BrightBoard –
A Course and Quiz System

CSCI 441 VA: Software Engineering

Jul 14, 2025

Team Members: Shadow Love-Erckert, Conner Erckert

Project URL: https://csci441-brightboard.onrender.com/
GitHub URL:https://github.com/Element713/CSCI441_BrightBoard.git

Work Assignment

---

a. Individual Contributions Breakdown

While both team members actively collaborated on all aspects of the project, specific responsibilities were divided to optimize workflow based on individual strengths. The following table summarizes each member's contributions across major report and development components.

Component / Task	Shadow Love-Erckert	Conner Erckert
1. Requirements Specification		
Functional Requirements (Use Cases)	A/R	A/R
Non-Functional Requirements (FURPS+)	A/R	A/R
Glossary of Terms	A/R	A/R
Requirements Consolidation	A/R	A/R
2. Domain Modeling		
Entity Identification	R	R
Relationship Mapping	C	R
Mongoose Schema Implementation	C	A/R
Data Validation Rules	A/R	A/R
3. Software Design		
Backend Architecture Design (Express.js)	C	A/R
File Structure Planning	C	A/R
Middleware (JWT Auth, Validation)	C	A/R
Routing Logic	A/R	A/R
API Endpoint Specification	A/R	A/R
4. Report Preparation		

Cover Page, Purpose, Work Plan, TOC	A/R	C
Requirements Section	A/R	A/R
Functional & UI Specification	A/R	C
Formatting and Final Edits	A/R	A/R
Diagrams and Illustrations	A/R	A/R
Customer Problem Statements	A/R	A/R
Glossary of Terms	A/R	A/R
Use Cases	A/R	A/R
System Architecture & System Design	A/R	A/R
Analysis & Domain modeling	A/R	C
Algorithms & Data Structures	C	A/R
Test Designs	A/R	A/R
Project Management	A/R	A/R
5. Other Contributions		
UI Design & Styling	R	C
Frontend Development (React, CSS)	R	A
Backend Integration (MongoDB, Express)	C	R/A
Quiz System Development	A/R	A/R
Testing & Debugging	A/R	A/R
Deployment (Local & Final)	A/R	A/R
Integration Management	A/R	C
Team Communication & Weekly Check-ins	A	C

Table of Contents

1. Customer Problem Statement

---

a. Problem Statement

The online learning landscape has experienced explosive growth in recent years, fueled by remote education, self-paced learning trends, and the rise of independent educators offering specialized content. However, the digital tools designed to support this new wave of learning have not kept pace with the needs of all instructors. Most existing Learning Management Systems (LMSs) were originally developed for large academic institutions or enterprises. As a result, they are often packed with advanced features that require technical knowledge, training, or IT support to operate effectively.

For independent tutors, freelance educators, corporate trainers, and hobby instructors, these systems can be more of a barrier than a benefit. The learning curve is steep, the interfaces are often unintuitive, and the setup process is overly complex for those who simply want to deliver straightforward, engaging content. These educators aren't looking for bloated platforms designed to manage thousands of users across departments; they need lightweight, practical tools that allow them to focus on teaching, not system configuration.

To cope with this gap, many small-scale instructors have resorted to using a mix of unrelated tools. A typical course might involve; Lesson content shared via email attachments or cloud drives (e.g., PDFs, Google Docs), quizzes built with free tools like Google Forms or Typeform, and progress and performance tracked manually in Excel or Google Sheets.

This approach creates a disconnected and inconsistent learning experience. There's no centralized hub for accessing course materials, submitting assignments, or checking quiz results. Feedback is often delayed or missing entirely. Instructors, meanwhile, have no clear visibility into how students are progressing or where they may be struggling.

The consequences of this fragmented approach are significant. Decreased student engagement, due to confusion and lack of structure, increased instructor workload, caused by repetitive tasks like grading and progress tracking limited feedback loops, preventing real-time support or course adjustments, lower course completion rates, as learners lose momentum or motivation

What’s clearly missing from the market is a simple, unified, and purpose-built LMS one that strips away the unnecessary complexity and focuses solely on what matters most: delivering content, assessing understanding, and supporting learner progress.

This modern LMS should support microlearning formats, ideal for short courses, private sessions, or skill-based workshops, and allow instructors to upload content easily in either PDF

or text form, enable the creation of quizzes with automatic scoring and instant feedback, provide a clean, centralized interface for students to view lessons, take assessments, and monitor their own progress, and offer intuitive analytics for instructors, helping them identify trends, assess outcomes, and support students proactively

In short, the solution must be streamlined, scalable, and accessible removing the friction that currently exists between small scale educators and the tools they use. By reducing technical overhead and centralizing essential functions, such a platform could dramatically improve the experience for both instructors and learners, fostering higher engagement, better outcomes, and broader access to high-quality educational content.

Independent Tutor

I've been teaching web design in short, one-month sprints to small groups of students. The hardest part isn't the content, it's delivering it in a way that's cohesive and engaging. Using a mix of Google Drive, emails, and Forms frustrates me and my students. I have no way to know who's keeping up, and grading quizzes manually is time-consuming. I've tried using some LMS platforms, but they're built for big institutions and are overkill for what I need.

What I really need is a lightweight, easy-to-use platform that allows me to organize my lessons, quizzes, and student progress in one place. It should give instant scoring, help me know how students are doing, and be fast to set up. Ideally, I should be able to upload a PDF or type a quick lesson, build a quiz, and be done in minutes. I don't have time to configure complex modules or install anything. If such a tool existed, it would save me hours each week and give my students a more engaging and clear learning experience.

Student

I'm taking short courses on topics like photography and personal finance from different instructors. Every time, it's a different process — sometimes I get an email, other times I'm sent to Google Classroom or a random website with links. It's confusing and annoying. I often forget where things are or miss a quiz just because I didn't check my email in time.

What I want is one platform where everything lives: the lesson, the quiz, and my scores all in one place, accessible on my phone or laptop. I'd like to see what I've completed and what I have left to do. I don't want to juggle five tabs just to learn something simple. It would be great if quizzes gave me my score right away, so I know if I understood the lesson or not. A system like that would make short courses much more enjoyable and help me stay committed to finishing them.

b. Decomposition into Sub-Problems

We identified the following sub-problems from the customer's narrative:

1. **Authentication & User Roles**
   - Users must register and log in as students or instructors.

2. **Course Management (Instructor Side)**
   - Instructors must be able to create courses, upload lessons (PDF or text), and add quizzes.

3. **Enrollment & Learning Flow (Student Side)**
   - Students need to browse courses, enroll, read lessons, take quizzes, and track progress.

4. **Quiz System**
   - Support multiple-choice quizzes with automated scoring and feedback.

5. **Progress Tracking**
   - Record student activity and display progress to both instructors and learners.

6. **Analytics Dashboard**
   - Instructors need a way to see which students are completing content and how well they are performing.

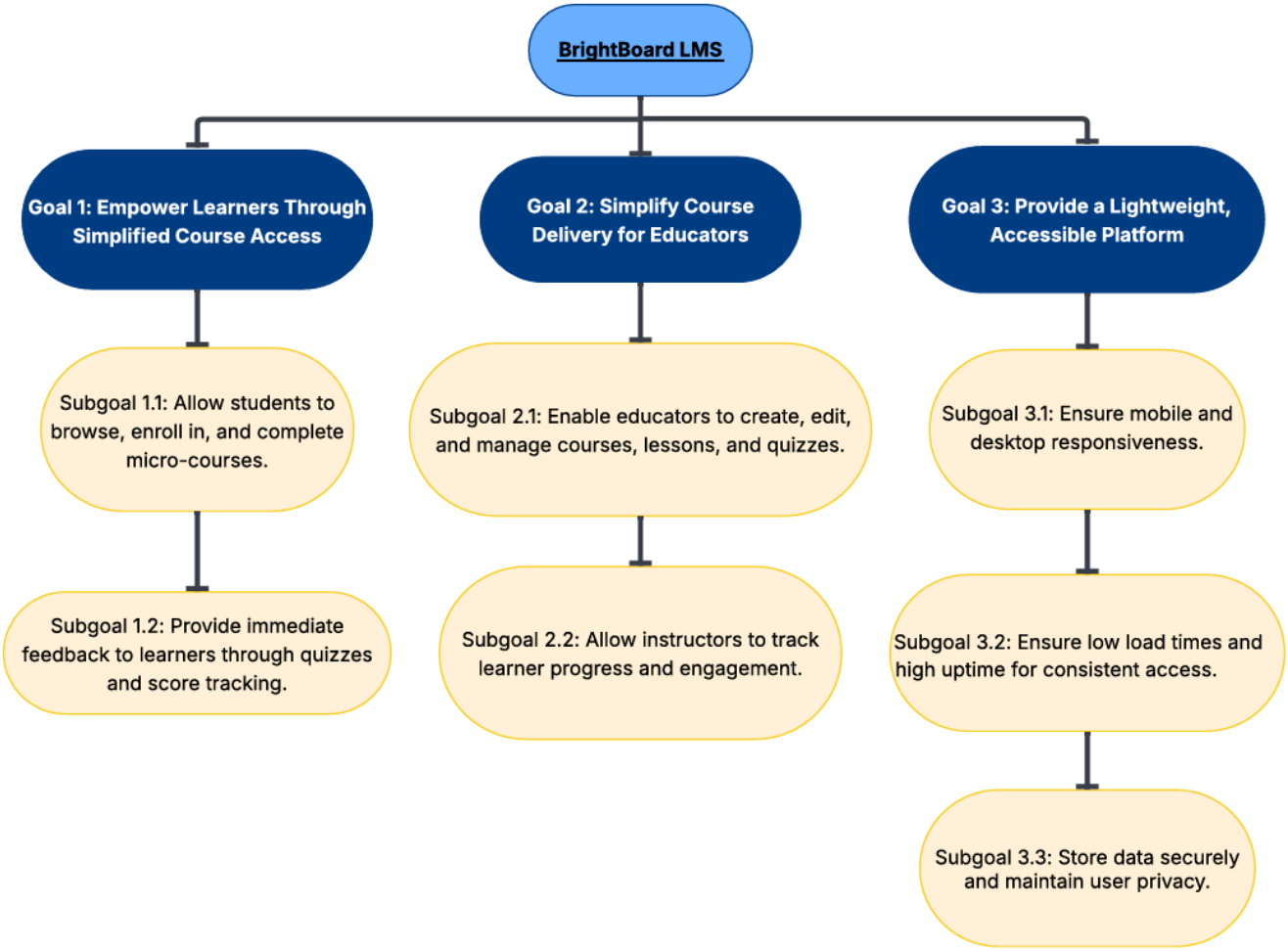
2. Glossary of Terms (Updated)

Term	Definition
BrightBoard	The name of the proposed lightweight Learning Management System (LMS).
Microlearning	Short, focused learning activities designed to achieve specific learning outcomes.
Course	A collection of lessons and quizzes grouped under a single subject or topic.
Lesson	A single unit of instructional content, presented as text or in PDF format.
Quiz	A set of multiple-choice questions associated with a specific lesson or course.
Instructor	A user who creates, manages, and publishes courses and their associated content.
Student	A user who enrolls in courses, studies lessons, and attempts quizzes.
Progress Tracker	A feature that displays a student's completion status for lessons and quizzes.
Educational Institutions	Small schools or informal learning organizations using BrightBoard as a supplementary platform.
Businesses	Organizations using BrightBoard for internal microlearning and training sessions.
System Developers	Individuals responsible for implementing the backend, frontend, and integration logic.
UI Designers	Team members who craft the user interface for usability and accessibility.
Web Application System	The underlying software system that delivers content and handles all user interactions.
Device	The physical hardware (desktop, tablet, or mobile) used to access BrightBoard.

Register/Login	The process of creating an account or signing in securely to access personalized dashboards.
Course Management	Instructor functionality for creating, editing, and deleting courses.
Lesson Uploading	Instructor capability to upload PDF/text content to courses.
Quiz Creation	Instructor capability to design quizzes for assessments.
Enroll in Course	Student action of joining an available course.
View Lessons	Student functionality to read and interact with uploaded lessons.
Take Quiz	Student interaction with course quizzes, including auto-scoring.
JWT Authentication	JSON Web Token–based system for securely verifying user identity.
UI (User Interface)	The visual and interactive part of BrightBoard that users engage with.
Dashboard	A role-specific home interface showing available actions and user-specific data.
MongoDB	The NoSQL database used in BrightBoard to store structured data like users, courses, lessons, quizzes, and progress.
Express.js	The Node.js-based backend framework used to create RESTful APIs for the BrightBoard platform.
Frontend	The part of the application the user interacts with directly (HTML/CSS/JS/React), responsible for rendering content and sending requests to the backend.
Backend	The server-side logic (Node.js/Express) that handles data storage, authentication, course management, and quiz operations.
Authorization	The process of verifying what actions a user is allowed to perform, often determined by their role (e.g., Student or Instructor).
API (Application Programming Interface)	A set of defined endpoints that allow communication between the frontend and backend systems, such as fetching quizzes or saving progress data.
CRUD Operations	Refers to Create, Read, Update, and Delete — the core actions performed on resources like courses, lessons, and quizzes.
Token Expiration	The configured duration after which a JWT (JSON Web Token) becomes invalid, requiring the user to log in again for security.

3. System Requirements (Updated)

a. Business Goals



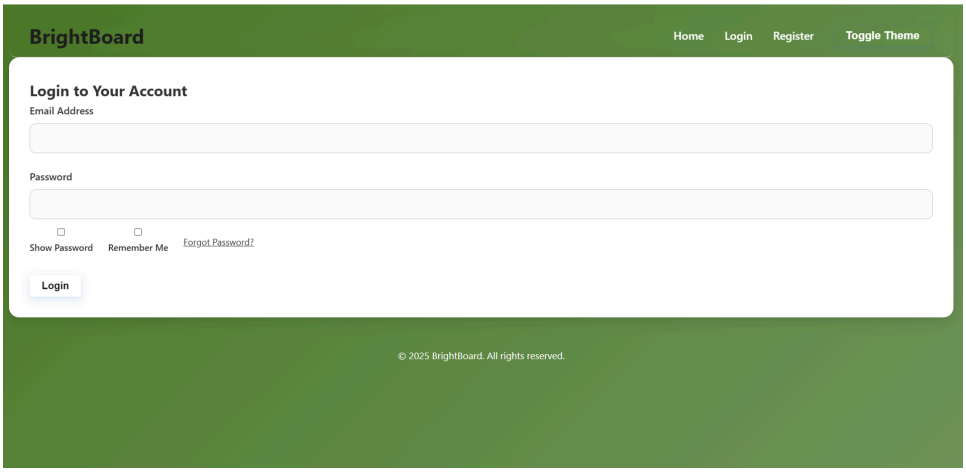
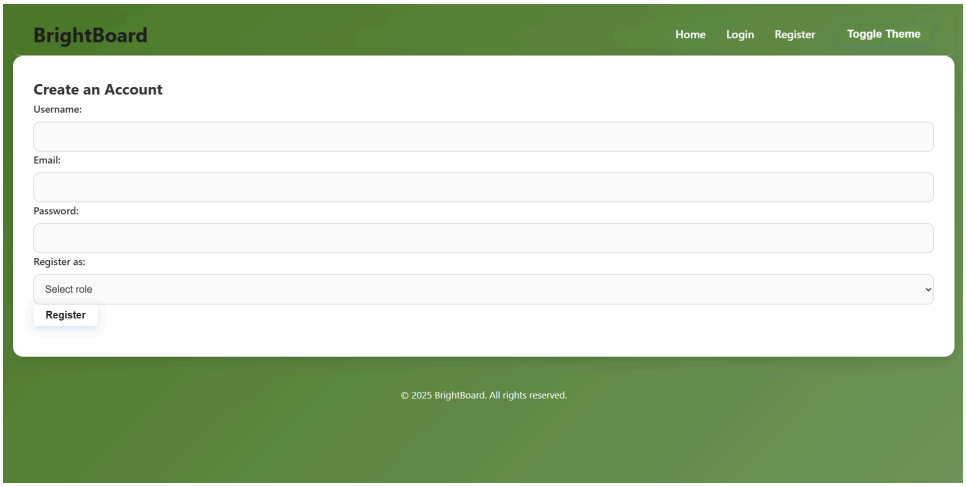
b. Functional Requirements

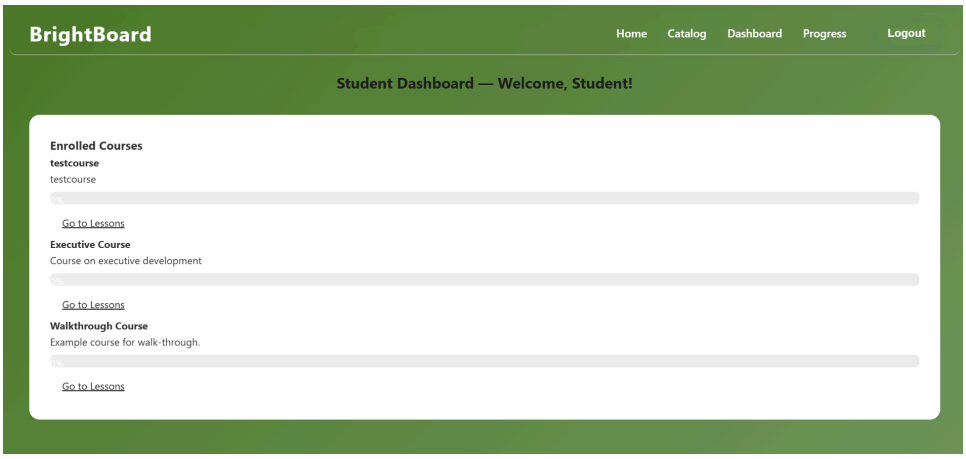
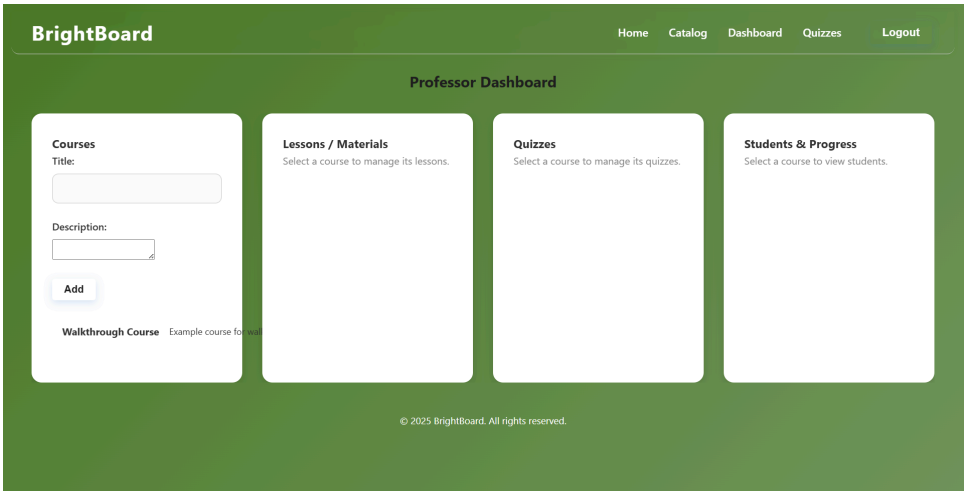
Req ID	Priority	Description
REQ-1	5	The system shall allow users to register and log in securely.
REQ-2	5	The system shall support two user roles: Instructor and Student.
REQ-3	5	Instructors shall be able to create, edit, and delete courses.
REQ-4	5	Instructors shall upload lessons in PDF or text format.
REQ-5	5	Instructors shall create multiple-choice quizzes.
REQ-6	5	Students shall browse and enroll in courses.
REQ-7	5	Students shall view lesson content.
REQ-8	5	Students shall take quizzes and receive automatic scoring.
REQ-9	3	The system shall track quiz scores and course progress.
REQ-10	3	Instructors shall view performance analytics.

