**Software Engineering 14:332:452**

Group: 9

Project Title: Minerva

Project Site URL: <https://rutgerssoftwareengineering.github.io/Minerva/>

Date Submitted: March 17, 2019

Team Members: Jacob Battipaglia, Jonathan Hong, Skyler Lee, Brian Ma, Justin May, Salman Omer, Joshua Olazo, Phurushotham Shekar

# 

# 

# Individual Contributions Breakdown

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Part | Subpart | Jacob | Jon | Skyler | Brian | Justin | Salman | Joshua | Phuru |
| Interaction Diagrams |  | 11.1% | 22.2% | 11.1% | 11.1% |  | 22.2% | 11.1% | 11.1% |
| Class Diagram and Interface Specification | Class Diagram |  |  |  | 100% |  |  |  |  |
| Data Types and Operation Signatures |  |  |  | 100% |  |  |  |  |
| Traceability Matrix |  |  |  |  |  |  |  | 100% |
| System Architecture and System Design | Architectural Styles |  |  |  |  |  |  | 100% |  |
| Identifying Subsystems |  | 100% |  |  |  |  |  |  |
| Mapping Subsystems to Hardware |  |  |  |  |  |  | 100% |  |
| Persistent Data Storage | 100% |  |  |  |  |  |  |  |
| Network Protocol |  |  |  |  |  | 100% |  |  |
| Global Control Flow |  |  |  |  |  | 100% |  |  |
| Hardware Requirements |  |  |  |  |  | 33.3% | 33.3% | 33.3% |
| Algorithms and Data Structures | Algorithms |  |  |  |  | 50% |  |  | 50% |
| Data Structures | 30% |  |  |  | 30% |  | 10% | 30% |
| User Interface Design and Implementation |  |  |  |  |  |  | 100% |  |  |
| Design of Tests |  | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% |
| Project Management and Plan of Work | Merging the Contributions from Individual Team Members | 11.1% | 11.1% | 11.1% | 11.1% | 11.1% | 11.1% | 22.2% | 11.1% |
| Project Coordination and Progress Report |  |  |  | 100% |  |  |  |  |
| Plan of Work | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% | 12.5% |
| Breakdown of Responsibilities | 25% |  |  | 50% |  |  |  | 25% |
| References |  |  |  |  |  |  |  |  | 100% |

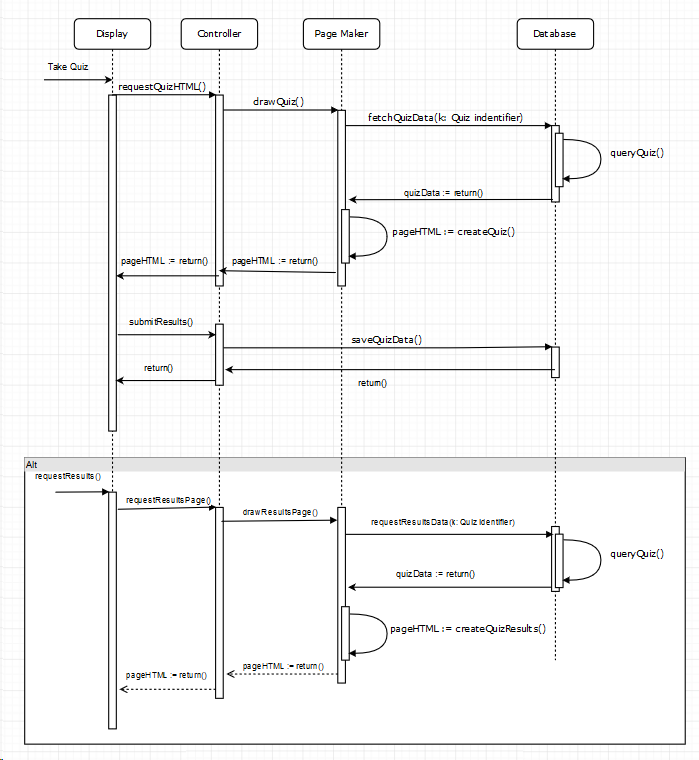
**Table Of Contents**

|  |  |
| --- | --- |
| Section | Page # |
| 1. Interaction Diagrams | 3 |
| 2. Class Diagram and Interface Specification | 11 |
| 3. System Architecture and System Design | 15 |
| 4. Algorithms and Data Structures | 19 |
| 5. User Interface Design and Implementation | 20 |
| 6. Design of Tests |  |
| 7. Project Management and Plan of Work |  |
| 8. References |  |

# 1. Interaction Diagrams

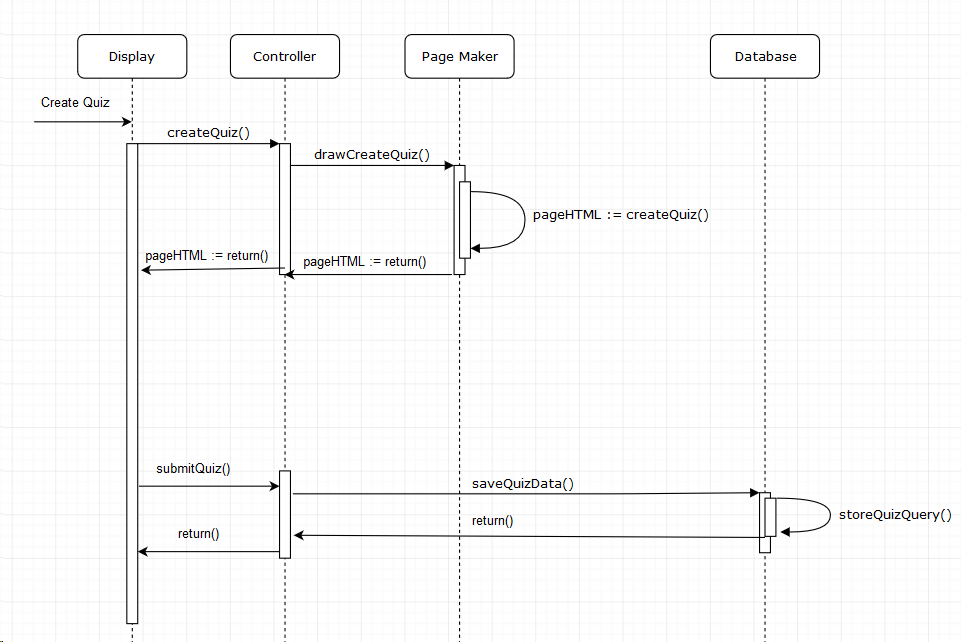
## Sequence diagram for UC 1: Take quiz and analyze results

The main scenario shown is for taking the quiz. The alternate scenario shown is for when a student requests quiz results. Both use the same domain concepts in similar ways, so they are shown in the same sequence diagram.



