SwipeInvite

Design Document

Team 16

Andrew Davis

Kyle Krynski

Paul Ryan

Tejaswi Namuduru

Zening Chen

PURPOSE

The system that is being designed is an Android mobile application with the intent of being used as an invitation engine. Specifically, this refers to giving the users the capability to replace their colloquial, verbal communication of events with a simple software system, primarily with events that are too commonplace to actually make it to a more advanced calendar system such as Google Calendar. With such a system, the user should be able to do the following:

1) Create an event for their personal group
2) Create a group and invite other people to it
3) Be able to accept/reject an invite to a group
4) Create an event to push to other members of the group
5) Decide whether group members can see all of the information for an event (i.e. how many people are attending)
6) Be able to remove or edit an existing event
7) Decide which group members can delete or edit an existing group event
8) Be able to accept/reject event invitations
9) Be able to resolve schedule conflicts when adding new events
10) Be able to invite non-app users via email or text message
11) Allow people to view events for groups or lists
12) Allow people to easily push events to all of their graphical calendars
13) Allow users to set reminder for important events.
14) View an organized schedule of upcoming events

Design Outline

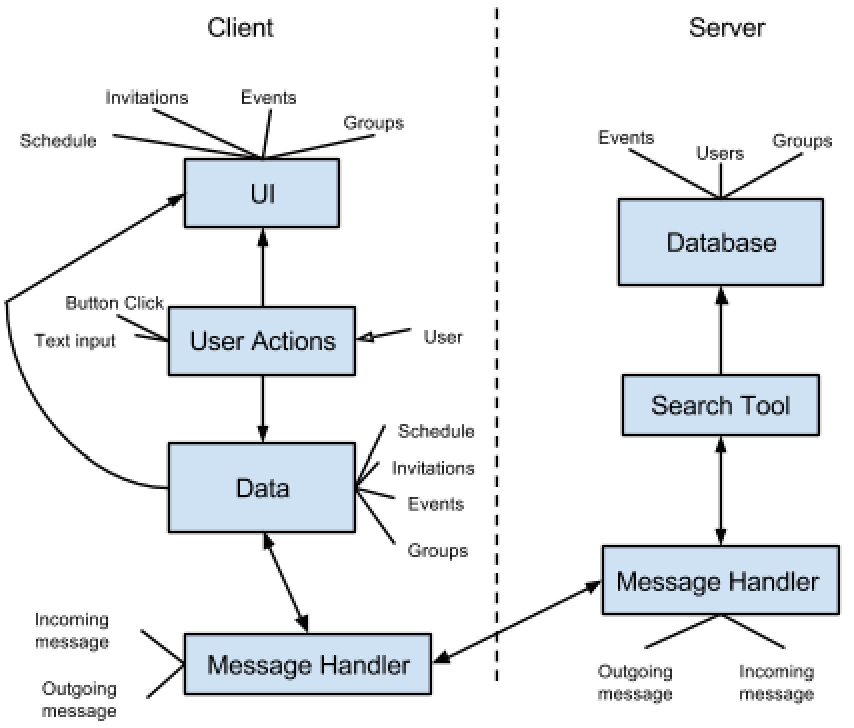
The overall system architecture will be divided into two main parts, according to standard client-server protocol. The client will be used to display information to the user, as well as store a local library of groups and events in an organized, simple format. The server will be used to handle event and group invitations on a global scale, allowing users to receive updates from a central hub.

Main tasks for the client:

1) Provide the users with a graphical interpretation of the events and groups they are part of or have been invited to join
2) Provide the users a graphical tool to successfully create an event or group
3) Provide the users a graphical way to decide whether to accept or reject an invitation
4) Provide the backend tools to transfer user graphical inputs into a data model.
5) Store the data model element locally for convenient representation
6) Send the data model element to the server to update global data
7) Receive data model updates from the server to display
8) Manage the instance where a connection is lost to the server

Main tasks for the server:

1) Manage a database of users, groups, and events
2) Receive update messages from clients about data model elements to store in the database
3) Receive request messages from clients to pull down any updated data models



A broad overview diagram of the various *structural units that both the client and server will have to implement, as well as the relations between them.*

Subcomponents of the Client

1) User Interface
    a) Display the data on events, groups, invitations, and other users to a single user.
    b) Display an interactive input area for the user to enter information to be sent to data.
    c) Prompt the user for a response to an event or group invitation

2) User Actions
    a) Control the flow of views to the screen based on user feedback
    b) Translate the user inputted information from a view to the data model

3) Data
    a) Hold all of the long term data for events, invitations, and groups.
    b) Hold any short term information like user or group searches received from the server.
    c) Receive data updates from the user actions or the server messages.
    d) Push any data updates to current view

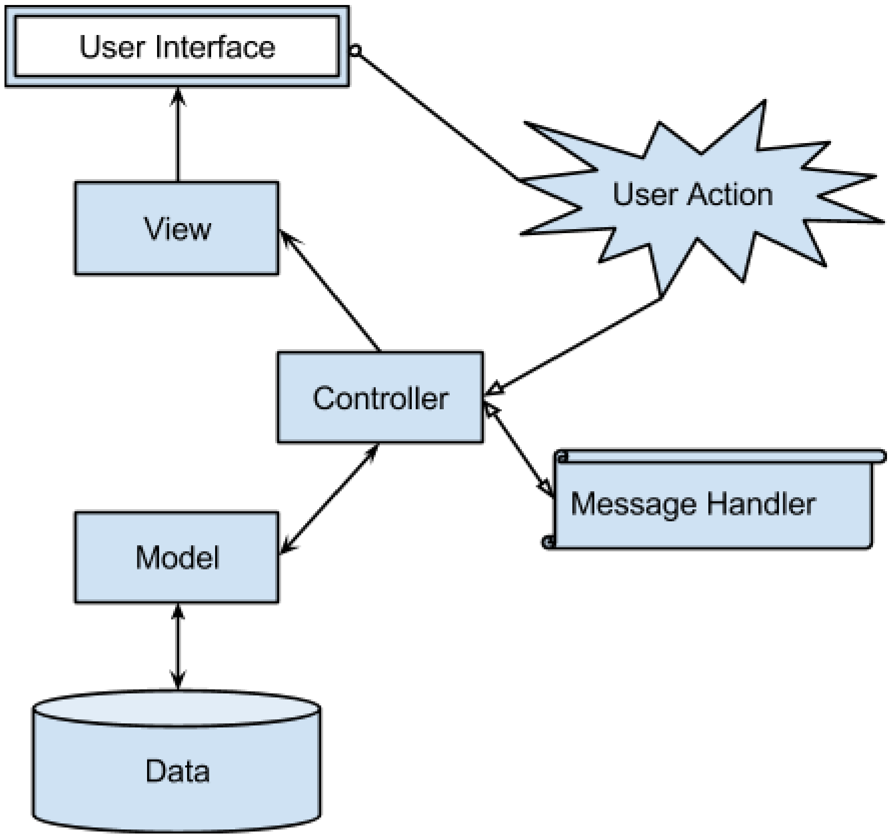
4) Message Handler
    a) Send any messages from the local data to the server
    b) Receive any data messages from the server
    c) Handle connection handshakes and user validation keys
    d) Handle any connection problems

Subcomponents of the Server

1) Database
    a) Store all of the user information, group information, and event information for remote access.
    b) Handle relations between data in an space and time efficient manner.
    c) Ensure secure transactions from server to client.
    d) Authenticate user queries.

