University of British Columbia Okanagan

COSC 499 – 201

Capstone Software Engineering Project

2023 Summer Term 1&2

Project Topic Number: #6

Title of project:

Gamified Coding Practice Platform

Assignment: Design and Testing document

Draft: June 4th, 2023

Team Members:

Joe Gaspari [Project Manager/Scrum Master]

Archita Gattani [Technical Lead]

Gyumin Moon [Integration Lead]

Jason Ramos [Software Manager]

Alrick Vincent [Client Liaison]

1. Abstract....3
2. Project Description....3
3. Use Case Design.... 4
3.1 User Groups.... 4
3.2 Use Cases....13
3.3 Use Case Descriptions & Scenarios....14
4. System Architecture....32
4.1 ER diagram.... 32
4.3 Dynamic models.... 33
4.3.1 Data flow diagram....33
4.3.2 State Diagram....36
4.3.3 Sequence Diagram.... 37
4.3.4 Activity Diagram.... 38
5. Technical Specifications.... 39
5.1 Tech Stack....39
5.2 Tools....39
6. Testing Strategies....40
6.1 Regression testing.... 40
6.2 Unit testing....40
6.3 Integration testing.... 40
6.4 Usability testing.... 41
6.5 User Acceptance Testing....41
7. User Interface Design.... 42
7.1 UI Mockups.... 42

1. Abstract

The purpose of this document is to provide a detailed description of the design and testing process for a software project. It includes the project description, highlighting the key features and functionality of the software. The use case design section explores user groups, usage scenarios, and presents a use case diagram and specific use cases. The system architecture section includes an ER diagram and dynamic models such as data flow, state, collaboration, sequence, and activity diagrams. The user interface design section presents UI mockups ie. how the website will look like and technical specifications for a user-friendly interface. The testing strategies section covers regression, unit, integration, system, usability, and user acceptance testing, ensuring the software's reliability and quality. Overall, this document serves as a comprehensive guide for the design and testing phases of the software project.

2. Project Description

The Gamified Coding Practice Application project, led by Learnification Technologies, aims to facilitate coding skill learning and practice for students. By leveraging modern technologies, personalization, and gamification, the application strives to create an engaging and efficient learning experience.

To achieve this, the project focuses on developing a dynamic web application that encourages students to practice coding in a gamified environment. The application allows students to select their preferred programming language, enabling them to tailor their learning experience according to their individual needs and interests. The system will allow course instructors to register and build classes that will give students a course portal that contains a guided set of questions that is specific to the course content.

The project utilizes React JS, a powerful JavaScript library, as the foundation for the application's front-end. This ensures a seamless and user-friendly interface for students. The core functionality of the Gamified Coding Practice Application revolves around AI APIs, which generate coding questions customized to the chosen programming language.

To enhance student engagement, the application incorporates gamification elements such as badges, leaderboards, and rewards. By making coding practice enjoyable and interactive, the application encourages students to dedicate more time and effort to their learning journey.

3. Use Case Design

3.1 User Groups

1. Students
2. Instructors
3. Self-learners
4. Administrators

3.2 User Personas

Mia the Student Learner:

Name: Mia Thompson **Age:** 20

Background:

Mia is a 20-year-old first year university student. She has a passion for computer science and wants to pursue a masters degree in software engineering. She enjoys learning new coding concepts and wants to strengthen her problem-solving skills. Mia is a motivated and diligent student, always looking for opportunities to challenge herself and improve her coding abilities.

Goals:

1.) *Improve coding skills:* Mia wants to enhance her coding skills by practicing different programming concepts and solving coding problems regularly.

2.) *Enhance problem-solving abilities:* Mia aims to strengthen her problem-solving skills and learn how to approach complex coding challenges effectively.

3.) Excel at course content & exams: Mia knows the courses she is taking can be quite difficult to learn without the proper resources. She wants the ability to sharpen her knowledge base so she is able to excel during exam season.

Pain Points:

1.) *Limited availability of coding practice resources:* Mia often struggles to find coding problems and exercises that are both challenging and engaging.

2.) *Lack of personalized feedback:* Mia finds it difficult to assess her progress and identify areas for improvement without detailed feedback on her coding solutions.

3.) *Monotonous learning experience:* Mia often feels bored with traditional learning methods and wants a more interactive and engaging way to learn coding concepts.

4.) *Lack of structured learning path:* Without a structured curriculum, Mia finds it challenging to navigate the vast landscape of coding and prioritize their learning goals effectively.

Tech Savviness:

Mia is comfortable using technology and has basic proficiency in computer skills. She regularly uses her laptop and smartphone for various activities such as researching coding topics, watching online tutorials, and completing coding assignments.

Personality Traits:

1.) *Motivated:* Mia is highly motivated and always seeks opportunities to learn and grow.

2.) *Persistent:* She doesn't easily give up on challenging problems and is willing to put in the effort to find solutions.

3.) *Curious:* Mia is naturally curious and enjoys exploring new concepts and technologies.

4.) *Competitive:* Mia enjoys challenges and likes to compare her progress with others, pushing herself to achieve higher goals.

5.) *Collaborative:* Mia is open to collaboration and enjoys working with peers to solve problems and learn from each other.

User Needs:

1.) *Engaging learning platform:* Mia is looking for a web application that offers a gamified learning experience to keep her engaged and motivated.

2.) *AI-generated coding problems:* Mia wants access to a variety of coding problems that are generated by AI, ensuring a continuous supply of fresh challenges.
3.) *Progress tracking and personalized feedback:* Mia seeks a platform that tracks her progress and provides personalized feedback on her coding solutions, helping her identify areas for improvement.
4.) *Interactive and intuitive interface:* Mia prefers a user-friendly interface that allows her to easily navigate through coding problems, access learning materials, and track her progress.
5.) *Social features:* Mia would appreciate social features like leaderboards or the ability to share achievements with peers, fostering a sense of healthy competition and community.

Using this student user persona as a reference, you can design and develop a web application that offers AI-generated coding problems related to course curriculum in a game format to help students like Mia learn coding effectively while staying motivated and engaged.

Sam the Free Learner:
Name: Sam Wilson **Age:** 25

Background:

Sam is a self-motivated individual who is passionate about learning to code. They have a full-time job unrelated to programming but want to explore coding as a hobby or potential career path. Sam is driven by curiosity and enjoys the flexibility of learning at their own pace, focusing on specific coding topics that interest them the most.

Goals:

1.) *Explore coding topics:* Sam wants to explore a wide range of coding topics, including programming languages, algorithms, and software development concepts, without the constraints of a structured curriculum.
2.) *Flexible learning experience:* Sam seeks a flexible learning experience that accommodates their busy schedule, allowing them to learn and practice coding at their own convenience.

3.) *Improve problem-solving skills:* Sam aims to enhance their problem-solving abilities and develop a logical approach to coding challenges.

4.) *Stay updated with industry trends:* Sam wants to stay informed about the latest trends and advancements in the coding and software development industry.

Pain Points:

1.) *Limited access to quality learning resources:* Sam struggles to find reliable and comprehensive learning resources that cover a wide range of coding topics in an accessible manner.

2.) *Difficulty in tracking progress:* Sam feels the need for a system to track their progress and receive feedback on their coding solutions to gauge their improvement and identify areas for growth.

Tech Savviness:

Sam is comfortable using technology and has basic computer skills. They are familiar with online learning platforms and have experience navigating web applications.

Personality Traits:

1.) *Self-motivated:* Sam is highly self-driven and proactive in pursuing their coding learning journey.

2.) *Curious:* Sam has a natural curiosity and enjoys exploring various coding topics and learning new concepts.

3.) *Independent:* Sam prefers to learn independently and at their own pace, without relying on structured courses or specific deadlines.

4.) *Resourceful:* Sam actively seeks out various resources, such as tutorials, documentation, and online communities, to enhance their coding knowledge.

5.) *Adaptable:* Sam is adaptable and open to experimenting with different coding languages and technologies.

User Needs:

1.) *Accessible learning platform:* Sam needs a web application that provides a diverse range of coding topics and exercises in an easily accessible and user-friendly format.
2.) *Vast coding content library:* The web application should offer a comprehensive library of coding topics, programming languages, algorithms, and software development concepts to cater to Sam's diverse interests.
3.) *Progress tracking and feedback:* Sam seeks a system that tracks their progress, provides feedback on their coding solutions, and offers recommendations for further learning based on their performance.
4.) *Engaging and interactive experience:* Sam prefers an interactive and engaging learning experience that includes coding challenges, quizzes, and interactive coding environments to practice and experiment with code.

By considering the needs of a free learner like Sam, you can design and develop a web application that caters to their self-directed learning style. The application should offer a wide range of coding topics, provide progress tracking and feedback mechanisms, and foster an engaging and flexible learning experience.

Alex the Instructor:
Name: Alex Johnson **Age:** 35

Background:

Alex is a seasoned computer science instructor with 10 years of teaching experience at a high school. They are passionate about computer science education and strive to provide their students with the best learning opportunities. Alex believes that interactive and hands-on learning experiences can significantly enhance student engagement and comprehension. They are always on the lookout for innovative teaching tools and methods to create a dynamic learning environment.

Goals:

1.) *Enhance student learning outcomes:* Alex aims to improve their students' understanding of coding concepts and problem-solving skills by offering interactive and engaging learning experiences.
2.) *Align curriculum with the web application:* Alex wants to integrate the web application into their existing curriculum, tailoring coding questions to match the topics covered in class.
3.) *Monitor student progress:* Alex seeks to track and analyze their students' progress to identify areas where additional support may be required.
4.) *Save time in creating coding questions:* Alex wants to save time and effort by utilizing the web application's course-building functionality to create tailored coding questions aligned with the curriculum.

Pain Points:

1.) *Limited interactive learning resources:* Alex finds it challenging to discover interactive learning resources that align with their curriculum and provide students with hands-on coding experiences.
2.) *Time-consuming question creation:* Alex spends a significant amount of time creating coding questions that are tailored to their curriculum and align with the difficulty level of their students.
3.) *Lack of comprehensive progress tracking:* Alex struggles to get a holistic view of each student's progress and identify specific areas where they might be struggling.

Tech Savviness:

Alex is proficient in using technology and regularly incorporates educational software and tools into their teaching practice. They have experience using learning management systems and online platforms to deliver content and assess student performance.

Personality Traits:

1.) *Dedicated:* Alex is dedicated to their profession and constantly strives to improve their teaching methods and student outcomes.

2.) *Detail-oriented:* Alex pays close attention to the specific needs and requirements of their students and curriculum.
3.) *Resourceful:* Alex is skilled at finding innovative solutions and tools to enhance student engagement and learning.
4.) *Analytical:* Alex values data-driven insights and uses them to inform instructional decisions and adapt teaching strategies to meet student needs.
5.) *Collaborative:* Alex enjoys collaborating with colleagues and staying up-to-date with the latest teaching practices and technologies.

User Needs:

1.) *Course-building functionality:* Alex needs a web application that allows them to create and customize courses or modules aligned with their curriculum. This functionality should enable them to add coding questions tailored to specific topics and learning objectives.
2.) *Question bank management:* Alex requires a comprehensive question bank management system that allows them to organize and reuse coding questions efficiently.
3.) *Progress tracking and analytics:* The web application should provide detailed analytics and progress tracking features to help Alex monitor individual student performance, identify areas for improvement, and adapt instruction accordingly.
4.) *User-friendly interface:* Alex prefers an intuitive and user-friendly interface that simplifies the process of creating courses, managing questions, and accessing student progress reports.

By considering the needs and requirements of an instructor like Alex, you can design and develop a web application that not only caters to student learning but also provides instructors with the tools and functionalities necessary to create tailored courses aligned with their curriculum and monitor student progress effectively.

Alex the Admin User:

Name: Alex Mitchell **Age:** 40

Background:

Alex is an experienced administrator with a background in educational management. They work in a role that involves overseeing multiple educational institutions and ensuring smooth operations. Alex recognizes the importance of technology in education and seeks to implement efficient systems that support instructors and students in their learning journey.

Goals:

1.) *Efficient institution management:* Alex aims to manage multiple educational institutions effectively, streamlining administrative processes, and ensuring optimal performance.
2.) *Support instructors and students:* Alex wants to provide instructors with the necessary tools and resources to create and deliver high-quality course content, while also ensuring a positive learning experience for the students.
3.) *Data monitoring and analysis:* Alex seeks to monitor data coming from each institution, track key performance indicators, and analyze trends to make informed decisions and identify areas for improvement.
4.) *User management:* Alex wants to have control over user accounts, including the ability to add new instructors, modify user details, and manage access permissions.

Pain Points:

1.) *Manual administrative tasks:* Alex finds it time-consuming to perform manual administrative tasks, such as user management and institution setup, without a centralized system.
2.) *Limited oversight:* Without a comprehensive monitoring system, Alex struggles to gather and analyze data from each institution, hindering their ability to make data-driven decisions.
3.) *Inefficient user management:* Alex faces challenges in managing user accounts, adding and removing instructors, and granting appropriate permissions across multiple institutions.

Tech Savviness:

Alex is proficient in using technology and has experience working with administrative systems, learning management platforms, and data analysis tools. They are comfortable navigating complex software systems and have a good understanding of user management principles.

Personality Traits:

1.) *Organized:* Alex is highly organized and pays attention to detail, ensuring that administrative tasks and processes are executed accurately.

2.) *Analytical:* Alex has a strong analytical mindset, using data to drive decision-making and improve operational efficiency.

3.) *Problem-solver:* Alex is proactive in finding solutions to challenges that arise in managing multiple educational institutions.

4.) *Collaborative:* Alex believes in fostering collaboration among instructors, administrators, and students to create a positive learning environment.

User Needs:

1.) *Institution management:* Alex requires a web-based system that allows them to add, modify, and manage multiple educational institutions, including administrative details and settings.

2.) *User management:* The system should provide comprehensive user management capabilities, allowing Alex to add, modify, and remove instructors, and manage user permissions and access levels across institutions.

3.) *Institution code generation:* Alex needs a functionality that generates unique institution codes that can be used by instructors to create and manage course content under their respective institutions.

4.) *Data monitoring and analysis:* The system should offer data monitoring and reporting features, allowing Alex to track and analyze key performance indicators, student progress, and overall system usage across institutions.

5.) *Access control and security:* The system should prioritize security measures to protect user data and ensure appropriate access controls are in place for administrators, instructors, and students.

By considering the needs of an admin user like Alex, you can design and develop a web-based system that allows for efficient institution management, user management, data monitoring, and analysis. The system should provide the necessary tools and features to streamline administrative processes and support instructors and students in their respective roles within each institution.

