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.

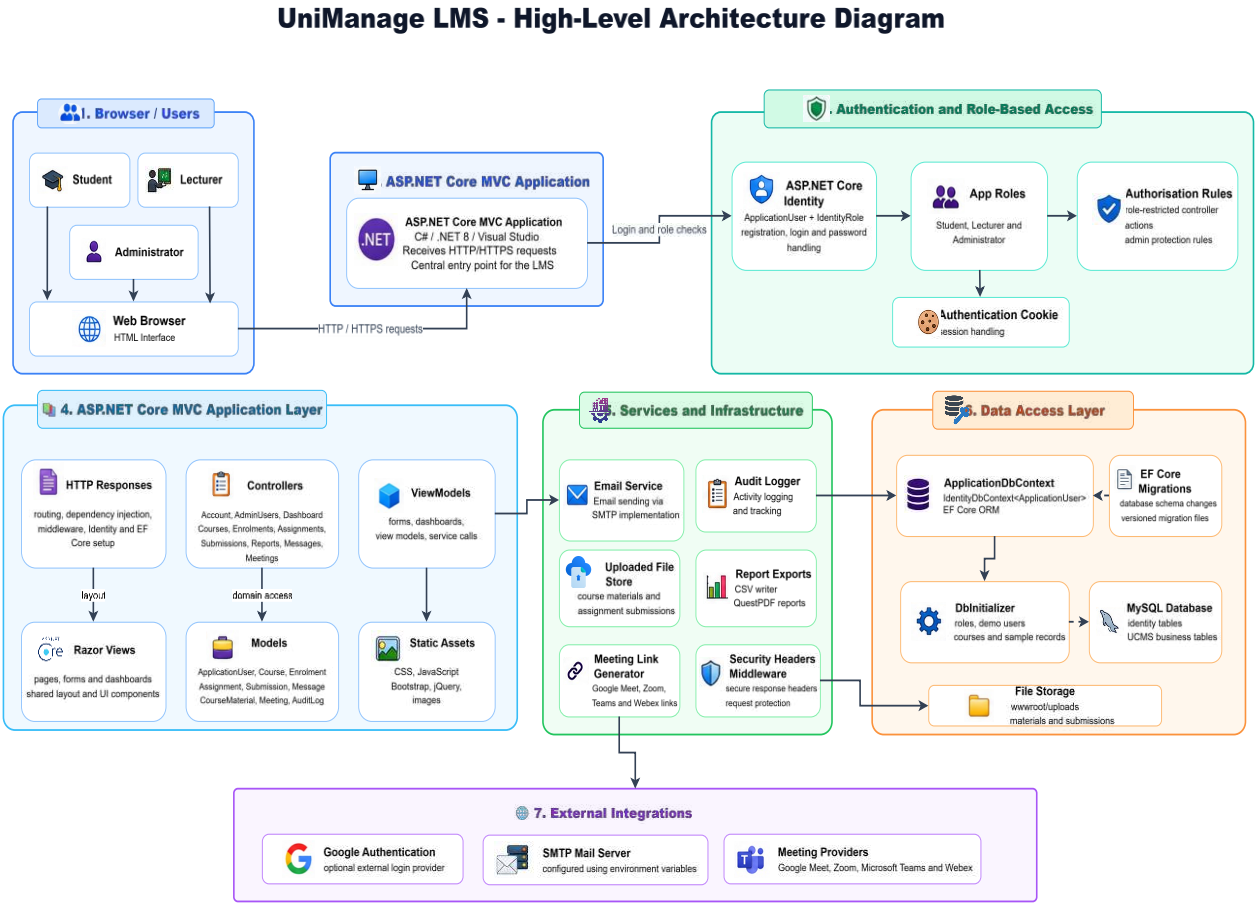

Figure 1: High-Level Architecture Diagram

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