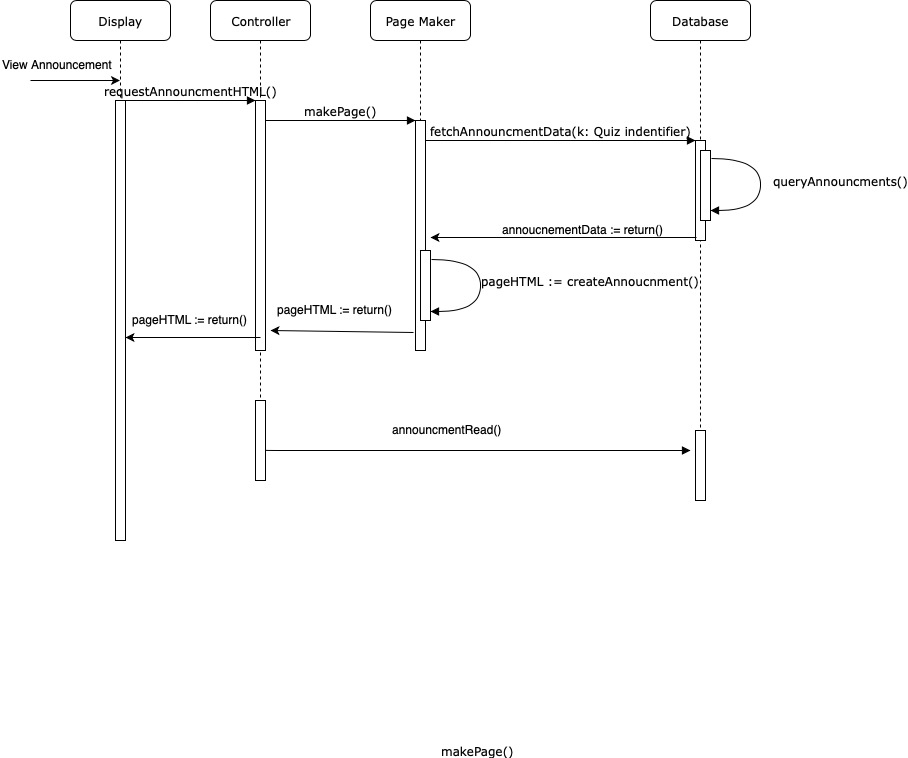
**Sequence diagram for UC 2: Create and Administer Quiz**

The sequence diagram below shows the sequence for creating a quiz. This includes the page maker which draws the appropriate page for quiz creation based on the particular initial request.



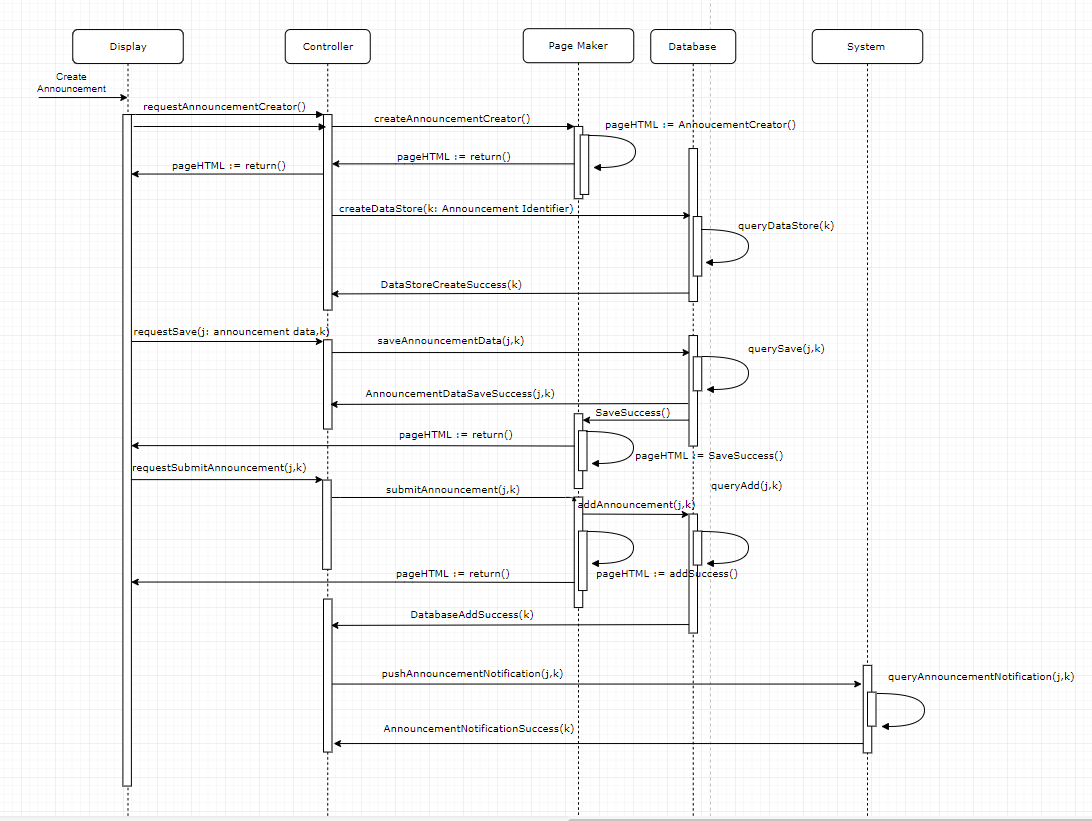
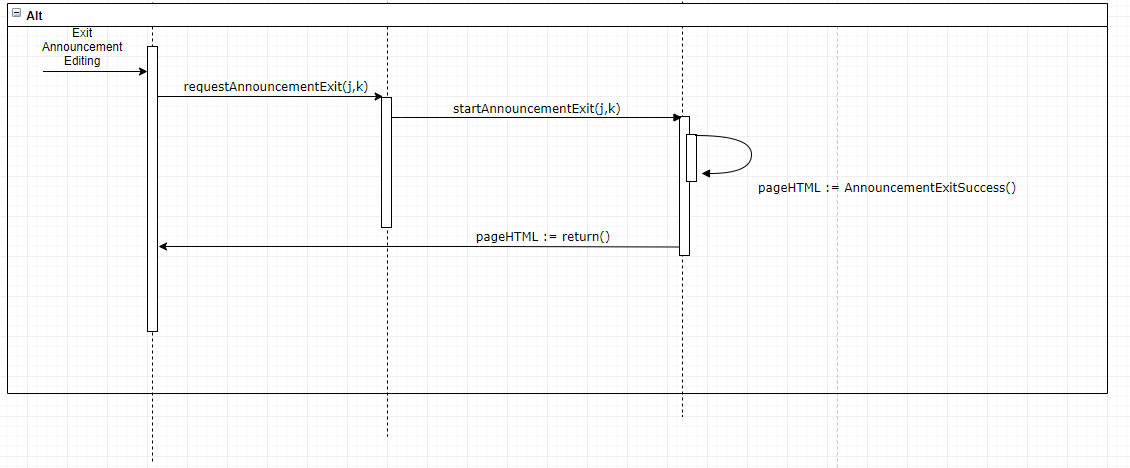
**Sequence diagram for UC 3: View Announcement**

