BrightBoard –
A Course and Quiz System
(Technical Documentation)

CSCI 441 VA: Software Engineering

Jul 7, 2025

Team Members: Shadow Love-Erckert, Conner Erckert

Project URL: https://csci441-brightboard.onrender.com/

GitHub URL:https://github.com/Element713/CSCI441_BrightBoard.git

Table of Contents

1. Use Cases

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

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

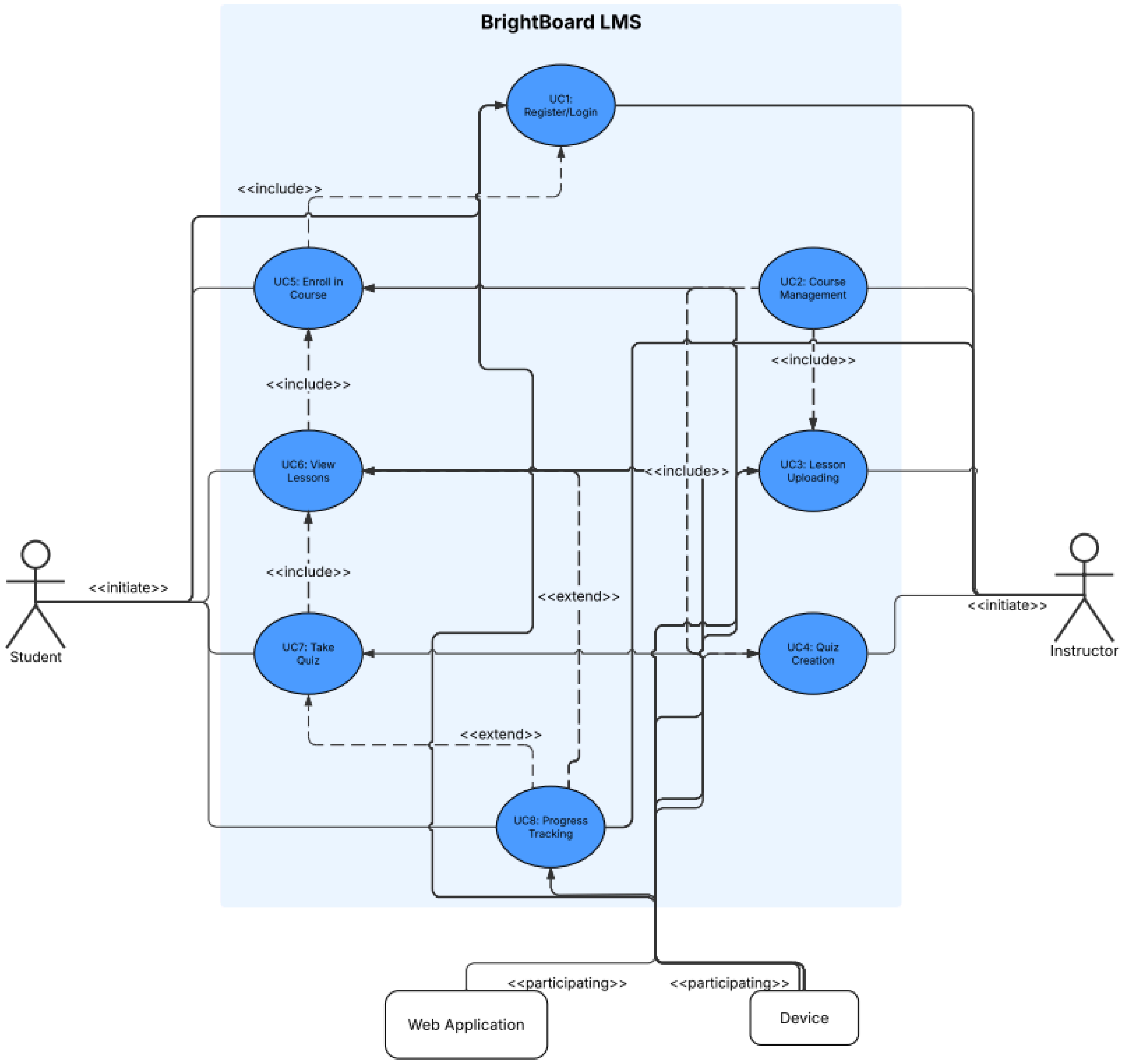
UC6: View Lessons
- Description: Students access course content.
- Responds to Requirements: REQ-7, UI-2, UI-4

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

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:

