# Preliminary Project Proposal

## Part 1

Note: none of the information has changed.

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

- Elmira Aliyeva (UNI: ea2970)
- Junyang Jin (UNI: jj3132)
- Ho Sanlok Lee (UNI: hl3436)
- Xihao Luo (UNI: 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 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, users can create events, send invitations to potential attendees, manage RSVPs, and facilitate any pre-event communications with various groups. During the event, organizers will be able to manage attendance, ask and answer any questions through our service. 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) 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.

## 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.