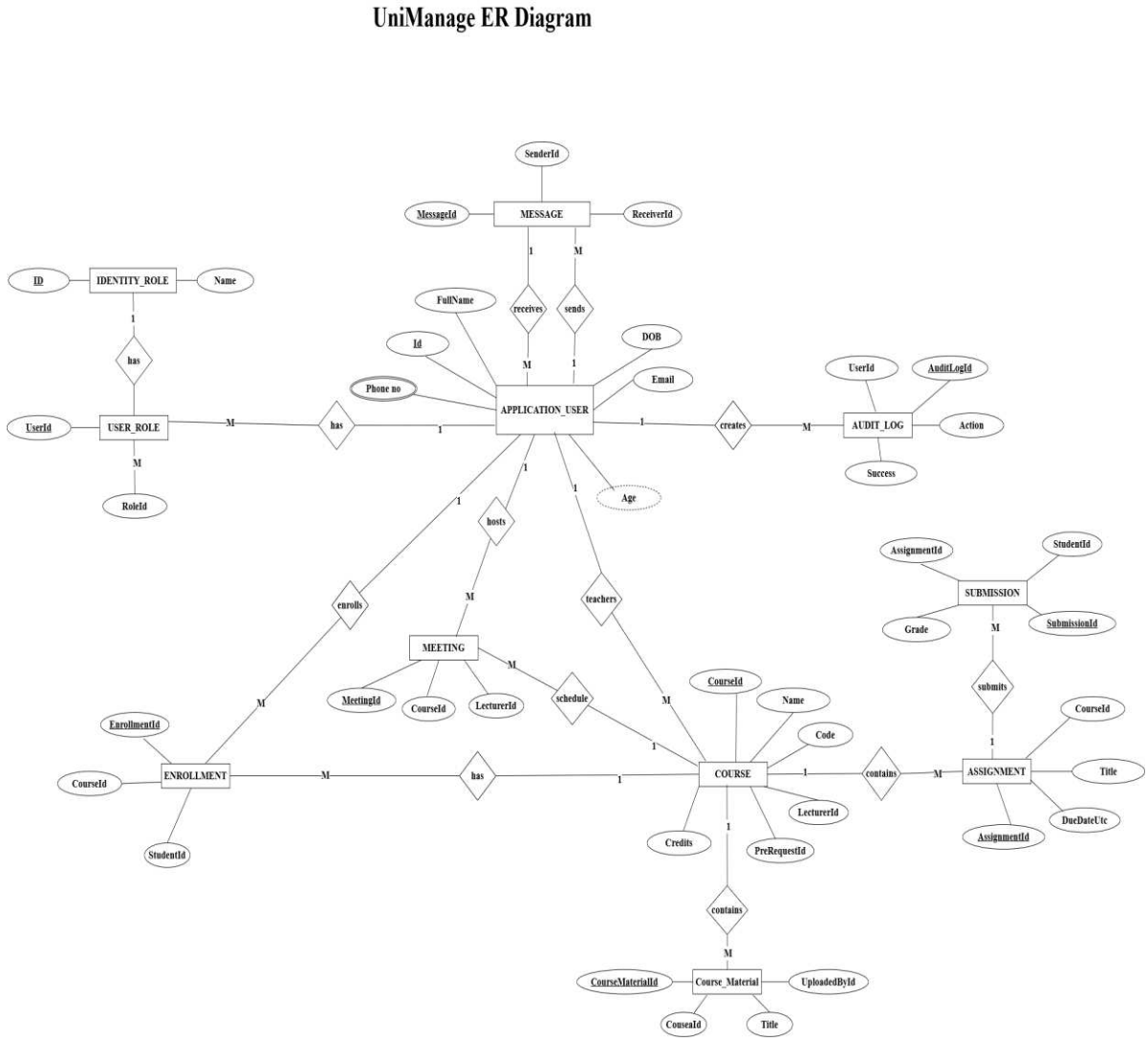

Figure 3: Entity Relationship Diagram

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

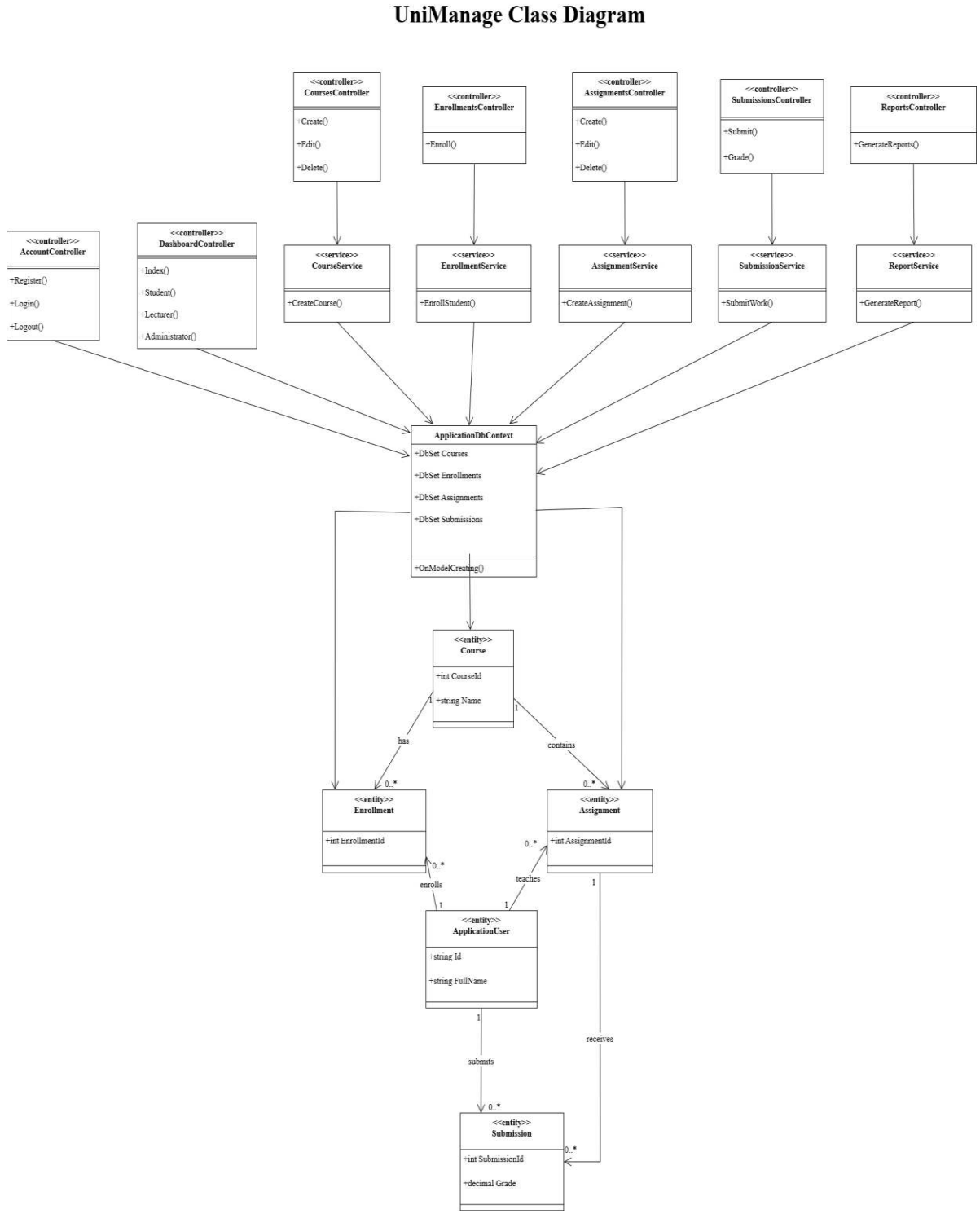

