

Learning Management System

DESIGN DOCUMENT

Group 46

Client:

Prof. Islam and Prof. Tyagi

Advisor:

Prof. Islam

Members:

Sam DeFrancisco

Nicholas Erickson

Jennifer Robles

Deepika Vempati

Nikhil Kuricheti

Brayton Rude

Contact:

sdmay24-46@iastate.edu

<https://sdmay24-46.sd.ece.iastate.edu/>

Revised on 11/26/2023

Executive Summary

Development Standards & Practices Used

- Agile Project Management
- Client - Server model
- Cloud Development
- CI/CD

Summary of Requirements

- Build learning platform that allows users to search and enroll in courses
- Courses should have various video lessons and track a users progress into a course
- Courses can have quizzes and will save a users score on a quiz
- Videos may be linked to YouTube or directly uploaded as a mp4
- Course marketplace allows users to search courses
- Course marketplace gives course recommendation based on previous taken courses
- Course creator studio where users can create their own courses

Applicable Courses from Iowa State University Curriculum

- COMS 227
- COMS 228
- COMS 309
- COMS 319
- COMS 339
- COMS 363
- SE 329

New Skills/Knowledge acquired that was not taught in courses

- Cloud Development
- CI/CD
- Material UI
- Creating and deploying live applications from complete scratch
- Image / video processing and storage

Table of Contents

1 Team, Problem Statement, Requirements, and Engineering Standards	10
1.1 INITIAL	10
1.2 REQUIRED SKILL SETS FOR YOUR PROJECT	10
1.3 SKILL SETS COVERED BY THE TEAM	10
1.4 Project Management Style Adopted by the team	10
1.5 Initial Project Management Roles	11
1.6 Problem Statement	11
1.7 Requirements & Constraints	11
1.8 Engineering Standards	13
1.9 Intended Users and Uses	13
2. Project Plan	14
2.1 Task Decomposition	14
2.2 Project Management/Tracking Procedures	18
2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria	18
2.4 Project Timeline/Schedule	21
2.5 Risks And Risk Management/Mitigation	22
2.6 Personnel Effort Requirements	26
2.7 Other Resource Requirements	26
3 Design	27
3.1 Design Content	27
3.2 Design Complexity	27
3.3 Modern Engineering Tools	28
3.4 Design Context	28
3.5 Prior Work/Solutions	29
3.6 Design Decisions	30
3.7 Proposed Design	31
3.7.1 Design 0 (Initial Design)	31
Application Design	31
Client Design	34
3.7.2 Design 1 (Design Iteration)	36
Application Design	36
Client Design	37
3.8 Technology Considerations	38
3.9 Design Analysis	38
4 Testing	39
4.1 Unit Testing	39
4.2 Interface Testing	40
4.3 Integration Testing	40
4.4 System Testing	43

4.5 Regression Testing	43
4.6 Acceptance Testing	44
4.7 Security Testing	45
4.8 Results	46
5 Implementation	47
6 Professionalism	48
6.1 Areas of Responsibility	48
6.2 Project Specific Professional Responsibility Areas	49
6.3 Most Applicable Professional Responsibility Area	50
7 Closing Material	51
7.1 Discussion	51
7.2 Conclusion	51
7.4 Appendices	51
7.4.1 Team Contract	51

List of figures/tables/symbols/definitions (This should be similar to the project plan)

