screenshot shows a form with one or more validation messages displayed.

The friendly error page Views/Shared/Error.cshtml is shown in production for any unhandled exception. The page does not reveal stack traces or internal server information. In development, the developer exception page provides detail to help locate the cause.

11. Reporting and Analytics

The Reports area is available to the Administrator, and selected report views are available to Lecturers with lecturer-scoped data. The following reports are implemented.

Report	Description	Source-Code Evidence
Course Popularity	Lists courses ordered by enrolment count.	ReportsController.CoursePopularity, Views/Reports/CoursePopularity.cshtml
Student Performance	Aggregates grades per student.	ReportsController.StudentPerformance, Views/Reports/StudentPerformance.cshtml
Lecturer Workload	Aggregates assignments and pending submissions per lecturer.	ReportsController.LecturerWorkload, Views/Reports/LecturerWorkload.cshtml
Enrolments	Lists enrolments with optional filters.	ReportsController.Enrollments, Views/Reports/Enrollments.cshtml
Pass and Fail	Groups submissions into pass and fail buckets using a configurable threshold.	ReportsController.PassFail, Views/Reports/PassFail.cshtml
Assignment Attendance	Shows the proportion of enrolled students who submitted each assignment.	ReportsController.AssignmentAttendance, Views/Reports/AssignmentAttendance.cshtml

To export a report, open the report and click the Export CSV or Export PDF button where present. Full export actions are administrator-focused where the controller restricts them to the Administrator role. The CSV file is generated through Infrastructure/CsvWriter.cs and the PDF file is generated through Infrastructure/PdfReport.cs using QuestPDF.

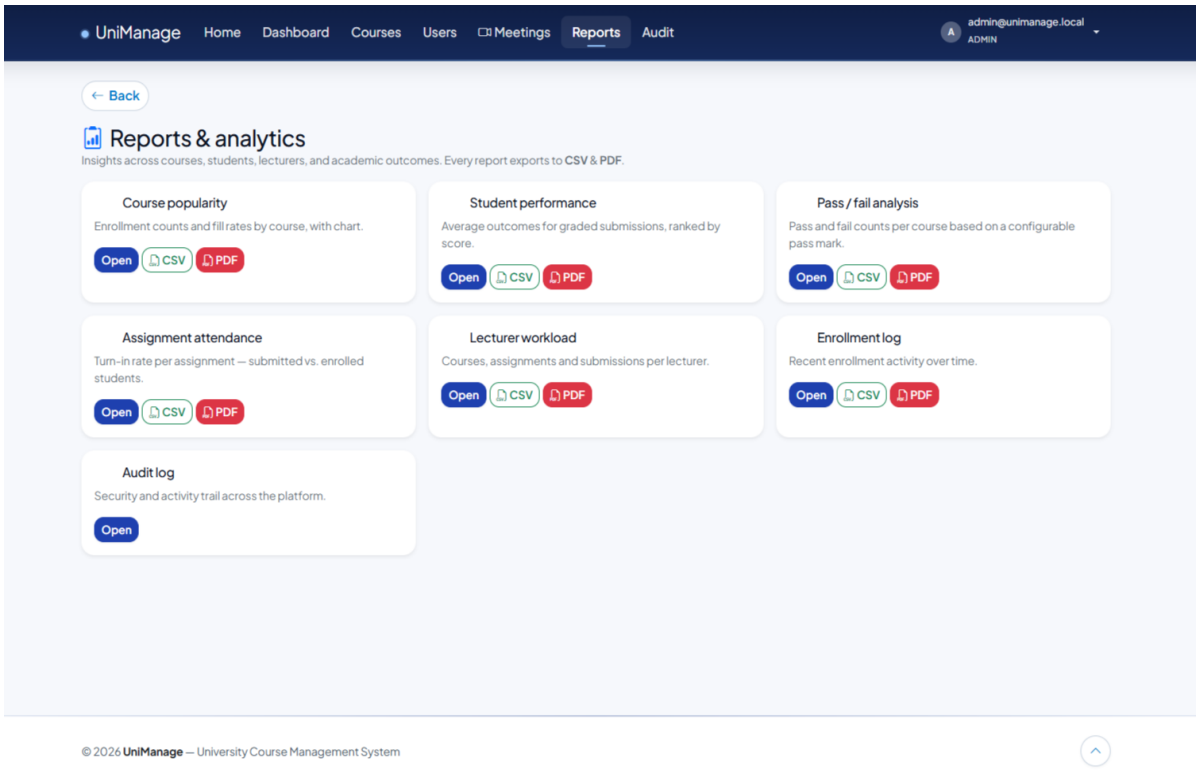


Figure UM14: Reporting and Analytics Screen

This screenshot shows the reports landing page

12. Security Notes for Users

The following security notes apply to all users of UniManage.

