# T2: Revised Project Proposal

**Team Name**: Aapi
**Team Members**:

| Elmira Aliyeva | Junyang Jin | Ho Sanlok Lee | Xihao Luo |
|---|---|---|---|
| ea2970 | jj3132 | hl3436 | xl3082 |

**Team Slogan**: It's not a bug, it's a feature.
**Language and Platform**: Python on Linux

**Github link:** https://github.com/junyanj1/advancedSE-Project

## Part 1:

Meeting with TA Claire: Oct 20, 2021 at 8:00pm on Discord.

## Part 2:

Our project idea is inspired by our classroom experience where the professor is often overloaded with responsibilities such as taking attendance, monitoring incoming student questions, and creating an engaging experience through in-class exercises. Although our project does not attempt to solve this problem directly, we hope that some implementation of our service could help professors with their classes.

The goal of our project is to help people organize events through streamlining the event-planning process from end to end. Our service's functionality can be separated into three stages. Before the event, organizers can create events, send invitations to potential attendees, manage RSVPs, and facilitate any pre-event communications with various groups of attendees. Attendees would be able to RSVP for the event. During the event, organizers will be able to manage attendance, ask and answer any questions through our service. Attendees will be  After the event, the organizers can get attendee information and post any follow-up communication to all attendees of the event.

The event organizers (and/or the engineers) and attendees are the users of our API. Our service could be used to support multiple clients where many events could be created. Examples where we could see our service being used include academic conferences, career fairs, or just classes in general. All of the users would want to use our service for similar purposes, such as managing attendees of their event and facilitating any communication with corresponding groups at various stages of the event-planning process. Additionally, our service could be integrated with existing platforms and APIs such as Canvas to get class, student, and faculty information and Mailchimp for any batch email communication with attendees.

Our service will accumulate information regarding the events, their corresponding organizers, and attendees. The data collected will be used to facilitate any services we provide and for record-keeping purposes for our users. Some examples of data we collect include a list of RSVPs for an event to support pre-event communications and a list of questions asked during the event for organizers to answer and engage with the attendees. In terms of object-oriented programming, we are currently planning to have an Event class that tracks any event details and communication logs, a Question class for any organizer-attendee interactions, a User class for any general user information and authentication, and an Attendance class that tracks and assigns roles to each User.

## User stories:

### US01

**Create event**: As an event organizer, I want to create/post an event with details that people with the link can see and join. My condition satisfactions are:

- Create an event with a title, description, location information, start time, end time, and max number of attendees
- Have option to choose whether the event is public or private
- Receive a link to the created event that I can share

### US02

**Send out invitation**: As an event organizer, I want to send out invitation emails to event attendees so that attendees could RSVP for the event.
My condition satisfactions are:

- Can specify potential attendees and email invitations are sent out to those I specified only.
- Each email invitation contains a unique RSVP link that is bound to each email address.
- Only those who received an email invitation could RSVP for the event by clicking the link.
- Once an attendee RSVPs, they can not RSVP again.

### US03

**Take Attendance**: As an event organizer, I want to manage attendance so that I can see who attended the event.
My condition satisfactions are:

- I can see who attended the event
- I can see who did not attend the event from the list of RSVP’d users

- Attendance will incorporate attendees who are walk-ins if the event is public (without invitation and/or without RSVP)

US04

**Follow-up Communication**: As an event organizer, I want to use follow-up communication so that attendees could receive post-event information.
My condition satisfactions are:

- Email follow-up info for those users who actually attended the event
- Optionally, I can instead send follow-up email to all RSVP'd users or all users who received an invitation.

US05

**Receive information**: As an attendee, I want to receive invitations and notifications from the organizers.
My conditions of satisfaction are:

- Receiving invitation emails in the inbox
- The URL embedded leads to event reservation
- Receiving notifications in email inbox

US06

**Read/Ask questions**: As an attendee currently attending the event, I want to read and ask questions anonymously/publicly.
My conditions of satisfaction are:

- I can ask questions under my name or anonymously
- I can ask private questions
- I can see questions that are private to me only.

## Use cases:

1. Title: Create event
   Actor: Event organizer
   Stakeholders: event organizer
   Description: Pre-event action. Event organizer creates the event after logging in.
   Preconditions: The event organizer is authenticated to create the event.
   Triggers: The organizer logs in and invokes event creation call.
   Postcondition: The event with all the information (event name, description, location, start time, duration, list of guest emails, and other additional information) provided by the organizer is created and stored in the database. After execution, the event organizer can send out invitations by email using Mailgun API. The invitations cannot be sent without executing Create event action.

