Design Document
Classroom360

Document by:

- Avala Chaitanya Lakshmi Prakash Yadav: 22114016
  - o avala_clpy@csiitr.ac.in
- Bezawada Sri Sai Anurag: 22114021
  - o bezawada_ssa@cs.iitr.ac.in
- Dharavath Madhu: 22114028
  - o dharavath_m@cs.iitr.ac.in
- Ogireddy Sree Sudeep Reddy: 22114063
  - o ogireddy_ssr@cs.iitr.ac.in
- Seemakurthi Nandan Sri Siva Ramakrishna: 22114089
  - o seemakurthi_nssr@cs.iitr.ac.in

1. **Summary:**

This document consists of a flowchart, DFD, use-case diagram and class UML diagram. So the user can have 3 possible roles and they have their own choices/ interface for their respective roles. It is clearly depicted in the flowchart of how the user interacts. We also segregated the different functionalities of different roles in the use case diagram. We listed 6 different models(databases) shown in the DFD diagram. In the DFD diagram, we specified how entities interact with the database and processes. Finally, in the class UML diagram, we classified the whole project into 11 classes, with the main class being the user class. Proper relations and their accesses are specified in the diagram. Our main objective was to see the whole thing split into the 3 roles.

2. **Questions/Comments:**

1. In making of class UML diagram:
    - So firstly, we need a main class. This is depicted by the user class having common information to all 3 roles.
    - How to put information specific to roles? In the diagram, it is clear that we created 3 classes- admin, student and teacher inherited from the user class. Then we added their unique properties.
    - Next, how to manage courses? We added a courses class which is aggregate to the teacher class. So, if the teacher object is destroyed, the course object survives as it can be given to some other teacher. The courses object has a display function kept public so anyone can access it. Some functions such as creating and deleting courses, and deleting courses are confined to the teacher's class.
    - To manage class resources, attendance, rooms and assignments, we gave their own classes. As these classes are a part of the courses, they have a composition relation between the classes and the course class. This means if the course's class is destroyed these objects get destroyed.
    - The student class inherits the course class so that the student can see/do non-sensitive methods of the course class and its composition.
    - How to handle sensitive methods? These are declared within the teacher class and not visible to the student class. This is also clearly done on the diagram
    - Now, to handle the to-do and calendar features, we made a class called to-do and created the basic functions within it and this class is inherited by the student and teacher class.
    - Coming to the admin class, he only has one functionality and can see the attendance of a particular student.
2. In making the flowchart:
    - So the basic question is how the user controls flow flows. We need somewhere to start, so we start at the general home page.
    - How does a person of a particular role handle this? We ask the user to login and check which role they belong to and send them to their respective pages/paths.
    - The main objective is to split the functionalities into their respective paths.
    - Each case is represented by a if-else case and what are the consequences of each possibility.
    - How to handle intermediate pages? We add intermediate pages such that the user enters via an if-else. After a certain page, to return back a user simply chooses an alternative path to an if-else.
3. In making the data-flow diagram:
    - So we start with the basic question. What are the entities and databases? We listed out 3 major entities interacting with the database via processes. We also declared 6 databases.
    - Next, we split the functionality for the three entities. Now how do we optimize this? We start segregating/sorting out the common patterns in the data flow

- We start grouping processes near the database that it accesses. Next, we take inputs for the processes from the entities and group them as much as possible.
- For the output also we try to send similar outputs to different entities.

