**University Student Data System**

The project described in this report is a simple database access system regarding “student” information at a hypothetical “university”. The idea was to create a system where an “Admin” can log into a website, view students’ information, and edit/delete student entries. On top of that, hypothetically, a “Database Admin” can log into the system and edit/delete Admin entries. The actual implementation in the code is that of an Admin being able to create a new Student entry and viewing the populated database.

**Group Members:**

**James Paule:**

Contributions: Co-wrote Use Case Diagram and Sequence Diagram.

Wrote appropriate use case specifications and co-wrote state diagrams.

**Harry Allen:**

Contributions: Co-wrote Use Case Diagram and Sequence Diagram.

Wrote appropriate use case specifications and co-wrote state diagrams.

**Cole Lewis:**

Contributions: Co-wrote Use Case Diagram and Sequence Diagram.

Wrote appropriate use case specifications and co-wrote state diagrams.

**Jacob Siau:**

Contributions: Wrote and implemented the database system with Eddie Gonzalez.

Co-wrote the class diagram with Eddie Gonzalez.

**Eduardo Gonzalez, Jr.:**

Contributions: Wrote and implemented the database system with Jacob Siau.

Co-wrote the class diagram with Jacob Siau.

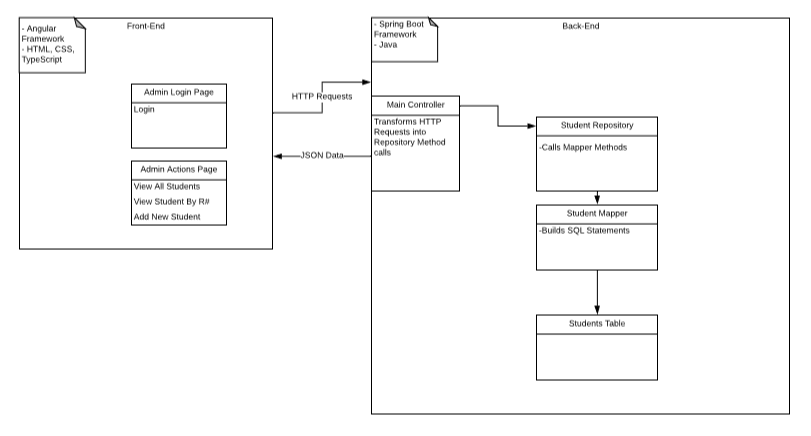
**Project Description:**

The system is designed to store student information in a central database for university administration purposes. The design allows for easy access of student information. Any Admin with access to the universities network can send requests to receive or edit student information. Additionally, adding or deleting students is another feature. For security purposes login credentials would be required on the front webpage to interact with the system. This prevents unauthorized access to sensitive information. Furthermore, one person would be able to be deemed a System/Database Admin. Their added ability apart from a normal Admin is the ability to add, edit or remove Admins to the system. A Database Admin also has the ability to delete the database if needed.

The management of the system consists of a central database, website (UI) and a network for communication between the two.

**Please note**: The implemented cases in the system are functionality for an admin to view all students and create a new student. The system is easily scalable to add the extra functionality previously discussed for a database admin to create/delete university admins from the admins table. For grading purposes, consider the main use case to be a registered user (admin) creating a new student in the students table, and viewing said table upon successful creation.

