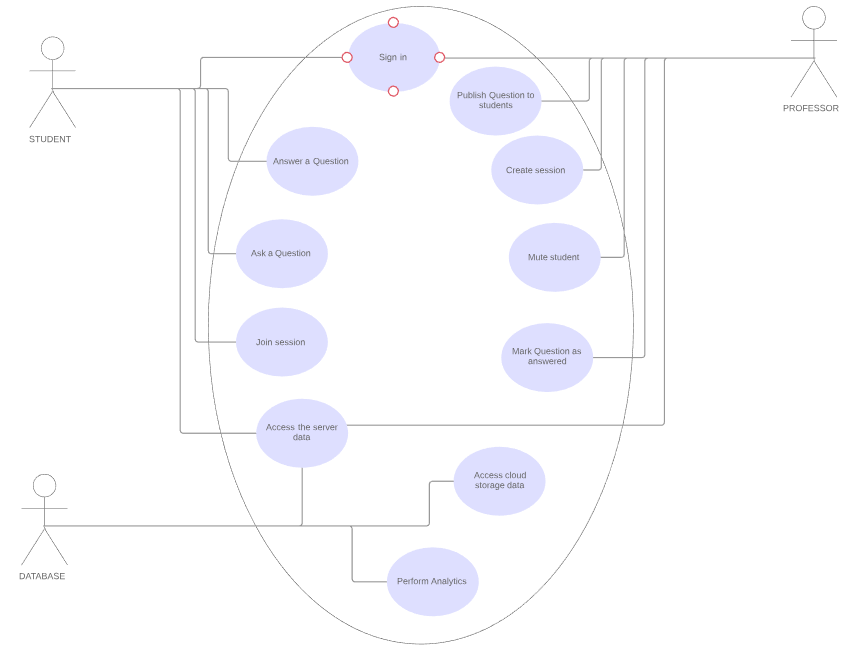
**mASK Specifications Document**

**3/12/2019**

**CSC 340**

**ROOT SYSTEM:**

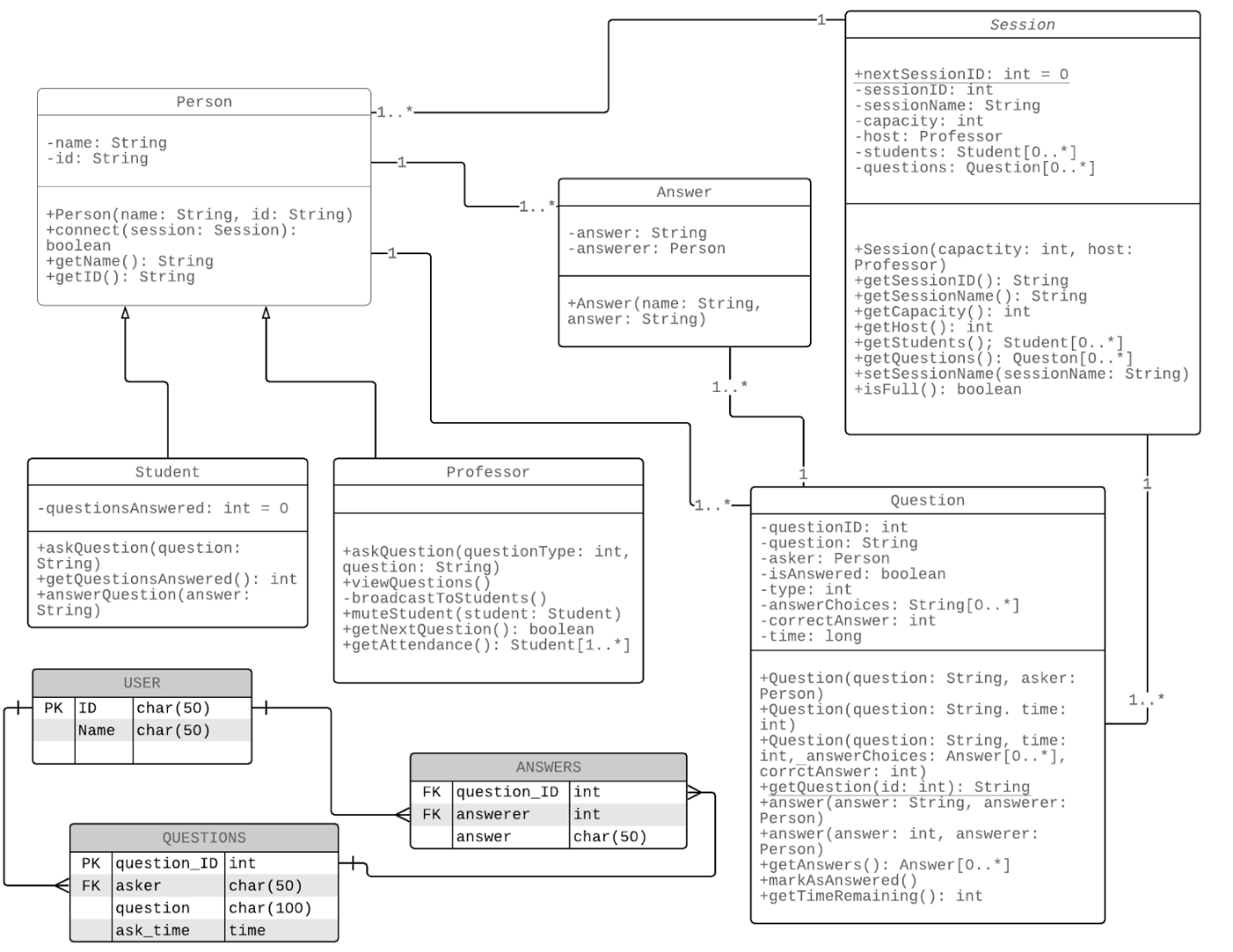
**Use-Case Diagram**



**USE-CASES**

1. Case: Sign-in
   1. Actors: Student, Professor
   2. Preconditions: App must be loaded on user’s device and on the home screen
   3. Related use case: None
   4. Steps:
      1. Provide username and password
      2. Press enter for verification
2. Case: Answer a Question
   1. Actors: Student
   2. Preconditions: Must be on a question’s page where the “answer question” button is
   3. Related use case: Ask a question
   4. Steps:
      1. Type out answer for a question in text box provided
      2. Press submit
3. Case: Ask a Question
   1. Actors: Student
   2. Preconditions: Be on the event screen that prompts user to ask a question
   3. Related use case: Answer a Question
   4. Steps:
      1. Type out question in text box provided
      2. Press submit
4. Case: Join Session
   1. Actors: Student
   2. Preconditions: Must be in a valid environment for server presence and on the “join session” screen
   3. Related use case: Create Session
   4. Steps:
      1. Get session information and provide it in the application
      2. Press submit and session should be joined
5. Case: Access Server Data
   1. Actors: System admin
   2. Preconditions: Server must have received data from front-end application
   3. Related use case: Access Cloud Data
   4. Steps:
      1. Login to system
      2. Query for a specific data set
6. Case: Access Cloud Storage Data
   1. Actors: System admin
   2. Preconditions: Cloud storage system must have received data from the server and analytics
   3. Related use case: Access Server Data
   4. Steps:
      1. Login to cloud system
      2. Query for a specific data set
7. Case: Perform Analytics
   1. Actors: System admin
   2. Preconditions: Data must have been received from the cloud storage unit
   3. Related use case: none
   4. Steps:
      1. Model Data
      2. Further analyze using classification algorithms
8. Case: Publish Question to Students
   1. Actors: Professor
   2. Preconditions: Be on publish question page that prompts user to do so
   3. Related use case: Ask a Question
   4. Steps:
      1. Type out text of question to be published
      2. Press submit
9. Case: Create Session
   1. Actors: Professor
   2. Preconditions: Be on home page where “create session” button is prompted
   3. Related use case: Join Session
   4. Steps:
      1. Provide application with proper session information
      2. Press submit
10. Case: Mute Student
    1. Actors: Professor
    2. Preconditions: Be on the question/answers page where student’s responses are displayed
    3. Related use case: Mark Question as Answered
    4. Steps:
       1. Toggle button next to student’s username to mute student
11. Case: Mark Question as Answered
    1. Actors: Professor
    2. Preconditions: Be on question/answers page
    3. Related use case: Mute Student
    4. Steps:
       1. Toggle button next to question header to mark question as answered

**Class Diagram**



In the class diagram above, the Person class is a superclass for both the Student and Professor class. It has two data fields that are defined in the constructor that includes both the user id and name of the person and will input their info into the database in the USER table. The class has a connect method that takes in a parameter for the session ID and will connect the person to the Session class.