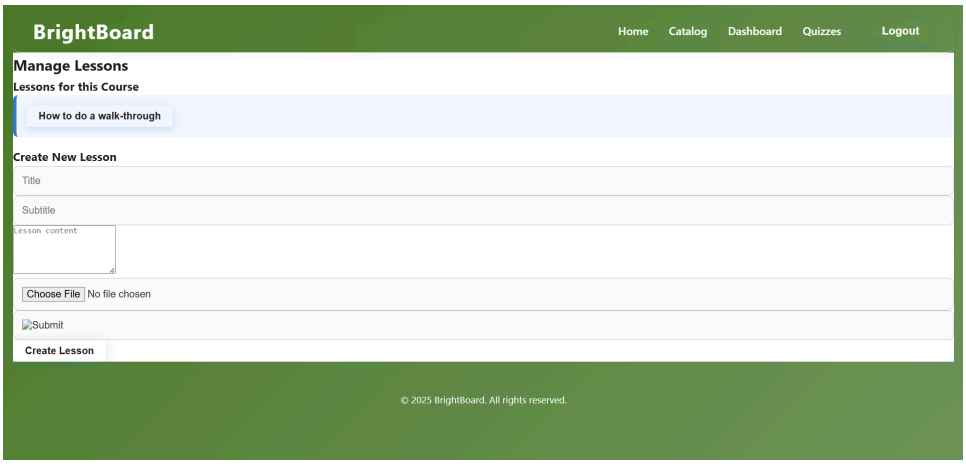
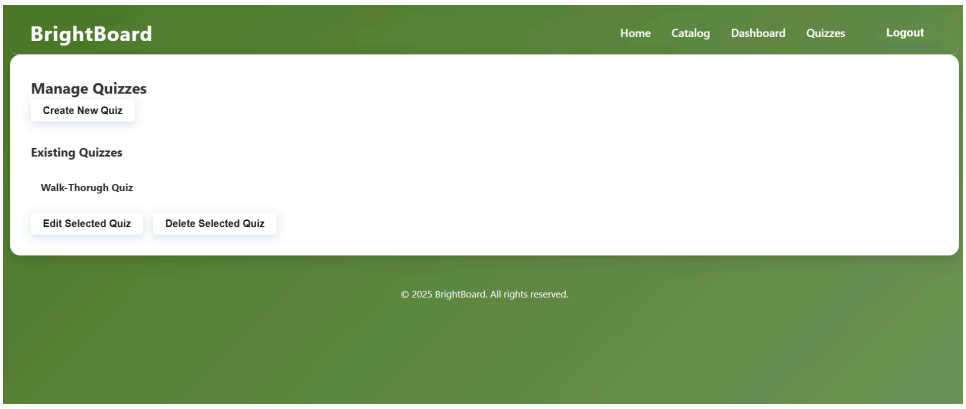
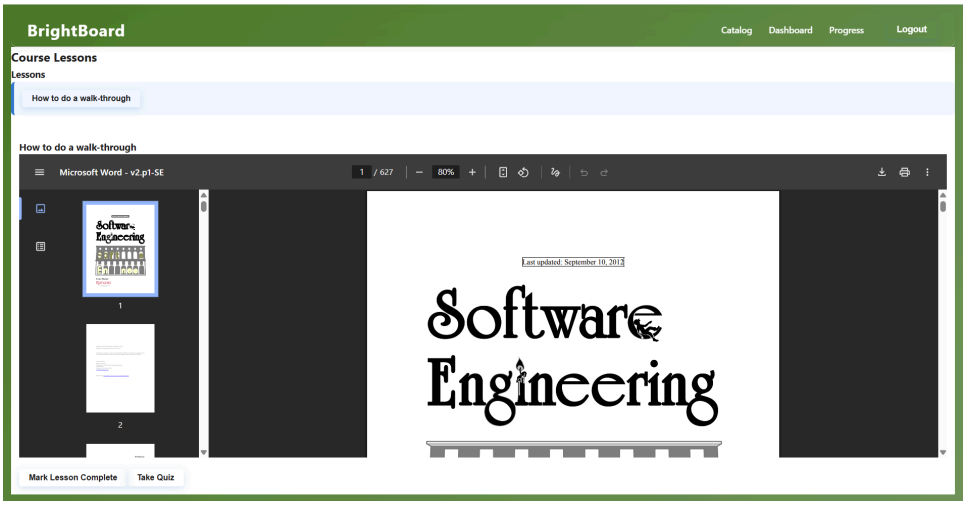
c. Non-Functional Requirements

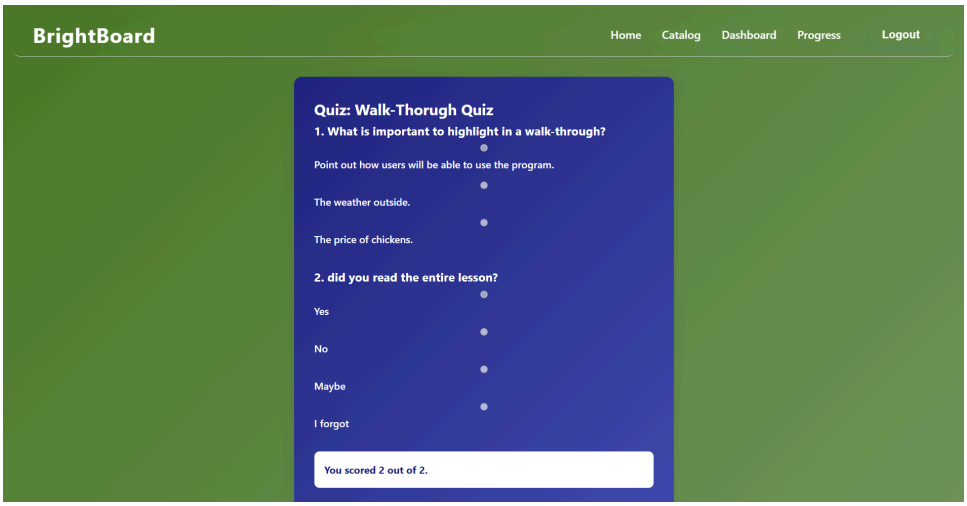
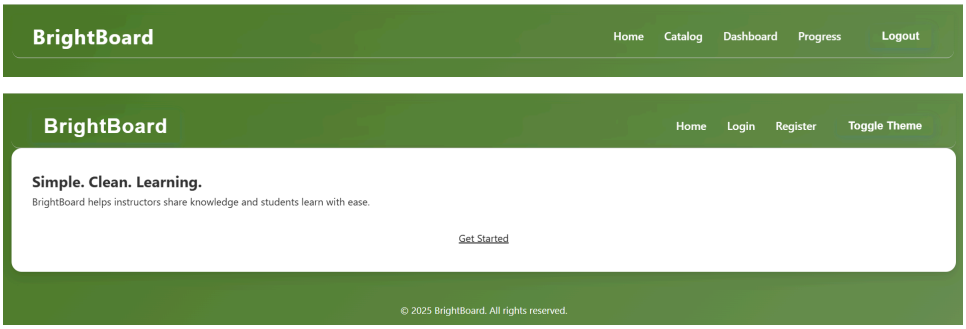
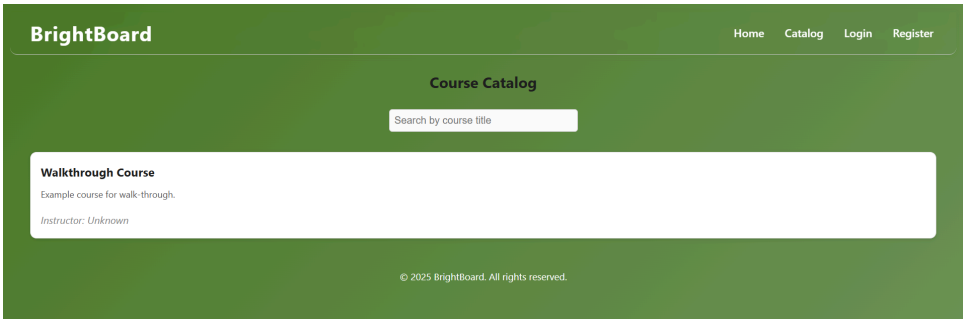
Req ID	Priority	Description
NFREQ-1	5	The system should have a responsive user interface (desktop & mobile).
NFREQ-2	4	The system should load content in under 2 seconds.
NFREQ-3	3	The system should support up to 10,000 concurrent users.
NFREQ-4	5	The system should use secure password encryption and comply with privacy laws (e.g., GDPR).
NFREQ-5	4	The system should maintain 99.9% uptime.

d. User Interface Requirements (Updated)

ID	Priority	Requirement Description
UI-1	5	The interface shall include a registration/login screen.  

UI-2	5	The student dashboard shall show enrolled courses and progress. 
UI-3	5	The instructor dashboard shall allow course and quiz creation. 

		 
UI-4	4	Lesson view should support formatted text OR pdf materials 

UI-5	3	Quizzes should present questions one at a time with immediate feedback. 
UI-6	3	Navigation should be consistent and minimal across all views. 
UI-7	1	Course list should support search and filter features. 

4. Use Cases (Updated)

---

a. Stakeholders

BrightBoard will serve a diverse group of stakeholders involved in microlearning and lightweight educational platforms. Below is a categorized list of stakeholders for BrightBoard:

- Educators: These are individuals teaching private or small group sessions who require simple tools for content delivery and assessment. They are primary users who directly benefit from streamlined course creation and progress tracking.
- Students: Learners who enroll in short courses or workshops and rely on BrightBoard for lesson access, assessments, and progress tracking.
- Businesses: Use BrightBoard for quick, targeted training sessions. Stakeholders here include HR departments or team leads looking for efficient internal learning tools.
- Educational Institutions (Small Schools or Informal Learning Organizations): May adopt BrightBoard for auxiliary teaching or as a resource for workshops and tutoring.
- System Developers and UI Designers: Responsible for building and maintaining the system. Their stake lies in meeting project deadlines and ensuring system quality.

b. Actors and Goals

Primary Initiating Actors:

Actor	Role	Goal
Instructor	Manages courses, uploads content, and creates quizzes. Uses the system to monitor student performance.	The goal of the instructor is to: Login and register account, Create, access and interact with lessons and quizzes, Monitor class progress, Navigate cleanly through a minimal UI.
Student	Enrolls in courses, accesses lessons, takes quizzes, and reviews progress.	The goal of the student is to login and register an account, Access and interact with lessons and quizzes, Monitor personal or class progress, Navigate cleanly through a minimal UI.

Secondary Participating Actors:

Actor	Role
Web Application System	The role of the Web Application System is to deliver content and facilitate all interactions between users and data.
Devices (Desktop, Mobile, Tablet)	The role of the device is to become a medium through which actors access BrightBoard.

c. Use Cases (Updated)

i. Casual Description

UC1: Register/Login
- Description: The system enables users to register and authenticate securely.
- Responds to Requirements: REQ-1, REQ-2, NFREQ-1, NFREQ-4, UI-1

UC2: Course Management
- Description: Instructors can create, edit, and delete courses.
- Responds to Requirements: REQ-2, REQ-3, UI-3

UC3: Lesson Uploading
- Description: Instructors upload lesson content as PDF or text.
- Responds to Requirements: REQ-4, UI-4

UC4: Quiz Creation
- Description: Instructors design multiple-choice quizzes.
- Responds to Requirements: REQ-5, UI-3, UI-5

UC5: Enroll in Course
- Description: Students enroll in available courses.
- Responds to Requirements: REQ-6, UI-2, UI-7

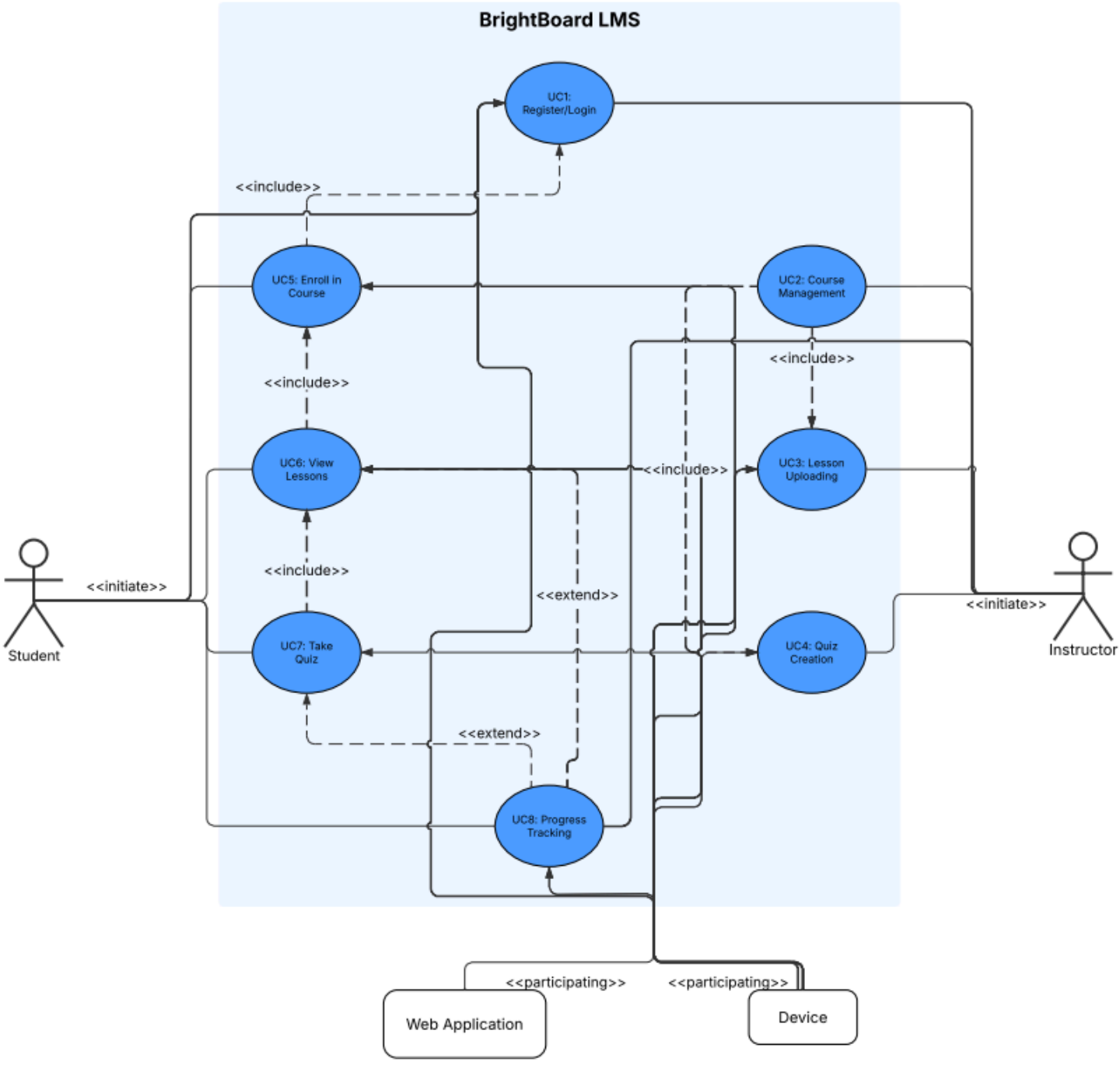
UC6: View Lessons
- Description: Students access course content.
- Responds to Requirements: REQ-7, UI-2, UI-4

UC7: Take Quiz and Score Automatically
- Description: Students take quizzes and receive immediate feedback.
- Responds to Requirements: REQ-8, UI-5

UC8: Progress Tracking

- Description: Tracks quiz completion and learning progress.
- Responds to Requirements: REQ-9, REQ-10, UI-2, UI-3

ii. Use Case Diagram



iii. Traceability Matrix

Req't	PW	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	UC11	UC12	UC13	UC14	UC15
REQ-1	5	X														
REQ-2	5	X	X													
REQ-3	5		X													
REQ-4	5			X												
REQ-5	5				X											
REQ-6	5					X										
REQ-7	5						X									
REQ-8	5							X								
REQ-9	3								X							
REQ-10	3								X							
NFREQ-1	5	X														
NFREQ-2	4															
NFREQ-3	3															
NFREQ-4	5	X														
NFREQ-5	4															
UI-1	5	X														
UI-2	5					X	X		X							
UI-3	5				X				X							

UI-4	4			**X**			**X**									
UI-5	3		**X**		**X**			**X**								
UI-6	3															
UI-7	1					**X**										

Use Case	**Related Requirements**	**Priority Weight**
Register/Login	REQ-1, REQ-2, NFREQ-1, NFREQ-4, UI-1	5 + 5 + 5 + 5 + 5 = 25
Course Management	REQ-2, REQ-3, UI-3	5 + 5 + 5 = 15
Lesson Uploading	REQ-4, UI-4	5 + 4 = 9
Quiz Creation	REQ-5, UI-3, UI-5	5 + 5 + 3 = 13
Enroll in Course	REQ-6, UI-2, UI-7	5 + 5 + 1 = 11
View Lessons	REQ-7, UI-2, UI-4	5 + 5 + 4 = 14
Take Quiz	REQ-8, UI-5	5 + 3 = 8
Progress Tracking	REQ-9, REQ-10, UI-2, UI-3	3 + 3 + 5 + 5

The weight calculated in the above table includes the Nonfunctional and User Interface Requirements. If we were to exclude these then the use case “Register/Login”, and “Course Management” have the highest priority weight.