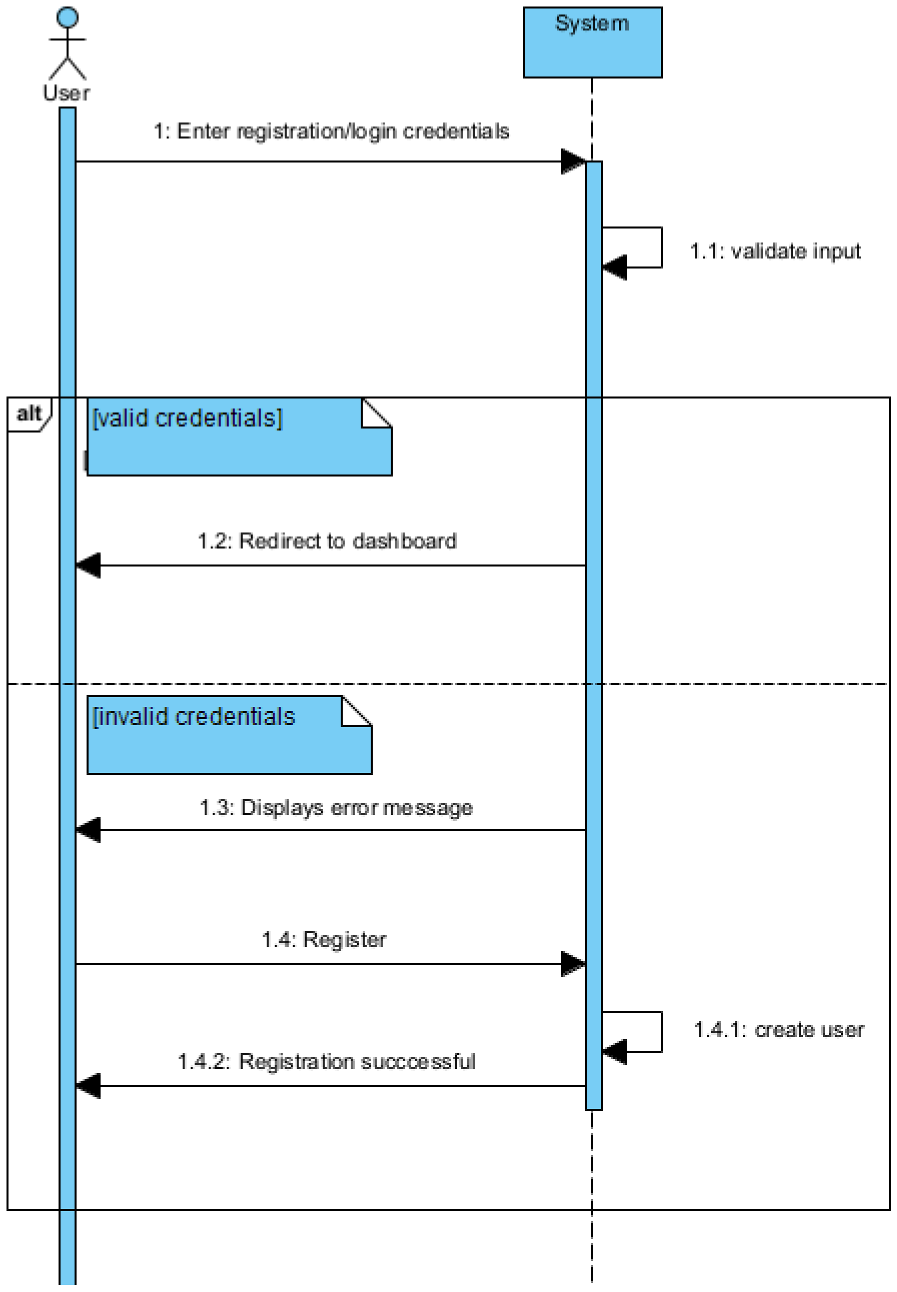
1. Instructor creates a new course by entering required details.
2. System saves and displays the new course.