The Student subclass contains a single extra data field, the questions answered. The number is incremented each time a student answers a question so that analytics regarding participation can be determined. The askQuestion method will take the question String as a parameter and submit the new question to the database in the QUESTIONS table. The getQuestionsAnswered method will determine the number of questions the student has answered. The answerQuestion method takes in the answer as a String parameter and will submit the answer to the database into the ANSWERS table.

The Professor subclass contains an askQuestion method that takes in as parameters the question type, which will either be short answer or multiple choice and will be determined by an int value of either 1 or 2, and the question as a String. The question will get added to the QUESTIONS database. The viewQuestions method will query the QUESTIONS database for a list of the current questions that have not been answered for the professor to view. The broadcastToStudents will send a message to all connected Student devices. It is marked private because it will be called by the askQuestion method and not directly by the Professor. The muteStudent method will take in the Student object and ban the student from asking and answering any more questions. It will most likely be used for students who decide to spam the server with useless or inappropriate data. The getNextQuestion method will receive a question from the QUESTIONS database that will be determined based on a highly complex machine learning algorithm that determines which question is the most important. The getAttendance method will return a list of all the current Students logged into the session.

The Session class has a static variable nextSessionID that will return the next int value. This will be used to ensure that all sessions have a unique session ID to connect to. The sessionID field is the session ID for the current instance of the Session class. The sessionName data field is the public name for the session, which will most likely be the name of the class that is being taught or the main topic for the days lecture. The capacity data field is used to set the max number of Students from logging in. This can be used to check to make sure each Student is only logged in once and that there are no excess users who log in remotely to disrupt the classroom data. The host data field will contain the instance of the class professor who will act as an administrator to the session. The constructor for the class will take in as parameters the capacity of the session and the instance of the Professor class that will act as the host. The getSessionID, getSessionName, getCapacity, and getHost methods will return the values of the respective data fields. The getStudents method will query the USER table in the database and return a list of the Students in the current session. The getQuestions method will query the QUESTION table in the database to return a list of all questions. The isFull method will return whether or not the session is currently at capacity with a true/false statement.

The Question class contains a data field named questionID which contains the unique ID of the question. It also contains a data field named question which contains the actual question being asked as a String value. The asker data field will contain a reference to the instance of the Person who asked the question. The type data field contains an int to determine if the question is 0 student asked, 1 short answer, or 2 multiple choice. The answer choices data field is a collection of Strings that will display on the student devices if the multiple choice type is selected for the student to choose from. The correct answer index will be contained in the correctAnswer int data field. The time data field will contain the time that the question will be closed. The class contains three separate constructors. The first one will be used by students and will default the value of the type data field to 0. This will be used by students to ask questions they do not understand. These questions will be added to the QUESTIONS database as well. The second and third constructor will be called on by the Professor. They are the questions that the professor will broadcast to the students. The second constructor is used to ask short answer questions and the third is used to ask multiple choice questions, with the answer choices and the correct answer index needing to be provided. The getQuestion() static method will return the instance of the question with the given questionID by querying the QUESTIONS table in the database. The answer method is overloaded with two possible values. The first method will be used by students to answer other student questions and short answer questions with the second being used to answer multiple choice questions supplies by the Professor. The getAnswers method will query the ANSWER table in the database and return the list of answers for the given question. The markAsAnswered method will mark the question as answered and will close the question off from future answered being made. This method can only be accessed by the Professor. The getTimeRemaining method will return the seconds remaining until the question is closed and will be used primarily by the Professor when a question is broadcasted to students.

The Answer class contains a String data field containing the answer to the question and an answerer data field that contains an instance of the Student who answered the question. The constructor will ask for the name and instance of the Person and store the values accordingly.

The USER SQL table will be used to store the Students in a database. The table has the ID of the Student as a primary key and a name field, both stored as a collection of chars (basically a String).

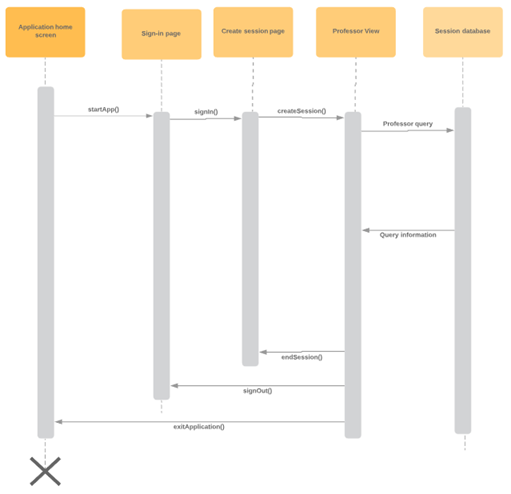
The QUESTIONS SQL table will contain as a primary key the question\_ID as an int. The table will contain the asker of the question as a foreign key that references the ID of the USER data table. The question field will be used to store the question being asked. The ask\_time field will store the time the question was asked.

The ANSWERS data table will contain the question\_ID field which is a foreign key to the question\_ID of QUESTIONS and is a link to the question it is answering. The answerer field is a foreign key to the USER table and references the ID field. It contains the person who answered the question. The answer field contains the actual answer to the question.

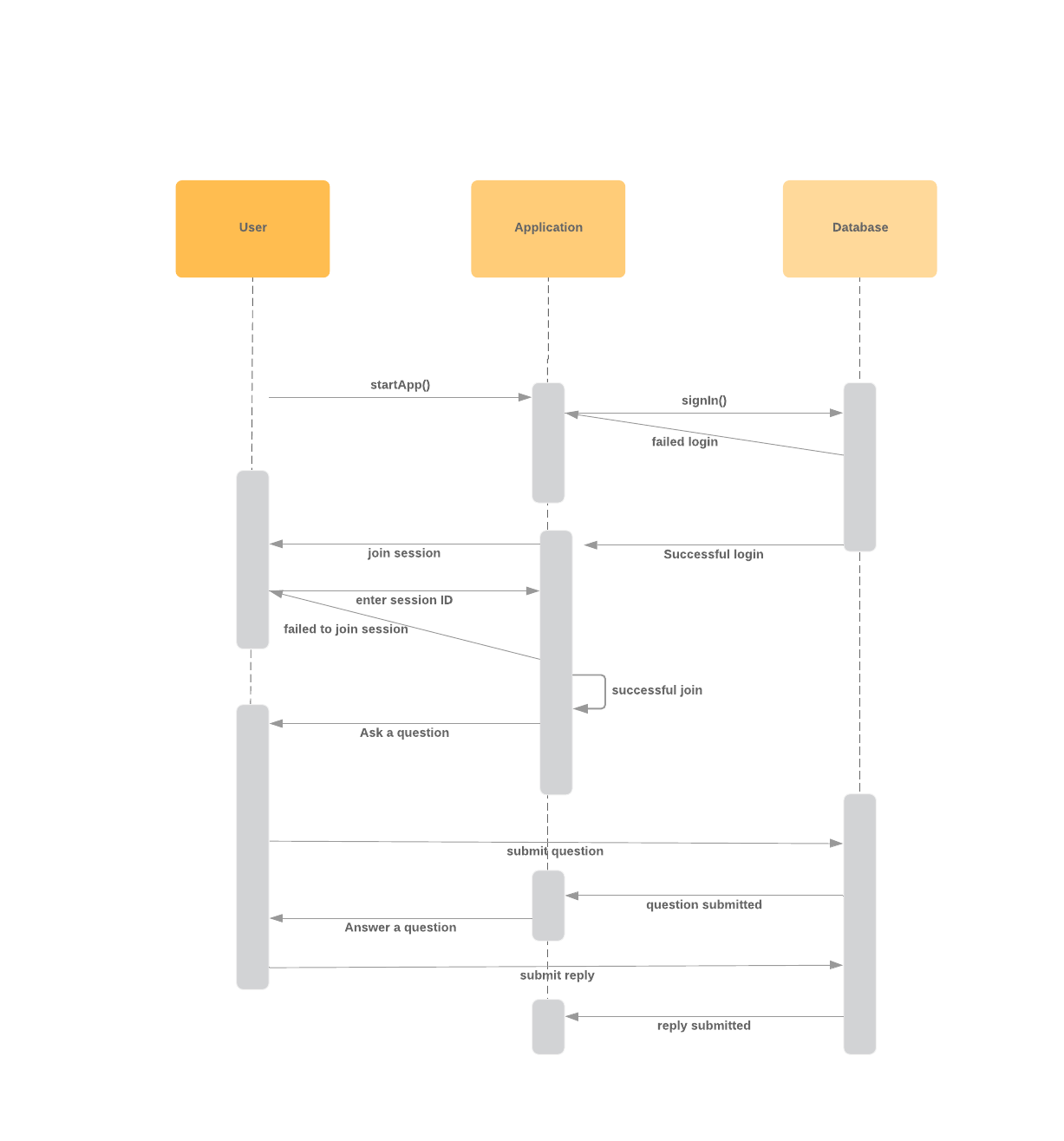
There is a One-to-Many relationship between the Person superclass and the Question and Answer classes because each Person can have many different Questions and Answers but each Question and Answer can only have one Person. There is a Many-to-One relationship between the Person and the Session classes because each session can have many Persons but each Person can only be in one Session. There is a Many-to-One relationship between Answer and Question because each Answer can only have one Question, but each Question can have many Answers. There is a Many-to-One relationship between Question and Session because each Session can have many Questions but each Question is only available in one Session.

