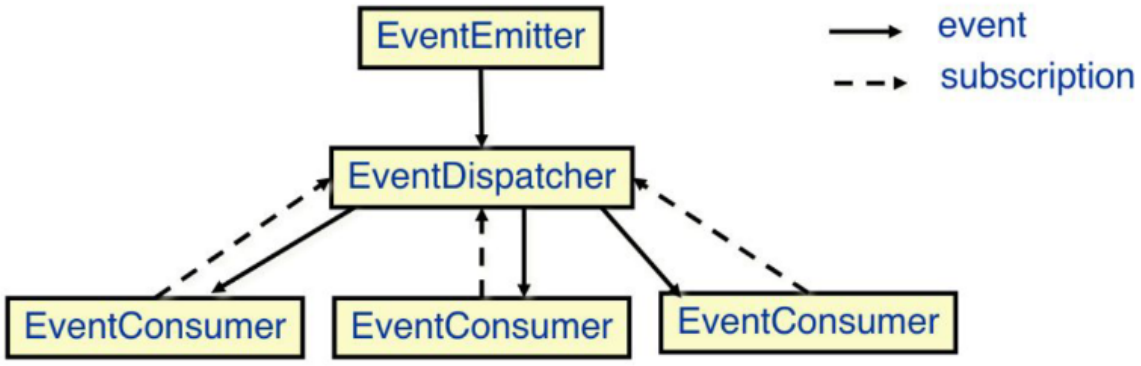
# Project Milestone 3

## High-Level Design

The architectural pattern that we would use to structure our system is event-based architecture. From class lectures, we know that event-based architecture “promotes the production, detection, consumption of, and reaction to events.” This perfectly describes our product (CodeCollab). Here is a brief review of CodeCollab: CodeCollab is a platform which allows software engineers within an organization to request assistance for issues relating to specific technologies. There is a database containing information about each software engineer and what technologies they are comfortable in assisting others with. When a software engineer requests for help through CodeCollab, our platform pairs them with a software engineer who has expertise in that technology. Finally, a meeting is automatically created between the two software engineers based on time availability inputted by them.

As shown in the logical flow of our product, each step is based on the result of an event. For example, the first event would be a software engineer requesting help for an issue. The next event would be to locate another software engineer within the organization who has expertise with that issue (this depends on the result of the previous event). The following diagram from class can also be used to support the event-based architecture for CodeCollab:

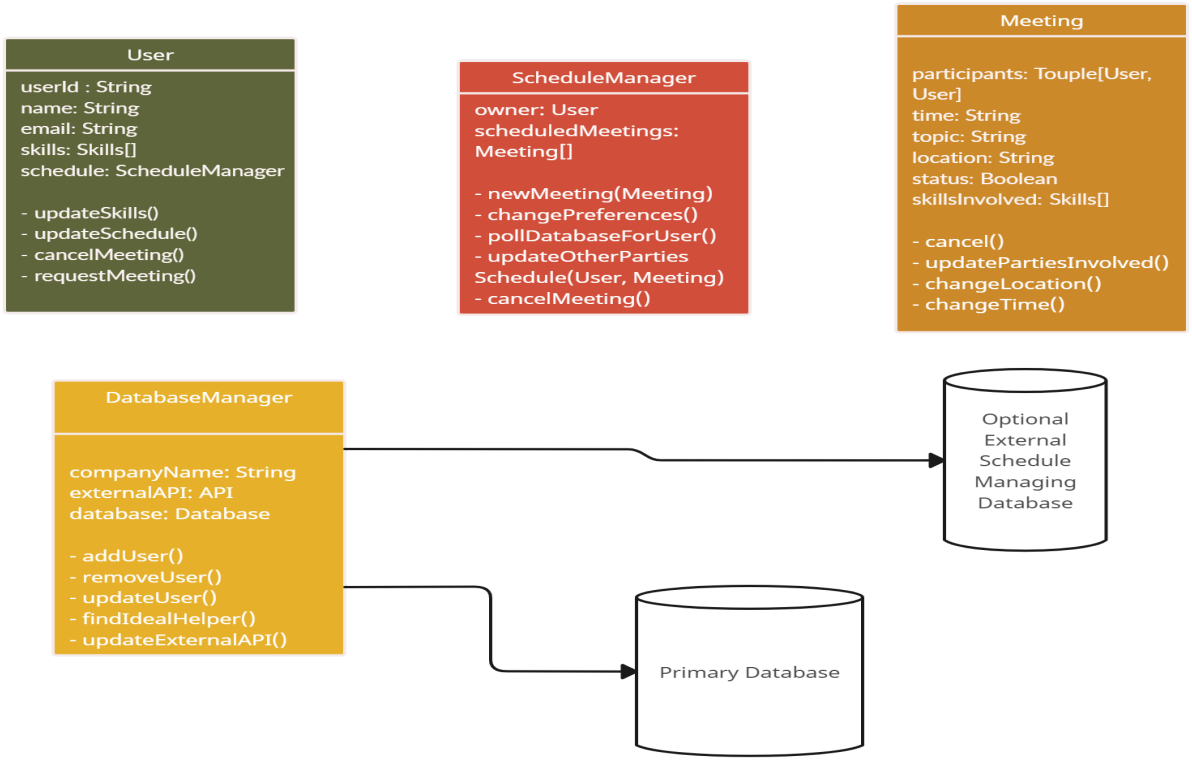



In the example I described above, the EventEmitter would be the creation of a request once a software engineer submits it. The EventDispatcher would be the search for another software engineer who can assist with the request. The EventConsumer would be the software engineer who is paired up to assist with the issue, and a meeting would be scheduled as a result.

In conclusion, because the logical flow of CodeCollab can be divided into different events (where each event depends on the result of the previous event), an event-based architecture would be the best pattern to structure CodeCollab.

## Low-Level Design

The design pattern family that would be most useful for how we want to structure our project is the behavioural design pattern. Our envisioned implementation of CodeCollab requires several distinct classes serving some essential function for the project, however each classes functionality depends on one or more of the others. The specific design pattern that best suits our design is the “Chain of Responsibiltiy” pattern. As mentioned in class, the Chain of Responsibilty design pattern is “a way of passing passing a request between a chain of objects”. This low-level design pattern makes the most sense considering our ideal High-Level design pattern for CodeCollab is Event-Based, which is essentially a sequence of actions being triggered by some action. The relationship between objects can be seen in the class diagram below. At the




low-level, the sequence of actions of a meeting request by a software engineer would look roughly like this. The software engineer would fill out a meeting request, which would prompt a search for another available software engineer who is capable of helping, which would require both User’s corresponding schedule objects to be updated with new Meeting objects, which would require, if used by the company, the external scheduling application’s database to be updated. This is one of the many possible requests that can be made by users, but all of them involve similar flows of actions.

Here is a psuedocode representation in a Java-like language of how a cancelled meeting would be handled.

```
Class User {
        ScheduleManager manager;
        ...

        User(ID, Name, Email, Skills) {
                manager = new ScheduleManager();
                ...
        }

        ...

        cancelMeeting(Meeting, String reason) {
                manager.cancelMeeting(Meeting, reason);
        }
}

Class ScheduleManager {

        List<Meeting> meetings;
        DatabaseManager dbM;
        ...

        ScheduleManager(DatabaseManager manage) {
                meetings = new List<Meetings>();
                dmb = manage;
        }
        ...

        cancelMeeting(Meeting, String reason) {
                User other = Meeting.getOtherParty();
                other.getManager().getMeetings().remove(Meeting);
                meetings.remove(Meeting);
                database.updateExtenralAPI(other, Meeting)

        }

}

Class DatabaseManager {

        updateExternalAPI(User, Meeting) {
                if (externalAPI != null) {
                        externalAPI.removeEvent(Meeting);
                }
        }

}
```

