Software Design Specification

Match on the Street

Lance Ogoshi (logoshi)
Hope Crandall (hopesc)
Hao Liu (liuh25)
Ruijia Wang (ruijiw)
Jesus E. Larios Murillo (larioj)
Ben Tebbs (bentebbs)

System Architecture

Modules

- **`MapActivity`**
  - Displays location information and event details.
- **`ListActivity`**
  - Displays events in a list format.
- **`AddEventActivity`**
  - Prompts for user inputs and registers user events.
- **`LoginActivity`**
  - Displays the login page with the Facebook log-in option.
- **`SidebarActivity`**
  - Contains extra options not accessible in the main view. Options include accessing the profile.
- **`EventDetailsFragment`**
  - Shows additional information about an event when the event is highlighted. Appears when an Event is selected on the map view.
- **`UserProfileActivity`**
  - Displays information about the current user. Information includes name of the person logged in and a list of Events they are planning on attending.

Interfaces

- **`DBManager`**
  - Provides the methods needed to contact the backend.
  - **`EventDBManager`**
    - Global interface to all event information. It can be used to retrieve sets of events matching particular parameters.
  - **`AccountDBmanager`**

        - Global interface to all user information. It can be used to retrieve the information for the current user, and later on will serve to retrieve information for arbitrary users.
- **`Account`**
    - Represents a user's account. Includes their access token (for the Facebook API), which is a unique identifier. Also includes a name, and a list of events they are signed up for attending.
- **`Event`**
    - Represents an event. Provides access to title, time, location, and a list of users attending.
- **`ViewController`**
    - Provides the view components with information of events and methods to update/search for events.
    - Information stored includes: 1. The set of events to be displayed across the map/list views; 2. The set of filters currently applied to the search results; 3. User location.

Data

The primary type of data in our system is event information. The data will be stored using a Database on Cubist. The format of the event and user data will be in three tables, specified as:

`Events`:

`id`: integer making each event entry unique
`title`: string indicating the name of the event
`time`: formatted time showing when the event is
`location`: coordinates representing the location of the event
`description`: description of the event. String.

`Accounts`:

`access_token`: token representing a unique entry in the Facebook API
`name`: name of the user with the access token

`Attending`:

`event_id`: the event being attended by the user with user_id
`user_id`: the access token of the user attending the event with event_id

Later we may expand the account information to include more fields, such as previous events attended, user rating, and skills. We will be using Facebook for identification, so we won't have to worry about specific user identification information such as emails and passwords.

Alternatives

Database Backend

We decided to use a database hosted on Cubist as our backend because it's simple and familiar. However, we could have used a different infrastructure. Primarily, we considered using App Engine as our backend. We decided not to pursue this route because it would have increased the complexity of our backend. One drawback to the database, is that it lacks flexibility. For example, it can't perform complex backend computations. At this point in our application's life, we don't need a lot of backend services, and since we will be abstracting the backend away in our applications, we should be able to change it later if we need to.

We also considered using Firebase, a noSQL datastore. It was alluring because it seems well supported, and provides a json interface. However we decided not to use it, because we were not very familiar with it and we did not want to risk spending too much learning new tools as we already have to learn to use two major APIs as well as Android.