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

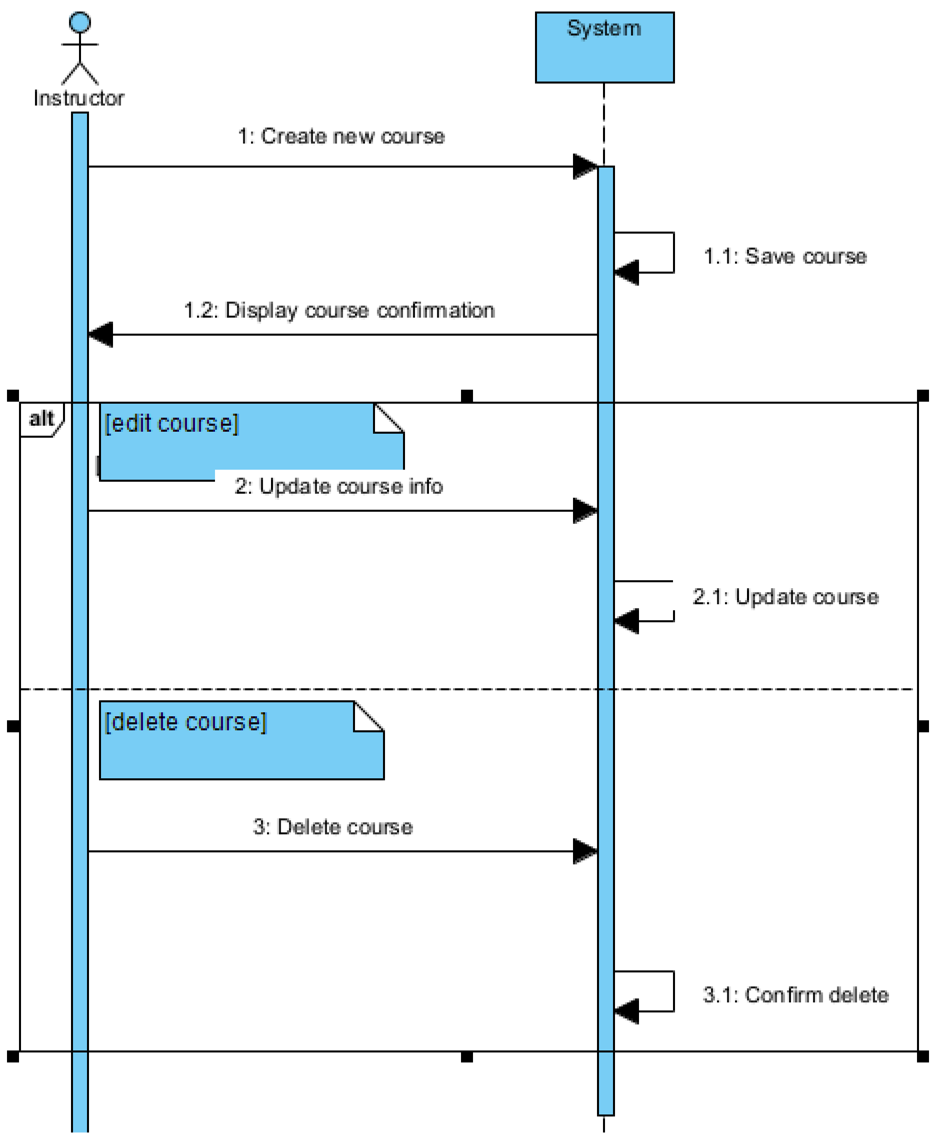
- System loads existing course for update.
- Changes are saved and reflected in the course list.
  1b. Instructor deletes a course
- System confirms deletion and removes course from the list.

d. System Sequence Diagrams

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


User
System
1: Enter registration/login credentials
1.1: validate input
alt
[valid credentials]
1.2: Redirect to dashboard
[invalid credentials
1.3: Displays error message
1.4: Register
1.4.1: create user
1.4.2: Registration succcessful