**Sequence Diagram**

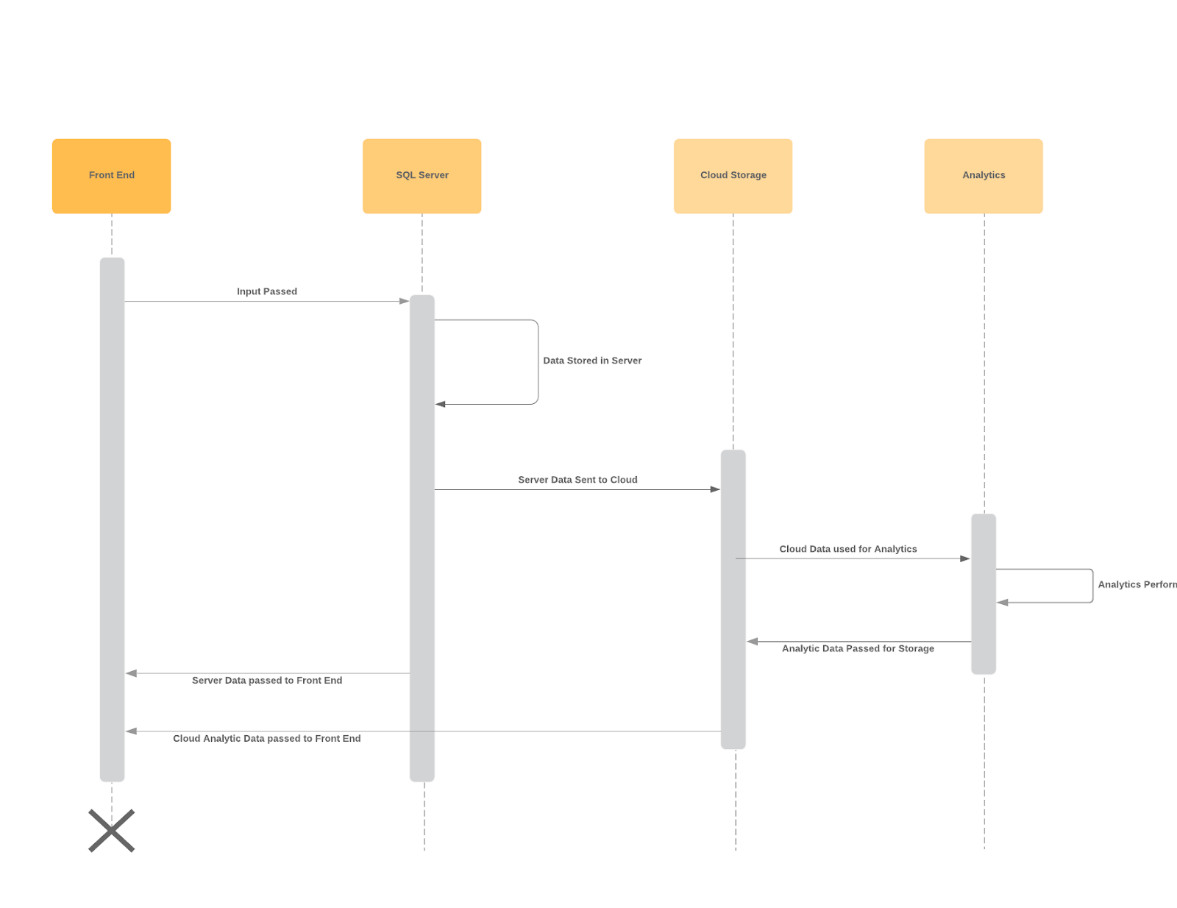
Professor Sequence (leading into backend):



Student Sequence (leading into the backend):

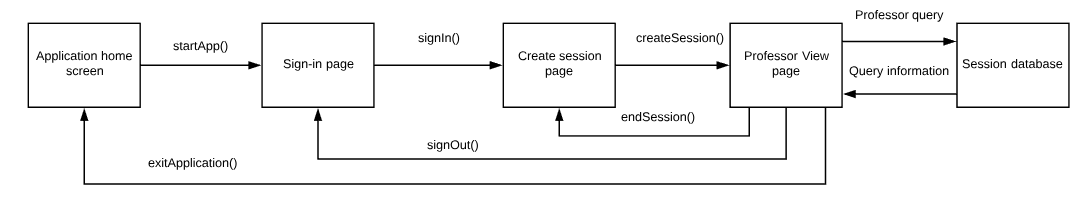
****

Backend Sequence (led in by Professor/Student users):

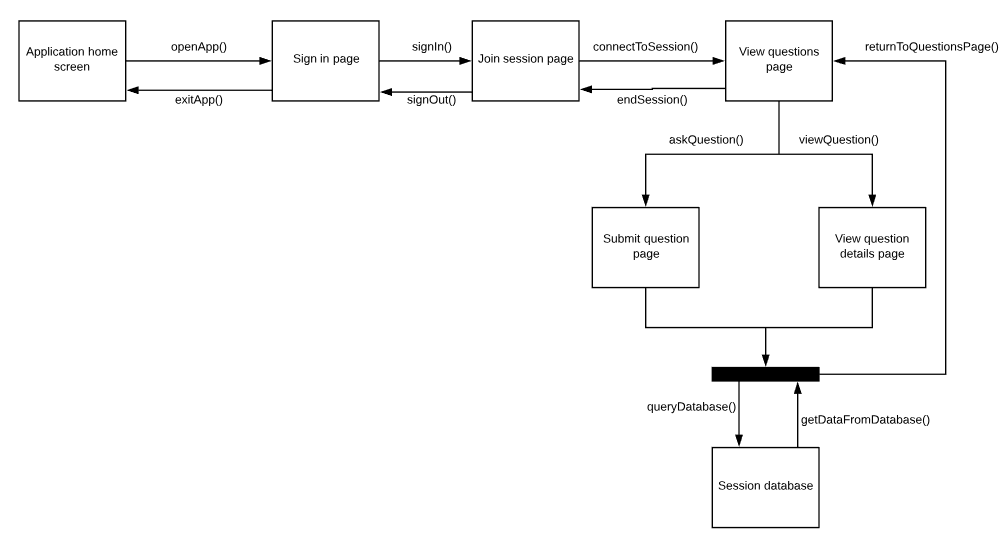


**State Diagram**

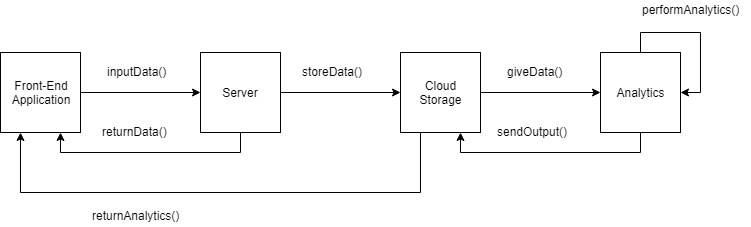
Professor View:



Student View:

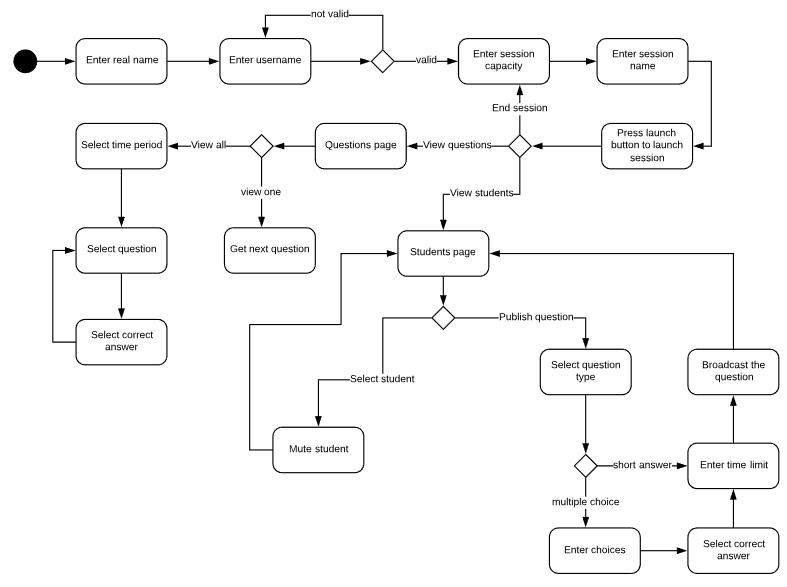


Backend:

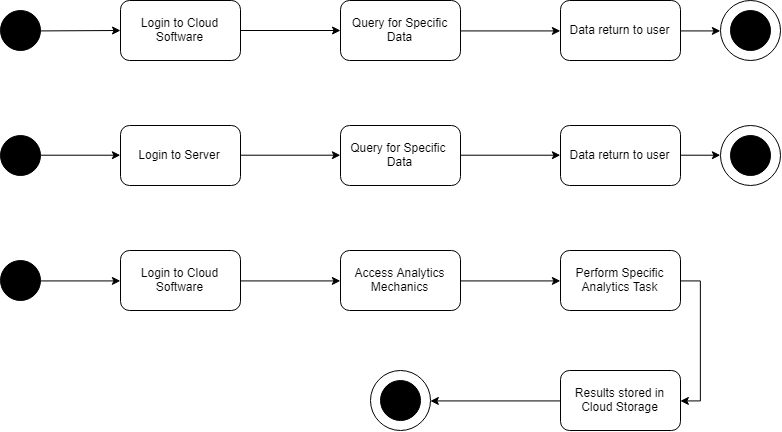


**Activity Diagram**

Professor View (leads into backend):

****

Backend (led in by Professor/Student views):

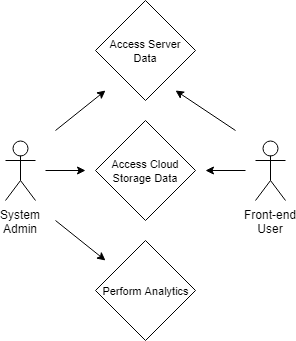


**GUI**

In separate document attached.

**SUBSYSTEM: Data Storage/Manipulation** (Alexander Darden)

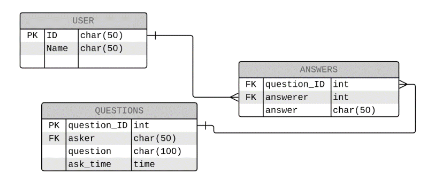
**Use-Case Diagram**



**Use-Cases**

1. Use case: Access Server Data
   1. Actors: System admin, Front-end user
   2. Preconditions: Server must have received data from front-end application
   3. Related use case: Access Cloud Data
   4. Steps:
      1. Login to system
      2. Query for a specific data set
2. Use Case: Access Cloud Data
   1. Actors: System admin, Front-end user
   2. Preconditions: Cloud storage system must have received data from the server and analytics
   3. Related use case: Access Server Data
   4. Steps:
      1. Login to cloud system
      2. Query for a specific data set
3. Use Case: Perform Analytics
   1. Actors: System admin
   2. Preconditions: Data must have been received from the cloud storage unit
   3. Related use case: none
   4. Steps:
      1. Model Data
      2. Further analyze using classification algorithms

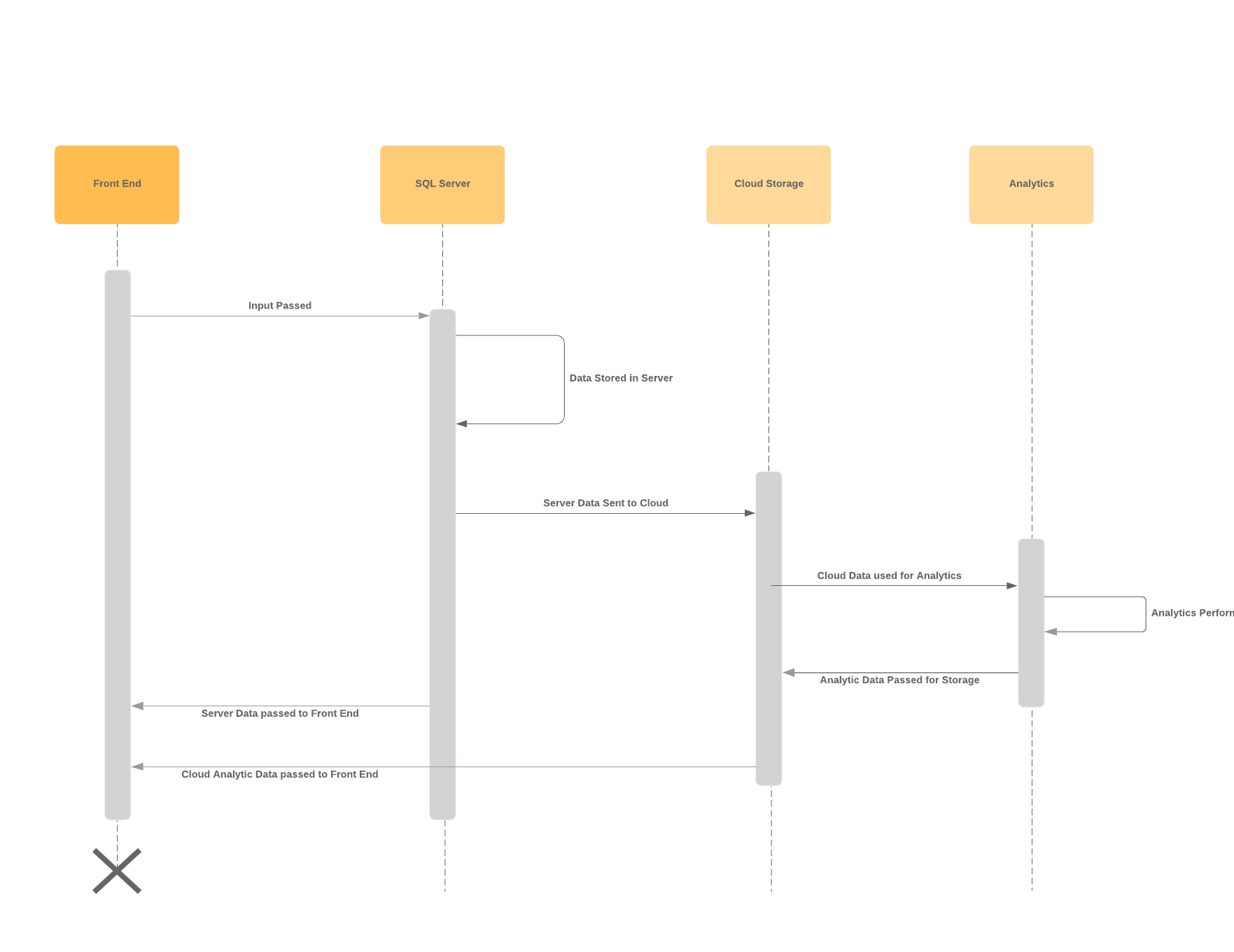
**Class Diagram**



The three relational tables listed above will be utilized for this subsystem. Each one’s data that is stored in this subsystem is detailed below.

1. User
   1. The data stored from this class is ID and Name
2. Answers
   1. The data stored from this class is question\_ID, answerer, and answer
3. Questions
   1. The data stored from this class is question\_ID, asker, question, and ask\_time

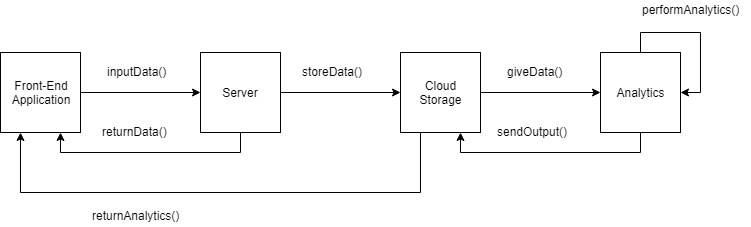
**Sequence Diagram**



There are four parts to this sequence diagram: the front-end application, the SQL server, cloud storage, and analytics. Each step of the sequence is outlined below.

1. Input is passed from the front-end application to the SQL server
2. The SQL server stores the data in tables designed specifically for the front-end application
3. The SQL server has its data also sent to a cloud storage service (in this case AWS)
4. The cloud storage then provides the data for the analytics program used
5. The analytics program performs whatever process is specified by the system administrator (i.e. modeling, classification, etc.)
6. The results of these processes are sent back to the cloud storage for future retrieval and data backup
7. The front-end application receives data from the server when accessed by the user
8. The front-end application receives data from the cloud storage (regarding the analytics processes) when accessed by the user

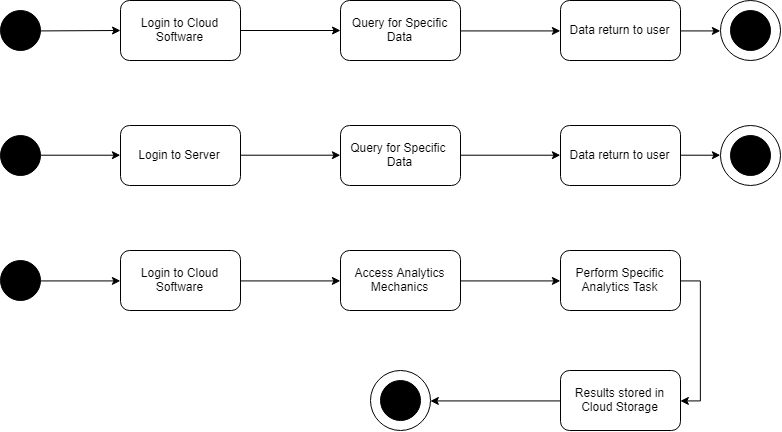
**State Diagram**



The state diagram describes the functions used throughout the processes of this specific subsystem. Each process is detailed below.