The sequence diagram below shows the sequence for viewing an Announcement.



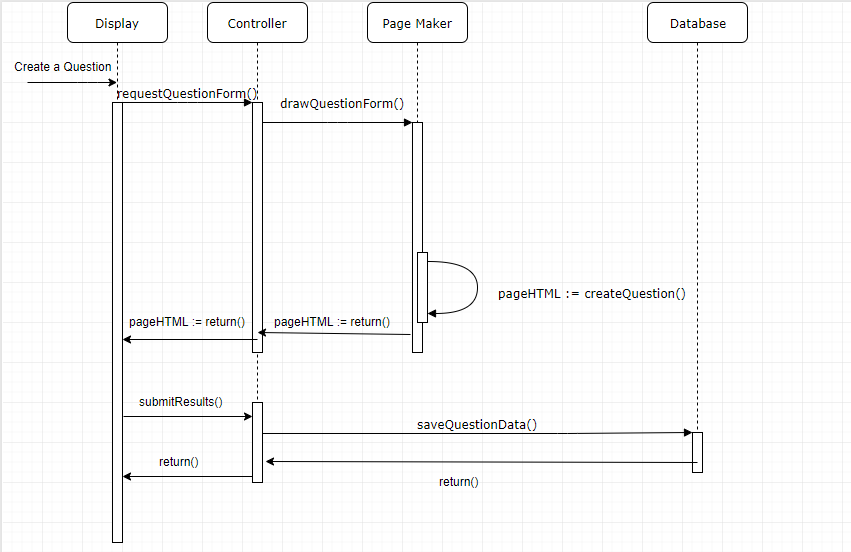
**Sequence diagram for UC 4: Create Announcement**

The following sequence diagram shows the the flow of object responsibility distribution when an authorized user (professors and teaching assistants) decides to create an announcement. Instances such as saving the announcement for another time, exiting the announcement creation process, and creating notifications through the system for announcements made have been included.

****

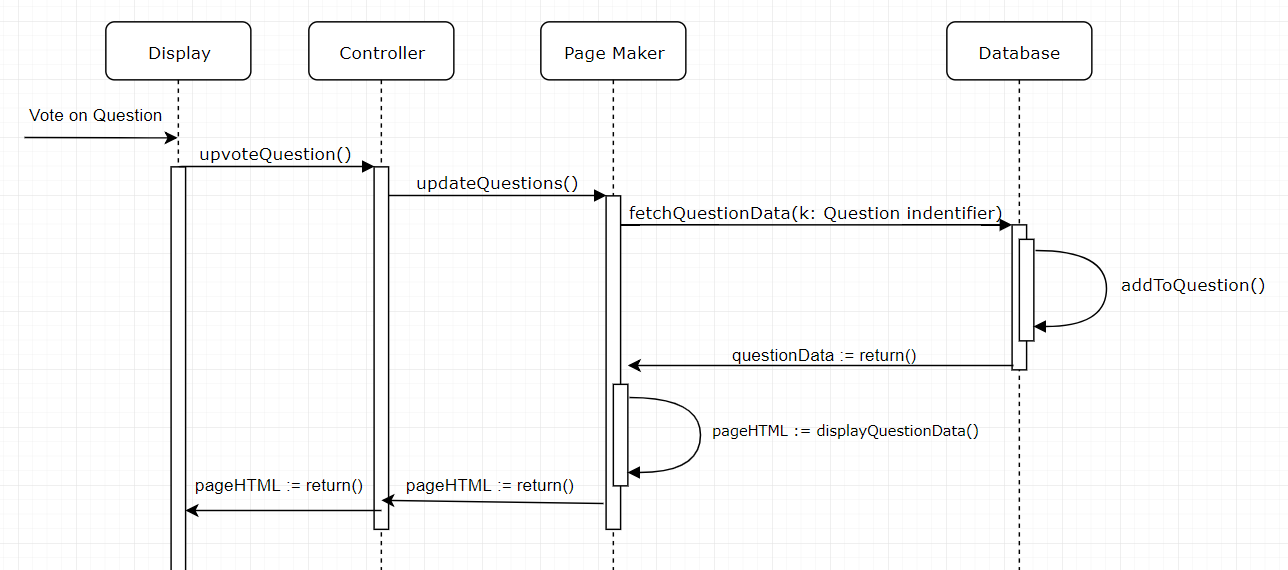
**Sequence diagram for UC 5: Create Question/Feedback**

The sequence diagram below shows the sequence for a student creating a question for the professor. This includes the page maker which draws the form for question creation once requested.



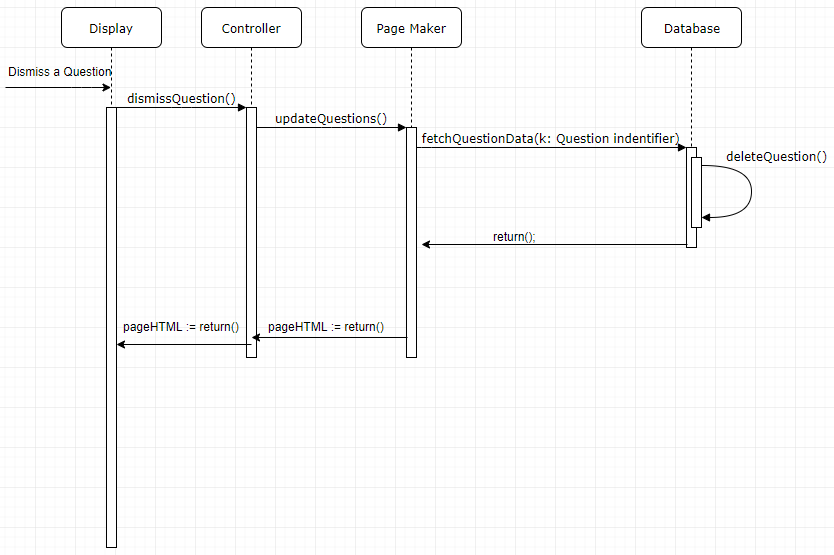
**Sequence Diagram for UC-6: Vote on Question:**

The diagram below shows the sequence for voting on a question. The user will vote on a question they deem relevant or helpful, and a request is made to update the score for the question in the database, and display the updated score on the page.



