Team Name SAVS
Team Members

Asmita Kumar	ak4581
Shikha Asrani	sa3864
Soamya Agrawal	sa3881
Vani Jain	vj2245

Programming Language: Python
Platform: Mac
GitHub repo: https://github.com/SoamyaAgrawal17/SAVS

Part 1
Nothing has changed

Part 2

1. **What will your service do? What kind of functionality or features will it provide?**
    - Our service enables clubs to host events. Club members or the general audience can register for these events.
    - Club-head can manage club members, edit events, and set the visibility for these events.
    - New clubs can also be registered by mentioning club name, club head, description, and category.
    - Students can search for new upcoming events based on filters like category, club, date, fees.
    - Students can see for their registered events (attended events, upcoming events, canceled events).
    - Club members can propose an event, which can be approved or rejected by the club-head.

2. **Who or what will be its users? What might they use the functionality for?**
   Following are the potential users of our application:
    - **Club Heads**
        - Create an event.
        - Set the visibility of an event. An event could be either visible to club members or to the general audience.
        - Manage club members. They can add any student and remove any club member from the club.
        - Manage club events (edit event details).
        - Approve/ reject the proposed events from club members.

    - **Club members**

    - Check for the events.
    - Register for an event.
    - Withdraw from any registered event.
    - Filter out events based on date, interests (category), club, fees.
    - Propose events.

- **Students** (General audience)
    - Check for the events.
    - Register for an event.
    - Withdraw from any registered event.
    - Filter out events based on date, interests (category), club, fees.
    - Create a club.

3. **What kind of data will your service create or accumulate? What will the data be used for?**
   We will be creating the following entities, and this data would be used to serve our previously mentioned use cases. Corresponding to every entity, we have mentioned the attributes associated with it.
   Event
    - Event ID (auto-generated)
    - Name
    - Club Name
    - Visibility (Club member/ Students)
    - Start Timestamp
    - End Timestamp
    - Location (Zoom/ Google meet link/ In-person location) Here, we can use Google Map API.
    - The maximum number of people that can be accommodated.
    - Description
    - Fee (In $)
    - Status (Approved/Proposed/Rejected)
    - Number of people registered

   Student
    - Name
    - Student ID
    - Email ID
    - College
    - DOB
    - Department

Student_Club

- Student_id
- Role
- Club_id

Student_Registered_Event

- Student id
- Event_id
- Status (Attended/ Upcoming/ Cancelled)

Club

- Club Name
- Club Head
- The category that the club belongs to. (Sports, Dance, etc.)
- Description

Club_members

- Student_id
- Club_id

Part 3

1. **How will you test that your service does what it is supposed to do and provides the intended functionality?**
   - Unit tests in python for all methods like register for an event/create club/ edit event, including both happy and error cases.
   - Writing integration tests to test if the service is working as a whole in addition to unit testing of individual components
   - Write a test suite in Postman(or any other tool like SoapUI, Rest Assured) for functionality testing like students are able to register for an event, club heads are able to create an event without approval. And all other scenarios - happy and sad as mentioned in Part1.
   - Maximizing coverage and branch coverage for all tests
   - Data-driven testing, with several combinations of data being tested - will also test that we can read-write the data into our database and not just append.
   - A/B testing.

2. **How will you check that your service does not behave badly if its clients use it in unintended ways or provide invalid inputs?**
   The test suite includes cases for :

- User authentication:check every user is university-affiliated and only the club head can edit events (User Authorization)
- Avoid duplication: does not allow creating duplicate entries for clubs or events.
- Avoid irregular data like empty event names, incorrect dates, negative fees by having validation in each API.
- Checking that error responses in all cases are clear and descriptive and all errors are logged
- Adding exception handling in all methods for invalid or unexpected conditions

3. **How will you test that your service handles its data the way it's supposed to?**
    - Write unit and integration tests and try to maximize coverage.
    - Fishfooding/ Dogfooding.
    - Read/Write data as mentioned in 2.3 through unit tests, integration, and Postman testing.
    - Ensure Atomic transactions: For example, checking that connected tables are updated - when a student withdraws from an event, the number of students attending the event should be reduced accordingly. Since there are two separate calls being made to the database, we want the entire functionality to be atomic so in case one call fails, everything is rolled back and the integrity of data is maintained.
    - Data-driven tests to check all possibilities in data (unit/integration tests and Postman). We will use a separate test database or an in-memory one to see if the various GET, POST, PUT, PATCH and DELETE requests are working properly and the integrity of the database is maintained