**Architecture of the System:**



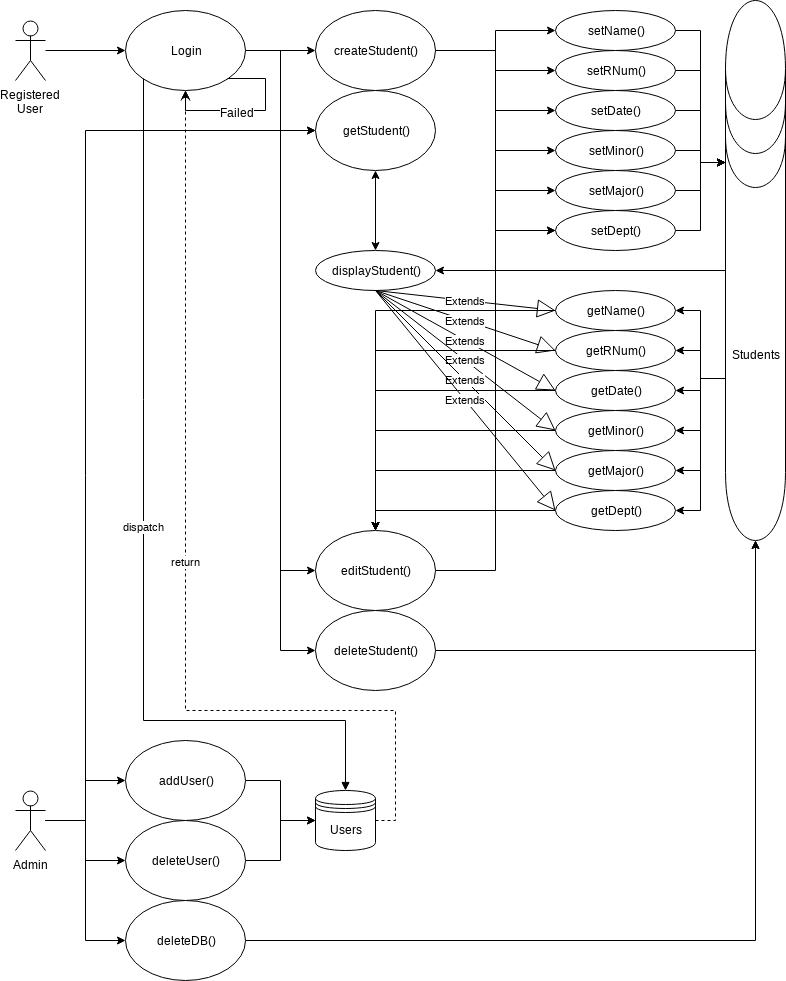
**Further Architectural Details:**

The system consists of an Angular Frontend and a Spring Boot backend, with a Hibernate in-memory database. The frontend takes user actions on the webpages and makes corresponding HTTP requests such as GET and POST to the backend controller. The controller then transforms those requests into method calls in the Student Repository, which calls the Student Mapper interface methods. These mapper interface methods prepare SQL statements to interface with the Students table to set/retrieve data.

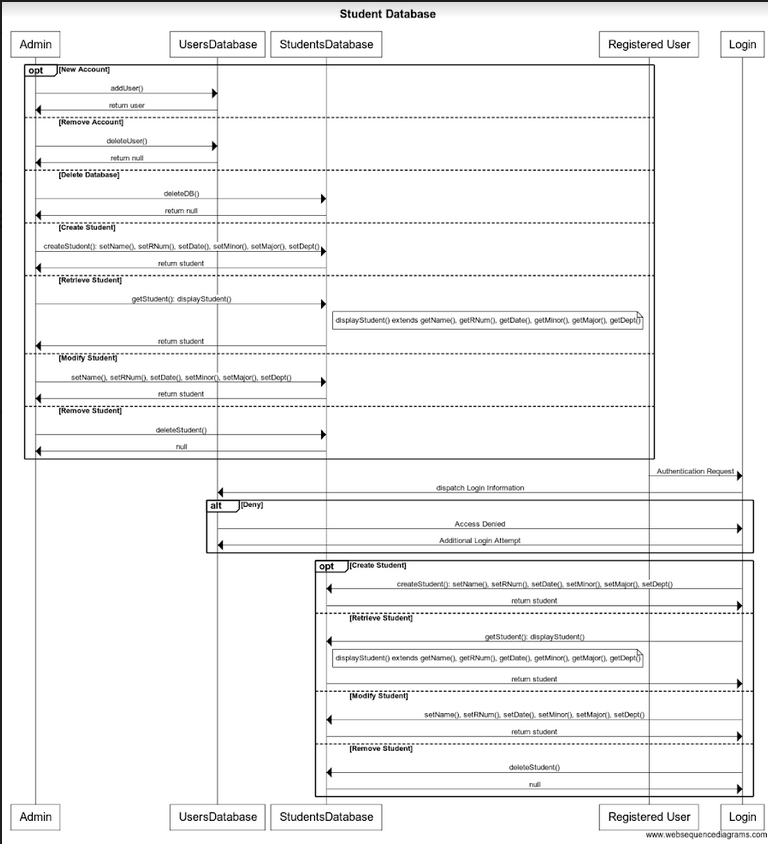
**Design Pattern Used: Model-View-Controller**