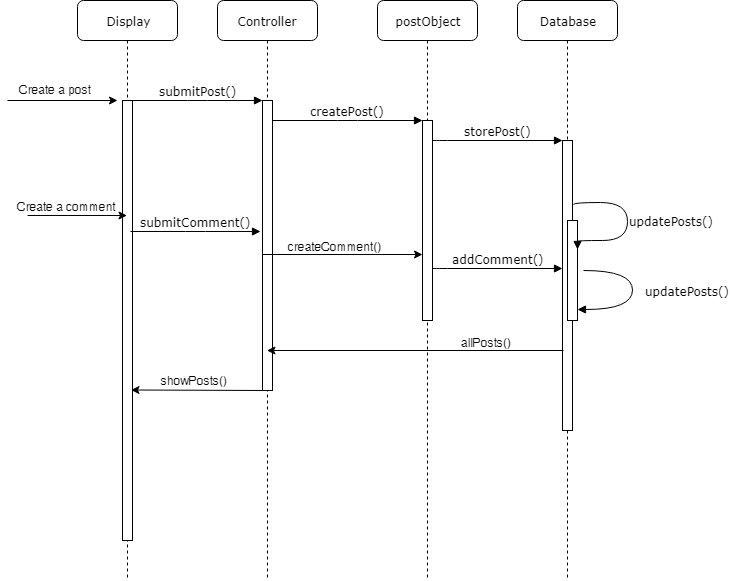
**Sequence diagram for UC 7 : Dismiss Question**

The sequence diagram below shows the sequence for dismissing an asked question. When a professor dismisses a question, he does not feel that it is necessary to address it, so the question is deleted.

****

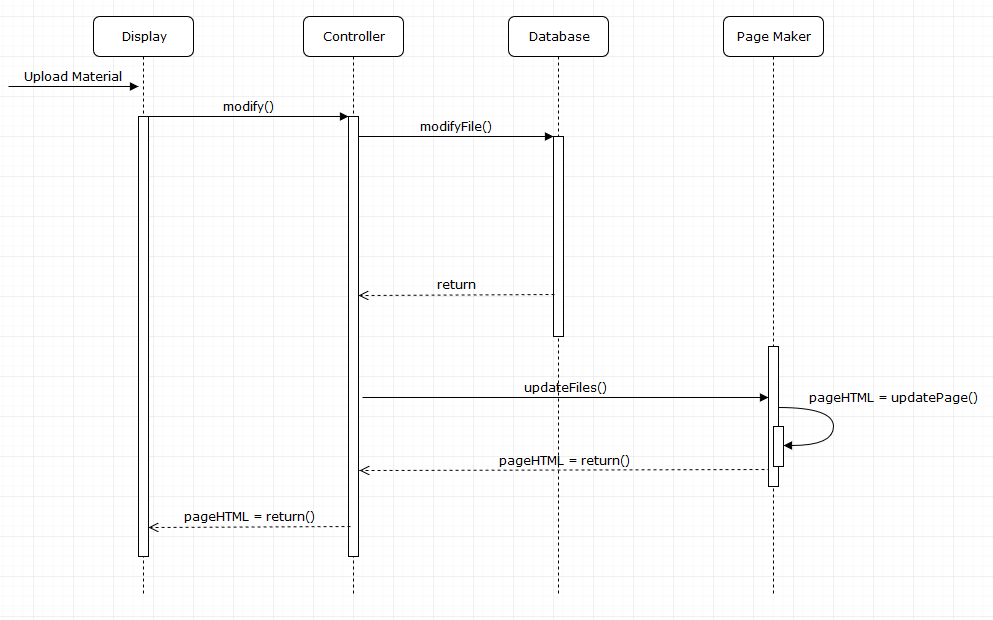
**Sequence Diagram for UC-8: Read Forum:**

The diagram below displays the sequence in which the users will interact with and read the forum. The user can submit a new forum post which will then be viewable by other users. The other users can then respond to the forum post.

****

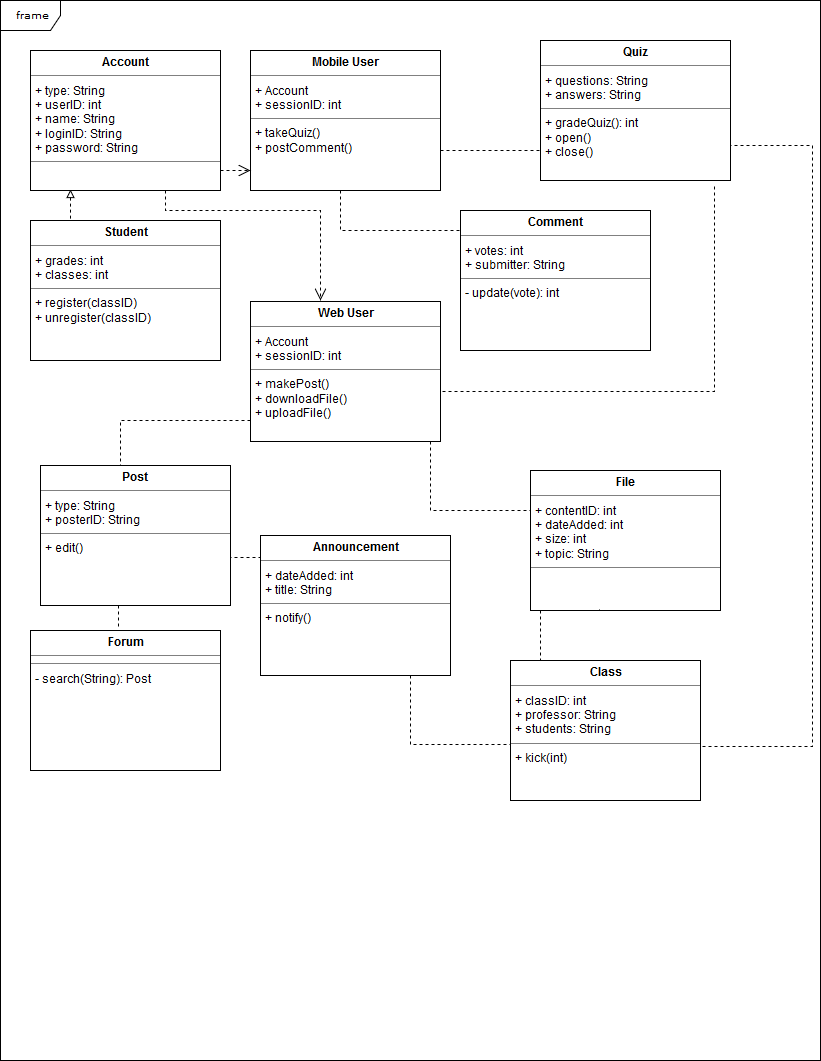
**Sequence Diagram for UC-14: Upload and Manage Resources:**