1. inputData()
   1. This function sends input data from the front-end application to the server
2. storeData()
   1. This function sends data from the server to be placed in cloud storage
3. giveData()
   1. This function gives data from the cloud storage to an analytics program
4. performAnalytics()
   1. This function is an umbrella term for the various processes that can be performed depending on the specific program and need of the system administrator
5. sendOutput()
   1. This function returns the results of whatever analytics process occurred to the cloud storage for placement and further usage
6. returnData()
   1. This function returns data stored in the server to the front-end application
7. returnAnalytics()
   1. This function returns the results of any analytics processes performed to the front-end application

**Activity Diagram**



There are three activity diagrams displayed above (for the three use-cases of this subsystem). Each diagram is detailed below.

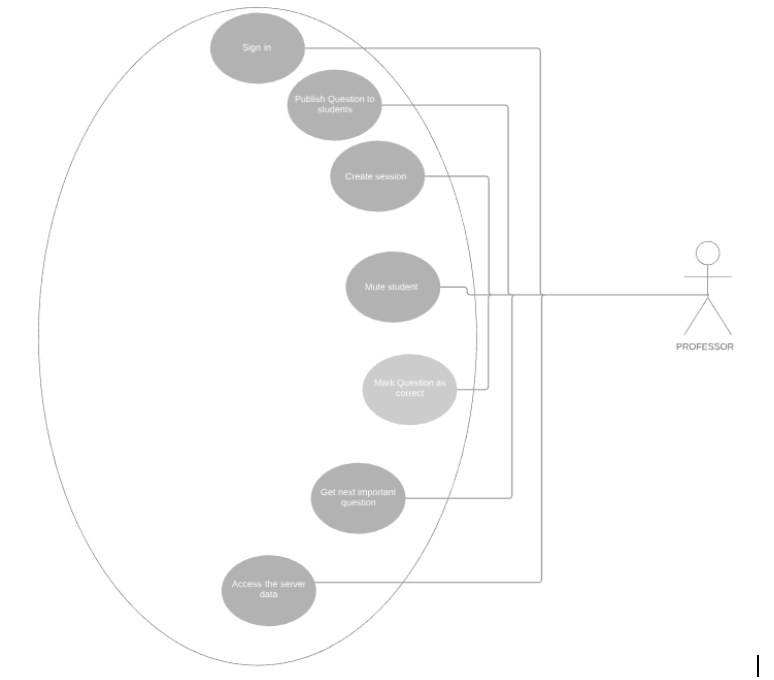
1. Top Diagram (Access Cloud Storage Data)
   1. The user logs into the Cloud Storage software used (AWS)
   2. The user queries for a specific subset of data from the database
   3. The data is returned to the user for use
2. Middle Diagram (Access Server Data)
   1. The user logs into the Server software used (Microsoft SQL Management Studio)
   2. The user queries for a specific subset of data from the database
   3. The data is returned to the user for use
3. Bottom Diagram (Perform Analytics)
   1. The user logs into the Cloud Storage software used (AWS)
   2. The user accesses the Analytics software provided
   3. The user performs a specific analytics process (i.e. modeling, classification)
   4. The results of the specific process are stored in the Cloud Storage software

**GUI**

The Graphical User Interface (GUI) for this specific subsystem is not specific to the main application (mASK). The system administrator uses both AWS and the Microsoft SQL Management Studio application in order to access, manipulate, and store the data.

**SUBSYSTEM: Professor** *(McKenzie Moize)*

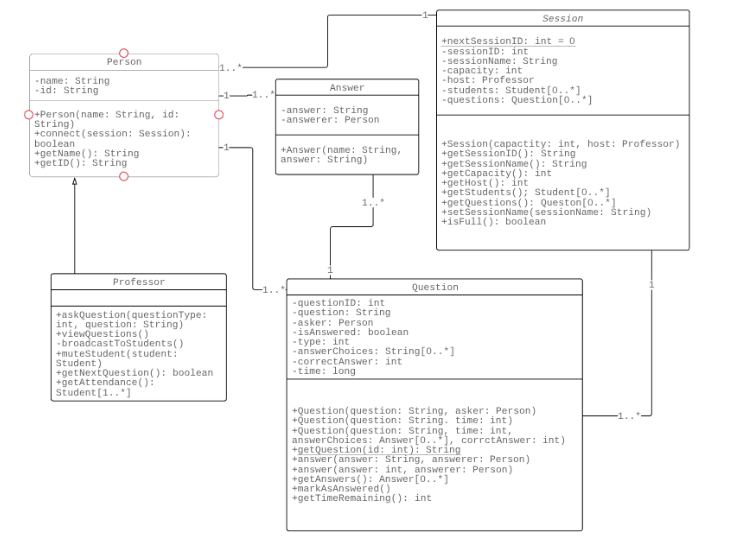
**Use-Case Diagram**



**USE CASES FOR PROFESSOR**

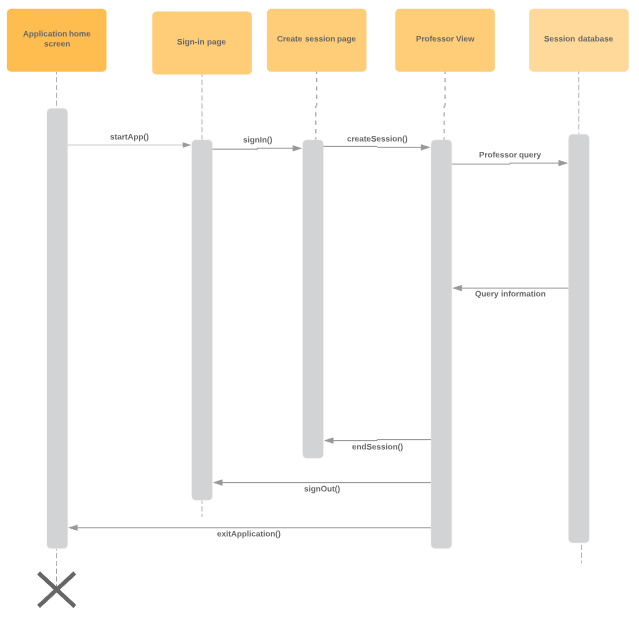
1. Use case: Sign In
   1. Actors: Student, Professor
   2. Preconditions: Must have application downloaded and open
   3. Related use case: Joining a session
   4. Steps:
      1. Enter name
      2. Enter username
      3. Press sign in button
2. Use case: Publish question to student
   1. Actor: Professor
   2. Preconditions: Must have created a session
   3. Related use case: Ask a question
   4. Steps:
      1. Type a question
      2. Select question type
         1. If multiple choice, enter choices and correct answer
      3. Select time limit for question
      4. Press send to broadcast to all students
3. Use case: Create session
   1. Actor: Professor
   2. Preconditions: Must have signed into the application
   3. Related use case: Joining a session
   4. Steps:
      1. Enter session capacity
      2. Enter session name
      3. Press launch to launch the session
4. Use case: Mute student
   1. Actor: Professor
   2. Preconditions: Have session running and have at least one student signed in
   3. Related use case: None
   4. Steps:
      1. Select student
      2. Select mute student
         1. Optional: Explain why student is being muted
      3. Press mute to save selection
5. Use case: Mark question as correct
   1. Actor: Professor
   2. Preconditions: Have a question that has been answered
   3. Related use case: Access the server data
   4. Steps:
      1. Select question
      2. Select correct answer
      3. Select mark question as correct
      4. Save selection
6. Use case: Get next important question
   1. Actor: Professor
   2. Preconditions: Have a question in the database
   3. Related use case: Access the server data
   4. Steps:
      1. Press button to receive next important question
         1. If no questions exist, continue until one is asked
      2. Display the question
7. Use case: Access the server data
   1. Actor: Professor, Student, System Admin
   2. Preconditions: Have a database server set up
   3. Related use case: Get next important question
   4. Steps:
      1. Press button for desired result including input where necessary
      2. Press submit to view server data

**Class Diagram**



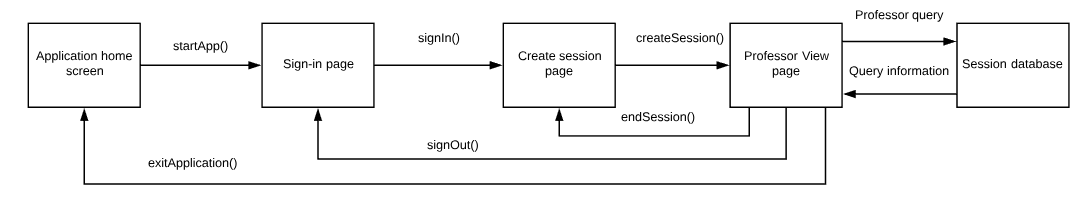
The Professor subsystem will use the various methods in the Professor and Session class to moderate and administrate the instance of the session they create. They will also evaluate Student answers for analytics.