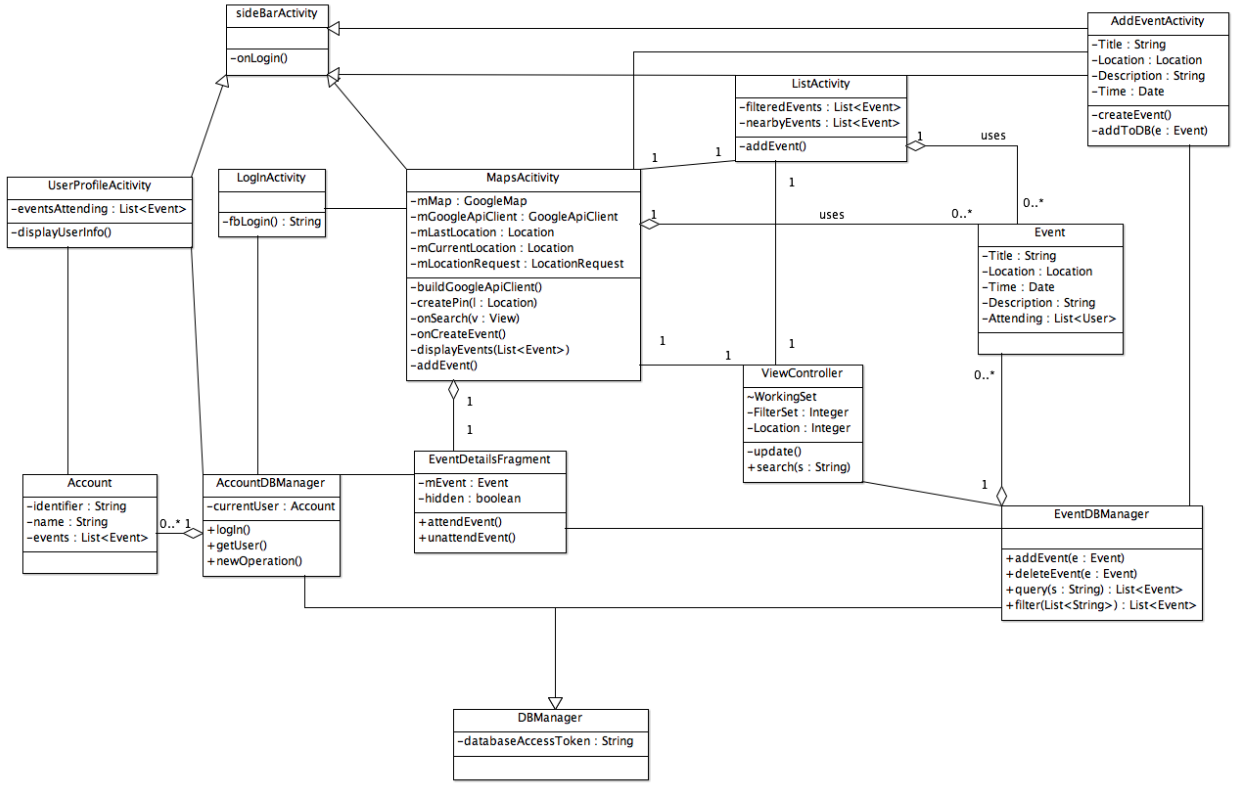
Account structure

We eventually settled on having a separate object to represent accounts. However, we originally thought we might not need a dedicated object. All the necessary data about the user is stored by the Facebook API, so all we would need to do is pass the access token around the app so that each activity could extract the necessary information. We could also use this access token to query the database. The advantage of this method is having fewer objects to work with, which means less code initially. Also, in our initial specification we really only need one account (the account of the currently logged in user), so having a whole class to represent accounts might seem gratuitous. We decided to include the class despite these issues for two reasons. 1.) It allows us to work with something other than a nasty access token that comes from a foreign source and could be difficult to deal with. We would have to read extensive documentation to understand all the methods this access token can use, and there would be many which weren't necessary. 2.) This way, if we want to, say, allow users to "follow" other users in a later development of the app we would be able to do this simply by wrapping other users' database information inside the Account class.

Assumptions

Since we are are relying on the Facebook API for user identification, we assume that: 1) the user is connected to the internet and 2) the user has a Facebook account. This may not always prove to be true. Individuals that do not have a Facebook account will not be able to use our app. We also assume that our application will only need to get data from the backend.

Class Diagram



MVC View

Model:

- The Cubist database (not pictured)
- `Account`
- `Event`

View:

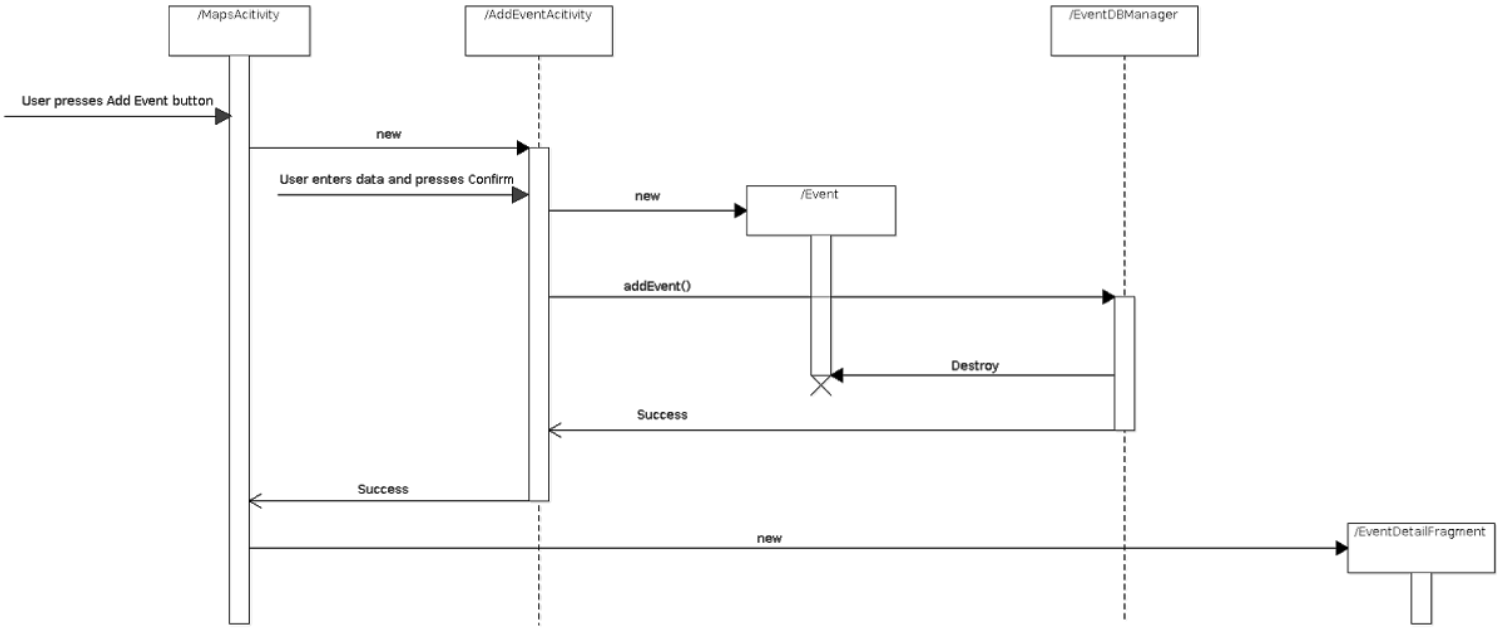
- `SideBarActivity`
- `UserProfileActivity`
- `EventDetailsFragment`
- `MapsActivity`
- `ListActivity`
- `AddEventActivity`
- `LoginActivity`

Controller:

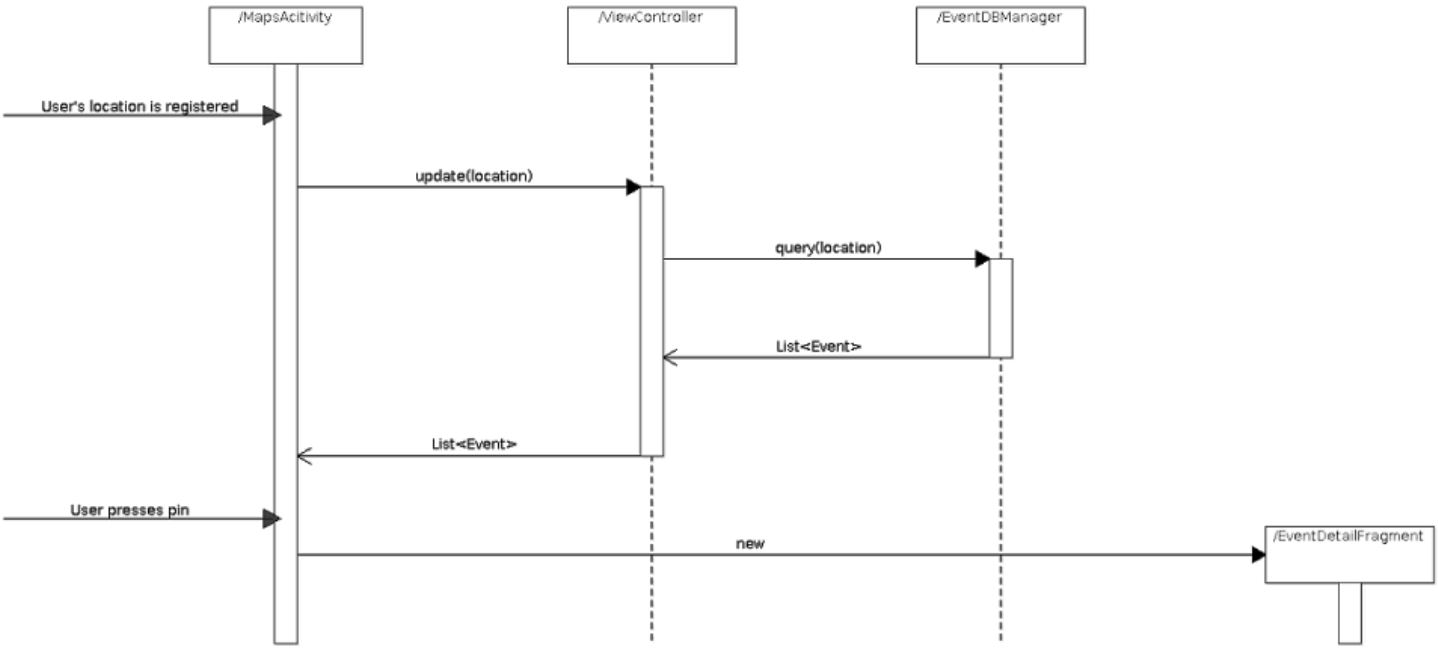
- `AccountDBManager`
- `EventDBManager`
- `DBManager`
- `ViewController`

Sequence Diagrams

Adding Event



Finding Event



Process

Risk Assessment

Since we completed the SRS, we have gained a better understanding of the APIs and tools that we are using. We have also implemented the Google Maps and Facebook API aspects of our app. With regards to the backend, we have decided to use a database on Cubist due to its simplicity but we have yet to integrate it with our app. The risks with the Google Maps and Facebook APIs are low risk since the respective teams understand the API, but the risks with the backend integration is still medium. Initially we thought that feature creep would be a large issue since it is easy to digress and start discussing extra features during our design meetings. However we've been able to steer the discussion back to our major features so although the chances that feature creep happens is still high as we are finalizing the design, the actual impact is low since we only waste a few minutes of discussion.

Falling Behind / Underestimating Time Requirements (high chance / high impact)

- It's easy with any "long" term project to get distracted by other courses and assignments resulting in missed deadlines. Even with a well defined schedule, there will be bugs of varying severities which may take additional time to fix. Since we know there will be bugs we will always try to work ahead of our set schedule to leave enough time for bug squashing. During our weekly meetings we check in with each member on their progress for each of the deadlines which will make detection easy. If/When we start falling behind we might be able to move people around to help the parts that are behind. Additionally we can have additional meetings during the week that are just for coding to eliminate procrastination. If necessary we could have a 'hackathon' for a day over the weekend and commit 24 hours solely to development.

Learning APIs and tools (medium chance / high impact)

- For the Facebook and Google Maps API's there hasn't been much difficulty integrating them into our app, but we have run into issues with git in Android Studio. Since our goal is to have regularly scheduled builds, everyone needs to be able to use git without any difficulty so we will all need to learn and use git on a regular basis, which will reduce the likelihood of these issues. Problems with these tools and API's will be reflected by schedule delays in the regularly scheduled builds. If/When problems occur we may decide to cut the use of the specific API and use a different method or we will divert more people to help learn and understand the specific API or tool.

Creating a robust backend (medium chance / medium impact)

- Our initial design for the backend focuses on simplicity. It's essentially just a database that provides our app with event and user data. At this point in the implementation it appears that a remote database is all we need. However, there is risk that we may need same unforeseen services from the backend later. Because of this, we have made our system design provide a local interface to the backend, which we can change later without having to change the rest of the application. If problems arise, we can also cut the expansion to other types of events and solely focus on pickup sports. Problems with the backend will be identified quickly since it is used by every other part of the app.