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



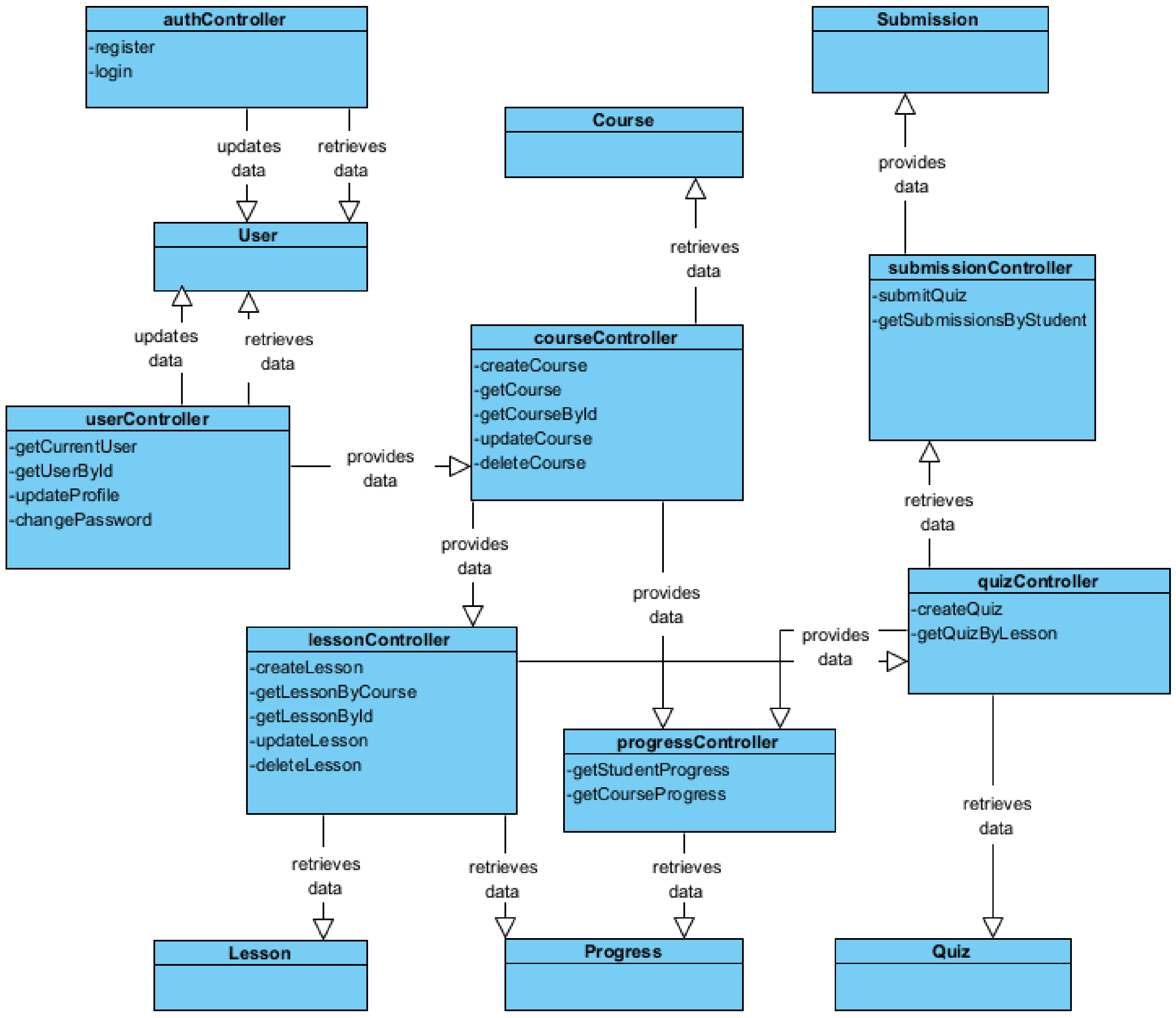
2. Analysis and Domain Modeling

---

a. Conceptual Model

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



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

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).
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.