## Design Sketch

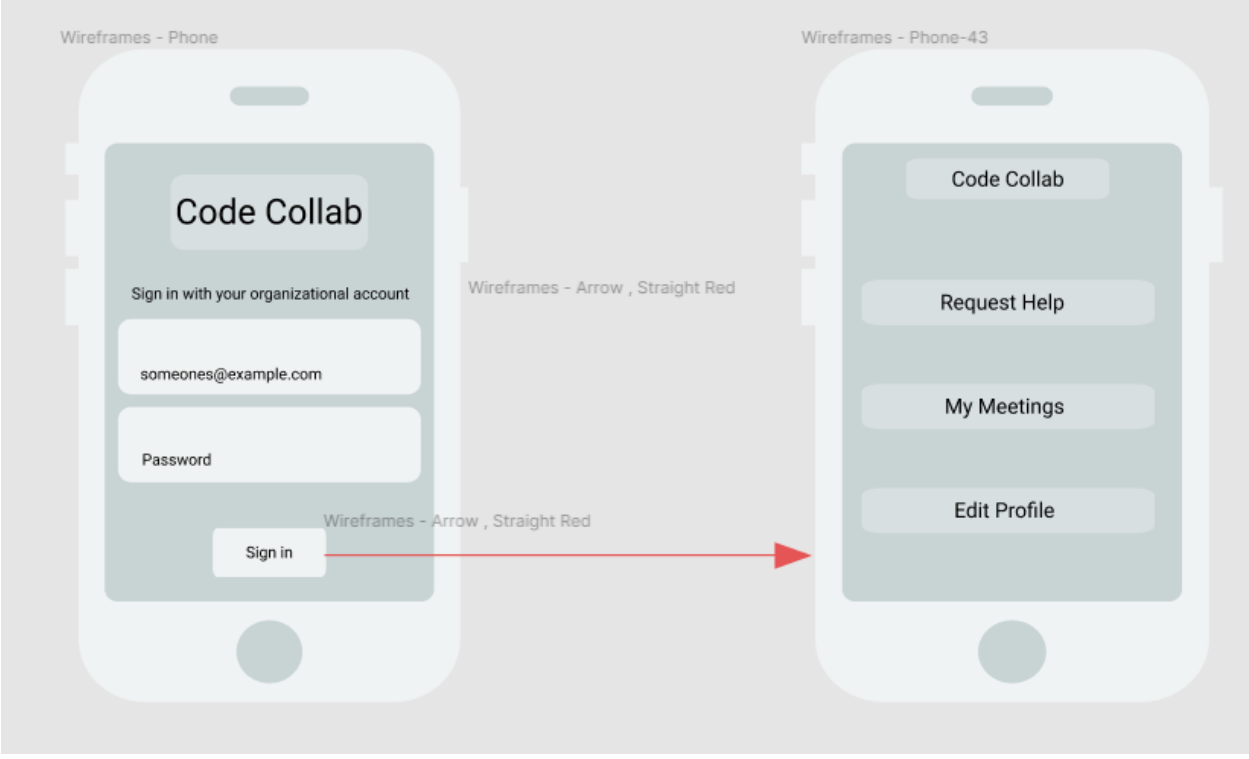


Figure 1.