3.2 Use Cases

(001) Student Registration

(002) Instructor Registration

(003) Student Login

(004) Instructor Login

(005) Reset password

(006) Instructor create/manage class content (questions/topics/students)

(007) Instructors delete class

(008) Students drop a class

(009) Student joining a class using code

(010) Learners choosing a language and proficiency

(011) Learners choosing a topic and get an AI generated question

(012) Student and Instructor view Class Leaderboard

(013) View Own User Profile

(014) Edit own user dashboard

(015) View Public Users Dashboard

(016) Instructor can view class and user analytics

(017) Edit user profile

(018) Instructor generating question

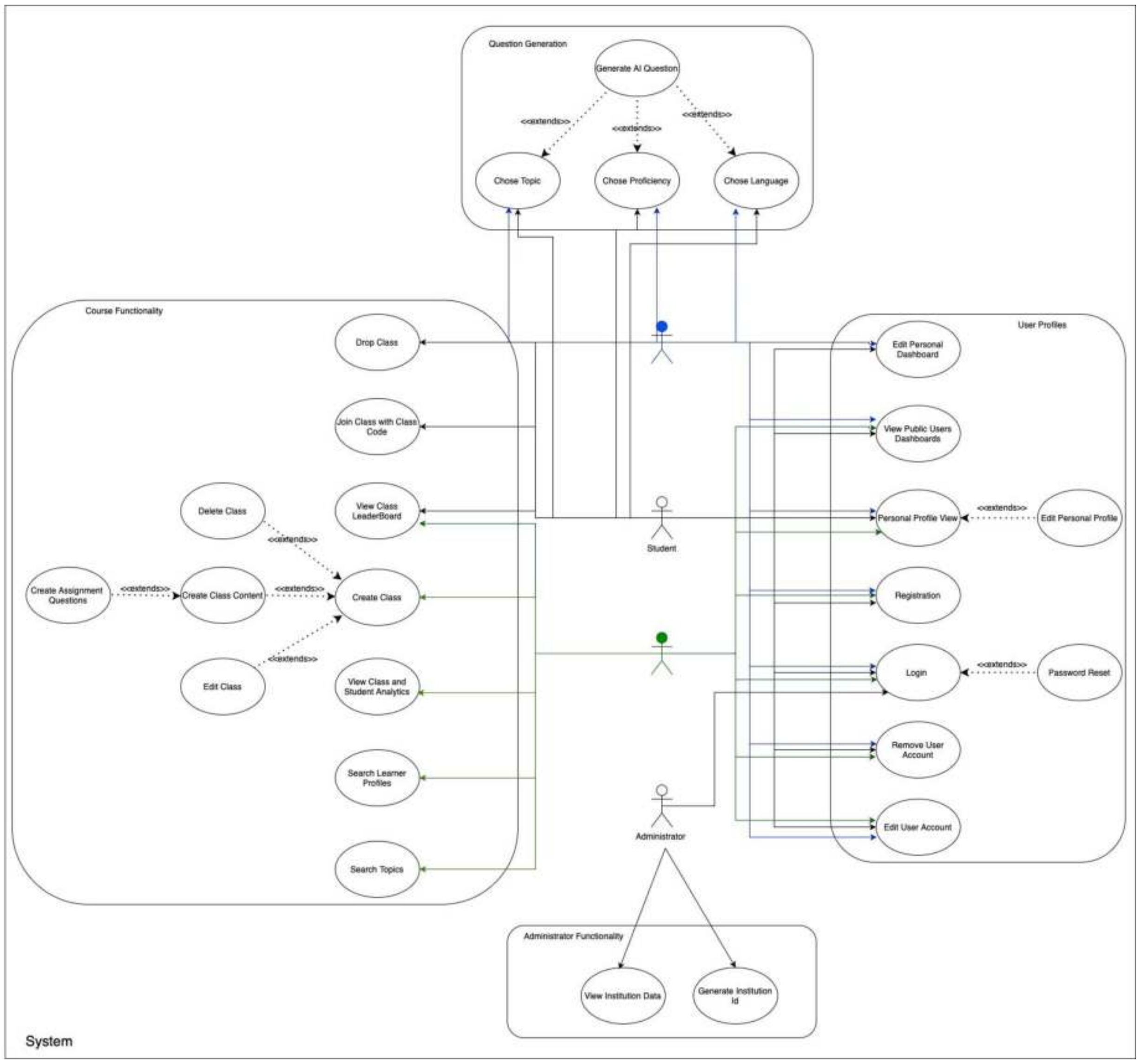
(019) Search topics

(020) Search learner profiles

(021) Admin can remove / edit users and courses

(022) Admin can view all data from institution machines

(023) Admin can generate Educational Organization Id. <u>(this will be used when an instructor creates an account)</u>



3.3 Use Case Descriptions & Scenarios

(001) Student Registration:

Primary actor: Student

Preconditions: None

Postconditions: Student successfully registers an account in the system.

Main Scenario:

1. Student accesses the registration page.
2. Student provides the required information, such as name, email address, and password.
3. Student submits the registration form.
4. The system validates the provided information.
5. If the information is valid, the system creates a new student account.
6. The system sends a verification email to the student's provided email address.
7. Student receives the verification email and clicks on the verification link.
8. The system verifies the email and activates the student's account.
9. Student is redirected to the login page and can now log in with their registered credentials.

Extensions:

4a. If the provided information is invalid or incomplete, the system displays an error message and prompts the student to provide valid information.

6a. If the verification email is not received, the student can request a new verification email or contact support for assistance.

(002) Instructor Registration:

Primary actor: Instructor

Preconditions: None

Postconditions: Instructor successfully registers an account in the system.

Main Scenario:

1. Instructor accesses the registration page.
2. Instructor provides the required information, such as name, email address, and password.
3. Instructor submits the registration form.
4. The system validates the provided information.
5. If the information is valid, the system creates a new instructor account.
6. The system sends a verification email to the instructor's provided email address.
7. Instructor receives the verification email and clicks on the verification link.

8. to the login page and can now log in with their registered credentials.

Extensions:
4a. If the provided information is invalid or incomplete, the system displays an error message and prompts the instructor to provide valid information.
6a. If the verification email is not received, the instructor can request a new verification email or contact support for assistance.

(003) Student Login:
Primary actor: Student
Preconditions: Student has a registered account in the system.
Postconditions: Student successfully logs into the system.

Main Scenario:

1. Student accesses the login page.
2. Student enters their email address and password.
3. Student submits the login form.
4. The system validates the provided credentials.
5. If the credentials are valid, the system authenticates the student and grants access to their account.
6. Student is redirected to their personalized dashboard or the main application interface.

Extensions:
4a. If the provided credentials are incorrect or do not match any registered accounts, the system displays an error message and prompts the student to enter valid credentials.
4b. If the student's account is not yet activated, the system displays a message indicating that the account needs to be verified.

(004) Instructor Login:
Primary actor: Instructor
Preconditions: Instructor has a registered account in the system.

Postconditions: Instructor successfully logs into the system.

Main Scenario:

1. Instructor accesses the login page.
2. Instructor enters their email address and password.
3. Instructor submits the login form.
4. The system validates the provided credentials.
5. If the credentials are valid, the system authenticates the instructor and grants access to their account.
6. Instructor is redirected to their personalized dashboard or the main application interface.

Extensions:

4a. If the provided credentials are incorrect or do not match any registered accounts, the system displays an error message and prompts the instructor to enter valid credentials.

4b. If the instructor's account is not yet activated, the system displays a message indicating that the account needs to be verified.

(005) Reset Password:

Primary actor: User (Student or Instructor)

Preconditions: User has a registered account in the system.

Postconditions: User's password is successfully reset.

Main Scenario:

1. User accesses the "Forgot Password" page.
2. User enters their registered email address.
3. User submits the form to request a password reset.
4. The system verifies the provided email address.
5. If the email address is valid, the system sends a password reset link to the user's email.
6. User receives the password reset email and clicks on the reset link.
7. User is redirected to a password reset page.
8. User enters a new password and confirms it.

9. User submits the form to reset the password.
10. The system validates and updates the user's password.
11. User receives a confirmation message that their password has been successfully reset.
12. User can now log in with the new password.

Extensions:
4a. If the provided email address is not found in the system, the system displays an error message indicating that the email address is not registered.
10a. If the new password does not meet the system's password requirements, the system displays an error message and prompts the user to choose a valid password.

(006) Instructor Create/Manage Class Content:
Primary actor: Instructor
Preconditions: Instructor is logged into their account and has access to their dashboard.
Postconditions: Class content (questions, topics, students) is created or managed by the instructor.

Main Scenario:
1. Instructor accesses their dashboard or the class management interface.
2. Instructor selects the option to create new class content.
3. Instructor provides the necessary details, such as class name, topic, and questions.
4. Instructor submits the form to create the class content.
5. The system validates the provided information.
6. If the information is valid, the system creates the class content and associates it with the instructor's class.
7. Instructor can view and manage the created class content, such as adding or editing questions, topics, or students.

Extensions:
5a. If the provided information is incomplete or invalid, the system displays an error message and prompts the instructor to provide valid information.

6a. If there are any errors during the creation of class content, the system displays an error message and informs the instructor about the issue.

(007) Instructors Delete Class:
Primary actor: Instructor
Preconditions: Instructor is logged into their account and has access to their dashboard.
Postconditions: The specified class is successfully deleted from the system.

Main Scenario:

1. Instructor accesses their dashboard or the class management interface.
2. Instructor selects the option to delete a class.
3. Instructor chooses the class they want to delete from the available list.
4. Instructor confirms the deletion action.
5. The system removes the specified class from the system and associated data, such as class content and enrolled students.
6. Instructor receives a confirmation message that the class has been successfully deleted.

Extensions:
3a. If the instructor does not have the necessary permissions to delete the class, the system displays an error message indicating the lack of authorization.

(008) Students Drop a Class:
Primary actor: Student
Preconditions: Student is logged into their account and has enrolled in at least one class.
Postconditions: The student is successfully unenrolled from the specified class.

Main Scenario:

1. Student accesses their dashboard or the class enrollment interface.
2. Student selects the option to drop a class.
3. Student chooses the class they want to drop from the available list of enrolled classes.
4. Student confirms the drop action.

5. The system removes the student's enrollment from the specified class.
6. Student receives a confirmation message that they have been successfully unenrolled from the class.

Extensions:

3a. If the student is not currently enrolled in any classes, the system displays an error message indicating that the student does not have any classes to drop.

3b. If the student does not have the necessary permissions to drop the class, the system displays an error message indicating the lack of authorization.

(009) Student Joining a Class Using Code:

Primary actor: Student

Preconditions: Student is logged into their account and has access to the class enrollment interface.

Postconditions: Student successfully joins the specified class using the provided class code.

Main Scenario:

1. Student accesses the class enrollment interface.
2. Student selects the option to join a class using a code.
3. Student enters the class code provided by the instructor.
4. Student submits the code.
5. The system validates the entered code.
6. If the code is valid and the class is open for enrollment, the system adds the student to the class.
7. Student receives a confirmation message that they have successfully joined the class.

Extensions:

5a. If the entered code is invalid or does not correspond to any available classes, the system displays an error message indicating an invalid code.

6a. If the class is not open for enrollment or the maximum capacity has been reached, the system displays a message indicating that the student cannot join the class at the moment.

(010) Learners Choosing a Language and Proficiency:

Primary actor: Student

Preconditions: Student is logged into their account and has access to language and proficiency selection options.

Postconditions: Student's chosen language and proficiency level are saved in their profile.

Main Scenario:

1. Student accesses the language and proficiency selection options.
2. Student chooses a programming language from the available options.
3. Student selects their proficiency level in the chosen language.
4. Student submits the language and proficiency selection.
5. The system saves the chosen language and proficiency level in the student's profile.
6. Student receives a confirmation message that their language and proficiency selection has been successfully saved.

Extensions:

2a. If the chosen programming language is not available in the system, the system displays an error message indicating an invalid language selection.

3a. If the chosen proficiency level is not within the valid range, the system displays an error message indicating an invalid proficiency selection.

(011) Learners Choosing a Topic and Getting an AI-Generated Question:

Primary actor: Student

Preconditions: Student is logged into their account and has access to topic selection and question generation options.

Postconditions: Student receives an AI-generated question based on their chosen topic.

Main Scenario:

1. Student accesses the topic selection options.
2. Student chooses a topic of interest from the available options.
3. Student submits the topic selection.

4. The system generates an AI-generated question based on the chosen topic.
5. The system presents the generated question to the student.
6. Student receives the AI-generated question for their chosen topic.

Extensions:

2a. If the chosen topic is not available in the system, the system displays an error message indicating an invalid topic selection.

4a. If there are any issues with the AI question generation process, the system displays an error message indicating the inability to generate a question.

(012) Student and Instructor View Class Leaderboard:

Primary actor: Student, Instructor

Preconditions: Student or Instructor is logged into their respective accounts and has access to the class leaderboard.

Postconditions: Student or Instructor successfully views the class leaderboard with relevant rankings.

Main Scenario:

1. Student or Instructor navigates to the class leaderboard section.
2. The system retrieves the necessary data to generate the class leaderboard.
3. The system ranks the students based on their performance metrics, such as points or completion rate.
4. The system displays the class leaderboard to the Student or Instructor, showing the rankings of students in the class.
5. Student or Instructor can view their own position on the leaderboard and the positions of other students.

Extensions:

4a. If there are no students enrolled in the class or no performance metrics available, the system displays a message indicating the absence of leaderboard data.

(013) View Own User Profile:

Primary actor: Student or Instructor

Preconditions: Student or Instructor is logged into their respective accounts and has access to their user profile.

Postconditions: Student or Instructor successfully views their own user profile information.

Main Scenario:

1. Student or Instructor accesses the user profile section.
2. The system retrieves the user profile data for the Student or Instructor.
3. The system displays the user profile information, including personal details, preferences, and achievements.
4. Student or Instructor can view and review their own user profile.

Extensions:

2a. If there is no user profile data available for the Student or Instructor, the system displays a message indicating the absence of profile information.

3a. If there are any restrictions or permissions applied to the user profile, the system displays the relevant information based on the Student or Instructor's access level.

(014) Edit Own User Dashboard:

Primary actor: Student or Instructor

Preconditions: Student or Instructor is logged into their respective accounts and has access to their user dashboard.

Postconditions: Student or Instructor successfully edits their own user dashboard by customizing the displayed information or layout.

Main Scenario:

1. Student or Instructor accesses their user dashboard.
2. The system presents the current layout and information displayed on the user dashboard.
3. Student or Instructor selects the option to edit their user dashboard.

4. Student or Instructor makes changes to the layout, such as rearranging widgets or adding/removing information.
5. Student or Instructor saves the changes made to their user dashboard.
6. The system updates the user dashboard according to the edited layout and information.
7. Student or Instructor receives a confirmation message that their user dashboard has been successfully edited.

Extensions:
3a. If there are any restrictions or permissions applied to editing the user dashboard, the system displays an error message indicating the lack of authorization.
4a. If there are limitations on the customization options or layout configurations, the system displays a message indicating the available customization features.

(015) View Public Users Dashboard:
Primary actor: Student or Instructor
Preconditions: Student or Instructor is logged into their respective accounts and has access to view public user dashboards.
Postconditions: Student or Instructor successfully views the dashboard of a public user.