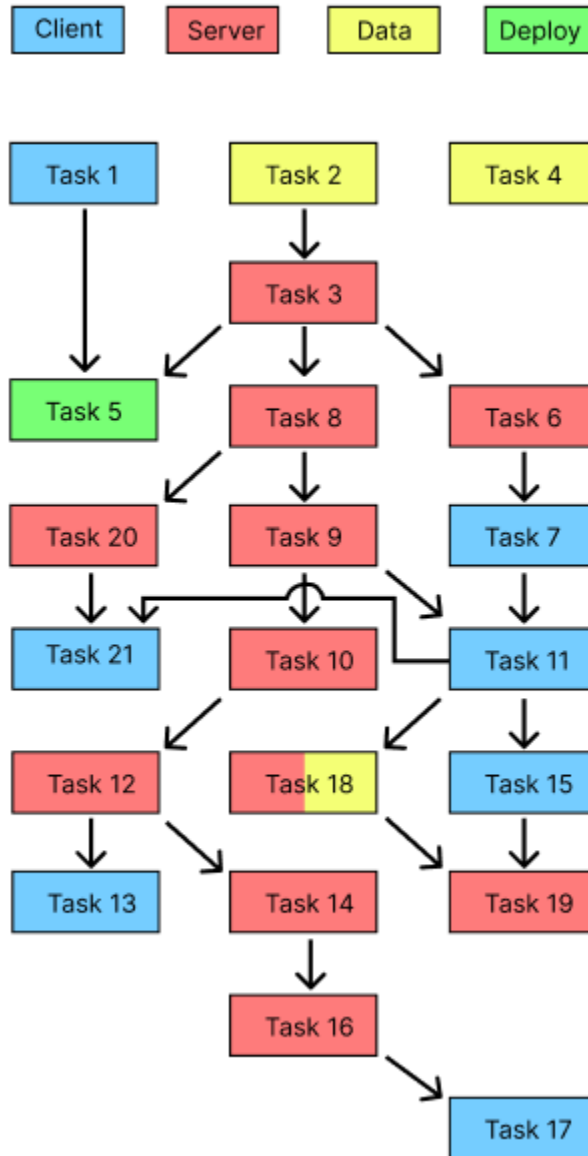


Figure 1

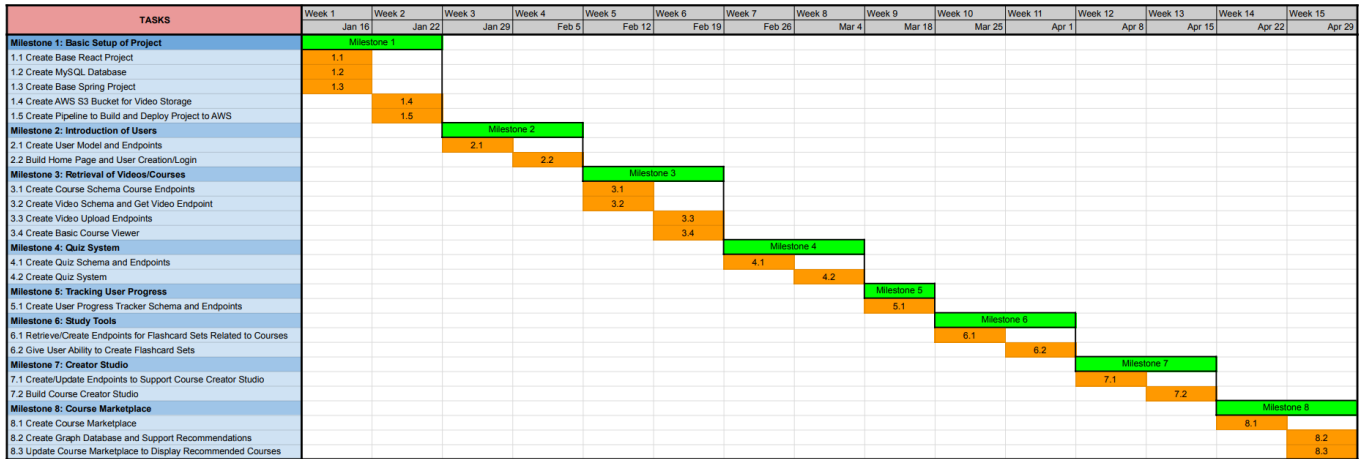


Figure 2

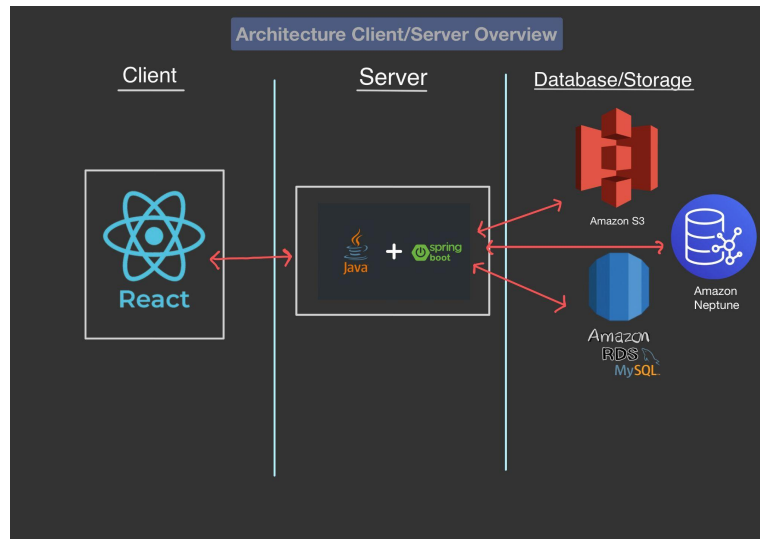


Figure 3

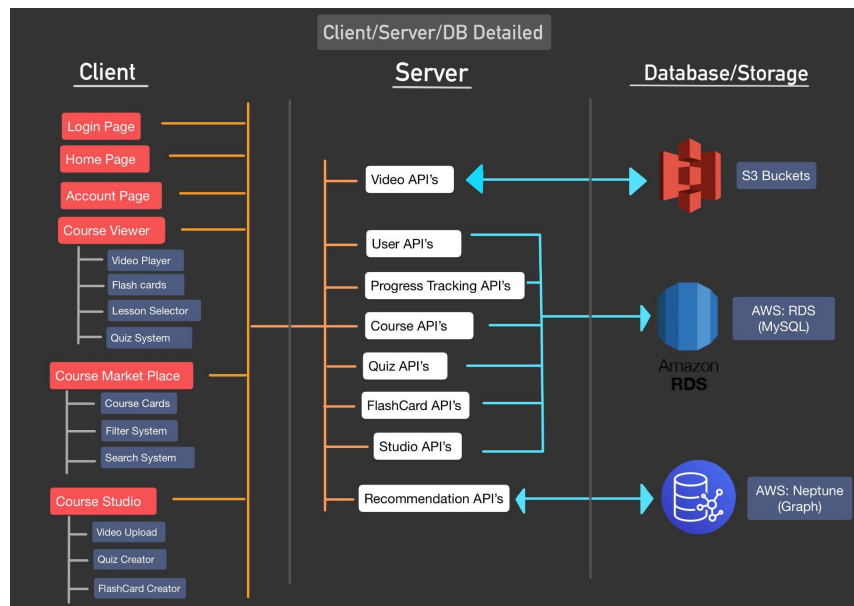


Figure 4

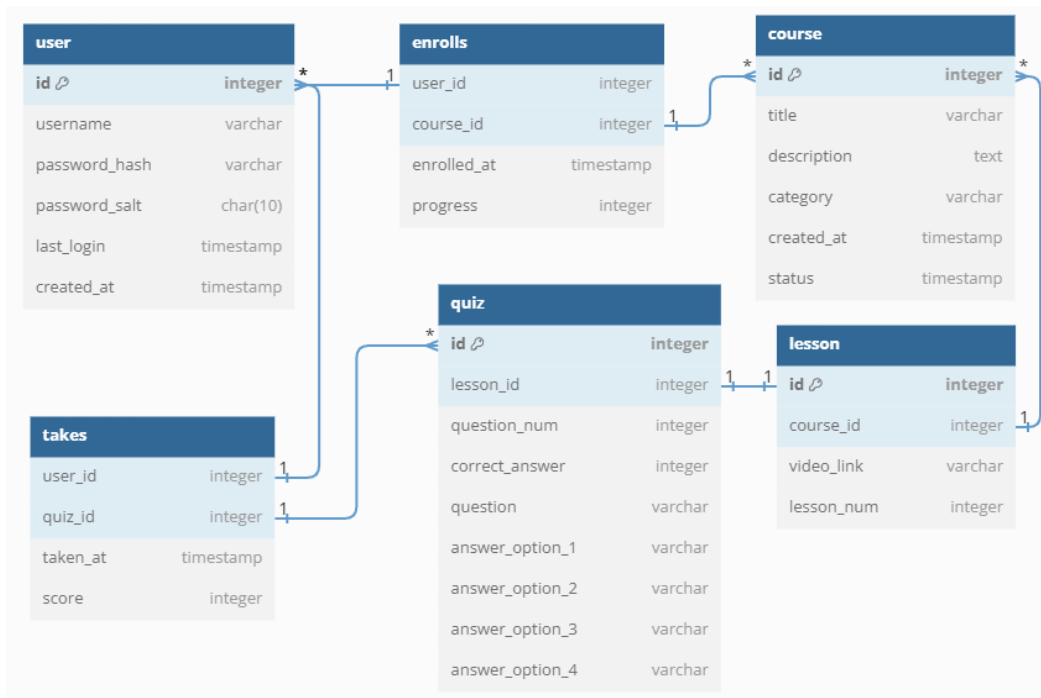


Figure 5

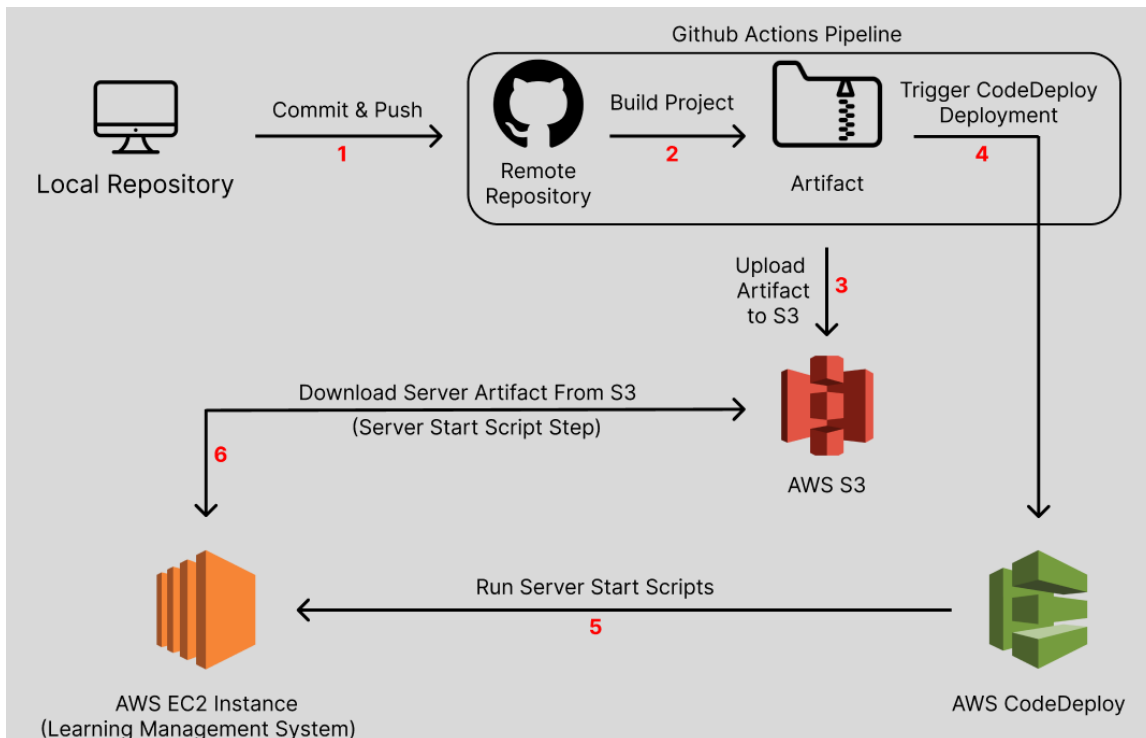


Figure 6

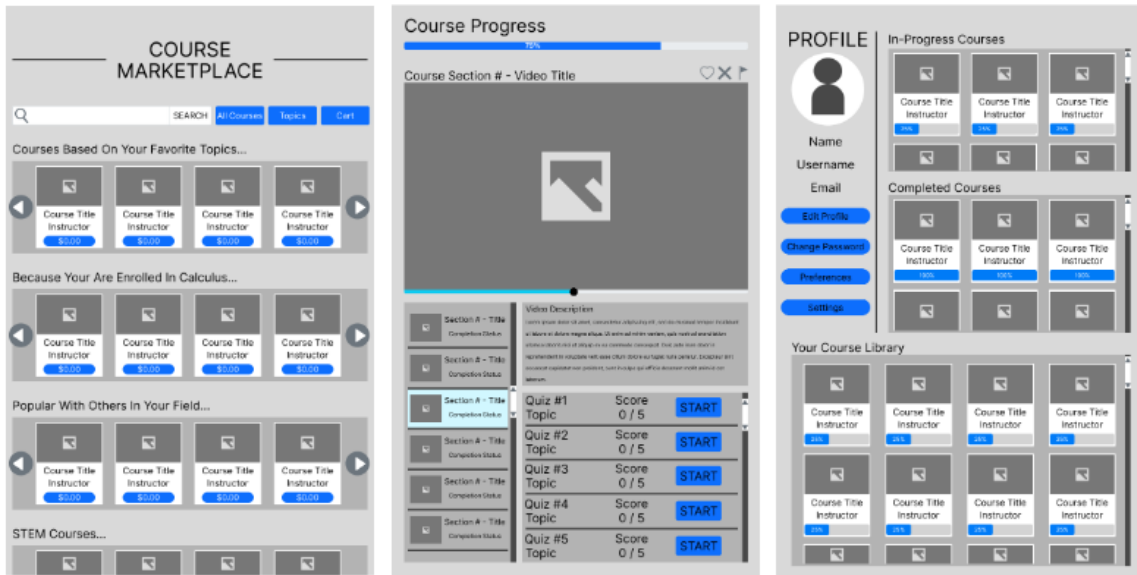


Figure 7

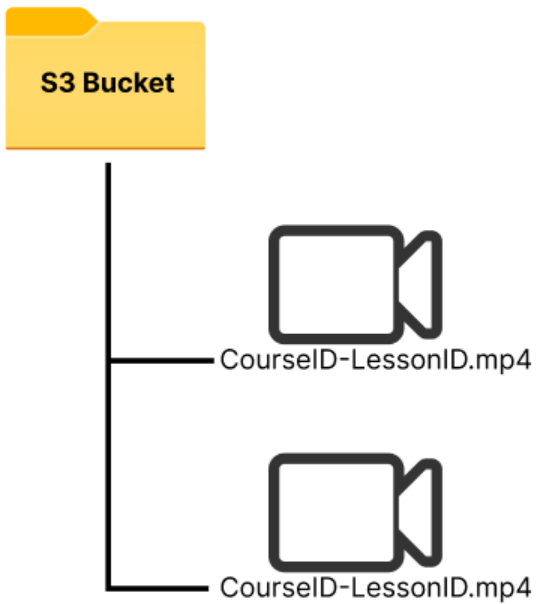


Figure 8

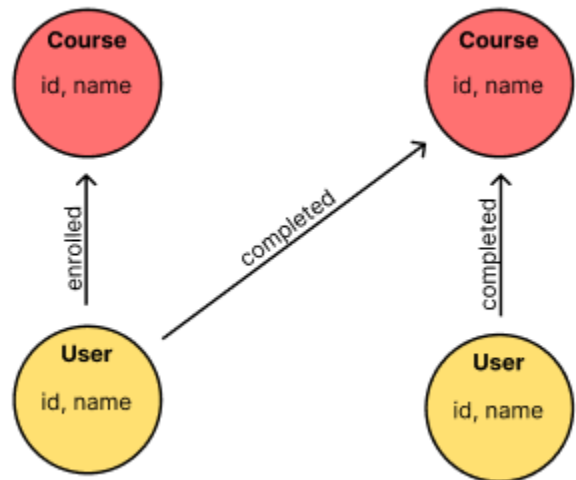



Figure 9

Course Progress

75%

- Lesson # - Title
Completion Status
- Lesson # - Title
Completion Status
- Lesson # - Title
Completion Status
- Lesson # - Title
Completion Status

Lesson # - Title



Notepad

Flashcards

TERM DEFINITION

Quiz #1
Topic Score 0/5

Quiz #2
Topic Score 0/5

Course Marketplace

All Courses Topics Cart

Courses Based On Your Favorite Topics...

- Course Title
Instructor
\$0.00
- Course Title
Instructor
\$0.00
- Course Title
Instructor
\$0.00
- Course Title
Instructor
\$0.00

Because You're Enrolled In Calculus...

- Course Title
Instructor
\$0.00
- Course Title
Instructor
\$0.00
- Course Title
Instructor
\$0.00
- Course Title
Instructor
\$0.00

Figure 10

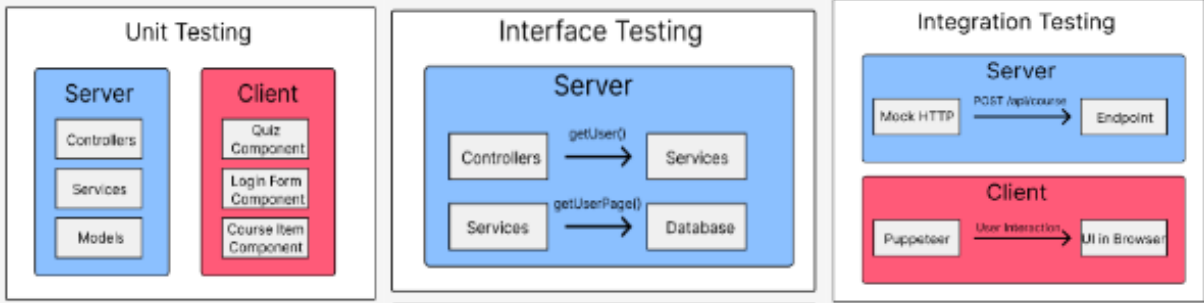


Figure 11

1 Team, Problem Statement, Requirements, and Engineering Standards

1.1 TEAM MEMBERS

Sam DeFrancisco, Jennifer Robles, Nicholas Erickson, Brayton Rude, Naga Vempati, Nikhil Kuricheti

1.2 REQUIRED SKILL SETS FOR YOUR PROJECT

- Frontend Development
 - Frontend framework and component system to build a user interface
 - UI/UX design
- Backend Development
 - REST API needed to handle user interactions from the web client
 - Database to store user and course information
- Cloud Development
 - AWS services to host the server, client, database, and image storage
- Continuous Integration / Continuous Delivery
 - Create pipeline to build, test, and deploy new versions of our system
- Version Control
 - Use Git for code repository and version control
- Software Architecture
 - Design client server application and an interface to communicate to each other
- Project Management
 - Communicate with client to determine project requirements
 - Plan all pieces of work and set milestones and time estimates for each
 - Delegate work items to team members in proper order

1.3 SKILL SETS COVERED BY THE TEAM

Sam: Full Stack Development

Jennifer: Frontend Development

Nicholas: Full Stack Development

Brayton: Full Stack Development

Naga: Frontend Development

Nikhil: Frontend Development

1.4 PROJECT MANAGEMENT STYLE ADOPTED BY THE TEAM

Our team will use an agile scrum methodology. We will have bi-weekly standup meetings and 2 week sprints. We will use GitHub as our tool to track issues and progress.

1.5 INITIAL PROJECT MANAGEMENT ROLES

We've taken a community project management role. During our design phase we have shared the responsibility of delegating tasks and planning work. This is subject to change if needed.

1.6 PROBLEM STATEMENT

Our goal is to create an online learning platform that meets the growing need for a comprehensive educational system by not only delivering quality content but also seamlessly integrating innovative study tools. We will be providing students with a complete learning experience, incorporating comprehension quizzes, personalized flashcards, and sophisticated recommendation algorithms. We aim to benefit a diverse student population, offering convenience and efficiency. Additionally, our platform will be designed to benefit educators by offering the ability to create courses. With features like an intelligent recommendation algorithm for courses and an integrated flashcard creation system, our technical approaches are geared towards ensuring a smooth user experience for both students and educators.

1.7 REQUIREMENTS & CONSTRAINTS

Functional Requirements

- Course System
 - Contains various video lessons and quizzes
 - Contains local flashcard set
 - Tracks user progress
- Lesson System
 - Displays video player with lesson video
 - Lesson can have a quiz
- Flashcard System
 - User can create flashcard sets and notes for a course
- Course Marketplace
 - Search courses by name and genre
 - Courses are recommended to student based on previously enrolled courses
- Creator Studio
 - Allow educators to create courses
 - Upload videos directly or link to YouTube for lessons
 - Allow creating quizzes for a lesson
- User
 - Accounts
 - Enroll in courses
 - Save user course progress and flashcard sets

Resource Requirements

- AWS
 - EC2 servers
 - AWS S3

- AWS RDS (MySQL)
 - AWS Neptune (Graph DB)
- Github
 - Repository (version control)
 - Github Actions (deployment)

Qualitative Aesthetics Requirements

- Aesthetics should be simple and calming as to not distract students

Economic/market Requirements

- Freely accessible to students and educators
- Low and manageable maintenance costs

Environmental Requirements

- No environmental requirements for the nature of this software project

UI Requirements

- Common theme and styling across website
- Errors are handled and appropriately displayed back to the user
- Simple and uncluttered UI with minimal actions per page
- MUI components
- Navigation bar
- Interface works for both web and mobile

Performance Requirements

- Quick page and video load times
- Responsive user interface interactions
- Elastic scaling to handle variable user activity

Legal requirements

- Ensure that credit is provided for video content
- Disclaimers regarding content validity

Maintainability Requirements

- Clearly documented code along with a HOW TO CONTRIBUTE document
- Images demonstrating software architecture, data schemas, etc.
- Expected functionality and requirements should be able to be derived from tests

Testing Requirements

- Tests should cover all main functionalities of the application
- Expected functionality and requirements should be able to be derived from tests

1.8 ENGINEERING STANDARDS

IEEE 829 - Software Test Documentation

In our project, the IEEE 829 standard would apply to our project by how we document the testing process. Our LMS needs testing to ensure its usability and functionality. Following this standard would make sure our testing is well-documented and include test plans and reports of the results and any issues.

IEEE 830 - Software Requirements Specifications

This standard focuses on defining software requirements in a standardized manner. The requirements would include features, functionality, and user interface specifications. We would outline what the system does such as user registration, the ability for users to upload their own courses and/or quizzes/notes, and content management. Following this standard will help in creating a clear documented set of requirements that will help serve as a foundation for the development process of our project.

IEEE 1074 - Software Development Life Cycle

We would be following a structured process for developing our LMS. This will include phases like project planning, analyzing requirements, designing the system, coding, testing, and deployment. Having a defined software development cycle is important for managing our timeline.

IEEE 2001 - Web Site Engineering, Web Site Management and Web Site Life Cycle

This standard is relevant to our online learning platform because it provides guidance on designing, developing, and maintaining web-based applications. The standard also addresses security aspects and best practices related to internet applications. These are crucial to ensuring the integrity of our LMS.