2) Search Tool
    a) Handle any queries into the server database to retrieve or set data elements.
    b) Avoid manipulation of database elements by multiple source at the same time.

3) Message Handler
    a) Establish connections with clients.
    b) Validate client identities
    c) Pass on search or set information to the search tool appropriately
    d) Receive any search or set requests from clients
    e) Send any invitations from other users to specific clients
    f) Handle any connection problems



A more detailed figure of the client-side architecture implementing the Model-View-Controller protocol within the subcomponents described earlier.

Design Issues

Functional Issues:

Issue 1: What type of database model should be used?

- ❖ Option 1: Relational (SQL)
- ❖ Option 2: Object oriented
- ❖ Decision: Option 1 was chosen because our team is familiar with usage of SQL databases and it fits all of our searching, selecting, adding, and deleting needs.

Issue 2: How should backend authentication be handled?

- ❖ Option 1: authentication token
- ❖ Option 2: Username/Password
- ❖ Decision: Option 1 was chosen because we want to provide the highest possible security to our users at minimal cost to us.

Issue 3: What should the backend language be for the server?

- ❖ Option 1: PHP
- ❖ Option 2: Ruby/Ruby-on-Rails
- ❖ Decision: Option 2 was selected due to ease of use and multitude of gems available for Google Calendar, SQLite, OAuth, etc.

Issue 4: What is the format of the messages between client and server?

- ❖ Option 1: Use a JSON formated message to interact with the server.
- ❖ Option 2: Use XML as the message format.
- ❖ Decision: Option 1 was chosen as the message length is often smaller than an equivalent XML style query, this minimizes bandwidth usage and will improve response speed in data bottlenecked environments.

Issue 5: Will the app use local space?

- ❖ Option 1: Use a local cache for storing data before shutdown and quick load-ups.
- ❖ Option 2: Do not use local space, send data to server as the user operates and pull it all back down again upon a startup procedure.
- ❖ Decision: Option 1 was chosen because using a small amount of space to store the model locally will really improve load times on startup and allow the user to open the app without a connection and still see information.

Nonfunctional Issues:

Issue 1: How large of a server system is needed?

- ❖ Option 1: Use a small server system (i.e. a local computer) for now, then move the database to a larger machine as necessary.
- ❖ Option 2: Start on a large scale machine to avoid any moving complications and prepare for the future.
- ❖ Decision: Option 1 was chosen because the need for a large, possibly expensive database is just not part of the reality for small scale testing that will be going on.

Issue 2: What type of cryptographic table and algorithm should be used to ensure that the server database is secure from attack?

- ❖ Option 1: The user will enter a password which will be hashed and sent to the server for authentication with each additional request using a stored session ID.
- ❖ Option 2: Use a Google provided OAuth server and let authentication be processed through Google, then use that generated session token for access.
- ❖ Decision: Option 2 was chosen for ease of use and to take advantage of the more mature security system they can provide.

Issue 3: How should users with older versions of Android be treated from the apps perspective?

- ❖ Option 1: Only use older functions within the Android SDK to allow compatibility with older versions.
- ❖ Option 2: Treat them like non-smartphone users and revert to text based notifications.
- ❖ Decision: Option 2 was selected because many of the most efficient and useful features of the Android SDK from a developer perspective do not exist before Android 4.2. These features will be useful to save time and space, so it is best to treat older versions of Android like a non-smartphone.

Issue 4: How can users invite friends without smartphone by text or email?

- ❖ Option 1: Parse emails server-side to try to figure out usernames, event details, etc.
- ❖ Option 2: Don't allow non-mobile users to invite. Treat them as temporary users and allow them to accept invites only.
- ❖ Decision: Option 2 was selected because it is easier to implement, and in a realistic use case, someone who doesn't have the app won't be a consistent user of our service, especially when it comes to inviting other users to events.

Design Details

Client Class Design

Profile:

- The class entity that handles all of the user information that is available to send to the server.
- This information includes name, identification key, gender, birth date, phone number, email address, etc.

Acquaintance:

- The entity representing a public form of a profile or user. It will contain only brief, non-sensitive information on a user for the purposes of being kept on a client-side level.
- Accessible from the profile of a client, but groups will contain direct links to them for information on who is in the group.

Group:

- Entity representing a list of people associated with a specific set of events. This is more of an association class that will allow for the easy organization of events, people, and the current user as it relates to scheduling.
- The data in this class will be a list of events and a list of people.
- Important note: there can never be zero groups, as each user has their own “personal” group that cannot be joined or removed. Without groups, there can be no events in this architecture.

Event:

- Entity representing an event in time, answering the questions: who, what, where, when, why.
- Contains names, start date, end date, location, description, etc.

View:

- Class containing all of the different view items and how they will be formatted on screen.
- Can only be invoked by the controller class, and has minimal runtime (running on the cpu when updating view).

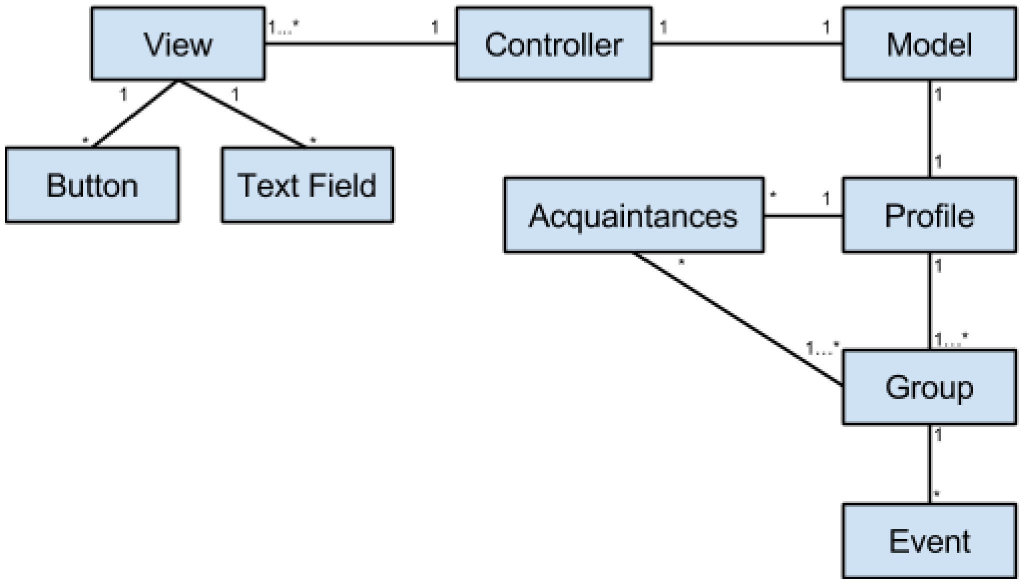
Controller:

- Entity responsible for handling all of the decisions of the client app. Such decisions will come from user interactions and messages from the server.
- Will be responsible for updating data to the model and ensuring that the data gets sent to the view for repopulation.
- Maximal runtime (always running on the cpu).

Model:

- Entity responsible for holding instances of data classes that are in use by the controller.

- Can only be called by the controller class, negligible runtime (only runs on the cpu when the controller asks it to).



A class interaction diagram showing the relationships between the main classes in the system on the client side application. Note: Many classes provided by the Android SDK have been left out for simplification purposes.

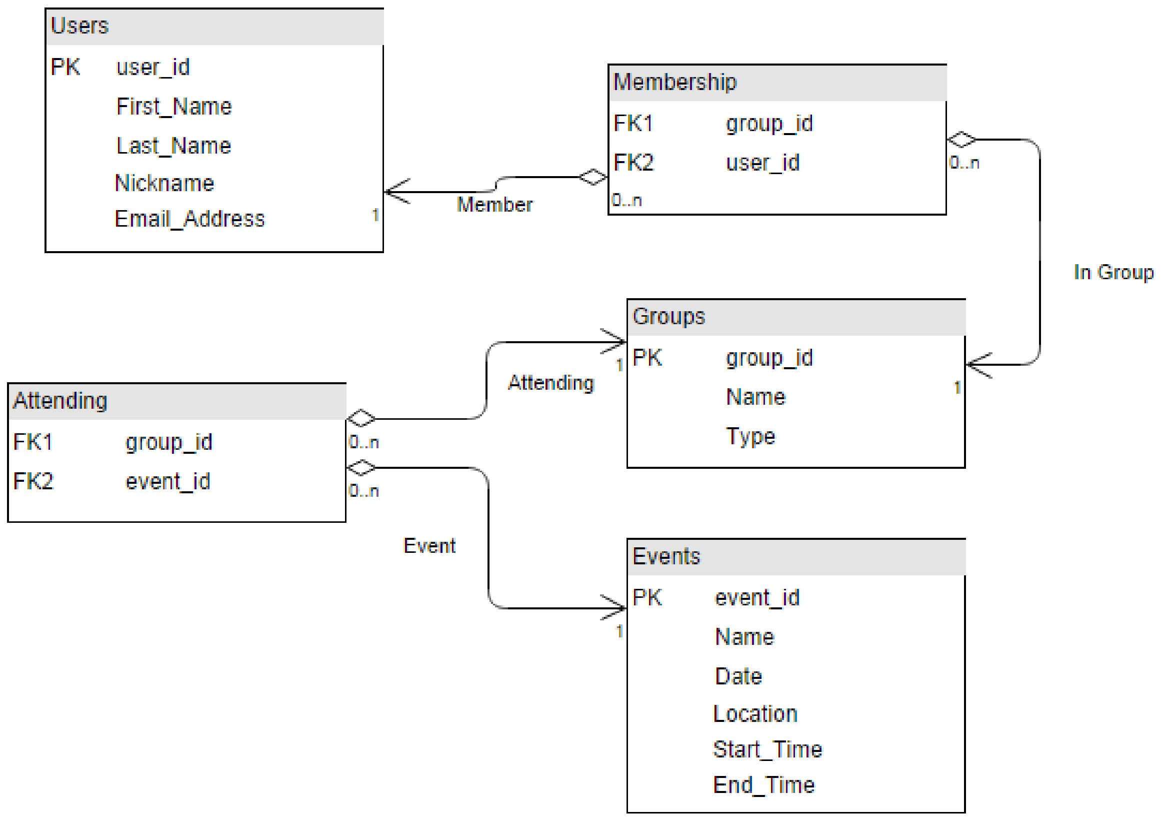
Database Design

User:
- Contains basic user information like name and Email
- May contain secure user information such as passwords and information the user has requested no public

Group:
- Will include information about a group including the group name and owner.
- Can contain statistics about the group for analytics

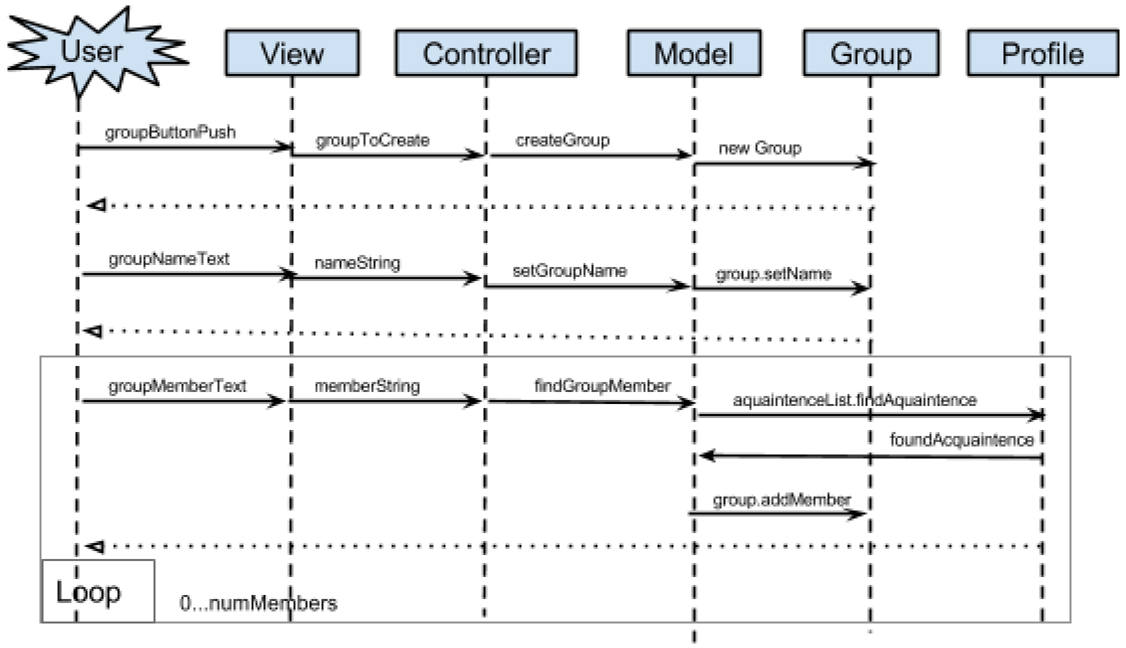
Event:
- Contains basic information on an event
- Includes name, date, time, location and if relevant the repeat pattern.