Main Scenario:

1. Student or Instructor navigates to the public user dashboard section.
2. Student or Instructor selects a specific public user whose dashboard they want to view.
3. The system retrieves the necessary data and layout configurations of the selected public user's dashboard.
4. The system displays the public user's dashboard, showing the information and layout as configured by the public user.
5. Student or Instructor can view the public user's dashboard, including the displayed information and layout.

Extensions:

2a. If the selected public user's dashboard is set to private or not available for public viewing, the system displays an error message indicating the lack of accessibility.

(016) Instructor Can View Class and User Analytics:

Primary actor: Instructor

Preconditions: Instructor is logged into their account and has access to class and user analytics.

Postconditions: Instructor successfully views the analytics related to classes and users.

Main Scenario:

1. Instructor accesses the class and user analytics section.
2. Instructor selects the specific class or user for which they want to view analytics.
3. The system retrieves the necessary data and generates analytics based on the selected class or user.
4. The system presents the analytics to the Instructor, displaying relevant information such as performance metrics, progress, or engagement levels.
5. Instructor can analyze and interpret the analytics data to gain insights into class or user performance.

Extensions:

2a. If there are no available analytics data for the selected class or user, the system displays a message indicating the absence of analytics information.

(017) Edit User Profile:

Primary actor: Student or Instructor

Preconditions: Student or Instructor is logged into their respective accounts and has access to edit their user profile.

Postconditions: Student or Instructor successfully edits their own user profile by modifying personal information, preferences, or settings.

Main Scenario:

1. Student or Instructor accesses their user profile section.
2. The system presents the current user profile information and settings.
3. Student or Instructor selects the option to edit their user profile.
4. Student or Instructor makes changes to the personal information, preferences, or settings as desired.
5. Student or Instructor saves the changes made to their user profile.
6. The system validates and updates the user profile with the edited information and settings.
7. Student or Instructor receives a confirmation message that their user profile has been successfully edited.

Extensions:
3a. If there are any restrictions or permissions applied to editing the user profile, the system displays an error message indicating the lack of authorization.
4a. If there are limitations on the allowed modifications to certain fields or settings, the system displays a message indicating the available editing options.

(018) Instructor Generating Question:
Primary actor: Instructor
Preconditions: Instructor is logged into their account and has access to the question generation functionality.
Postconditions: Instructor successfully generates a coding question for a specific topic or programming language.

Main Scenario:
1. Instructor accesses the question generation interface.
2. Instructor selects the desired topic or programming language for the coding question.
3. Instructor specifies any additional parameters or requirements for the question, such as difficulty level or constraints.
4. Instructor initiates the question generation process.

5. The system utilizes the AI-based question generation algorithm to generate a coding question based on the specified parameters.
6. The system presents the generated question to the Instructor, displaying the problem statement, input/output requirements, and any additional instructions.
7. Instructor reviews the generated question and makes any necessary edits or modifications.
8. Instructor finalizes the question and saves it to the question bank or assigns it to a specific class or student.

Extensions:

2a. If the desired topic or programming language is not available in the system, the system displays an error message indicating an invalid selection.

5a. If there are any issues with the question generation process, such as generating an incomplete or invalid question, the system displays an error message indicating the inability to generate the question.

(019) Search Topics:

Primary actor: Student or Instructor

Preconditions: Student or Instructor is logged into their respective accounts and has access to the topic search functionality.

Postconditions: Student or Instructor successfully searches for specific coding topics.

Main Scenario:

1. Student or Instructor accesses the topic search interface.
2. Student or Instructor enters the desired keyword or topic in the search bar.
3. Student or Instructor initiates the search.
4. The system performs a search based on the entered keyword or topic, filtering the available coding topics.
5. The system presents the search results, displaying a list of coding topics that match or are related to the entered keyword.

6. Student or Instructor can browse and select the desired coding topic from the search results.

Extensions:
4a. If there are no coding topics available in the system or no matching results for the entered keyword, the system displays a message indicating the absence of search results.

(020) Search Learner Profiles:
Primary actor: Instructor
Preconditions: Instructor is logged into their account and has access to the learner profile search functionality.
Postconditions: Instructor successfully searches for and views learner profiles based on specific criteria or filters.

Main Scenario:

1. Instructor accesses the learner profile search interface.
2. Instructor specifies the desired search criteria or filters, such as programming language proficiency or completion rate.
3. Instructor initiates the search.
4. The system performs a search based on the specified criteria, filtering the available learner profiles.
5. The system presents the search results, displaying a list of learner profiles that match the specified criteria.
6. Instructor can browse and select a specific learner profile from the search results.
7. The system displays the selected learner profile, showing relevant information such as programming language proficiency, completion rate, or achievements.

Extensions:
4a. If there are no learner profiles available in the system or no matching results for the specified criteria, the system displays a message indicating the absence of search results.

(021) Admin can remove/edit users and courses:

Primary actor: Admin

Preconditions: Admin must be logged into the system. Admin must have appropriate permissions to remove/edit users and courses.

Postconditions: Users and courses are successfully removed/edited.

Main Scenario:

1. Admin selects the option to remove/edit users or courses.
2. The system presents the list of available users or courses.
3. Admin selects a specific user or course to remove/edit.
4. The system verifies the admin's permissions to perform the action.
5. If the admin has permission, the system allows the removal/editing process to proceed.
6. Admin provides the necessary information for removal/editing (e.g., user ID, course details).
7. The system validates the provided information and performs the removal/editing action.
8. The system updates the user or course records accordingly.
9. The system notifies the admin of the successful removal/editing.

Extensions:

If the admin does not have appropriate permissions:

- The system displays an error message indicating insufficient privileges.
- The removal/editing process is aborted.

If the provided information is invalid or incomplete:

- The system prompts the admin to provide valid and complete information.
- The admin corrects the information and retries the removal/editing process.

If the removal/editing action fails:

- The system displays an error message indicating the reason for the failure.
- The admin can try again or contact support for assistance.

(022) Admin can view all data from institution machines:

Primary actor: Admin

Preconditions: Admin must be logged into the system. Admin must have appropriate permissions to access institution machine data.

Postconditions: Admin successfully views the data from institution machines.

Main Scenario:

1. Admin selects the option to view institution machine data.
2. The system retrieves the list of machines associated with the institution.
3. Admin selects a specific machine to view its data.
4. The system verifies the admin's permissions to access machine data.
5. If the admin has permission, the system presents the data of the selected machine.
6. Admin reviews the machine data, which may include information such as usage statistics, status, configurations, etc.
7. The system provides appropriate controls or filters to navigate and analyze the machine data.
8. Admin can perform actions based on the machine data, such as generating reports, troubleshooting, or making informed decisions.
9. Admin can navigate back to the list of institution machines to view data of other machines.

Extensions:

If the admin does not have appropriate permissions:

- The system displays an error message indicating insufficient privileges.
- The access to institution machine data is denied.

If there are no institution machines:

- The system displays a message indicating that there are no machines associated with the institution.
- Admin is informed that there is no data available to view.

(023) Admin can generate Educational Organization Id:

Primary actor: Admin

Preconditions: Admin must be logged into the system. Admin must have appropriate permissions to generate Educational Organization Id.

Postconditions: Educational Organization Id is successfully generated.

Main Scenario:

1. Admin selects the option to generate Educational Organization Id.
2. The system prompts the admin to provide necessary information or configurations for generating the Id (e.g., organization name, location, unique identifiers).
3. Admin provides the required information.
4. The system validates the provided information.
5. If the information is valid, the system generates the Educational Organization Id.
6. The system assigns the generated Id to the specified educational organization.
7. The system displays the generated Id to the admin.
8. Admin can copy, download, or share the generated Id as needed.

Extensions:

If the provided information is invalid or incomplete:

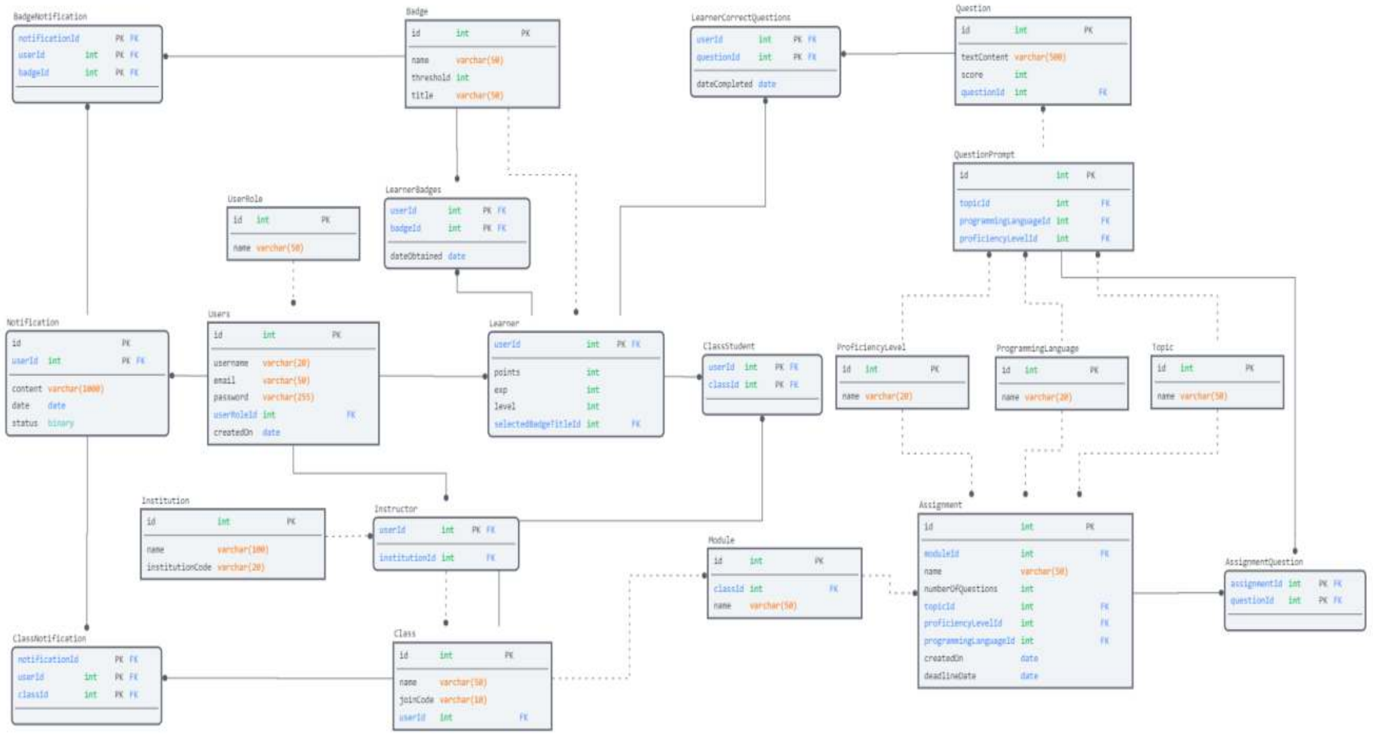
- The system prompts the admin to provide valid and complete information.
- The admin corrects the information and retries the generation process.

If the generation process fails:

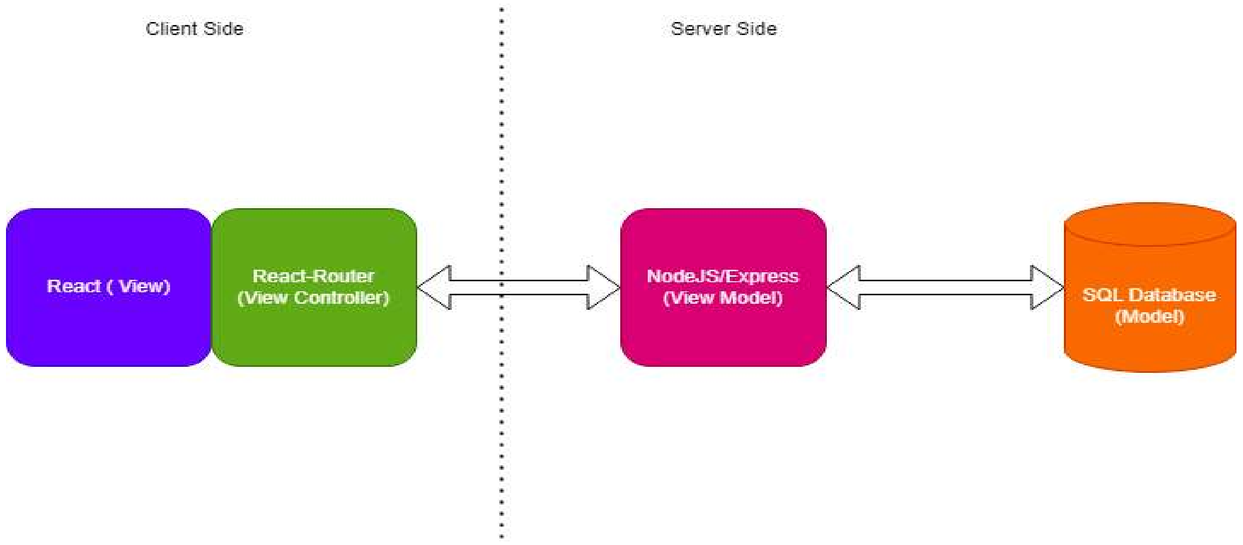
- The system displays an error message indicating the reason for the failure.
- The admin can try again or contact support for assistance.

