

Florida International University
School of Computing and Information Sciences

Software Engineering Focus

Final Deliverable

Project Title: MathBotics

Team Members: Carlos A. Barbachano, Martyn Mallo, Jessica Moreno, Edgar Bermudez, Diane Abdullah, Lloyd Isaac, Mariella Alejandra Massuh, Veronica Parra

Product Owner(s): Gregory Reis

Mentor(s): Gregory Reis

Instructor: Masoud Sadjadi

The MIT License (MIT)

Copyright (c) 2016 *Florida International University*

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Abstract

This document presents the information necessary to gain a good understanding of the MathBotics learning platform. MathBotics is a robotics learning application designed for K-12 students. The platform will serve as a portal for educators and students to interact with courses and learning material. Educators will have the ability to create their own course based on lessons provided by an administrator. It is designed for students to learn math and robotics in a fun and interactive way.

Table of Contents

Introduction	6
Current System	6
Purpose of New System	7
User Stories	9
Implemented User Stories	9
Pending User Stories	13
Project Plan	15
Hardware and Software Resources	15
Sprints Plan	16
Sprint 1	16
Sprint 2	17
Sprint 3	18
Sprint 4	19
Sprint 5	19
Sprint 6	20
Sprint 7	21
System Design	22
Architectural Patterns	22
System and Subsystem Decomposition	23
Deployment Diagram	25
Design Patterns	26
System Validation	27
Glossary	28
Appendix	29
Appendix A - UML Diagrams	29
Static UML Diagrams	29
Dynamic UML Diagrams	34

Appendix B - User Interface Design	39
Appendix C - Sprint Review Reports	49
Appendix D - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and other documents	56
References	64

INTRODUCTION

The main purpose of this product is to provide K-12 students a way to learn mathematics and robotics in a fun, interactive manner. The application serves as an online portal that will connect educators and students with courses, assessments, and learning material. Since this product was developed before the current team took over the workload, the team focused on improving the portal, user interface, and user interaction as well as adding new features to the platform. The current system emphasizes on versatile and easy-to-use tools for content creation. In addition, the system has core management functionalities that allows proper hierarchical organization of content and user alike. Management functionalities are available according to the user's hierarchical rank. This section will focus on describing the current system and the purpose of the new system.

Current System

The current system for the MathBotics learning platform provides an all around inclusive environment to understand and educate the understanding of robotics to elementary, middle, and highschool students. The current platform offers versatile and easy-to-use tools for content-creators to create content, or lessons, and instructors to create and manage classes full of students using the interactive content created by the content creators. The platform is intended to provide a rich environment for the students to learn and interact with the content, and for their guardians to view and stay tuned to their progress through slightly more limited tools than those provided to their instructors.

The Mathbotics portal also has a management side that allows administrators to manage the content that instructors and students will be interacting with. The content is customized and curated for students of different grade levels and learning methods. The managerial side also has the potential to add other “admins” or instructors, so that they may use the platform to fit their needs.

Purpose of New System

The purpose of this new version is to provide new features to the MathBotics learning application as well as a more intuitive and easy to use user interface.

The new version of MathBotics has the following features:

- Student information can be deleted
- Student information can be edited
- User can log out of the application
- Alert when username already exists
- User can reset their password
- Instructor can modify course grade level
- Instructor can view student grades
- Students can view their grades
- Instructor can search lessons by title
- Lesson catalogue was modified to allow for a better user experience
- Lesson plan interface was modified to allow for better utilization of course removal

- Lessons can be deleted by the administrator
- Lessons are greyed out when added to a lesson plan
- Allows for a multi-collaborative environment through port sharing by utilizing liveshare

USER STORIES

The following section provides the detailed user stories that were implemented in this iteration of the MathBotics project. These user stories served as the basis for the implementation of the project's features. This section also shows the user stories that are to be considered for future development.

Implemented User Stories

1. Clone Github Repository
 - a. As an engineer, I should be able to clone and push to a remote repository
 - b. As an engineer, I should be able to version control my code by branching and making pull requests.
2. Setup Github Projects for Engineering team
 - a. As an engineer, I should see stories which are assigned to me through a github project board
 - b. As a product manager I should be able to view all the current stories, backlog tasks and work already done.
3. Read existing documentation
 - a. As an engineer, I should understand the given codebase and know which developing environment to begin developing in.
4. Setup Development Environment

- a. As an engineer, I should be able to begin developing within an environment and install needed dependencies/tools: npm, yarn, Node, TypeScript, Docker, GraphQL, React, Prisma, Relay.
- 5. Spike: Research and Learn Relevant Technologies
 - a. As an engineer, I should be able to understand and begin developing with the following technologies: npm, yarn, Node, TypeScript, Docker, GraphQL, React, PostgreSQL, Jest, Prisma, Relay.
- 6. Setup TravisCI or GH Actions integration with github
 - a. As a developer, I can see that new builds are being automatically deployed and tested as
- 7. Fix instructor edit button
 - a. As an instructor, I should be able to edit content.
- 8. Polish the codebase for the frontend, apply linting, formatting, etc
 - a. As a developer, Husky prevents me from pushing unformatted, poorly linted code on the frontend, just like it does on the backend.
- 9. Delete student information after courses ends [FrontEnd]
 - a. As an instructor, I should be able to delete and edit students/guardians in my course
- 10. Delete student information after courses ends [BackEnd]
 - a. As an instructor, I should be able to delete and edit students/guardians in my course
- 11. Log out button functionality
 - a. As a user, I should be able to log out from my session.
- 12. Bug: Fix yarn install command
 - a. As an engineer, I should be able to run yarn install to set up backend

13. Bug: Fix stuck on loading screen
 - a. As an engineer, I should be able to immediately load screen
14. Bug: Fix yarn gen command
 - a. As an engineer, I should be able to generate the gql type and prisma files
15. Bug: Fix yarn seed command
 - a. As an engineer, I should be able to create an admin/users
16. Bug: Fix prisma.schema file syntax
 - a. As an engineer, I should be able to have a schema of the database
17. Feature: Allow Windows OS Development
 - a. As an engineer, I should be able to develop in a Windows OS environment
18. Edit single student in course [Front End]
 - a. As a user, I should edit a student so that I can edit Student username, first name, last name and/or grade.
19. Delete single student in course [Front End]
 - a. As a user, I should be able to delete a student, in case a student drops from a course.
20. Edit single student in course [Back End]
 - a. As a user, I should edit a student so that I can edit Student username, first name, last name and/or grade..
21. Delete single student in course [Back End]
 - a. As a user, I should be able to delete a student, in case a student drops from a course.
22. Fix bug where page doesn't refresh when deleting all students
 - a. As a user, I should be able to see no students in my course when I delete them all without refreshing.

23. Add alert when username already exists
 - a. As a user, I should be alerted when the username I chose to input already exists in the database.
24. Connect Delete and edit student front end and back end
 - a. As an Instructor, I should be able to delete and/or edit a student in my course.
25. Add a plus button to add a lesson
 - a. As an instructor, I should be able to add a lesson, so that I can create a course for my students.
26. Adding student email to student entity in PostgreSQL
 - a. As a student, I should be able to use my email to reset my password.
27. Implement forgot password [FrontEnd]
 - a. As a user, I can click “forgot password” on the frontend and have a password reset link sent to the email I provide, if there is a user associated with that email.
28. Course creation doesn't have a grade level associated with it
 - a. As an instructor, I should be able to select the grade level of the course I am creating.
29. Style problems when lesson has a really long name bug
 - a. As a user, the user interface should be consistent.
30. Implement forgot password [BackEnd]
 - a. As a developer, I can reset someone's password with a token from GQL playground, like we do to register a new user.
31. Grades page for instructor [frontend]
 - a. As an instructor, I should be able to see my students' grades.
32. Grades page for students [frontend]
 - a. As a student, I should be able to view my grades.

33. Search lesson by title needs to filter the lessons object to match the input value
 - a. As an instructor, I should be able to search for lessons to add to my course.
34. Add delete button for deleting a lesson in admin view
 - a. As an admin, I should be able to delete lessons that I created.
35. Add implementation to grey out lessons that have already been added to the lesson plan
 - a. As an instructor, when I should be able to easily identify courses that have already been added to the lesson plan.

Pending User Stories

1. Grades page for instructor [backend]
 - a. As an engineer, I should make sure that lessons have a grade property for instructors to have a gradebook.
2. Grades page for students [backend]
 - a. As an engineer, I should make sure that lessons have a grade property for students to view their grades.
3. Add implementation to populate the data when you edit
 - a. As a user, data should be populated when I edit a student or course details.
4. Student view of progress, as a progress bar for lesson
 - a. As a student, I should be able to see how far along I am in the lesson.
5. Instructor view for students and their progress.
 - a. As an instructor, I can view, from my courses' students tab, my students' progress through the course.
6. No implementation for sending out emails through a provider
 - a. As a developer, I want emails to be sent to the users email.
7. Implement guardian creation for students [FrontEnd]

- a. As a guardian, I should be able to view my student's progress.
- 8. Implement guardian creation for students [BackEnd]
 - a. As a developer, I can create new guardians for students from graphql playground.
- 9. An instructor should only see the courses they created (Currently other instructors can see courses created by other instructors)
 - a. As an instructor, I should be able to only see the courses I created.
- 10. A student should only be able to see the course(s) they are enrolled in (currently students can see other courses that were made by any instructor that they're not enrolled in)
 - a. As a student, I should be able to only view the courses I am enrolled in.
- 11. Student can view edit button & Add course button
 - a. As a student, I should not be able to view the edit button and add course button.
- 12. Add delete button for updating the course form
 - a. As an instructor, I should be able to delete a course.
- 13. Allow lesson to be added into multiple courses
 - a. As an instructor, I should be able to have the same lesson in multiple courses.
- 14. R&D Heroku deployment pipeline
 - a. As a developer, I have a clear direction as to how to deploy incremental updates to MathBotics.

Modified User Stories

This story was implemented and later discarded. This was because we realized that there is no need for an administrator to view courses created by instructors. The main role of administrators should be to create lessons for instructors to use.

- 1. Add course page for admin
 - b. As an admin, I should be able to add/edit/view courses.

PROJECT PLAN

This section describes the planning that went into the realization of this project. This project incorporated the agile development techniques and as such required the sprints to be planned. These sprint plannings are detailed in the section. This section also describes the components, both software and hardware, chosen for this project.