The diagram below shows the sequence for modifying course materials uploaded online. The professor has to ability to upload, delete, and move files that are available to all other users. The most up to date version of the files will be displayed and updated as modifications are made.

****

# **2. Class Diagram and Interface Specification**

# Class Diagram



# Data Types and Operation Signatures

1. Account
   1. Attributes
      1. Type: whether this account is professor, TA, or student
      2. userID: number identification for the account
      3. Name: name registered to account
      4. loginID: login user name
      5. Password: login password
2. Student
   1. Attributes
      1. Grades: grades of students classes
      2. Classes: classes student is registered in
   2. Methods
      1. register(classID): registered a student for the class with classID
      2. unregister(classID): unregisters a student for the class
3. Mobile User
   1. Attributes
      1. Account: The account that is currently logged into the mobile device
      2. sessionID: unique session ID for identification
   2. Methods
      1. takeQuiz(): starts a quiz to be taken on the app
      2. postComment(): creates a comment for other students and professor to view
4. Web user
   1. Attributes
      1. Account: account currently logged into web page
      2. sessionID: unique session ID for identifications
   2. Methods
      1. makePost(): makes a post to upload to forum or announcement if professor
      2. downloadFile(): downloads a file from the database
      3. uploadFile(): uploads a file to the database if professor
5. Post
   1. Attributes
      1. Type: what kind of account created the post
      2. posterID: id of the user that created the post
   2. Methods
      1. edit(): edits the post
6. Announcement
   1. Attributes
      1. dateAdded: the date the announcement was added
      2. Title: title of the announcement
   2. Methods
      1. notifty(): sends out a notification to students about the announcement
7. Forum
   1. Methods
      1. search(): searches for a specific post
8. Class
   1. Attributes
      1. classID: unique class ID
      2. Professor: the professor of the clas
      3. Students: list of students currently registered to take the class
   2. Methods
      1. kick(userID): kicks the user with the userID from the class
9. Quiz
   1. Attributes
      1. Questions: the questions of the quiz
      2. Answers: the answers of the quiz
   2. Methods
      1. gradeQuiz(): returns the grade of the quiz with the given answers
      2. open(): opens the quiz to be available to complete
      3. close(): closes the quiz and prevents answers to be submitted
10. File
    1. Attributes
       1. contentID: unique content ID
       2. dateAdded: the date the file was added
       3. Size: the size of the file
       4. Topic: a topic the professor says the file pertains to
11. Comment
    1. Attributes
       1. Votes: the current voting score the students have given the comment
       2. Submitter: who submitted the comment
    2. Methods
       1. update(vote): updates the vote count of the comment

# 

# Traceability Matrix

# 

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Use Cases** | **Account** | **Mobile User** | **Quiz** | **Student** | **Comment** | **Web User** | **Post** | **File** | **Announcement** | **Forum** | **Class** |
| UC-1 |  | x | x | x |  |  |  |  |  |  | x |
| UC-2 | x |  | x |  |  |  |  |  |  |  | x |
| UC-3 |  |  |  | x |  |  |  |  | x |  |  |
| UC-4 |  |  |  |  |  |  |  |  | x |  |  |
| UC-5 |  | x |  |  | x |  |  |  |  |  | x |
| UC-6 |  | x |  | x |  |  |  |  |  |  | x |
| UC-7 | x |  |  |  |  |  |  |  |  |  | x |
| UC-8 |  |  |  |  |  | x | x |  |  | x |  |
| UC-9 | x |  |  |  |  |  |  |  |  |  | x |
| UC-10 | x |  |  |  |  |  |  |  |  |  |  |
| UC-11 | x |  |  |  |  |  |  |  |  |  |  |
| UC-12 |  |  |  |  |  | x | x |  |  | x |  |
| UC-13 |  |  |  | x |  | x | x |  |  | x |  |
| UC-14 | x |  |  |  |  | x |  | x |  |  |  |
| UC-15 |  |  |  | x |  | x |  | x |  |  | x |
| UC-16 | x |  |  | x |  |  |  |  |  |  |  |
| UC-17 | x |  |  |  |  | x |  |  |  |  |  |
| UC-18 | x |  |  | x |  | x |  |  |  |  |  |
| UC-19 | x | x |  |  |  |  |  |  |  |  | x |
| UC-20 | x |  |  | x |  |  |  |  |  |  | x |
| UC-21 | x |  |  | x |  | x |  |  |  |  |  |

# 

# 3. System Architecture and System Design

# Architectural Styles

For this project we are using a *layered* architectural design. Our layers consist of:

1. UI Layer / Presentation Layer

This layer will be solely be used for UI and UX concepts. It will be easy to change and edit without impacting the application as a whole.

1. Application Layer / Service Layer

Beginning here, all of our services will exist on this layer and above. Concepts and features like polling and announcements are such applications.

1. Domain Layer / Logic Layer

This layer will control the logic of the application and how they work together.