Elaboration:

1. **Register/Login:**

The Register/Login functionality is the most essential entry point into the BrightBoard platform and is therefore a top priority for the first demo. It establishes the foundational user authentication and access control system upon which all other features depend. Without it, no user whether student or instructor can securely access their personalized dashboard or interact with otherwise protected content. This component not only demonstrates secure user authentication using industry standards like JWT but also introduces role-based navigation,

immediately showing how the platform tailors itself to different user types. By including Register/Login in the first demo, the system will be displayed with basic integrity and usability while laying the groundwork for deeper functionality in subsequent stages.

2. **Course Management:**

Course Management is the core of the instructor experience and highlights BrightBoard's value as a lightweight, focused learning platform. It allows instructors to create courses, upload lessons, and build quizzes all critical to delivering educational content. Including this feature in the Demonstration of course management early also gives a clear sense of how instructor role users will interact with the system on a daily basis. It reflects the workflow of an educator and reinforces BrightBoard's mission to simplify teaching and enhance learning.

iv. Fully-Dressed Description (updated)

UC-1: Register/Login

Related Requirements: REQ-1, REQ-2, NFREQ-1, NFREQ-4, UI-1
Initiating Actor: User
Participating Actors: None
Preconditions: The user is not logged into the system.
Postconditions: The user is authenticated and gains access to their dashboard.

Flow of Events for Main Success Scenario:

1. User enters registration/login credentials.
2. System validates user credentials.
3. Upon success, the user is logged in and redirected to their dashboard.

Flow of Events for Extensions (Alternate Scenarios):
2a. Credentials are invalid

- System displays an error message.
- User re-enters valid credentials.
  2b. New user registration

- User enters an email and password.
- System registers new user and sends confirmation.

UC-2: Course Management

Related Requirements: REQ-2, REQ-3, UI-3
Initiating Actor: Instructor
Participating Actors: None
Preconditions: Instructor is logged in.
Postconditions: Course is created, updated, or deleted in the system**.**

Flow of Events for Main Success Scenario:

1. Instructor creates a new course by entering required details.
2. System saves and displays the new course.

Flow of Events for Extensions:
1a. Instructor edits existing course

- System loads existing course for update.
- Changes are saved and reflected in the course list.
  1b. Instructor deletes a course
- System confirms deletion and removes course from the list.

UC-3: Lesson Uploading

Related Requirements: REQ-4, UI-4
Initiating Actor: Instructor
Participating Actors: None
Preconditions: Instructor is logged in and has access to a course they manage.
Postconditions: Lesson content is uploaded and associated with the selected course.

Flow of Events for Main Success Scenario:

4. Instructor selects a course to add a lesson.
5. Instructor uploads a PDF or enters text-based lesson content.
6. System validates and saves the content.
7. The lesson appears in the course's lesson list.

Flow of Events for Extensions:

- **2a. Invalid file format or upload failure**
    - System displays an error message.
    - Instructor retries with a valid file or content.

UC-4: Quiz Creation

Related Requirements: REQ-5, UI-3, UI-5
Initiating Actor: Instructor
Participating Actors: None
Preconditions: Instructor is logged in and has access to a course.
Postconditions: A quiz is created and linked to the appropriate course.

Flow of Events for Main Success Scenario:

1. Instructor selects the course and opens the quiz creation interface.
2. Instructor enters quiz title and adds multiple-choice questions with correct answers.
3. System validates and saves the quiz.
4. Quiz becomes available to enrolled students.

Flow of Events for Extensions:

- **2a. Instructor omits required fields**
    - System highlights missing fields and prompts for completion.
    - Instructor completes and resubmits the form.

UC-5: Enroll in Course

Related Requirements: REQ-6, UI-2, UI-7
Initiating Actor: Student
Participating Actors: None
Preconditions: Student is logged in.
Postconditions: Student is enrolled in the selected course.

Flow of Events for Main Success Scenario:

1. Student browses available courses.
2. Student selects a course and clicks "Enroll".
3. System processes the request and confirms enrollment.

4. Course appears in the student's dashboard.

Flow of Events for Extensions:

- **3a. Student already enrolled**
    - System shows a message: "You are already enrolled in this course."

UC-6: View Lessons

Related Requirements: REQ-7, UI-2, UI-4
Initiating Actor: Student
Participating Actors: None
Preconditions: Student is logged in and enrolled in a course.
Postconditions: Lesson content is displayed.

Flow of Events for Main Success Scenario:

1. Student navigates to an enrolled course.
2. Student selects a lesson.
3. System loads and displays the lesson content (PDF or text).

Flow of Events for Extensions:

- **2a. Lesson not available or file missing**
    - System displays an error message or placeholder: "Lesson unavailable."

UC-7: Take Quiz and Score Automatically

Related Requirements: REQ-8, UI-5
Initiating Actor: Student
Participating Actors: None
Preconditions: Student is logged in and enrolled in a course with an available quiz.
Postconditions: Quiz is submitted and scored automatically; student receives immediate feedback.

Flow of Events for Main Success Scenario:

1. Student opens an available quiz.
2. Student answers all questions and submits the quiz.
3. System automatically scores the quiz.
4. Student receives their score and feedback.

Flow of Events for Extensions:

- **2a. Student submits quiz with unanswered questions**
    - System prompts confirmation: "You have unanswered questions. Submit anyway?"
    - Student confirms or returns to complete.

UC-8: Progress Tracking

Related Requirements: REQ-9, REQ-10, UI-2, UI-3
Initiating Actor: System (background), Student (viewing)
Participating Actors: Instructor (viewing)
Preconditions: Student has engaged with quizzes or lessons.
Postconditions: Progress data is stored and available for viewing.

Flow of Events for Main Success Scenario:

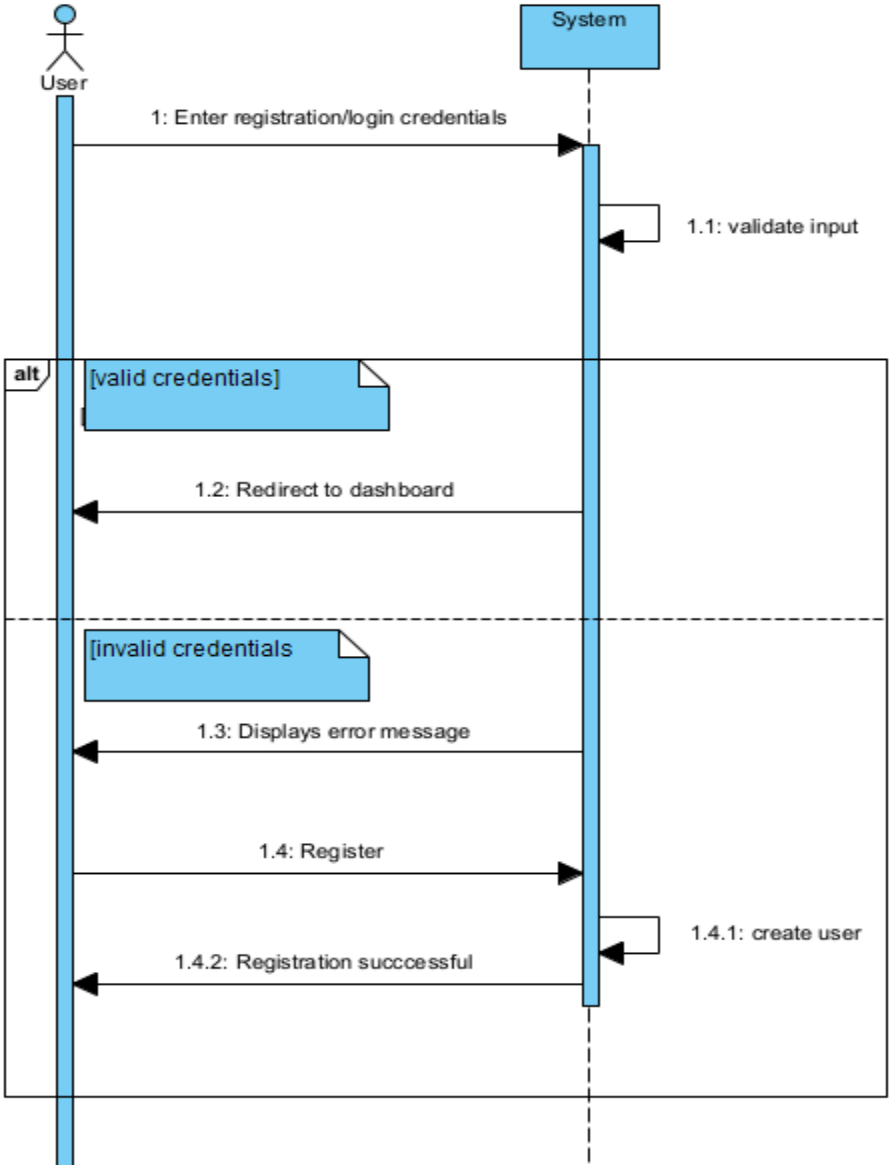
1. Student completes lessons or quizzes.
2. System updates progress data (e.g., completion %, scores).
3. Student views progress on their dashboard.
4. Instructor can view class-level or individual student progress.

Flow of Events for Extensions:

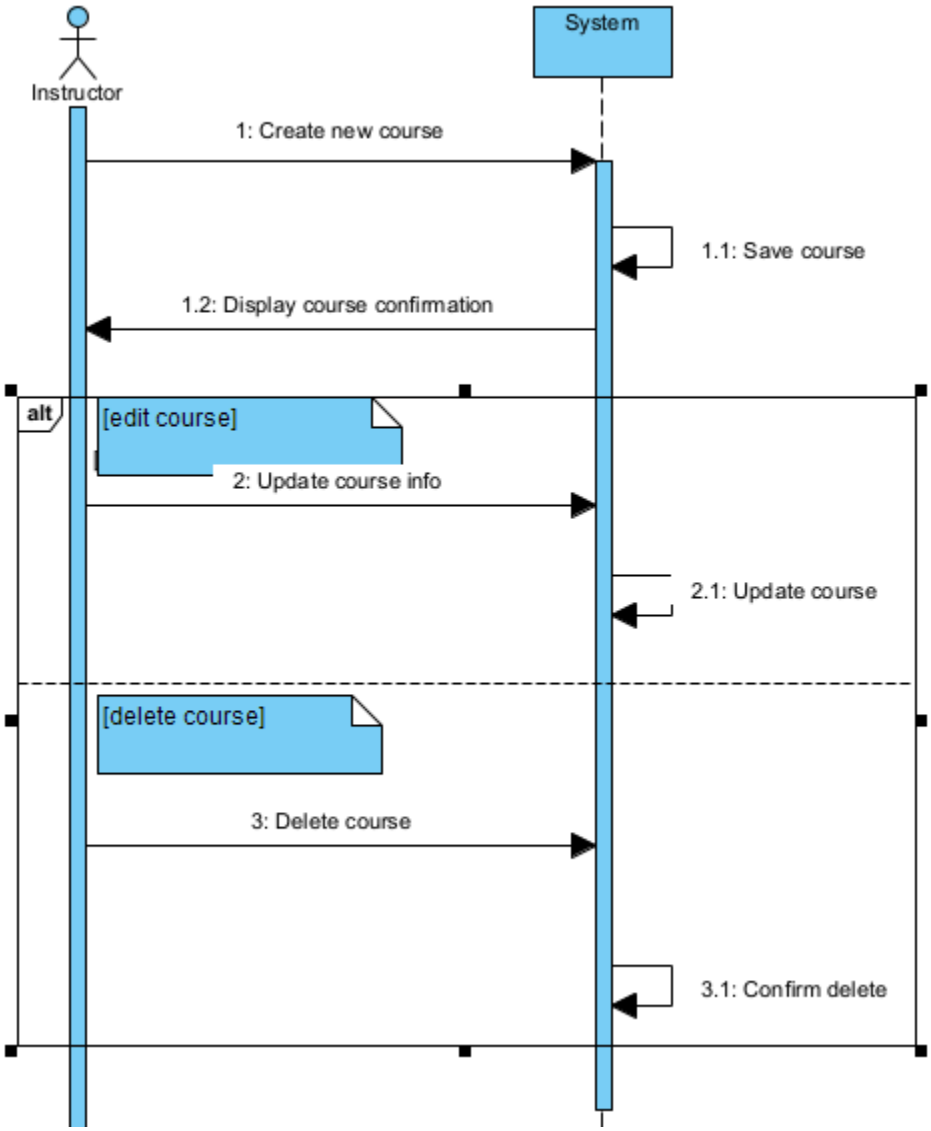
- **2a. Progress data not syncing (e.g., due to network issue)**
    - System retries syncing or shows "Progress update failed – try again."

d. System Sequence Diagrams (Updated)

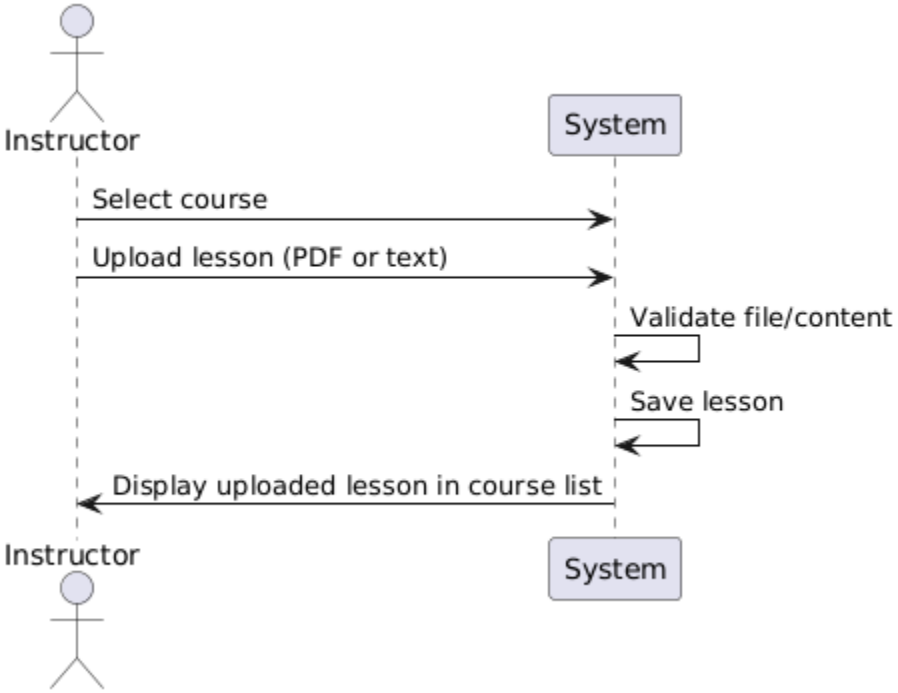
1. System Sequence Diagram– UC-1: Register/Login



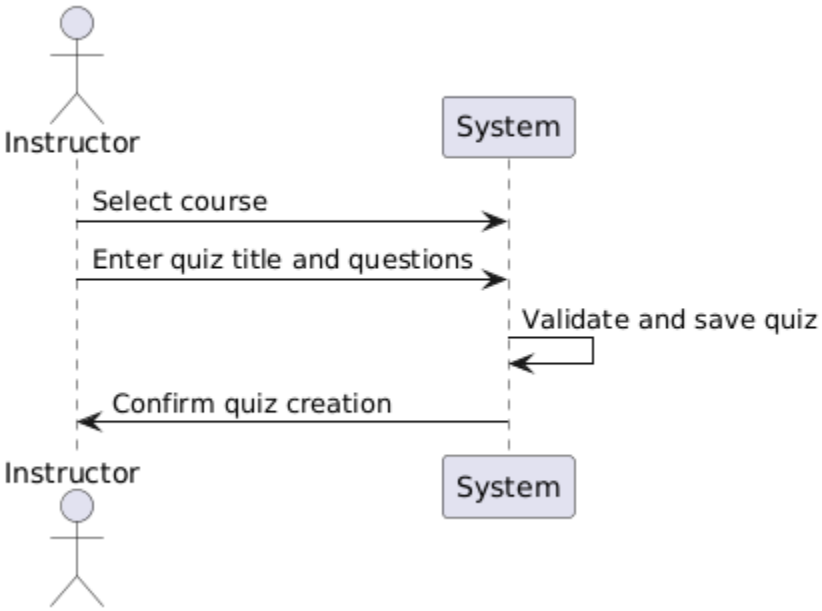
2. System Sequence Diagram – UC-2: Course Management