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, email, or role.
	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

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

b. System Operation Contracts

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.

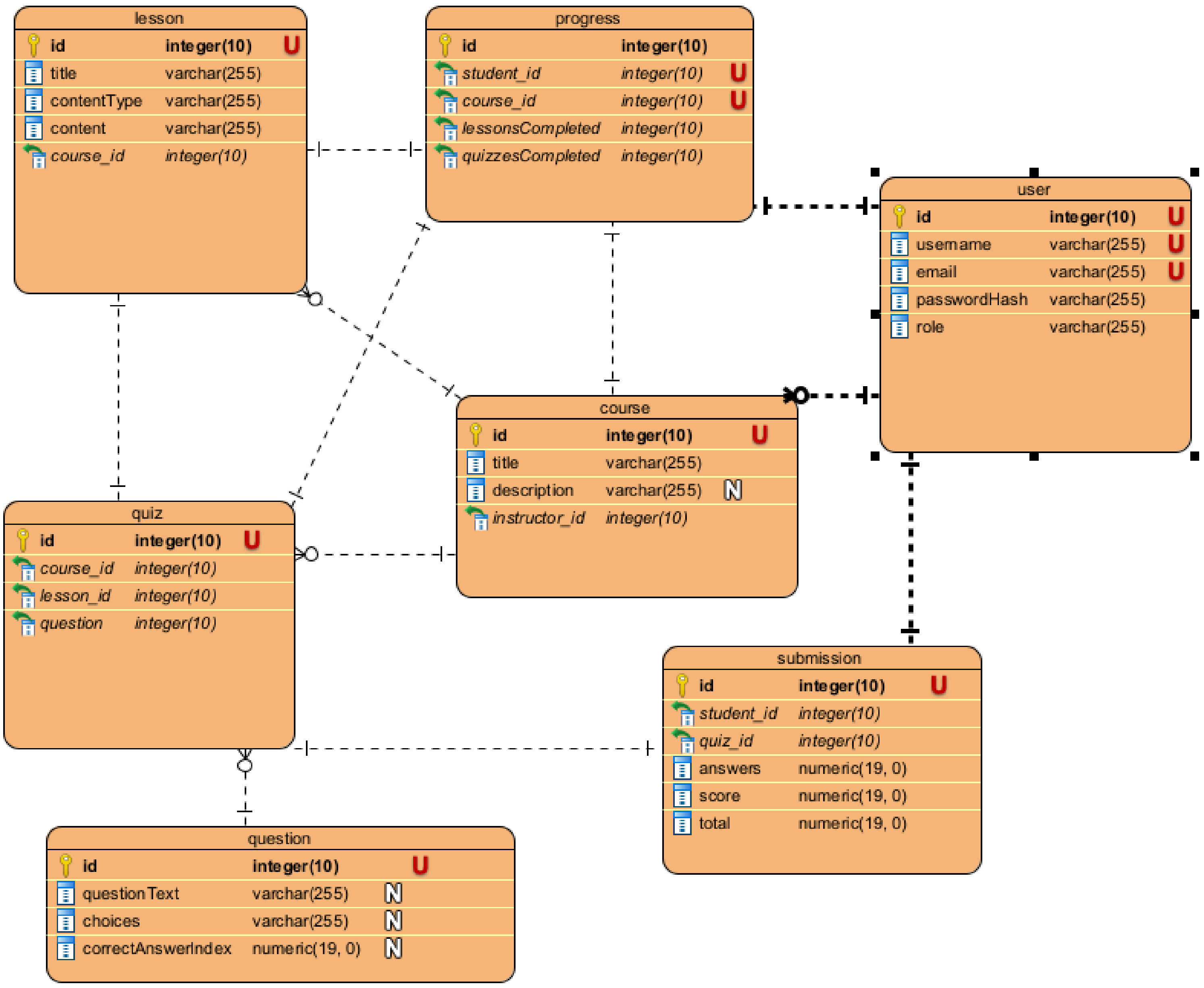
c. Data Model and Persistent Data Storage