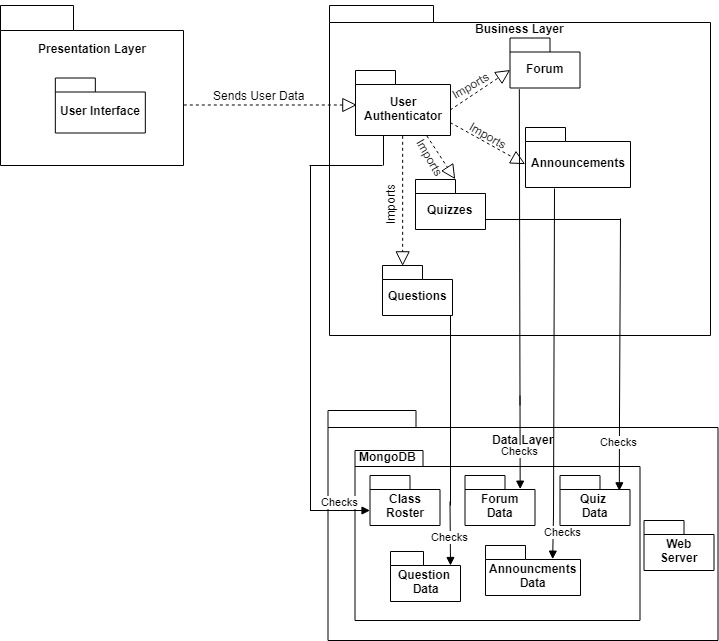
1. Persistence Layer / Data Access Layer

Our database will exist on this layer.

The benefits of using a layered style for our app are numerous. Firstly, lack of dependence on other layers will allow for more code reuse, which is key in a team of this size and on an app on multiple platforms. Second, the encapsulation and abstractions of each layer makes testing easier. Additionally, as the app gets more and more featured, it will be easier to maintain. Lastly, performance is not a big requirement for this application, which is the main drawback of a layered system.

# Identifying Subsystems

UML Package Diagram



# Mapping Subsystems to Hardware

1. Mobile App - Android and IOS
2. Professor Web app - Web Browser
3. Student Web app - Web Browser
4. Database - AWS EC2 + S3

# Persistent Data Storage

We decided to use a MongoDB-powered database to manage data for the different components of our application. The main actors in our application include students and professors, both of which will be stored as users in our database. Each user will have a unique user ID and password that will be used to authenticate the users to access their relevant courses, files, and grades through their Minerva portal. Our database will allow us to retain permissions for different users, such as allowing teachers to edit grades and upload materials, and for students to view only their grades for assignments and assessments. Some components of our project will rely heavily on database for storage, such as our forums, and gradebook features. Forums will be stored as threads of posts, which will store relevant information such as the user that created them, the time at which they were created, and of course the actual content of the post. Gradebooks will exist for each unique course, and a numerical grade for each assignment will be given to each unique student in the class. A major reason we chose MongoDB to implement our database was because of its flexibility and ability to easily store many different file types of varying sizes. This will be incredibly useful for the “resource folder” portion of our application, where teachers will be able to upload relevant course materials such as syllabi, assignments, lecture slides, etc. which could be many different file formats (.pdf, .docx, .ppt). Other SQL database management systems would likely make this much more difficult as they are very strict about data types and sizes of elements stored, unlike MongoDB.

# Network Protocol

Our project has four different components to keep track of: the webserver, the frontend website, the mobile app, and the server for the app information. Thus, we need multiple different channels of communication. We will consolidate the webserver and app server into one server written in Node.js hosted on Amazon Web Services. The server will act as a RESTful endpoint for communication with the frontend for the website and mobile app through the HTTP protocol. This will allow us to have a consistent protocol for data transfer across both of our interfaces. The use of a Node.js server will give us access to many publicly available packages such as code to make sure data is correctly encrypted or making sure that the data transfer between the database is reliable. The server will maintain a connection to a MongoDB database as described in the above “Persistent Data Storage” section of this report.

# Global Control Flow

For the webapp, the system as a whole is event driven for deciding which functionality the user wants to use. However, specific functionalities are procedure driven. For example, when taking a quiz in the webapp, the user must select their options for the quiz questions and submit the form in order to continue. Similarly, for chat and forum functions, the user must input certain messages in order to an outcome to appear, which is a message being sent in this case. Since there are different functions to choose from, the initial interface will be event driven. Once a specific functionality is chosen, the app will be procedure driven until the procedure is complete, at which point it will return to being event driven.

Certain functions of the system are time dependent, specifically the quiz features. For the quizzes administered through the webapp, a quiz remains open for an amount of time as determined by the instructor. Once that time has elapsed, the student is locked from the page and the results are automatically submitted. Furthermore, there is also a timing feature in how long the quiz should be available for. For example, the instructor might specify a 24 hour window in which the student can take the quiz. Once that window has elapsed, the quiz can no longer be taken by the students and the option will not be available. Both of these are real-time constrained systems for a time frame specified by the instructor.

There are no concurrency specifications for this system.

# Hardware Requirements

1. Screen Colored Display 1920 x 1080 (1080p resolution) 400 ppi minimum
2. Web server ( AWS free tier )
   1. Amazon S3:
      1. 5 GB of Standard Storage
      2. 20,000 Get Requests
      3. 2,000 Put Requests
   2. Amazon EC3
      1. 750 hours of usage a month
3. Touch Screen enabled Smart Phone (latest Android or iOS) for Mobile app
4. Network Bandwidth of 256kbs
5. 1 GB ram
6. 40 MB hard drive space

# 4. Algorithms and Data Structures

# Algorithms Our backend resource search will use fuzzy string matching algorithms to match the search results with pre-hashed data. Fuzzy string matching is computed using Levenshtein distances between two words, or the shortest number of insertions, deletions, or letter swaps it takes to get from one word to another.

The time limit of 150 seconds (2 mins 30 sec) after a question is asked by a student in class for it to be voted on is based on a study that states, “around 20% of a students time in class is spent on their phone" 1. In an 80 minute lecture this would amount to 16 minutes. Assuming that the 16 minutes of phone time would be a random linear distribution throughout class, each student would check their phone an average of once per 5 minutes. Assuming a random distribution among all of the students of when they check their phones in 5 minute intervals, it can safely be assumed that in the 2 min 30 sec time interval, approx. ½ of the class may check their phone and see a new question that was posted. Since 50% of the class would potentially view a question, if ⅕ of them or a total 10% of the class thought a question had merit or was wondering the same thing and upvoted the question, it would pop up as a notification for the Professor.

This timer is set by default, only the 10% value will be calculated separately for each class based on number of students in lecture.

1. **Data Structures**

Any and all resources uploaded to the backend will be pre-filtered and inserted into a word frequency hashmap (key: “word”, value: “resource”). We will generate a predetermined number of hashmaps, representing the frequency of words. For example, if a document called “BipolarJunctionTriodes.pdf” has the following frequency of words in descending order [it, the, a, BJT, Gaussian Curve, to, diode], the list would first be filtered of non meaningful words such as it, the, and to resulting in [BJT, Gaussian Curve, diode]. Then in the most-frequent-word-hashmap, (key: BJT, value: “BipolarJunctionTidode.pdf”) would be entered, in the second-most-frequent-word-hashmap (key: Gaussian Curve, value: BipolarJunctionTriode.pdf) would be entered. This greatly decreases lookup time for the end user.