The application uses HTTPS by default during local development. If the browser shows a certificate warning, run dotnet dev-certs https --trust once to trust the development certificate.

The application uses anti-forgery tokens on every form. Submitted forms must include the token; otherwise the request is rejected.

The application uses rate limiting on authentication and upload endpoints. Submitting too many requests in a short period of time produces a 429 Too Many Requests response.

The application uses account lockout. After five failed sign-in attempts, the account is locked for fifteen minutes.

Passwords are not stored in plain text. The Identity framework hashes passwords using PBKDF2 with a per-user salt.

Audit log entries are recorded for security-sensitive actions. The Administrator can review these entries through the Audit Logs page.

Sensitive configuration values such as database, SMTP and Google OAuth values are kept outside the report and shared screenshots. In deployment, these values are supplied through environment variables or server-level configuration rather than being written into the public report.

Personal data shown in screenshots must also be redacted before inclusion in the final submission.

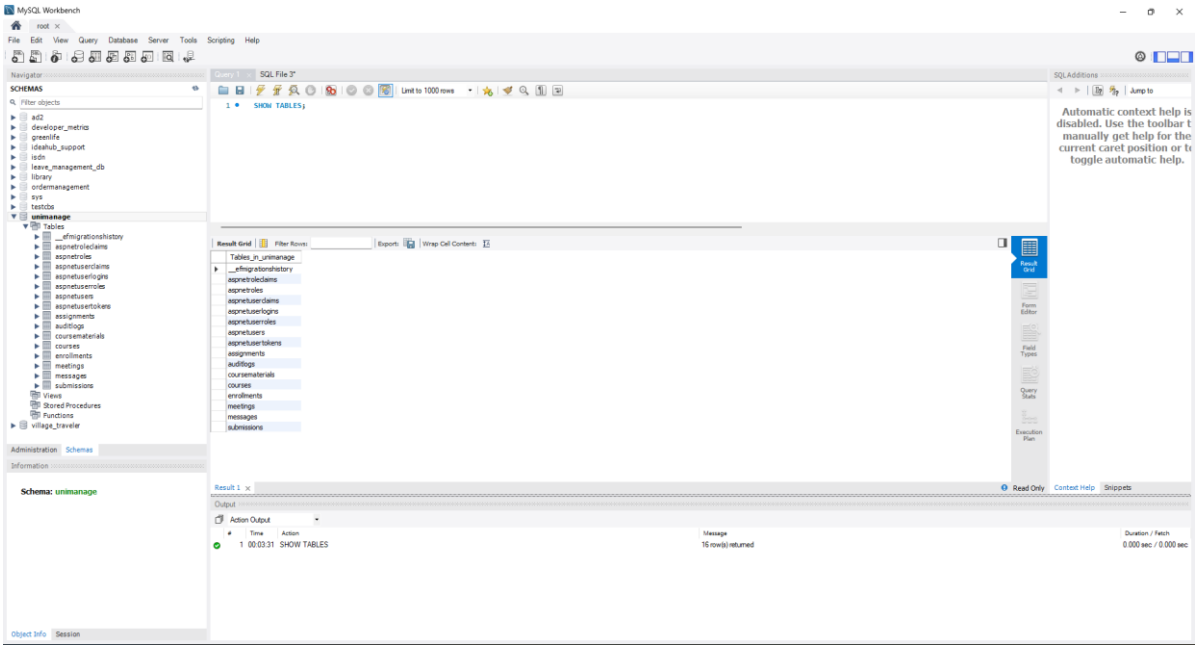


Figure UM18: Database and Seed Data Evidence

This screenshot shows a database table such as Courses, Enrollments, or Submissions after a few actions, with personal data redacted.

12A. Responsive / Mobile Usage Note

UniManage is a responsive web application rather than a separate mobile application. The Bootstrap layout allows the login page, navigation, dashboards, forms, reports, and course or assignment pages to adjust to narrower browser widths. During demonstration, browser developer tools or a smaller browser window can be used to check that the main controls remain visible and usable.

The responsive check supports the interface design and usability evidence, but it is not a full accessibility audit across every device. The screenshots below provide selected mobile viewport evidence.