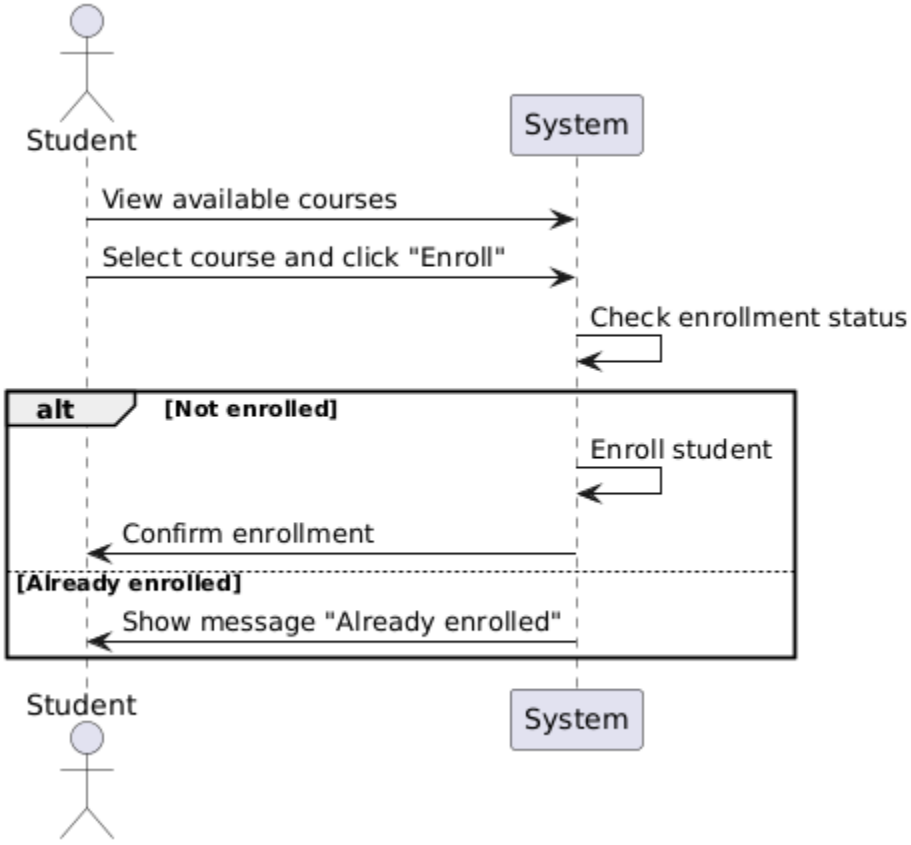
3. System Sequence Diagram – UC-3: Lesson Uploading



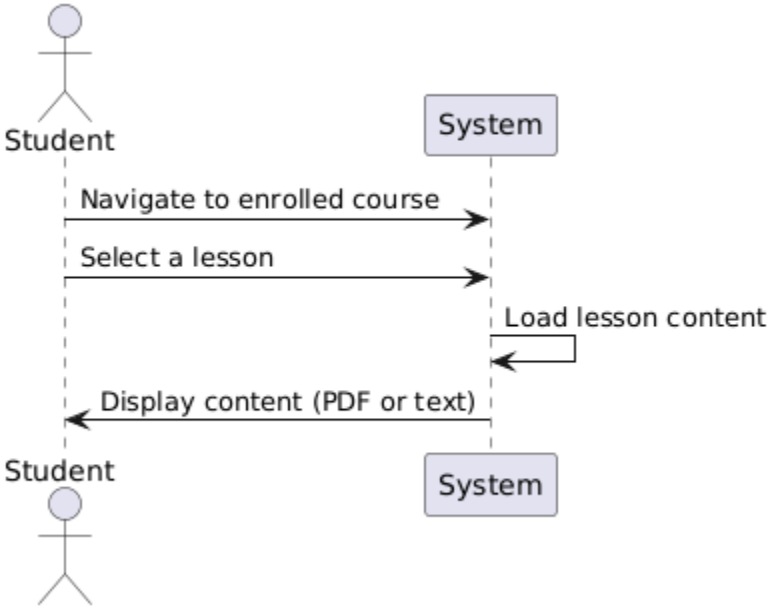
4. System Sequence Diagram – UC-4: Quiz creation



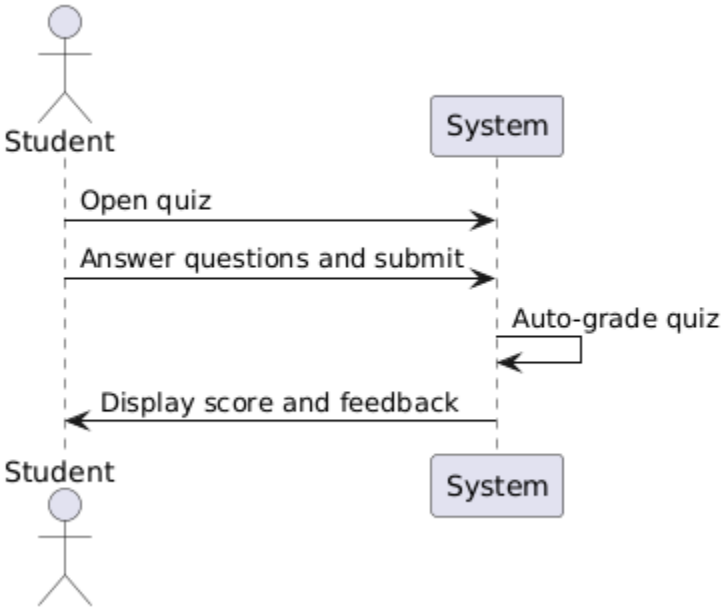
5. System Sequence Diagram – UC-5: Enroll in Course



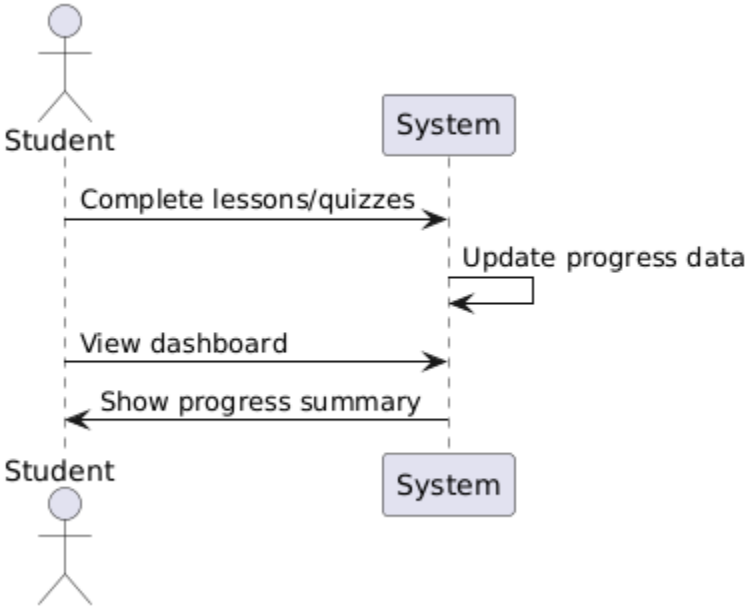
6. System Sequence Diagram – UC-6: View Lesson(s)



7. System Sequence Diagram – UC-7: Take Quiz and Score Automatically



8. System Sequence Diagram – UC-8: Progress tracking



5. User Interface Specification (Updated)

---

a. Preliminary Design

1. Course Management – User Interface Specification

a. After successfully signing in to their role-specific interface,Instructors will be able to navigate to the “Manage Courses” section, where they can create, view, edit, and delete courses. This section is accessible via a clearly labeled button on their dashboard.

b. Instructors can interact with the course list using intuitive action buttons like “Edit”, “Delete”, or “Create New Course”, streamlining course organization and updates within the BrightBoard platform.

2. **Register/Login- User Interface Specification**

a. On accessing the BrightBoard platform, users are greeted with a Login/Register screen. This interface allows users to either sign in using existing credentials or register for a new account.

New users click the “Register” button, which opens a form where they enter: full name, email address, password, select role (Instructor or Student),confirm password. Returning users use the “Login” form by providing email address and password. After successful login, users are directed to their role-specific dashboard: Instructors are taken to the “Instructor dashboard” interface. Students are taken to the “Student dashboard” view.

b. If login credentials are incorrect or required fields are missing during registration, error messages are displayed clearly under each relevant field (e.g., "Invalid email or password").

The navigation is optimized for ease of use, allowing users to quickly transition from registration or login to their core platform tasks.

b. User Effort Estimation (Updated)

Scenario: Completing a 2-question quiz

- *Figure 1: BrightBoard System Login page*

Figure 2: Student Dashboard

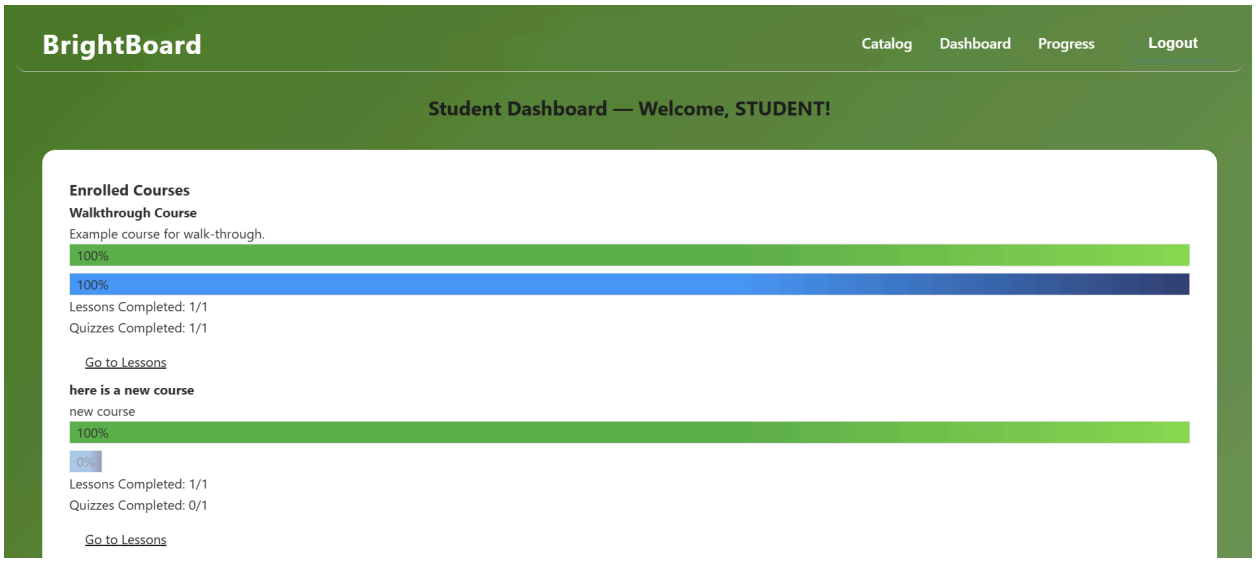


Figure 3: Course/lesson selection

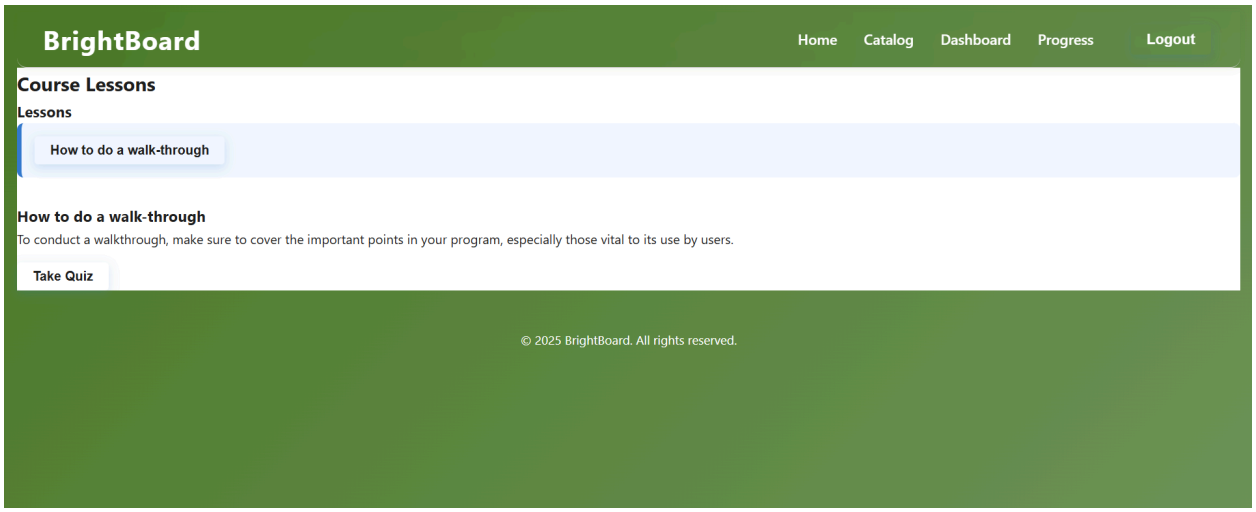
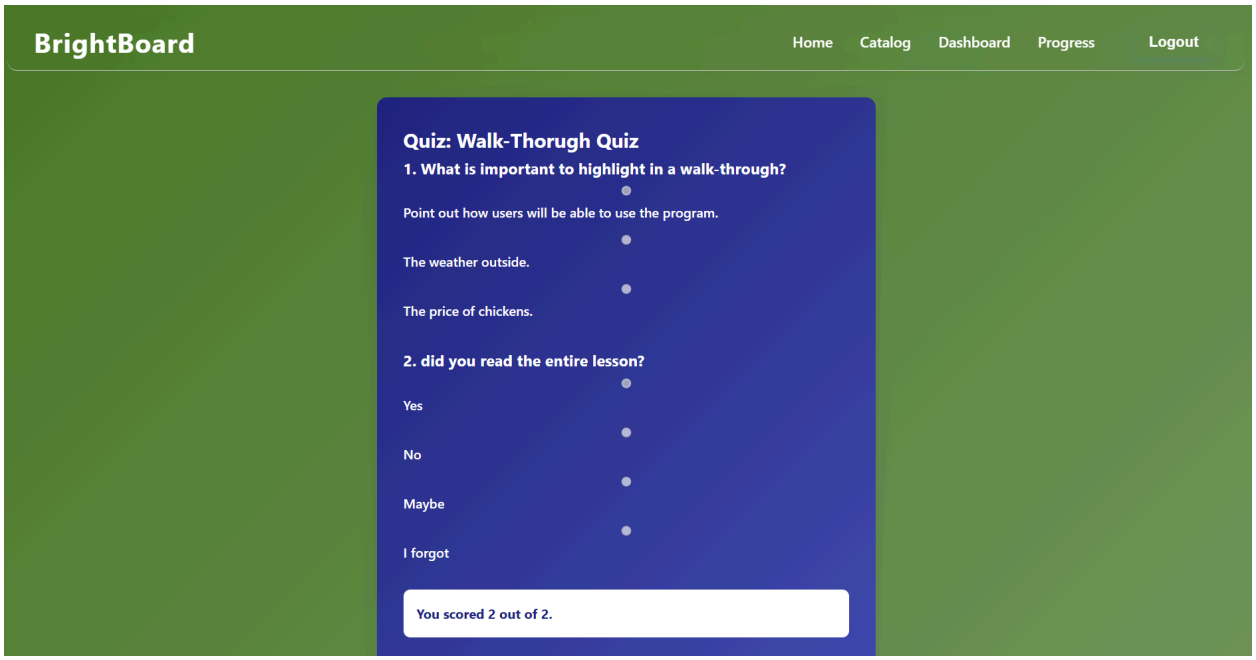


Figure 4: Sample Quiz



1. Quiz:

The following sequence of actions would allow a student enrolled in a course to complete a 2 question quiz

NAVIGATION: Total 9 mouse clicks, as follows:

a. From the login screen, click on the username and password fields.
b. Click on the "Login" button.
c. From the student dashboard click on the desired course.
d. Click the desired lesson within the course.
e. Click the begin quiz button.
f. Click on answers for quiz questions (in this example, 2).
g. Click the submit quiz button.

DATA ENTRY: Total of 2 sets of keystrokes, as follows. **a.** Type Username **b**. Type password

Estimated Time: 1–2 minutes per quiz.

2. Enroll in Course

The following sequence of actions would allow a student to enroll in an available course using 7 mouse clicks and 2 sets of keystrokes.

NAVIGATION

Total: 7 mouse clicks, as follows:

a. From the login screen, click on the username and password fields.
b. Click on the "Login" button.
c. From the student dashboard, click on the "Course Catalog" tab.
d. Scroll or browse the list and click on the desired course.
e. Click on the "Enroll" button to open the confirmation dialog.
f. Click on the "Confirm" button to finalize enrollment.
g. Click on "My Courses" to view the newly enrolled course.

DATA ENTRY

Total: 2 sets of keystrokes, as follows:

a. Type username
b. Type password

Estimated Time: 30 seconds to 1 minute per course enrollment.

Here's the detailed effort estimation using the Use Case Points (UCP) method:

Project Duration Effort Estimation Using Use Case Points

Total Use Case Points (UCP)

Based on the traceability matrix and priorities:

Use Case	Priority Weight
Register/Login	5
Course Management	5
Lesson Uploading	5
Quiz Creation	5
Enroll in Course	5
View Lessons	5
Take Quiz	5
Progress Tracking	3
Export Progress Reports	2
Send Notifications	2

Total UCP = 5 + 5 + 5 + 5 + 5 + 5 + 5 + 3 + 2 + 2 = **42 UCP**

Productivity Factor (PF)

Using a standard industry average of PF = 20–28 hours/UCP: PF = 28 hours for balanced complexity.