Hardware and Software Resources

The following is a list of all hardware and software resources that were used in this project:

Software:

- Docker: [here](#)
- Operating systems: Windows 10, macOS Catalina
- Git: [here](#)
- Github
- Postgresql database
- Graphql querying tool
- Ant design: [here](#)
- Text Editors: Visual Studio Code, JetBrains
- Web Browser: Chrome, Firefox, Safari

Hardware:

Hardware	Windows	Macbook Pro	Macbook Pro
Processor	AMD Ryzen 3950x	2.6 GHz Quad-Core Intel Core i7	2.3 Ghz Quad-Core Intel Core i5
Capacity	1TB Flash Storage	250GB Flash Storage	250GB Flash Storage
Memory	64 GB RAM	16 GB RAM	8 GB RAM

Sprints Plan***Sprint 1***

1. Clone Github Repository - Story Point 1
 - a. As an engineer, I should be able to clone and push to a remote repository
 - b. As an engineer, I should be able to version control my code by branching and making pull requests.
2. Setup Trello for Engineering team - Story Point 1
 - a. As an engineer, I should see stories which are assigned to me through a kanban board
 - b. As a product manager I should be able to view all the current stories, backlog tasks and work already done.
3. Read existing documentation - Story Point 3

- a. As an engineer, I should understand the given codebase and know which developing environment to begin developing in.
- 4. Setup Development Environment - Story Point 1
 - a. As an engineer, I should be able to begin developing within an environment and install needed dependencies/tools: npm, Node, TypeScript, Docker, GraphQL.
- 5. Spike: Research and Learn Relevant Technologies - Story Point 5
 - a. As an engineer, I should be able to understand and begin developing with the following technologies: npm, Node, TypeScript, Docker, GraphQL, React, PostgreSQL, Jest.
- 6. Bug: Fix yarn install command
 - a. As an engineer, I should be able to run yarn install to set up backend
- 7. Bug: Fix stuck on loading screen
 - a. As an engineer, I should be able to immediately load screen
- 8. Bug: Fix yarn gen command
 - a. As an engineer, I should be able to generate the gql type and prisma files
- 9. Bug: Fix yarn seed command
 - a. As an engineer, I should be able to create an admin/users
- 10. Bug: Fix prisma.schema file syntax
 - a. As an engineer, I should be able to have a schema of the database
- 11. Feature: Allow Windows OS Development
 - a. As an engineer, I should be able to develop in a Windows OS environment

Sprint 2

- 1. Setup TravisCI or GH Actions integration with github - Story Point 3
 - a. As a developer, I can see that new builds are being automatically deployed and tested as

2. R&D Netlify deployment pipeline - Story Point 8
 - a. As a developer, I have a clear direction as to how to deploy incremental updates to MathBotics.
3. Implement guardian creation for students [FrontEnd] - Story Point 8
 - a. As an instructor, I can add new Guardians for my students.
4. Log out button functionality - Story Point 5
 - a. As a user, I should be able to log out from my session.
5. Fix instructor edit button - Story Point 1
 - a. As an instructor, I should be able to edit content.
6. Add course page for admin - Story Point 1
 - a. As an admin, I should be able to add/edit/view courses.
7. Polish the codebase for the frontend, apply linting, formatting, etc. - Story Point 1
 - a. As a developer, Husky prevents me from pushing unformatted, poorly linted code on the frontend, just like it does on the backend.

Sprint 3

1. Delete student information after courses ends [FrontEnd] - Story Point 8
 - a. As an instructor, I should be able to delete and edit students/guardians in my course
2. Delete student information after courses ends [BackEnd] - Story Point 8
 - a. As an engineer, I should be able to delete and edit students/guardians with a script in the graphql playground
3. Log out button functionality - Story Point 8
 - a. As a user, I should be able to log out from my session.

Sprint 4

1. Edit single student in course [Front End] - Story Point 3
 - a. As a user, I should edit a student so that I can fix incorrect info.
2. Delete single student in course [Front End] - Story Point 3
 - a. As a user, I should be able to delete a student, in case a student drops from a course.
3. Edit single student in course [Back End] - Story Point 5
 - a. As a user, I should edit a student so that I can fix incorrect info.
4. Delete single student in course [Back End] - Story Point 5
 - a. As a user, I should be able to delete a student, in case a student drops from a course.
5. Fix bug where page doesn't refresh when deleting all students - Story Point 5
 - a. As a user, I should be able to see no students in my course when I delete them all without refreshing.
6. Add alert when username already exists - Story Point 1
 - a. As a user, I should be alerted when the username I chose to input already exists in the database.
7. Log out button functionality - Story Point 8
 - a. As a user, I should be able to log out of the site.

Sprint 5

1. Connect Delete and edit student front end and back end - Story Point 8
 - a. As an Instructor, I should be able to delete and/or edit a student in my course.
2. Add a plus button to add a lesson - Story Point 3
 - a. As an instructor, I should be able to view and press a button to add a lesson, so that I can create a course for my students.

3. Adding student email to student entity in PostgreSQL - Story Point 3
 - a. As a student, I should be able to use my email to reset my password.
4. Implement forgot password [FrontEnd] - Story Point 8
 - a. As a user, I can click “forgot password” on the frontend and have a password reset link sent to the email I provide, if there is a user associated with that email.
5. Fix bug where page doesn't refresh when deleting all students - Story Point 8
 - a. As a developer, I want the page to refresh automatically when an instructor deleted a student
6. Course creation doesn't have a grade level associated with it - Story Point 3
 - a. As an instructor, I should be able to select the grade level of the course I am creating.
7. Style problems when lesson has a really long name bug - Story Point 1
 - a. As a user, the user interface should be consistent.

Sprint 6

1. Delete single student in course [Back End] - Story Point 1
 - a. As a user, I should be able to delete an individual student, in case a student drops from a course.
2. Implement forgot password [BackEnd] - Story Point 3
 - a. As a developer, I can reset someone's password with a token from GQL playground, like we do to register a new user.
3. Student can view edit button & Add course button - Story Point 5
 - a. As a student, I should not be able to view the edit button and add course button.
4. Grades page for instructor [frontend] - Story Point 3
 - a. As an instructor, I should be able to see my students' grades.
5. Grades page for students [frontend] - Story Point 3
 - a. As a student, I should be able to view my grades.

6. Search lesson by title needs to filter the lessons object to match the input value - Story Point 5
 - a. As an instructor, I should be able to search for lessons to add to my course.
7. Add delete button for updating the course form - Story Point 5
 - a. As an instructor, I should be able to delete a course.
8. Add delete button for deleting a lesson in admin view - Story Point 5
 - a. As an admin, I should be able to delete a lesson.
9. Allow lesson to be added into multiple courses - Story Point 8
 - a. As an instructor, I should be able to have the same lesson in multiple courses.
10. Add implementation to grey out lessons that have already been added to the lesson plan - Story Point 3
 - a. As an instructor, I should be able to easily identify courses that have already been added to the lesson plan.

Sprint 7

1. Final Deliverables Documentation - Story Point 8
2. Record Videos - Story Point 3
3. Create Poster - Story Point 3
4. Create Presentation Slides - Story Point 3
5. Shadow Project Wrap-up - Story Point 5
6. Approve Recorded Videos - Story Point 3

SYSTEM DESIGN

This section contains information on the design decisions that went into this project. The architecture patterns are outlined and explained. The entire system is shown in a package diagram and the subsystems are explained. Finally, the design patterns used in the project are discussed.

Architectural Patterns

The MathBotics platform has kept to the same architectural system as the previous version of the MathBotics project which is a Full Stack Application. The front-end part of the application remains the same; utilizing the React Javascript framework used to display the web application that targets both Mobile and Web platforms. The back-end side continues to use PostgreSQL database to save all data entered in the application as well as fetch data to display students and verify login information pertaining to the user. This architecture was initially decided in version one for its easy to use front-end design to make it easier for new engineers to contribute to. The architecture also took into account what back-end technologies to incorporate in order to achieve fast fetching data as well as fast modification considering large data sets.

System and Subsystem Decomposition

Front-End Architecture

The front-end of the application utilizes React, Typescript, Styled Components, and Ant Design. Previously, the design had a lack of user interface functionality which most of the sprints targeted to include. In addition to supplying functionality to sections that didn't already include functionality, more features were included to supply a smooth transition of user interface so the user could infer actions based on human computer interaction guidelines. Since this is the second iteration of the project, a great amount of logic and functionality had to be integrated into the project to consider it a fully functional product. This includes the ability to log in and out of the account. The front-end interacts with the back-end through a GraphQL layer generated by Relay. POST requests made by the front-end contain information made by Relay to provide GraphQL schema conformity of type passing to ensure the correct data being given.

The full description and explanation of the structure of the front-end folders are explained in Appendix D.

Back-End Architecture

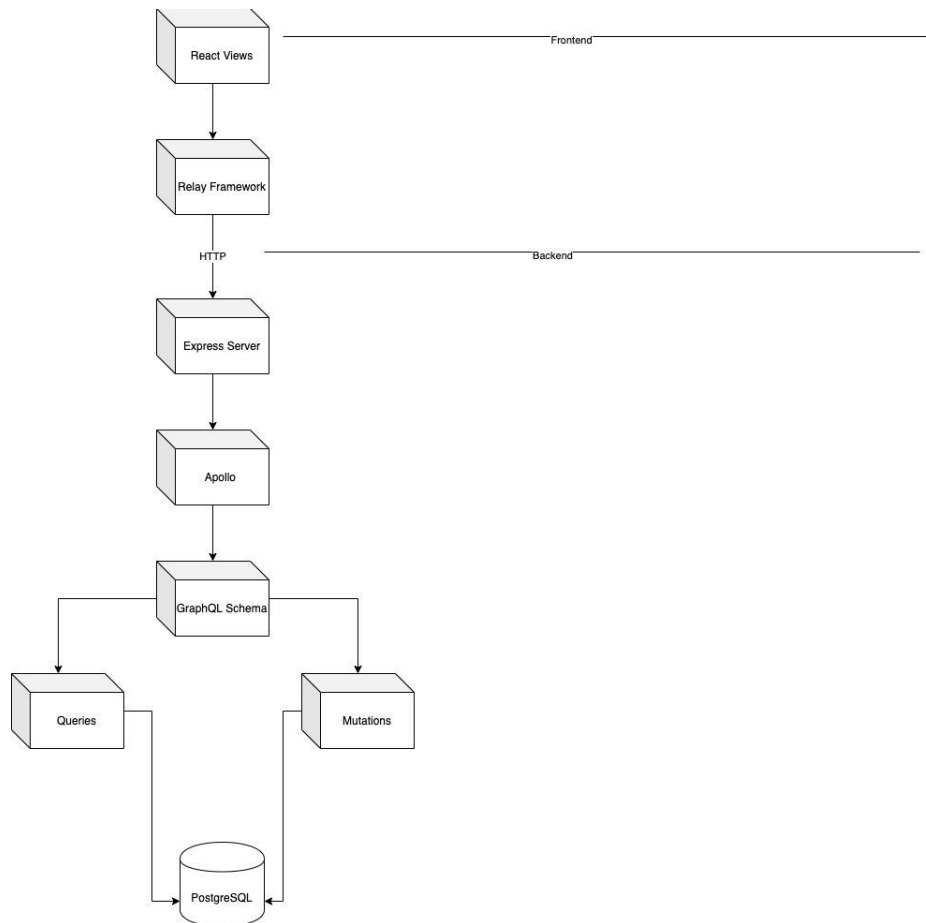
The back-end of the application utilizes Express to create the application programming interface for the front-end to communicate with the database. Typescript, GraphQL, Nexus, and Prisma were all utilized for the creation of the backend. Typescript was used to define the types of information coming in and out to identify which information to use and access. In addition to

typescript, Nexus and Prisma were used to generate mutations and queries to communicate with the postgresql database. The purpose of these technologies is to have CRUD(Create, Read, Update and Delete) functionality with minimal run time complexity and to access, modify, delete, update database information in real time. Essentially, the back-end provides all the logic for the data that is requested by the front-end.