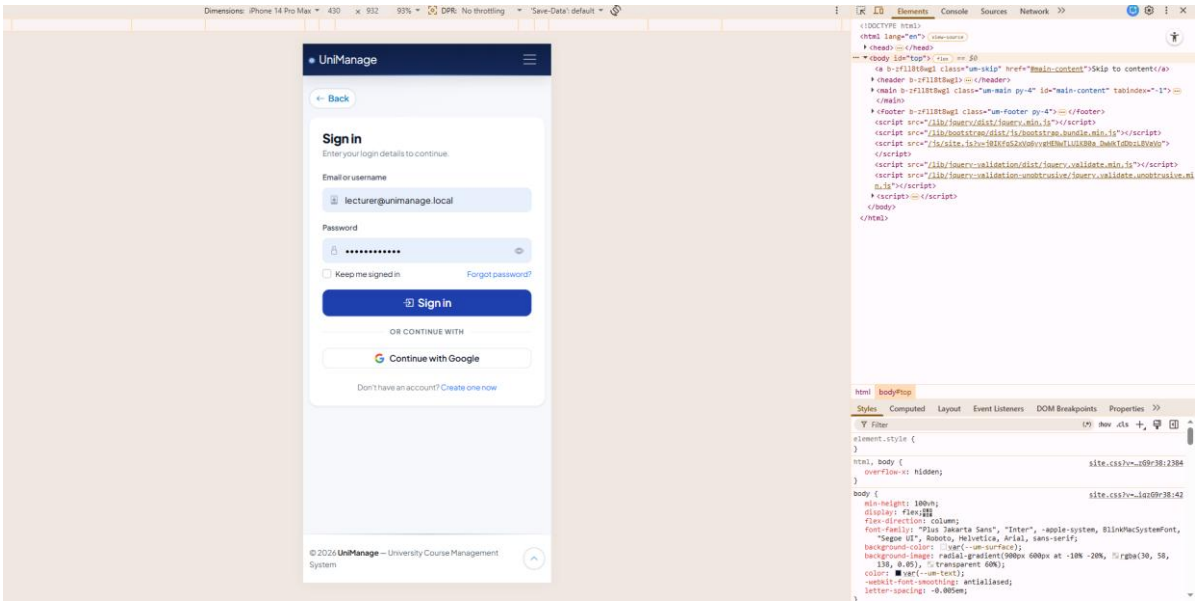


Figure M1: Mobile View of Login Page

This screenshot shows the login page in a mobile browser viewport.

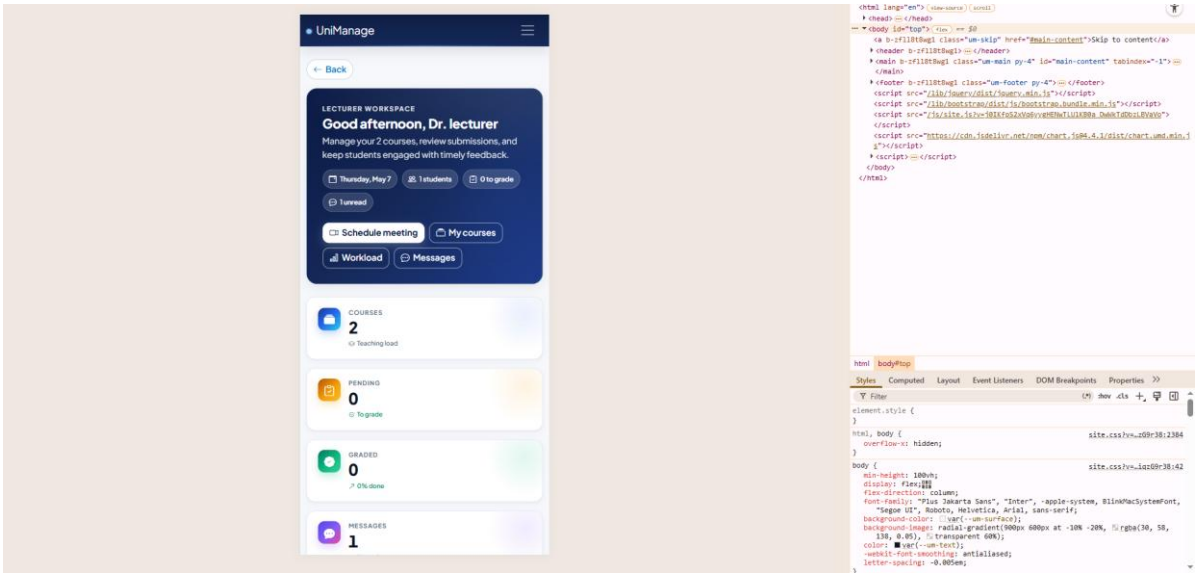


Figure M2: Mobile View of Dashboard

This screenshot shows a role dashboard in a mobile browser viewport.