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.

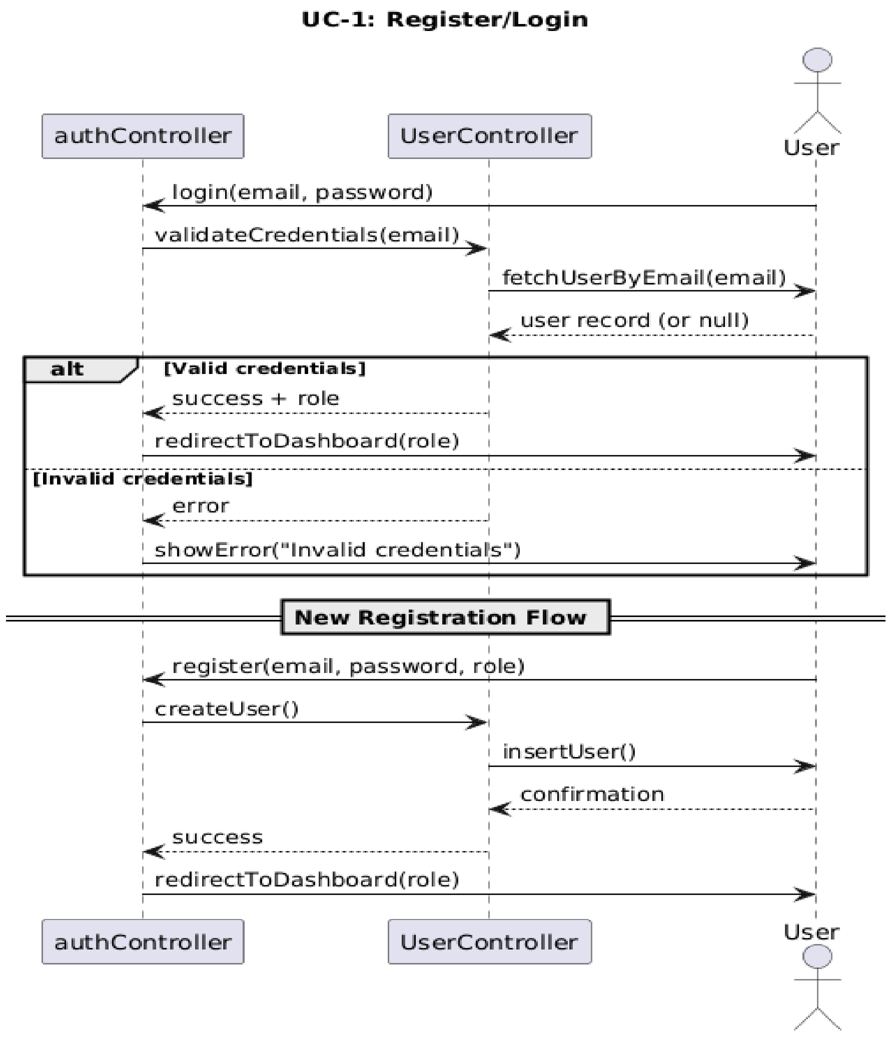
Progress Table

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

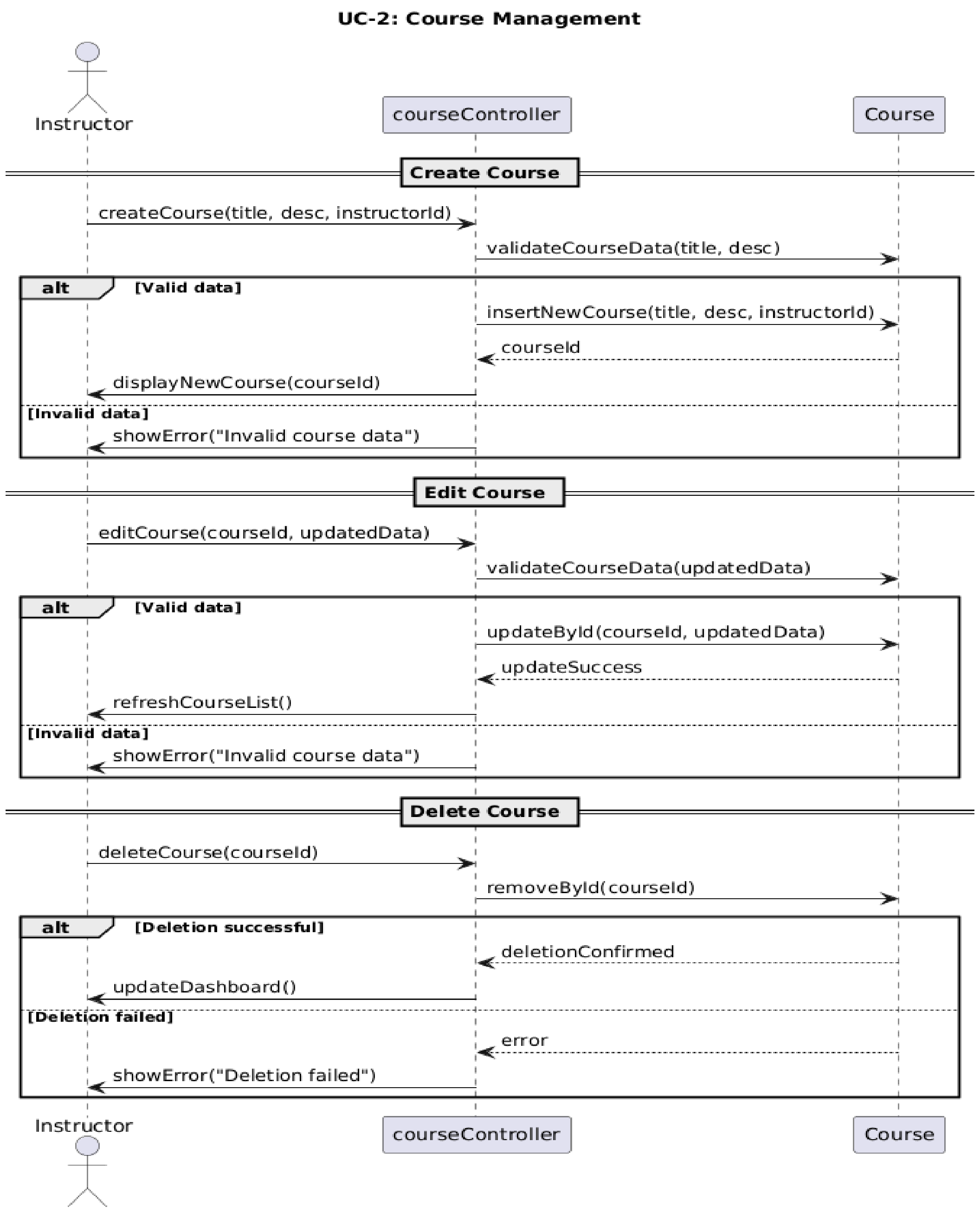
3. Interaction Diagrams



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
Instructor
courseController
Course
Create Course
createCourse(title, desc, instructorId)
validateCourseData(title, desc)
alt [Valid data]
insertNewCourse(title, desc, instructorId)
courseId
displayNewCourse(courseId)
[Invalid data]
showError("Invalid course data")
Edit Course
editCourse(courseId, updatedData)
validateCourseData(updatedData)
alt [Valid data]
updateById(courseId, updatedData)
updateSuccess
refreshCourseList()
[Invalid data]
showError("Invalid course data")
Delete Course
deleteCourse(courseId)
removeById(courseId)
alt [Deletion successful]
deletionConfirmed
updateDashboard()
[Deletion failed]
error
showError("Deletion failed")
Instructor
courseController
Course