1.9 INTENDED USERS AND USES

There are two main groups of users: students and educators.

Students who are seeking to learn educational content are the main user type. Students will be provided with quality content that engages their comprehension. Users will be able to create study notes and flashcards for a course and keep track of their overall progress. Additionally, students will be recommended other courses in the marketplace to continue their learning.

Educators are the secondary user type. Educators can provide their educational content in a more productive platform with our application. They can take either existing course series from youtube or directly upload their content to our system and create an organization course from it. Educators can build quizzes for individual lessons. With our platform, educators can be sure that their students are better prepared to learn from their content.

2. Project Plan

2.1 TASK DECOMPOSITION

Task Dependency Graph

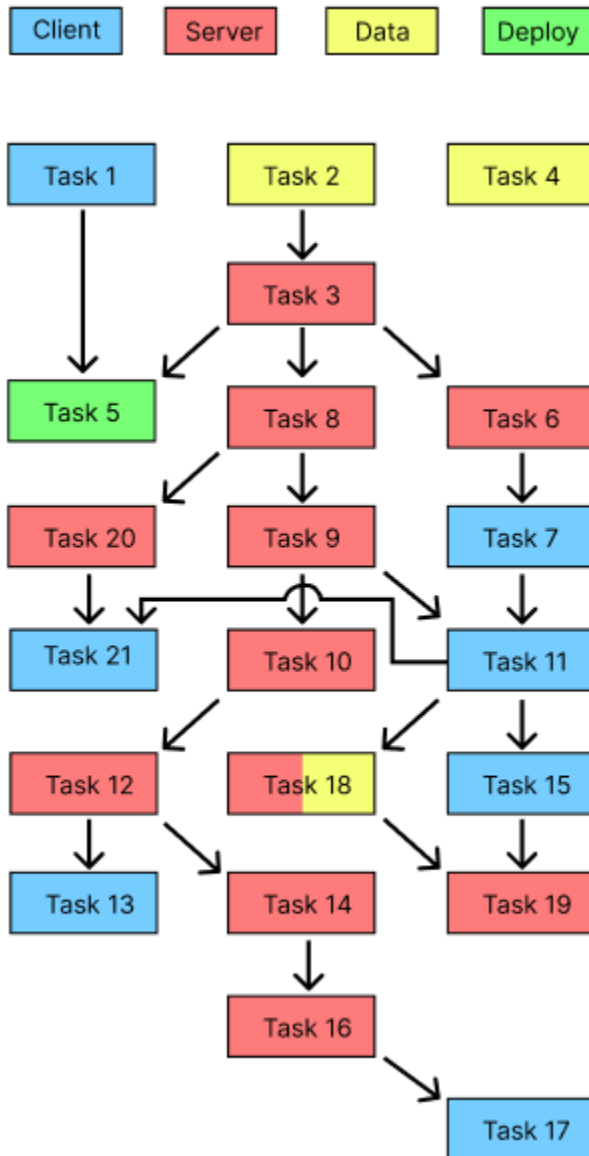


Figure 1

Individual Tasks

Task #1 [client]: Create base react project

- **Desc:** Initialize project and folder structure
- **Dependencies:** none

Task #2 [data]: Create MySQL database

- **Desc:** Initialize MySQL database creation and deployment
- **Dependencies:** none

Task #3 [server]: Create base spring project

- **Desc:** Initialize spring project and folder structure. Connect to MySQL database
- **Dependencies:** 2

Task #4 [data]: Create AWS S3 bucket for video storage

- **Desc:** Create a public S3 bucket to store lesson videos in
- **Dependencies:** none

Task #5 [deploy]: Create pipeline to build and deploy project to AWS

- **Desc:** Create a pipeline in Github Actions that will package the React and Spring Boot project and deploy to AWS on every new merge into master
- **Dependencies:** 1, 3

Task #6 [server]: Create user model and endpoints

- **Desc:** Create user table. Build getUser, createUser, and loginUser endpoints
- **Dependencies:** 3

Task #7 [client]: Build home page and user creation / login

- **Desc:** Create website home page. Create screens where users can create accounts and log in.
- **Dependencies:** 6

Task #8 [server]: Create course schema course endpoints

- **Desc:** Create course schema that has basic information such as creator, title, and lesson count. Create getCourse and getCourses endpoints.
- **Dependencies:** 3

Task #9 [server]: Create video schema and get video endpoint

- **Desc:** Create video schema that takes in name, course, lessonNumber, and video link. Link will be either a youtube link or AWS S3 bucket link. Create getVideos endpoint that returns all videos of a course.
- **Dependencies:** 8

Task #10 [server]: Create video upload endpoints

- **Desc:** Create upload video endpoints. One should create video entry based on youtube link. The second should take in mp4, upload to S3 bucket, and then create video entry.

- **Dependencies:** 9

Task #11 [client]: Create basic course viewer

- **Desc:** Create course viewer screens that display a course and its lessons. Each lesson shows a video. You can navigate between each lesson and the respective video will display.

- **Dependencies:** 7, 9

Task #12 [server]: Create quiz schema and endpoints

- **Desc:** Create quiz schema. Quiz should have a way to store questions and answers. Quizzes should belong to a lesson. Create getQuiz and scoreQuiz endpoints.

- **Dependencies:** 10

Task #13 [client]: Create quiz system

- **Desc:** Create a quiz system where quiz objects from getQuiz are converted into questions and answers. Track user answers and score the quiz at the end (scoreQuiz). Display score to user.

- **Dependencies:** 12

Task #14 [server]: Create user progress tracker schema and endpoints

- **Desc:** Create user progress tracking schema. Stores userId, course, lastLessonNumber, quizScores. Create getUserProgress and sendUserProgress endpoints

- **Dependencies:** 12

Task #15 [client]: Create course marketplace

- **Desc:** Create a shopping-like view that displays courses. Should be able to search and filter by courses. Can click into a course and choose to enroll in it from this screen. Additionally, show course recommendations based on previously taken courses in this view.

- **Dependencies:** 11

Task #16 [server]: Create / update endpoints to support course creator studio

- **Desc:** Create and update any required endpoints to support the build course creator studio such as add permissions or adding additional required parameters to createCourse, etc.

- **Dependencies:** 14

Task #17 [client]: Build course creator studio

- **Desc:** Build course creator studio that allows users to make their own courses. They should be allowed to add or remove lessons. They will be allowed to upload a video or reference a youtube video for each lesson. They may create their own quizzes and answers. After a course is created, it should display in the course marketplace.

- **Dependencies:** 16

Task #18 [data / server]: Create graph database and support recommendations

- **Desc:** Create graph database to store course types. Update getCourses endpoint to

support returning recommended courses based on courses a user has already taken.

- **Dependencies:** 11

Task #19 [server]: Update course marketplace to display recommended courses

- **Desc:** Add additional tab / header to marketplace to show a recommended courses section. Courses retrieved from asking for recommended courses from getCourses endpoint.

- **Dependencies:** 15, 18

Task #20 [server]: Retrieve/Create endpoints for flashcard sets related to courses

- **Desc:** Create any endpoints relevant for creating flashcard sets and pairing them with their course, as well as creating endpoints to retrieve flashcards by course

- **Dependencies:** 8

Task #21 [client]: Give user ability to create flashcard sets

- **Desc:** Within a course a user will be able to create flashcards or other similar tools to add to the course they are currently taking. Other users will be able to view these sets to use for their own studying.

- **Dependencies:** 11, 20

2.2 PROJECT MANAGEMENT/TRACKING PROCEDURES

Our team will use Agile. With our project being client focused, following an agile methodology will allow us to make continuous changes and improvements. Agile also includes integration testing, which we will be needing to continuously do to ensure a stable LMS. We will be using Agile's iterative approach when developing our site.

We will use GitHub Boards to track issues and progress. Discord will be used for communication.

2.3 PROJECT PROPOSED MILESTONES, METRICS, AND EVALUATION CRITERIA

Milestone 1 Basic Setup of Project

Tasks included: 1, 2, 3, 4, 5

In this milestone we aim to get the basic needs for our project setup. This includes getting a base react project started that is able to communicate with a dummy endpoint on our server side. To do this we will also need to get our AWS infrastructure setup to be available for requests. This will need our spring backend to be created to be hosted by AWS. From there we plan on getting a deployment pipeline setup for ease of future development.

Completion:

1. Have a static website that can hit an endpoint from our server successfully.
2. Pushing will redeploy

Milestone 2 Introduction of Users

Tasks included: 6, 7

This will be the start to our actual website. Starting with user tables and endpoints to be used for account creation/deletion/retrieval. We've also included the login system for the client in this milestone as it will use these user functionalities created on the backend.

Completion:

1. User can login from client side

Milestone 3 Videos/Courses

Tasks included: 8, 9, 10, 11

For this milestone we will work on introducing the main focus of the project which is the course system. This will require creating schemas for courses as well endpoints to retrieve them. We plan on storing videos in s3 buckets so we will need to connect courses to the videos/links to them. Creators will need to be able to upload to our s3 buckets so this will require endpoints for that as well.

Completion:

1. Can upload video from client side
2. Can retrieve video from client side
3. Clicking a course will retrieve relevant videos

Milestone 4 Quiz System

Tasks included: 12, 13

Users/Creators will have the ability to create quizzes as study tools. This will require both a client side building tool, as well as the relevant endpoints to be able to save them to the database.

Completion:

1. Client side has tools available to create a quiz
2. Quiz is saved to database and linked with relevant course
3. Client side retrieves relevant quizzes when requested

Milestone 5 Tracking User Progress

Tasks included: 14

This milestone is small but could become larger in the future. We want to have the ability to track how far into a course a user has made it. We may go down to the seconds level within the video or just the right lesson number. This will require frequent updating of their progress from the client side to the backend. So we will need the necessary endpoints set up to allow for this.

Completion:

1. On refresh/redirect a user will be taken to the video they left off on within the course.

Milestone 6 Study Tools

Tasks included: 20, 21

Something we want to add to make our project more learning friendly compared to Coursera or Udemy is the ability to add flashcard sets to courses. We will need to prototype what type of tools we want to have available. This involves adding the ability to create flashcards on the client and appropriate endpoints on the server.

Completion:

1. Client side can make a simple study set and upload it to the course
2. When a course is clicked the relevant study sets are displayed and can be clicked to be revealed

Milestone 7 Creator Studio

Tasks included: 16, 17

One of our main sources for content will come from creators of the platform. We want to create a course creator kind of system where they can upload their content, quizzes, and study sets. This will once again require us to create relevant endpoints on server side, as well as the user interfaces to complete these actions on the client side.

Completion:

1. Client side can create a new course
 - a. Client can upload a lesson (video)
 - b. Client can upload a quiz
2. Server side
 - a. When new course is created relevant information is stored to the database
 - b. Lessons are uploaded to s3 buckets and linked to the relevant courses

Milestone 8 Course Marketplace

Tasks included: 15, 18, 19

One of the big parts of this project will be finding relevant courses. We plan on creating a sort of marketplace to find these courses. This will require being able to retrieve all courses, or filtering courses by content. It will also need the user interface to be created to display the courses

Completion:

1. Client side can view all available courses
2. Client side can filter courses by topic

2.4 PROJECT TIMELINE/SCHEDULE

The following gantt chart starts on Tuesday January 16th, the first day of classes of the 2024 Spring Semester, and ends on Friday May 3rd, the final day of regular classes of the 2024 Spring Semester. Each column consists of one week of work, which totals to 15 weeks of work when excluding the week of Spring Break. The main structure of the chart follows the milestones derived in section 2.3 and attaches each milestone's corresponding tasks from section 2.1 within each milestone's timeframe.