We implemented the MVC design pattern in our system. The Model is the Student data that is marshalled from the Students table by the aforementioned entities. The Controller is the Main Controller of the application, which takes the HTTP requests from the front end, and transforms them into method calls to retrieve Student data. The View is represented by the front-end webpages.

**Overall Use Cases Diagram:**

****

**Overall Sequence Diagram**

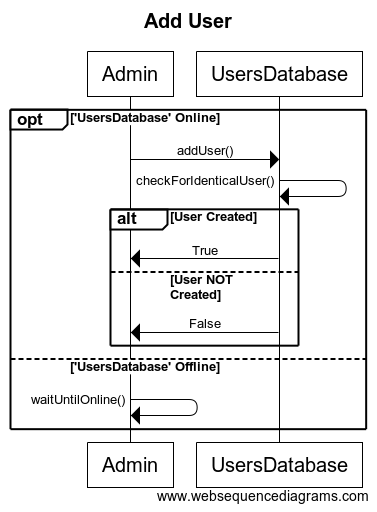


**Use Case Specifications, Sequence Diagrams, State Diagrams:**

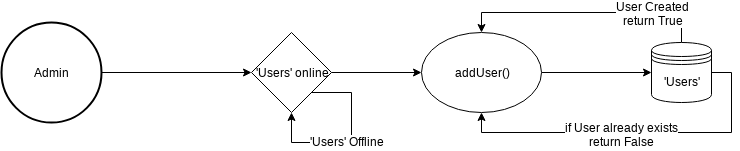
**Use Case 1: DB Admin Add User(Admin)**

|  |  |
| --- | --- |
| **Actor(s)** | Administrator |
| **Summary Description** | Adds a user previously unknown to the ‘Users’ database, that will allow access into the ‘Student’ database |
| **Priority** | Essential to system function |
| **Status** | Low Level of details |
| **Pre-Condition(s)** | Administrator must be the accessor  ‘Users’ database must be online and active |
| **Post-Condition(s)** | User added to ‘Users’ database |
| **Basic Path** | 1. ‘Users’ database online 2. Admin accesses database 3. Admin selects ‘Add User’ 4. Admin provides details for new user 5. ‘Users’ database checks of identical users 6. ‘Users’ database returns true once new user is created |
| **Alternative Path(s)** | 1a. Database offline  2a. Access error returned  4a. Admin does not give enough information to all the fields  6a. Returns false if user already exists |
| **Non-Functional Requirement(s)** | NF1: Access timeout  NF2: Illegal field entry (email in name, vice versa)  NF3: Language support  NF4: Timezone support |