Membership and Attending:
- Relational tables used for database normalization

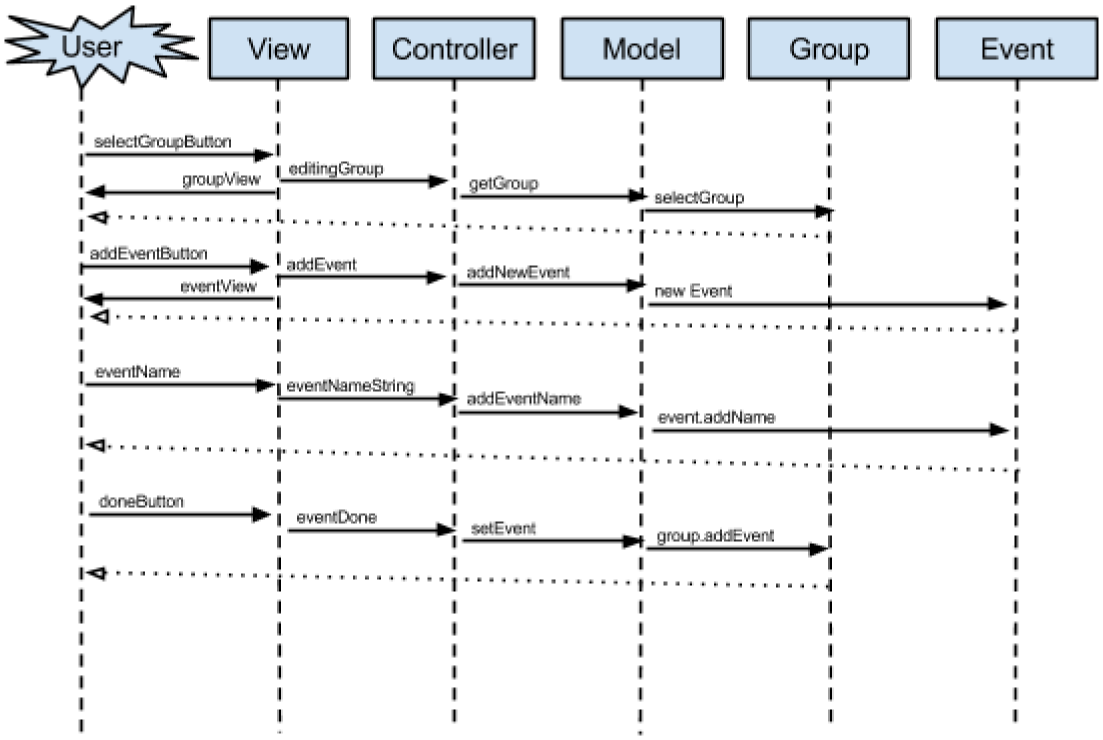


Sequence Diagrams

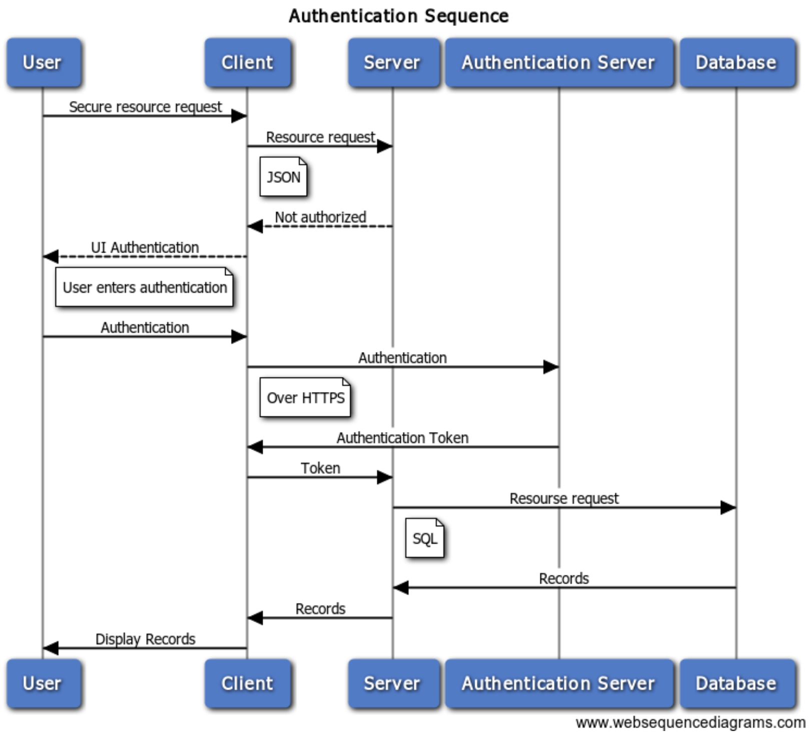
1) Creating a group, while adding users



Creating a new group and adding members to it. This allows a user to interface with a group creation tool on the user interface and send the various bits of data down to the proper data classes through the controller and model. The server interactions of the controller class have been left out for simplicity.

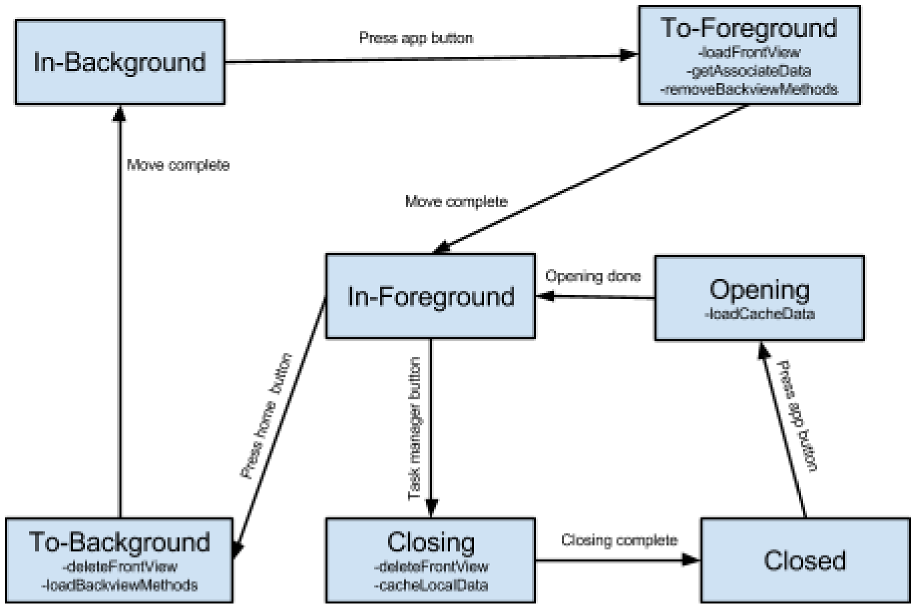


Creating an event and adding it to a group. This allows a user to push an event to a group. The backend classes will allow the event to be stored in the proper group, with the proper information. The interactions between the server and controller class have been left out for simplicity.