4. System Architecture

4.1 ER diagram



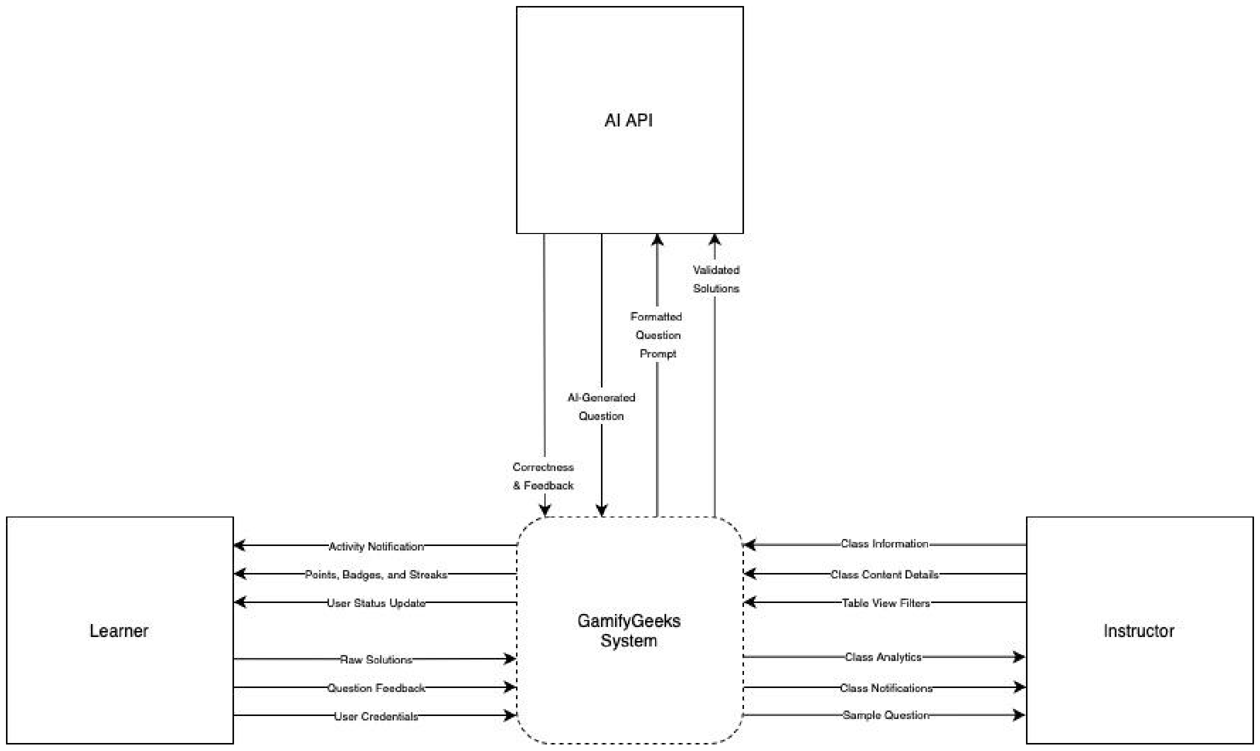
4.2 Architecture Diagram



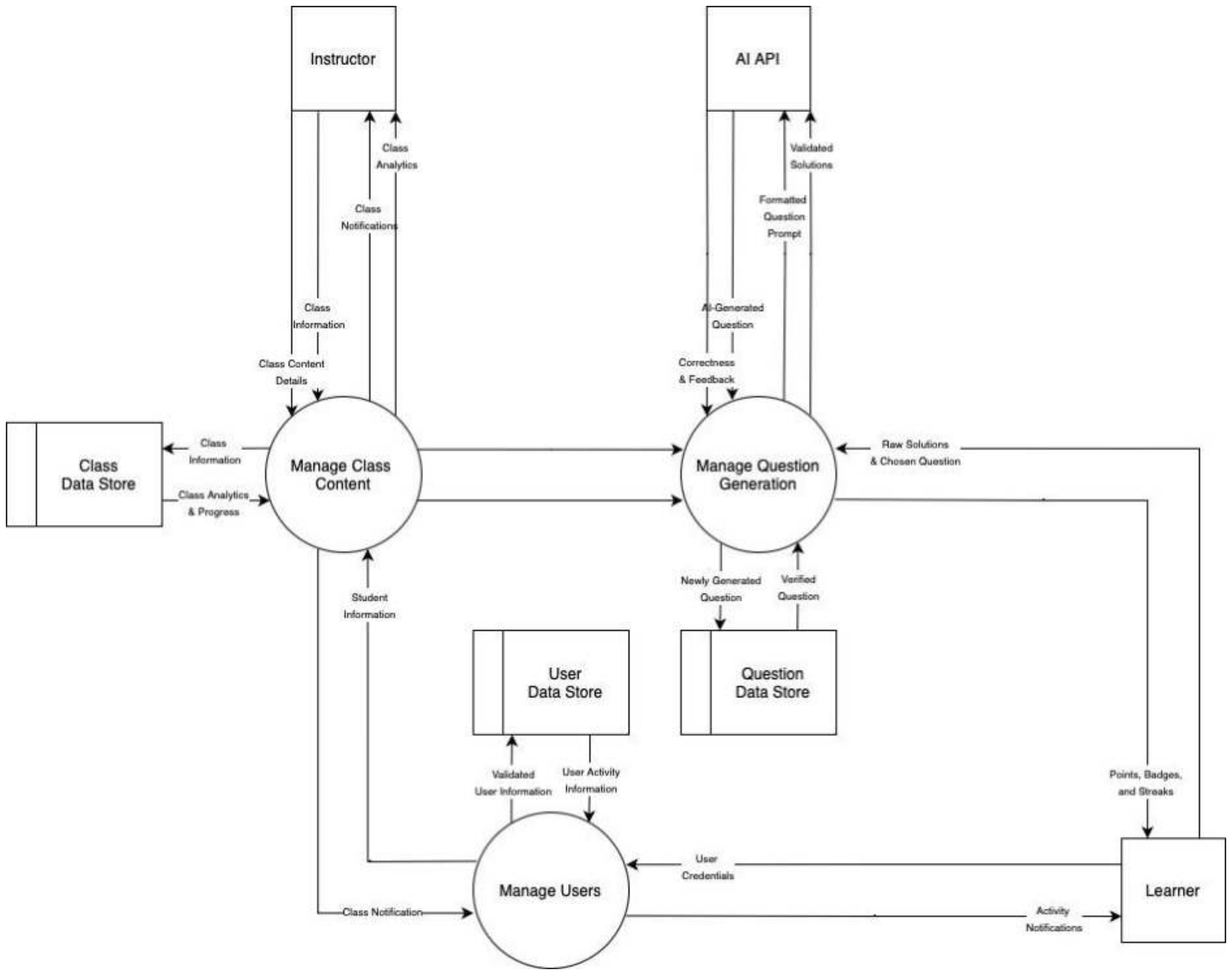
4.3 Dynamic models

4.3.1 Data flow diagram

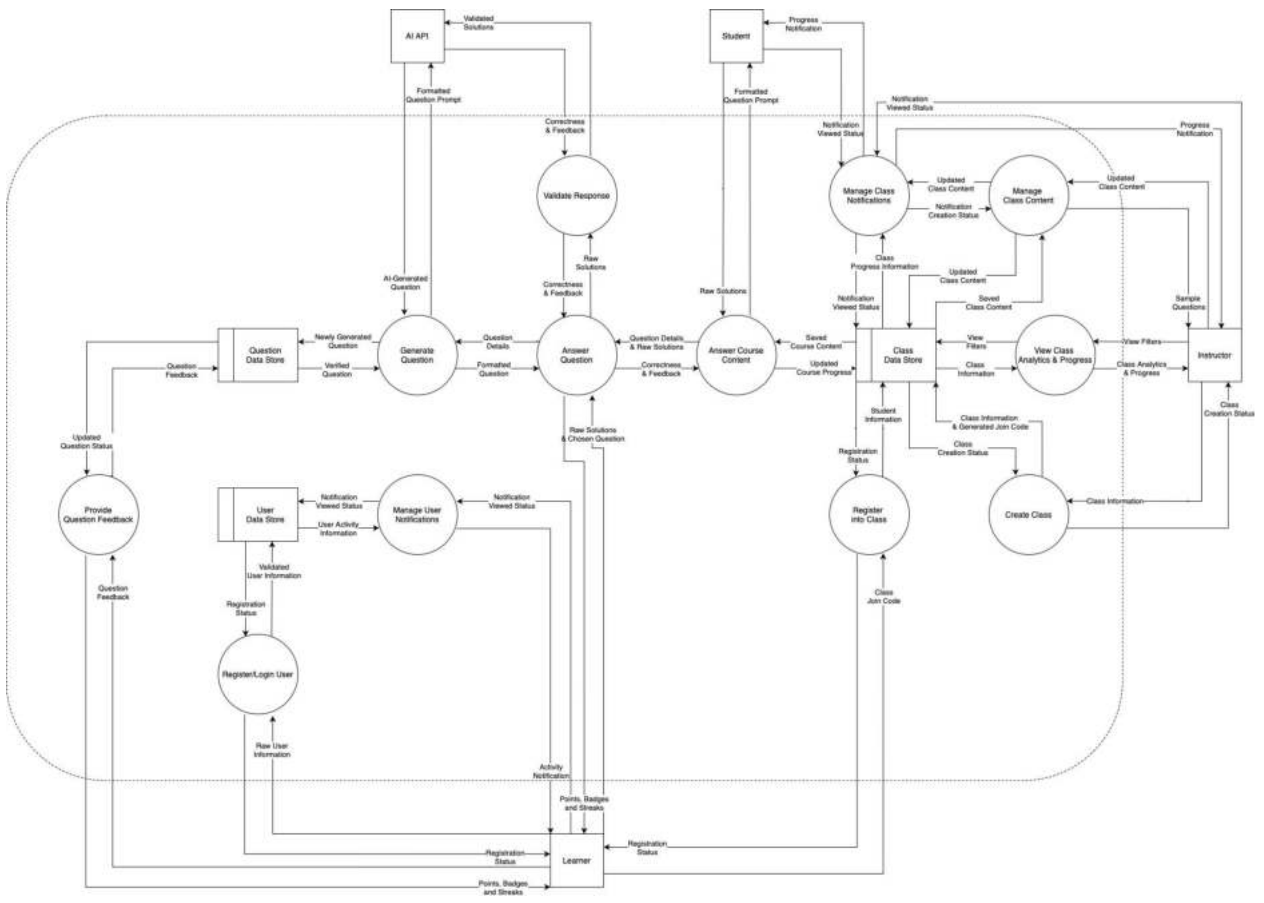
Level 0:



Level 1:

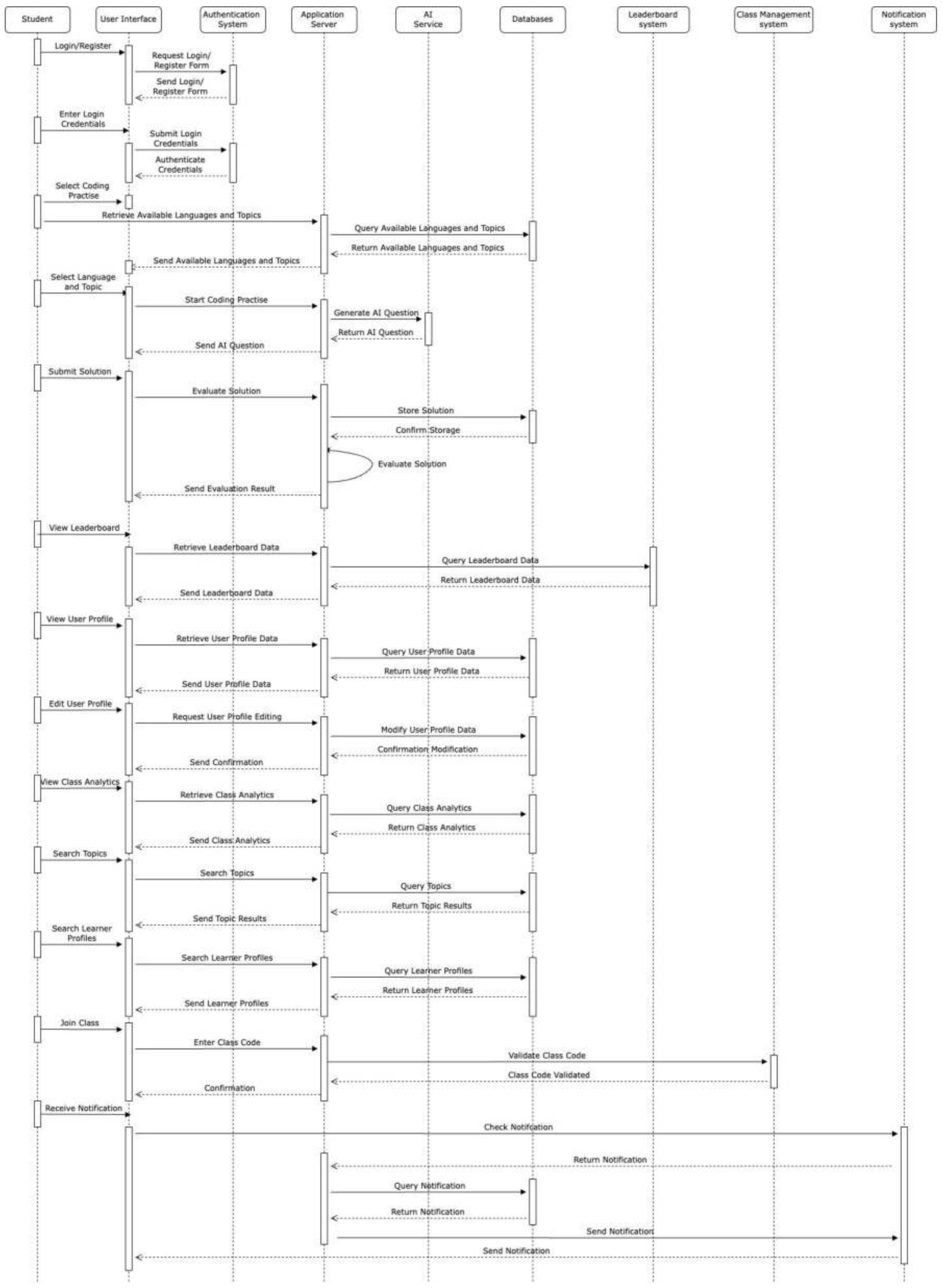

Instructor
AI API
Class Analytics
Class Notifications
Class Information
Class Content Details
Validated Solutions
Formatted Question Prompt
AI-Generated Question
Correctness & Feedback
Class Data Store
Class Information
Class Analytics & Progress
Manage Class Content
Manage Question Generation
Raw Solutions & Chosen Question
Student Information
Newly Generated Question
Verified Question
User Data Store
Question Data Store
Validated User Information
User Activity Information
Points, Badges, and Streaks
User Credentials
Manage Users
Learner
Class Notification
Activity Notifications

Level 2:

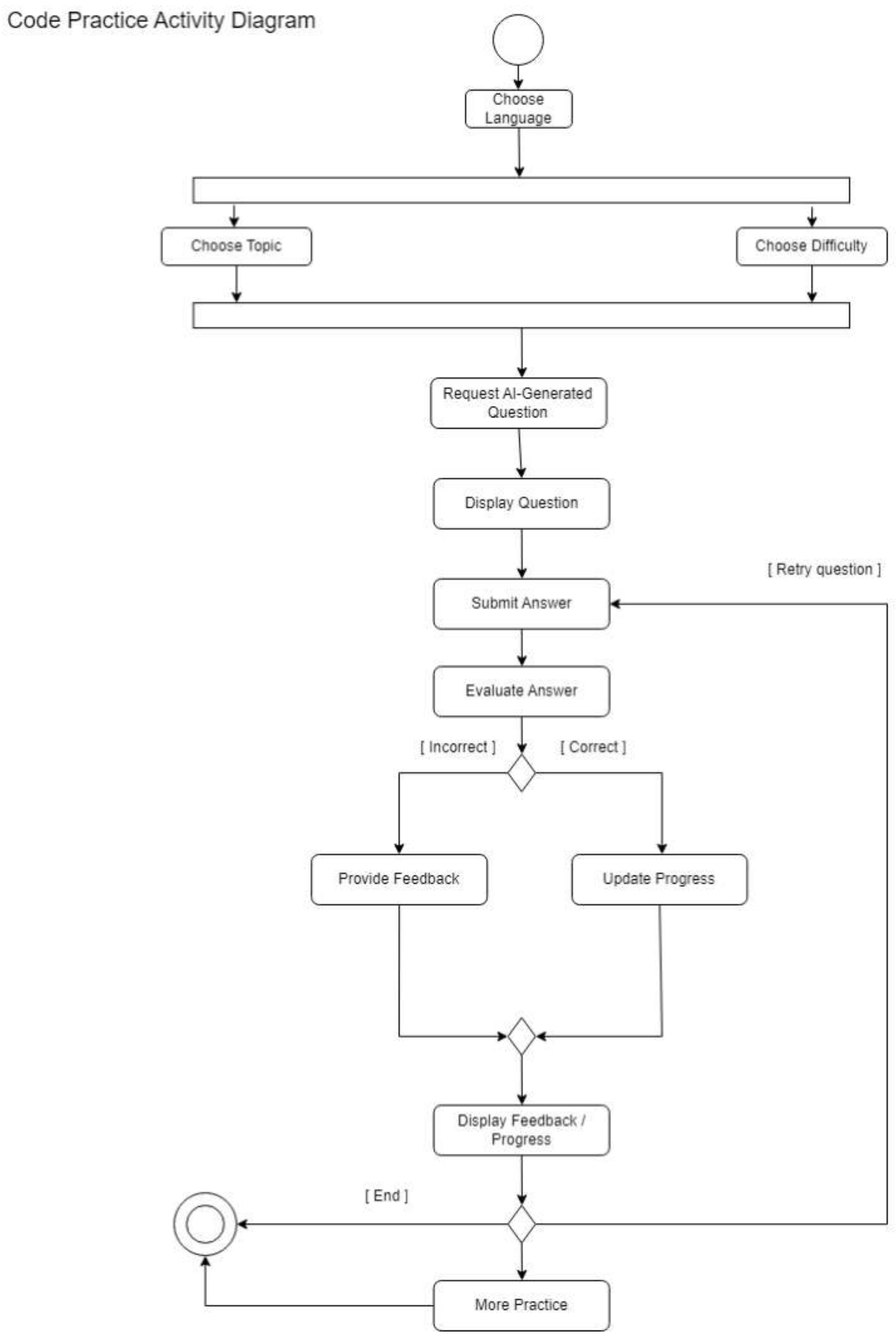

AI API
Validated Solutions
Formatted Question Prompt
Correctness & Feedback
Validate Response
Raw Solutions
Correctness & Feedback
AI-Generated Question
Student
Progress Notification
Formatted Question Prompt
Raw Solutions
Notification Viewed Status
Notification Viewed Status
Progress Notification
Manage Class Notifications
Updated Class Content
Notification Creation Status
Manage Class Content
Updated Class Content
Class Progress Information
Updated Class Content
Saved Class Content
Sample Questions
Notification Viewed Status
Question Data Store
Newly Generated Question
Verified Question
Question Feedback
Generate Question
Question Details
Formatted Question
Answer Question
Question Details & Raw Solutions
Correctness & Feedback
Answer Course Content
Saved Course Content
Updated Course Progress
Class Data Store
View Filters
Class Information
View Class Analytics & Progress
View Filters
Class Analytics & Progress
Instructor
Class Creation Status
Updated Question Status
Raw Solutions & Chosen Question
Student Information
Class Information & Generated Join Code
Registration Status
Class Creation Status
Provide Question Feedback
User Data Store
Notification Viewed Status
User Activity Information
Manage User Notifications
Notification Viewed Status
Register into Class
Create Class
Class Information
Validated User Information
Registration Status
Question Feedback
Class Join Code
Register/Login User
Raw User Information
Activity Notification
Points, Badges and Streaks
Registration Status
Learner
Registration Status
Points, Badges and Streaks

4.3.2 State Diagram

4.3.3 Sequence Diagram



4.3.4 Activity Diagram



5. Technical Specifications

5.1 Tech Stack

The project will utilize the SERN tech stack, also known as the SQL, Express, React, and Node tech stack. This is a modern web development tech stack that allows the use of Javascript for both front and backend development which eliminates the need to switch between different programming languages when developing the full stack. This will consist of the following:

1. **SQL:** Programming language primarily used for managing and manipulating relational databases. SQL will be used in the database management system to query and update the stored data.
2. **Express:** A minimal web application framework for NodeJS primarily focused on providing a set of server-side features and tools. Express will be used to handle HTTP requests, routing, and middleware management.
3. **React:** A popular JavaScript library for building reusable UI components and user interfaces. React will be used for the client-side rendering to create dynamic and interactive user interfaces.
4. **NodeJS:** A JavaScript runtime environment used in the creation of server applications. NodeJS will be used for handling external libraries, modules, and packages for different features required in the system.

5.2 Tools

IDE: Visual Studio Code
Version control systems: Git
Project management tools: Google Drive, Slack, Discord.
Unit testing: Jest
CI/CD: GitHub Action
Deployment & Hosting: Docker, Digital Ocean
APIs: Cohere API, OpenAI API
Software Development Methodologies: ScrumBan

6. Testing Strategies

6.1 Regression testing

Our testing strategy for regression testing revolves around automated regression testing integrated with our GitHub Actions workflow. Critical parts of the system will be covered by unit tests, integration tests, and system tests. These tests will be automated to allow efficient and repetitive execution, which is vital for regression testing and ensuring that new features do not break existing code bases. On every push and pull request to the repository, GitHub Actions will be configured to automatically install dependencies and run these tests on Ubuntu servers. The results will then be either validated or rejected, depending on whether the test cases pass. This should prevent any broken code from reaching the production (main) branch, as it will catch issues on push and pull requests.

6.2 Unit testing

Our testing strategy for unit testing involves the use of the Jest testing framework for both the front-end and back-end components of our system. Our team will follow test-driven development. For the front-end, built with React, we will write white-box testing to ensure that every component behaves as expected in isolation, including rendering correctly and responding to user interactions. For the back-end, built in Node.js, we are planning to have a very modular structure. Our unit tests will verify the functionality of individual methods and APIs, ensuring each performs its intended task with the expected output. Following a TDD approach, these tests will be written before implementation to ensure that all code has a corresponding test case. This method, while harder on the developers, will ensure high cohesion and low coupling, resulting in maintainable, modular code.

6.3 Integration testing

1. Set up the test environment, including a test database and necessary dependencies.
2. Conduct end-to-end testing by simulating user scenarios and validating expected results.
3. Test the server-side APIs using frameworks like Jest, Mocha, or Postman.

4. Validate the client-side UI using frameworks like Jest, React Testing Library, or Enzyme.
5. Test database interactions with a dedicated test database.
6. Integrate integration testing into your CI/CD process for automated testing.
7. Report any discovered bugs or issues to the development team for resolution.

6.4 Usability testing

1. Define User Profile - Identify the target user for the website.
2. Create Scenario: Develop specific tasks for users to perform, such as using login or search capabilities.
3. Participant Recruitment: Find individuals who match your user profile and are willing to participate in usability testing.
4. Test Environment Settings: Configure the required environment, including the server hosting the website.
5. Conduct the test: Provide scenarios to participants and observe their behavior and feedback while using the website.
6. Data Analysis: Evaluate collected feedback and observation data to identify usability issues and areas for improvement.
7. Improvements and iterations: Address identified issues and incorporate proposed improvements into the website. Repeat the usability test process as necessary.

6.5 User Acceptance Testing

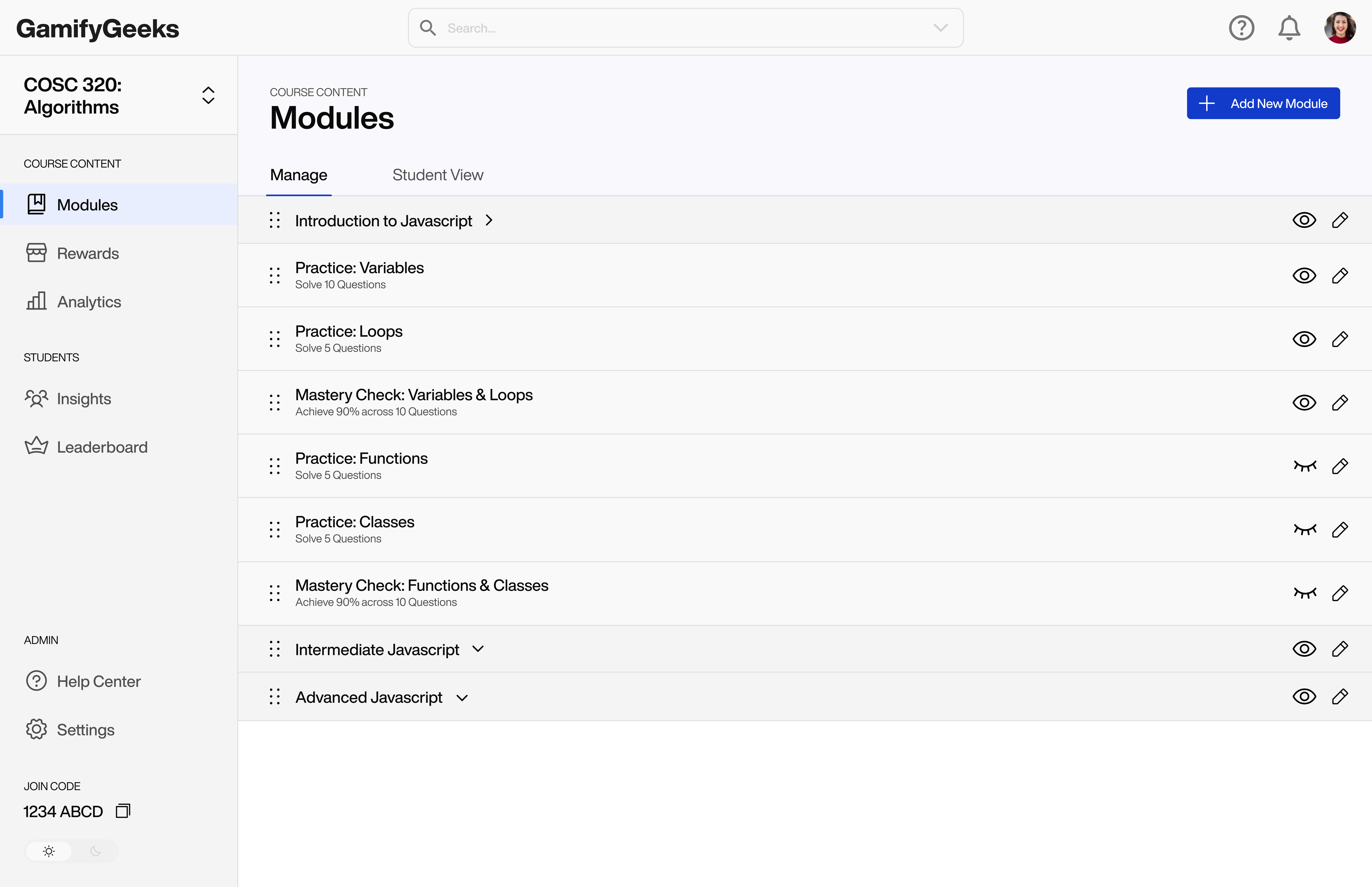
Structured user acceptance testing will be conducted before major releases of the system such as the minimum viable product and the final delivery. This will be used to validate whether the product meets business needs, requirements, and usability expectations. The user acceptance testing will consist primarily of the following steps:

1. Create & identify test scenarios: Using the system requirements, user stories, and use cases, prepare test scenarios to represent typical user workflows and business processes. The set of test scenarios should sufficiently cover the majority of systems' applications and critical functionalities.

2. Acquire & prepare test users: Find a set of test users which sufficiently represent each major user group for the system. Additionally, familiarize the test users will the test scenarios and the objectives of the system
3. Execute & Observe: Test users are to go through the prepared test scenarios and identify any missed expectations, issues, or defects in the system. Test data should be tracked and categorized according to their related features and components.
4. Resolve & Retest: Issues identified in the previous step are assigned to the backlogs to be resolved or moved to a future release of the system. Furthermore, the user acceptance training is repeated until the system sufficiently satisfies user expectations and business requirements.

7. User Interface Design

7.1 UI Mockups



GamifyGeeks

Search...

COSC 320: Algorithms

COURSE CONTENT

Modules

Rewards

Analytics

STUDENTS

Insights

Leaderboard

ADMIN

Help Center

Settings

JOIN CODE

1234 ABCD

COURSE CONTENT

Modules

Add New Module

Manage

Student View

Introduction to Javascript

Practice: Variables
Solve 10 Questions

Practice: Loops
Solve 5 Questions

Mastery Check: Variables & Loops
Achieve 90% across 10 Questions

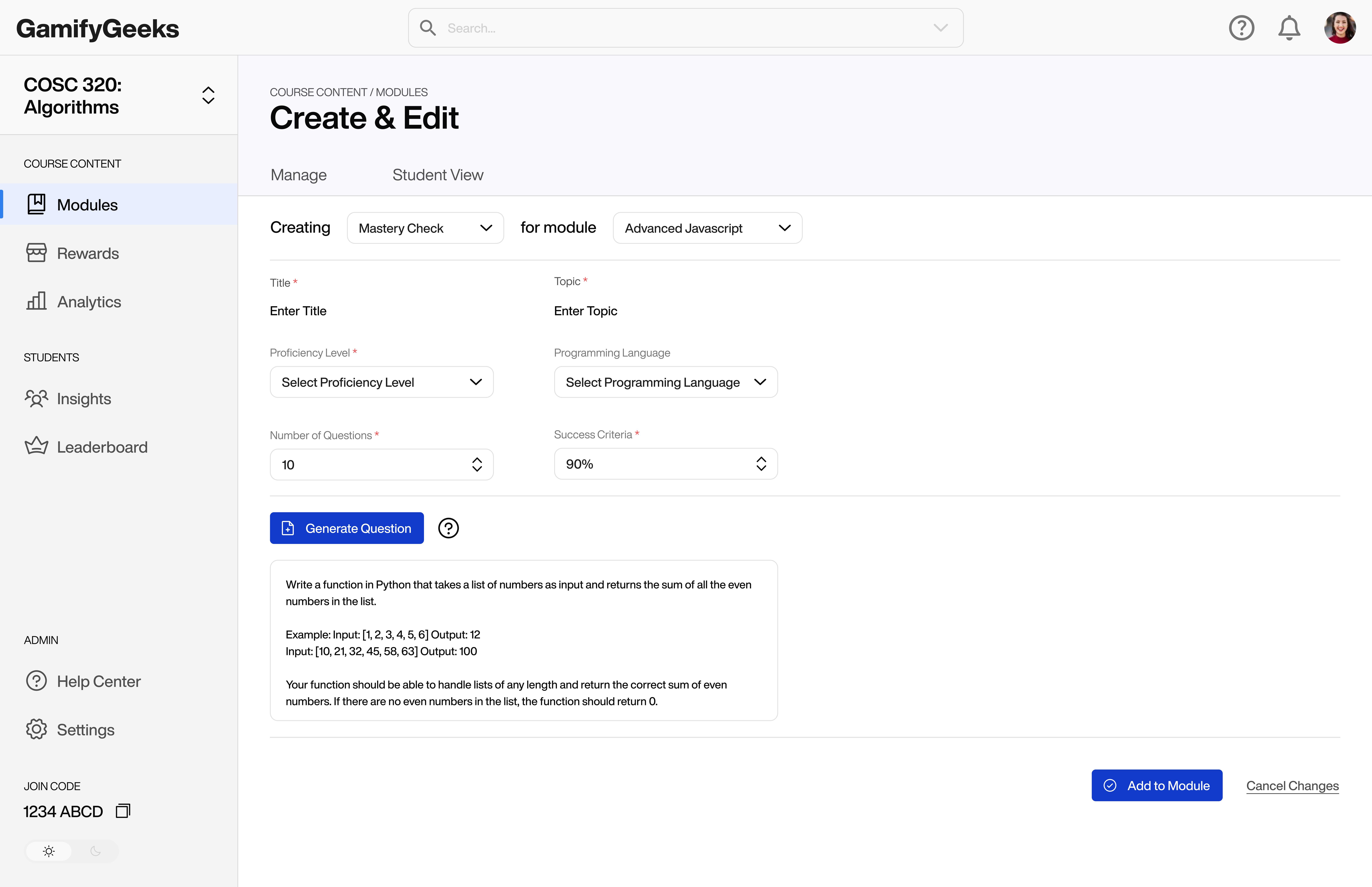
Practice: Functions
Solve 5 Questions

Practice: Classes
Solve 5 Questions

Mastery Check: Functions & Classes
Achieve 90% across 10 Questions

Intermediate Javascript

Advanced Javascript


GamifyGeeks

Search...