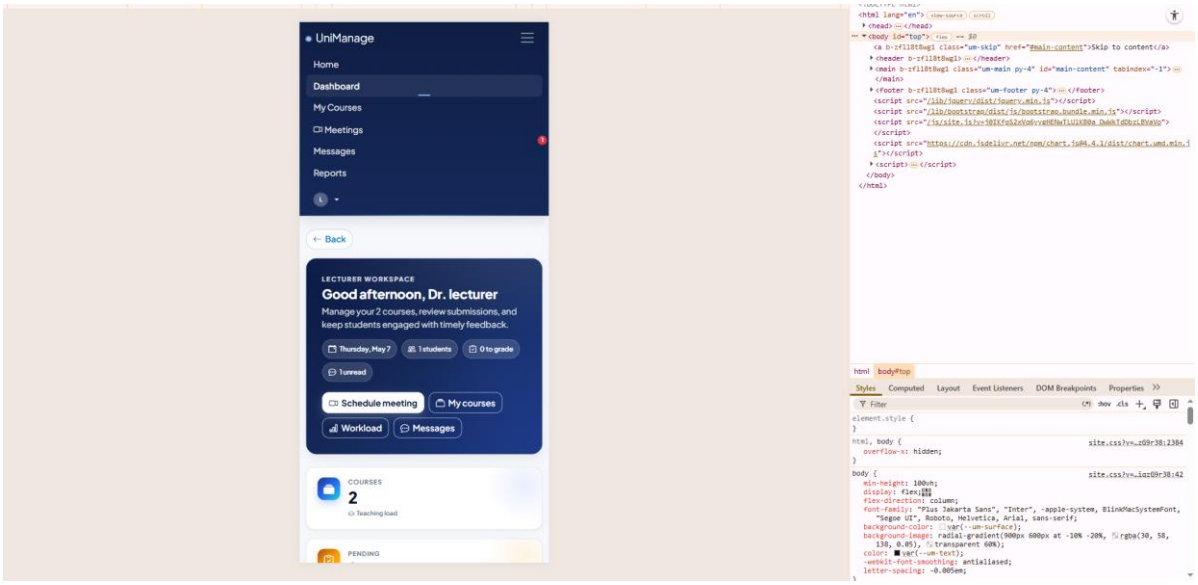


Figure M3: Mobile Responsive Navigation

This screenshot shows the collapsed navigation menu in a mobile browser viewport.

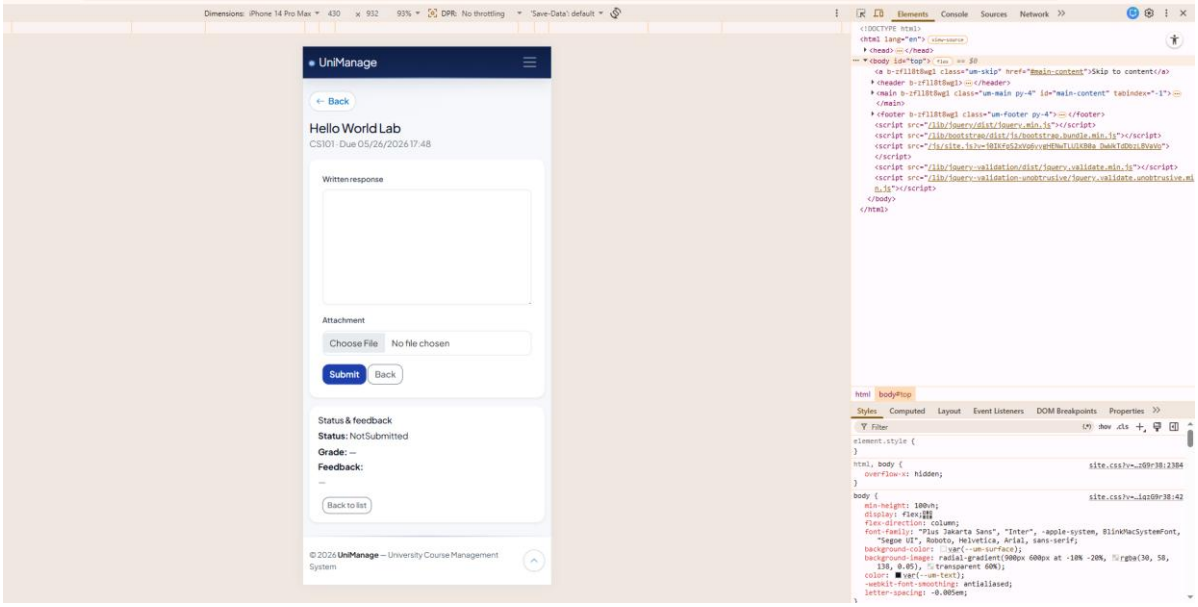


Figure M4: Mobile View of Course or Assignment Form

This screenshot shows an assignment submission form in a mobile browser viewport.