The full description and explanation of the structure of the back-end folders are explained in Appendix D.

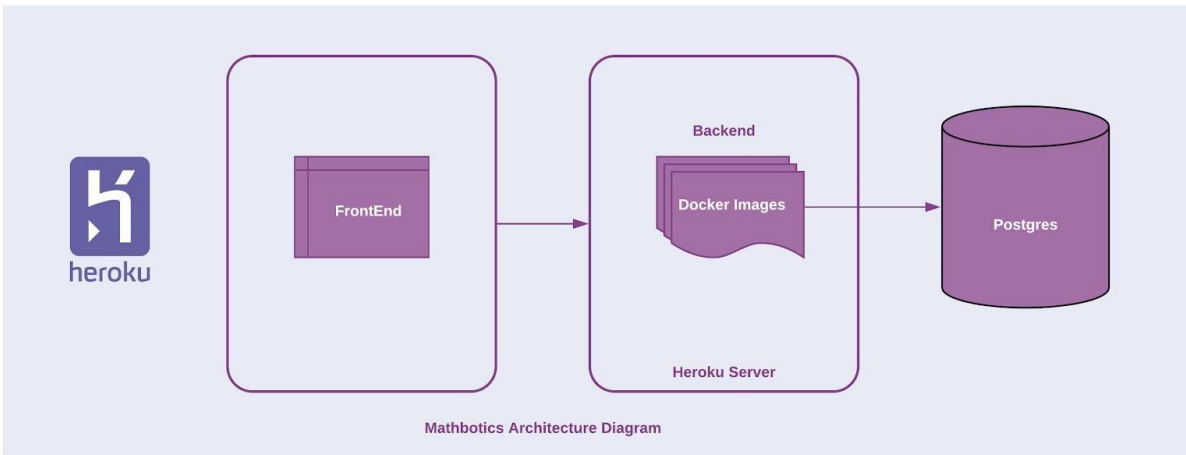
Database (PostgreSQL)

The structure of the database was not modified since version one of the MathBotics app.



Deployment Diagram

The front-end can, and should be served a static site, the back-end should be deployed using docker, which it already uses as a result of the docker-compose work that we did to make sure that future developers had a painless environment to develop on, and the database should be provided by Heroku. All the services should live under 1 team in Heroku for easy management. Below, find an example of what the deployment diagram looks like under these assumptions:



Design Patterns

- GraphQL
- Functional Programming
- Relational Data
- Layered Architecture: Models, Views, Graphs
- Singleton
- Domain Driven Design

Because this project was inherited, we continued to use the same design patterns that were utilized. We adopted the paradigms that were already included in version one of the project. Working with these design patterns, we found that they were tremendously easy and powerful to work with.

The application's front-end and back-end were built entirely using TypeScript. Giving us access to a good amount of strong well equipped libraries to help with the creation of the project.

Due to the vast amazing tools of the JavaScript and TypeScript ecosystem, it allows us to create aesthetic, functioning software. This pattern was used throughout the project structure.

Given that this application was also a GraphQL application, the design patterns were very design focused. We continued to utilize the restrictions and patterns that were created using the GraphQL framework from the previous instance of the inherited project. We also used various other frameworks such as Nexus.js, and the functional programming paradigm that TypeScript gave us access to. Utilizing GraphQL, we also followed the Domain Driven Design structure that was left from the previous students -- since GraphQL forces you to think only about the model, or domain at hand when designing the code. We didn't have to touch much of it as the foundation was solid in terms of relations.

Due to the installment of GraphQL, a Layered Architecture structure was used with the progressive installments of updates of the project. (as described in the above architecture section.)

Due to the setup of the previous project installment, Object Oriented Programming was utilized to conform with an understabled structure. Thanks to the already setup project, we inherited the singleton pattern which was used to avoid wasting expensive computations utilizing the Prisma and Nexus libraries and allowing us to simply use the same instance of the object when it was meant to be utilized multiple times. This also allowed us to reduce the computation time needed.

SYSTEM VALIDATION

Version one of the MathBotics learning platform already implemented testing tools. For this reason, the quality of code across the system has been maintained.

The following tools were implemented in version one to validate our system:

- TypeScript
- Eslint
- Husky
- Prettier
- Jest (Integration, Unit Testing)
- ApolloServerTesting (Integration Testing)
- Codegen

GLOSSARY

MathBotics: A robotics learning application designed for K-12 students. The platform will serve as a portal for Educators and students to interact with courses and learning material. It is designed for students to learn math and robotics in a fun and interactive way.

Ant design: Design principles developed for Ali Baba. Ant aims to be natural, clear, concise by relying on natural user cognition and natural user behavior to dictate where elements are most likely to be seen and enable users to quickly identify information presented. React uses this design principle.

Full Stack Application: Application that is designed to work in both front-end and back-end.

Back-End: “behind-the-scenes” design in mind. It’s main purpose is to solve tasks that are not necessary for the user to know or see.

Front-End: Design that is heavily reliant on user’s perspective and user interaction. Usually refers to user interface.

React: Open-source, front-end, JavaScript library for building web based user interfaces.

TypeScript: Programming language. It is a strict syntactical superset of JavaScript and adds optional static typing to the language. Used for this project for both back-end and front-end.

Eslint: Static code analysis tool for identifying problematic patterns found in JavaScript code.

Prettier: Code formatter tool used to enforce consistent style.

Jest: JavaScript testing framework that ensures correctness of any JavaScript codebase.

ApolloServerTesting: Package that provides tooling to make testing easier and accessible to users of all of the apollo-server integrations. This is used for Integration testing of the project.

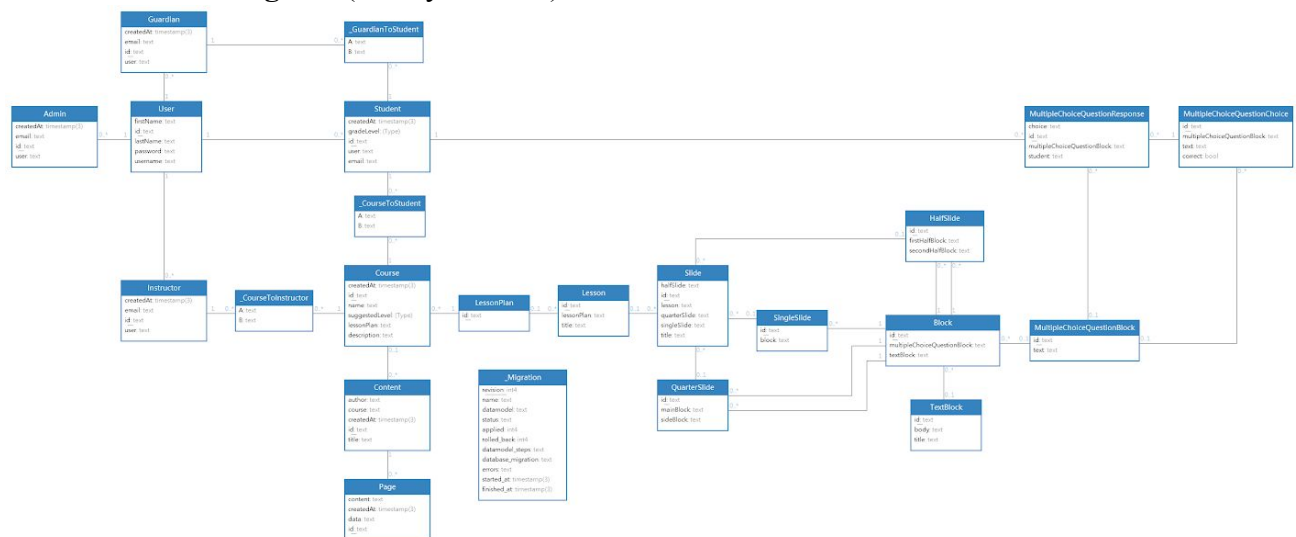
Codegen: An ECMAScript code generator from Mozilla's Parser API. Helps with testing.

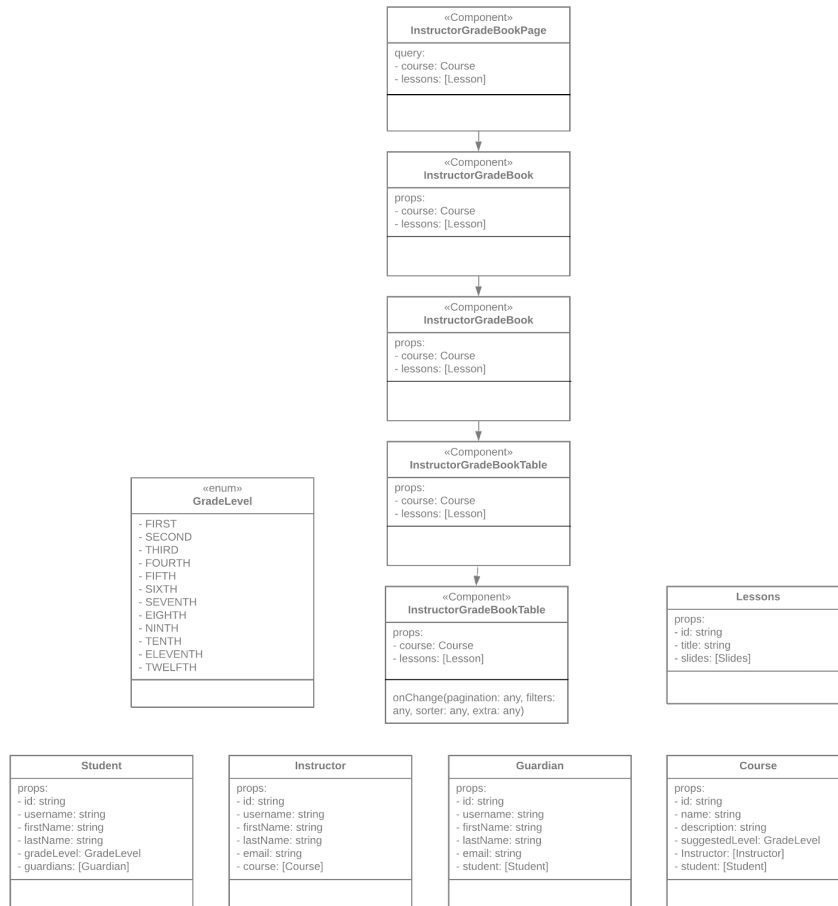
APPENDIX

Appendix A - UML Diagrams

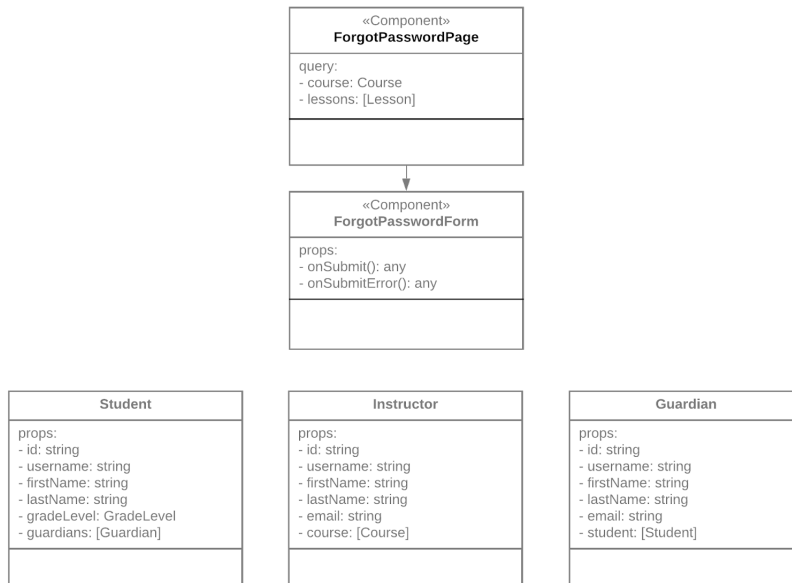
Static UML Diagrams

Database Class Diagram (Martyn Mallo)