COSC 320: Algorithms

COURSE CONTENT

Modules

Rewards

Analytics

STUDENTS

Insights

Leaderboard

ADMIN

Help Center

Settings

JOIN CODE

1234 ABCD

COURSE CONTENT / MODULES

Create & Edit

Manage   Student View

Creating [Mastery Check] for module [Advanced Javascript]

Title *
Enter Title

Topic *
Enter Topic

Proficiency Level *
Select Proficiency Level

Programming Language
Select Programming Language

Number of Questions *
10

Success Criteria *
90%

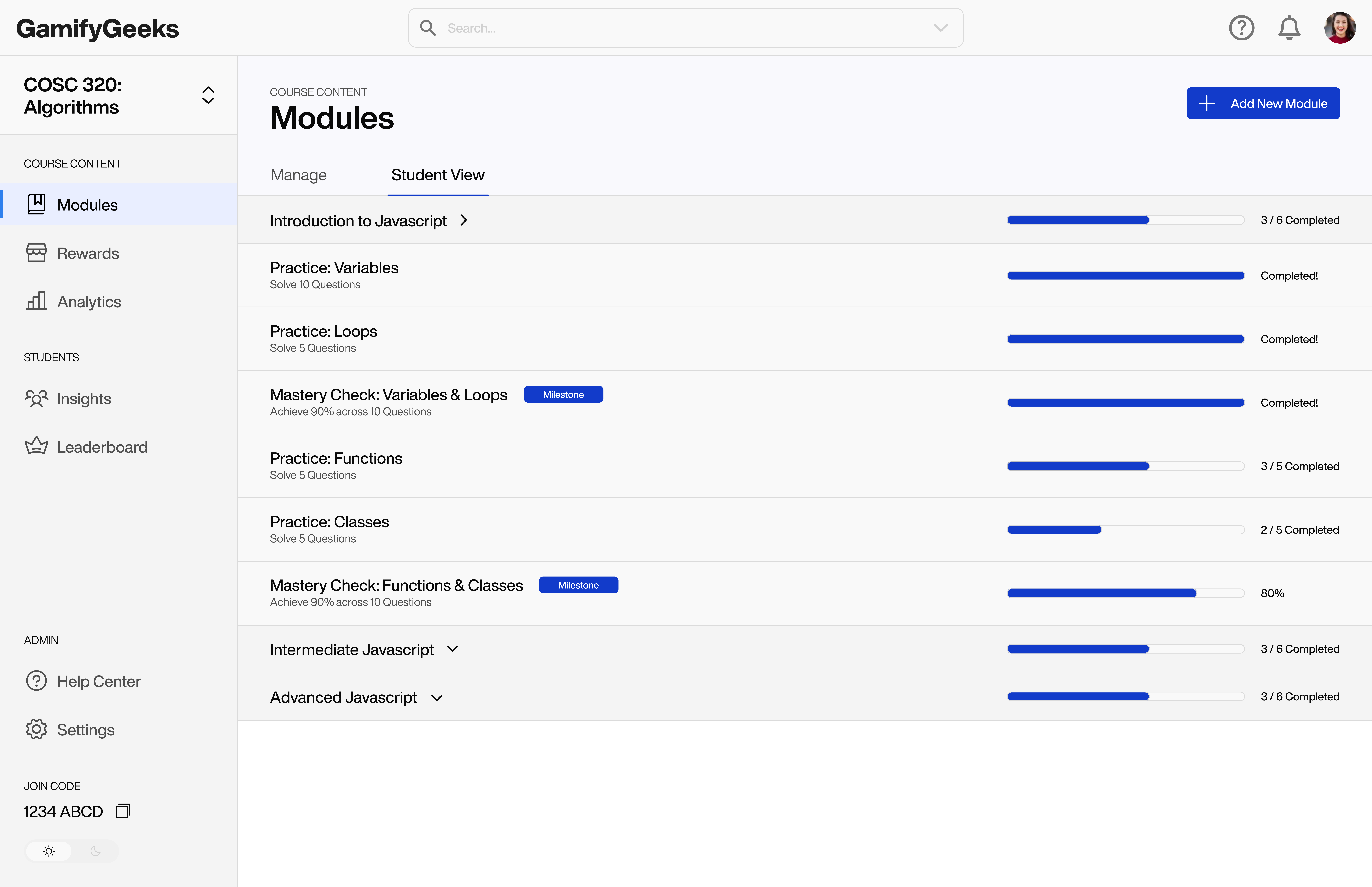
Generate Question

Write a function in Python that takes a list of numbers as input and returns the sum of all the even numbers in the list.

Example: Input: [1, 2, 3, 4, 5, 6] Output: 12
Input: [10, 21, 32, 45, 58, 63] Output: 100

Your function should be able to handle lists of any length and return the correct sum of even numbers. If there are no even numbers in the list, the function should return 0.

Add to Module   Cancel Changes



GamifyGeeks

Search...

COSC 320: Algorithms

COURSE CONTENT

Modules

Rewards

Analytics

STUDENTS

Insights

Leaderboard

ADMIN

Help Center

Settings

JOIN CODE

1234 ABCD

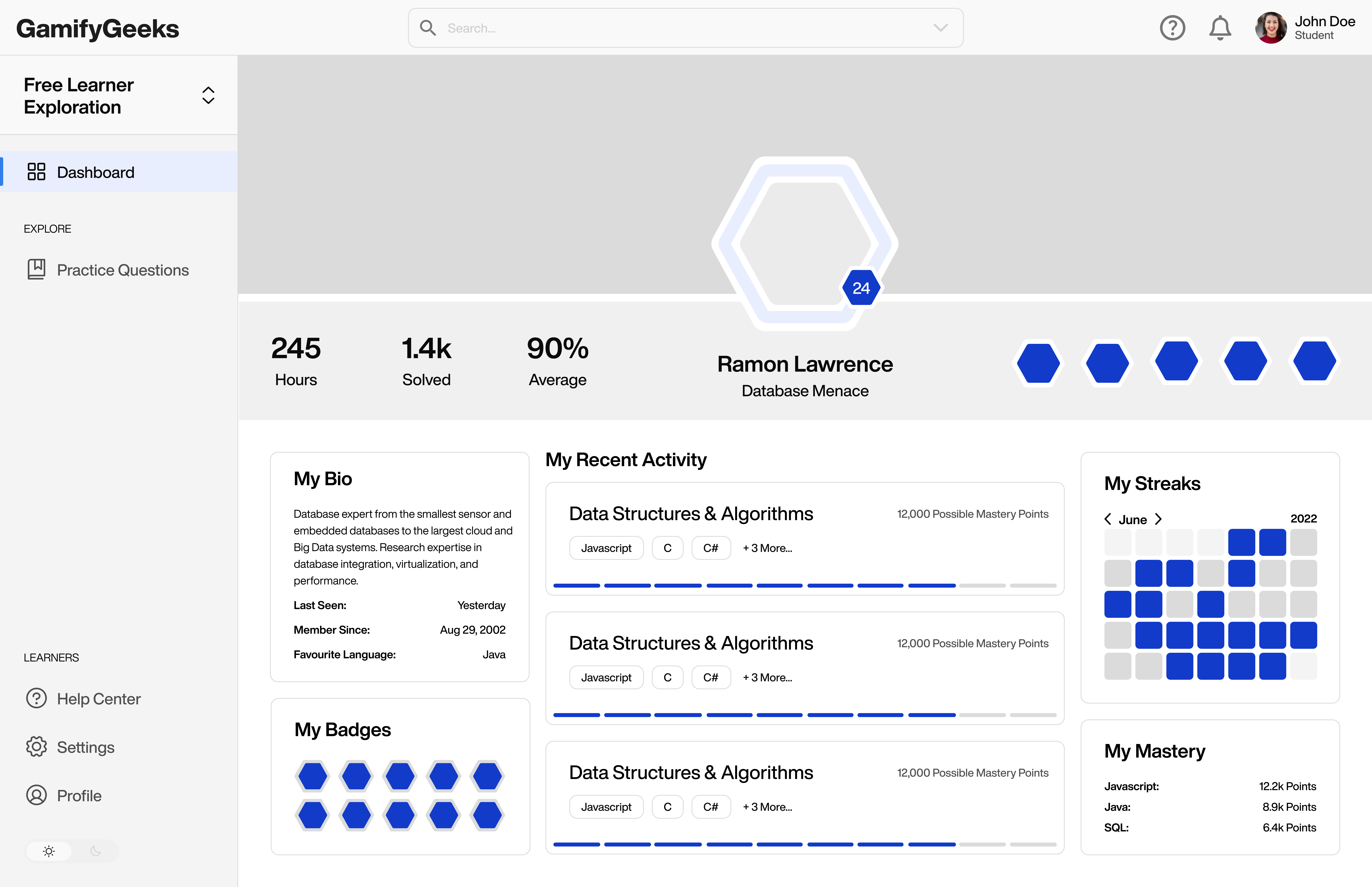
COURSE CONTENT

Modules

Add New Module

Manage | Student View

Introduction to Javascript	3 / 6 Completed
Practice: Variables Solve 10 Questions	Completed!
Practice: Loops Solve 5 Questions	Completed!
Mastery Check: Variables & Loops Milestone Achieve 90% across 10 Questions	Completed!
Practice: Functions Solve 5 Questions	3 / 5 Completed
Practice: Classes Solve 5 Questions	2 / 5 Completed
Mastery Check: Functions & Classes Milestone Achieve 90% across 10 Questions	80%
Intermediate Javascript	3 / 6 Completed
Advanced Javascript	3 / 6 Completed



GamifyGeeks

Search...

John Doe
Student

Free Learner Exploration

Dashboard

EXPLORE

Practice Questions

LEARNERS

Help Center

Settings

Profile

24

245	1.4k	90%
Hours	Solved	Average

Ramon Lawrence
Database Menace

My Bio

Database expert from the smallest sensor and embedded databases to the largest cloud and Big Data systems. Research expertise in database integration, virtualization, and performance.

Last Seen:	Yesterday
Member Since:	Aug 29, 2002
Favourite Language:	Java

My Badges

My Recent Activity

Data Structures & Algorithms — 12,000 Possible Mastery Points
Javascript | C | C# | + 3 More...

Data Structures & Algorithms — 12,000 Possible Mastery Points
Javascript | C | C# | + 3 More...

Data Structures & Algorithms — 12,000 Possible Mastery Points
Javascript | C | C# | + 3 More...

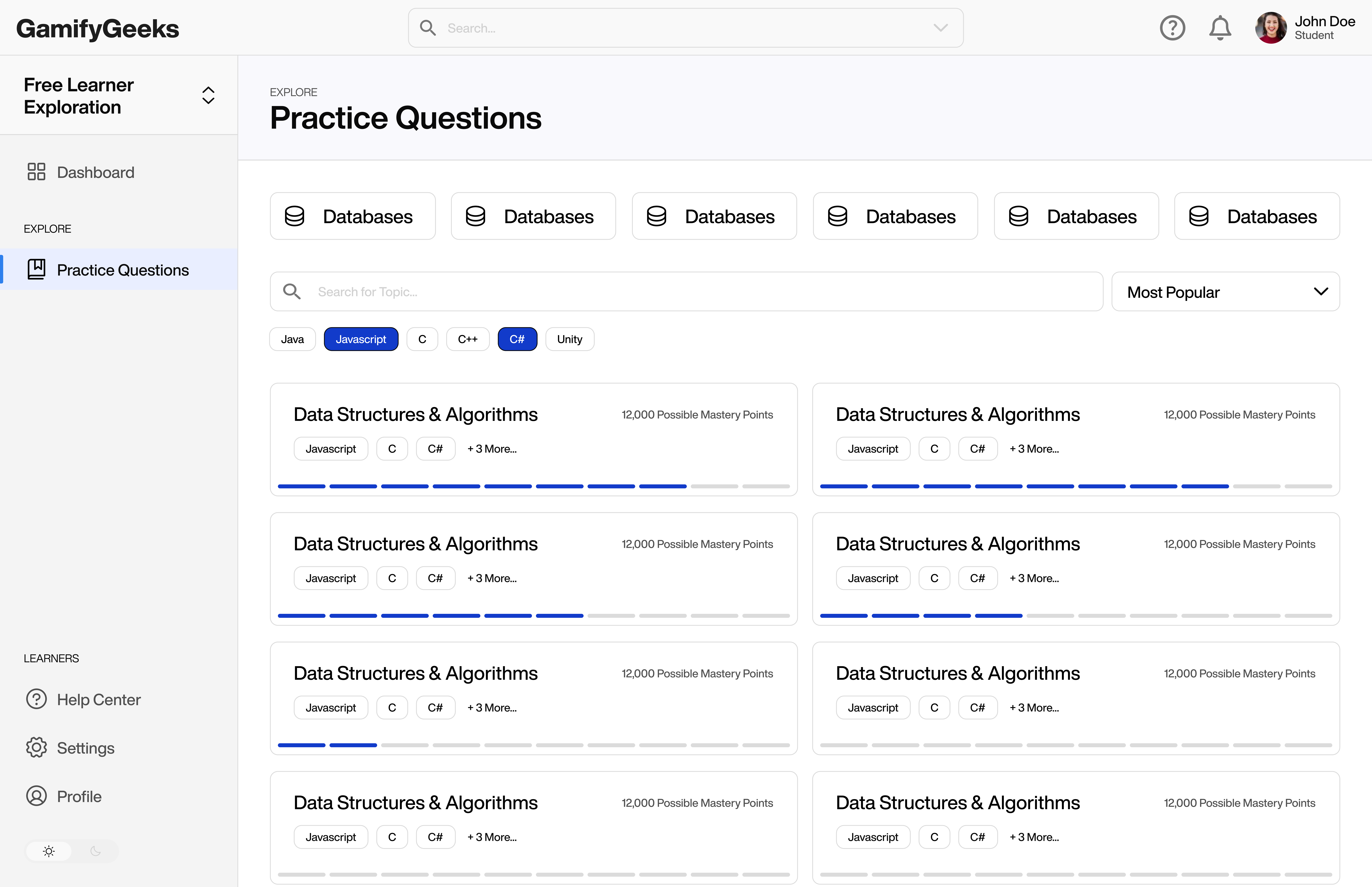
My Streaks

< June > 2022

⬜	⬜	⬜	⬜	🟦	🟦	▨
▨	🟦	🟦	▨	🟦	▨	▨
🟦	🟦	▨	🟦	▨	▨	▨
▨	🟦	🟦	🟦	🟦	🟦	🟦
▨	▨	🟦	🟦	🟦	🟦	⬜

My Mastery

Javascript:	12.2k Points
Java:	8.9k Points
SQL:	6.4k Points



GamifyGeeks

Search...