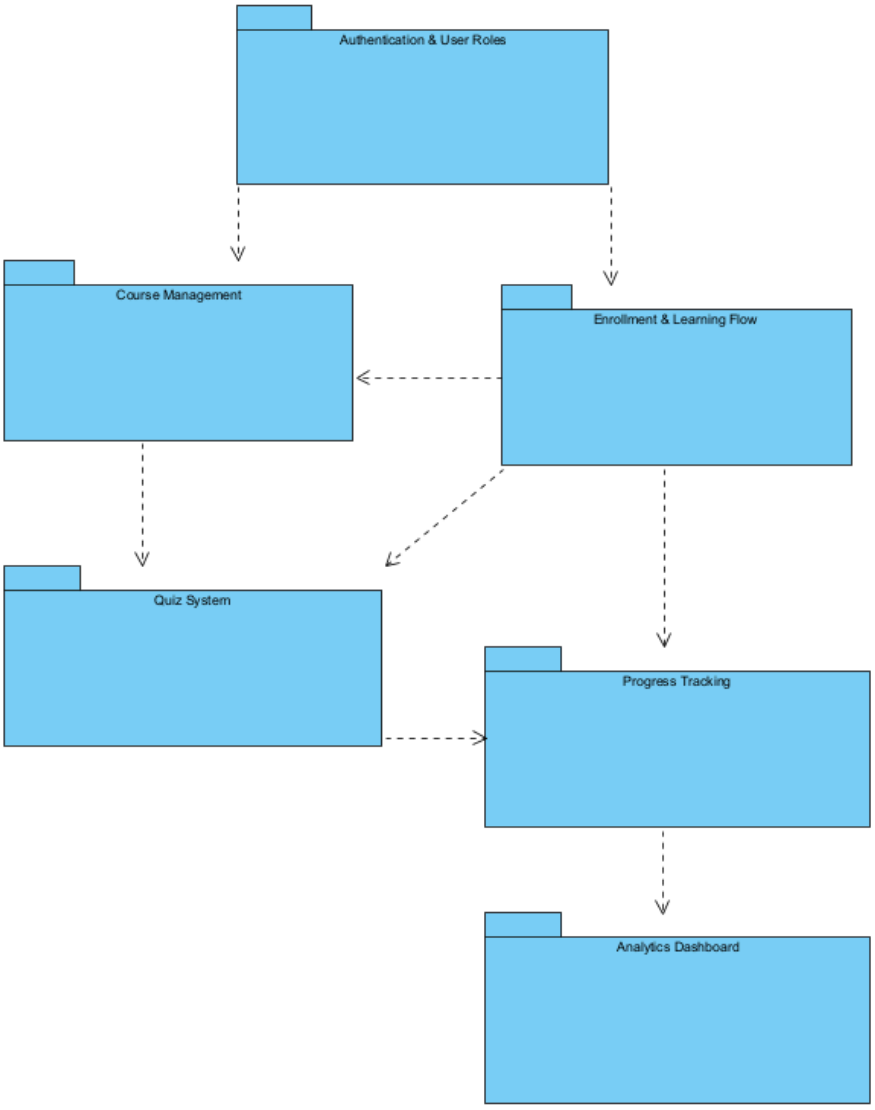
Effort = UCP × PF = 42 × 28 = 1,176 hours **Updated**

6. System Architecture and System Design

---

a. Identifying Subsystems

The architecture of BrightBoard was designed with simplicity, modularity, and scalability in mind, catering specifically to independent educators and small learning cohorts. The system provides a centralized platform for course delivery, quiz administration, and student progress tracking, removing the need for multiple disjointed tools.



UML Package Diagram Description:

While not presented as a formal diagram in the original report, the subsystems of BrightBoard can be visualized in a package-oriented UML structure. The **Authentication** module connects with both **Course Management** and **Enrollment**, setting the foundation for user access and interaction. **Course Management** interfaces with the **Quiz System**, while **Progress Tracking** and **Analytics** tie together performance data from across the platform. This modular setup supports extensibility and maintenance.

b. Architectural Styles

BrightBoard is structured around a client-server architecture, a well-established architectural style that separates the user interface (client) from the core application logic and data handling (server). This separation of concerns ensures that the system remains maintainable, scalable, and easy to deploy in diverse environments. The frontend and backend operate independently yet communicate seamlessly via a well-defined interface, allowing for modular development and the potential for future upgrades or third-party integrations.

On the client side, BrightBoard uses React, a modern JavaScript library for building user interfaces. React was chosen for its component-based architecture, which promotes code reusability, state management, and fast rendering performance. This makes it ideal for delivering an interactive and responsive user experience across various devices, including desktops, tablets, and smartphones. Components are responsible for rendering views, managing user interactions (like course enrollment or quiz taking), and sending asynchronous requests to the backend via RESTful APIs.

The backend is implemented using Node.js with the Express.js framework. This combination allows for fast, non-blocking server-side logic and robust API creation. Express handles routing, middleware, and HTTP request management, serving as the backbone of server-side operations. Business logic, such as user authentication, lesson processing, quiz scoring, and progress tracking, is processed server-side to ensure consistency, security, and centralized control.

BrightBoard also integrates the Model-View-Controller (MVC) architectural pattern within its backend logic. This layered structure enhances clarity and separation within the codebase:

- Models are created using Mongoose, an ODM (Object Data Modeling) library for MongoDB. These schemas define the structure and constraints of data entities such as users, courses, lessons, and quizzes.

- Controllers manage the business logic and serve as intermediaries between models and views. For example, when a student submits a quiz, the controller validates the data, calculates the score, and updates the relevant records.

- Views, while traditionally rendered server-side in MVC, are managed entirely on the client side via React components in BrightBoard's architecture. This hybrid model allows for a clean API-driven interface and decouples backend logic from frontend rendering responsibilities.

This combination of client-server and MVC styles provides several advantages. It allows teams to work in parallel on frontend and backend components, improves scalability by enabling horizontal distribution (e.g., load-balancing frontend servers), and simplifies debugging by keeping concerns well-organized. Additionally, the system is well-prepared for future enhancements, such as adding real-time features via WebSockets or expanding to mobile platforms using React Native.

In summary, BrightBoard's architectural design ensures both flexibility and robustness, using tried-and-true patterns that support the project's goals of maintainability, user responsiveness, and straightforward deployment.

c. Mapping Subsystems to Hardware

BrightBoard is designed to operate across multiple machines, following a distributed system model. The frontend runs in the user's web browser, serving both students and instructors. The backend operates on a remote server, handling all authentication, course logic, and data storage. The database, MongoDB, may be hosted either locally or on a cloud instance, depending on deployment needs. This setup ensures that user-facing interfaces remain lightweight, while processing and data management occur server-side.

d. Connectors and Network Protocols

BrightBoard relies on a combination of well-established web communication protocols and frameworks to ensure reliable, secure, and efficient data exchange between system components. The architecture is intentionally designed around widely adopted technologies to promote compatibility, ease of deployment, and long-term scalability.

At the core of communication between the frontend (React) and the backend (Node.js/Express.js) is the HTTP/HTTPS protocol, with all client-server interactions conducted over HTTPS to ensure encrypted and secure data transmission. HTTPS (Hypertext Transfer Protocol Secure) is essential for protecting sensitive user data, such as login credentials and quiz results, against man-in-the-middle attacks and other forms of interception during transmission.

The application uses a RESTful API architecture, enabling the frontend to interact with backend resources in a stateless manner. Each API endpoint corresponds to specific actions within the system, such as creating courses, submitting quiz responses, or retrieving user progress. RESTful APIs ensure a consistent and scalable interface that allows future enhancements, such as third-party integrations or mobile clients, without major architectural changes.

For authentication and access control, BrightBoard uses JWT (JSON Web Tokens). When a user logs in, the server issues a token that is securely stored on the client side (typically in local storage or cookies). This token is included in the header of subsequent requests, allowing the server to verify the user's identity and role without the need for session persistence. JWT is well-suited for distributed systems and improves performance by avoiding frequent lookups in a session store.

The backend also uses MongoDB, a NoSQL, document-oriented database, to persist application data. Communication with MongoDB is handled through Mongoose, an Object Data Modeling (ODM) library for Node.js that defines schemas, validation rules, and query logic. This structured approach helps maintain data integrity while still benefiting from the flexibility of a NoSQL system.

In terms of connectors, the application stack is composed entirely of web-based protocols and libraries, including:

- **HTTPS** for encrypted client-server communication
- **HTTP REST** endpoints for functional API interaction
- **JWT** for stateless authentication and session management
- **MongoDB** as the database backend, connected via Mongoose ODM

There is no use of low-level socket communication (e.g., TCP sockets or WebSockets), as BrightBoard does not require real-time updates or peer-to-peer messaging. Its event-driven but non-real-time nature makes RESTful HTTP over HTTPS the ideal choice.

This architecture and protocol stack together offer a secure, scalable, and maintainable foundation that aligns well with the needs of a modern, web-based learning management system.

e. Global Control Flow

BrightBoard operates as an event-driven system, meaning the application responds to user-generated actions such as logging in, enrolling in a course, viewing lessons, or completing

quizzes. These actions do not follow a strict sequence; instead, they trigger specific server-side responses and dynamically update the user interface based on the user's role and progress. This interaction model ensures flexibility and a personalized learning experience.

- **Execution Orderness:**

The flow of execution in BrightBoard is non-linear and entirely driven by user input. For instance, one student might choose to take a quiz immediately after enrolling in a course, while another might prefer to read through all available lessons before attempting any assessments. This flexibility supports self-paced learning and accommodates a variety of learning styles. The system is designed to allow users to navigate content in the order that best fits their needs, rather than enforcing a rigid, predetermined path.

- **Time Dependency:**

BrightBoard is not a real-time system and does not depend on timers or continuous scheduling mechanisms. However, the system does incorporate timestamps on quiz submissions to record the exact date and time each quiz was completed. This information is valuable for both students and instructors, as it provides transparency, helps track progress over time, and may be used for analytics or time-based feedback. Despite this time-aware feature, the overall system remains asynchronous and event-driven, with no hard real-time constraints or requirements for continuous uptime synchronization.

f. Hardware Requirements

BrightBoard is designed with modest hardware requirements to ensure ease of deployment and broad accessibility for both instructors and students. The system relies on commonly available technology components without the need for specialized hardware or sensors. Below is a breakdown of the essential hardware and system resources:

Client-Side Requirements:

- Screen Display:
  Minimum resolution of 640 × 480 pixels to support responsive interface design and lesson readability.
- Web Browser:
  A modern web browser such as Chrome, Firefox, or Edge capable of rendering dynamic, JavaScript-based user interfaces.
- Communication Network:
  An active internet connection with a recommended minimum bandwidth of 56 Kbps (modern broadband is preferred for optimal performance and loading times).

- Input Devices:
  Standard keyboard and mouse or touchscreen interface for navigating lessons, quizzes, and dashboard interactions.

Server-Side Requirements:

- Processing Environment:
  A hosting environment capable of running a Node.js backend and connecting to a MongoDB database.
- Disk Storage:
  Minimum of 2 GB storage capacity, sufficient for handling uploaded PDF files, user data, and database growth.
- Network Availability:
  Continuous uptime with at least 99.9% availability to ensure user access and minimal downtime.
- Communication Protocols:
  Support for secure HTTPS connections to protect data transmission and comply with privacy requirements.
- Memory & CPU:
  Modest RAM and CPU resources suitable for handling lightweight requests, expected user concurrency, and real-time feedback (e.g., 2 GB RAM and a dual-core processor or cloud equivalent)

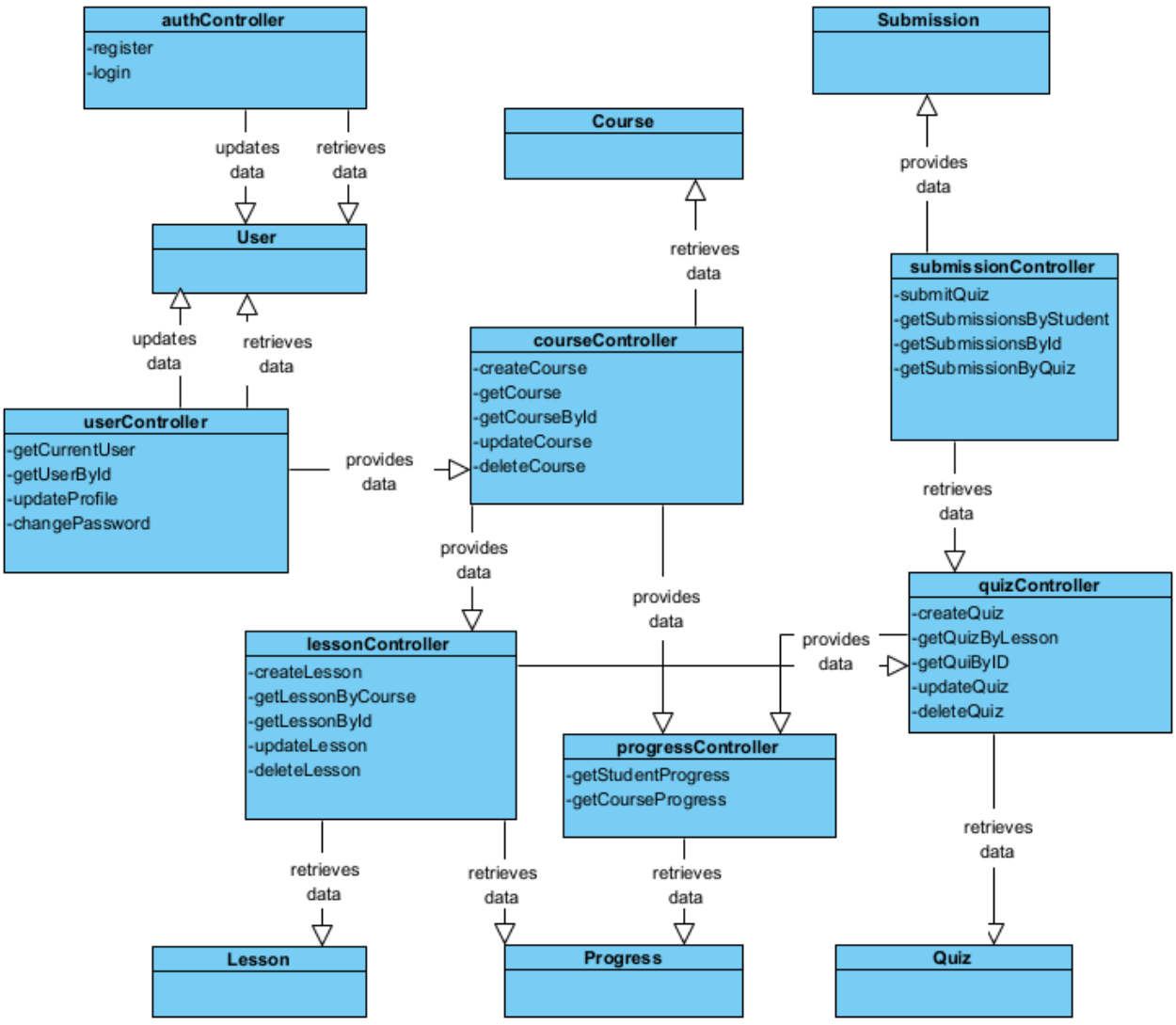
7. Analysis and Domain Modeling (Updated)

---

a. Conceptual Model (Updated)

In this part we have designed a conceptual model for our BrightBoard LMS. This is meant to display the structure that our application will operate on. The conceptual model is meant to convey concepts used, the association between them, attributes

Domain Model (Updated)



i. Concept definitions

Concept Name	Type (D=Doing, K=Knowing)	Responsibility
authController	D	Handles user registration, login, token generation, and authentication flows.
userController	D	Manages user profile access, including fetching current user and user by ID.
User	K	Represents platform users (instructors or students), stores login and role data.
courseController	D	Handles course creation, deletion, listing, and fetching by ID.
Course	K	Represents a course with title, description, and associated instructor.
lessonController	D	Manages creation, listing, and retrieval of lessons tied to courses.
Lesson	K	Represents instructional content (text or PDF) linked to a course.
quizController	D	Creates and fetches quizzes associated with lessons.
Quiz	K	Represents a multiple-choice assessment linked to a lesson.
submissionController	D	Handles quiz submission, scoring, and result storage.
Submission	K	Handles quiz submission, scoring, and result storage.
progressController	D	Tracks student progress through courses, lessons, and quizzes.
Progress	K	Tracks student progress through courses, lessons, and quizzes.

ii. Association definitions

Concept Pair	Association Description	Association Name
User ↔ Course	A User with role instructor can create multiple Courses.	teaches / isTaughtBy
Course ↔ Lesson	Each Course contains multiple Lessons.	hasLessons / belongsToCourse
Course ↔ Quiz	Quiz is indirectly linked to Course through Lesson.	hasQuizzes / belongsToCourse
Lesson ↔ Quiz	Each Lesson can have one associated Quiz.	hasQuiz / forLesson
User ↔ Submission	A User (student) creates a Submission when taking a Quiz.	submits / submittedBy
Quiz ↔ Submission	Each Submission is linked to one Quiz.	submittedFor / hasSubmissions
User ↔ Progress	A User (student) has a Progress tracker per Course.	tracksProgress / belongsToStudent
Progress ↔ Lesson	Progress tracks Lessons completed by a student in a course.	lessonsCompleted
Progress ↔ Quiz	Progress tracks Quiz completions per student per course.	quizzesCompleted
courseController ↔ Course	courseController performs create, fetch, and update actions on Course.	manages
lessonController ↔ Lesson	lessonController handles creation and retrieval of Lessons.	manages
quizController ↔ Quiz	quizController handles creation and fetching of Quiz data.	manages
submissionController ↔ Submission	submissionController handles quiz submission and storage of Submission.	manages

progressController ↔ Progress	progressController retrieves or updates Progress based on user/course actions.	manages
authController ↔ User	authController handles registration, login, and authentication of User.	authenticates
userController ↔ User	userController provides access to user profile and role information.	managesProfile
User ↔ Course	A User with role instructor can create multiple Courses.	teaches / isTaughtBy

iii. Attribute definitions (Updated)

Concept	Attributes	Attribute Description
courseController	createCourse	Creates a new course by an instructor.
	getCourse	Retrieves a list of all courses available.
	getCourseById	Fetches detailed information for a single course using its ID.
	updateCourse	Updates course title or description.
	deleteCourse	Deletes a course (typically by instructor or admin).
	enrollInCourse	Enrolls a student in a course
	addMaterial	Adds material to a course
	getEnrolledCourses	Gets courses a student is enrolled in

lessonController	createLesson	Adds a new lesson (text or PDF) under a specific course.
	getLessonByCourse	Retrieves all lessons linked to a specific course.
	getLessonById	Fetches a specific lesson using its unique ID.
	updateLesson	Updates content or metadata of a lesson.
	deleteLesson	Deletes a specific lesson.
progressController	getStudentProgress	Retrieves quiz/lesson completion data for a specific student.
	getCourseProgress	Shows aggregated progress data for a course's students.
quizController	createQuiz	Creates a quiz for a specific lesson.
	getQuizByLesson	Retrieves the quiz linked to a particular lesson.
	getQuizById	Fetches a quiz by its ID
	listQuizzes	Retrieves Quizzes
	updateQuiz	Update quizzes
	deleteQuiz	Deletes specific quizzes
submissionController	submitQuiz	Submits a student's answers, calculates score, and stores the submission.
	getSubmissionByStudent	Retrieves all submissions made by a specific student.
userController	getCurrentUser	Fetches data of the currently authenticated user.
	getUserById	Returns profile data for a user by ID.
	updateProfile	Allows users to update their name or email.

	changePassword	Lets users securely change their password.
authController	register	Registers a new user with hashed password and assigned role.
	login	Authenticates user credentials and returns a JWT token.

iv. Traceability matrix (2) - Use Cases to Domain Concept Objects (Updated)

Domain Concepts	**Use Case**	**UC1**	**UC2**	**UC3**	**UC4**	**UC5**	**UC6**	**UC7**	**UC8**
	PW	25	15	9	13	11	14	8	5
authController		X							
userController		X				X			
User		X	X				X	X	X
courseController			X						X
Course			X	X	X	X	X		
lessonController				X			X		
Lesson				X	X		X		X
quizController					X				
Quiz					X			X	X
submissionController								X	
Submission								X	X
progressController							X	X	X
Progress						X	X	X	X

This Use Case to Domain Concept Traceability Matrix establishes clear links between the major system use cases (UC1 to UC8) and the domain objects and controllers that support them.