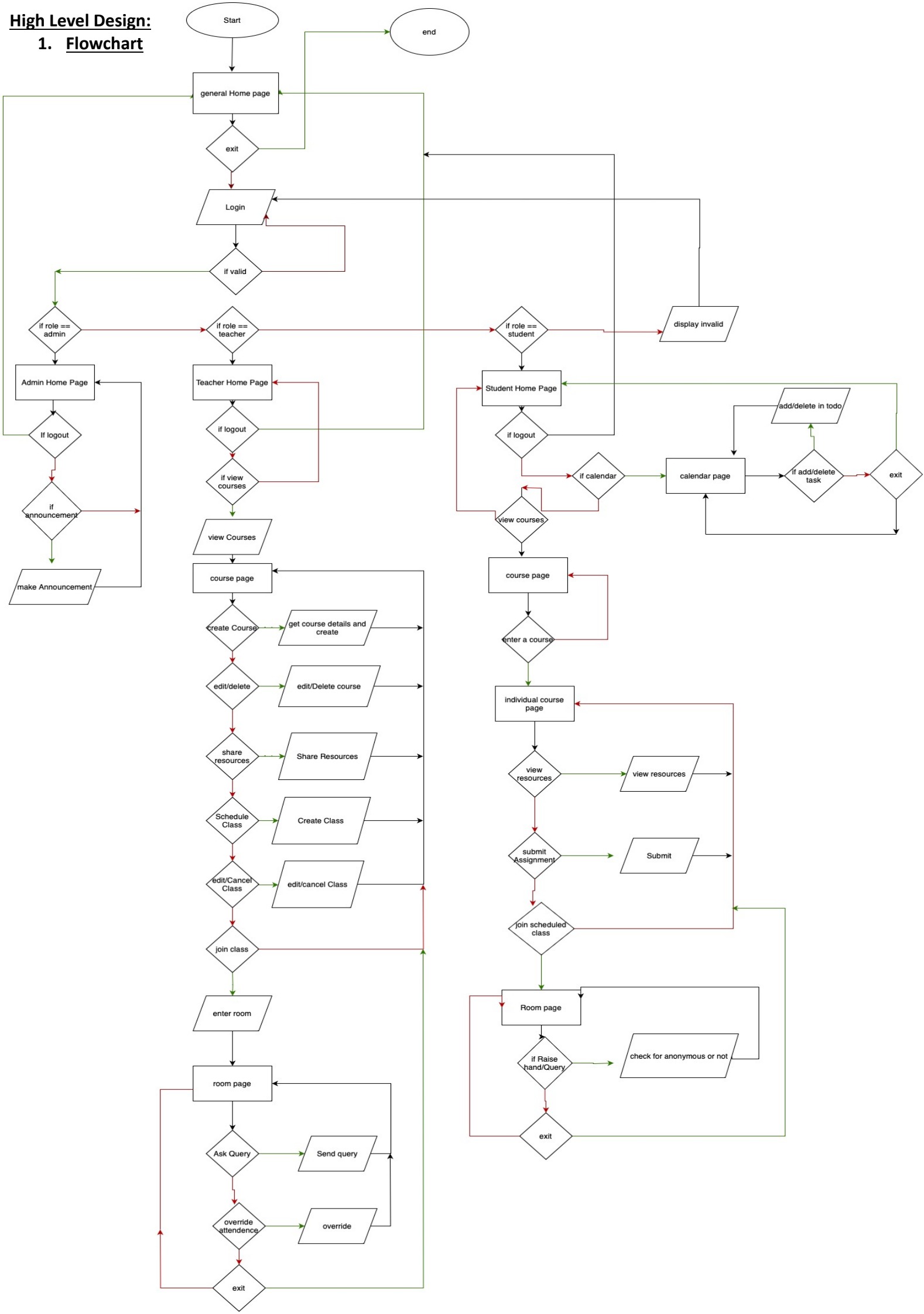
Low-Level Design:

1. User Authentication:
   - We implement a simple secure login for 3 roles.
   - We basically use Django authentication to implement this process
2. Admin:
   - As the admin has only functionality i.e., to make announcements. The algorithm for this is to take input from the announcement in the form and add it to the database.
   - To display from the database we take the data from the database and display it in the front.
3. Teacher:
   - To manage courses and course announcements, we use basic Django CRUD to implement this.
   - To create assignments, rooms and resources we take the input from forms and update the database.
   - To display the student who raised their hand is done by taking data from the backend and displaying it via the front end.
   - The queries posted by the teacher are updated to the backend and queries asked by the students are displayed.