**Sequence Diagram**

****

The above sequence diagram shows the sequence of events for a professor using mASK. The professor will start on their respective home screens and then start the app. Then they will be led to a sign in page where they will provide their name and username. Once they are signed in they will be led to the create session page, where they will provide the session information to create a session. Once a session is created, they will be led to the professor-specific view. They will be able to use various buttons and inputs to query the session database for information, which will then be returned to display on the professor view. They will have the option to end their session from the professor view, which will take them back to the create session page, sign out of their session which will take them back to the sign in page, or exit the application which will take them back to their respective application home page.

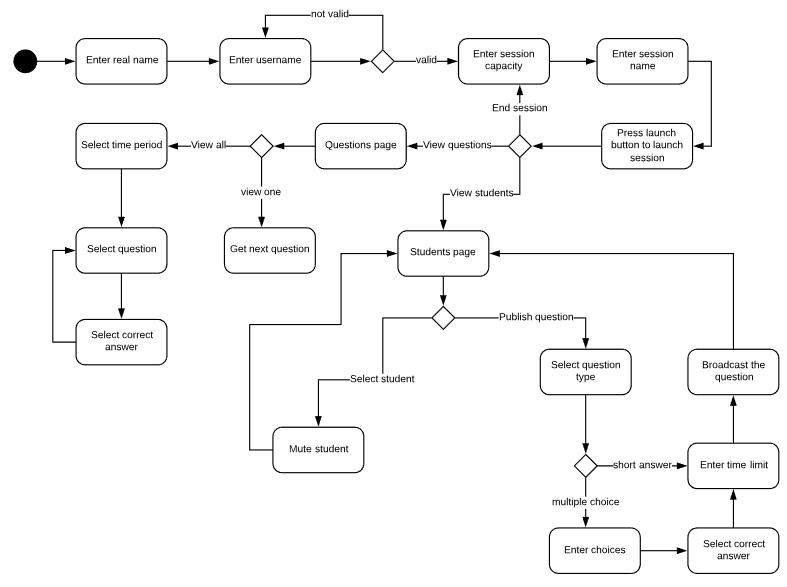
**State Diagram**



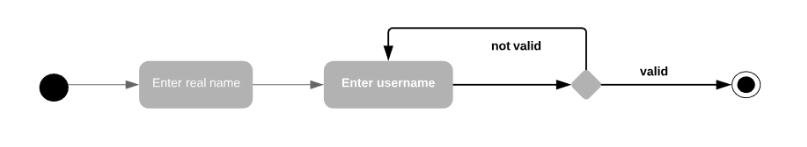
The state diagram describes the functions used throughout the subsystem. Each function is detailed below:

1. startApp()
   1. This function will launch the app from the users home screen
   2. They will be directed to the sign-in page
2. signIn()
   1. This will prompt the user to enter their name and username
   2. If the username is valid, they will be signed into the application and will be led to the create session page
3. createSession()
   1. The user will provide the page with the desired capacity and name of the session
   2. If the input is valid, the session will be created and the user will automatically be installed as the session host
   3. They will be directed to the professor view page
4. Professor query
   1. The professor can use the various forms of input to process various queries on the users
      1. View the student questions
      2. Mark answers as correct
      3. Ask questions of the students
      4. Get important question from server
      5. Mute students
   2. These queries will be sent to the session database, which will run queries defined in the classes to determine the desired output
5. Query information
   1. The database will run the SQL commands and algorithms to return the desired output to the Professor
6. endSession()
   1. The professor will be able to press a button to end the session from this page, which will redirect them back to the create session page
7. signOut()
   1. The professor will be able to sign out from this page through a specified button, which will redirect them back to the sign in page
8. exitApplication()
   1. The professor will be able to press a button to exit the application, which will redirect them back to their home screen

**Activity Diagram**

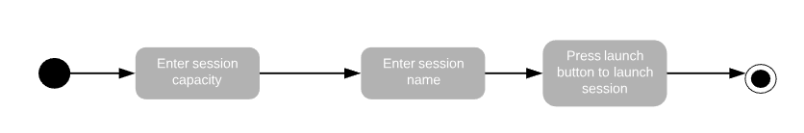
****

**Activity diagram for Signing In**



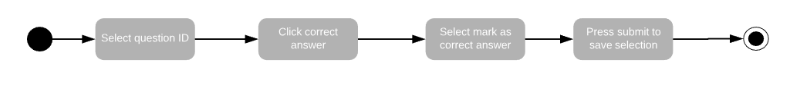
The above activity diagram depicts the activity stages for signing in. The user will enter their real name to be seen only by backend information. The user will then enter a username that will be displayed when questions are asked and answered. The activity will return to enter a username if the one provided is not valid.

**Activity diagram for launching a session**



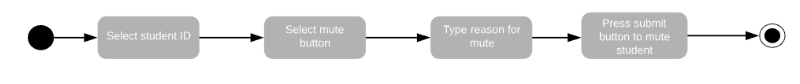
The above activity diagram is used for launching a new session. The user will enter the capacity for the session, followed by the public name of the session. The session name will usually contain a descriptor for the session such as the class name or the topic of the day’s lecture. Then the user will press the launch button to launch the session.

**Activity diagram for marking an answer as correct**



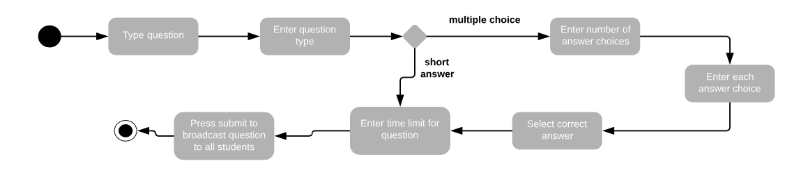
The above activity diagram is used for a Professor marking a student answer as correct. The professor will select the question, click the answer for the question, press the checkbox indicating that the answer is correct, then press the submit button to save the selection.

**Activity diagram muting a student**



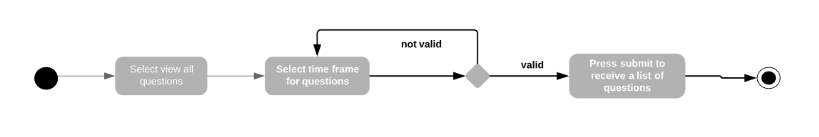
The above diagram is used for muting a student. The professor will select the student in question, then select the mute button to indicate they intend on muting them. A short menu will appear where the professor will be able to optionally tell the student why they are being muted. Then the professor will press the submit button to mute the student.

**Activity diagram for asking the students a question**



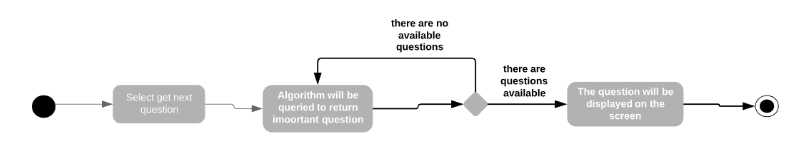
The above activity diagram is used by professors when they wish to broadcast a question to the students. The professor will type a question and select the question type. If the question is multiple choice, they will be prompted to enter the number of choices and then provide the answer choices. They will also have to click a checkbox next to the correct answer so the system will know what is right. After they have filled out the multiple choice information, or if the question type was short answer, the professor will enter the desired time limit for the question. The professor will then press the submit button to broadcast the question to all the students.

**Activity diagram for viewing all questions**



The above diagram is used for the user viewing a list of questions. They will input the desired time period and, if it is valid, will be given a list of questions on pressing the submit button.

**Activity diagram for getting next important question**



The above diagram is used for the professor to get the single best question back from the database. The professor will select a button labeled get next question and the database will use an advanced machine learning algorithm to determine what the best available question is and return it to the user. If there are no questions available it will loop indefinitely until there is one. This feature can be used to show questions in real time, as they will be displayed as soon as they are asked.