- authController: Handles User Registration and Login (UC1) using JWT authentication.

- userController and User domain model: Support user account creation, role-based routing, dashboard rendering, and profile-related operations (UC1, UC2, UC5–UC8).
- courseController and Course: Involved in use cases such as Course Creation, Enrollment, and Course Browsing (UC2–UC6, UC7).
- lessonController and Lesson: Support Lesson Upload, Viewing, and Content Management (UC3–UC4, UC6, UC8).
- quizController and Quiz: Implement Quiz Creation, Quiz Taking, and Quiz Review functionality (UC4, UC7, UC8).
- submissionController and Submission: Enable storage and review of quiz attempts for future feedback and progress reporting (UC7, UC8).
- progressController and Progress: Powers the Progress Tracker for both students and professors to monitor course and quiz completion (UC5–UC8).

This traceability ensures the system is modular, maintainable, and clearly structured to fulfill all defined functional requirements. Each component can be updated or extended independently while maintaining system coherence.

b. System Operation Contracts (Updated)

UC-1:

- *Preconditions:* User is not logged in.
- *Postconditions:* User is authenticated.
- *Flow:* Submit credentials ~ Validate ~ Redirect based on role.
- *Extensions:* Invalid login or email already used ~ Error prompts.

Use Case ID	Use Case Name	Contract	Inputs	Outputs	Exceptions
UC-1	Register/Login	When a user enters login or registration details, the system validates the information and authenticates them.	Email,password, role:(on registration)	User is logged in and redirected to their dashboard	If credentials are invalid or already registered, show an error message.

UC-2:

- *Preconditions:* Instructor must be logged in.
- *Postconditions:* Course list is modified.

- *Flow:* Create/Edit/Delete course ~ Save changes ~ Update dashboard.
- *Extensions:* Update or delete fails ~ Error prompt shown.

Use Case ID	Use Case Name	Contract	Inputs	Outputs	Exceptions
UC-2	Course Management	When an instructor creates, edits, or deletes a course, the system processes the changes and updates the course list.	Course title, description, instructor ID	Course is created, updated, or deleted accordingly	If course data is invalid or save/delete fails, display an error message.

UC-3:

- *Preconditions:* Instructor must be logged in and have access to a course.
- *Postconditions:* A lesson is added to the course.
- *Flow:* Select course ~ Upload PDF or enter text ~ Validate ~ Save content.
- *Extensions:* Invalid file format or upload fails ~ Show error prompt.

Use Case ID	Use Case Name	Contract	Inputs	Outputs	Exceptions
UC-3	Lesson Uploading	When an instructor uploads a lesson, the system validates and adds the content to the selected course.	Lesson file (PDF) or text, course ID	Lesson is saved and associated with the course	If the upload fails or format is invalid, an error is shown

UC-4:

- *Preconditions:* Instructor must be logged in and own the course.
- *Postconditions:* Quiz is available to enrolled students.
- *Flow:* Enter quiz title ~ Add MCQs ~ Validate ~ Save quiz.
- *Extensions***:** Incomplete quiz or errors ~ Prompt for corrections.

Use Case ID	Use Case Name	Contract	Inputs	Outputs	Exceptions
UC-4	Quiz Creation	When an instructor creates a quiz, the system validates the input and adds it to the course.	Quiz title, list of multiple-choice questions, correct answers	Quiz is saved and visible to students	If required fields are missing, show an error message

UC-5:

- *Preconditions:* Student is logged in.
- *Postconditions:* Student is enrolled in the course.
- *Flow:* Browse courses ~ Select course ~ Click Enroll ~ Confirm enrollment.
- *Extensions***:** Already enrolled ~ Show "already enrolled" message.

Use Case ID	Use Case Name	Contract	Inputs	Outputs	Exceptions
UC-5	Enroll in Course	When a student enrolls in a course, the system adds them to the enrollment list and updates their dashboard.	Student ID, course ID	Enrollment is confirmed and course appears in dashboard	If student is already enrolled, show a relevant message

UC-6:

- *Preconditions:* Student is logged in and enrolled in a course.
- *Postconditions:* Lesson content is displayed.
- *Flow:* Select course ~ Choose lesson ~ Load and display content
- *Extensions:* Lesson not found ~ Show "lesson unavailable" message.

Use Case ID	Use Case Name	Contract	Inputs	Outputs	Exceptions
UC-6	View Lessons	When a student selects a lesson, the system loads and displays the corresponding content.	Student ID, lesson ID	Lesson content is shown (PDF or text)	If lesson file is missing, display an error or placeholder

UC-7:

- *Preconditions:* Student is logged in and has access to a quiz.
- *Postconditions:* Quiz is scored, and feedback is displayed.
- *Flow:* Open quiz ~ Submit answers ~ Auto-grade ~ Show results.
- *Extensions:* Incomplete answers ~ Prompt confirmation before submission.

Use Case ID	Use Case Name	Contract	Inputs	Outputs	Exceptions
UC-7	Take Quiz and Score Automatically	When a student submits a quiz, the system scores it and shows results immediately	Student ID, quiz ID, selected answers	Quiz score and feedback	If answers are missing, prompt confirmation before submission.

UC-8:

- *Preconditions:* Student has completed lessons or quizzes.
- *Postconditions:* Progress data is updated and available.
- *Flow:* Complete learning activity ~ Update progress ~ View on dashboard.
- *Extensions:* Sync fails ~ Show retry message.

Use Case ID	Use Case Name	Contract	Inputs	Outputs	Exceptions
UC-8	Progress Tracking	When a student completes activities, the system tracks and updates progress data.	Student ID, activity ID (lesson/quiz), score/status	Updated progress data shown in dashboard	If sync fails, display a retry or error notification

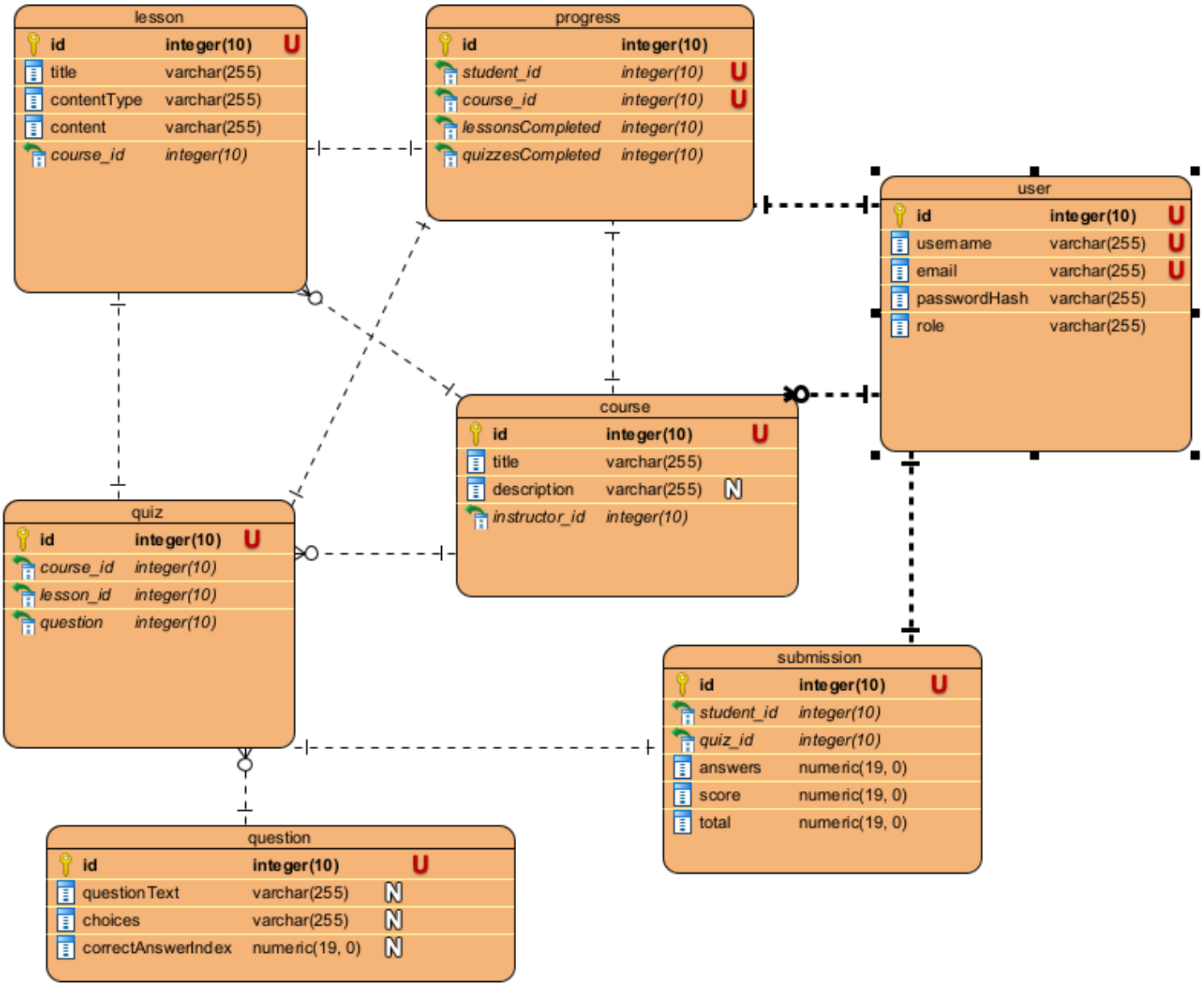
c. Data Model and Persistent Data Storage

BrightBoard requires persistent data storage, as it supports long-term user interactions, course management, quiz tracking, and progress analytics.

The application uses a document-oriented NoSQL database management system to store user profiles, course data, lesson content, quiz information, and student progress. We selected MongoDB as our database solution because it integrates seamlessly with our backend (built on Node.js with Express) and supports flexible, schema-based modeling via Mongoose, an ODM tool. MongoDB allows us to efficiently handle the hierarchical and nested data structures inherent in educational content, such as lessons within courses and questions within quizzes.

Given that BrightBoard is designed to be lightweight and scalable, MongoDB's ability to scale horizontally and accommodate varying document structures made it a natural fit. Additionally, our team has significant prior experience with JavaScript and JSON-based data modeling, which made MongoDB both a practical and efficient choice. Data is organized across a focused set of collections that mirror the system’s core entities: users, courses, lessons, quizzes, and progress records.

Database Schema



User Table

The user table, which stores information for all platform users, including both students and instructors. Each user has a unique ID, username, email, encrypted password (passwordHash), and a role that designates their permission level. This structure allows BrightBoard to enforce role-based access and navigation, enabling different capabilities depending on whether the user is an instructor or a student.

- Instructors are associated with courses they create through a one-to-many relationship from the user table to the course table via the instructor_id foreign key. Students, meanwhile, are linked to their quiz submissions and progress through foreign keys in the submission and progress tables, respectively.

Course Table

The course table represents instructional units created by instructors. It includes a title, a description, and an instructor_id that references the creator. Each course may contain multiple lessons and quizzes, which are managed through one-to-many relationships with the lesson and quiz tables.

Lesson Table

Lessons are the instructional materials associated with courses. Each entry in the lesson table is linked to a specific course via course_id and contains metadata such as a title, contentType (e.g., text or PDF), and the actual content. Lessons may also serve as containers for quizzes, supporting structured progression through course material.

Quiz and Question Tables

The quiz table stores assessment modules used to evaluate students' understanding. Each quiz is tied to both a course and a lesson using foreign keys. Though the current structure uses a question field within the quiz, it's complemented by the question table, which holds reusable question entries. Each question includes a prompt (questionText), multiple choice choices, and the correctAnswerIndex.

- This arrangement allows quizzes to be composed of multiple questions, promoting flexibility in quiz creation and question reuse. The implied many-to-many relationship between quiz and question supports dynamic quiz building.

Submission Table

Student interactions with quizzes are captured in the submission table. Each submission links a student_id and a quiz_id, recording the student's answers, the calculated score, and the total possible points. This structure supports detailed analytics and feedback on student performance.

Progress Table

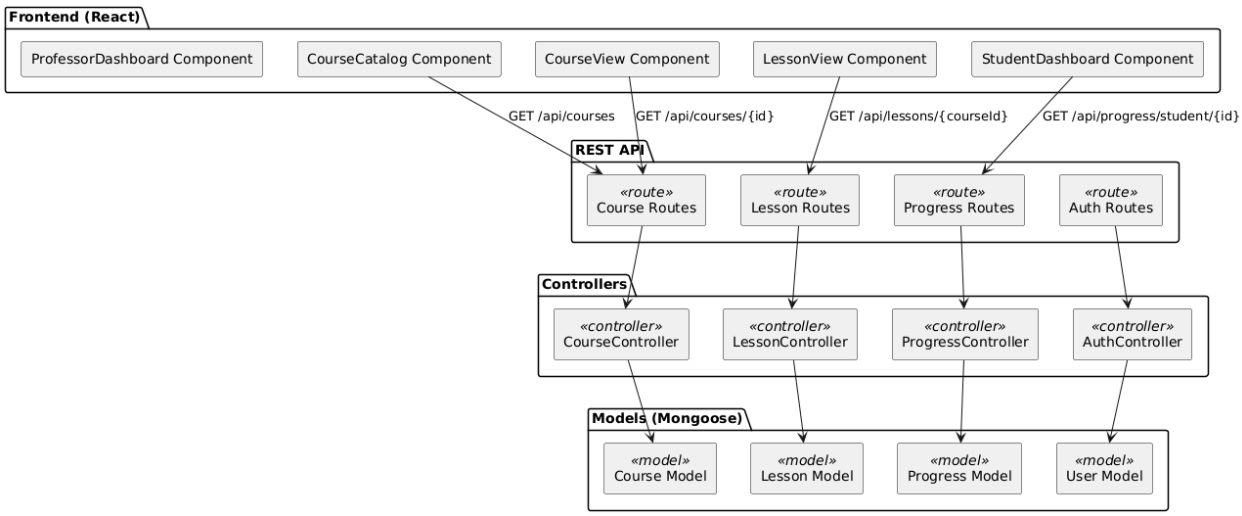
The progress table tracks each student's advancement through individual courses. It uses a composite key of student_id and course_id to uniquely identify records. For each course a student is enrolled in, the system tracks how many lessonsCompleted and quizzesCompleted, providing the foundation for progress bars and completion percentages.

Entity Relationships Summary

- Users (instructors) create multiple courses
- Each course contains many lessons and quizzes
- Lessons may be linked to one or more quizzes
- Quizzes are composed of multiple questions
- Students (users) complete submissions and accumulate progress through courses

8. Interaction Diagrams (Updated)

MVC‑styled REST application with a React frontend the architecture is modeled based on a pub-sub (observer) pattern due to the React components “subscribing” to state (via hooks) and re-render when data (fetched over the API) changes, while the backend “publishes” updates to client apps via REST endpoints.