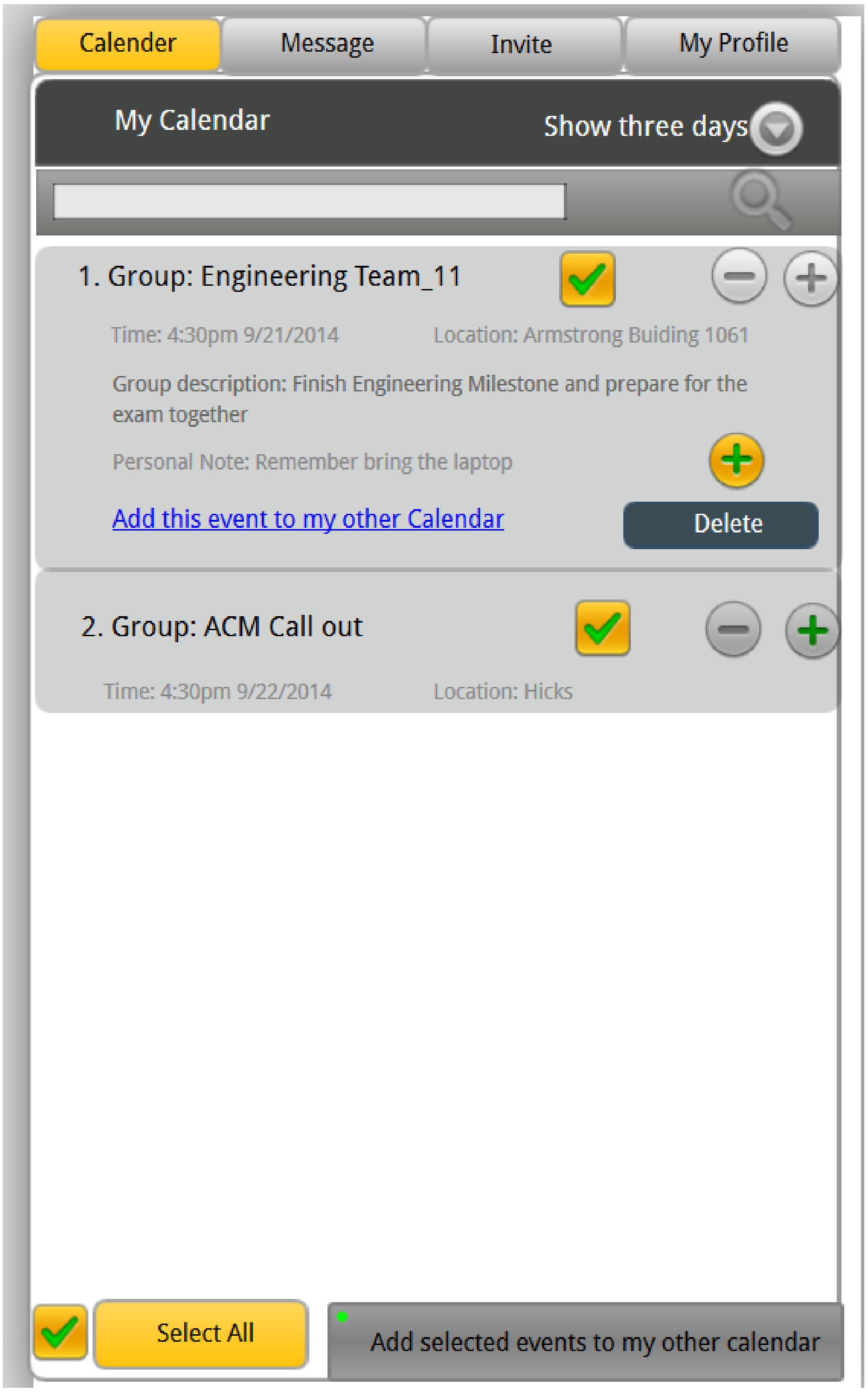
Handling specific interactions between high level classes. These class interactions exist between the server and the client, and are more abstract than the previously designed classes. Includes authentications and database requests.

State Diagrams



Managing the process surrounding opening, closing, and going to the background as an app. Basically, methods will be needed to respond to the transfer of states of the system. If the app is moving between the foreground and background, the essential functions will be around how the view is handled. The controller will still be able to operate as a message listener from the server in the background, but the view will need to behave around whatever app is actually running in the foreground. When the app closes, however, the entire system will be lost, so data will need to be stored in a local cache for a quick opening process.