The in-class feedback/question will be a utilizing an AVL tree. The priority of each feedback/question will be 0 and will be updated whenever a user votes on the question. Feedback needs to be ordered so that the most voted one will be at the top. Updating the priority of a feedback/question needs to be fast for the page to update in real time for the users will take O(log n). Inserting, Deleting feedback/questions will take also O(log n) time.

For the purpose of online quizzes, arrays will be implemented to store the questions in our database. Quizzes will be formatted as an array of questions, which will contain an array of possible answers for each question. The first possible answer stored in the array will always be the correct answer, but the answers will be listed in random order when displayed to the students. This will combat sharing answers among students, simplify the quiz creation process a bit, and reduce space needed in the database to label the correct answer for each question. Quizzes will always be static, so arrays are ideal for this purpose. They allow for O(1) access and can easily be created and removed from the database.

# 5. User Interface Design and Implementation

1. **Mobile:**We took the designs from report 1 and implemented them using React Native so that it deploys in both iOS and Android. We used javascript to code some React Native components which define the styling and UI of the front end. For example, the input for net-id and passwords are handled with textinput component, where the values are stored as props. To navigate from screen to screen, we used the react native navigator, a built in library that helps navigate between screens (segues in iOS).

The user effort should be reduced in most cases as the react native navigator reduces the number of taps required to navigate between pages as there is a persistent navbar at the bottom of the screen.



1. **Web App:**There are no significant changes to the UI mocked up in the initial report.

# 6. Design of Tests

For sign-in, the tests will be conducted by determining which portions of the sign-in are working. Since we are using React and React Native, it is easy to see which portions of the components are storing the login string and password and check if they are properly being authenticated.

In order to log in we will be testing many scenarios including:

* User creates a new account
* User tries to create an account with an already registered email
* User registers for a class (both when they aren’t registered to any and if they already are registered to one)
* User tries signing in with incorrect credentials
* User tries signing in with insufficient credentials (missing required information)
* User signs in successfully (both professor case and student case)
* Unknown error
* Too many sign-in attempts

For the forum, there are many individual components that must be tested to ensure the application is working correctly.

In the below test cases, the user is assumed to be logged in as an existing user in the database

* When the user submits a post on a brand new thread, the database is populated with a new thread containing that post
* When the user attempts to view a specific thread, no other threads or posts from other threads will be shown to the user
* When the user submits a post on a specific thread, that post is associated with that thread in the database only
* When the user submits a post on a specific thread, and is not allowed to post on that thread, the post will not be associated anything in the database, and will be deleted
* When the user attempts to view a post on a specific thread, and is not allowed to view that thread, they will not be able to see that thread
* When the user is creating a post, the post will not be submitted to the database until the user explicitly submits the post

Teaching assistants and teachers should have all existing privileges of users as well as additional permissions. These test cases outline the additional actions and information TA’s and teachers should have access to:

* Users with administrative powers (teachers and teaching assistants) try to delete a post, thread, or comment
* Authorized users try to undo the deletion of a post or comment
* Authorized users try to endorse an answer as the correct answer
* Authorized users try to access the edit history of a post or comment
* Authorized users try to “pin” a post (have said post appear first at the top of its category) for a set amount of time
* Authorized users try to “unpin” a post, regardless of which original authorized user pinned it
* Authorized users try to set tags to posts
* Authorized users try to edit their pinned announcements, and have a subsequent announcement pushed for it

For the quizzing features, there are various test cases that need to be passed to ensure that the application is working correctly. However, not all of these test cases are programmable in unit test cases. The unit test cases are similar to the actual usability test cases, so they will be defined jointly.

In the below test cases, the user is assumed to be an administrator who has the permissions to make a quiz. The test cases for making a quiz are defined as:

* User is an administrator
* When user submits a quiz with one question, the database is populated with a one question quiz
* When a user submits a question with one answer, one answer is associated with that question in the question table in the database.
* When the user submits a quiz with multiple questions, all questions are associated with the quiz in the quiz table and are not associated with other quizzes.
* When the user submits multiple answers for one question, the answers are only related to that particular question in the question table.
* When the user is trying to make quiz, the data is not submitted to the database until the user explicitly submits it or until the auto-save clock saves the quiz.
* When the user makes a second quiz, all fields are empty on the quiz creation page.
* When the user submits the information for a second quiz, all questions are confined to their particular quiz and all answers are confined to their particular question.
* When the user tries to edit a quiz, only the questions and answers for that particular quiz are displayed.

The following are test cases for *accessing* a quiz:

* User is a student
* When user opens a quiz, a call to the database will return the quiz data necessary to be rendered.
* When the user is taking a quiz, a timer counts down to indicate how much time a user has remaining.
* When a user submits a quiz without answer all the questions, a confirmation popup is displayed to notify the user that some answers were left blank.
* When a user submits a quiz with all the answers selected, the answers are automatically graded, and the results are submitted to the database (final score, answer selections, etc.).
* When a user fails to submit within the time-limit, the current state of the quiz will graded and submitted to the database.
* If a user begins a quiz within a timeframe of less than the given time-limit, the user will not have the full time limit to complete the quiz. All current answers at the time of the close date will be submitted promptly.
* If a user does not attempt the quiz within the given timeframe, the user will receive an automatic ‘incomplete’ or a grade of 0.
* A user cannot see a quiz until it is released.
* A user cannot alter their answer choice selections after it has been submitted.
* A user will receive feedback after the quiz deadline has passed.

For the announcement page, there will be a good mix of programmable unit tests and manual tests. Tests cases are as defined:

* User is anyone
* When user clicks on announcement page, a call to the database will be made for the current announcements completed and another call to mark the ones read for the user.
* Creators are Professors and TAs
* When the creator types in the textbox an option to submit and save the announcement will be available.
* When the announcement gets saved it gets populated in the announcement table in the database as in-progress.
* When the announcement gets submitted it gets populated in the announcement table in the database as completed.

For the resources folder and page, there are some considerations that need to be made to maintain consistent access for all users. Test cases are as follows:

* Users is anyone
* Only professors and TAs have permission to upload new materials
* Creators include professors and TAs
* When a creator attempts to upload a new document or file, the file will be stored in the database and be visible in the resources folder on the web application
* When a creator attempts to upload a file that already exists with the same name, the file will not be uploaded.
* When a creator attempts to upload a file greater than 25MB, the upload will fail (This is only for our current version, which has limited database storage space)
* When a student attempts to upload a file, the application will not allow them.
* The resource folder will only show files for the class specified.