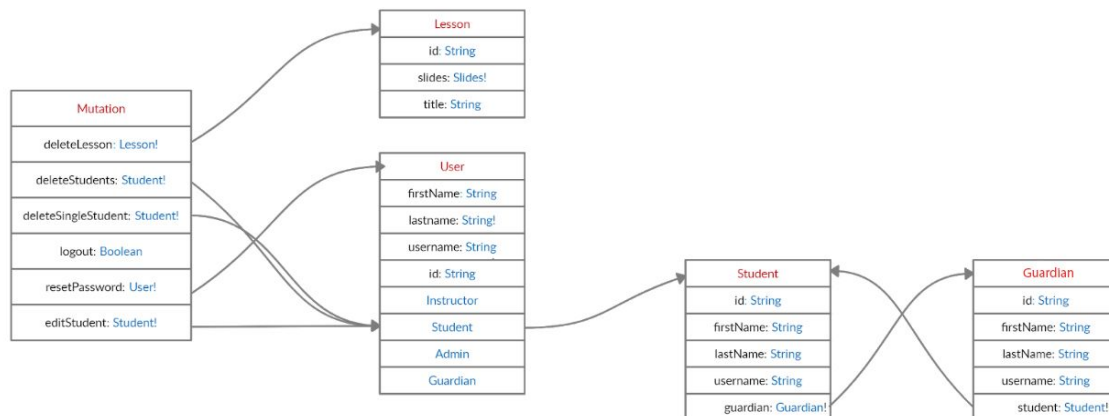


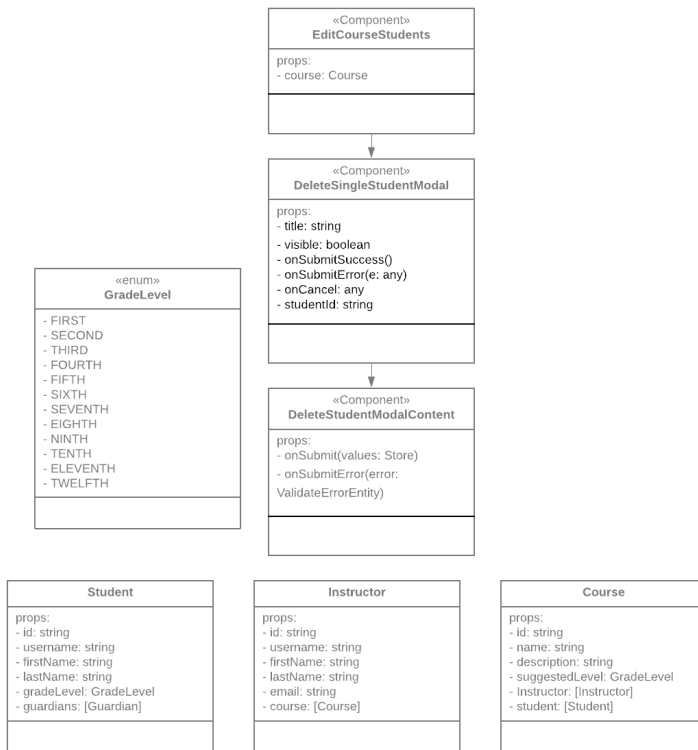
Front-End InstructorGradeBookPage Class Diagram (Jessica Moreno)

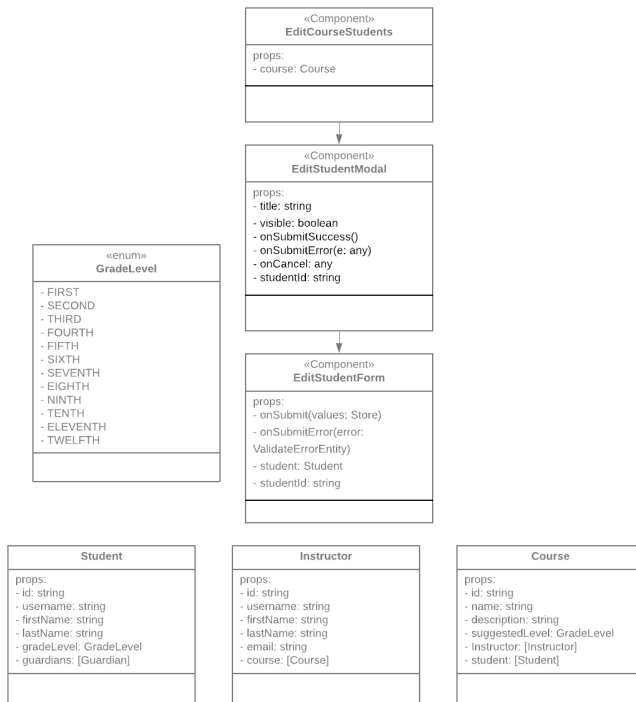
Forgot Password Class Diagram (Jessica Moreno)

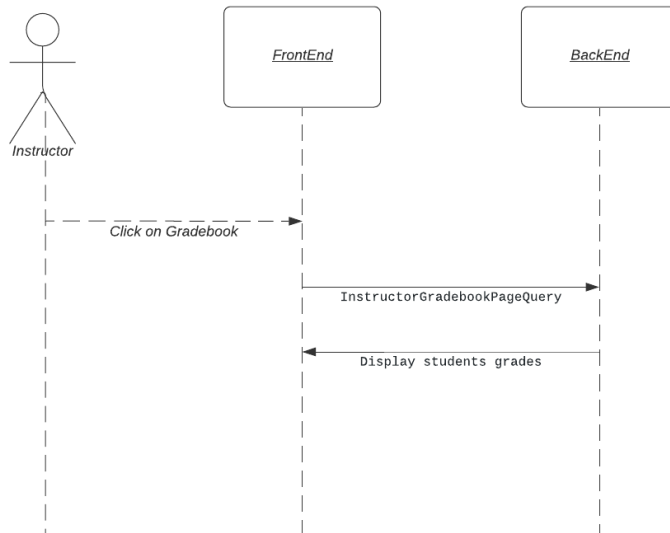
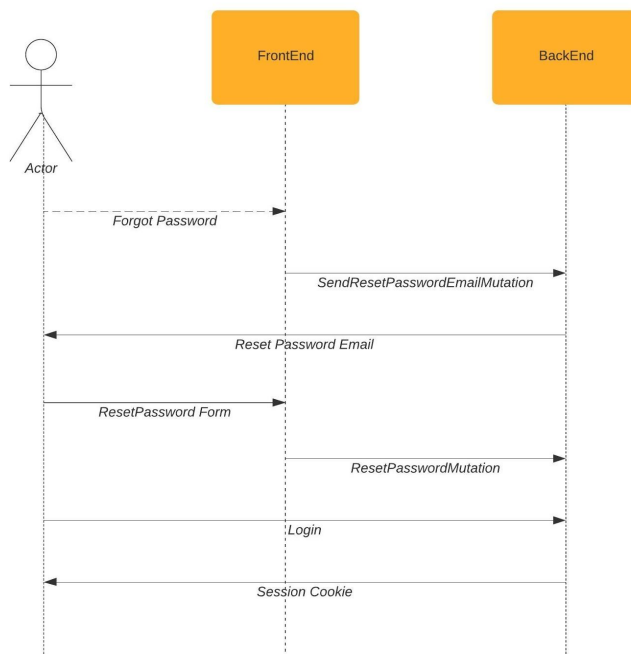


Backend Class Diagram (GraphQL Schema Diagram of newly implemented mutations) (Carlos A Barbachano)

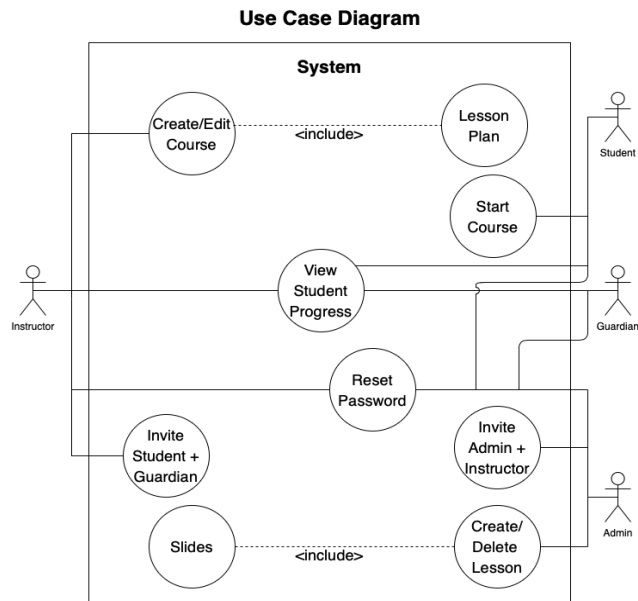


Delete Student Information Class Diagram (Jessica Moreno)