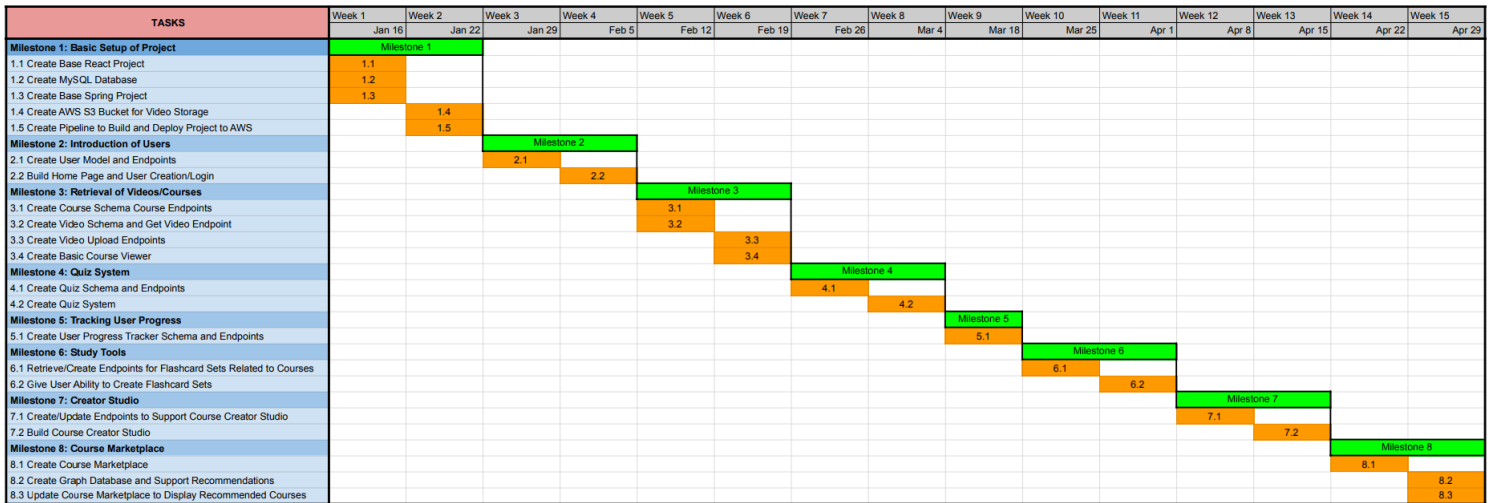


Figure 2

[Link](#) to Spreadsheet for a Better View

2.5 RISKS AND RISK MANAGEMENT/MITIGATION

Task 1-4 Initial Setup

Risk:

1. Trouble with setting up developer environment installed on each developer's machine

Solution

1. Meet with other members to troubleshoot environment

Task 5 Create pipeline to build and deploy project to AWS:

Risk:

1. Might be slight learning curve of deploying to AWS rather than something like GitLab that we have used in the past
2. Pipeline might need to change in the future

Solution:

1. Take necessary steps to learn about deployment to AWS before starting. Contact instructors and mentors for additional assistance.
2. Being prepared to further develop the pipeline as the project progresses. We can't be married to our V1 of the pipeline

Task #6 [server]: Create user model and endpoints

Risk:

1. One risk that could arise is security concerns
 - a. too much information being returned to specific user
 - b. Bad encryption leaking information
 - c. Permission issues

Solution

1. Security
 - a. Ensure that only relevant information is being returned in requests
 - b. Use heavy testing to ensure encryption/decryption is working properly
 - c. Ensure that users are given only the amount of access/permissions that are required. Can be proven through testing

Task #7 [client]: Build home page and user creation / login

Risk

1. User password security

Solution

1. Ensure we hide passwords as they are typed in such as *****

Task #8 [server]: Create course schema course endpoints

Risk

1. As development progresses our original schema becomes dated
 - a. Could cause issues with created endpoints

Solution

1. Try to create a flexible schema from the start that includes as much information as possible
 - a. Have well documented methods that explain exactly what it does, so when we need to update them it is easier

Task #9 [server]: Create video schema and get video endpoint

Risk

1. Inconsistencies between when we are linking a youtube video / aws s3 bucket link

Solution

1. Clearly identify which service is being used so the client side can handle embedding the videos correctly

Task #10 [server]: Create video upload endpoints

Risk

1. Handling different video formats when they are being uploaded

Solution

1. Possibly restrict video formats for creators? If not there is probably some sort of module we can use to differentiate the formats/normalize them.

Task #11 [client]: Create basic course viewer

Risk

1. Different browsers may handle embedded videos differently which could cause issues

Solution

1. We could either have a message that recognizes problem browsers and recommends a different one or we could try to handle based on browser how content is displayed

Task #12 & #13 [server]: Create quiz schema and endpoints && Create quiz system

Risk

1. Quizzes could have errors such as wrong answers
2. If a quiz includes a short answer, validating the answer could be difficult

Solution

1. Adding a user feedback system that can notify the creator
2. Require creator to provide a solution to short answer that will be displayed after user submits theirs for comparison

Task #14 [server]: Create user progress tracker schema and endpoints

Risk

1. If we try to track progress by the second we could run into issues with that
 - a. If users system were to crash we would fail to send an update to our system
 - b. Will need to come up with solutions for frequency of updates being sent to our system

Solution

1. Progress tracking
 - a. Restore from the most recent progress status from our system
 - b. Send updates on pauses, redirects, tab closes etc.

Task #15 [client]: Create course marketplace

Risk

1. Filters return irrelevant results

Solution

1. Continue to manually refine the algorithm that returns results
 - a. Add a button that hides unwanted results, and then sends the feedback to our system

Task #16 [server] && Task #17: Create / update endpoints to support course creator studio && Build course creator studio

1. Creator tries to upload things that aren't supported
2. Creator tries to upload malicious content

Solution

1. Restrict file types for uploads
2. Have feedback system that allows for users to flag bad content, possibly a review process on our side to deny/accept

Task #18 [data / server] and Task #19 [server]: Create graph database and support recommendations, and Update course marketplace to display recommended courses
Risk

1. Recommendation algorithm fails, and no recommendations are able to be displayed.

Solution

1. System will default to a set of predetermined courses to display on the marketplace in place of the recommendations

Task #20 [server] and Task #21[client]: Retrieve/Create endpoints for flashcard sets related to courses and Give user ability to create flashcard sets
Risk

1. User creates a flash card set that is not related to the course.
2. User content regulation could be a problem, since the feature allows for user created content to be public.
3. A user could attempt to spam the site by making large quantities of flash card sets.

Solution

1. User's can flag unrelated sets.
2. Have a system that flags unacceptable key words. When a set has been flagged, the user cannot make their set public until it has been reviewed.
3. Set a limit to the number of sets a user can create in a set window of time.

2.6 PERSONNEL EFFORT REQUIREMENTS

Tasks	Effort Hours	Description
Create base react project	4	Initializing project and folder structure to align to our project needs
Create MySQL database	4	Initializing database creation and deployment to our project needs
Create base spring project	4	Initialize project structure and connect to database
Build home page and user creation / login	24	Creating screens where user can create accounts and log in as well as the home page
Create basic course viewer	15	Creating a basic screen to display a course and its lessons
Create user model and endpoints	5	Creating basic endpoints for needed functionality
Create video schema and get video endpoint	5	Creating basic endpoints for needed functionality
Create video upload endpoints	5	Creating basic endpoints for needed functionality
Create course schema and course endpoints	5	Creating basic endpoints for needed functionality
Create quiz schema endpoints	5	Creating basic endpoints for needed functionality
Create flashcard endpoints	5	Creating basic endpoints for needed functionality
Give user ability to create and view flashcard sets	48	Users should be able to create flashcards and view other sets in another section of the site
Create user progress tracker schema and endpoints	10	Should be able to track users progress throughout all the courses the user is signed up for. Involves adding a bar under each course which shows how far you are in the course and auto updates.
Create quiz system	48	Based on the contents of the course, it should be able to create a quiz that accurately tests the user from what was taught in the course. The quiz should be a combination of flashcards and multiple choice questions.
Build course "marketplace"	48	This involves creating the screen that allows users to search for a particular course or see recommended courses.
Update course marketplace to display recommended courses	24	Adding tabs/header to marketplace and have it retrieve courses from getCourse endpoint
Create course creator studio	72	This involves creating multiple features within the website that will allow users to upload their own course. This means different types of videos should be compatible and that they comply with the format of the rest of the courses.
Create graph database and support recommendations	72	This will take a quite amount of time because we need to come up with and implement an algorithm that will recommend courses based on user's previous activity.
Create AWS S3 bucket to store videos in	1	Create S3 bucket to store videos
Build pipeline	24	For building and deploying throughout project
UI Testing - design	24	Find group of users to test functionality of site. Take their feedback and implement needed changes/improvements.
Testing	48	Includes continuous testing as we develop different functionalities and create and test endpoints

2.7 OTHER RESOURCE REQUIREMENTS

We will require various AWS services for hosting our web application and CD actions.

3 Design

3.1 DESIGN CONTENT

The design content for our project involves a combination of UI elements such as wireframes and mockups that will help us visualize the layout and structure of our platform as well as UX elements that will map out user flows and interactions. Since the frontend of our website will be programmed in React, we will need to define the hierarchy of React components that will help to make the website's design consistent. Overall, the website will be user-friendly and focused on making the user's interaction with our application as smooth as possible. In terms of the backend design considerations, the application will be designed using a database scheme such as MySQL to store user data, course information, videos, quizzes, and user progress. The design will ensure smooth integration between the React frontend and Spring Boot backend, including handling data retrieval and updates. Additionally, the algorithm that recommends courses will be designed to suggest courses to users based on their previous course history and selected interests among other data analytics. The design will be implemented to give users the ability to report inappropriate content and provide feedback.

3.2 Design Complexity

As opposed to plain Javascript, using React for frontend development means that our application is built on a component-based architecture, which allows us to break down the user interface into reusable components. This approach enhances modularity and each component can have its state, props, and lifecycle methods, making the project structure more intricate. Managing the state of different components is crucial in ensuring they reflect the correct data. Also using AWS S3 integration allows for scalability because the application can seamlessly handle the storage and retrieval of large volumes of data. Using a graph database for course recommendations utilizes the engineering principle of efficiency because a graph database can model data in a way that represents complex relationships between data points, which is crucial for generating accurate and efficient recommendations. We will be keeping with modern software development practices by creating a CI/CD pipeline. Our pipeline will build our project, run tests, and deploy the last version to an EC2 instance on merges to master.

Our application involves multiple challenging requirements ranging from features like a recommendation algorithm to integrated video storage, and user-generated content. These advanced features are in line with the industry's demand for interactive and personalized learning platforms. Another example is the course creator studio, which allows users to upload lessons, videos, quizzes, and study sets. This feature expands the platform's capabilities and complexity and requires sophisticated content management and user permissions systems. Tracking user progress is also a technically demanding task with significant implications for the user experience. It involves continually recording the user's progress and timestamp, storing it efficiently, and ensuring that when a user returns to a course, they can pick up from where they left off without delay. Achieving this relies on data structures and synchronization mechanisms to ensure that progress data remains accurate and up to date. All of these requirements match industry standards that top learning platforms like Khan Academy or Coursera use.

3.3 Modern Engineering Tools

Several modern engineering tools play essential roles in the design and development process of our application. React is used as a front-end framework, providing an efficient way to build dynamic user interfaces and manage complex client-side logic. Spring Boot is used as the backend framework which helps in facilitating data management and user authentication. AWS integration provides many cloud-based services for us to deploy our application. AWS assists in our server hosting, data storage, and continuous delivery. GitHub is also a crucial tool in our project's foundation. GitHub actions allows us to create the CI portion of our pipeline while GitHub repositories allow us to manage version control.

3.4 DESIGN CONTEXT

The communities that our learning platform is designed for and affects include students, educators and content creators, and educational institutions. The primary users of the platform are students. It caters to their educational needs by providing a centralized platform for accessing educational resources and user-driven study tools. Educators and content creators form another community. They can use the platform to share their expertise by creating courses. Educational institutions can also be impacted because they may choose to utilize the platform as part of their teaching curriculum. The societal needs addressed are access to quality education, customized learning, and efficient learning. We provide a one-stop solution for students to access educational content and user-driven study tools. Promoting customized learning caters to individual learning needs. Our platform helps the learning process by organizing content based on topics, tracking user progress, and offering study tools like flashcards and quizzes, making learning more efficient and effective.

Considerations related in each area:

Public health, safety, and welfare

Our project positively affects the public well-being by providing students with quality educational resources, potentially improving their learning outcomes.

Global, cultural, and social

Our platform should respect cultural and ethical values, ensuring that the content provided doesn't violate ethical standards.

Environmental

As a software product, the project has minimal direct environmental impact. However, we need to ensure that infrastructure management follows sustainable practices to minimize energy usage.

Economic

As a team, we need to take into account the cost of maintaining servers and cloud infrastructure. We also need to take into consideration keeping our platform affordable for users to ensure accessibility.

3.5 Prior Work/Solutions

The field of online education has had growth over the years, especially with the rise of digital platforms and educational technology. There are various existing online learning platforms that aim to enhance the learning experience by providing a wide range of educational resources and interactive tools. Some of the prominent learning platforms include Coursera, Khan Academy, Udemy, and Quizlet. Coursera offers a wide range of courses, including those from top universities and institutions. It provides video lectures, quizzes, and assignments. Khan Academy focuses on K-12 education and provides video lessons, practice exercises, and personalized learning dashboards. Udemy is a platform that allows educators to create and sell courses on various topics. It offers a marketplace for both free and paid courses. Quizlet provides a variety of study resources and tools. These include flashcards, quizzes, and practice tests.

Pros/Cons of our target solution:

Pros

- *User-driven study tools*: allows students to customize their learning resources
- *Personalization*: personalized recommendations based on content can help students find the most relevant resources.
- *Content organization*: can make it easier for students to navigate and find what they need
- *Integration of learning resources*: integrates educational content and study tools in one place, reducing the need to search multiple sources; combines the interactive study tools from Quizlet with the course offerings from Coursera, offering students a well-rounded learning experience.

Cons

- *Competition*: there are existing established platforms like Coursera, Udemy, Quizlet, etc.
- *Content quality*: ensuring the quality of user-generated content and study tools is crucial
- *Marketing*: attracting users and educators to a new platform is challenging when there are already established learning platforms
- *Monetization*: charging users could limit accessibility, but offering it for free can be challenging for sustaining the project financially.
- *Content diversity*: If we offer a wide range of subjects/topics, it may require significant content generation and quality control

3.6 DESIGN DECISIONS

1. User interface design

We need to make clear decisions on layout, navigation, and color schemes to ensure a positive user experience. The layout of pages, menus, and navigation bars should make it easy for users to find and access different content. Having a consistent uniform style/design throughout the platform will create a professional cohesive look.

2. REST API interface design

Our REST API interface determines how the client will communicate with the server. Ensuring our requests and data models follow a similar pattern is crucial to ensure ease of development and maintainability. Additionally, we need to consider keeping our interface abstract as tasks down the road will slightly alter existing data models and endpoints.

3. Software Architecture

We had to make a decision on how to deploy our application. While a requirement was to deploy the platform to the cloud, there were no specifics on the implementation. We decided to use AWS services as they are the most popular provider and will have the largest information base. Additionally, we chose to use GitHub actions for the pipeline to keep everything local and confined as possible. Due to our small team and experience, we wanted to keep the amount of applications we use to a minimum.