Graphical User Interface Mockup



1. Group: Engineering Team_11

Time: 4:30pm 9/21/2014 Location: Armstrong Buiding 1061

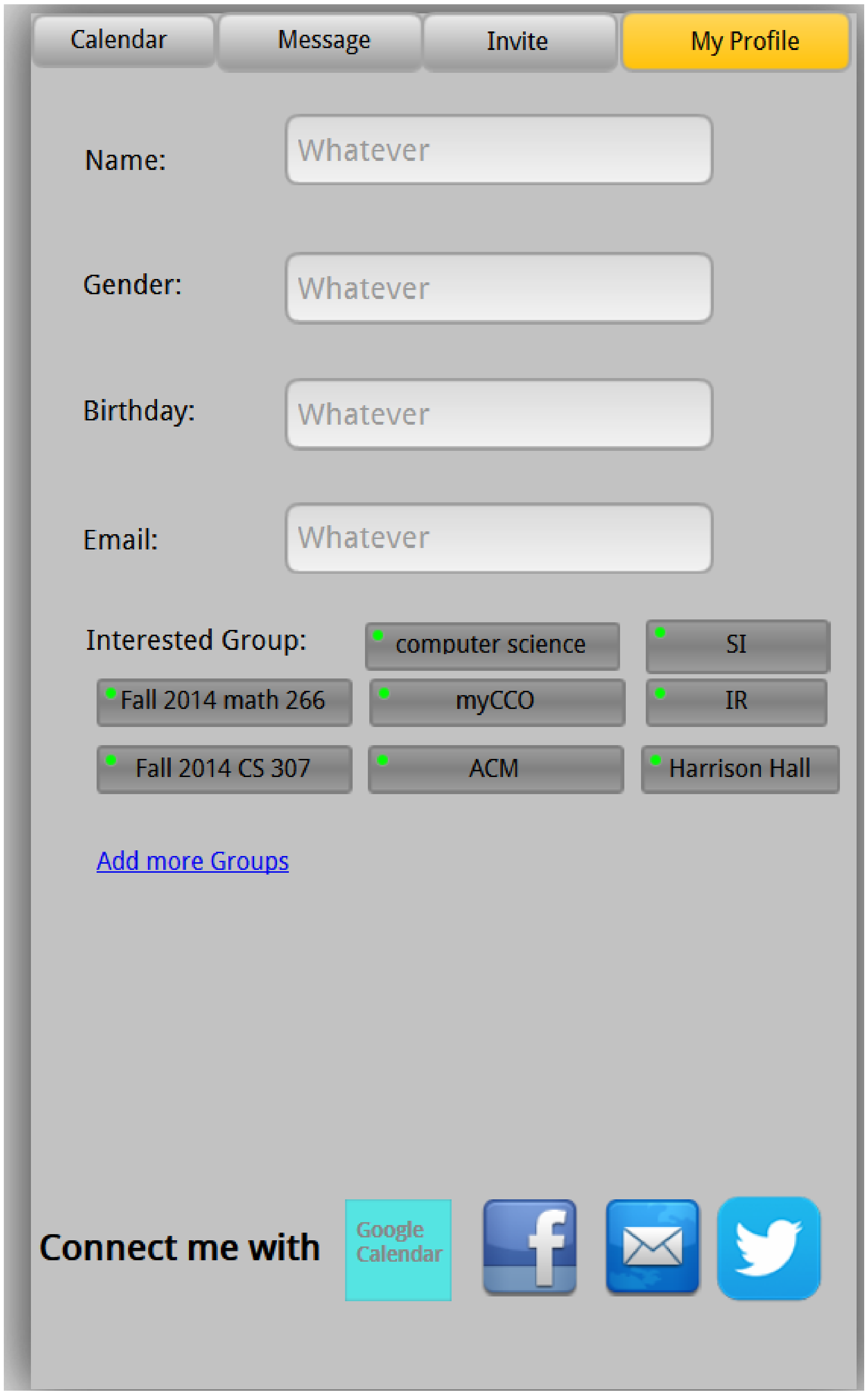
Group description: Finish Engineering Milestone and prepare for the exam together

Personal Note: Remember bring the laptop

Add this event to my other Calendar

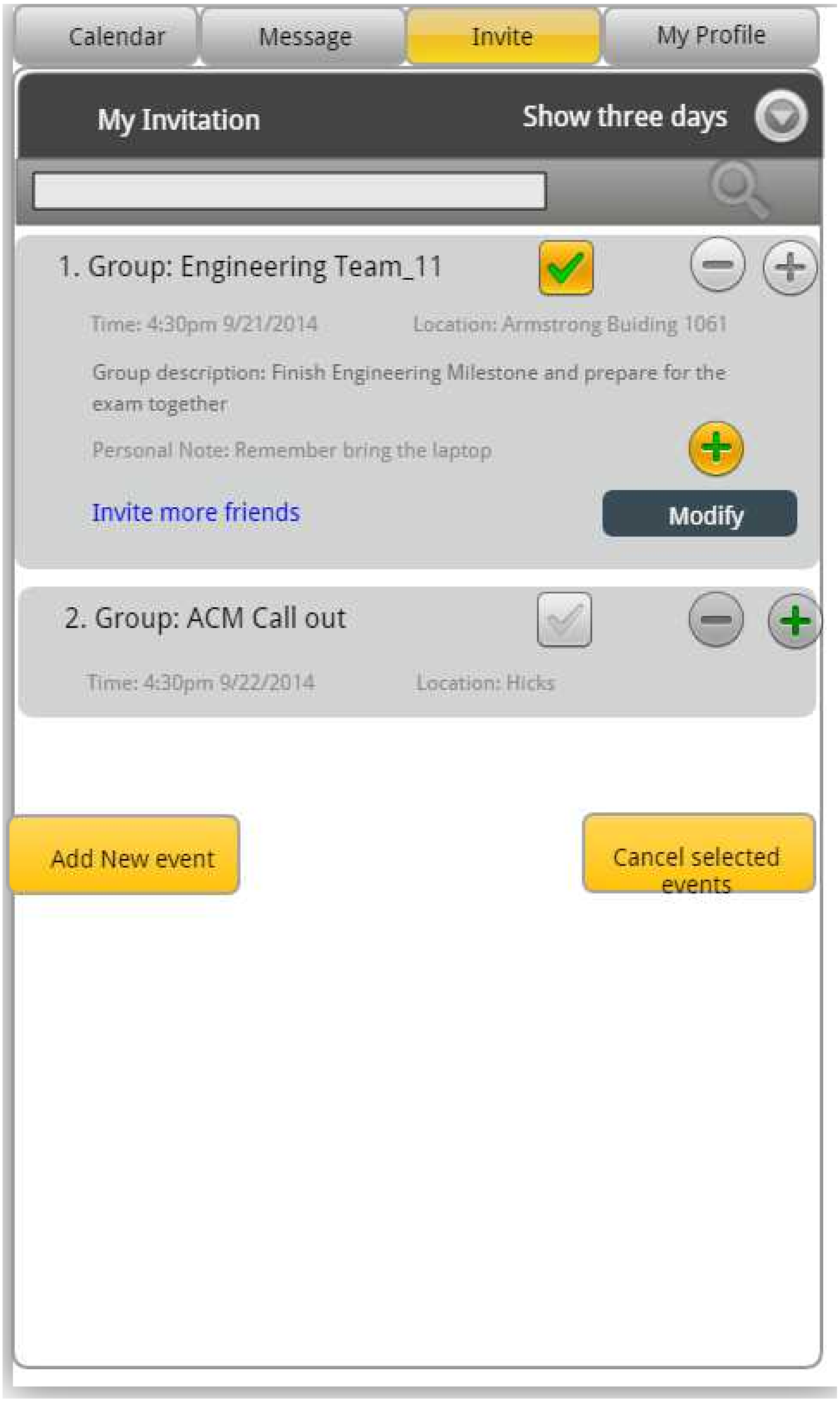
2. Group: ACM Call out

Time: 4:30pm 9/22/2014 Location: Hicks



Name:	Whatever
Gender:	Whatever
Birthday:	Whatever
Email:	Whatever

Interested Group: computer science, SI, Fall 2014 math 266, myCCO, IR, Fall 2014 CS 307, ACM, Harrison Hall