John Doe
Student

Free Learner Exploration

Dashboard

EXPLORE

Practice Questions

LEARNERS

Help Center

Settings

Profile

EXPLORE

Practice Questions

Databases | Databases | Databases | Databases | Databases | Databases

Search for Topic...

Most Popular

Java | Javascript | C | C++ | C# | Unity

Data Structures & Algorithms — 12,000 Possible Mastery Points
Javascript | C | C# | + 3 More...

Data Structures & Algorithms — 12,000 Possible Mastery Points
Javascript | C | C# | + 3 More...

Data Structures & Algorithms — 12,000 Possible Mastery Points
Javascript | C | C# | + 3 More...

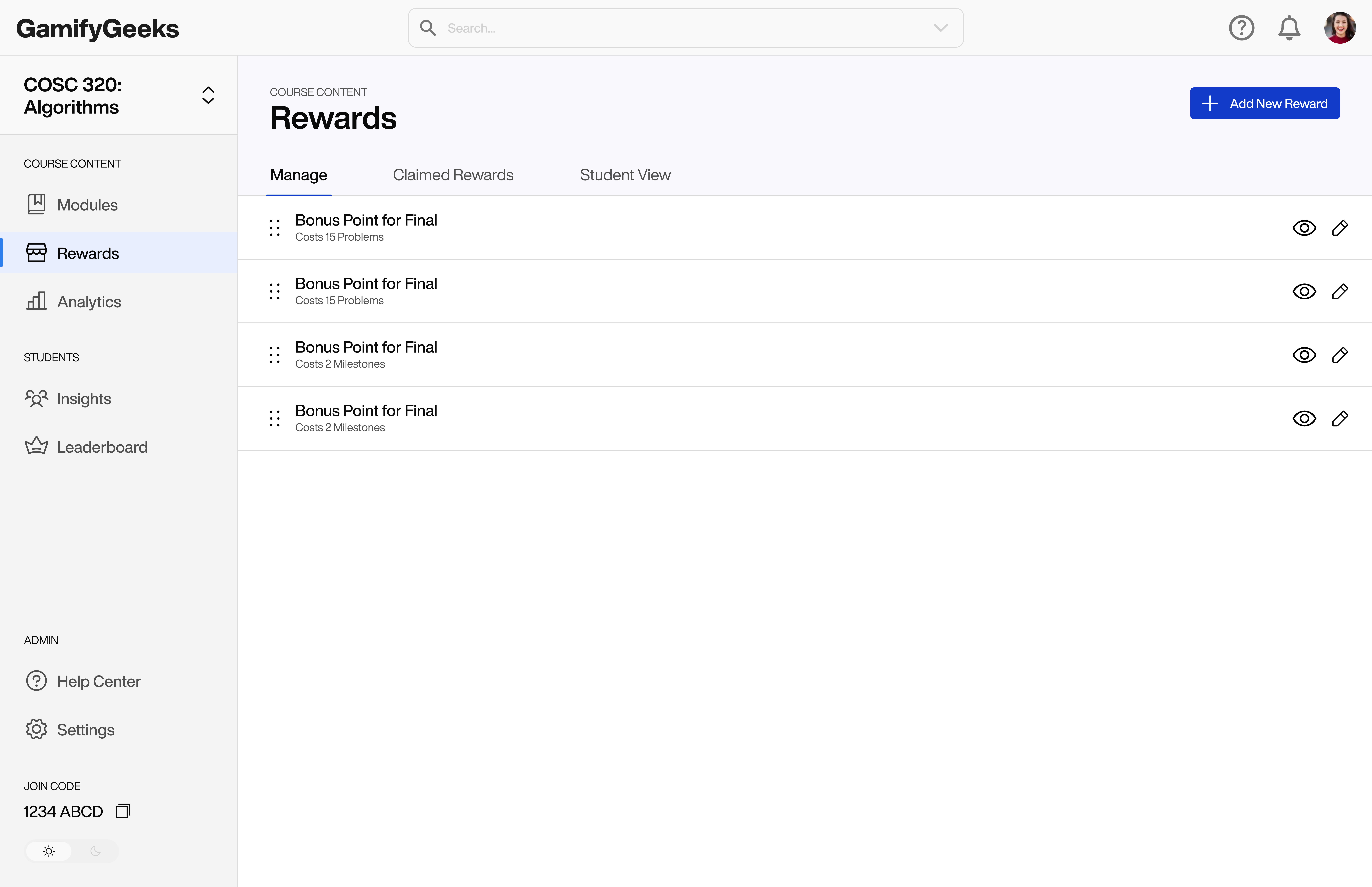
Data Structures & Algorithms — 12,000 Possible Mastery Points
Javascript | C | C# | + 3 More...

Data Structures & Algorithms — 12,000 Possible Mastery Points
Javascript | C | C# | + 3 More...

Data Structures & Algorithms — 12,000 Possible Mastery Points
Javascript | C | C# | + 3 More...

Data Structures & Algorithms — 12,000 Possible Mastery Points
Javascript | C | C# | + 3 More...

Data Structures & Algorithms — 12,000 Possible Mastery Points
Javascript | C | C# | + 3 More...



COURSE CONTENT

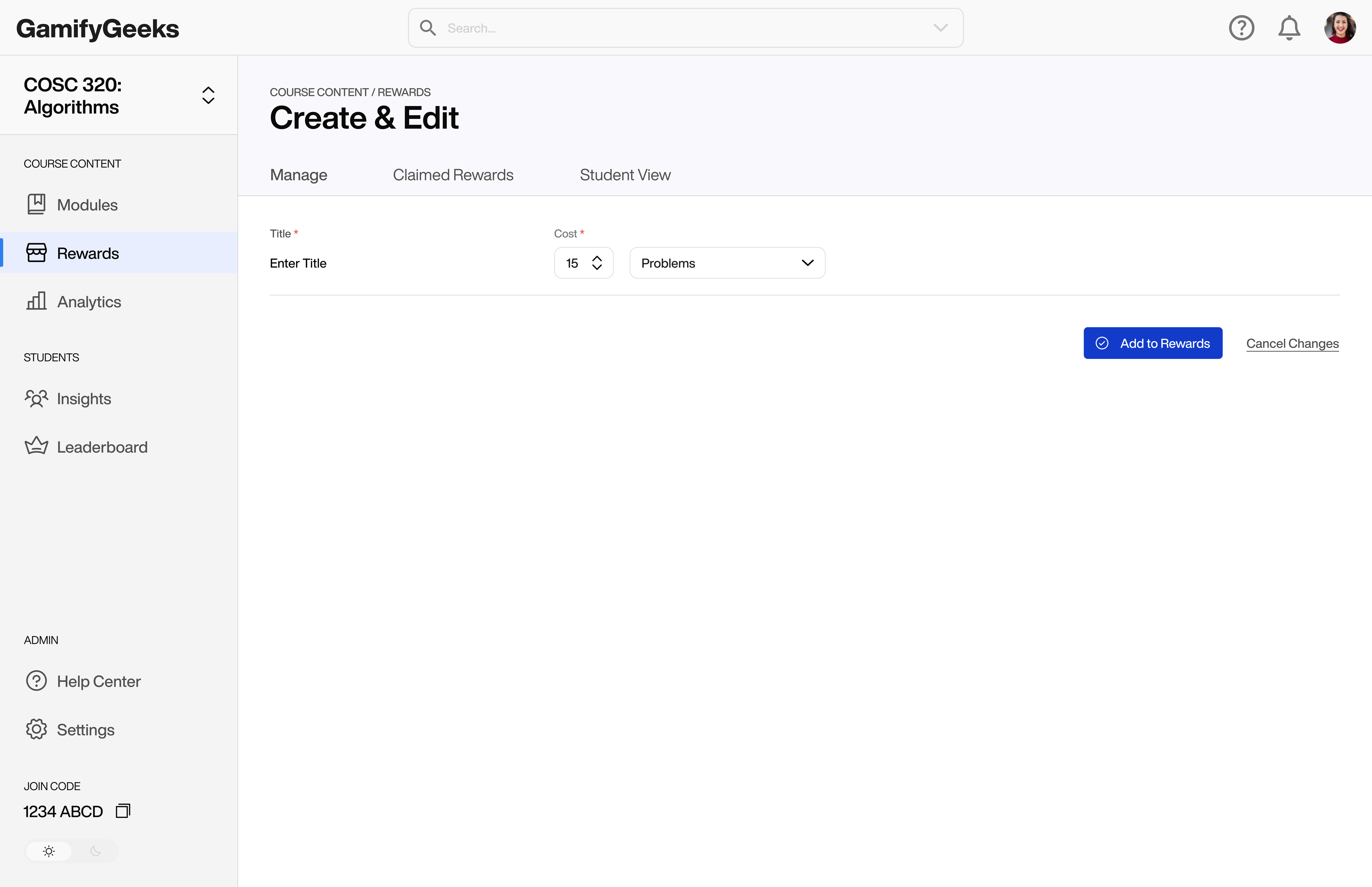
Rewards

Bonus Point for Final
Costs 15 Problems

Bonus Point for Final
Costs 15 Problems

Bonus Point for Final
Costs 2 Milestones

Bonus Point for Final
Costs 2 Milestones



GamifyGeeks

Search...