4. Content management

We will determine how the content will be organized and tagged based on topics and subjects. This involves building a recommendation system that uses algorithms to suggest relevant content to users based on their preferences and learning history. This also involves quality control to ensure that content is appropriate and meets certain standards. Lastly, this involves the decision on monetization and whether we will charge users to use certain features of our platform.

5. Data security/privacy

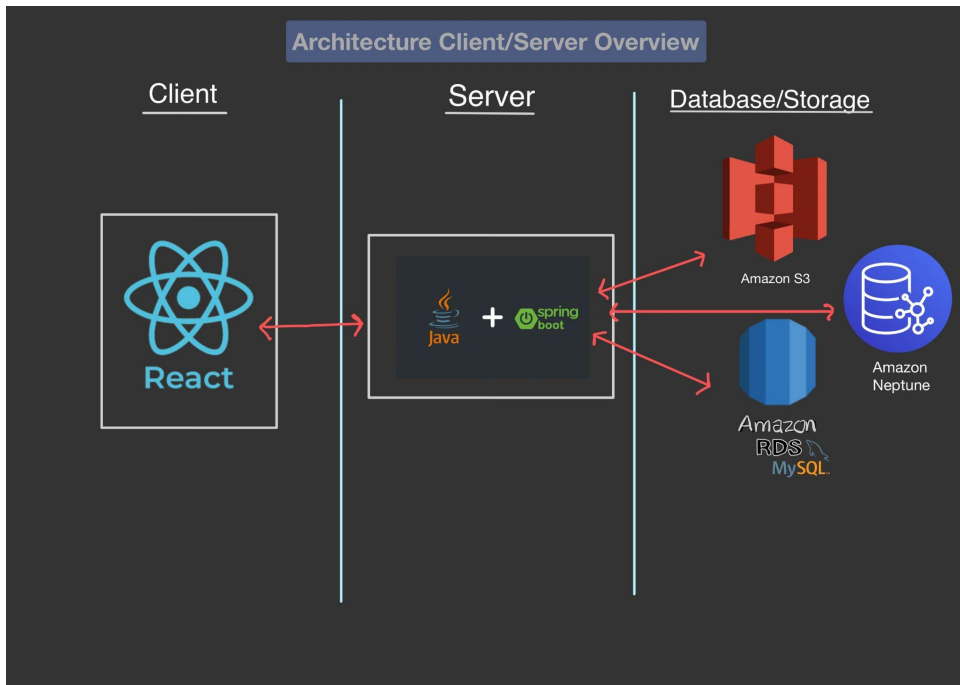
We will make decisions related to data security and privacy. This includes encryption methods for user data and content, and user authentication and authorization. Deciding on user data retention policies will ensure that users' personal and learning data are stored and managed securely.

3.7 PROPOSED DESIGN

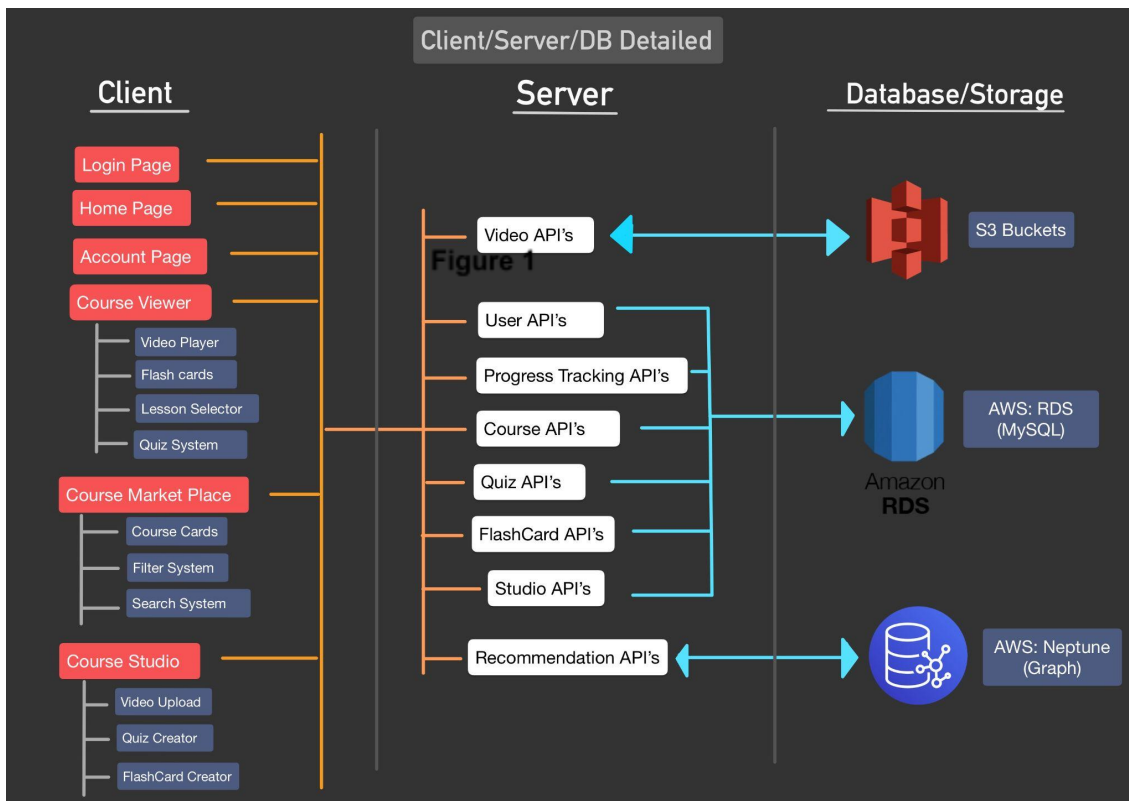
3.7.1 Design o (Initial Design)

Application Design

Application Architecture Overview (Figure 3)



Client/Server/Database More Detailed (Figure 4)



Architecture Overview Description (Figure 3)

This figure shows a basic overview of the tech stack we have chosen to work with. For the client side we are using ReactJS which will interface with our server built with the Java framework Spring Boot. For storage and database purposes we will be utilizing AWS services S3 buckets, RDS (relational database service), & Neptune.

Client/Server/Database More Detailed Description (Figure 4)

This diagram gives a brief model of each important system in our application.

Client

Main Pages/Components highlighted in red, branches represent subcomponents

- Login Page: Basic user interface that allows users to signup/login for our website
- Home Page:
 - Allows users to see current courses they are enrolled in
 - See recommendations for new courses/content
 - Navigate to other pages
- Account Page:
 - Change password
 - Change avatar
 - View learning badges/certificates
 - Change settings
- Course Viewer: Screen users will see when clicked into a course
 - Video Player: window displaying current lesson video
 - View available flash card/study tool sets
 - View available quizzes created for course
 - Choose lesson in course to view
 - Progress Tracking
- Course Marketplace
 - Browse new courses
 - Filter results
 - Search for courses
 - View recommendations
- Course Studio: What creators will use to publish new courses
 - Upload videos (lessons)
 - Create quizzes
 - Create study tools
 - Edit course description and other details

Server

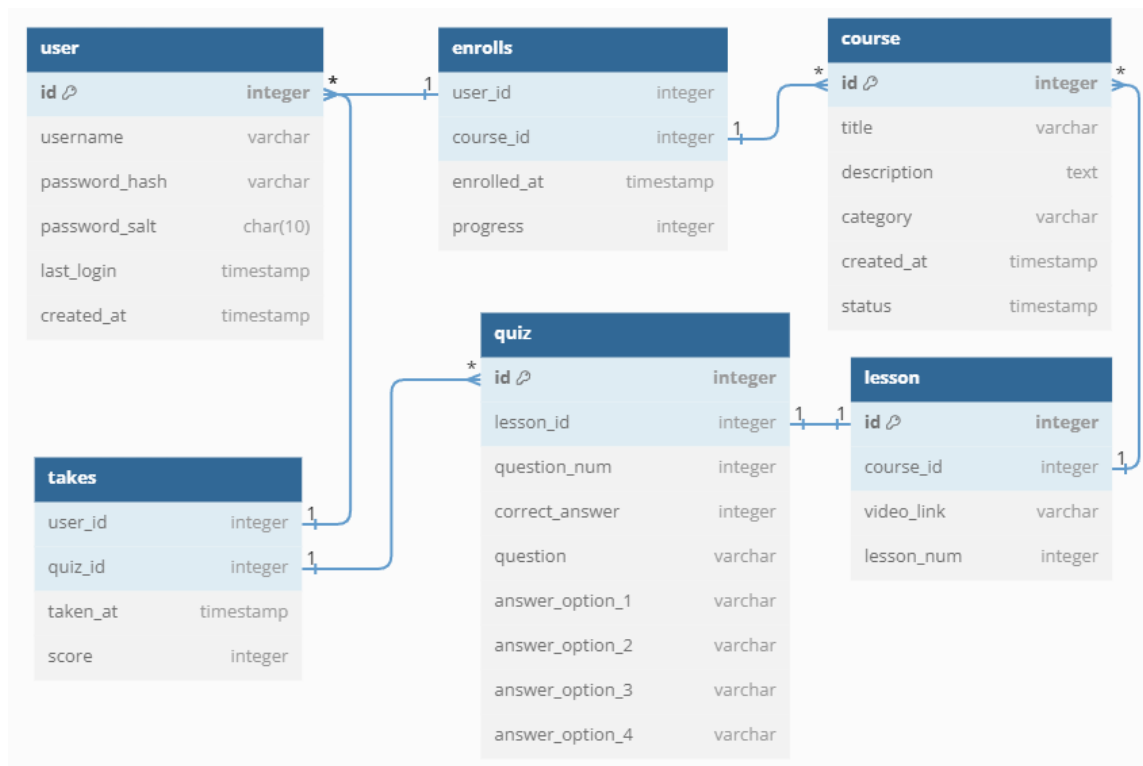
Each box represents a different functionality of our backend. API's work with different database services shown by the branching arrows

- Video API's: CRUD operations for the lessons in our system which are represented by video. We plan on storing video when needed in S3 buckets.
- User API's: CRUD operations for users. Users will be stored into our relational database and used throughout the application.
- Progress Tracking API: CRUD operations. We plan on allowing users to continue where they left off these api's will provide functionality to frequently update by users progress in their current courses
- Quiz API's: CRUD operations. Users/Creators will make quizzes for specific courses, these will be stored into tables within our relational database.
- FlashCard API's: CRUD operations. Users/Creators will also be able to create study tools similar to quizlet to hook to courses, the metadata for these cards will be stored in the relational database.
- Studio API's. CRUD operations for the creation of new courses.
- Recommendations API's. CRUD operations. Will interface with the graph database as well as the client to introduce new content to users

Database/Storage

- S3 Buckets: The main purpose of the s3 buckets will be to store video needed for courses
- RDS: Relational database that will store most of the metadata to our site.
- Neptune: Graph database service that will provide our recommendations to users

Application Database Schema (Figure 5)



Application Deployment Process (Figure 6)

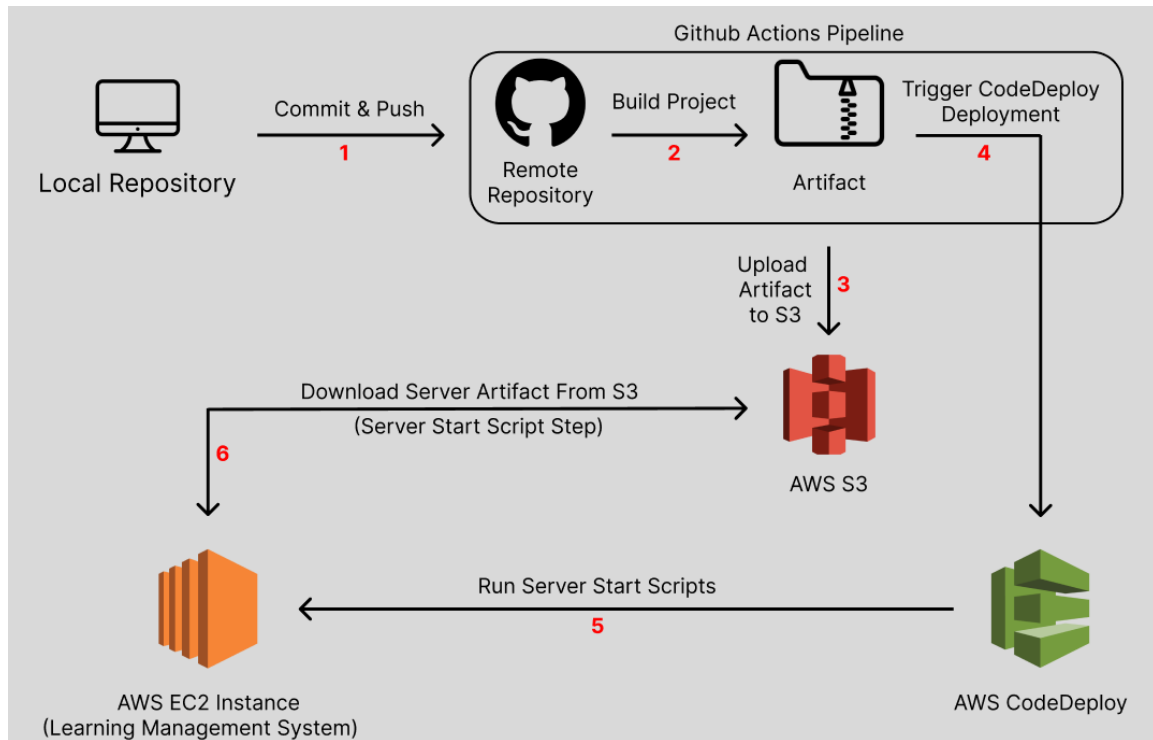


Figure 6 above shows the application deployment process. The red numbers label the steps in order. A brief summary of what goes on is as follows:

1. Developer adds and merges code to the repository.
2. The React client and Spring server will be built into one jar
3. The built artifact (jar file) will be uploaded to a S3 bucket
4. On successful Github Actions pipeline finish, a signal will be sent to CodeDeploy
5. CodeDeploy will enter an EC2 instance and run bash scripts to stop the current app
6. CodeDeploy then does a start script that downloads the new artifact from S3 and starts it

Client Design

When users initially navigate to our page, they will be taken to the login screen, and from here, users will be able to log in as either a student or instructor. Once logged in, users will land on the home page, which will act as the main hub for our users. The home page will act as a bridge to all the main aspects of our application. There will be navigation buttons to guide the users to other main pages within the application, like the user account page, the course marketplace page, or the creator studio. Throughout the application, we are implementing a recommendation system that will suggest various courses that users will find interesting or relevant to their current studies. The home page and course marketplace pages will make use of the recommendation system. There will be a Course Viewer page where users can see their course progress, watch course content, and select

study tools like quizzes and flashcards to use for practice. Users will be able to create their own courses with Creator Studio. They'll also be able to search for and find new courses to enroll in through the Course Marketplace.