My Invitation

1. Group: Engineering Team_11

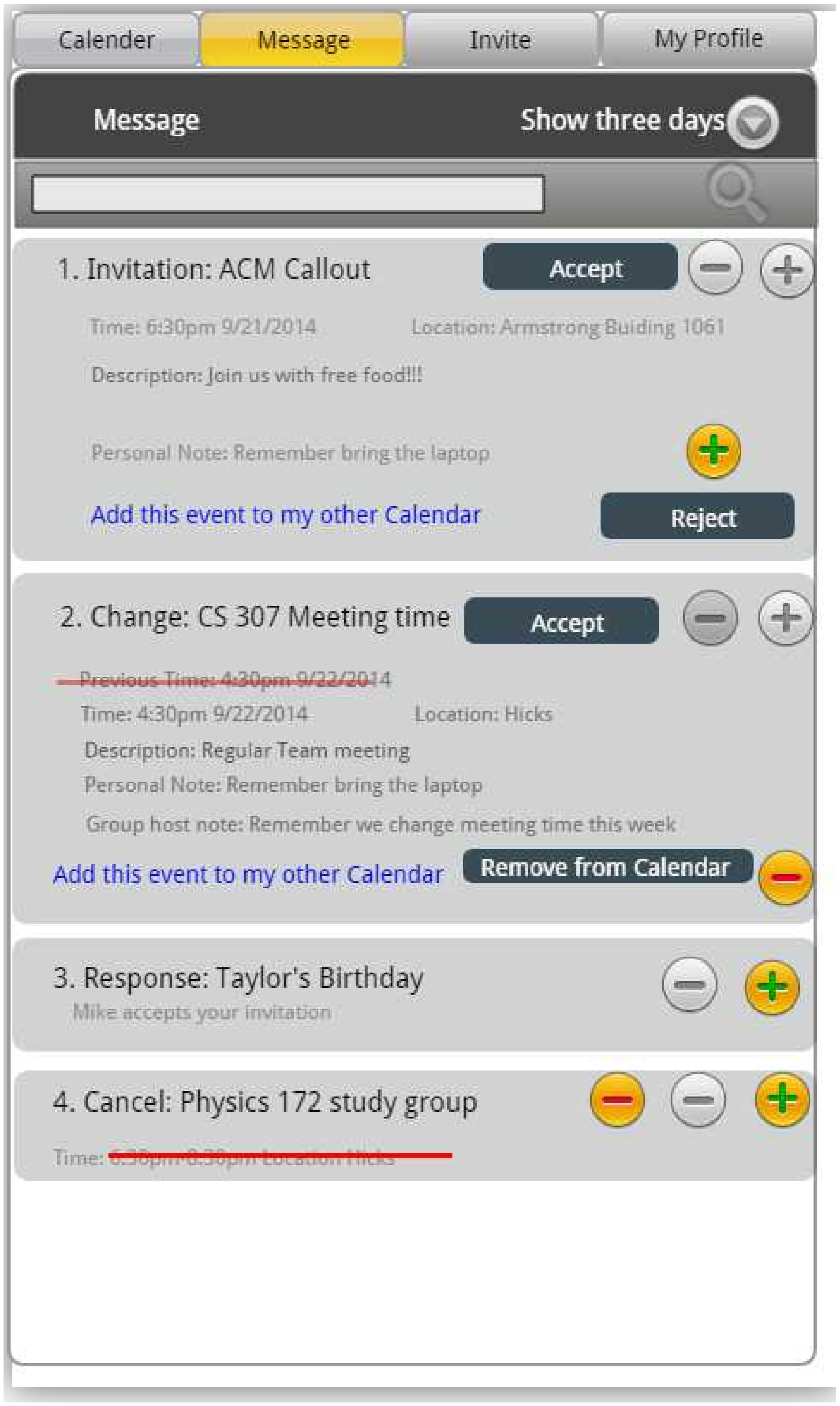
Time: 4:30pm 9/21/2014 Location: Armstrong Buiding 1061

Group description: Finish Engineering Milestone and prepare for the exam together

Personal Note: Remember bring the laptop

2. Group: ACM Call out

Time: 4:30pm 9/22/2014 Location: Hicks



Calender | Message | Invite | My Profile

Message   Show three days

1. Invitation: ACM Callout   Accept

Time: 6:30pm 9/21/2014   Location: Armstrong Buiding 1061

Description: Join us with free food!!!