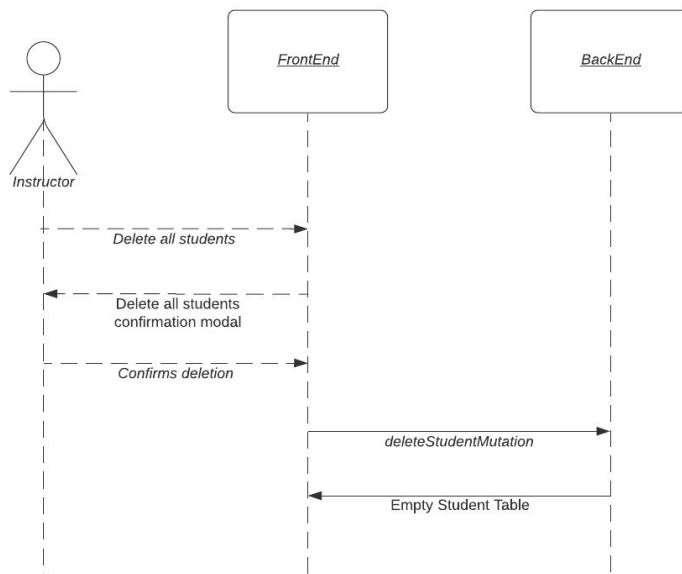
Edit Student Class Diagram (Jessica Moreno)

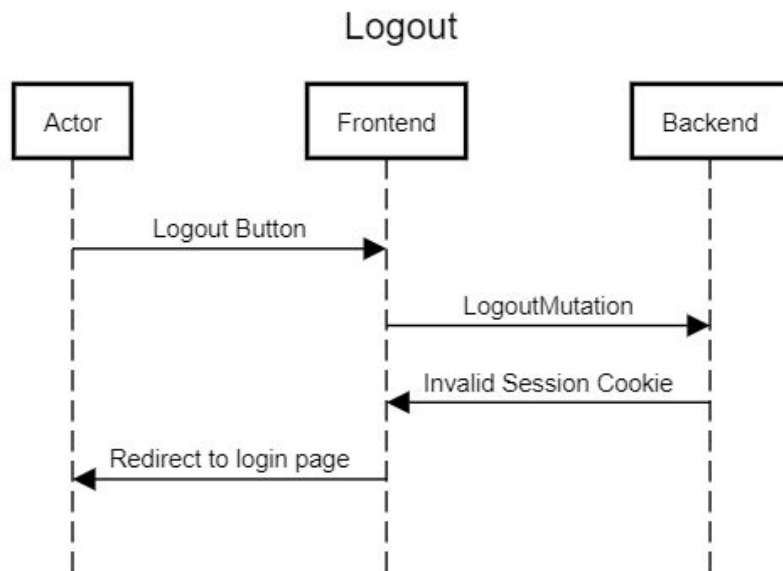
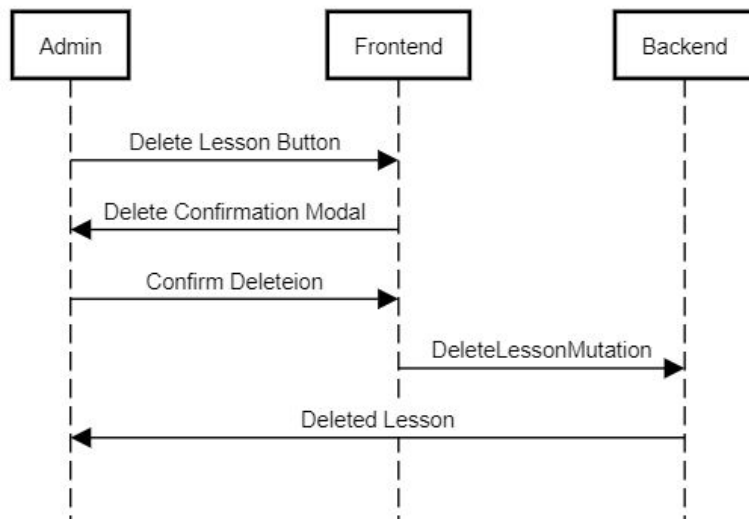
Dynamic UML Diagrams**InstructorGradeBookPage Sequence Diagram (Martyn Mallo)****Forgot Password Sequence Diagram (Jessica Moreno)**

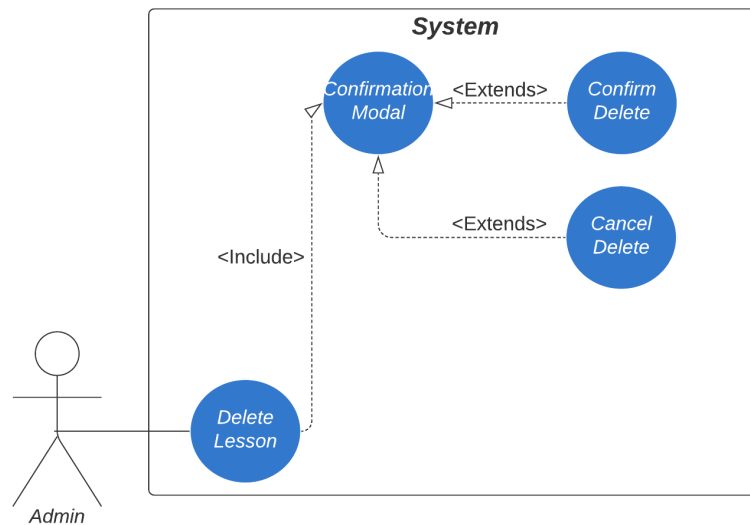
Use Case Diagram (Martyn Mallo, Jessica Moreno, Carlos A. Barbachano)



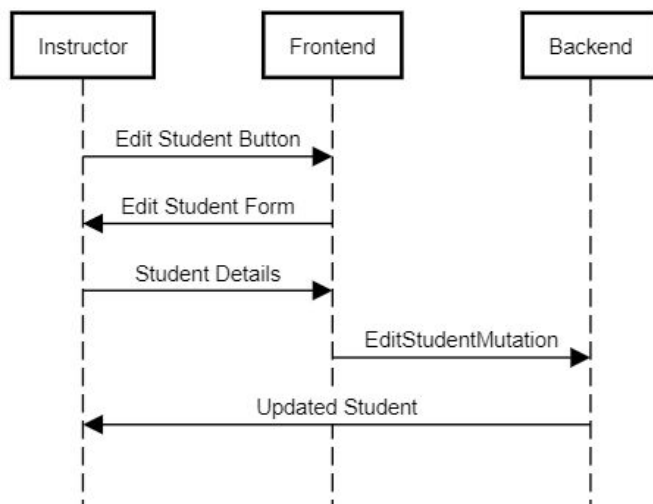
Delete Student Information Sequence Diagram (Martyn Mallo)

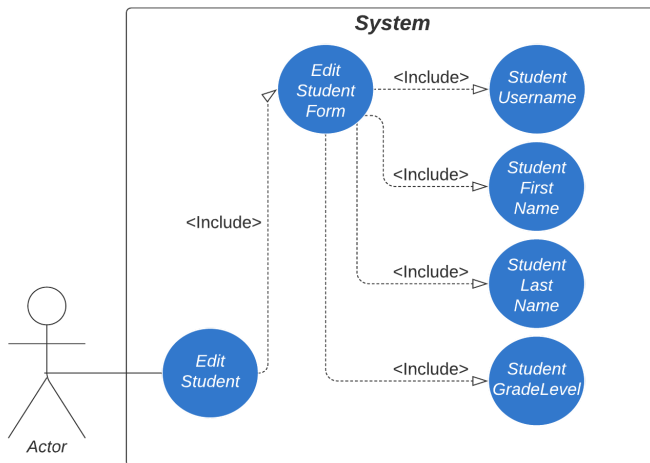


Logout Sequence Diagram (Carlos A. Barbachano)**Admin Delete Lesson Sequence Diagram (Carlos A. Barbachano)**

Admin Delete Lesson Use Case Diagram (Jessica Moreno)**Edit Student Sequence Diagram (Carlos A. Barbachano)**

Instructor: Edit Student

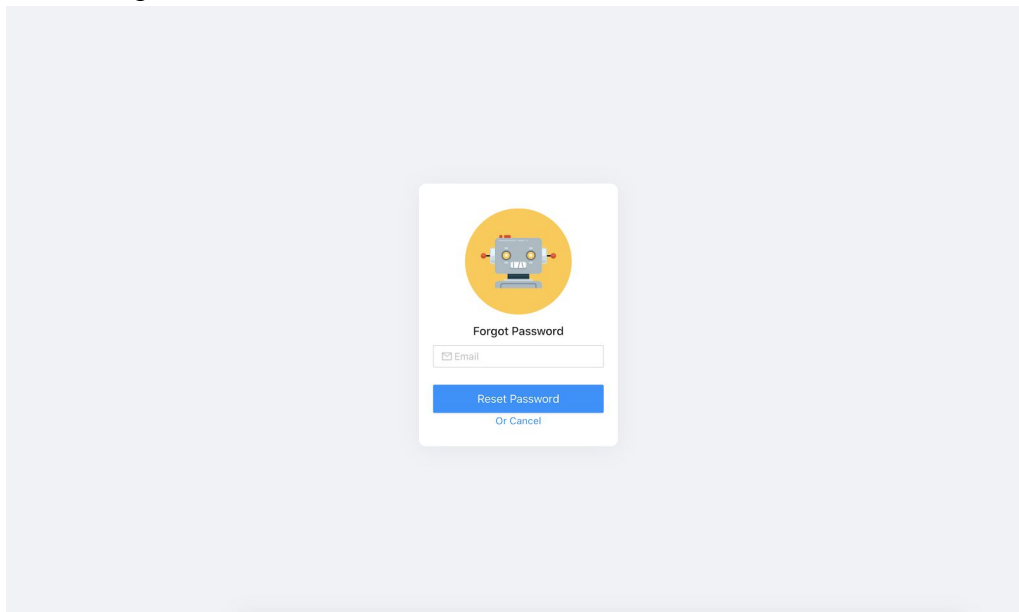


Edit Student Use Case Diagram (Jessica Moreno)

Appendix B - User Interface Design

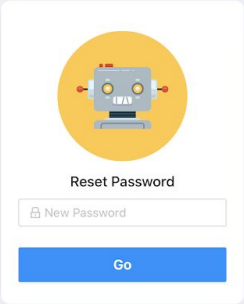
Forgot Password Page:

A user can click forgot password and enter this page where they can input their email address to reset their password.



Reset Password Page:

After receiving the email link to reset their password, the user will be sent to this page where they can input their new password.

A screenshot of a 'Reset Password' form. The form is a white card centered on a light gray background. At the top of the card is a circular logo with a yellow background and a gray robot head. Below the logo is the text 'Reset Password'. Underneath is a text input field with a small key icon and the placeholder text 'New Password'. At the bottom of the card is a blue button with the text 'Go' in white.

Reset Password

New Password

Go

Delete and Edit Students:

Instructors can now delete or edit a single student by clicking the pencil or trash in the corresponding student row. In addition, Instructors can delete all students in a course to preserve FERPA rights when a course ends.