A few mockups of the pages are provided below:

- Courser Marketplace
- Course Viewer
- Account

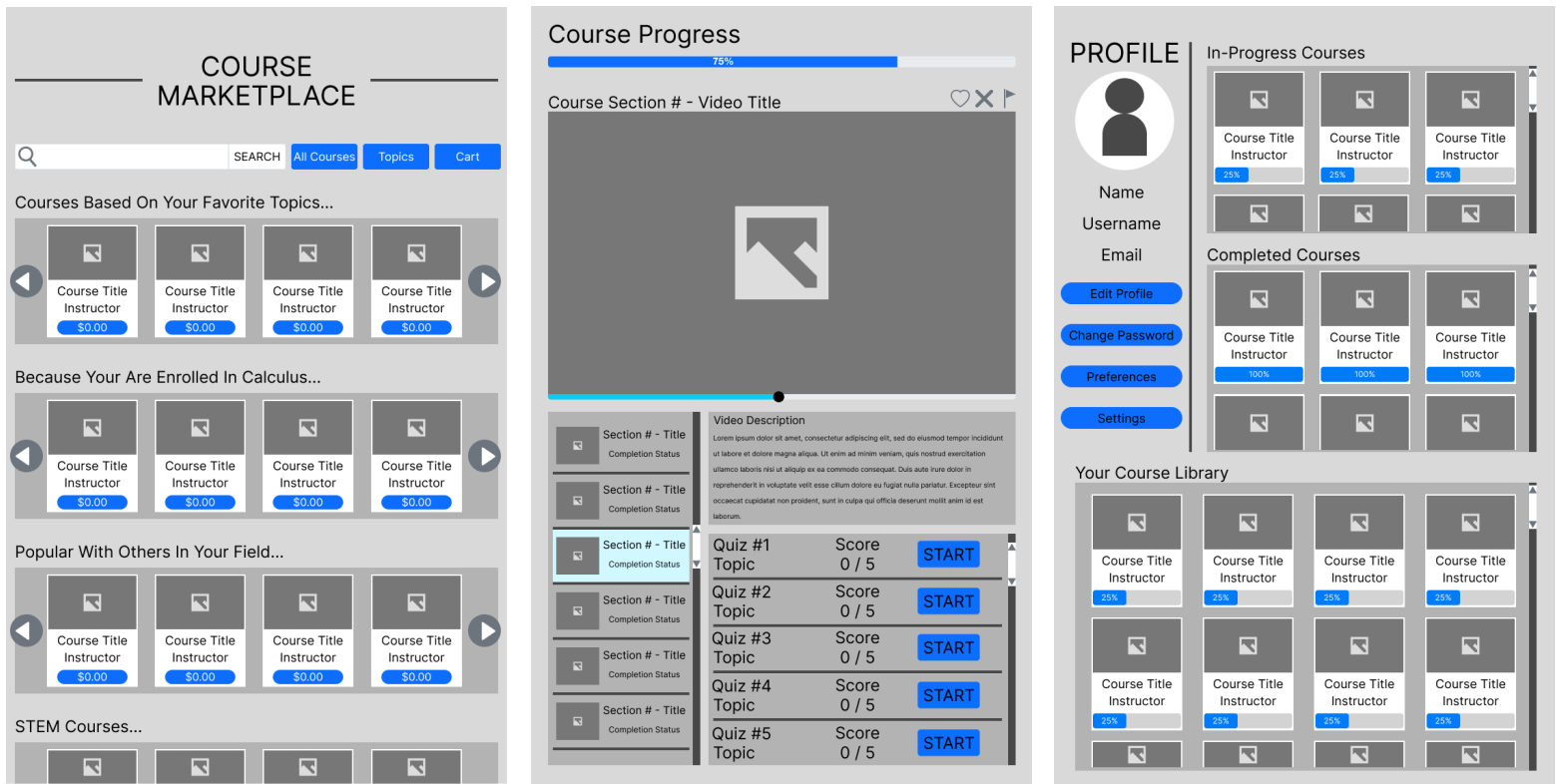


Figure 7

3.7.2 Design 1 (Design Iteration)

Application Design

During our first design we were not confident on video storage and graph databases. While our research had pointed us to use these technologies, we weren't certain on how to implement them. After doing further research and experiments, we have come up with plans and have created visuals to demonstrate our data schemas.

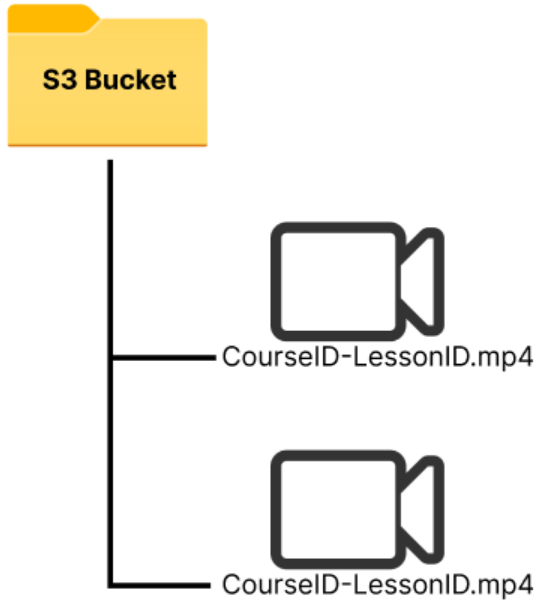


Figure 8

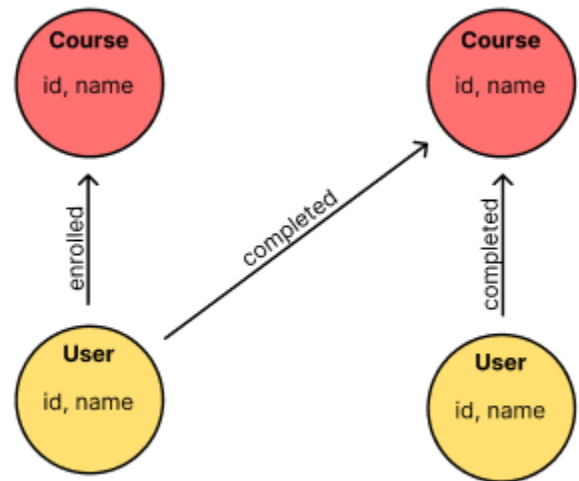


Figure 9

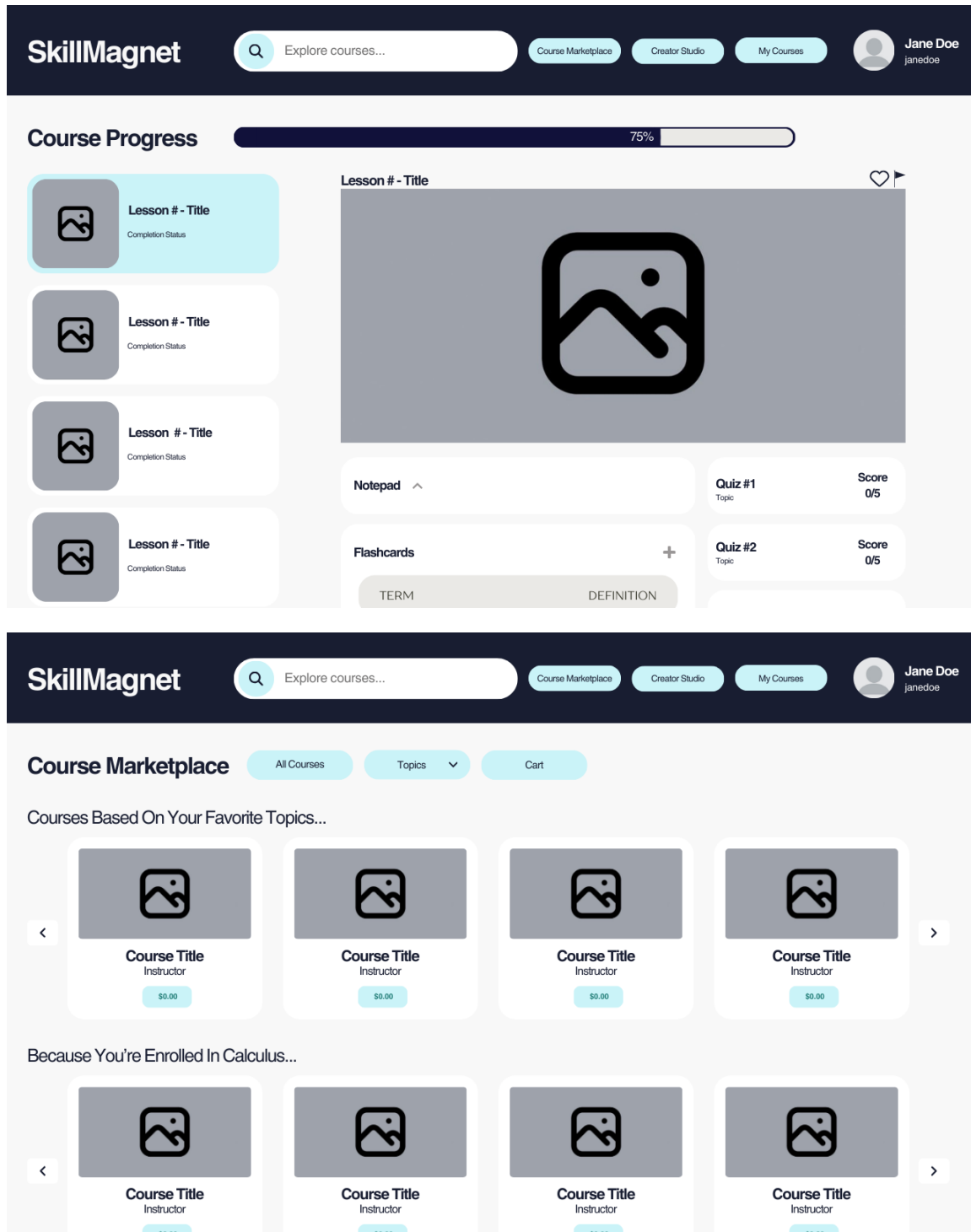
Figure 8 shows our AWS S3 storage schema. S3 works very similar to a folder on your computer however it lives on AWS. The benefits of this include very cheap storage as well as being practically infinitely scalable. The largest benefit is our ability to serve content from AWS S3 straight to our website. Once we get videos to S3 we can then play those videos on our website without having to manage the video data ourselves.

Figure 9 shows our AWS Neptune graph database schema. The circles are nodes and the arrows are edges. The main purpose is to track which courses different users have enrolled, favorited, and completed. Due to the nature of graphs, we can very quickly parse through course enrollment data to find related courses for a given user.

Client Design

In our second iteration we reworked our layout of the client. There was discussion about how we really wanted the course layout to look as well as how the flashcards and quizzes played in. Additionally, we plan on using Material UI components so we reworked the mocks to use these instead.

Figure 10 shows mocks of the course viewer and the course marketplace - the two main screens.



Take note of the new MUI style components and more modern UI style. The design has large and simple 1-use buttons to ensure ease of use. This design gives a much smoother and relaxing feel with the objective being on the courses and lessons rather than the interface itself.

3.8 TECHNOLOGY CONSIDERATIONS

Deployment: We chose to use a combination of Github Actions and AWS services for deployment for ease of developer use. While using cloud technology and AWS is an additional discussion, the deployment process is worth considering. From researching and experience, using CodeDeploy is a nice way to run scripts on our EC2 instance. We could've used Github Actions with Terraform or other tools. However, from a developer perspective you can write and edit these scripts from the repo and these will automatically be updated in CodeDeploy. We won't have to edit the CodeDeploy but rather only the bash scripts it runs. This keeps our code and points of contact all local to one main spot. A reason for using Github Actions for the CI part of the pipeline rather than using AWS's entire suite is ease of access as well. Logging into AWS can be a hassle especially just to check on CI stages. Keeping that in Github is convenient for us to debug and see when our newest merge might be live. We will be issue tracking and doing code reviews in Github so it's nice if that's the only website we ever have to pull up during development (versus having Jira, AWS, Jenkins, Github, etc. open).