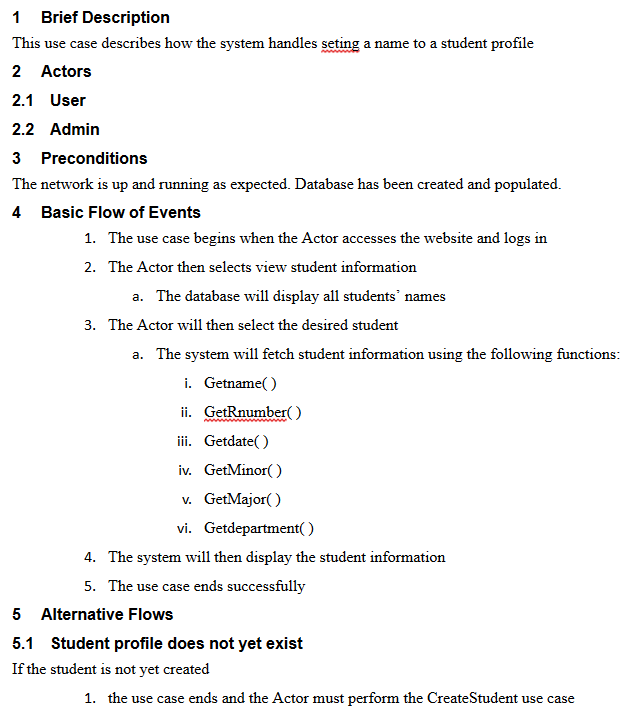
**Sequence Diagram 1:** **Add User**



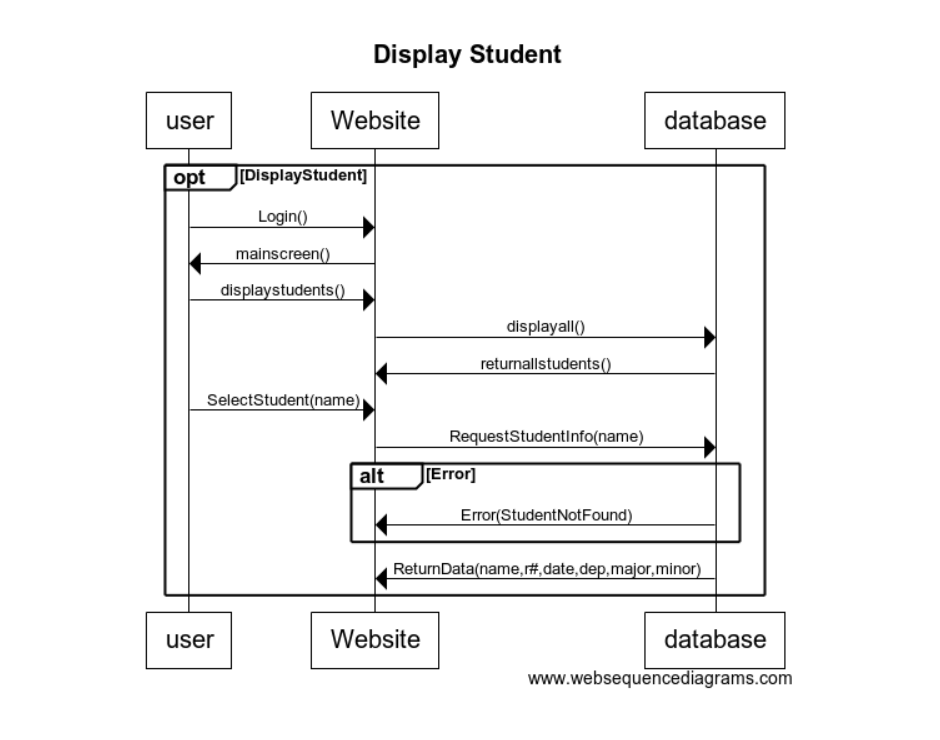
**State Diagram 1: Add User**



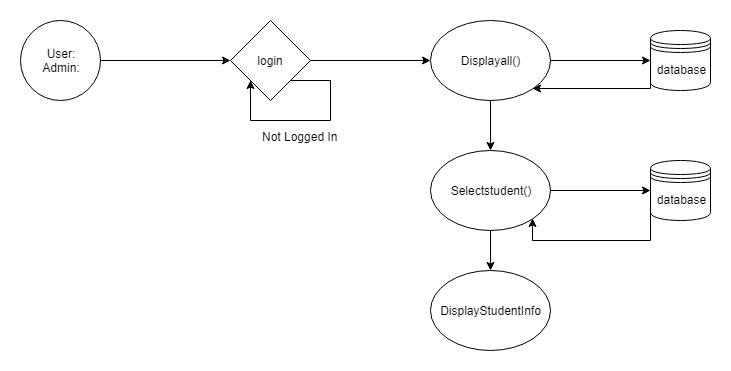
**Use-Case 2: Get Student**



**Sequence Diagram 2: Display Student**



**State Diagram 2: Display Student**



**Use Case 3: Create Student**

Brief Description: This use case allows the admin to create a new student record into the database

Actors: Admin

Preconditions:

- The admin is signed in

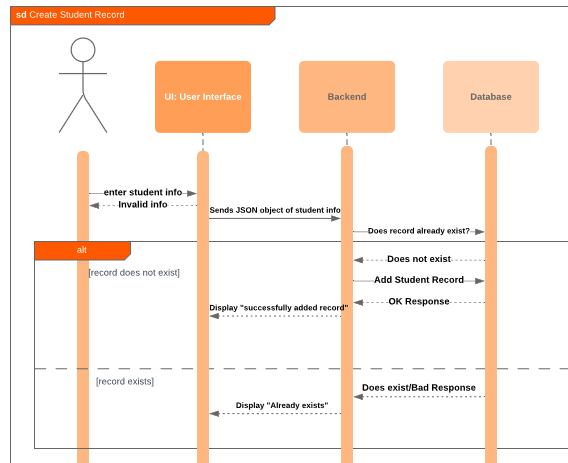
Description:

The admin can enter all the necessary fields to create a student, such as R#, first name, last name, graduation date, department, major, minor, username, email, password. When submit is clicked, it will then send that information to the backend and store it into the database.

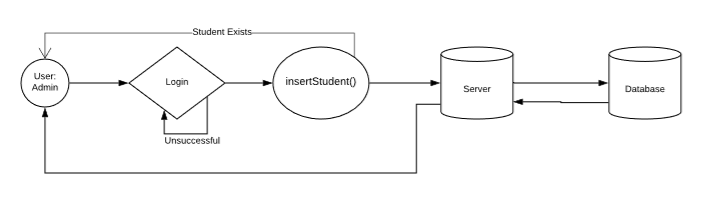
Alternatives: The student info that the admin entered is already in the database, so there will be an unsuccessful message.

Post-Conditions: The admin will be redirected to the home page where they can view the updated student records table.