Figure 4: UML Class Diagram

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

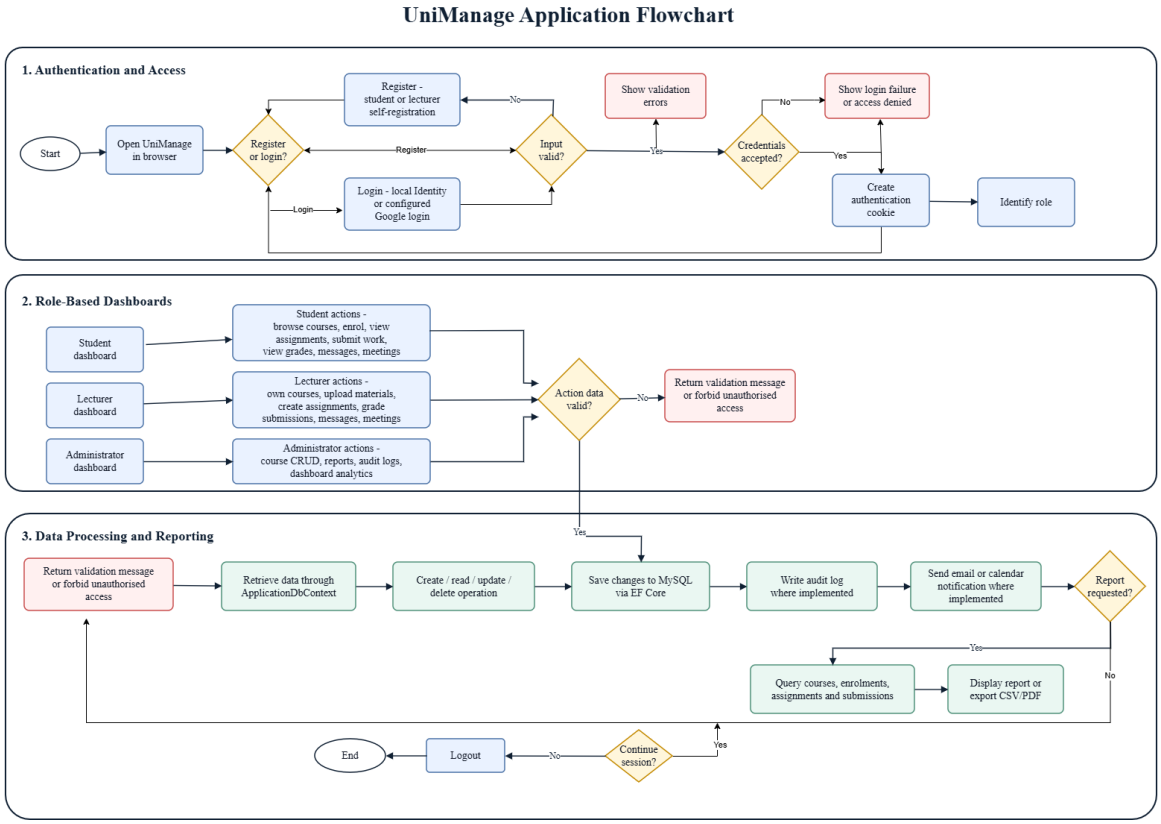

Figure 5: Application Flowchart