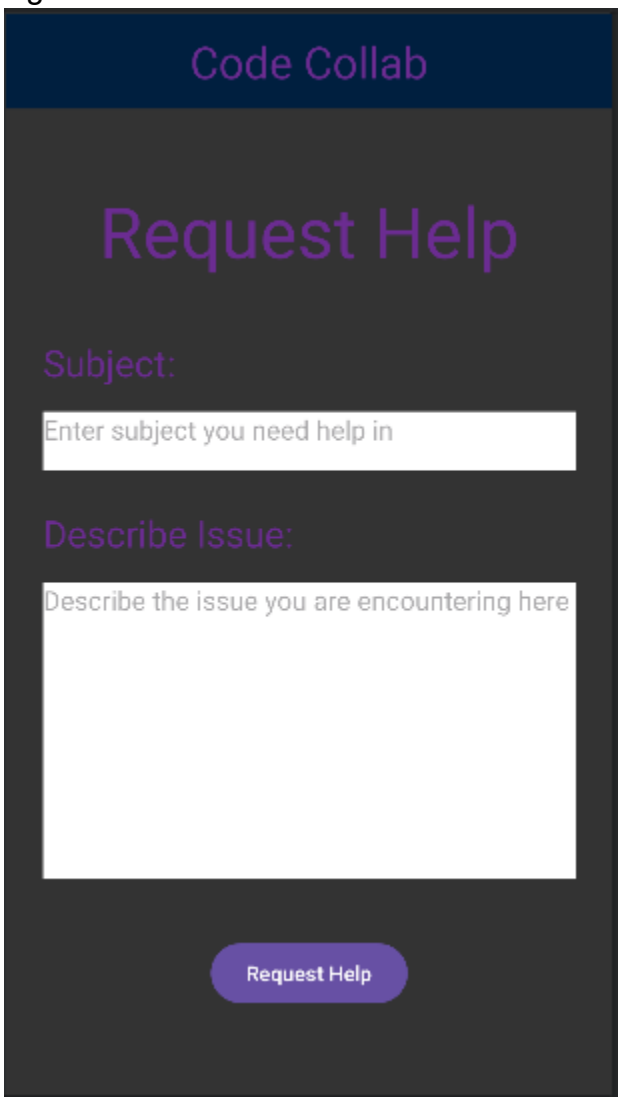



Figure 2.


Figure 3.

Our design sketch is a wireframe mockup of our app's interface with 4 main pages. First, we designed a sign-in page similar to Microsoft's organizational sign-in. It's a simple login page that needs the user's organizational email. Signing in leads to the homepage with three buttons. The simple design is to not overwhelm the user with choices. Request help on the homepage leads to the app's main functionality in Figure 2. The user submits a ticket describing the issue and the subject they require help with. This design is a very simple ticket interface that the user has encountered in their work before. When the user submits the ticket using Request Help, this will lead to the mark availability screen. It showcases a calendar so the user can mark a good time to meet. The user will likely want to meet soon, so one month is shown.

**Project Check-In**

Rushi has completed the project check-in survey form and submitted it.

**Process Deliverable**

Our team is using the agile SE process model. Specifically, we are using scrum. We will provide notes for each team member from our most recent scrum meeting. These notes were written while working on this assignment so this work is included in the notes.

Rushi Bhut

Work completed so far:
- Completed the “Related Work” section in project milestone 1.
- Created the “Relation to Overall Project Goal” slide for our lightning talk which discussed how CodeCollab meets the project goals.
- Presented the lightning talk.
- Completed project milestone 2 (specifically, I worked on use case 1 and use case 2).

Work in-progress:
- High-level design portion of project milestone 3.

Work being blocked:
- Cannot start on the final presentation slides and final report until project milestone 3 is completed.

John Nguyen

Work completed so far:
- Completed “Abstract” section of proposal in PM1
- Completed “Use Case” section in Lightning Talk slides in PM1.
- Presented lightning talk.
- Wrote one fully-dressed use case for CodeCollab.
- Completed wireframing and writing for Design Sketch

Work in-progress:
- Researching architecture for application
- Setting up test plan

Work being blocked:
- No blockers

Max Lane

Work completed so far:

- Completed the “Introduction” section for our proposal in PM1
- Presented the slide that explained the fundamental ideas of CodeCollab for lighting talk
- Wrote one fully-dressed use case for CodeCollab for PM2

Work in-progress:
- Working on completing “Low-Level Design” section for PM3

Work being blocked:
- I’m not sure which design pattern family would be ideal for this project

Patrick Walsh

Work completed so far:
- Completed “Software Engineering Process” in PM1
- Presented in Lightning talk
- Completed Use Case for CodeCollab
- Designed some pages of UI and made wireframes for them

Work in-progress:
- Continuing design of UI for CodeCollab

Work being blocked:
- No work currently being blocked