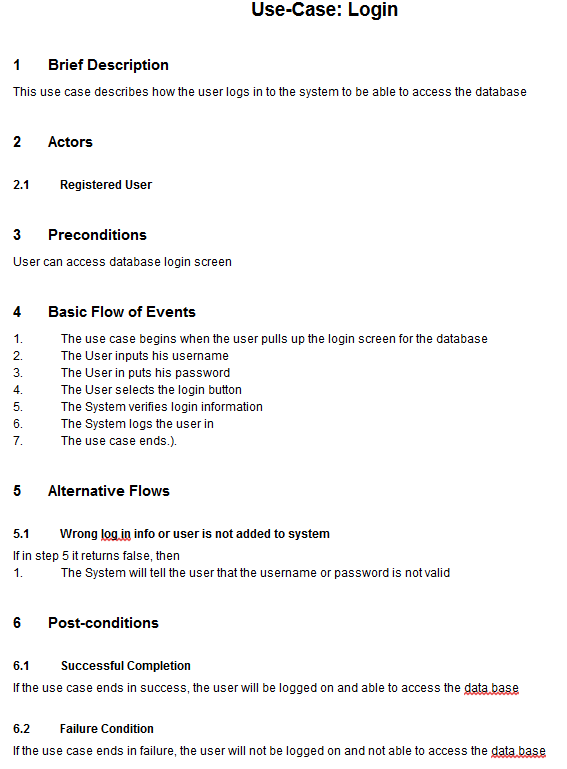
**Sequence Diagram 3: Create Student**



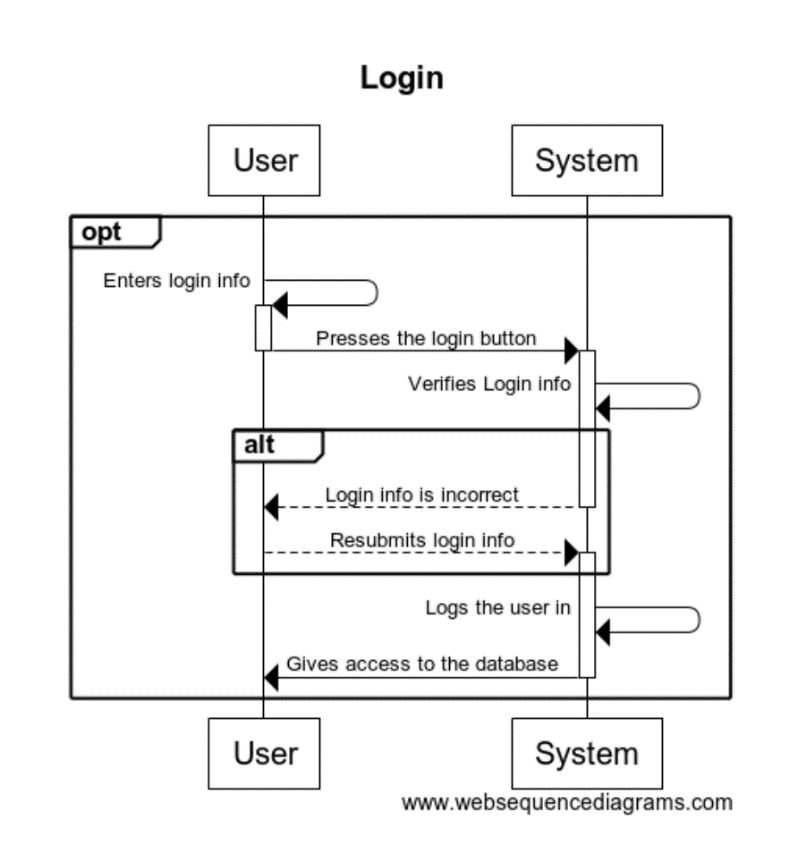
**State Diagram 3: Create Student**



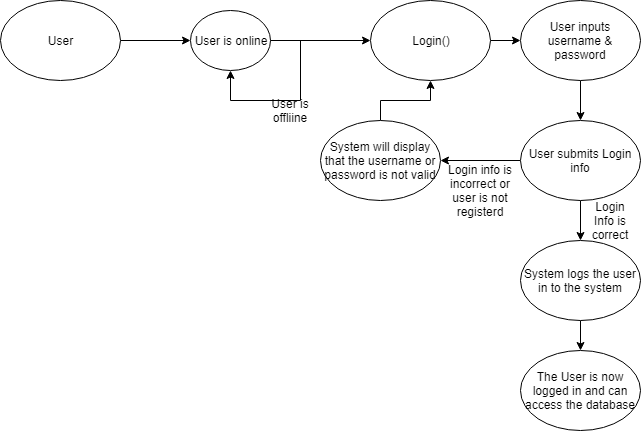
**Use Case Specification 4: Admin Login**



**Sequence Diagram 4: Admin Login**



**State Diagram 4: Admin Login**



**Use Case Specification 5: Delete Student**

Brief Description: This use case describes the actions of the system handling the deleting of a Student entry by an Admin.

Actors: Admin

Preconditions: Server on. Admin Logged in.