13. Troubleshooting

Problem	Suggested fix
The application does not start	Confirm that the .NET 8 SDK is installed and that the connection string is valid. Inspect the console output for an exception.
The login page rejects valid credentials	Confirm that the seeded users have been created. Check the application logs and re-run the application.
The login page shows a rate-limit error	Wait one minute and try again. The auth policy permits ten requests per minute per IP.
The account is locked	Wait fifteen minutes for the lockout window to expire, or reset the password.
A POST form is rejected	Reload the page to obtain a new anti-forgery token, then submit again.
File upload fails	Confirm that the file extension is allowed and that the file is under the maximum size.
The dashboard is empty	Confirm that the seeded data has been created. The Administrator dashboard counts use the seeded users and any data added during testing.
The friendly error page is shown	Inspect the application logs for the exception detail; the page does not reveal stack traces.
Email is not delivered	Check the Email section in appsettings.json and confirm that the SMTP server is reachable.
Google login is not available	Confirm that the Authentication:Google:ClientId and Authentication:Google:ClientSecret values are populated. If either is empty, Google login is disabled by design.
Migrations fail	Confirm database privileges. Run dotnet ef migrations list to see the current migrations and dotnet ef database update to apply them.
The reports page is empty	Confirm that data has been seeded or added through the application. Empty reports indicate no underlying data, not an error.

14. Notes for Viva Demonstration

The viva demonstration should follow the full script provided in Appendix H of the main technical report. During the demonstration, the author should be ready to show the application structure, successful build output, local or hosted application launch, and the main workflows for Administrator, Lecturer, and Student users.

The demonstration should also include one validation example, one audit log example, and a brief walkthrough of Program.cs to explain Identity, EF Core, rate limiting, anti-forgery protection, security headers, and routing.

The author should refer to the final evidence summary for screenshots, diagrams, database evidence, build output, migration evidence, selected unit-test output, source/build links, and the user manual. Genuine limitations, such as formal accessibility testing and production-level deployment hardening, should be mentioned honestly if asked during the viva..

Appendix S: Source and Build Links

Source and Build Links

Project: UniManage - University Course Management System

Student: Yasas Pasindu Fernando

Student ID: 25026764

Following approval to complete the coursework individually, the author completed the implementation, testing, documentation, and final preparation as an individual submission. This file records the source, build, hosted demonstration, database, and evidence locations used by the final report.

Diagram Source Links

Shared diagram folder:
https://drive.google.com/drive/folders/1PvvXtEf_6VFk632ysTXCwjlybSpcOaef?usp=sharing

Use Case Diagram editable draw.io source:
https://drive.google.com/file/d/1H9c4yXOcljag8dTxCizQyQIkxixHglvC/view?usp=sharing

Entity Relationship Diagram editable draw.io source:
https://drive.google.com/file/d/15IA6BQnpswKNplfang6vebdsNYfXlYU8/view?usp=sharing

UML Class Diagram editable draw.io source:
https://drive.google.com/file/d/189YrFogTsnGSPoH0Se6xBBerum1HaiOl/view?usp=sharing

Project Repository

Repository: https://github.com/YasasPasinduFernando/AD-COURSEWORK-2.git

Application Source Code

https://github.com/YasasPasinduFernando/AD-COURSEWORK-2/tree/main

Selected Unit Test Evidence

https://github.com/YasasPasinduFernando/AD-COURSEWORK-2/tree/unit-test/UniManage.Tests

Selected automated test evidence is included through the UniManage.Tests source files and selected unit-test output in Appendix U and Appendix V.

Packaged Build / Deployment

Local build command verified:

```
dotnet build --no-restore
```

Build result:

```
Build succeeded.
0 Warning(s)
0 Error(s)
```

Hosted Demonstration

Hosted Demonstration:

https://lms.yasaboy.com/

Previous IP-Based Endpoint:

http://34.171.6.234:8080

Hosting Platform:

Google Cloud (Compute Engine virtual machine).

Deployment Method:

Docker container deployment.

Domain / HTTPS:

The custom domain lms.yasaboy.com was configured for the hosted demonstration. Figure A7 shows the hosted UniManage application, while Figure A9 and Figure A10 provide DNS and server reachability support. HTTPS/domain access was used where available during final testing; full production hardening and formal security testing remain future improvements.