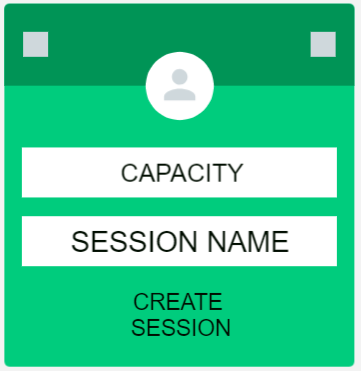
**GUI**

**GUI for Sign-in page**

****

The user will enter their name and desired username and will be signed into the application.

**GUI for create session**

****

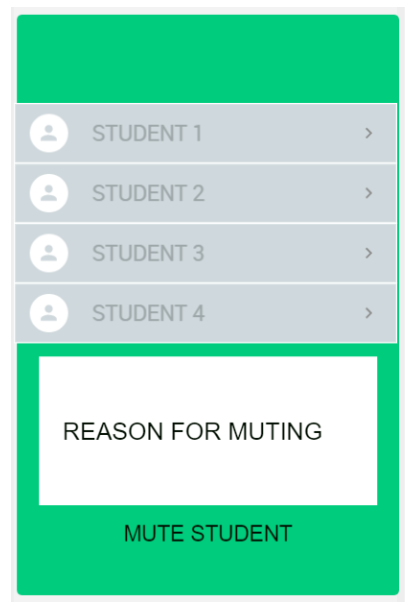
The user will enter the capacity for the session and the session name. They will then press the create session button to create the session.

**GUI for get next question**

****

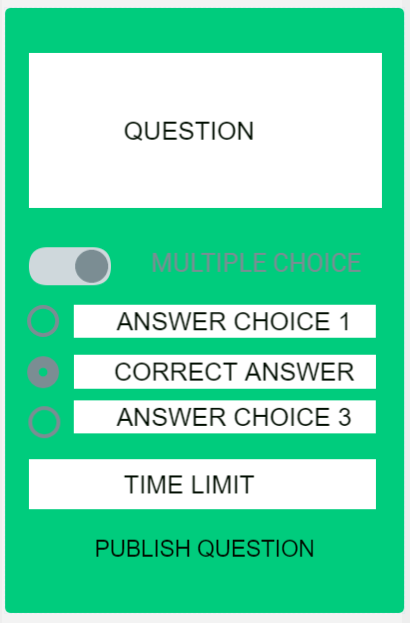
The user will press the button to see the next question on their screen.

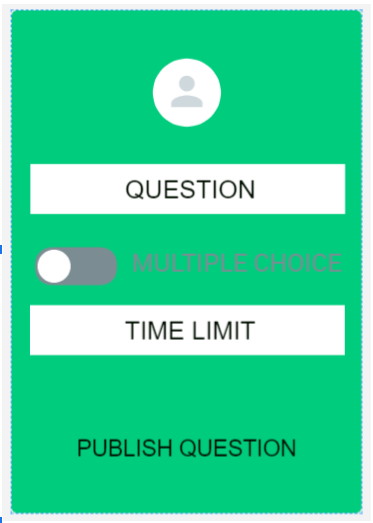
**GUI for muting a student**



The user will select the student from the list of students given. They will be able to optionally enter a reason for muting the student. They will press the mute student button to actually mute the student.

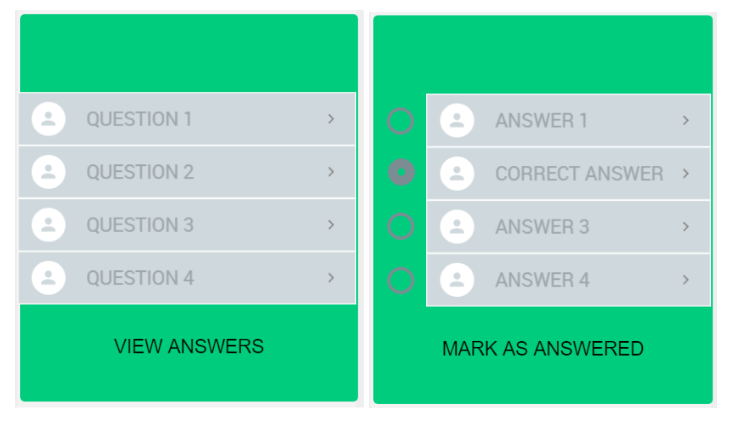
**GUI for publishing a question**

****



The GUI on contains a field for entering the question and time limit for the question. There is a toggle button for selecting whether or not the question is multiple choice. If it is, the user will enter their choices and mark the select answer from the radio buttons given.

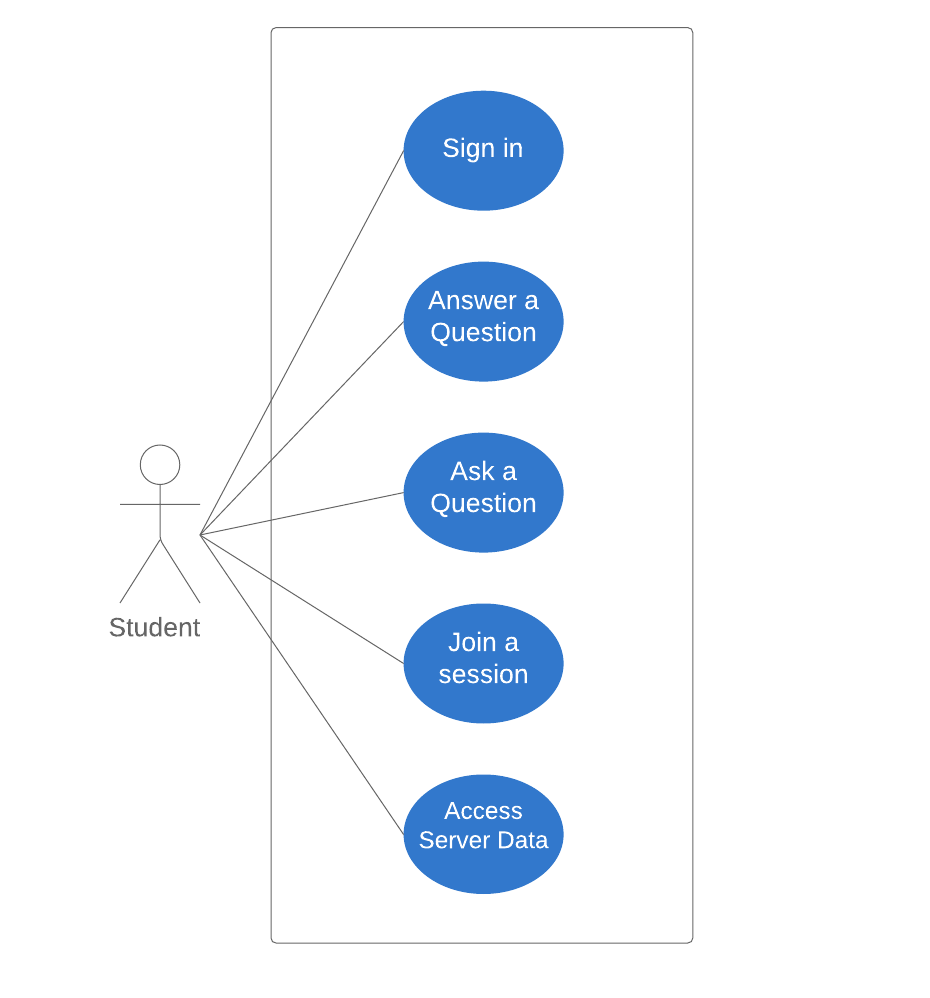
**GUI for viewing questions**



The user will see a list of questions that have been asked by the students and can select the individual question to view the answers proposed by other students. When the answers are shown they will be able to select an answer and mark it as correct.