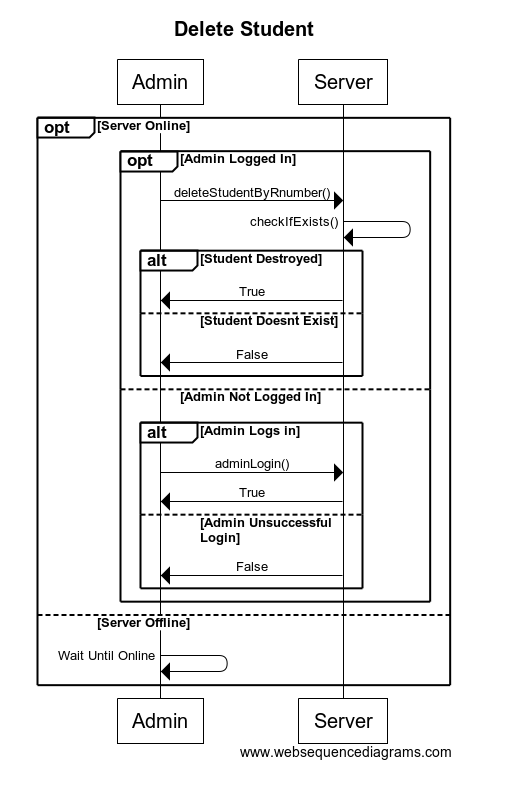
Basic Flow of Events:

* + - 1. The use case begins when the Admin accesses the website and logs in.
      2. The Admin then would select to delete a student by R#.
      3. The Admin would then type a valid R#, validated by the front-end.
         1. If invalid, the system would not do anything else.
      4. The Admin would then submit the R# of the student to be deleted.
      5. The Server would look up the Student in the Database by the R# and check if it exists.
         1. If not exists, returns error
         2. If exists, removes the entry successfully
      6. The Use case ends successfully

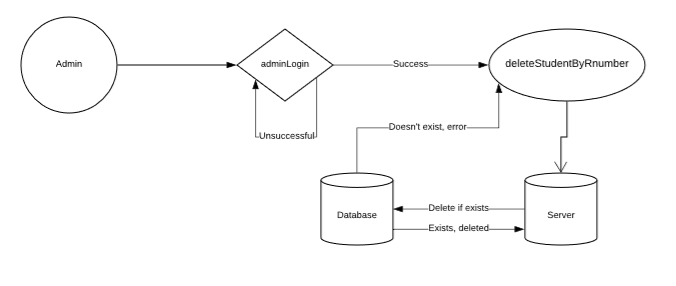
Alternative Flows:

* + - 1. Student profile nonexistent
         1. Then the admin would have nothing else to do.

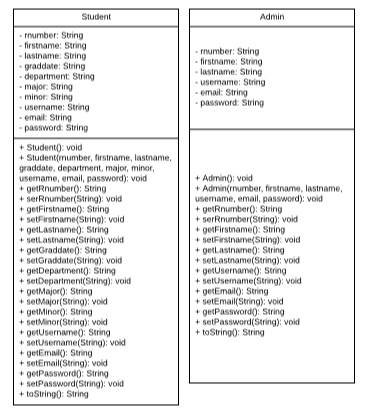
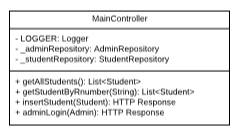
**Sequence Diagram 5: Delete Student**