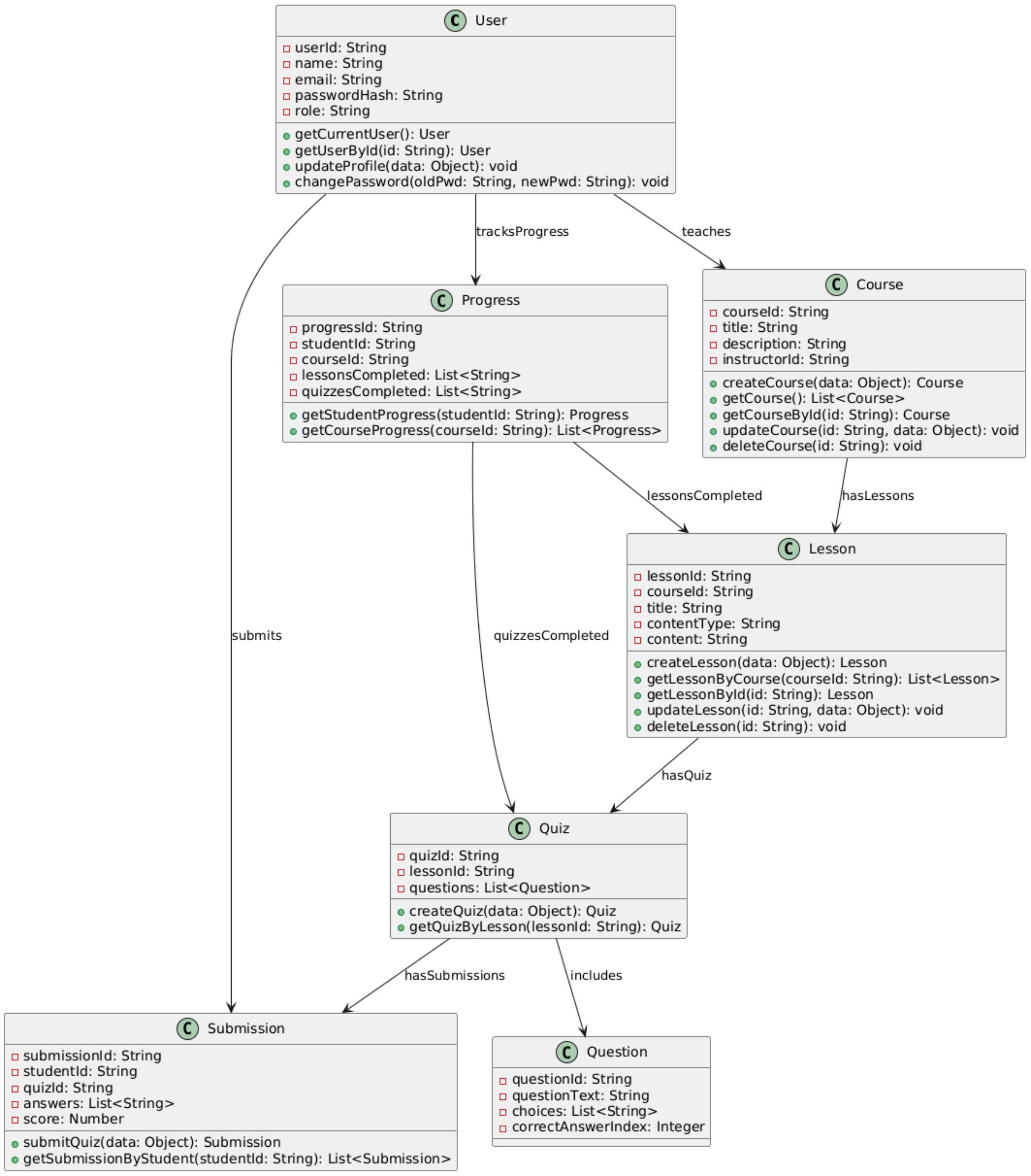
In Course Management, the Single Responsibility Principle guides the design by assigning courseController to handle user interactions and Course to manage data and business logic. This separation improves focus and simplifies maintenance.

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.