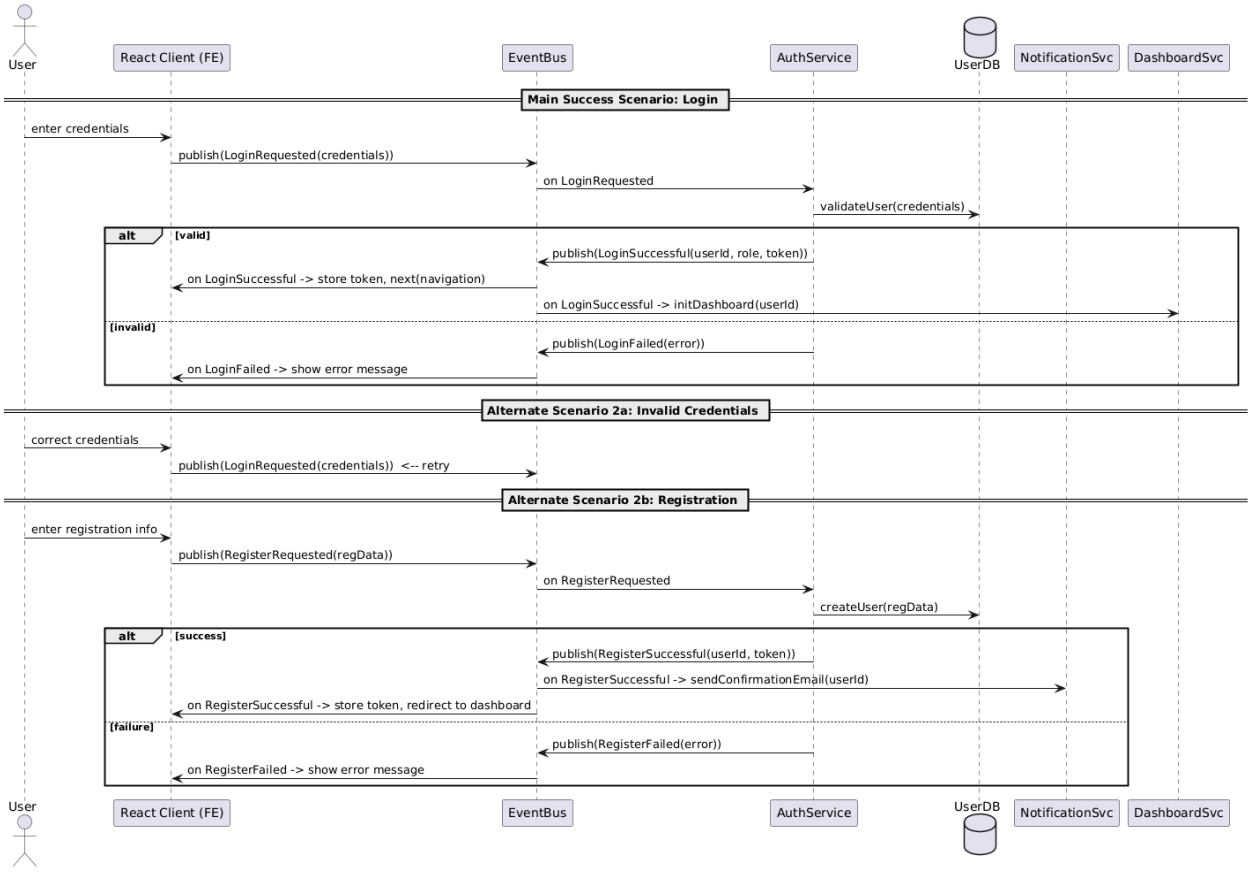
We’ve improved our architecture by applying two complementary design patterns: MVC with REST on the server, and an Observer-style (Publish-Subscribe) event bus on the client. On the server side, we use the Model-View-Controller (MVC) pattern combined with RESTful routes. Models are defined using Mongoose schemas, controllers (like authController and courseController) handle business logic, and routes are managed using express.Router. This clear separation of responsibilities, data, logic, and HTTP plumbing makes the system easier to extend, test, and secure. Compared to our earlier approach where route handlers mixed data access and HTTP concerns, this structure allows for cleaner code, more isolated unit testing, and centralized control of authentication and authorization through middleware.

On the client side, we adopted a lightweight event bus using tools like Node’s EventEmitter or React Context with subscriptions, replacing the previous pattern of passing callbacks and state through multiple layers of props. For example, in the authentication flow, a successful login triggers a loginSuccess event, and components like the navbar or dashboard automatically respond without needing direct updates.

In course management, actions like creating or updating a course emit events such as CourseCreated or CourseUpdated, keeping the catalog, dashboard, and progress panels in sync without tightly coupling them. This event-driven approach eliminates prop drilling, avoids cascading updates when new subscribers are added, and makes individual components easier to test in isolation.

Overall, these patterns significantly improve maintainability, reduce coupling, enhance reusability, and make the system much easier to evolve compared to our original tightly coupled design.

UC-1: Register/Login

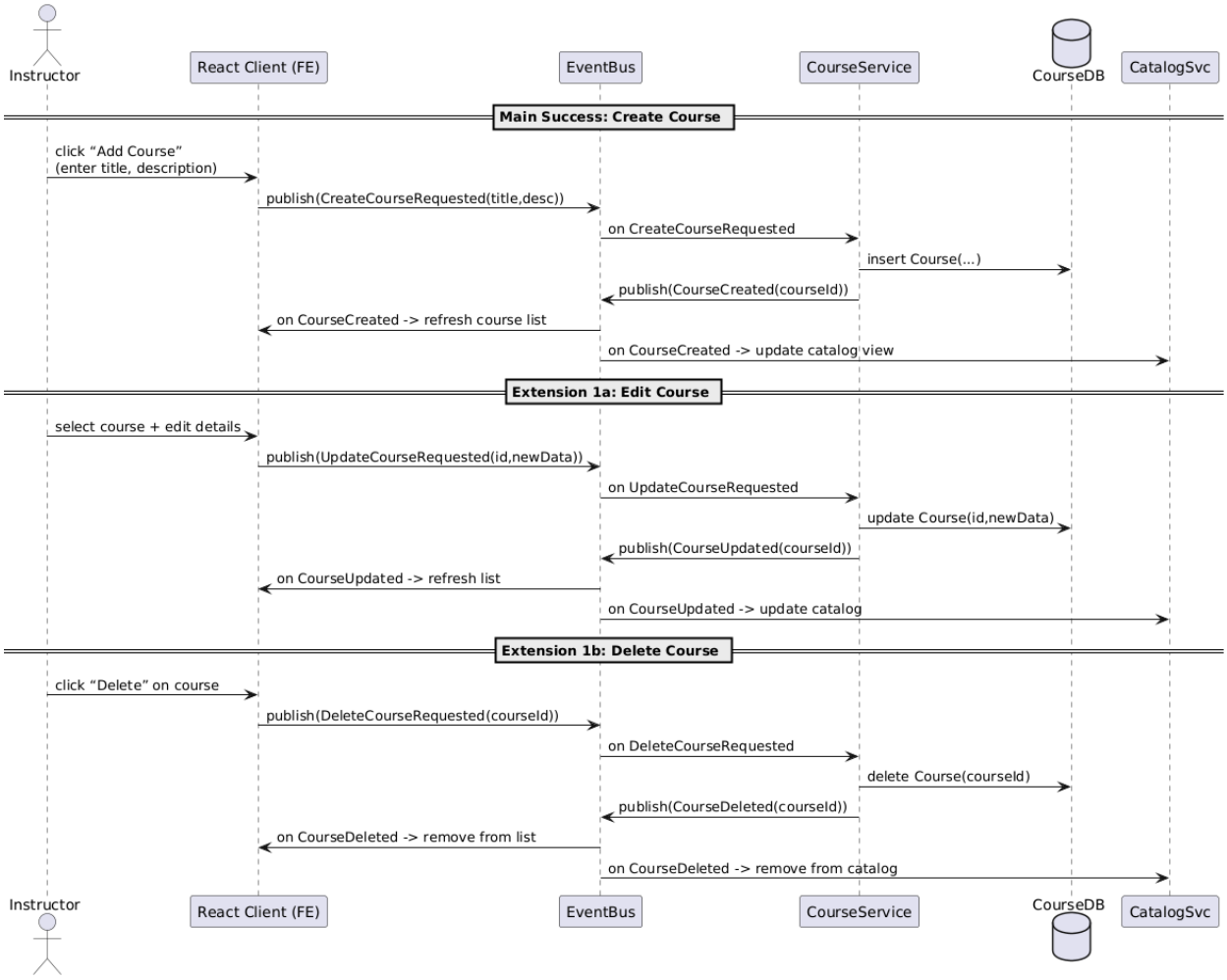


In the Register/Login use case, responsibilities are assigned using the Single Responsibility Principle: authController handles authentication logic, while UserController manages user data operations. This separation supports modularity and independent evolution of each component.

The Information Expert principle places credential validation and user creation within UserController, which directly accesses the User model. To maintain low coupling and high cohesion, authController delegates tasks without manipulating user data directly.

Applying the Controller Pattern, authController acts as a coordinator between user requests and domain logic. Finally, following the Law of Demeter, each controller only interacts with objects it directly depends on ensuring encapsulated, maintainable code.

UC-2: Course Management



In Course Management, the Single Responsibility Principle guides the design by assigning courseController to handle user interactions and Course to manage data and business logic. This separation improves focus and simplifies maintenance.

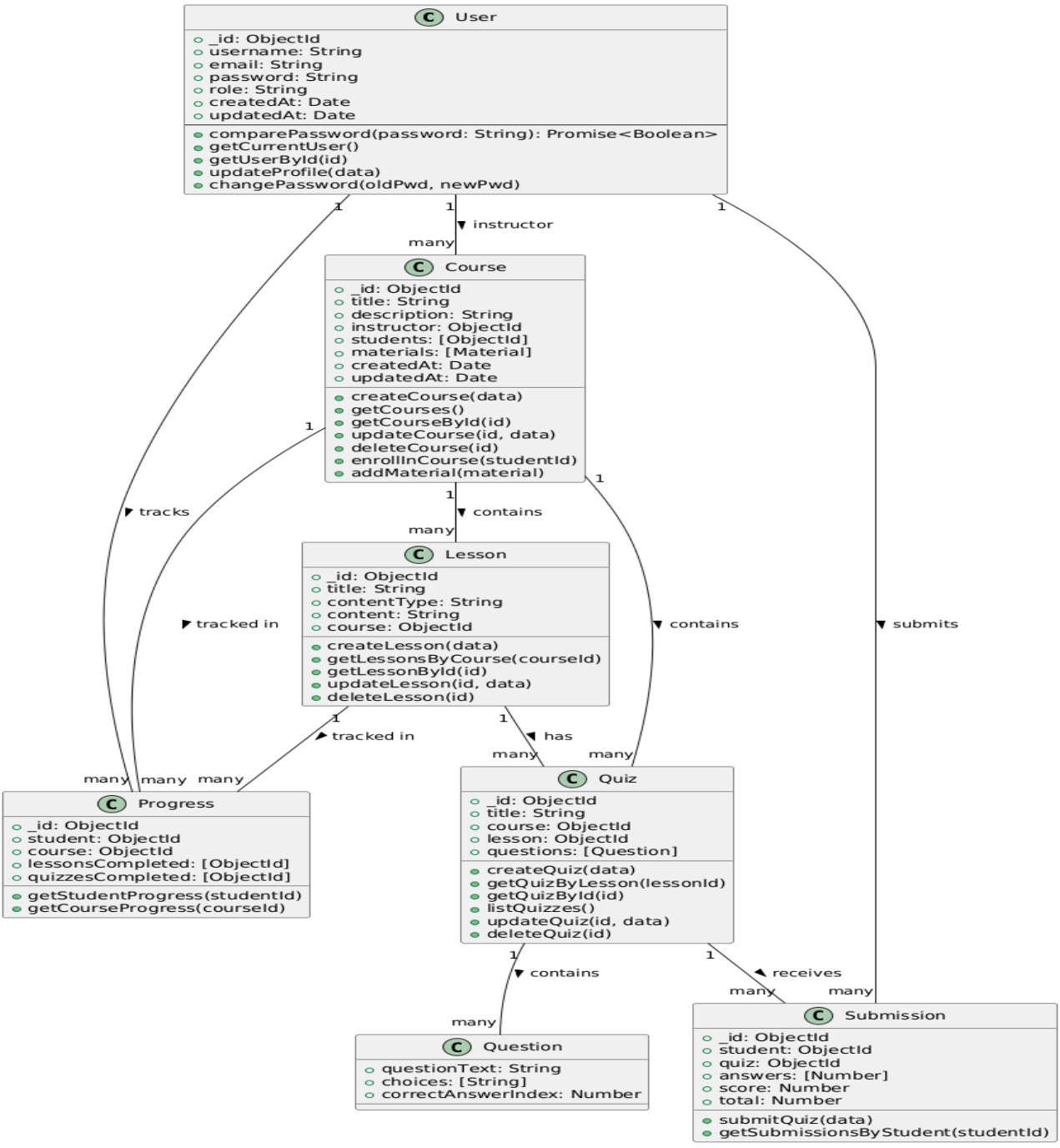
Following Information Expert, the Course model owns operations like create, update, and delete since it contains the necessary data. Low coupling and high cohesion are achieved by having the controller delegate all data tasks to the model.

Using the Controller Pattern, courseController orchestrates workflows without containing domain logic. The Law of Demeter ensures each controller communicates only with its immediate collaborators, reinforcing modularity and minimizing dependencies.

9. Class Diagram and Interface Specification (Updated)

---

a. Class Diagram (Updated)



b. Data Types and Operation Signatures (Updated)

1.User

Class: User
Attributes: + _id: ObjectId (or id: String when returned to frontend) + username: String + email: String + password: String (hashed) + role: String // "student" or "instructor"
Operations: + getCurrentUser(): User + getUserById(id: String): User + updateProfile(data: Object): void + changePassword(oldPwd: String, newPwd: String): void

User Table Definitions:

- Represents a person using BrightBoard.
- id: Unique ID for each user (MongoDB ObjectId).
- username, email: Personal details.
- password: Securely stored (hashed) password.
- role: Defines whether a user is a student or instructor.
- getCurrentUser(): Returns the user who is currently logged in.
- getUserById(id): Finds a user using their ID.
- updateProfile(data): Allows user to change their info.
- changePassword(oldPwd, newPwd): User can change password securely.

2.Course

Class: Course
Attributes: + _id: ObjectId + title: String + description: String + instructor: ObjectId + students: ObjectId + materials: Array
Operations: + createCourse(data: Object): Course + getCourse(): List<Course> + getCourseById(id: String): Course + updateCourse(id: String, data: Object): Course + deleteCourse(id: String): void

Course Table Definitions:

- A digital learning module created by an instructor.
- title, description: Course name and summary.
- createCourse(data): Instructor creates a course.
- getCourse(): Get all available courses.
- getCourseById(id): Fetch specific course.
- updateCourse(id, data): Edit course info.
- deleteCourse(id): Remove a course from the system.
- Represents: A digital learning module created by an instructor.
- _id: Unique ID for the course (MongoDB ObjectId).
- title, description: Course name and summary.
- instructor: The user (instructor) who created the course.
- students: Array of enrolled student IDs.
- materials: Array of course materials (optional).

3. Lesson

Class: Lesson
Attributes: + _id: String (MongoDB ObjectId) + course: String + title: String + contentType: String + content: String (URL or text body)
Operations: + createLesson(data: Object): Lesson + getLessonsByCourse(courseId: String): List<Lesson> + getLessonById(id: String): Lesson + updateLesson(id: String, data: Object): void + deleteLesson(id: String): void

Lesson Table Definitions:

- Represents an individual unit or topic within a course.
- _id: Unique ID for the lesson (MongoDB ObjectId).
- course: Reference to the course it belongs to (ObjectId).
- title: Lesson title.
- contentType: Format of the lesson, such as "text" or "video".
- content: The actual lesson content or a link to it.
- createLesson(data): Create a new lesson.
- getLessonsByCourse(courseId): Fetch all lessons in a course.
- getLessonById(id): Fetch a specific lesson.
- updateLesson(id, data): Edit lesson content.
- deleteLesson(id): Remove a lesson from the system.

4. Quiz

Class: Quiz
Attributes: + _Id: String + title: String + course: String + lesson: String + questions: Array<String>
Operations: + createQuiz(data: Object): Quiz + getQuizByLesson(lessonId: String): Quiz + getQuizById(quizId: String): Quiz + updateQuiz(id: String, data: Object): Quiz + deleteQuiz(id: String): void

Quiz Table Definitions:

- Represents an assessment tied to a lesson.
- _id: Unique ID for the quiz (MongoDB ObjectId).
- title: Quiz title.
- course: Reference to the course (ObjectId).
- lesson: Reference to the lesson it tests (ObjectId).
- questions: List of questions, each with text, choices, and correct answer index.
- createQuiz(data): Add a new quiz to a lesson.
- getQuizByLesson(lessonId): Retrieve quiz tied to a lesson.
- getQuizById(quizId): Retrieve quiz by its ID.
- updateQuiz(id, data): Edit quiz content.
- deleteQuiz(id): Remove a quiz from the system.

5. Question

Class: Question
Attributes: + _Id: String + questionText: String + choices: List<String> + correctAnswerIndex: Integer
Operations: Intentionally Blank

Question Table Definitions:

- Represents a multiple-choice question.
- _id: Unique ID for the question (MongoDB ObjectId).
- questionText: The actual question text.
- choices: Array of possible answer options.
- correctAnswerIndex: Index of the correct option in the choices array.

6. Submission

Class: Submission
Attributes: + _Id: String + student: String + quiz: String + answers: Array<String> + score: Number
Operations: + submitQuiz(data: Object): Submission + getSubmissionsByStudent(studentId: String): List<Submission> + getSubmissionById(id: String): Submission + getSubmissionsByQuiz(quizId: String): List<Submission>

Submission Table Definitions:

- Represents a student's answer to a quiz.
- _id: Unique ID for the submission (MongoDB ObjectId).
- student: Reference to the student who submitted (User ObjectId).
- quiz: Reference to the quiz (Quiz ObjectId).
- answers: Array of student's selected answers.
- score: Grade assigned after submission.
- submitQuiz(data): Save the student's answers.
- getSubmissionsByStudent(studentId): Review past attempts by a student.
- getSubmissionById(id): Retrieve a specific submission.
- getSubmissionsByQuiz(quizId): Review all submissions for a quiz.