2. Title: Ask question
   Actor: Attendee
   Stakeholder: Event organizer

Precondition: Attendee has been invited to the event and RSVPed
Trigger: Attendee wants to know more information
Description:

1. Attendees send their question and access restrictions to the ask question API
2. If attendees want to send private question, they should include the receiver's Id in the request as well

Postconditions: +1 question stored in the database

3. Title: Follow up communication
   Actor: Event organizer
   Preconditions: The event must be over before follow-up communication is initiated.
   Postconditions: There must be a log of the communication after the follow-up is done.
   Trigger: The organizer might want to follow up with necessary contact information or send out a survey for post-event feedback.
   Description: post-event action - organizer sends follow-up information.
   Special cases:
   a. The event organizer might want to send a follow up after the event that is also an invitation to another event (with the option to RSVP). So this is the case where the post-event communication for one event is the pre-event communication for another event.

   Exception flow: the email API (Mailgun) stops working due to hitting limits on the number of emails sent.

## Data:

- Event details:
  - EventID
  - Title
  - Organizer Email
  - Description
  - Location (name, long/lat, address)
  - Time
  - Duration
  - Limit on number of attendees
  - hasStarted - whether an event has started
  - hasEnded - whether an event has ended
- Attendance:
  - Email
  - EventID
  - IsRSVPed
  - IsAttended
- User details:
  - Email
  - Name
  - Type (attendee/organizer)

- Questions:
    - QuestionID
    - EventID
    - Type (anonymous/private/public)
    - Email
    - Title
    - Description
    - TimePosted
- Response
    - QuestionID
    - Email
    - Response
    - TimePosted
- CommunicationLog
    - MessageID
    - MessageDetails
    - Time sent
    - Sender
    - Receivers

## Part 3

We plan to have four testing stages - unit test, integration test, system test, and load test. We will use PyUnit for both unit testing and integration testing. During unit testing, we will test individual methods and classes to make sure each building block of our service is working as expected. During integration testing, we will test to see how these building blocks behave when they are put together. The integration tests will involve the database layer.

After passing all unit tests and integration tests, we will set up a test server on Google Cloud Platform and perform system testing. We will use Postman and Requests to simulate the actual REST API requests that will be made by the target users of our service. Lastly, we will perform load/stress testing with Locust to ensure multiple users can access our service at the same time.

We will use tools like codecov to keep track of test coverage and complexity. We will also implement a CI/CD pipeline that can automate most of our testing process.

In every testing tage, we will include negative test cases to verify that our service can gracefully handle incorrect or malicious inputs, and that our service does not fall into an erroneous state. Our test cases will include input sanitization, cross-site scripting, and user authenticity checks. When testing the event creation feature, for example, we can test whether a valid event time is provided during the unit testing stage, and whether the organizer/User exists during the integration testing stage.

During the integration testing stage, we will have test cases that validate the integration between our server layer and the database layer. The test cases in the integration testing stage are responsible for ensuring all server layer actions are correctly persisted in the database layer.

During the system testing stage, we can monitor the database layer directly and verify that all data gets stored in the correct format and in the appropriate tables as intended when we make requests using the Postman or Requests library.

US01 Common case.

a. The tester creates an organizer account and tries to log in. It verifies that the user is logged in.
b. The tester creates the event and verifies that the API correctly stores all provided data.
c. Tester tries to send out email invitations to the list of guests and verifies that emails are sent to all guests.
d. Tester tries to generate the link for the event and verifies that the link is valid and accessible.

US01 Special case.

a. The tester tries to create an event without logging in. It verifies that the event is not being created.
b. The tester tries to create an event with invalid input data (non-existing location, invalid start/end time, negative number of max attendees) and checks that this event is not being allowed to be created.

US02 Common case

a. Authenticated tester
b. The tester invokes invitation API with list of emails
c. The tester confirms that the API returned 200
d. Receive email within a few minutes

US02 Special case

a. The tester invokes the invitation API with invalid email addresses and receives an error message that the email address is not valid.
b. The tester invokes the invitation API with email addresses that already received an invitation to and receive error messages that the invitation was already sent
c. The tester invokes invitation API with invalid event (event does not exists, event already started/over) and receives error message that the event is not valid

## Part 4

- For facilities, we plan to use:
  - VS Code for our IDE
  - Flask for backend framework
  - PostgreSQL for our database
- For APIs, we plan to use:
  - OAuth2 for user authentication
  - Mailgun to send out emails for any communication needs
  - Google Maps API for location of event
  - Google Calendar API for allowing users to sync event to their calendar
- For testing, we plan to use:
  - PyUnit
- For style checking, we plan to use:
  - Flake8
- For static analysis bug finding, we plan to use:
  - Pylint
- For coverage analysis, we plan to use:
  - Coverage
- API platform
  - Postman