Feature creep (high chance, low impact)

- In almost every meeting the discussion has turned to discuss more features and stretch goals based on other similar designs. This has wasted a fair bit of time during our meetings, but we always manage to steer the discussion back to our major features. As we come up with more concrete design details the likelihood will decrease since there shouldn't be much discussion on features we've decided. In order to detect feature creep, we just need to remember the major features and focus our plans on completing them. If/When we encounter feature creep we will refer to our major features and if we have a feature that doesn't contribute to the major features, it will be cut.

Integration of various modules (medium chance, high impact)

- Since android development is new for everyone on our team piecing all of our modules together will probably be difficult since we've already run into issues with using git in android studio. The impact is high because if our modules can't operate together then we've failed the basic requirements of our app. To reduce the likelihood we will make sure that everyone has a solid understanding of how the modules of our app will interact with one another and how that process is implemented. In order to detect problems as soon as possible we will have daily builds to ensure that everything is working smoothly. If/When we have problems then we will revisit our design specification and ensure the implementation is correct and make any changes to the design spec. if needed.

Project Schedule

Week	All	Event Management	Location Services	Profiles and Authentication	Search and Sidebar
Jan 22 - SRS	UI Design				
Jan 26 Weekly Report 2	Learn Android SDK	Learning Cubist Database Schema	Learning Google Maps API	Learning Facebook API	
Feb 1 - SDS Feb 2 - Weekly Report 3	Learning of Tools	Backend Design	Implement EventDetailFragment	Designing visual layout of login page and account page	Setting up XML layout scheme and initial ListActivity draft
Feb 8 - Zero Feature Release Feb 9 - Weekly Report 4	Usability testing	Implement AddEventActivity Preliminary Implementation of Backend Implement DBManager Class	Clean up interface. Start work on ViewController	Login page implemented	Further Implementation on ListActivity and SidebarActivity. Preliminary integration with Map view and backend.
Feb 16 - Weekly Report 5 Feb 19 - Beta Release		Implement EventDBManager Implement UserDBManager	Finish ViewController	Login integrated with app	Finish ListAcitivity, including its filter and search functionalities

		Implement Add Event			Sidebar usable from both map and list views
Feb 23 - Weekly Report 6 Feb 26 - Feature Complete Release		Extend User and Event Information Backend Integration Testing	Complete integration with all other views	Account page implemented	Complete integration of ListActivity with database Further improvements/debugging
Mar 1 - Weekly Report 7 Mar 4 - Release Candidate	Extensive debugging, final testing	Debugging	Bug cleaning, Performance enhancements		
Mar 8 - Final Release	Party		Sleep		

Team Structure

Jesus E. Larios Murillo (larioj) will be the team manager.
Other members are divided into the following teams.

- **Event Management:** (larioj, ruijiw)
  - Members
    - Jesus E. Larios Murillo (larioj)
    - Ruijia Wang (ruijiw)
  - Responsibilities
    - Implementation of the backend.
    - Implementation of application's interface to backend.
  - Milestones
    - Database Schema (Done)
    - Get Database Running
    - Implement AddEventActivity
    - Implement DBManager
    - Implement EventDBManager
    - Implement UserDBManager
- **Location Services:**
  - Members
    - Lance Ogoshi (logoshi)
  - Responsibilities
    - Implementation of Map View UI
    - Implementation of interface to Maps API
  - Milestones
    - Learn Google Maps API (Done)
    - Implementation (Done)
- **Basic Profiles and Authentication:**
  - Members
    - Ben Tebbs (bentebbs)
  - Responsibilities
    - Implementation of Login UI
    - Implementation of interface to Facebook API
    - Implementation of profile page where currently joined events are listed
  - Milestones
    - Familiarize with Facebook API (done)
    - Designing login and account page
    - Build login page
    - Integrate login with the rest of the app
    - Build profile page
- **Search and Sidebar**
  - Members