Deployment Notes:

The custom-domain URL is included as the supporting demonstration address. The earlier IP-based endpoint is retained as backup evidence. The application should also be tested locally in Visual Studio because the local build remains the main assessment route. The cloud virtual machine and Docker container must be running for the hosted URL to respond.

Reverse Proxy and DNS:

Cloudflare DNS and Nginx/server reachability evidence are included as supporting deployment evidence. These screenshots show the hosting setup path, while full CDN/security hardening is not claimed as a completed production feature.

Security Note:

Sensitive deployment values, including database passwords, SSH keys, Google Cloud service account keys, SMTP credentials, Google OAuth client secrets, and DNS or provider credentials, are excluded from the report and public repository. Screenshots use redacted or non-sensitive evidence where configuration is shown.

Database Script / Backup

Database evidence is provided through EF Core migrations, migration screenshots, and database table screenshots.

Detected migration folder:

Data/Migrations/

Detected migrations:

20260408041036_InitialUniManage.cs
20260501122117_AddAuditLogAndSecurity.cs
20260501125135_AddMeetings.cs
20260506142007_AddApplicationUserDateOfBirth.cs
ApplicationDbContextModelSnapshot.cs
Raw SQL script or database backup: Not separately provided.

Environment and Secret Handling

Safe template:

deployment.env.example

Confirmed environment-variable areas:

ConnectionStrings_DefaultConnection for the database connection string in Docker deployment.

Authentication_Google_ClientId and Authentication_Google_ClientSecret for Google OAuth where configured.

Email_Host, Email_Port, Email_SenderEmail, Email_Username, and Email_Password for SMTP where configured.

Real secrets are not included in this file, the report body, screenshots, or the public repository.

Evidence Folder

Evidence screenshots are included in the final report appendices and screenshots folder.

Local folder:

documentation_application_development/evidence/

User Manual

User_Manual_Application_Development.docx

Supporting document:

Installation Guide and User Manual

Local Project Paths

Solution:

AD COURSEWORK 2.sln

Project:

AD COURSEWORK 2.csproj

Launch settings:

Properties/launchSettings.json

Detected local URLs:

https://localhost:7212

http://localhost:5103

Configuration Warning

Private database passwords, SMTP passwords, Google client secrets, and other sensitive settings are excluded from the final report. Configuration evidence uses redacted or non-sensitive screenshots where required.

Appendix T: Final Evidence Summary

The final report is supported by source-code references, inserted screenshots, six design diagrams, database evidence, EF Core migration evidence, local build evidence, hosted demonstration evidence, responsive/mobile screenshots, and selected automated test output. The evidence package also includes source/build links, the installation and user manual, and selected unit-test appendices.

The main evidence covers registration and login, Student, Lecturer and Administrator dashboards, Admin User Management, course management, enrolment, assignment submission, lecturer grading, reporting, messaging, meeting scheduling, validation, database setup, migration history, build verification, unit testing, and responsive layout behaviour.

Some additional screenshots, such as separate audit-log views, report export evidence, access-denied examples, and further deployment dashboard captures, are treated as optional supporting evidence rather than required evidence.

Appendix U: Selected Unit Test Evidence

Unit Test Evidence (Supporting Coursework)

Project: UniManage - University Course Management System

Test project: UniManage.Tests

Framework: xUnit 2.9 with FluentAssertions 6.12

Test type: selected unit tests (no integration / no end-to-end)

Application target: .NET 8 (AD COURSEWORK 2.csproj)

Why tests were added

The coursework is assessed primarily through the running application and Visual Studio. A small xUnit evidence project can be included so that Chapter 15 of the main report can cite selected automated checks alongside the existing manual test plan. The tests provide quick, deterministic confirmation that important data-annotation rules and small infrastructure utilities behave as documented. They do not claim full behavioural or regression coverage.

Test classes created

File	Purpose
UniManage.Tests/Helpers/ValidationTestHelper.cs	Shared helper used to validate data-annotation rules in view models.
UniManage.Tests/AppRolesTests.cs	Confirms the Student, Lecturer and Administrator role constants used for RBAC.
UniManage.Tests/ViewModelValidationTests.cs	Tests validation rules for login, registration, course, assignment and message view models.
UniManage.Tests/InfrastructureUtilityTests.cs	Tests small infrastructure helpers such as CSV output, calendar links, meeting URL generation, email templates, security headers and grading rules.