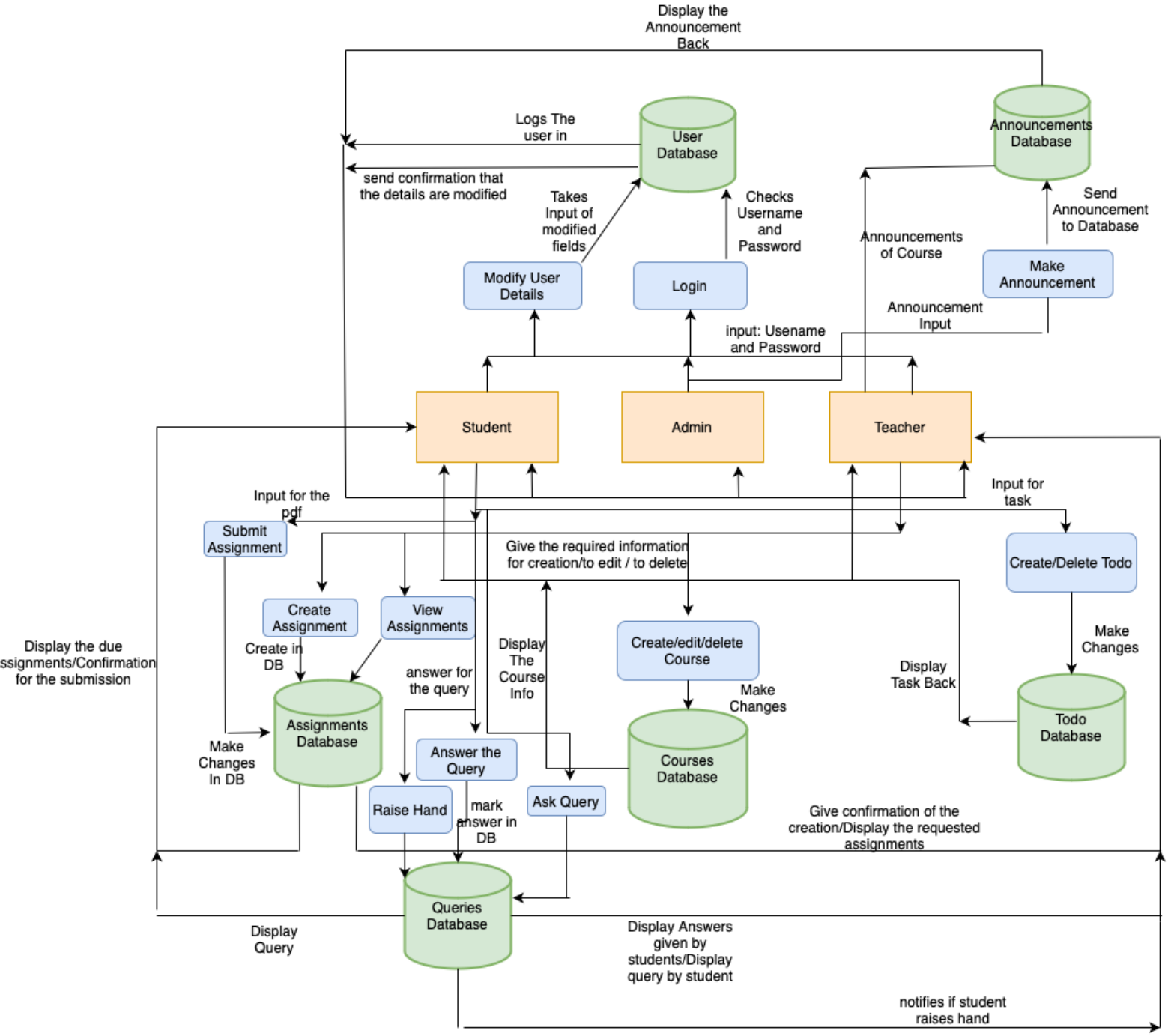
4. Courses and their composition:
   - The basic display functions of courses, assignments, classes, attendance, etc are done by simply taking data from the database and displaying it via the front end.
   - The mark attendance function works by taking the student's input in the class whether he entered the class or not. This function automatically takes the Boolean value of whether the student is present or not when he enters the class. This updates the database.
5. Student:
   - For the join class function, they are redirected to their respective classrooms. Student assignment submission is done by taking PDF input and updating the database.
   - The raise hand/ Ask query is done by taking the Boolean value of the raise hand status or taking the string input of the query and updating the respective database.
   - To see the queries by the teacher, it is displayed by taking data from the back end.
6. To-do:
   - To-do list and calendar works on basic CRUD principles of Django.
   - Add task function works by taking input for the description and the date to which it should be added and it updates the database. The same goes for delete/edit functions
   - To display the tasks it takes data from the backend and displays it via the frontend.