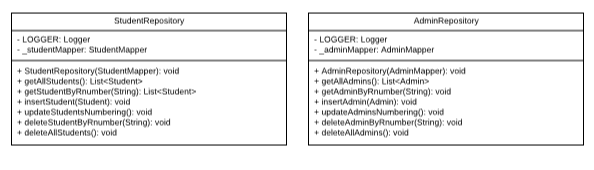


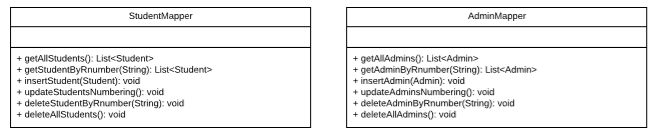
**State Diagram 5: Delete Student**



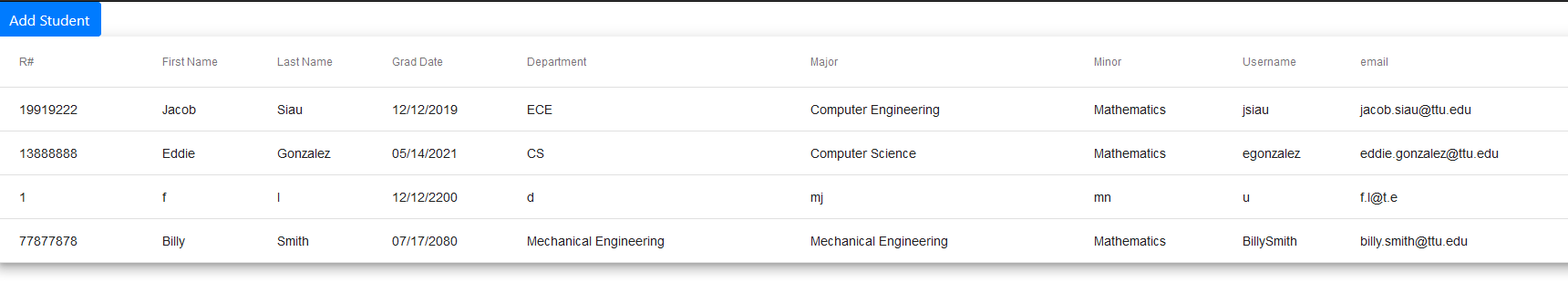
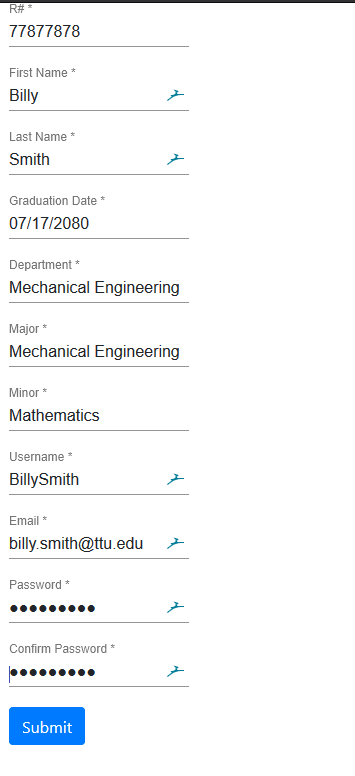
**Class Diagrams:**







**Actual Implementation Example**



**Example 1 Above:**

Let this use case be the “coded” implementation of Insert Student. Technically, the home page automatically displays all student info in the database, so that would be able to count for the “display all” use case; however, please consider on our behalf the “Insert Student” use case as shown in the pictures. The test student, “Billy Smith”, has their information put into the front-end form by an “Admin”, and submitted. The database then inserts the entry, as can be seen with the other, pre-existing entries in the Students database.

**User/Developer Guide**

You can see the proof in the picture above, but if you are needing to view the application yourself, you will need to take a few steps.

Please consider emailing [jacob.siau@ttu.edu](mailto:jacob.siau@ttu.edu) for assistance. Jacob is willing to come in with his laptop (where the code was eventually put together) and do a demo for the grader if needed. There are a lot of steps needed to run our project on another platform, since it is a full-stack web application that has not been exported to a JAR or a small, standalone package yet. It is a two-part project consisting of an Angular front-end API consuming a RESTful Spring Boot API, in conjunction with a Hibernate in-memory database. There are a lot of configuration steps needed, so please, contact Jacob Siau [jacob.siau@ttu.edu](mailto:jacob.siau@ttu.edu) for assistance in seeing the live project demo if needed.