Test cases

Test ID	Test class	Test method	Purpose	Expected result	Status
UT1	AppRolesTests	AppRoles_ShouldContainExpectedRoleNames	Stable role string constants	Values match Administrator, Lecturer, Student	Pass
UT2	AppRolesTests	AppRoles_ShouldNotContainEmptyValues	No accidental empty role	Each constant is non-empty	Pass
UT3	ViewModelValidationTests	LoginViewModel_WithMissingIdentifierAndPassword_ShouldBeInvalid	Required login identifier and password	Validation errors on both	Pass
UT4	ViewModelValidationTests	LoginViewModel_WithMissingIdentifier_ShouldBeInvalid	[Required] on LoginIdentifier	Validation error on login identifier	Pass
UT5	ViewModelValidationTests	RegisterViewModel_WithValidInput_ShouldBeValid	Happy path for registration	No validation errors	Pass
UT6	ViewModelValidationTests	RegisterViewModel_WithMissingRequiredFields_ShouldBeInvalid	Required attributes	Errors on FullName, Email, Password, Role	Pass
UT7	ViewModelValidationTests	RegisterViewModel_WithMismatchedConfirmPassword_ShouldBeInvalid	[Compare] on ConfirmPassword	Error on ConfirmPassword	Pass
UT8	ViewModelValidationTests	RegisterViewModel_WithPasswordBelowMinimumLength_ShouldBeInvalid	[StringLength] minimum	Error on Password	Pass
UT9	ViewModelValidationTests	RegisterViewModel_AllowedRoles_ShouldContainOnlyStudentAndLecturer	Self-service role list	Equals {Student, Lecturer}	Pass

Test ID	Test class	Test method	Purpose	Expected result	Status
UT10	ViewModelValidationTests	CourseInputViewModel_WithMissingCodeAndName_ShouldBeInvalid	Required Code and Name	Errors on both	Pass
UT11	ViewModelValidationTests	CourseInputViewModel_WithEnrolmentLimitOutOfRange_ShouldBeInvalid	[Range] on EnrollmentLimit	Error on EnrollmentLimit	Pass
UT12	ViewModelValidationTests	CourseInputViewModel_WithMissingLecturerId_ShouldBeInvalid	Required LecturerId	Error on LecturerId	Pass
UT13	ViewModelValidationTests	AssignmentInputViewModel_WithMissingTitle_ShouldBeInvalid	Required Title	Error on Title	Pass
UT14	ViewModelValidationTests	AssignmentInputViewModel_WithInvalidMaxPoints_ShouldBeInvalid	[Range] on MaxPoints	Error on MaxPoints	Pass
UT15	ViewModelValidationTests	MessageComposeViewModel_WithMissingRecipientSubjectAndContent_ShouldBeInvalid	Required messaging fields	Errors on RecipientId, Subject, Content	Pass
UT16	ViewModelValidationTests	MessageComposeViewModel_WithAllRequiredFields_ShouldBeValid	Happy path for messaging	No validation errors	Pass
UT17	InfrastructureUtilityTests	CsvWriter_Build_ShouldIncludeHeadersAndUtf8Bom	Report CSV helper	UTF-8 BOM, header row, data row	Pass
UT18	InfrastructureUtilityTests	CalendarLink_BuildIcs_ShouldContainTitleStartAndUrl	Meeting .ics content	Contains VCALENDAR, SUMMARY, URL	Pass
UT19	InfrastructureUtilityTests	CalendarLink_GoogleCalendarUrl_ShouldContainEncodedTitleAndCalendarHost	Google Calendar deep link	URL host and encoded title	Pass
UT20	InfrastructureUtilityTests	MeetLinkGenerator_GenerateGoogleMeetUrl_ShouldReturnNonEmptyMeetUrl	Generated meeting link	Starts with https://meet.google.co	Pass

Test ID	Test class	Test method	Purpose	Expected result	Status
				m/, accepted by IsValidMeetingUrl	
UT21	InfrastructureUtilityTests	MeetLinkGenerator_IsValidMeetingUrl_ShouldRejectUnknownHostsAndEmptyValues	URL validator	Rejects empty / unknown host; accepts Zoom and Teams hosts	Pass
UT22	InfrastructureUtilityTests	EmailTemplates_BuildPasswordResetEmail_ShouldContainResetGuidanceAndEncodedLink	Password reset HTML	Expected copy and encoded reset URL	Pass
UT23	InfrastructureUtilityTests	SecurityHeadersMiddleware_ShouldAddExpectedResponseHeaders	Security response headers	X-Frame-Options, X-Content-Type-Options, CSP, etc.	Pass
UT24-UT28	InfrastructureUtilityTests	SubmissionGradingRules_IsValidGrade_ShouldRespectInclusiveZeroToMax (theory)	Inclusive bounds for grading	Five [InlineData] rows match expected	Pass

(UT24-UT28 are the five [InlineData] rows of one [Theory] method; xUnit reports them as five separate outcomes.)