# 7. Project Management and Plan of Work

1. **Merging Contributions**

A consistent format was kept throughout all the reports, and previous reports were saved.

Everyone worked in the same Google Documents page and each group member is responsible with keeping to the same format. So combining every individual piece of work work was simply just copying the previous reports into the new document. The only issue was coordinating backend and database use between the team. Since the team is large, detailed plans must be made before coding starts to ensure consistency.

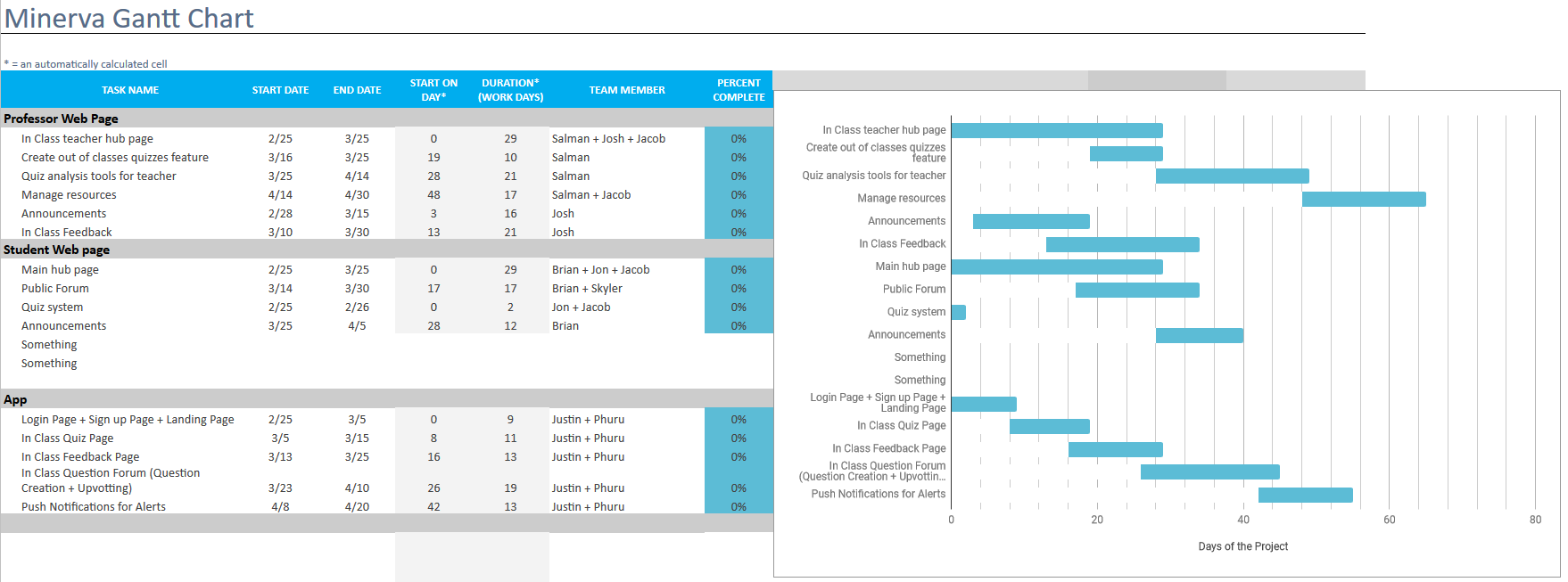
1. **Progress Report**

Currently there is a very basic working shell for each team: the mobile app, student web page, and teacher web page. A general framework and navigation around each project is generally completed, however overall functionality is still severely lacking and is the current focus of every team. Until functionality progresses to a presentable degree, linking the three programs together is not an active goal. The database system is also currently being worked on and some teams are currently working on integrating their project with the database.

After some basic functionality has been achieved, the three teams will be interacting closely with team member Jacob Battipaglia as he is responsible with syncing the web pages with the mobile app. For large decisions, meetings will be arranged on an as-needed basis. For general communication between teams, Slack is the preferred method. Individual teams use whichever method of communication is most comfortable among the team members.

1. **Plan of Work**

As of now, the project is still on track to make the deadlines set in the gantt chart submitted in report 1 (also shown below).



1. **Breakdown of Responsibilities**

|  |  |
| --- | --- |
| **Module** | **Team Member** |
| Student Web Page Quiz | Jon Hong |
| Student Forums | Brian Ma |
| Teacher Forums | Skyler Lee |
| Student Web Page Announcements | Jon Hong |
| Student Web Page Grades | Brian Ma |
| Student Web Hub Page | Brian Ma and Jon Hong |
| Teacher Web Page Announcements | Joshua Olazo |
| Teacher Web Page Quiz | Salman Omer |
| Teacher hub page | Joshua Olazo and Salman Omer |
| Teacher Feedback page | Joshua Olazo |
| Mobile Login Page | Justin May |
| Mobile Push Notifications | Justin May |
| Mobile Announcement Page | Phurushotham Shekar |
| Mobile In-Class Main Page | Justin May |
| Mobile Forum Page | Phurushotham Shekar |
| Mobile Quiz Page | Phurushotham Shekar |
| Database Management | Jacob Battipaglia |
| Database Integration | Jacob Battipaglia |
| Mobile/Web App Interaction | Jacob Battipaglia |

The integration between the web page portion and mobile app portion will be coordinated by Jacob Battipaglia, however the integration of the two parts of the web page will be handled by the leaders of the two teams. Due of the nature of the project, many modules and classes are very closely related, and thus each team may need to work closely with another team on many parts of the overall product. Because of this, we decided to leave the integration to those involved because they would already be working closely together on some degree of integration. Also as a result of this, the integration testing would most likely be handled by the web page teams together for the web page, and Jacob Battipaglia for the project as a whole.

For the mobile portion, the members Phurushotham Shekar and Justin May divided up the work by pages. All of the components in a page will be handled by the person in charge of the page. Both Justin and Phurushotham will work on integrating their pages with the back-end and database and work with Jacob Battipaglia as needed.

# 8. References

1. “Barney McCoy Publishes New Digital Distractions Study.” *Barney McCoy Publishes New Digital Distractions Study | CoJMC | Nebraska*, Jan. 2016, journalism.unl.edu/news/barney-mccoy-publishes-new-digital-distractions-study.