3.9 DESIGN ANALYSIS

Our current final design covers all basic requirements at this point.

The application architecture and deployment model are set in stone. We are following a very standard client server model. Once our CI/CD pipeline is set up and we have our initial project deployed, we should no longer need to be worried about any infrastructure / architecture items.

We have created our data schemas to the best of our abilities. The S3 and graph database are unlikely to change very much if at all during implementation as those are straightforward and very single-use. The SQL schema was created to the best of our knowledge. During implementation there exists a possibility that we need to add extra properties to models. However the main data types and their relationships should remain the same. With data abstraction that comes packaged with Spring, changing the data storage hosts (e.g. using GCS instead of AWS for MySQL db) should be of no issue should anything like that be necessary.

We have mocked our main screens on the client and are content with their ability to serve all of our core requirements. Students can find and enroll in courses in the marketplace. Students can go through lessons in the course with the option to take quizzes and write down notes. Minor styling changes may occur during development however the current layout is to stay.

One thing that could be iterated on is the course structure. Currently our structure is very rigid in that lessons must be in a progressive order and quizzes must exist only on a lesson. We could open up the design to have lessons be more open to choose as you go as well as have standalone quizzes.

4 Testing

4.1 UNIT TESTING

We will be writing unit tests for both the client and server of our codebase. The goal is to ensure that core units give the correct outputs based on what we input.

Client

Our tool of choice will be Jest and React Testing Library.

We will write unit tests on main reusable React components. This includes the video player, quiz taker, course viewer, navigation bar, marketplace items, flashcards, and our high level page shell. All of these components will be duplicated many times throughout the application and thus will have an interface of some sort. Other individual pages and views (such as login) will be tested through integration testing.

Our unit tests will supply different data into the various units. We will programmatically ensure the UI is displayed correctly. We will call various functions at the component level and verify their output to make sure data is being reacted to and processed correctly.

Here is an example of a quiz taker unit test:

- Instantiate quiz taker component
- Pass in question and list of answers
 - One case passes in normal questions + answers
 - One case passes in empty question or empty answers
 - One case passes in extremely long questions and answers
- Verify UI is correct in initial state
 - Are all the questions and answer choices displayed?
 - Is anything disabled or items appearing that should be hidden
- Simulate action
 - Call quiz taker component chooseAnswer() method
 - Call chooseAnswer() multiple times on different answers
 - Call submitAnswer() before chooseAnswer()
- Verify that the UI is correct after action
 - Check that the styling of a selected answer choice is changed
 - Check that other answers aren't marked as correct
 - Make sure the correct answer is still hidden until submitted
- Verify the internal output is correct
 - Did the component properly output a finished signal?
 - Did the component properly score the quiz?

The primary goal here is to ensure that the basic view items and component's logic is solid.

Server

Our tool of choice will be JUnit.

We will write unit tests for all of our various models, controllers, services, and helper classes. Our models include items like User and Quiz. Controllers include our API groups like the Course API or User API. Services include things like CourseService that is used to communicate to the database. Helper classes can be things such as AuthenticationUtil that performs password encryption and decryption.

Unit testing's goal is much simpler here. We will simply verify that each class and its methods produce what we desire.

We will verify that User and Quiz getters and setters return the correct items. All helper classes will act as calculators where we will pass data and expect a complete answer. In the case of an AuthenticationUtil, we will pass in various password strings and verify the hash output is correct.

Controller testing is a little trickier. Our controllers will be tested to make sure we get the correct data and format. This will require mocking out any service calls that are made. In unit testing, we are not concerned that a database writes the correct data or the service properly gets data from storage. Rather we care more generally that a service call was made at all and that we output the correct data in its correct form.

4.2 INTERFACE TESTING

Interfaces are heavily involved in our system. However, by nature of a software system, unit tests and integration tests provide very comprehensive coverage of this indirectly.

Our client interface tests involve child components passing data to each other which are covered in integration tests. Any client-server communication is both mocked and tested live in integration tests. Puppeteer and Jest drive the client interface interactions.

On the server side, the main interface interactions are between the controllers, services, and database. The controller is responsible for taking in requests and services for communicating with the database. In the unit tests, we solely care that the controllers and services give us the correct output and format of data. In interface tests, our tests will closely resemble the unit tests. The difference is that service calls will not be mocked in the controller to service relation and database response will not be mocked in the service to database relation. This will require use of the in memory database for service interface testing.

4.3 INTEGRATION TESTING

Our server and database are the largest critical paths in our system. While the client is responsible for showing and requesting the correct data, any damage and issues that may be caused from the client's request should be properly handled on the server and database. The server needs to reject bad requests (e.g. bad data, unauthorized) while also ensuring the database remains in a safe state.

As mentioned, the client is its own entity that is free to make requests as it pleases. Any requests made to the server are not the client's responsibility in terms of preventing damage. However, the

client is responsible for displaying an experience for the user and should be able to properly display data and handle any data request errors (i.e. bad requests made to the server).

Server

Integration testing will be crucial and the bulk of our tests on the server. We will test data against an in-memory H2 database using JUnit.

Integration tests will overlap the existing controller unit tests slightly. While many of them will take similar forms, these tests will be more concerned about the entire request flow. Rather than directly calling controller methods, we will be using MockAPI to make calls to the endpoint's URL (e.g. GET 'lms.com/api/user/john-doe'). These integration tests will contain a series of requests to test the various main functions of our app.

Here is an example of some integration tests on Create Courses:

- Make create course request (i.e. POST lms.com/api/course)
 - Send with empty data
 - Send normal course data
 - Send wildcard characters and some blank values
- Verify course request response
 - Verify server returns a 200 on valid data
 - Verify server returns appropriate 4xx code on any bad data
- Verify action completed by sending GET requests (**CRITICAL STEP**)
 - Get course with the returned ID from the create request
 - Verify all the data is there
 - Send a getAllCourse request to verify a course with bad data wasn't created

This testing is critical and is the largest telltale of how solid our application is. This should test all possible requests that the server may receive. The results we receive from this will inform us of any faulty request handlers or services, database issues, uncaught errors or exceptions, etc.

Client

We will test the client with a combination of Puppeteer and Jest. Puppeteer allows us to simulate the client in the browser and acts as a user clicking and inputting on rendered buttons.

For our client integration testing, our goal is to demonstrate that all UI components work with data. We will not use the live server but rather mock the server responses that the client will receive in this step.

Here is a list of the core interactions that will be tested:

- Login / signup
- Page navigation (go to profile, go to course marketplace)
- View account / edit account
- View course
- View lesson
 - Watch video
 - Take quiz
- View course marketplace
 - Search courses
 - Enroll course
- View course creator studio
 - Upload video
 - Create quiz
- View / create flashcards

These are the main user interactions on our platform and should be tested. Each of the pages / components involved in these core interactions will have their data mocked out. To ensure the UI is safe, we will mock not only 200 server responses with various data, but also 400 and 500 to make sure the UI can properly respond to server errors.

Additionally, these interactions will be strung together for more in depth integration testing. We may have a test that does the following:

1. Signup for account
2. Navigate to course marketplace
3. Enroll in course
4. View course
5. View lesson 1
6. View lesson 2
7. Take quiz on lesson 2
8. View account and verify progress

The super goal of integration tests is to verify that the core functionality of our application works. This is super important in terms of regression testing when making changes to a smaller component that is used in many ways throughout.

4.4 SYSTEM TESTING

Our system is not a hardware device or a smaller component but rather a complete application for human use. Our main end to end criteria is if a user can perform all the interactions necessary to use the platform. Puppeteer is the powerhouse of the system testing as it completely simulates the user experience. It will open a browser, click buttons and type inputs, and view the page from the same perspective as a user. While small bugs and hiccups can exist in our application, system testing gives us the confidence that the main purpose is still served.

The system level testing strategy is to ensure that the application works overall for an end user. The goal is to trust our system is safe when we see that all tests passed. We want confidence that everything works without having to manually test things once our changes are deployed. All of our previous tests work together to get us close to this.

Our server unit tests are helpful in determining that we get the correct outputs from inputs. Our client unit tests are beneficial in determining that our various components (e.g. quiz taker, quiz controller) properly work under all possible states and inputs.

Our interface tests ensure that the layers of our server can communicate properly with each other. While unit tests have proven each individual layer works, interface tests show that pieces work together as a system.

Our integration tests supply major overlap with system testing. For semantics purposes, we isolate server/database integration tests as a group and all client integration tests as a group. We mock at least one part of the end to end experience in our integration tests. Our integration tests demonstrate that both the client and server should work very well together and in all cases. Our server integration tests prove that a server can supply any data as well as handle errors and bad requests. Our client integration tests prove that the UI can use data from the server and properly support the main user interactions. However, they don't account for the noise and chaos introduced between actually communicating over the wire.

So while our system tests are backed by all the lower level tests, we can put the final confirmation of our application by adding end to end tests from the client to a live server. These tests will be primarily repeats of our client integration tests but against a live server. The smaller pieces (such as view marketplace, take quiz) can be left to the mock data however larger client integration tests (such as the sequence described in the client section of 5.3) will be simulated against the server. Doing so will demonstrate that our deployed application functions while communicating with HTTP.

4.5 REGRESSION TESTING

Our regression testing is mainly driven by the user experience. While small breaking code changes are not good, our number one priority is serving a learning platform to the public. If our platform is fully functional from the user perspective, our system is working successfully.

Here are the following tools we use to verify new changes do not break old functionality:

- Github
 - Using Git for version control and viewing other's branches and previous work is a major advantage in controlling any changes. GitHub's code reviews are a very useful tool for a developer to point out any questionable changes or future issues that may arise.
- Semantic versions
 - Versioning our releases plays a large part in finding issues. If a bug or breaking change manages to avoid our tests, we can track down the cause of this issue through versioning. We can view when a bug started to occur and find the version that contains the breaking change.
- CI/CD
 - We benefit greatly from a CI/CD pipeline. Every build will also run all of our tests. Any tests that fail will fail the build and prevent the changes from being merged into our master branch.
- Unit/Interface/Integration tests
 - We maintain comprehensive test suites that cover our application. We account for all server and client interactions that a user may make. Any breaking changes to our code should have a corresponding test(s) to alert us.
- System testing
 - System testing ensures the application fully works end to end from a user perspective. This also functions as a smoke test to catch any core functionalities that were broken that lower level tests couldn't catch individually.

Our critical features include:

- User authentication
 - Ensure users can sign up, sign in, password change, and verify email address
- Content creation and management
 - Validate creation and retrieval of videos, courses, and study tools
 - Ensure APIs work for content creation and retrieval
- Course enrollment and marketplace
 - Ensure users can browse, search, and enroll in courses
 - Ensure course marketplace displays courses correctly
- Progress tracking
 - Verify progress tracking system accurately records users' progress within courses

4.6 ACCEPTANCE TESTING

Functional Requirements

User Acceptance Testing

- Alpha/Beta testing
 - For an alpha version, we will let our friends/classmates/family play around with an initial version of the site while providing feedback
 - We could possibly make a survey to go along with the alpha version to get more helpful criticism/comments

- For beta testing, we will have a similar approach to the alpha version, but it will be more widespread, allowing anyone who wants to give it a try
- For testing our creator studio, we have thought of reaching out to creators on YouTube who create educational content and have them provide feedback on what could be improved/fixed

Taking in all of the feedback from these processes should help guide us in ensuring that our software is doing what is expected. There will probably be multiple versions of alphas and betas as new features arrive in our product, repeating this process will help find early bugs that allow for the best possible experience for users.

Non-Functional Requirements

Performance testing idea: Apache JMeter

For non-functional requirements, we will have four main areas to ensure are up to standards. Performance, reliability, usability, and security.

Performance

- Response Time: ensure that response time averages are fast, plays into scalability below
- Scalability: When user counts rise we need to ensure that our services scale upwards to allow for fast response time (availability)

Reliability

- Availability: services are available as often as possible, will aim for 95% uptime
- Fault Tolerance: Ensure within our code that we are constantly adding support nets for errors/exceptions, when something goes wrong we want to have some sort of exception handling in every situation

Usability

- User Interface: Make sure that all buttons work as intended, links are not dead. Ensure different screen sizes are usable, as well as browsers.
- Accessibility: Do all that we can to ensure that anyone can use our site regardless of disadvantages.

Security (discussed more in 5.7)

- Data Encryption
- Authorization and Authentication

4.7 SECURITY TESTING

Security testing is critical in our application as we are storing user information.

Our code linter serves as a form of testing for security. We will use ESLint which performs static code analysis to find out any major vulnerabilities. Common client vulnerabilities ESLint can help prevent are XSS and SQL injection.

Server integration tests will include authorization tests to ensure that requests without the correct permissions will be properly denied access. We also have unit tests to ensure our password encryption system properly works and outputs expected hashes.

Our server will be deployed on an AWS EC2 instance which provides us security against the infrastructure of our application. The responsibility of protection against things like DDOS attacks are managed by AWS. Our data storages will also all be sourced from AWS services thus leaving the outer protection of our data up to AWS.

4.8 RESULTS

The results of our testing achieve one central goal: ensure our application provides a learning platform for users. Great software design is something to shoot for however the user's benefit has the highest priority.

Our testing is very user focused with the bulk of our critical and the most useful tests come in the forms of integration and end to end tests. Our testing covers every individual piece from a user's perspective. We have a large focus on ensuring any server requests give us the correct output based on an input. We have a large focus on ensuring the client can handle all types of user inputs, states, and handle any errors.

Figure 11 represents basic diagrams showing a high level view of our tests. This depicts a better view of what / how each is being tested.