Robotics 101

Lesson Plan **Students** Course Details

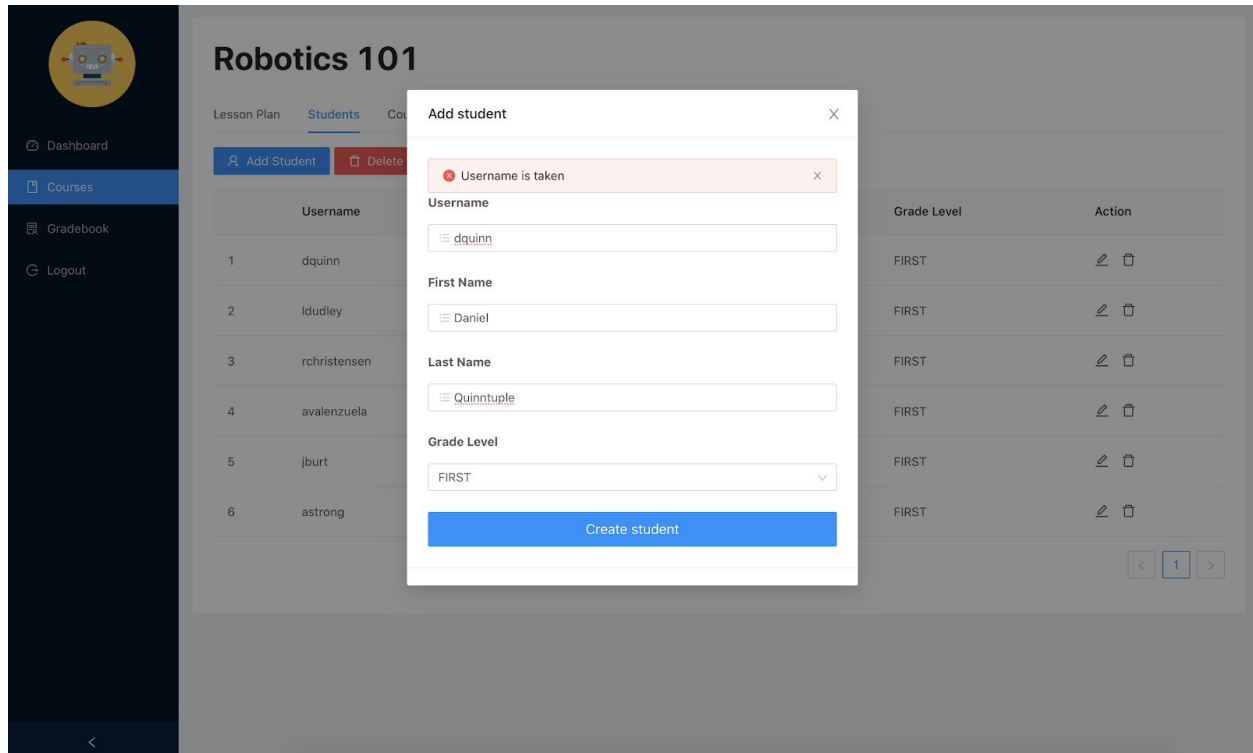
[Add Student](#) [Delete All Students](#)

	Username	First Name	Last Name	Grade Level	Action
1	dquinn	Daniella	Quinn	FIRST	Edit Delete
2	ldudley	Lottie	Dudley	FIRST	Edit Delete
3	rchristensen	Rudi	Christensen	FIRST	Edit Delete
4	avalenzuela	Ariana	Valenzuela	FIRST	Edit Delete
5	jburt	Johnathon	Burt	FIRST	Edit Delete
6	astrong	Arnie	Strong	FIRST	Edit Delete

< 1 >

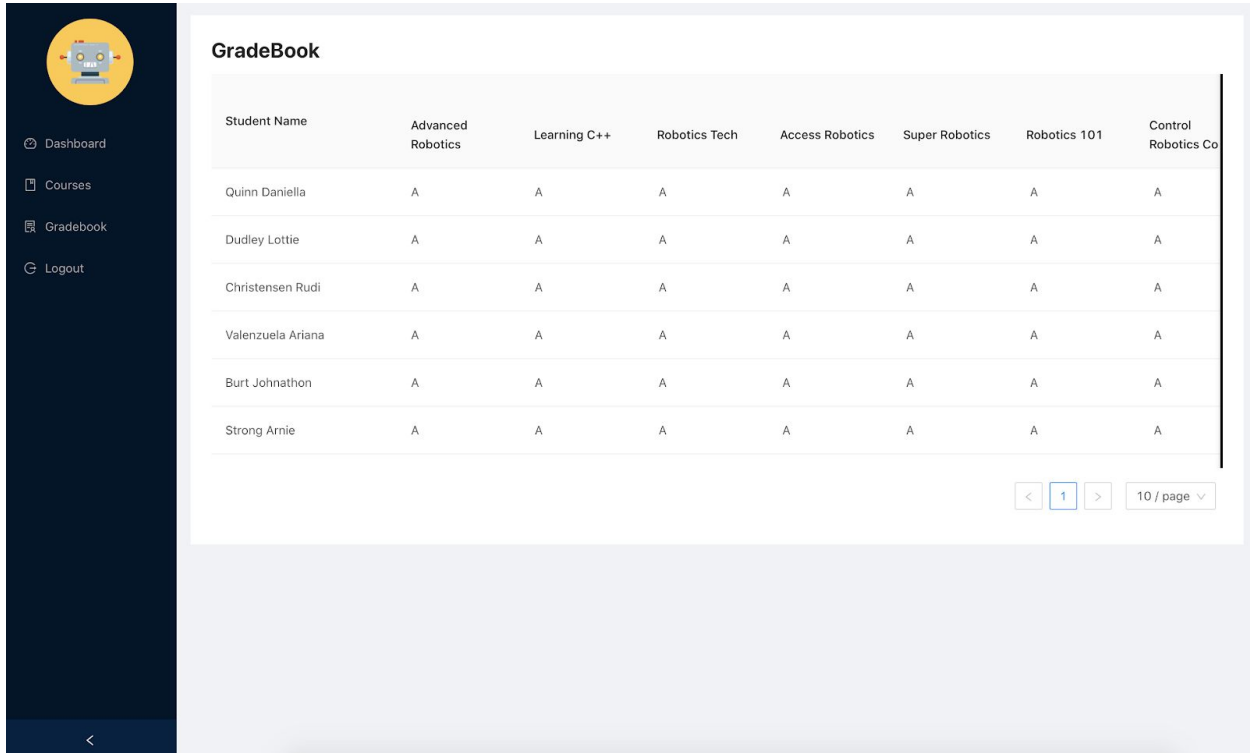
Username Taken:

The instructor gets an alert if a student username is taken.



Instructor Gradebook Page:

The instructor can check students progress using the gradebook page.

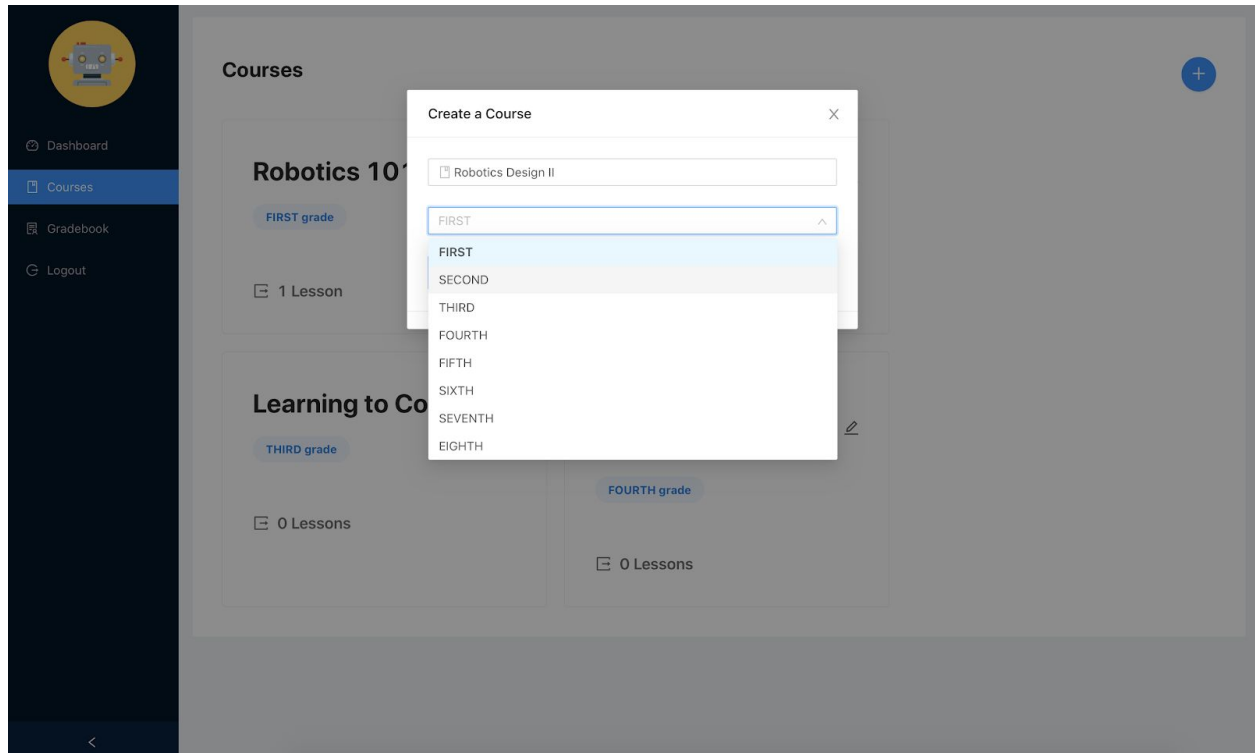


Student Name	Advanced Robotics	Learning C++	Robotics Tech	Access Robotics	Super Robotics	Robotics 101	Control Robotics Co
Quinn Daniella	A	A	A	A	A	A	A
Dudley Lottie	A	A	A	A	A	A	A
Christensen Rudi	A	A	A	A	A	A	A
Valenzuela Ariana	A	A	A	A	A	A	A
Burt Johnathon	A	A	A	A	A	A	A
Strong Arnie	A	A	A	A	A	A	A