Command

From the repository root, with the test project included in the source package:

```
dotnet test "AD COURSEWORK 2.sln"
```

Result summary

```
Passed. Failed: 0; Passed: 28; Skipped: 0; Total: 28. Duration: approximately 800
ms - UniManage.Tests.dll (net8.0).
```

Figure 19 reflects the selected unit-test result used in the final report.

Limitations

Selected unit tests only; not a full behavioural or regression suite.

No live MySQL or DbContext integration tests.

No SMTP, no real email delivery.

No Google OAuth.

No browser automation, no Razor view rendering tests.

No Google Cloud or Docker dependency in the test process.

Appendix V: Selected Unit Test Code Examples

Unit Test Code Examples (Appendix Snippets)

These are short excerpts from the UniManage.Tests project where the test source is included as supporting evidence. The snippets below are intended for inclusion in the report appendix as illustrative evidence of test style; they are not full source listings.

1. ValidationTestHelper excerpt

File: UniManage.Tests/Helpers/ValidationTestHelper.cs

Purpose: wrap Validator.TryValidateObject so each test can collect every data-annotation error in one call.

```
using System.ComponentModel.DataAnnotations;

namespace UniManage.Tests.Helpers;

public static class ValidationTestHelper
{
    public static IList<ValidationResult> ValidateModel(object model)
    {
        var results = new List<ValidationResult>();
        var context = new ValidationContext(model);
        Validator.TryValidateObject(model, context, results,
validateAllProperties: true);
        return results;
    }
}
```

The helper validates all properties (not just [Required]), which is what the controller experiences during model binding.

2. AppRolesTests excerpt

File: UniManage.Tests/AppRolesTests.cs

Purpose: lock the RBAC role string constants so accidental renames are caught quickly.

```
[Fact]
public void AppRoles_ShouldContainExpectedRoleNames()
{
    AppRoles.Administrator.Should().Be("Administrator");
    AppRoles.Lecturer.Should().Be("Lecturer");
    AppRoles.Student.Should().Be("Student");
}
```

These constants are referenced by [Authorize(Roles = AppRoles.X)] attributes throughout the project, so any drift would cascade into broken authorisation.

3. LoginViewModel validation test excerpt

File: UniManage.Tests/ViewModelValidationTests.cs

Purpose: confirm the login form rejects empty credentials before the controller runs.

```
[Fact]
public void LoginViewModel_WithMissingIdentifierAndPassword_ShouldBeInvalid()
{
    var model = new LoginViewModel();

    var results = ValidationTestHelper.ValidateModel(model);

    results.Should().NotBeEmpty();
    results.Should().Contain(r =>
r.MemberNames.Contains(nameof(LoginViewModel.LoginIdentifier)));
    results.Should().Contain(r =>
r.MemberNames.Contains(nameof(LoginViewModel.Password)));
}
```

This protects the [Required] rules on LoginViewModel. A regression that removes either attribute would surface here as soon as dotnet test runs.

4. RegisterViewModel valid input test excerpt

File: UniManage.Tests/ViewModelValidationTests.cs

Purpose: prove the registration view model is valid for a realistic happy-path payload (FullName, Email, matching Password/ConfirmPassword, allowed Role).

```
[Fact]
public void RegisterViewModel_WithValidInput_ShouldBeValid()
{
    var model = new RegisterViewModel
    {
        FullName = "Test Student",
        UserName = "teststudent",
        Email = "student@test.com",
        Password = "Password123!",
        ConfirmPassword = "Password123!",
        Role = AppRoles.Student
    };

    var results = ValidationTestHelper.ValidateModel(model);

    results.Should().BeEmpty();
}
```

This complements the negative tests for missing fields, mismatched confirm-password, and password-too-short, all of which live in the same file.

5. Infrastructure utility test excerpt - MeetLinkGenerator

File: UniManage.Tests/InfrastructureUtilityTests.cs

Purpose: confirm the meeting link generator returns a non-empty Google Meet URL that the project's own validator accepts.

```
[Fact]
public void
MeetLinkGenerator_GenerateGoogleMeetUrl_ShouldReturnNonEmptyMeetUrl()
{
    var url = MeetLinkGenerator.GenerateGoogleMeetUrl();

    url.Should().NotBeNullOrWhiteSpace();
    url.Should().StartWith("https://meet.google.com/");
    MeetLinkGenerator.IsValidMeetingUrl(url).Should().BeTrue();
}
```

A second test in the same file exercises IsValidMeetingUrl directly with empty input, an unknown host, and known hosts (Zoom, Teams), which keeps the URL allow-list honest.