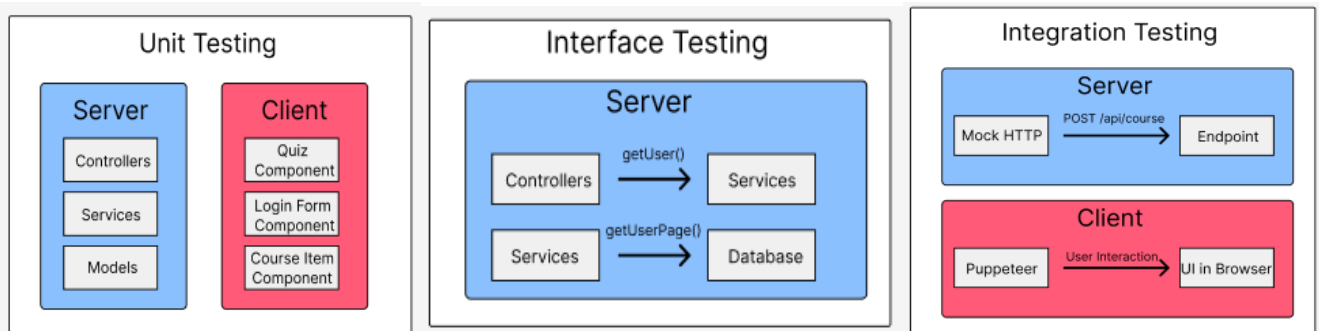


Figure 11

Our client integration tests as well as systems tests correspond one to one with requirements. Every action and procedure a user can make will be covered in a test. Software systems are hard to get correct and have a very dynamic range of input. Manually testing our software on each iteration becomes more impossible with the more features that are added. Thus by having a complete coverage of all user interactions, we can ensure our platform remains functional for the user.

5 Implementation

Our preliminary implementation plan for the upcoming semester begins with coming to a group consensus on certain design decisions. Some of the things we will agree on before the start of development are a:

- Linter
- API style (API urls, endpoint and controller names)
- Folder structure (page and component organization, global/local styling, models/api locations)

We will then begin the GitHub repo setup and have our sprint board ready with tasks. We'll set up the basic spring + react projects and have the pipeline to AWS built. Since we would have agreed on the different screens/pages, we can then start with frontend implementation of our designs.

6 Professionalism

6.1 AREAS OF RESPONSIBILITY

Area of Responsibility	Definition	NSPE Canon	SE Code
Work Competence	Perform work of high quality, integrity, timeliness, and professional competence.	Perform services only in areas of their competence; Avoid deceptive acts.	Software Engineers must ensure their products and modifications meet the highest professional standards possible.
Financial Responsibility	Deliver products and services of realizable value and at reasonable costs.	Act for each employer or client as faithful agents or trustees.	Software Engineers should provide products and services that bring realizable value.
Communication Honesty	Report work truthfully, without deception, and understandably to stakeholders.	Issue public statements only in an objective and truthful manner; Avoid deceptive acts.	Software engineers shall act in a manner that is in the best interests of their client and employer. They are expected to tell the truth, emphasizing clear and understandable communication.
Health, Safety, Well-Being	Minimize risks to safety, health, and well-being of stakeholders.	Hold paramount the safety, health, and welfare of the public.	The SE Code stresses the importance of keeping people safe and healthy by ensuring good management and reduction of risk.
Property Ownership	Respect property, ideas, and information of clients and others.	Act for each employer or client as faithful agents or trustees.	The SE Code emphasizes the importance of respecting others' property and ideas. Software Engineers must ensure that there is a fair agreement concerning ownership of any software, processes, research, writing, or other intellectual property.
Sustainability	Protect environment and natural resources locally and globally.		The SE Code emphasizes the responsibility to protect the environment. Software Engineers must identify environmental issues related to work projects and approved projects must not harm the environment.

Social Responsibility	Produce products and services that benefit society and communities.	Conduct themselves honorably, responsibly, ethically, and lawfully so as to enhance the honor, reputation, and usefulness of the profession.	Software Engineers should create things that help society and communities. Software engineers shall act consistently with the public interest. The ultimate effect of the work should be to the public good.
-----------------------	---	--	--

1. Work Competence
 - The SE code emphasizes performing work of high quality and integrity, but the NSPE canon emphasizes performing work only in areas of their competence.
2. Financial Responsibility
 - The SE code focuses on providing services/products at a reasonable cost. It doesn't specifically address financial responsibilities like the NSPE canon, which mentions acting as faithful agents or trustees.
3. Communication Honesty
 - Both the SE code and NSPE canon stress truthful and objective communication.
4. Health, Safety, Well-Being
 - Both the SE code and NSPE canon prioritize the safety and health of the public.
5. Property Ownership
 - Both the SE code and NSPE canon highlight the importance of respecting the property and information of clients/others and being faithful.
6. Sustainability
 - The SE code explicitly addresses protecting the environment, while the NSPE canon does not address environmental sustainability.
7. Social Responsibility
 - The NSPE canon however states that engineers conduct themselves honorably for the good of their profession. The SE code has a broader societal and community-oriented focus, emphasizing working for the public good.

6.2 PROJECT SPECIFIC PROFESSIONAL RESPONSIBILITY AREAS

1. Work Competence
 - Ensuring high-quality code, system reliability, and meeting project deadlines are crucial for the success of our project.
 - Team performance: Medium
 - Justification: Our team has thoroughly been planning and designing our learning management system, but until actual implementation begins, it's challenging to assess the competence in execution. We have plans set for ensuring high-quality code such as required code reviews and testing.
2. Financial Responsibility
 - Financial responsibility is essential for sustainability, as we must consider our use of AWS services and the need for a cost-effective infrastructure.
 - Team performance: Medium
 - Justification: Our team is considering the project's architecture and resource requirements and cost considerations will be factored into the design phase.

3. Communication Honesty
 - Clear communication within our team and transparent communication with users is essential.
 - Team performance: High
 - Justification: Clear and honest communication has been a priority during the planning phase. We have open communication to ensure everyone is on the same page and are committed to maintaining this and a higher level of communication during the development phase. We will have regular team meetings and keep each other updated throughout our sprints.
4. Health, Safety, Well-Being
 - While our project is software-based, ensuring data security and user privacy contributes to the well being of users.
 - Team performance: N/A
 - Justification: Since our project is software-based and doesn't involve physical safety/health concerns, this area is not applicable during our planning phase.
5. Property Ownership
 - Respecting intellectual property rights, especially when dealing with others educational content, is crucial.
 - Team performance: Medium
 - Justification: Our team is constantly considering how to respect intellectual property rights throughout our planning.
6. Sustainability
 - Choosing AWS services and considering the environmental impact of data storage aligns with sustainability goals.
 - Team performance: Medium
 - Justification: Sustainability considerations such as the use of AWS services have been factored into the design of our project. There will be ongoing efforts to monitor the environmental impact during our development.
7. Social Responsibility
 - We are providing a platform for accessible and personalized education, which aligns with the societal responsibility of benefiting a broader community (students and educators).
 - Team performance: High
 - Justification: Our team is committed to developing a site that is clear and easy to navigate to ensure a positive user experience and be able to provide access to quality education, customized learning, and efficient learning.

6.3 MOST APPLICABLE PROFESSIONAL RESPONSIBILITY AREA

The most applicable professional responsibility area for our project is "Social Responsibility". Our project aims to provide an online learning platform that has quality educational resources and user-driven study tools accessible in one place. Societal responsibility involves contributing to the well-being of society and communities. We aim to be a one-stop solution for educational resources, benefiting students by making their learning more efficient and effective. Our project aligns with the broader goal of contributing positively to education, and by extension, society.

7 Closing Material

7.1 DISCUSSION

Our project requirements have been met. We have designed a scalable web application that allows students to find and enroll in courses, view lessons and quiz their comprehension along with access to study tools. Our UI design is very simple and accessible for students. The server and data model follow standard software engineering principles. Our application is very scalable due to being deployed to AWS. We have improved the developer experience with the creation of our CI/CD pipeline.

7.2 CONCLUSION

We have conducted user experience research on various online learning platforms such as Udemy, Coursera, and Quizlet to create a better experience on our platform. We have investigated various database types and technologies to best determine how to model our data and where to store it. We have followed good software engineering practices with our application architecture, CI/CD pipeline, and test suite design.

Our overall project goal is very similar to our project statement. Briefly put, our goal is to provide students a platform that best encourages them to learn and study in various courses. Our milestones represent our bi-weekly project completion goals.

To ensure the project is completed on time, we must start strong next semester. Our plan of action is to create the GitHub repository and lay out the foundations of the basic project. We plan to get a working round trip between the client, server, and databases. We also want to have all our developer's local environments ready to go at the start of next semester. This will help reduce non-technical issues and ensure we are programming off the bat at the start of next semester.

7.4 APPENDICES

7.4.1 Team Contract

Team Name Learning Management System - Group 46

Team Members:

- | | |
|-----------------------------|---------------------------|
| 1) <u>Nicholas Erickson</u> | 2) <u>Jennifer Robles</u> |
| 3) <u>Sam DeFrancisco</u> | 4) <u>Brayton Rude</u> |
| 5) <u>Nikhil Kuricheti</u> | 6) <u>Naga Vempati</u> |
| 7) _____ | 8) _____ |

Team Procedures

- 1. Day, time, and location (face-to-face or virtual) for regular team meetings:**
Fridays ~ 1:00 - 5:00 (hybrid)
- 2. Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-mail, phone, app, face-to-face):** Group Discord server
- 3. Decision-making policy (e.g., consensus, majority vote):** Majority vote

4. **Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be shared/archived):** Scheduled meetings will be posted in the discord server and any important information from the meeting will be posted afterwards.

Participation Expectations

1. **Expected individual attendance, punctuality, and participation at all team meetings:**
Attendance: Try best to make every meeting on time, letting the team know in advance of absences, and give your thoughts/opinions.
2. **Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:**
Expect teammates to do their share of what we task out. If a teammate struggles, they should ask for help in advance. If a teammate does not communicate their struggles or shortcomings, they should be responsible for it.
3. **Expected level of communication with other team members:**
Respond to discord messages/emails in a reasonably quick time.
4. **Expected level of commitment to team decisions and tasks:**
Expect teammates to do what they say they will do. If their available effort or time is low, they should communicate this and the rest of the team should readjust.

Leadership

1. **Leadership roles for each team member (e.g., team organization, client interaction, individual component design, testing, etc.):**
Team org: all members share the same responsibilities. If leaders start to form naturally that is ok.
Individual Component Design: Try to build what is assigned in tasks. Obviously ok to get help/collaborate with other team members.
Client Interaction: Team meetings w/ company head to discuss requirements. Whoever can make it attends.
2. **Strategies for supporting and guiding the work of all team members:**
Effective and timely communication between team members on discord or gitlab. Utilize git lab issues and project management features to guide team tasks and project milestones.

- 3. Strategies for recognizing the contributions of all team members:**
If someone does a good job, let them know. Give shoutouts!

Collaboration and Inclusion

- 1. Describe the skills, expertise, and unique perspectives each team member brings to the team.**
 - Nicholas: 2 full stack web development internships touching common tech
 - Sam: Built a few websites (fullstack), one internship that was heavily involved with databases
 - Jennifer: Frontend development experience in job, experience working in agile scrum team
 - Naga: Have experience working in frontend and backend primarily from my internship and also from classes.
 - Nikhil: Have experience with frontend development from internship and classes.
 - Brayton: Full Stack App Development through class projects.
- 2. Strategies for encouraging and support contributions and ideas from all team members:**
 - Gitlab tasks will be assigned out during starts of sprints
- 3. Procedures for identifying and resolving collaboration or inclusion issues (e.g., how will a team member inform the team that the team environment is obstructing their opportunity or ability to contribute?)**
Make issues known early and discuss issues as a group. If necessary, cite the TA's as well to join in on the discussion.

Goal-Setting, Planning, and Execution

- 1. Team goals for this semester:**
Create an effective Project Document and approach that covers all aspects of the final product and satisfies both the client and ISU faculty.
- 2. Strategies for planning and assigning individual and team work:**
Weekly meetings where we will utilize Agile workflow and gitlab issues to plan out team tasks and work.
- 3. Strategies for keeping on task:**
Weekly meetings where we discuss the progress towards the semester objectives. As well as consistent open communication between team members, advisors, and clients via communication paths (Discord, Email, GitLab, Text, etc.).

Consequences for Not Adhering to Team Contract

- 1. How will you handle infractions of any of the obligations of this team contract?**

Team will communicate with the person that is breaking the contract.

2. What will your team do if the infractions continue?

Will escalate to TA/Professor if needed

a) *I participated in formulating the standards, roles, and procedures as stated in this contract.*

b) *I understand that I am obligated to abide by these terms and conditions.*

c) *I understand that if I do not abide by these terms and conditions, I will suffer the consequences as stated in this contract.*

- | | |
|-----------------------------|----------------------|
| 1) <u>Nicholas Erickson</u> | DATE <u>9/8/2023</u> |
| 2) <u>Naga Vempati</u> | DATE <u>9/8/2023</u> |
| 3) <u>Sam DeFrancisco</u> | DATE <u>9/8/2023</u> |
| 4) <u>Nikhil Kuricheti</u> | DATE <u>9/8/2023</u> |
| 5) <u>Jennifer Robles</u> | DATE <u>9/8/2023</u> |
| 6) <u>Brayton Rude</u> | DATE <u>9/8/2023</u> |
| 7) _____ | DATE _____ |
| 8) _____ | DATE _____ |