< 1 > 10 / page ▾

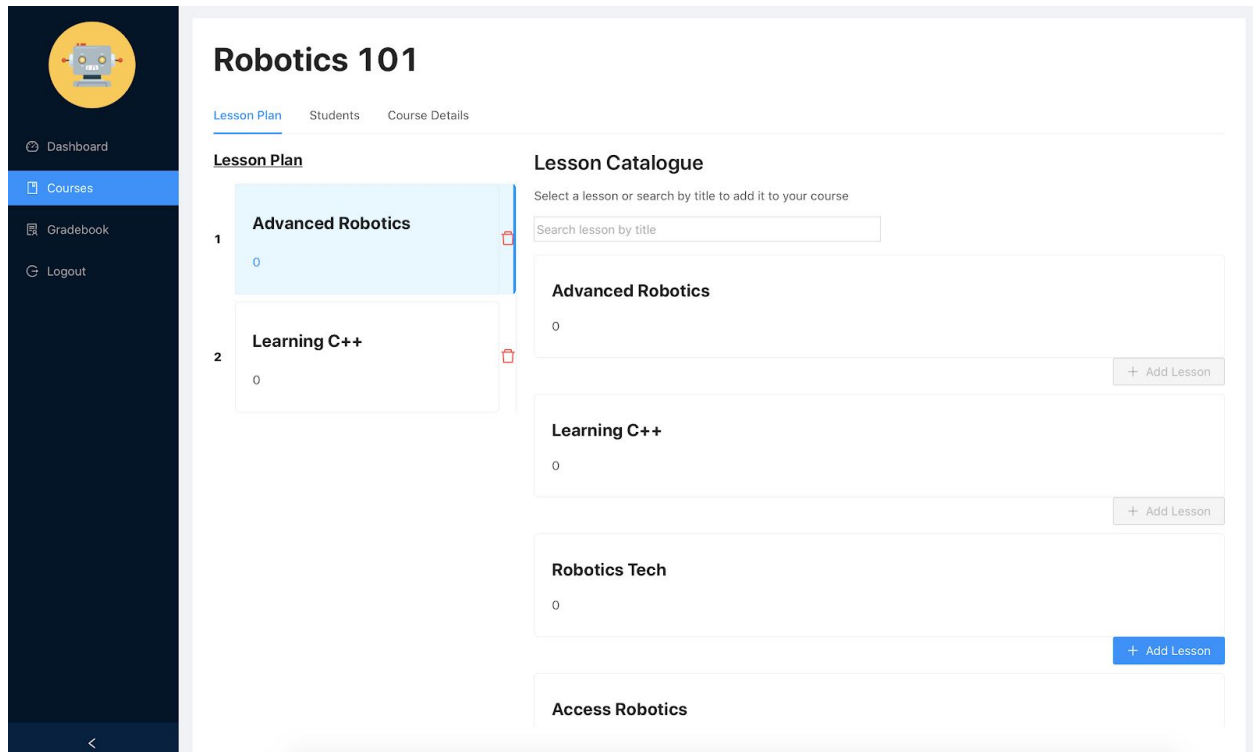
Creating a Course:

Instructors can now choose a grade level when adding a course.



Lesson Plan Page:

Lessons in the lesson catalogue are now greyed out when added to a course lesson plan to allow for a better user experience.



Search Lesson:

Instructors can now search lessons by title.

Robotics 101

[Lesson Plan](#) [Students](#) [Course Details](#)

Lesson Plan

1 **Learning C++** 0

2 **Advanced Robotics** 0

Lesson Catalogue

Select a lesson or search by title to add it to your course

Advanced

Advanced Robotics Intro

0 + Add Lesson

Advanced Robotics

0 + Add Lesson

Lessons Page:

Admins can now delete a lesson.

Successfully created a lesson

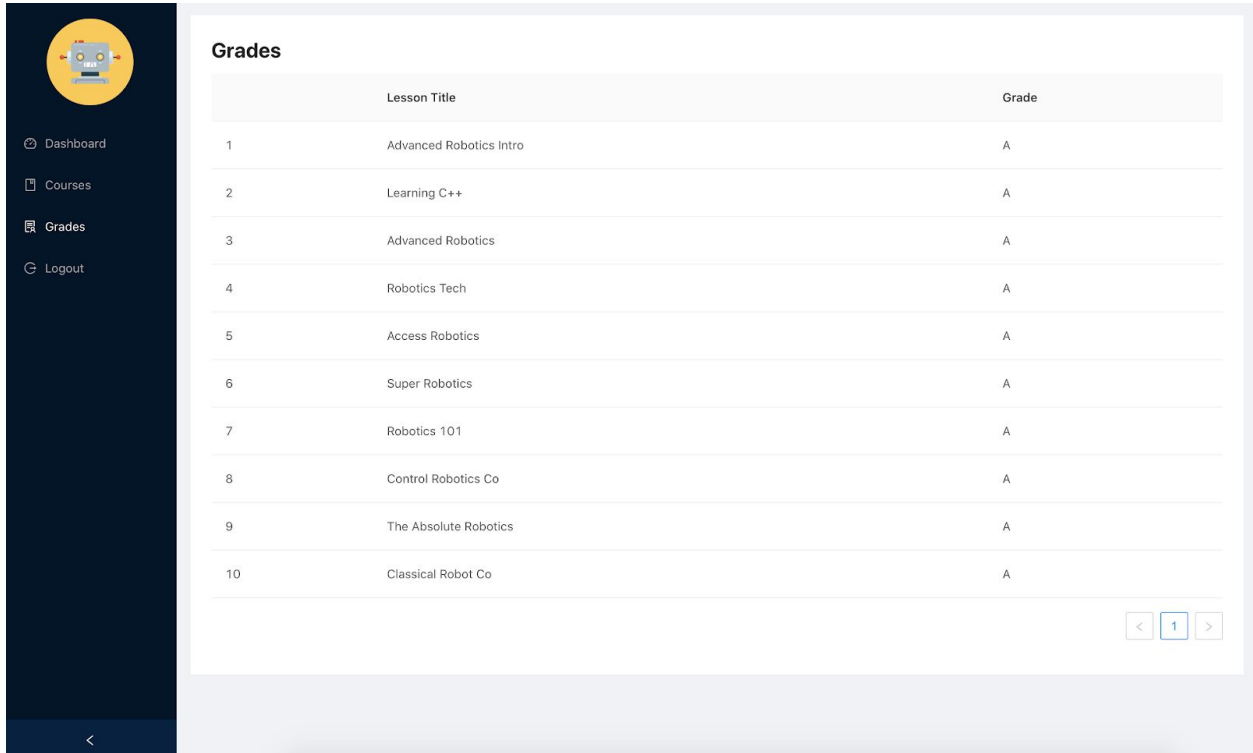
Lessons

	Lessons Title	Number of slides	Action
1	Advanced Robotics	0	Delete Lesson
2	Learning C++	0	Delete Lesson
3	Robotics Tech	0	Delete Lesson
4	Access Robotics	0	Delete Lesson
5	Super Robotics	0	Delete Lesson
6	Robotics 101	0	Delete Lesson
7	Control Robotics Co	0	Delete Lesson
8	The Absolute Robotics	0	Delete Lesson
9	Classical Robot Co	0	Delete Lesson
10	Advanced Robotics Intro	0	Delete Lesson

< 1 >

Student Grades Page:

Students can now view their grades for each lesson.



	Lesson Title	Grade
1	Advanced Robotics Intro	A
2	Learning C++	A
3	Advanced Robotics	A
4	Robotics Tech	A
5	Access Robotics	A
6	Super Robotics	A
7	Robotics 101	A
8	Control Robotics Co	A
9	The Absolute Robotics	A
10	Classical Robot Co	A

Appendix C - Sprint Review Reports

Sprint 1

FA20 Team 12 Sprint Review 09/14/20

List of Approved User Stories:

1. Clone Github Repository
 - a. As an engineer, I should be able to clone and push to a remote repository
 - b. As an engineer, I should be able to version control my code by branching and making pull requests.
2. Setup Trello for Engineering team
 - a. As an engineer, I should see stories which are assigned to me through a kanban board
 - b. As a product manager I should be able to view all the current stories, backlog tasks and work already done.
3. Read existing documentation
 - a. As an engineer, I should understand the given codebase and know which developing environment to begin developing in.
4. Setup Development Environment
 - a. As an engineer, I should be able to begin developing within an environment and install needed dependencies/tools: npm, Node, TypeScript, Docker, GraphQL.
5. Spike: Research and Learn Relevant Technologies
 - a. As an engineer, I should be able to understand and begin developing with the following technologies: npm, Node, TypeScript, Docker, GraphQL, React, PostgreSQL, Jest.
6. Bug: Fix yarn install command

- a. As an engineer, I should be able to run yarn install to set up backend
- 7. Bug: Fix stuck on loading screen
 - a. As an engineer, I should be able to immediately load screen
- 8. Bug: Fix yarn gen command
 - a. As an engineer, I should be able to generate the gql type and prisma files
- 9. Bug: Fix yarn seed command
 - a. As an engineer, I should be able to create an admin/users
- 10. Bug: Fix prisma.schema file syntax
 - a. As an engineer, I should be able to have a schema of the database
- 11. Feature: Allow Windows OS Development
 - a. As an engineer, I should be able to develop in a Windows OS environment

List of Rejected User Stories:

None

Sprint 2

FA20 Team 12 Sprint Review 09/25/20

List of Approved User Stories:

- 1. Setup TravisCI or GH Actions integration with Github - Story Point 3
 - a. As a developer, I can see that new builds are being automatically deployed and tested as
- 2. Implement guardian creation for students [FrontEnd] - Story Point 8
 - a. As an instructor, I can add new Guardians for my students.
- 3. Fix instructor edit button - Story Point 1
 - a. As an instructor, I should be able to edit content.

4. Polish the codebase for the frontend, apply linting, formatting, etc. - Story Point 1
 - a. As a developer, Husky prevents me from pushing unformatted, poorly linted code on the frontend, just like it does on the backend.

List of Rejected User Stories:

1. Add course page for admin - Story Point 1
 - a. As an admin, I should be able to add/edit/view courses.
2. Log out button functionality - Story Point 5
 - a. As a user, I should be able to log out from my session.
3. R&D Netlify deployment pipeline - Story Point 8
 - a. As a developer, I have a clear direction as to how to deploy incremental updates to MathBotics.

Sprint 3

FA20 Team 12 Sprint Review 10/09/20

List of Approved User Stories:

1. Delete student information after courses ends [FrontEnd] - Story Point 8
 - a. As an instructor, I should be able to delete and edit students/guardians in my course
2. Delete student information after courses ends [BackEnd] - Story Point 8
 - a. As an instructor, I should be able to delete and edit students/guardians in my course

List of Rejected User Stories:

1. Log out button functionality - Story Point 8

- a. As a user, I should be able to log out from my session.

Sprint 4

FA20 Team 12 Sprint Review 10/23/20

List of Approved User Stories:

1. Edit single student in course [Front End] - Story Point 3
 - a. As a user, I should edit a student so that I can fix incorrect info.
2. Delete single student in course [Front End] - Story Point 3
 - a. As a user, I should be able to delete a student, in case a student drops from a course.
3. Edit single student in course [Back End] - Story Point 5
 - a. As a user, I should edit a student so that I can fix incorrect info.
4. Delete single student in course [Back End] - Story Point 5
 - a. As a user, I should be able to delete a student, in case a student drops from a course.
5. Add alert when username already exists - Story Point 1
 - a. As a user, I should be alerted when the username I chose to input already exists in the database.
6. Log out button functionality - Story Point 8
 - a. As a user, I should be able to log out of the site.

List of Rejected User Stories:

1. Fix bug where page doesn't refresh when deleting all students - Story Point 5
 - a. As a user, I should be able to see no students in my course when I delete them all without refreshing.