4. Class Diagram and Interface Specification

a. Class Diagram


User
userId: String
name: String
email: String
passwordHash: String
role: String
getCurrentUser(): User
getUserById(id: String): User
updateProfile(data: Object): void
changePassword(oldPwd: String, newPwd: String): void
tracksProgress
teaches
Progress
progressId: String
studentId: String
courseId: String
lessonsCompleted: List<String>
quizzesCompleted: List<String>
getStudentProgress(studentId: String): Progress
getCourseProgress(courseId: String): List<Progress>
Course
courseId: String
title: String
description: String
instructorId: String
createCourse(data: Object): Course
getCourse(): List<Course>
getCourseById(id: String): Course
updateCourse(id: String, data: Object): void
deleteCourse(id: String): void
lessonsCompleted
hasLessons
Lesson
lessonId: String
courseId: String
title: String
contentType: String
content: String
createLesson(data: Object): Lesson
getLessonByCourse(courseId: String): List<Lesson>
getLessonById(id: String): Lesson
updateLesson(id: String, data: Object): void
deleteLesson(id: String): void
submits
quizzesCompleted
hasQuiz
Quiz
quizId: String
lessonId: String
questions: List<Question>
createQuiz(data: Object): Quiz
getQuizByLesson(lessonId: String): Quiz
hasSubmissions
includes
Submission
submissionId: String
studentId: String
quizId: String
answers: List<String>
score: Number
submitQuiz(data: Object): Submission
getSubmissionByStudent(studentId: String): List<Submission>
Question
questionId: String
questionText: String
choices: List<String>
correctAnswerIndex: Integer

b. Data Types and Operation Signatures

1.User

Class: User
Attributes: + userId: String + name: String + email: String + passwordHash: String + role: String // e.g., "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.
- userId: Unique ID for each user.
- name, email: Personal details.
- passwordHash: Securely stored 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

Course
Attributes: + courseId: String + title: String + description: String + instructorId: String
Operations: + createCourse(data: Object): Course + getCourse(): List<Course> + getCourseById(id: String): Course + updateCourse(id: String, data: Object): void + deleteCourse(id: String): void

Course Table Definitions:

- A digital learning module created by an instructor.
- courseId: Unique ID for the course.
- title, description: Course name and summary.
- instructorId: The user who created the course.
- 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.

3. Lesson

Class Lesson
Attributes:
+ lessonId: String
+ courseId: String
+ title: String
+ contentType: String   // e.g., "text", "video"
+ content: String   // URL or text body
Operations:
+ createLesson(data: Object): Lesson
+ getLessonByCourse(courseId: String): List<Lesson>
+ getLessonById(id: String): Lesson
+ updateLesson(id: String, data: Object): void
+ deleteLesson(id: String): void

Lesson Table Definitions:

- An individual unit or topic within a course.
- lessonId: Unique ID for the lesson.
- courseId: Which course it belongs to.
- title: Lesson title.
- contentType: Format, like text or video.
- content: Actual lesson content or link.
- createLesson(), updateLesson(): For managing lesson content.
- getLessonByCourse(): Fetch all lessons in a course.
- deleteLesson(): Remove a lesson.

4. Quiz

Class Quiz
Attributes: + quizId: String + lessonId: String + questions: List<Question>
Operations: + createQuiz(data: Object): Quiz + getQuizByLesson(lessonId: String): Quiz

Quiz Table Definitions:

- An assessment tied to a lesson.
- quizId: Unique ID.
- lessonId: Which lesson it tests.
- questions: List of questions included.
- createQuiz(): Add a new quiz to a lesson.
- getQuizByLesson(): Retrieve quiz tied to a lesson.

5. Question

Class Question
Attributes: + questionId: String + questionText: String + choices: List<String> + correctAnswerIndex: Integer
Operations:

Question Table Definitions:

- A multiple-choice question.
- questionId: Unique ID.
- questionText: The actual question.
- choices: The answer options.
- correctAnswerIndex: Which option is correct (e.g., 0 = first choice).

6. Submission

Class Submission
Attributes: + submissionId: String + studentId: String + quizId: String + answers: List<String> + score: Number
Operations: + submitQuiz(data: Object): Submission + getSubmissionByStudent(studentId: String): List<Submission>

Submission Table Definitions:

- A student's answer to a quiz.
- submissionId: Unique ID.
- studentId: Who submitted.
- quizId: Which quiz it relates to.
- answers: Student's selected answers.
- score: Grade assigned after submission.
- submitQuiz(): Save the student's answers.
- getSubmissionByStudent(): Review past attempts.

7. Progress

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

Progress Table Definitions:

- Tracks what a student has finished.
- progressId: Unique ID.
- studentId, courseId: Who and what course.
- lessonsCompleted: List of lessons finished.
- quizzesCompleted: List of completed quizzes.
- getStudentProgress(): Overview for a user.
- getCourseProgress(): See how all students are doing.

5. 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.