7. Progress

Class: Progress
Attributes: + _id: String + student: String + course: String +lessonsCompleted: Array<String> // lessonIds + quizzesCompleted: Array<String> // quizIds
Operations: + getStudentProgress(studentId: String): List<Progress> + getCourseProgress(courseId: String): List<Progress>

Progress Table Definitions:

- Tracks what a student has finished in a course.
- _id: Unique ID for the progress record (MongoDB ObjectId).
- student: Reference to the student (User ObjectId).
- course: Reference to the course (Course ObjectId).
- lessonsCompleted: Array of completed lesson IDs.
- quizzesCompleted: Array of completed quiz IDs.
- getStudentProgress(studentId): Overview of a user's progress in all courses.
- getCourseProgress(courseId): See how all students are progressing in a course.

c. Traceability Matrix (3) - Domain Concept Objects to Class Objects

Some of our domain concepts did not directly translate into derived classes because they represent the logic and behavioral flow of the application rather than concrete, persistent entities. These controller-based concepts (e.g., authController, userController) are responsible for handling interactions between users and the system or coordinating data operations, but they do not represent real-world objects or require data storage. As such, no data classes were derived from them.

Below is a summary of how each domain concept maps to one or more classes and why:

Domain Concept	Class(es) Derived	Explanation
authController		Handles user authentication logic like registration and login using JWT, not tied to a data class.
userController		Manages operations related to retrieving and modifying user information; interacts with the User class.
User	User	Core class representing users of the system (students and instructors), storing credentials and roles.
courseController		Manages creation, listing, updating, and deletion of courses; operates on the Course class
Course	Course	Represents a structured learning module containing lessons and created by instructors.
lessonController		Controls operations for creating and accessing lessons tied to specific courses
Lesson	Lesson	Represents individual learning units containing text or PDF content, associated with a course.
quizController		Facilitates creation and retrieval of quizzes tied to lessons; operates on the Quiz class.
Quiz	Quiz, Question	Quiz models an assessment with multiple questions; Question defines the structure of each quiz item.

submissionController		Handles quiz submissions, scoring, and stores results; updates student progress based on submissions.
Submission	Submission	Captures student quiz responses and calculated scores for performance tracking.
progressController		Tracks how much of a course a student has completed, using lesson and quiz completion data.
Progress	Progress	Records completed lessons and quizzes for each student-course pair, supporting progress visualization.

10. Algorithms and Data Structures

---

a. Data Structures

Brightboard utilizes the following data structures:

Arrays (Lists)

- Storing multiple items such as:
    - Lessons in a Course
    - Questions in a Quiz
    - Choices in a Question
    - Answers in a Submission
    - Completed lessons and quizzes in Progress

Arrays are utilized due to their effectiveness in Ordered data: Lessons and questions often need to be accessed in a specific sequence. Performance: Arrays offer O(1) access by index and efficient iteration. Simplicity: Easy to use and serialize in JSON (which the system uses for API communication).

Hash Tables (Objects or Maps)

- User sessions or tokens: mapping user IDs to session data.
- Fast lookup of:
    - Users by ID
    - Courses by ID
    - Lessons or quizzes by ID

Hash Tables are utilized due to Fast access (average O(1) time complexity).Flexibility: Easy to grow or modify key-value pairs without reordering data. Scalability: Efficient even as data grows.

BrightBoard doesn't rely on more advanced structures like trees or linked lists, but it strategically uses arrays and hash tables to maintain performance, simplicity, and data organization particularly for features like course content delivery, quiz systems, and user tracking.

b. Concurrency:

BrightBoard does not use multiple threads explicitly.

BrightBoard is built using Node.js, which is single-threaded by design (based on the event-driven, non-blocking I/O model). Instead of spawning multiple threads, Node.js uses an event loop and asynchronous programming to handle concurrent operations like:

- API request handling
- Database queries
- File uploads
- Authentication flows
- Quiz submissions and scoring

In place of multithreading brightboard utilizes:

- **Asynchronous Callbacks / Promises / async-await:**
  These allow operations (e.g., reading a file or fetching from MongoDB) to run without blocking the main thread.

- **Express.js Route Handlers:**
  Each incoming request is handled independently and non-blocking. While it seems like parallel processing, it's actually asynchronous execution within a single thread.

There is no need for thread synchronization mechanisms (like mutexes, semaphores, or locks), because Node.js ensures that only one operation runs at a time in the main thread, while background tasks complete asynchronously.

If in the future BrightBoard needs to handle CPU-intensive tasks (e.g., real-time analytics or media processing), it could benefit from Worker Threads (Node.js module), Child Processes, Message queues (e.g., RabbitMQ, Redis) but those are not currently used.

11. User Interface Design and Implementation

No significant changes have been made to the initial screen mock-ups developed for Report #1, nor are any major revisions currently planned. Our original interface design emphasized clarity, simplicity, and minimal user effort. We have continued to follow those principles throughout development.

The existing layout supports intuitive navigation, with clear labeling and minimal required input per screen. By maintaining a clean and structured interface, we aim to reduce user confusion and eliminate unnecessary steps in common workflows (such as logging in, accessing content, and tracking progress). Minor visual updates may occur during implementation, but these will not affect usability or introduce additional complexity.

Overall, our design remains focused on maximizing ease-of-use by minimizing user actions and promoting intuitive interaction, in line with best practices for effective GUI design.

12. Test Designs

Note that for this report you are just *designing* your tests; you will *program and run* those tests as part of work for your first demo, see the list here.

a. Test Cases

Test Case List and Descriptions

Test Case Identifier: TC-1 **Input Data:** Email, Password **Use Case Tested:** UC-2: Log in / Log out **Pass/Fail Criteria:** ● **Pass:** If the user is able to successfully log in with valid credentials. ● **Fail:** If the user enters an invalid email or password.	
Test Procedure:	**Step 1:** User submits quiz with missing or malformed answers. **Step 2:** User submits quiz with all required answers correctly formatted.
Expected Result:	● Server rejects the submission and notifies the user to complete the quiz. ● Server accepts submission, evaluates the score, and returns the result.

Test Case Identifier: TC-2

Use Case Tested: UC-6: Take Quiz

Pass/Fail Criteria:

- **Pass:** If the user is able to submit a completed quiz and receive a score.
- **Fail:** If submission is rejected or scoring fails.

Input Data: Quiz ID, Question Answers

Test Procedure:	**Step 1:** User enters an invalid email or password. **Step 2:** User enters a valid email and password.
Expected Result:	• Server denies the login attempt with a message indicating incorrect credentials. • Server allows login, and depending on the user role (student or instructor), redirects to the correct dashboard.

Test Case Identifier: TC-3

Use Case Tested: UC-4: Create Course (Instructor)
Pass/Fail Criteria:

- **Pass:** If a course is created with a unique name and valid data.
- **Fail:** If required fields are missing or course already exists.

Input Data: Course title, description, instructor ID

Test Procedure:	**Step 1**: Instructor attempts to create a course with missing title. **Step 2**: Instructor submits course with valid data.
Expected Result:	● Server denies creation and displays error message. ● Server creates and stores the course.

Test Case Identifier: TC-4

Use Case Tested: UC-7: Track Progress

Pass/Fail Criteria:

- **Pass:** If the progress data is correctly calculated and shown.
- **Fail:** If progress is not updated after completing activities.

Input Data: User ID, lesson/quiz completion data

Test Procedure:	**Step 1:** User completes a quiz. **Step 2:** User views progress tracker.
Expected Result:	● Server updates progress and reflects changes in dashboard. ● If error, no update is shown and user is informed. ● Server creates and stores the course.

Test Case Identifier: TC-5 **Use Case Tested:** UC-3: Lesson View and Completion **Pass/Fail Criteria:** ● **Pass:** If user can access lesson content and mark it as complete. ● **Fail:** If content fails to load or completion is not recorded. **Input Data:** Lesson ID, user token	
Test Procedure:	**Step 1:** User opens a lesson that doesn't exist. **Step 2:** User completes a valid lesson and marks it complete.
Expected Result:	● Error message shown for missing lesson. ● Lesson marked complete, status updated. ● If error, no update is shown and user is informed. ● Server creates and stores the course.

b. Test Coverage

Our test cases currently cover core system functionality, including authentication and access control (TC-1), key user actions such as taking quizzes, creating courses, and tracking learning progress (TC-2 through TC-5), as well as validation on both the client and server sides. We ensure differentiated support for user roles(students and instructors)with each role tested for proper access and behavior.

We follow a bottom-up testing strategy, starting with individual backend components like controller logic (e.g., register, login, course creation, submission scoring), input validation, authentication/authorization middleware, and data models. As development progresses, test coverage will expand to include more advanced features such as certification generation, reminders, and analytics dashboards.

We aim to achieve at least 85% unit test coverage across all backend modules. Specific metrics tracked include:

- Line coverage
- Branch coverage
- Function coverage
- Statement coverage

Any code areas falling below 80% coverage will be flagged and revisited to maintain overall quality and reliability.

c. Integration Testing

We use the bottom-up strategy First, test low-level modules: API endpoints, models, and individual features (quizzes, lessons). Then, test how components work together: submitting quizzes updates progress, logging in redirects based on roles, instructors see aggregated data from students.

This layered approach helps isolate bugs quickly and validate each component in the real-world context of user behavior.

Approach:

- Use Supertest to simulate HTTP requests across controller endpoints
- Use a MongoDB test instance with seeded mock data
- Focus areas:
    - User registration + course creation
    - Lesson creation + quiz linkage
    - Quiz submission + progress update
    - Authenticated routes and access control logic

Planned Integration Tests:

- POST /register → GET /me → POST /courses (registration-to-course flow)
- POST /lessons → POST /quizzes → POST /submitQuiz (learning-to-assessment flow)
- POST /submitQuiz → GET /progress (assessment-to-progress tracking)

d. System Testing

In system testing, we will validate the system as a complete, integrated solution to ensure that all components function correctly and meet both the functional and non-functional requirements of the platform.

We will begin by validating the core algorithms that drive user interaction and content progression. This includes verifying that quiz and assignment submissions are scored accurately according to predefined rubrics or answer keys. We will also test for edge cases, such as partial submissions, retry limits, and randomized question order. Lesson completion checks will be evaluated to ensure that a lesson is marked complete only when all required actions—such as viewing embedded media or answering questions—are fulfilled. Additionally, we will test the adaptive content rendering functionality to confirm that the platform adjusts learning materials appropriately based on user performance and progress, such as unlocking the next module only when a minimum score threshold has been achieved.

We will also assess the system's non-functional requirements to confirm its robustness in real-world usage. Performance testing will include evaluating how the system responds under high load conditions, such as multiple simultaneous logins or bulk quiz submissions. We will measure latency during key actions like page transitions, content loading, and dashboard rendering to ensure a smooth user experience.

Reliability testing will involve running long-duration tests to detect potential memory leaks, improper session handling, or data persistence failures. We will simulate failover scenarios, including network interruptions and database unavailability, and verify the system's ability to recover gracefully without data loss. Security will be a major focus as well. We will confirm that all authentication and authorization mechanisms are properly enforced across the platform, preventing unauthorized access. Common web vulnerabilities such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF) will be tested against, and we will review the secure handling of sensitive data, including the use of HTTPS and proper password hashing.

The user interface will undergo thorough testing to verify usability and accessibility across all roles. We will validate that dashboards render correctly for students, instructors, and administrators, with each role having access only to its relevant features and functionality. Role-specific actions such as course creation or grade submission will be tested to confirm that they are visible and usable only by authorized users.

We will also review the effectiveness of error messages, ensuring they are clear, informative, and helpful to the user. This includes testing how the system responds to invalid input, failed submissions, or expired sessions. All user-interactive features such as links, buttons, menus, and form controls will be tested to confirm their responsiveness, functionality, and visual feedback under normal and edge-case interactions. We will further ensure the interface is fully responsive

and accessible on various device types, including desktops, tablets, and mobile phones. Lastly, we will evaluate the intuitiveness of navigation flows, verifying that users can move through the system logically and efficiently, whether returning to the dashboard, editing profiles, or accessing learning progress.

13. Project Management (Updated)

---

a. History of Work

Task	Developer	Est. Begin	Est.Complete	Actual End
Set up GitHub repo & task board	Both	6/7/2025	6/8/2025	6/10/2025
Define tech stack	Both	6/7/2025	6/8/2025	6/11/2025
Design DB schema (users, courses, etc.)	Conner	6/7/2025	6/10/2025	6/10/2025
Set up backend project (Express.js)	Conner	6/7/2025	6/10/2025	6/11/2025
Wireframe UI: login, dashboard, courses	Shadow	6/7/2025	6/10/2025	6/15/2025
Setup frontend scaffolding (React, CSS)	Shadow	6/9/2025	6/11/2025	6/25/2025
Implement auth (login, register)	Conner	6/12/2025	6/15/2025	6/28/2025
Role-based access (Instructor/Student)	Conner	6/12/2025	6/15/2025	6/28/2025
Build login/register UI	Shadow	6/12/2025	6/14/2025	6/18/2025
Responsive styling + validation	Shadow	6/13/2025	6/15/2025	7/01/2025
Create course + lesson API	Conner	6/16/2025	6/19/2025	7/01/2025
Instructor dashboard UI	Shadow	6/16/2025	6/19/2025	6/28/2025
Lesson upload + preview	Shadow	6/18/2025	6/20/2025	6/30/2025
Student course listing + enroll API	Conner	6/23/2025	6/25/2025	6/20/2025
Serve lessons securely (student view)	Conner	6/24/2025	6/26/2025	6/30/2025
Student dashboard UI (browse/enroll/view)	Shadow	6/23/2025	6/26/2025	7/04/2025
Add mobile responsiveness	Shadow	6/25/2025	6/27/2025	7/12/2025
Quiz creation API (MCQs)	Conner	6/30/2025	7/2/2025	7/1/2025
Quiz-taking + feedback API	Conner	7/2/2025	7/4/2025	7/2/2025
Quiz builder UI (MCQs)	Shadow	6/30/2025	7/2/2025	7/2/2025

Quiz interface + feedback design	Shadow	7/2/2025	7/4/2025	7/3/2025
Progress tracking API	Conner	7/7/2025	7/9/2025	7/11/2025
Visual progress UI (bars, badges)	Shadow	7/7/2025	7/9/2025	7/12/2025
UI polish + accessibility	Shadow	7/9/2025	7/11/2025	7/13/2025
User docs + test cases	Shadow	7/9/2025	7/11/2025	7/04/2025
Final testing + bug fixes	Both	7/10/2025	7/11/2025	7/14/2025
Deploy app to server	Both	7/11/2025	7/12/2025	7/13/2025
Prepare final demo & presentation	Both	7/14/2025	7/18/2025	7/14/2025

b. Current Status

As of Report 3, the BrightBoard system successfully delivers key features for both Professor and Student Dashboards, addressing the core use cases outlined in our project plan. Professors can create and manage courses, lessons, and quizzes, while students can view lessons and take quizzes tailored to their enrolled courses. The system is built with a Node.js/Express backend and MongoDB for data storage, using JWT for secure authentication, and features a responsive React frontend with role-based access control.

BrightBoard has proven useful in enabling structured course delivery and interactive assessments. It meets critical use cases such as content management, student engagement through quizzes, and secure access by role. Its clean interface and modular backend design ensure the system is easy to understand, maintain, and extend.

For Demo 2, we aim to introduce progress tracking so students can view their learning progress and quiz performance, and professors can monitor individual student activity across courses. Future enhancements will also include quiz analytics, embedded media support, and lesson preview tools to further improve usability and educational value.

c. Key Accomplishments

- Implemented secure user authentication using JWT, with role-based access for students and professors.
- Developed Professor Dashboard allowing course, lesson, and quiz creation and management.
- Built Student Dashboard for viewing enrolled courses, accessing lessons, and completing quizzes.
- Integrated RESTful backend with Node.js/Express and MongoDB, supporting scalable course and quiz data operations.

- Created dynamic React frontend with clean UI and intuitive navigation for both user roles.
- Enabled quiz creation/editing workflow, including automatic question formatting and grading logic.
- Supported lesson content with PDF uploads for enhanced learning materials.
- Established clean backend design, ensuring separation of concerns and ease of future expansion.
- Prepared for progress tracking features, setting up database and frontend structures to support student learning analytics.
- Successfully tested core use cases, validating the system's reliability, usability, and correctness.

d. Future Work

While BrightBoard currently provides a robust platform for online course management, there are several areas where future enhancements could significantly improve functionality,accessibility, and user experience:

- Real-Time Features
  Integrate real-time notifications (e.g., for assignment deadlines, new materials, or messages).

- Enhanced Analytics and Reporting
  Implement advanced analytics dashboards for both students and instructors, including detailed progress tracking, quiz performance trends, and course engagement metrics.

- Accessibility Improvements
  Ensure the platform meets WCAG accessibility standards, making it usable for all students, including those with disabilities.

- Offline Support
  Enable offline access to course materials and quizzes, syncing progress when the user reconnects.

- User Feedback and Support System:
  Implement in-app feedback, helpdesk, or chatbot support for users to report issues or get assistance.

- Improve PDF file support:

Render the disk where PDF uploads is ephemeral. Causing the local folder to be wiped if/when the services spins down or if the container restarts. As a result, any PDFs saved vanish when the instance comes back up. To enable future application use and improvements uploads such as PDFs will be placed on truly persistent storage (S3, GridFS, a Render volume, etc.)

These enhancements would further establish BrightBoard as a comprehensive, modern, and scalable learning management system.

14. References

Almrashdeh, I. A., et al. "Distance Learning Management System Requirements from Student's Perspective." *Journal of Theoretical and Applied Information Technology*, vol. 24, no. 1, 2011, pp. 17–27.

Frand, Jason L. "The Information Age Mindset: Changes in Students and Implications for Higher Education." *Educause Review*, vol. 35, no. 5, 2000, pp. 14–24.

Rhode, Jason, et al. "Understanding Faculty Use of the Learning Management System." *Online Learning*, vol. 21, no. 3, 2017, pp. 68–86. https://doi.org/10.24059/olj.v21i3.1217.

Turnbull, Deborah, Rajeev Chugh, and Jo Luck. "Learning Management Systems, An Overview." *Encyclopedia of Education and Information Technologies*, edited by Arthur Tatnall, Springer, 2020. https://doi.org/10.1007/978-3-030-10576-1_248.