        - Hope Crandall (hopesc)
        - Hao Liu (liuh25)
    - Responsibilities
        - Implementing search functionality
        - Implementing filtering of results
        - Implementing search UI
    - Milestones
        - Implement Search UI
        - Implement Sidebar
        - Implement ListActivity
        - Implement ViewController Filter Methods

Date and time of weekly meeting:

- Tuesdays 9:30am at CSE 002
- Thursdays 9:30am at MOR 220 and CSE 021
- (Occasional, notification in advance) Sundays 1:00pm at CSE 002

Weekly status reports:
https://drive.google.com/drive/u/0/folders/0B12V2Dmj3JoyNGFHQjg3bGtmdGs

We communicate by a Facebook group (CSE 403) and through Google Drive. The Facebook group is helpful for announcements as it will send notifications to all the members, allowing everyone to stay on top of things. In addition, the chat service on Facebook allows a much more immediate way to speak to the group, or to talk to a specific person informally. Google Drive allows simultaneously collaboration for documents such as this one. This allows these documents to be created quickly and seen by everyone. In addition, it is a centralized hub for things such as UI and UML diagrams, so all members can stay on the same page with respect to the design of the application.

Test Plan

Unit Test Strategy

- Tests
  - Team specific tests
  - Test compatibility with older Android versions
- Methods
  - Using Java's Junit tests for logic
  - Using Espresso to test UI elements
- Each Team is responsible for developing the tests for their UI elements
- Should be run on a daily basis for each team

System Test Strategy

- Using Espresso
- Tests
  - Transitions between views
  - User inputs
  - Connection to databases
  - Queries to databases
- Can be automated with Espresso and added to the git repository

Usability Test Strategy

- Tests
  - UX
  - Ensure that design choices are logical and consistent with android apps
- Methods
  - Paper Prototype testing: Using a paper prototype with individuals around campus to test the ease of use of the application. Good for finding what is hard for users to figure out from a UI standpoint.
  - Digital Prototype testing: Deploy to android phone and ask individuals in our target audience to try to use our prototype app and observe their behaviors. Good for discovering bugs.
- Frequency

  Before every major release of the project: Zero-feature, Alpha, Beta, Full-feature, Final and incoming major updates.

Adequacy of Test Strategy

We aim to heavily test the aspects of our app that are completely under our control. For example, we aim to have extensive unit tests to verify that the Event and Account classes work. In addition, we want to ensure that the UI does not allow certain interactions such as sql injection. However, we will not test as extensively our application's interactions with various APIs such as Facebook and Google Maps. We will for the most part assume that the user's GPS is working, as well as their internet connection. We aim to test the local aspects of our app frequently to ensure that no bugs emerge unnoticed.

Bug Tracking Mechanism and Plan of Use

- Google spreadsheet with the bug, timestamp, severity (low, med, high), who is assigned to fix the bug, and the description of the bug.
- Each team is responsible to report the bugs they discover and add unit tests to prevent the bug from re-emerging.

Documentation Plan

- JavaDocs for “man page”
- User guides:
  Includes step-by-step instruction on using major functionalities of this application including but not limited to:
  1. Logging into the application
  2. Viewing/attending events
  3. Searching for events by location
  4. Viewing events in a list format
  5. Searching for events by name/tag
  6. Applying filters on search results
  7. Managing user profile

  The user guides will be provided on the application website, with a link to them within the application.
- Developer guides:
  1. Obtaining the source code
  2. Structures of the source code
  3. Making a build of the project
  4. Running test suites on the project
  5. Known and fixed bugs

  The developer guides will be provided on the application website.

Coding Style Guidelines

Members of the project should read the following guidelines on coding styles:

- Code Style for Contributors on Android Source website: https://source.android.com/source/code-style.html

Members are expected to write and review their code conforming to the above guidelines before submitting it. When a peer review is held during each week's team meeting, the above guidelines will be applied to assess the code quality and areas of improvements.

The following guidelines are recommended as additional references, but not required:

- Google Java Guide: https://google.github.io/styleguide/javaguide.html
- Google XML Document Format Style Guideline: https://google.github.io/styleguide/xmlstyle.html