Personal Note: Remember bring the laptop

Add this event to my other Calendar   Reject

2. Change: CS 307 Meeting time   Accept

~~Previous Time: 4:30pm 9/22/2014~~

Time: 4:30pm 9/22/2014   Location: Hicks

Description: Regular Team meeting

Personal Note: Remember bring the laptop

Group host note: Remember we change meeting time this week

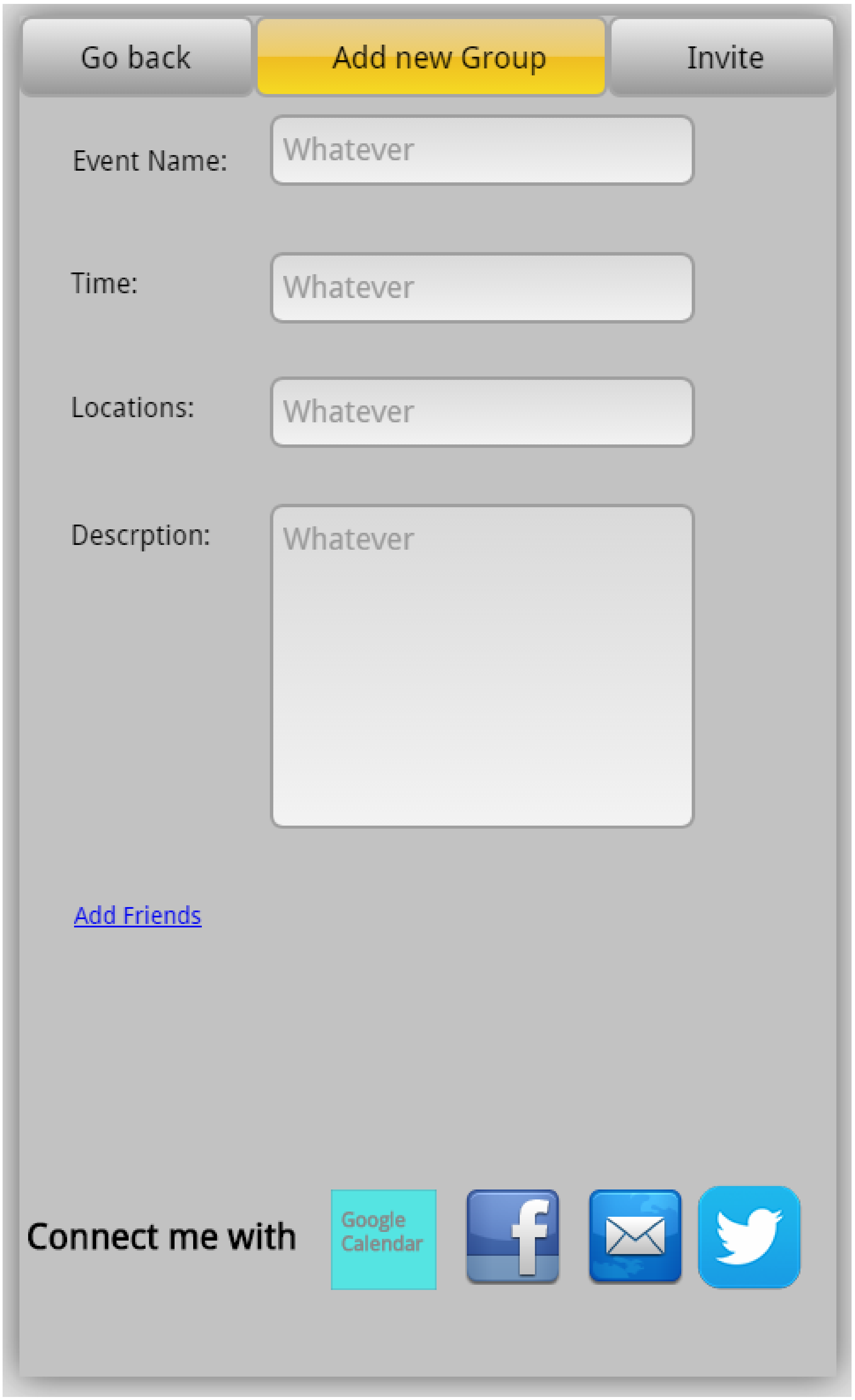
Add this event to my other Calendar   Remove from Calendar

3. Response: Taylor's Birthday

Mike accepts your invitation

4. Cancel: Physics 172 study group

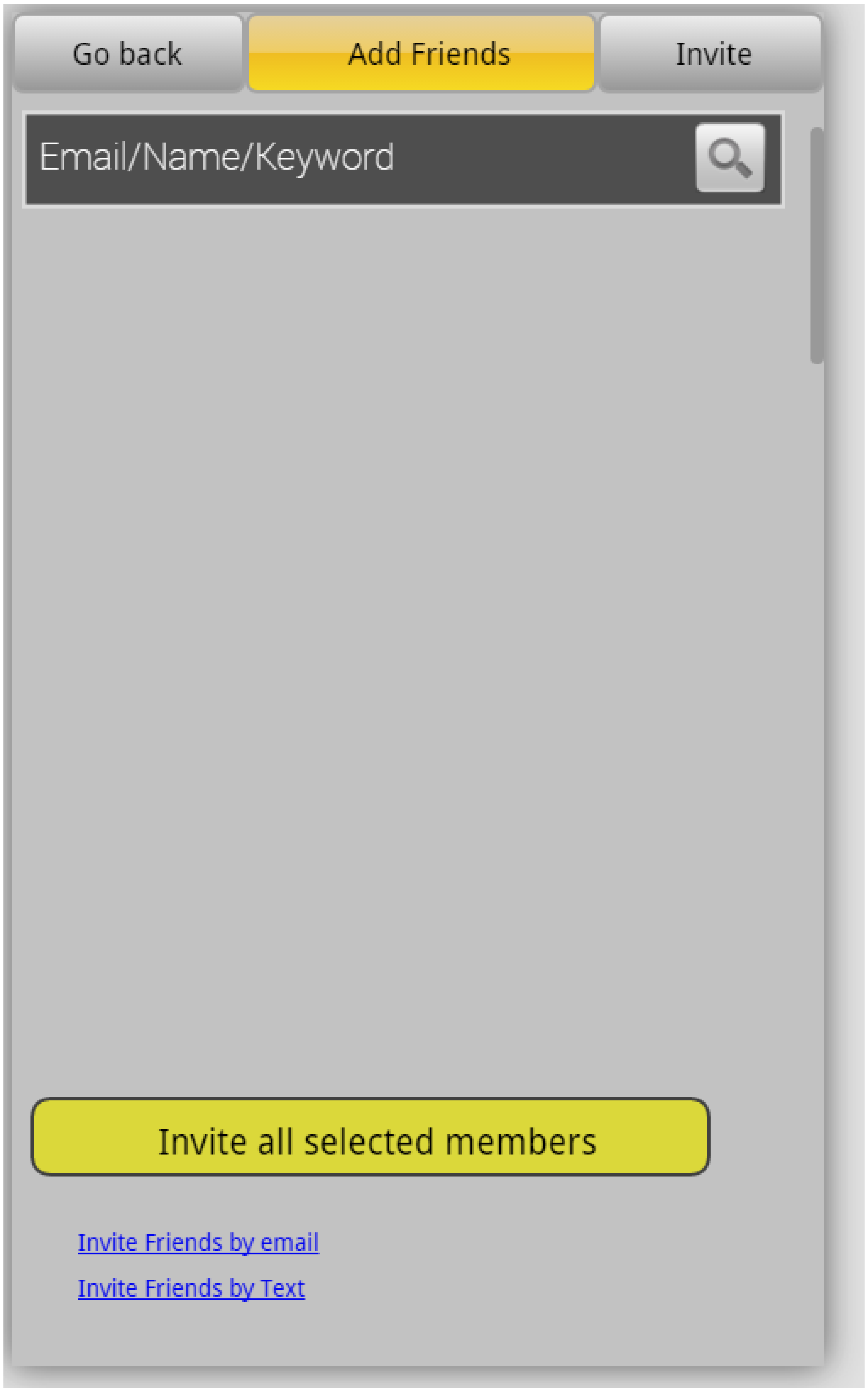
Time: ~~6:30pm-8:30pm Location Hicks~~



Event Name:	Whatever
Time:	Whatever
Locations:	Whatever
Descrption:	Whatever

Add Friends

Connect me with


Go back
Add Friends
Invite
Email/Name/Keyword
Invite all selected members
Invite Friends by email
Invite Friends by Text