**SUBSYSTEM: Student** *(Emily Ashburn)*

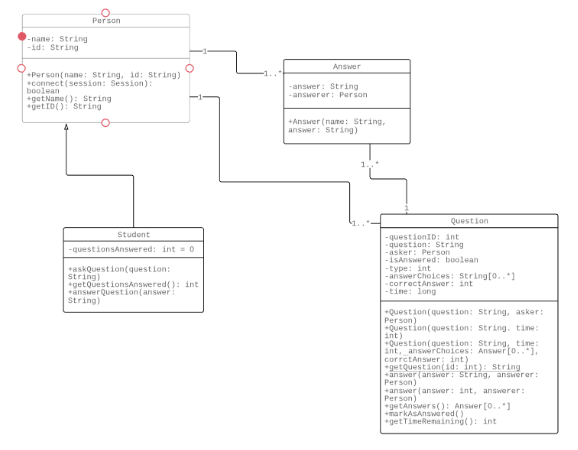
**Use-Case Diagram**

****

**Use-Cases**

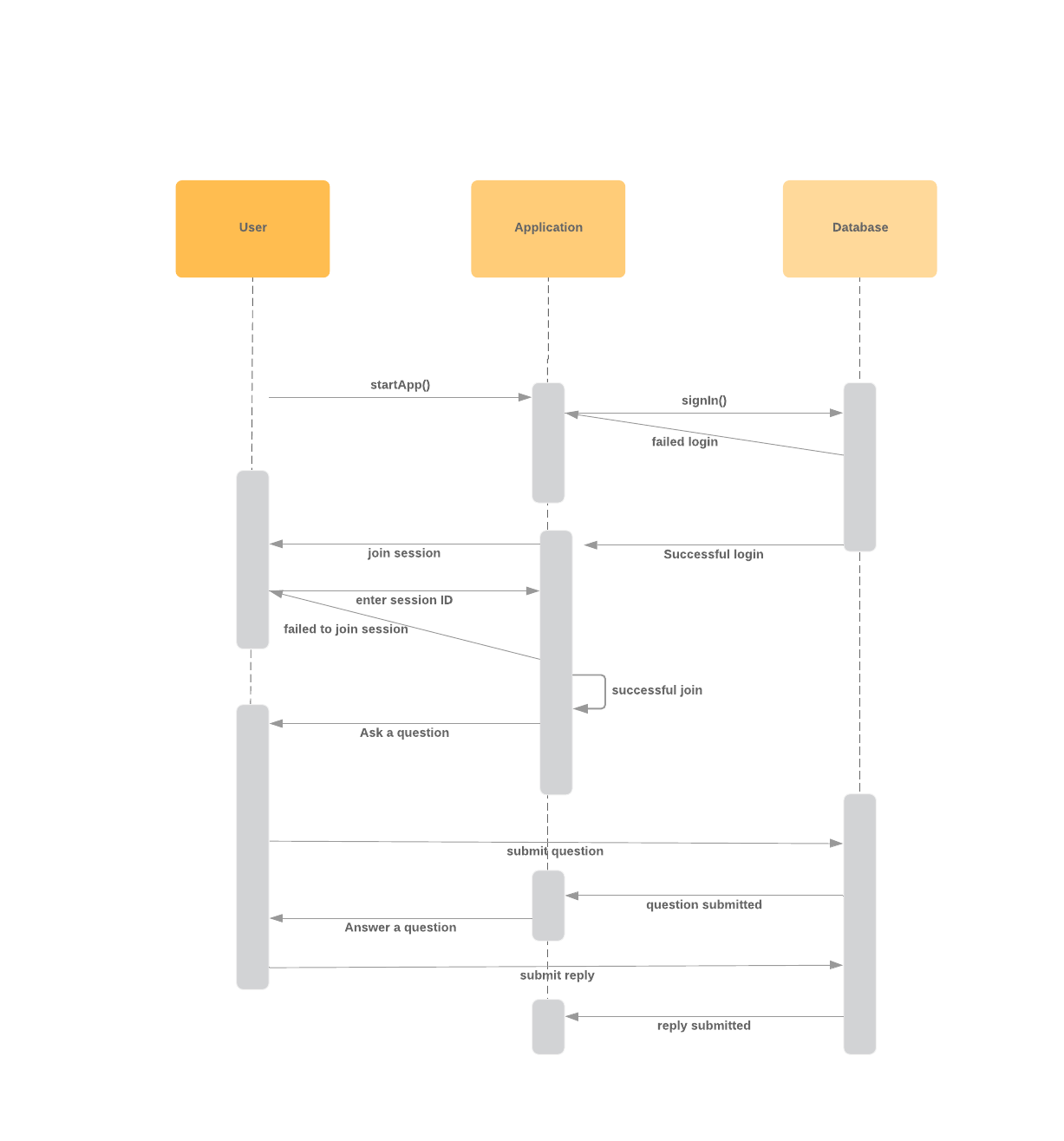
1. Case: Sign-in
   1. Actors: Student, Professor
   2. Preconditions: App must be loaded on user’s device and on the home screen
   3. Related use case: None
   4. Steps:
      1. Provide username and password
      2. Press enter for verification
2. Case: Answer a Question
   1. Actors: Student
   2. Preconditions: Must be on a question’s page where the “answer question” button is
   3. Related use case: Ask a question
   4. Steps:
      1. Type out answer for a question in text box provided
      2. Press submit
3. Case: Ask a Question
   1. Actors: Student
   2. Preconditions: Be on the event screen that prompts user to ask a question
   3. Related use case: Answer a Question
   4. Steps:
      1. Type out question in text box provided
      2. Press submit
4. Case: Join Session
   1. Actors: Student
   2. Preconditions: Must be in a valid environment for server presence and on the “join session” screen
   3. Related use case: Create Session
   4. Steps:
      1. Get session information and provide it in the application
      2. Press submit and session should be joined
5. Case: Access Server Data
   1. Actors: System admin
   2. Preconditions: Server must have received data from front-end application
   3. Related use case: Access Cloud Data
   4. Steps:
      1. Login to system
      2. Query for a specific data set

**Class Diagram**

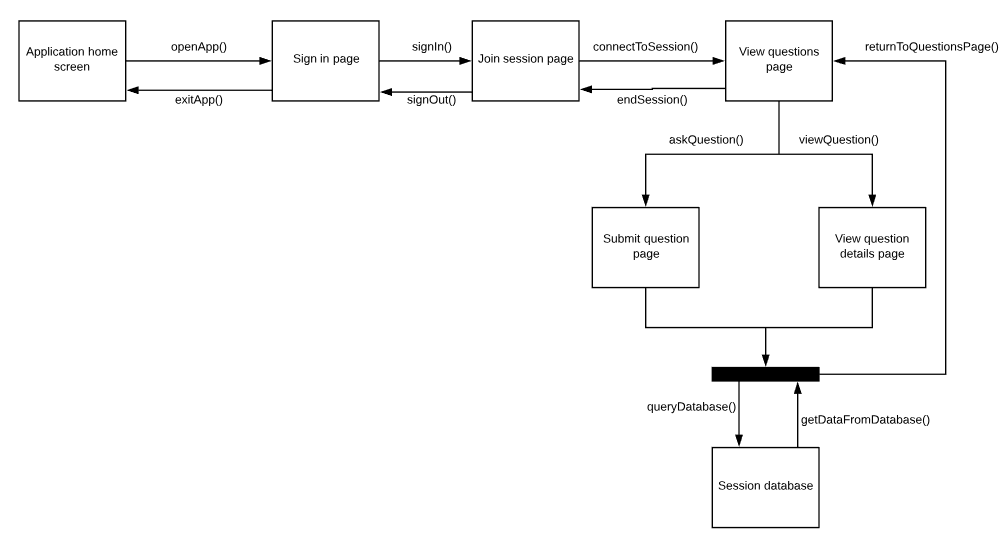


The student subsystem will utilize the methods contained in the Student subclass as well as the methods in the Question class to answer questions posted by the Professor and other Students as well as ask their own questions.

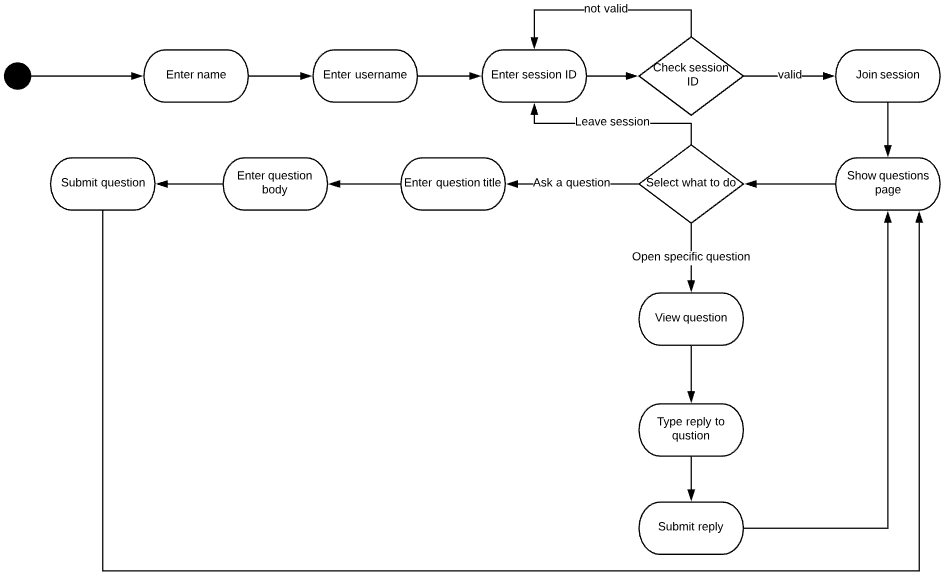
**Sequence Diagram**

****

**State Diagram**



**Activity Diagram**



**GUI**