High Level Design:

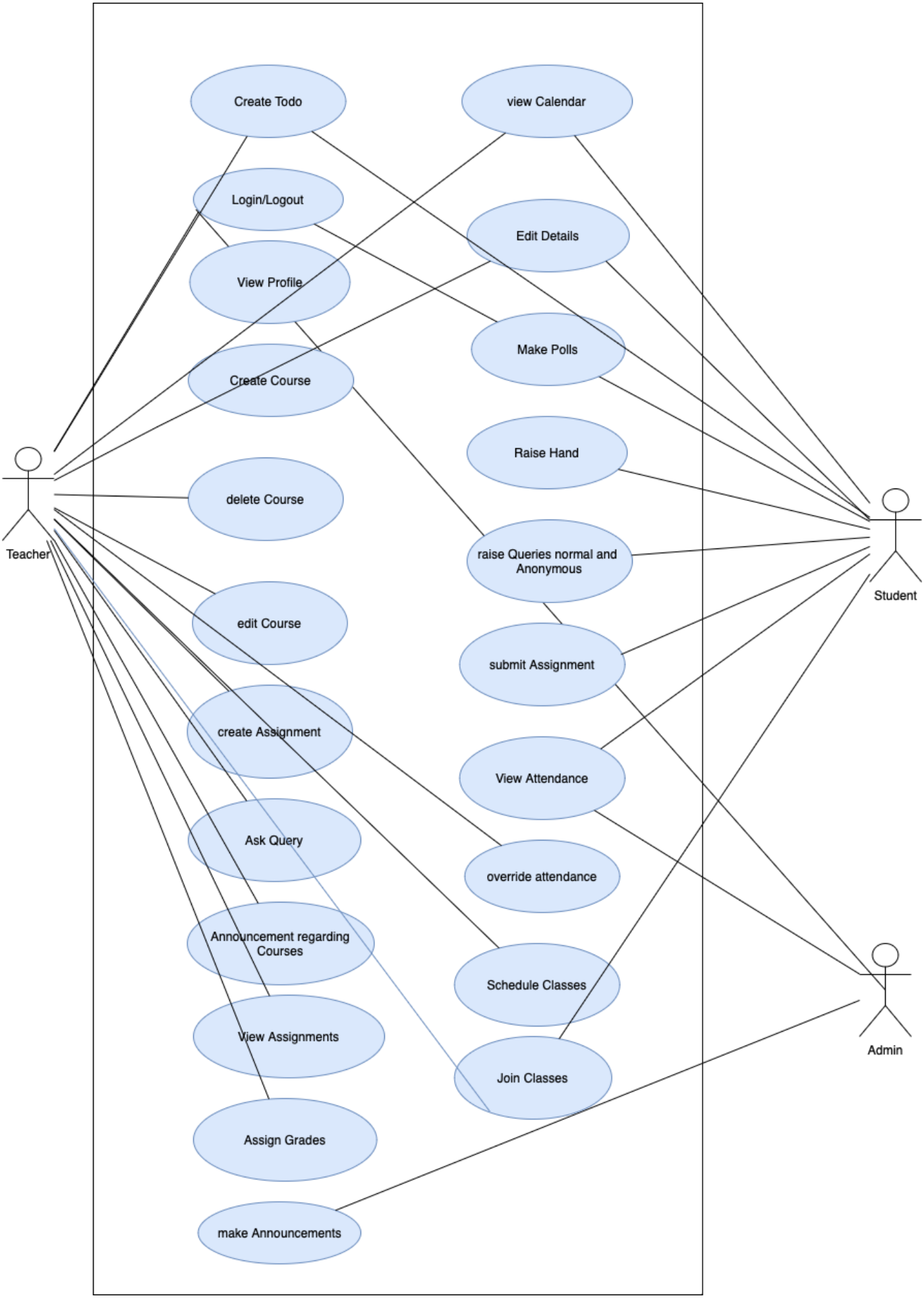
1. Flowchart



2. Data Flow Diagram:



3. Use Case Diagram:



4.Class Diagram