Mandatory Extension:

- There will now be many types of events. A one-speaker event is the same as a "talk" from Phase 1. You can have multi-speaker events, like a panel discussion, and no-speaker events, like a party. Events can last as long as you want. You can measure the duration of an event in hours, or minutes. You get to decide.
- Events can be canceled by at least one organizer.
- At least one more type of user will be included in your program. For example, an Admin user who can delete messages or events with no attendees, or a VIP user who can access VIP-only events.
- Organizers can also create Speaker accounts, Attendee accounts, and any other type of user accounts that are in your program.
- Each event has a maximum number of people who can attend it. This amount can be set when the event is created and also changed later, by an organizer. Your program should check the maximum capacity for the room where the event will be held and prevent the number of attendees from going above the room's capacity.

Optional Extension:

- Allow the same users to log in and select which conference they want to participate in. Here, participation means viewing and signing up for events. The inbox we did was event specific.
- Enhance the user's messaging experience by allowing them to "mark as unread", delete, or archive messages after reading them.
- Add additional constraints to the scheduling for various types of events (e.g. technology requirements for presentations, tables vs rows of chairs). When organizers are creating events, they can see a suggested list of rooms that fit the requirements of their event.
- Replace your text UI with a Graphic User Interface (GUI), which should follow the Model-View-Presenter architecture. See the note below.

Additional Features:

- VIP user now has a point system. If they join a VIP event, they get 50 points else they get 10 points. They are a Platinum user if they have 1000 points, Gold 500, Silver 100 and Bronze 0. They can message organizers.
- Calendar to show what the user is doing on particular day.

Design Patterns:

- Dependency Injection:

  For the controller classes, we decided to eliminate the constructor call of user case classes by introducing it as a parameter in the constructor and/or a method.

How our code has improved since Phase 1:

- We stored strings to identify objects of classes instead of storing the objects itself.
- We fixed the code to implement the Single Responsibility Principle from the SOLID principle, by making user abstract and also separating some classes.
- We also made it so it follows Open Closed Principle, so we could add functionality with inheritance without affecting other classes. Specifically, Event and User class that follows this principle.
- We also ensured we followed Liskov principle since the child classes have more specifications than the parent class.
- We clearly identified all the layers of clean architecture and ensured they didn't overlap.