COSC 320: Algorithms

COURSE CONTENT

Modules

Rewards

Analytics

STUDENTS

Insights

Leaderboard

ADMIN

Help Center

Settings

JOIN CODE

1234 ABCD

COURSE CONTENT / REWARDS

Create & Edit

Manage | Claimed Rewards | Student View

Title *

Enter Title

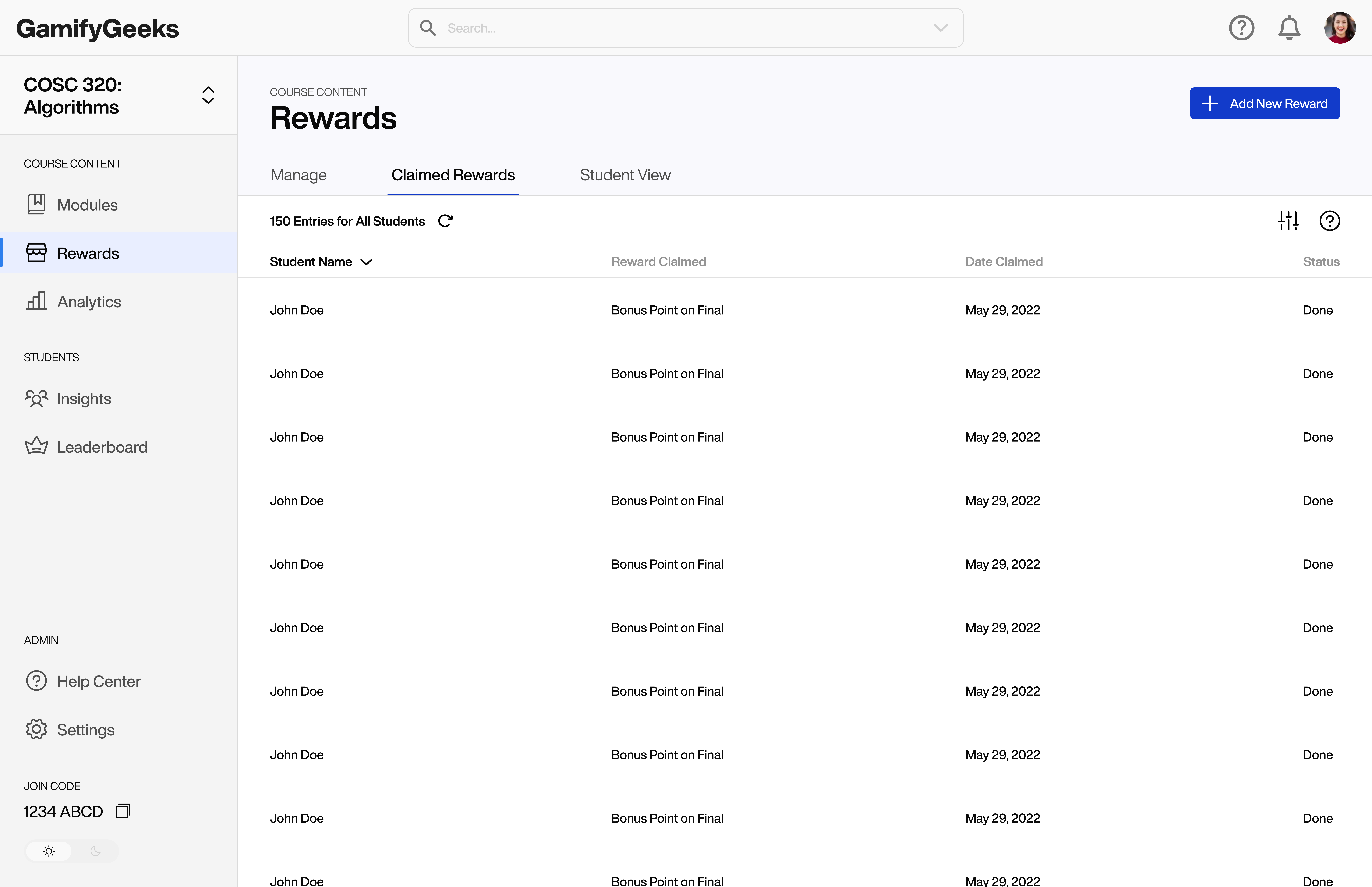
Cost *

15

Problems

Add to Rewards

Cancel Changes

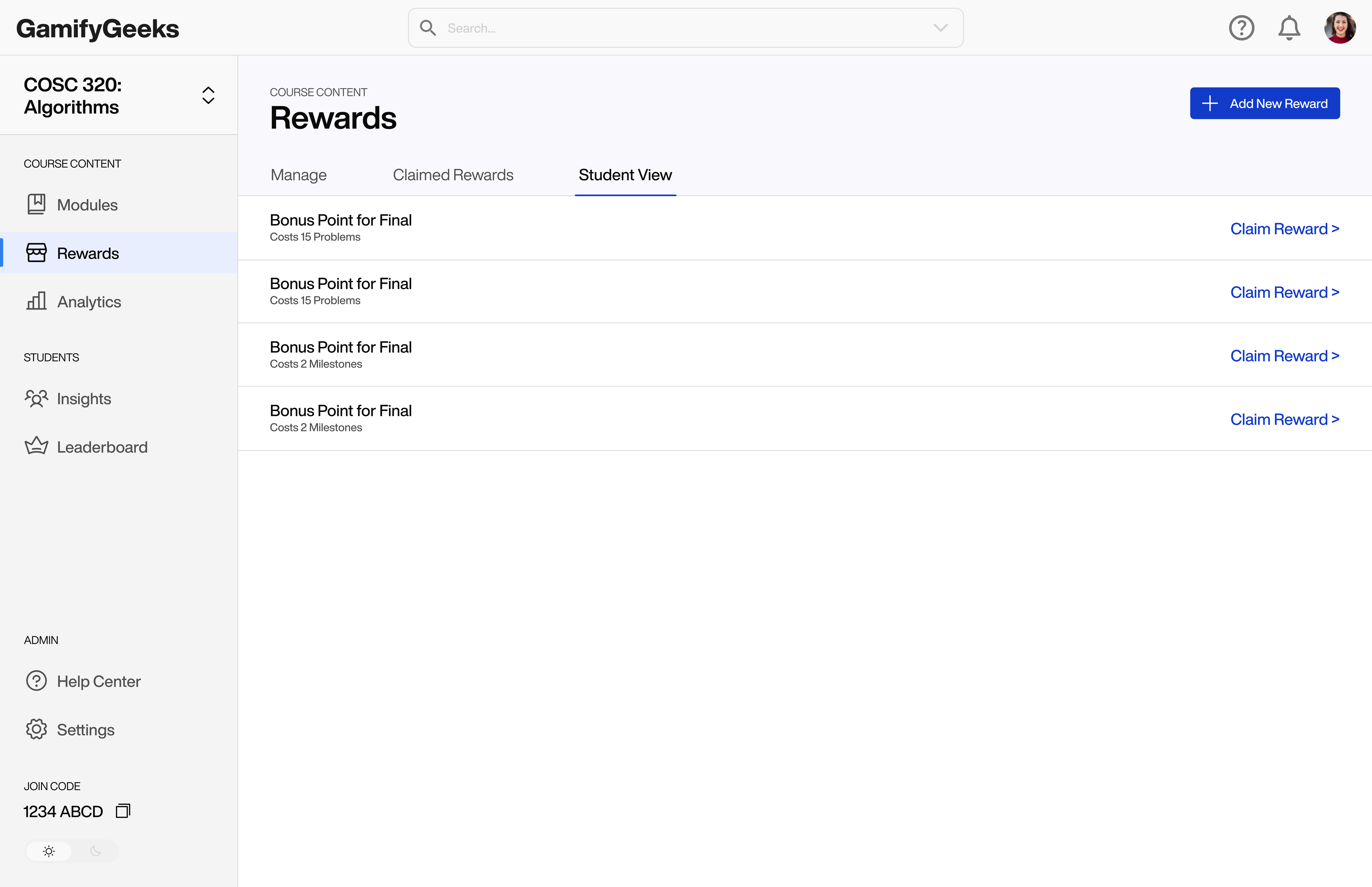


COURSE CONTENT

Rewards

150 Entries for All Students

Student Name	Reward Claimed	Date Claimed	Status
John Doe	Bonus Point on Final	May 29, 2022	Done
John Doe	Bonus Point on Final	May 29, 2022	Done
John Doe	Bonus Point on Final	May 29, 2022	Done
John Doe	Bonus Point on Final	May 29, 2022	Done
John Doe	Bonus Point on Final	May 29, 2022	Done
John Doe	Bonus Point on Final	May 29, 2022	Done
John Doe	Bonus Point on Final	May 29, 2022	Done
John Doe	Bonus Point on Final	May 29, 2022	Done
John Doe	Bonus Point on Final	May 29, 2022	Done
John Doe	Bonus Point on Final	May 29, 2022	Done



GamifyGeeks

Search...

COSC 320: Algorithms

COURSE CONTENT

Modules

Rewards

Analytics

STUDENTS

Insights

Leaderboard

ADMIN

Help Center

Settings

JOIN CODE

1234 ABCD

COURSE CONTENT

Rewards

Add New Reward

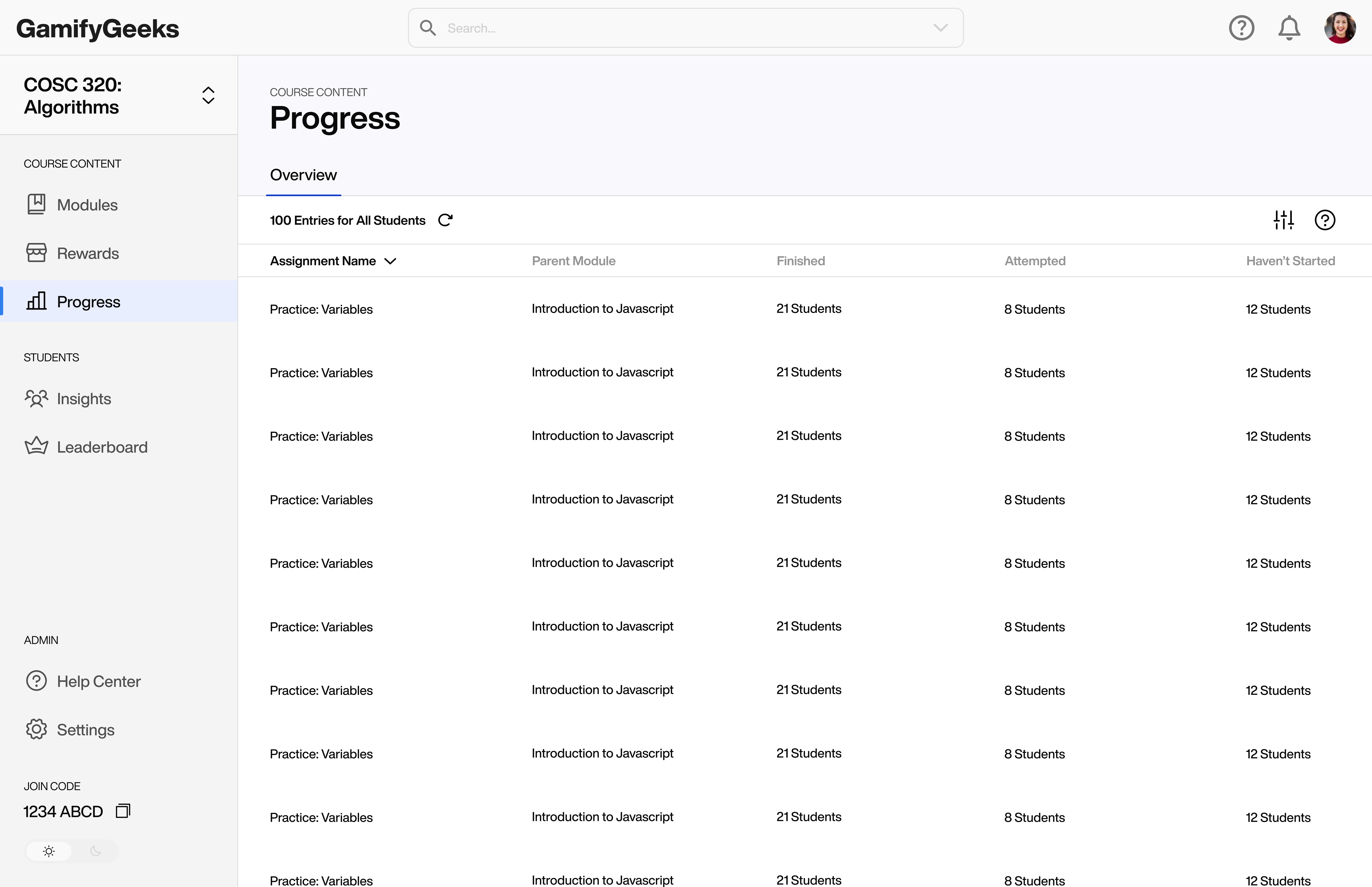
Manage | Claimed Rewards | Student View

Bonus Point for Final
Costs 15 Problems
Claim Reward >

Bonus Point for Final
Costs 15 Problems
Claim Reward >

Bonus Point for Final
Costs 2 Milestones
Claim Reward >

Bonus Point for Final
Costs 2 Milestones
Claim Reward >



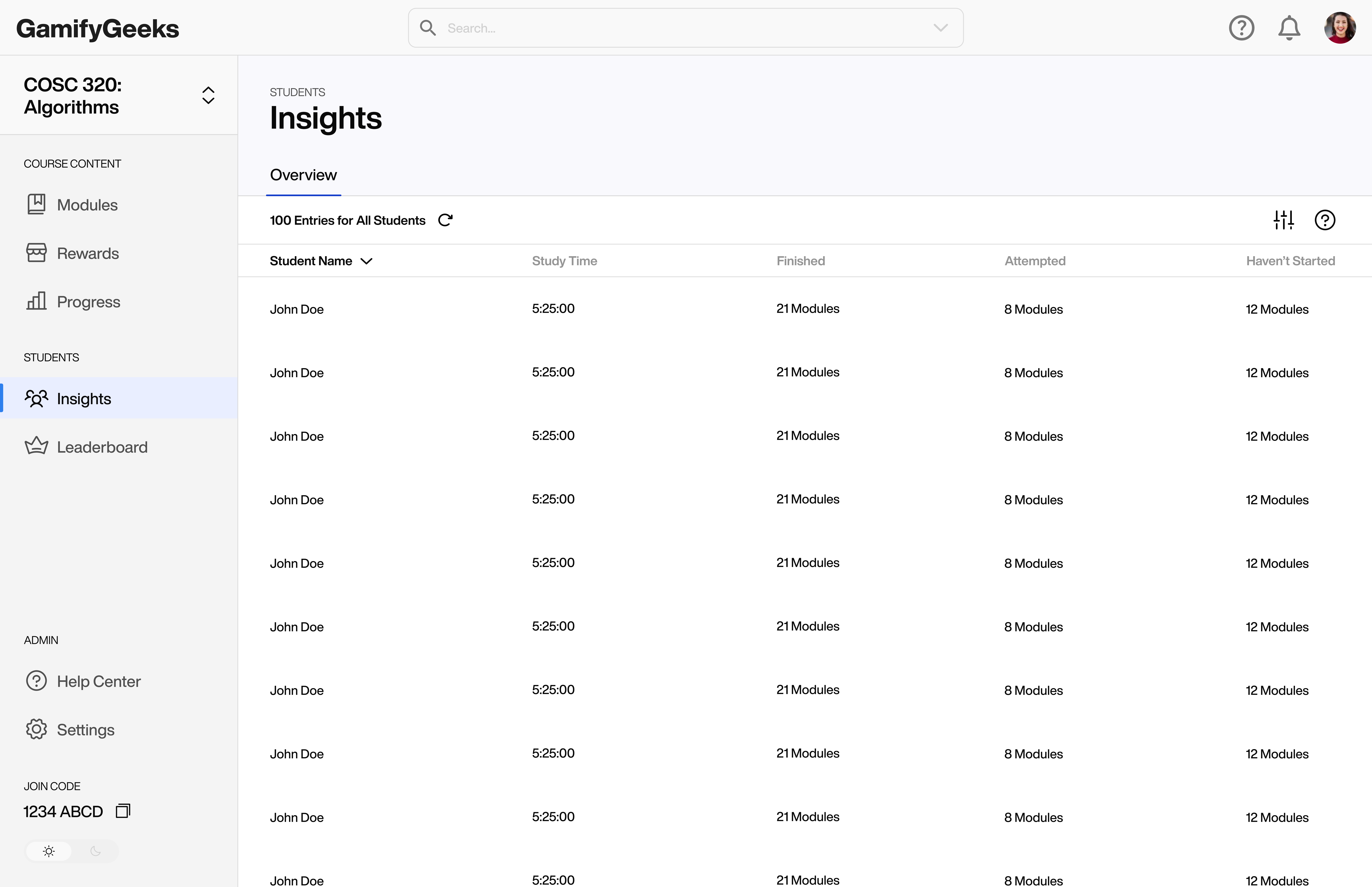
COURSE CONTENT

Progress

Overview

100 Entries for All Students

Assignment Name	Parent Module	Finished	Attempted	Haven't Started
Practice: Variables	Introduction to Javascript	21 Students	8 Students	12 Students
Practice: Variables	Introduction to Javascript	21 Students	8 Students	12 Students
Practice: Variables	Introduction to Javascript	21 Students	8 Students	12 Students
Practice: Variables	Introduction to Javascript	21 Students	8 Students	12 Students
Practice: Variables	Introduction to Javascript	21 Students	8 Students	12 Students
Practice: Variables	Introduction to Javascript	21 Students	8 Students	12 Students
Practice: Variables	Introduction to Javascript	21 Students	8 Students	12 Students
Practice: Variables	Introduction to Javascript	21 Students	8 Students	12 Students
Practice: Variables	Introduction to Javascript	21 Students	8 Students	12 Students
Practice: Variables	Introduction to Javascript	21 Students	8 Students	12 Students



STUDENTS

Insights

Overview

100 Entries for All Students

Student Name	Study Time	Finished	Attempted	Haven't Started
John Doe	5:25:00	21 Modules	8 Modules	12 Modules
John Doe	5:25:00	21 Modules	8 Modules	12 Modules
John Doe	5:25:00	21 Modules	8 Modules	12 Modules
John Doe	5:25:00	21 Modules	8 Modules	12 Modules
John Doe	5:25:00	21 Modules	8 Modules	12 Modules
John Doe	5:25:00	21 Modules	8 Modules	12 Modules
John Doe	5:25:00	21 Modules	8 Modules	12 Modules
John Doe	5:25:00	21 Modules	8 Modules	12 Modules
John Doe	5:25:00	21 Modules	8 Modules	12 Modules
John Doe	5:25:00	21 Modules	8 Modules	12 Modules

GamifyGeeks

Search...

COSC 320: Algorithms

COURSE CONTENT

Modules

Rewards

Progress

STUDENTS

Insights

Leaderboards

ADMIN

Help Center

Settings

JOIN CODE

1234 ABCD

STUDENTS

Leaderboards

Overview

100 Entries for All Students

Rank	Student	Problems solved	Study Time
#1	John Doe	21 Modules	5:25:00
#1	John Doe	21 Modules	5:25:00
#1	John Doe	21 Modules	5:25:00
#1	John Doe	21 Modules	5:25:00
#1	John Doe	21 Modules	5:25:00
#1	John Doe	21 Modules	5:25:00
#1	John Doe	21 Modules	5:25:00
#1	John Doe	21 Modules	5:25:00
#1	John Doe	21 Modules	5:25:00
#1	John Doe	21 Modules	5:25:00