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

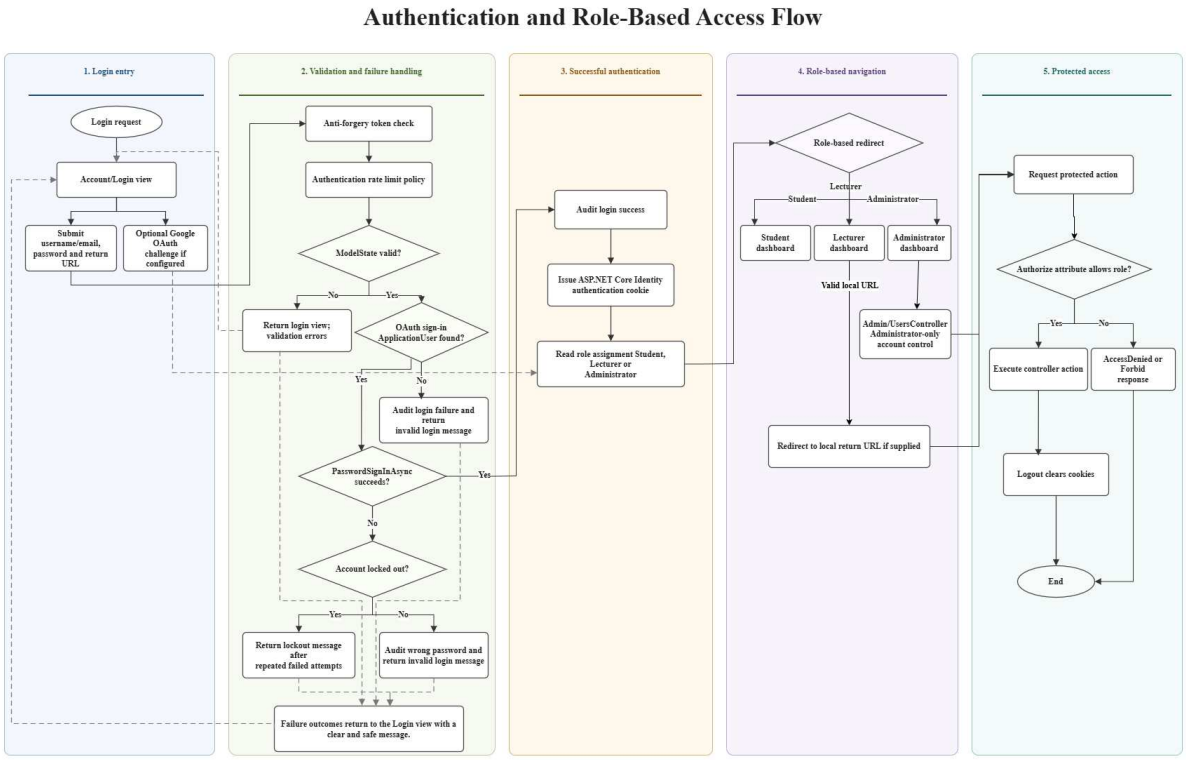

Figure 6: Authentication and Role-Based Access Flow

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,