Sprint 5**FA20 Team 12 Sprint Review 11/06/20**

List of Approved User Stories:

1. Connect Delete and edit student front end and back end - Story Point 8
 - a. As an Instructor, I should be able to delete and/or edit a student in my course.
2. Add a plus button to add a lesson - Story Point 3
 - a. As an instructor, I should be able to add a lesson, so that I can create a course for my students.
3. Adding student email to student entity in PostgreSQL - Story Point 3
 - a. As a student, I should be able to use my email to reset my password.
4. Implement forgot password [FrontEnd] - Story Point 8
 - a. As a user, I can click forgot password on the frontend and have a password reset link sent to the email I provide, if there is a user associated with that email.
5. Fix bug where page doesn't refresh when deleting all students - Story Point 8
 - a. As a developer, I want the page to refresh automatically when an instructor deleted a student
6. Course creation doesn't have a grade level associated with it - Story Point 3
 - a. As an instructor, I should be able to select the grade level of the course I am creating.
7. Style problems when lesson has a really long name bug - Story Point 1
 - a. As a user, the user interface should be consistent.

List of Rejected User Stories:

None

Sprint 6**FA20 Team 12 Sprint Review 11/20/20**

List of Approved User Stories:

1. Delete single student in course [Back End] - Story Point 1
 - a. As a user, I should be able to delete a student, in case a student drops from a course.
2. Implement forgot password [BackEnd] - Story Point 3
 - a. As a developer, I can reset someone's password with a token from GQL playground, like we do to register a new user.
3. Grades page for instructor [frontend] - Story Point 3
 - a. As an instructor, I should be able to see my students' grades.
4. Grades page for students [frontend] - Story Point 3
 - a. As a student, I should be able to view my grades.
5. Search lesson by title needs to filter the lessons object to match the input value - Story Point 5
 - a. As an instructor, I should be able to search for lessons to add to my course.
6. Add delete button for deleting a lesson in admin view - Story Point 5
 - a. As an admin, I should be able to delete a lesson.
7. Add implementation to grey out lessons that have already been added to the lesson plan - Story Point 3
 - a. As an instructor, when I should be able to easily identify courses that have already been added to the lesson plan.

List of Rejected User Stories:

1. Allow lesson to be added into multiple courses - Story Point 8
 - a. As an instructor, I should be able to have the same lesson in multiple courses.
2. Student can view edit button & Add course button - Story Point 5
 - a. As a student, I should not be able to view the edit button and add course button.
3. Add delete button for updating the course form - Story Point 5
 - a. As an instructor, I should be able to delete a course.

Sprint 7**FA20 Team 12 Sprint Review 12/04/20**

List of Approved User Stories:

7. Final Deliverables Documentation - Story Point 8
8. Record Videos - Story Point 3
9. Create Poster - Story Point 3
10. Create Presentation Slides - Story Point 3
11. Shadow Project Wrap-up - Story Point 5
12. Approve Recorded Videos - Story Point 3

List of Rejected User Stories:

None

Appendix D - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and other documents

Documentation for the Front-End of MathBotics

Front-End Project Structure:

The MathBotics FrontEnd is a React based frontend and was created using CRA(CreateReactApp), therefore it has a familiar folder structure.

- public: Contains index.html which will serve up the page rendering a single element by the id of "root"
- src: Where the react code lives
 - components: React Components used within the MathBotics application
 - block: Components related to the blocks
 - course: Components related to the course/courses
 - dashboard: Components related to the dashboard
 - form: Where most of the form components are stored
 - grades: Components related to Instructor Gradebook and Student Grades
 - hoc: Any Higher Order Components should go here ie. withSidebar hoc - icons: Contains icons related to MathBotics
 - lessons: Components related to lessons/lesson
 - slides: Components related to slides/slide
 - students: Components related to students/student
 - graphql: Relay configuration and mutations
 - mutations: Mutations used by components

- pages: Contains the entry point to all the page components served up by MathBotics application
- routes: Contains logic for specific routes ie. ProtectedRoute = logic for a protected route - types: Contains the types which aren't defined by a library or the application
- utils: Helper functions

FrontEnd Installation Guide / Environment Setup:

- Use the docker compose in /docker file to start up the frontend + other related services

FrontEnd User Manual Docs:

- Generating gql type: If you update or add to any graphql query or mutation on the frontend you will have to generate the types, run `yarn relay` at the root of the /frontend directory

NOTE: Version Two of MathBotics should not have an infinite loading screen issue. However, in the case that you encounter this problem, remove the cookie on the browser (CMD + SHFT + C -> Application -> cookies -> clear)

Application Deployment:

We were not able to deploy the application, however the following information was left over from version one of the project.

- Since this is a react application there are tons of guides on packaging and deploying: Ref <https://blog.heroku.com/deploying-react-with-zero-configuration>

NOTE: docker-compose is only meant for developing locally, however you can deploy the application using the compose

(<https://www.docker.com/blog/how-to-deploy-on-remote-docker-hosts-with-docker-compose/>).

We recommend loosely coupling all the services. This means deploying frontend as static page and deploying backend using docker on herokus docker deployment

(<https://devcenter.heroku.com/categories/deploying-with-docker>). Having Postgres hosted on heroku (<https://www.heroku.com/postgres>) and separate from where the backend/frontend are running. There is a deployment diagram you can check out that we included somewhere in all these files/folders

FrontEnd Shortcoming:

- Fix styling for the sidebar which edits a block
- Add drag and drop functionality for adding a lesson to your lesson plan (for course)
- Add progress on courses page to show students how they are doing, same thing for the course page
- Adding a guardian to a student
- Getting all courses attached to a student and displaying those to the guardian
- Initial dashboard page, display progress, stats, current courses etc..
- Edit lesson title

- Course Catalogue selection gets reset despite being in the lesson plan
- Instant reloading of components when deleting students/lessons
- Add indicator for block when editing a slide
- Change Dashboard page depending on user role
- Off-centered delete and add lesson buttons for instructor when picking lessons

Documentation for the back-end of MathBotics

Back-End Project Structure:

The MathBotics backend is a GraphQL API with ApolloServer to expose the graphql route and handle graphql parsing,

- prisma: Contains the prisma.schema and all the migrations (explanation in the user manual).
- scripts: Contains index.html which will serve up the page rendering a single element by the id of "root"
- src:
 - data: Where the services relating to data retrieval exist.
 - graphql: Where things relating to graphql reside, this is where the "endpoints" and objects reside.
 - context: The context type resides here -- the context is passed with every single request to the resolve function of every single field.
 - mutations: These are fields that change the state of the application (they "do" things rather than "get" things).

- objects: These are the objects, these are essentially they "types" that can be returned by mutations or queries, or be used as fields in other "types". They may also have computed fields (fields with resolve() functions).
- queries: These fields are pure functions that simply return the applications state or something constant (these "get" things rather than "do" things).
- provider: The services reside here, APIs for interacting with external libraries or application to simplify the code within the graphql resolvers.
- server: The server configuration and bootstrapping occurs here.
 - middlewares: These process requests either before or after they are passed to the main application logic (the stuff in the graphql folder)
 - express: These process the request at a HTTP level (i.e. Headers).
 - passport.ts: This checks the request for an Authentication token in the cookies, these tokens contain the userId, it then finds the correct user and attaches it to the context.viewer to be used in the resolvers of all fields.
 - graphql: These process the request at a GraphQL level (after the graphql has been validated and parsed).
 - authentication.ts: This grabs the user returned from the logIn mutation and attaches its userId to the request's cookies to be fetched by the passport middleware.
 - permissions.ts: This uses GraphQL shield to set field-level permissions. At the time of writing, there are no permissions here, so the entire API may be accessed by any authenticated actor, this must be changed to only allow access to certain fields to certain actors.

- factories: Code to create database entities.
- seeders: Scripts to run code that creates database entities.
- tests: Code to test other code. Run yarn test to run the tests.
 - integration: Code to test the entire feature of the backend from top to bottom (query/mutation to response).
 - unit: Code to test specific functions of the application. Please read <https://testing.googleblog.com/2014/03/testing-on-toilet-what-makes-good-test.html> and abide by it before writing tests. Bad tests are worse than no tests. In general, a unit test should test a SINGLE path down a function, try to maintain this for integration tests as well.

BackEnd Shortcoming:

- Add backend login for delete course button
- Fix bug where lessons can't be in multiple courses
- Instructors can see other instructor's courses
- Students can see all courses when they should only see the ones they are enrolled in
- Add in lesson grades for students to be shown in gradebook

Installation Guide / Environment Setup:

- Use the docker compose in /docker file to start up all services.

User Manual Docs:

- Before creating mutations or queries: We use nexus-prisma-plugin, which exposes very basic "CRUD", or Create-Read-Update-Delete, mutations and queries to reduce the

amount of boilerplate code. When you find yourself needing to make a mutation or query involving some CRUD action, please first check to see that you cannot simply generate it using this plugin. (There are examples of doing this in the queries/index.ts as well as the mutations/index.ts in the extendType portion)

- Generating gql type: If you add any mutation, query, or object, make sure that it is exported at the queries/index.ts or the objects/index.ts or the mutations/index.ts. Afterwards, run yarn gen in the backend folder.
- Migrating Database (Prisma): When making changes to the schema, edit the prisma.schema in the prisma folder. Afterwards, docker exec into the running api container (via docker exec -it mathbotics_api bash) and run yarn migrate:save, fill the required info, this will create a new migration with the changes made to the schema, then run yarn migrate:up to apply that migration. Then, on your own machine, run yarn gen in the backend folder to generate a new version of the prisma client reflecting the new changes.
- Seeding, creating users: Run yarn seed src/seeder/AdminSeeder.ts to create an admin, or for other types as well.

Deployment:

We were not able to get to deploy the application, but these are the instructions that were left over from version one of the project:

- Here are some Heroku guides to deploying the application, the former may be more interesting since we can leverage the Dockerfiles we've created already
<https://devcenter.heroku.com/categories/deploying-with-docker>,
<https://devcenter.heroku.com/articles/deploying-nodejs>

- **NOTE:** docker-compose is only meant for developing locally, however you can deploy the application using the compose (<https://www.docker.com/blog/how-to-deploy-on-remote-docker-hosts-with-docker-compose/>). We recommend loosely coupling all the services. This means deploying frontend as a static page and deploying backend using docker on heroku's docker deployment (<https://devcenter.heroku.com/categories/deploying-with-docker>). Having Postgress hosted on heroku (<https://www.heroku.com/postgres>) and separate from where the backend/frontend are running. There is a deployment diagram you can check out that is included somewhere in all these files/folders

REFERENCES

1. <https://graphql.org/>
2. <https://graphql.org/learn/>
3. <https://www.typescriptlang.org/>
4. <https://www.postgresql.org/>
5. <https://www.nexusjs.org/>
6. <https://relay.dev/>
7. <https://restfulapi.net/>
8. <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>
9. <https://graphql.org/learn/schema/>
10. <https://ant.design/components/overview/>
11. <https://reactjs.org/>
12. <https://styled-components.com/>
13. <https://www.docker.com/>
14. <https://github.com/graphql/graphiql>