Party Pass

Design Document
CompSci320, Spring 2017

Team Party Hardy

Manager: Tamara Zbitnaja

Nick Merlino	Roman Ganchin
Josh Silvia	Michael Gallant
Matt Cassano	Will Sattanuparp

Table of Contents

1- System Overview

1.1 - Purpose

PartyPass is a mobile app designed to let users register parties to an administrative body in order to allow for warnings to be sent digitally instead of in person. Party hosts will send information about their party to the administrators with relevant information such as the address, start and end time, and host information. The real-world application of this app is of a police force interacting with house parties. The benefit to the police force is they do not have to waste time sending officers to issue warnings in person, when they could be doing much more pressing things. The benefit to party hosts is that they do not have to risk in person interactions.

The app is built with Meteor and Ionic, an HTML-5 based hybrid mobile app framework that is built on top of Cordova and AngularJS. Documentation for these can be found under section 5 of this document. The system is broken up into a Model, View, and Controller in order to allow for better testability and extensibility. The Model class is responsible for storing the database entries. The View is responsible for presenting the system to the user or administrator. The Controller is responsible for updating the model. Their relationships are visible in 2.1.

1.2 - Design Rationale

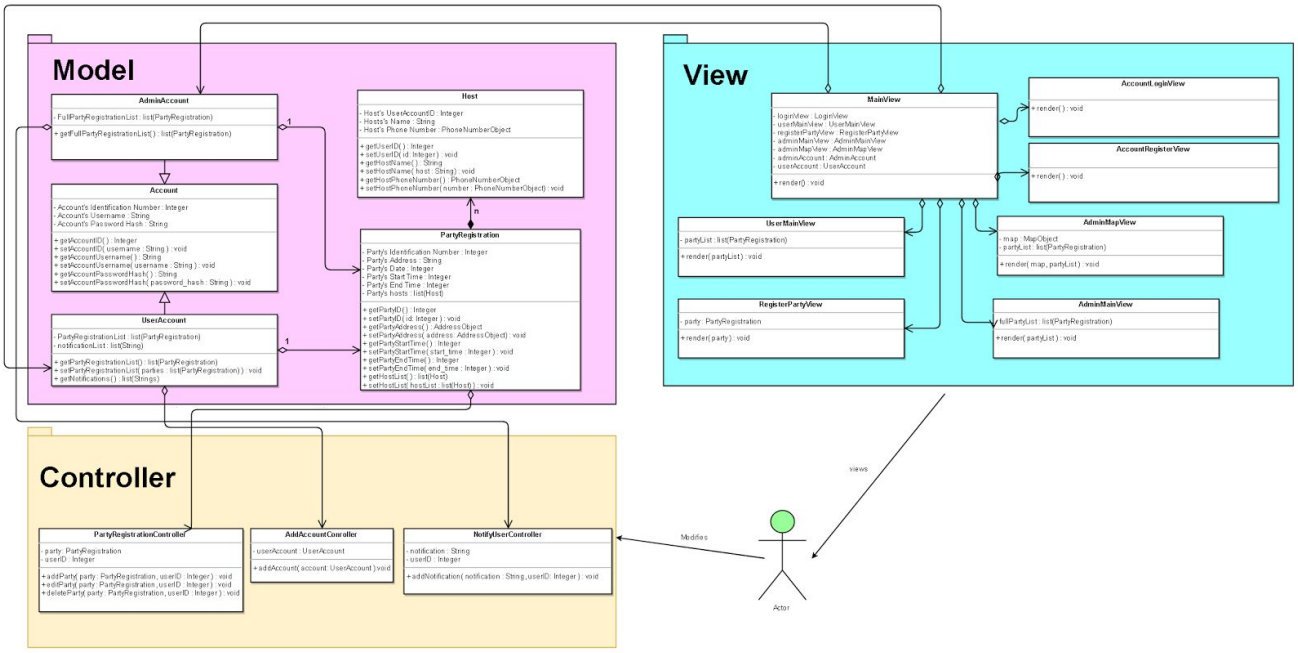
We chose to use the Meteor and Ionic frameworks as they are designed to employ the cross-platform mobile integration that we must have. We chose to use a MVC architecture as it allows for easy extensibility/testability and is in our opinion a good representation of the tasks we need to accomplish. We also chose to make a single MainView which renders aggregated views into it. This continuous rendering allows us to dynamically and hopefully seamlessly modify what is being presented to the user at any time. We decided to have 3 separate controllers because their functionality is slightly different in each of them and they are related to different components in the model. In the model, our main reasoning was to represent all of our information through the 2 different possible user types: adminAccount and userAccount. These two accounts will contain all the possible information about our app, and the view will display the corresponding information. Given there may be infeasible/undesired aspects of the UML design, we were advised to mention that we did our best on the UML and that we might have a slightly different implementation.

1.3 - Design Constraints

In order to update the user/administrator instantly when a change is made, we will have a single centralized database and only allow the mobile app to display data when connected to that database. This will guarantee that out of date information isn't being displayed. This does mean, however, that an internet connection is necessary for the app to function.

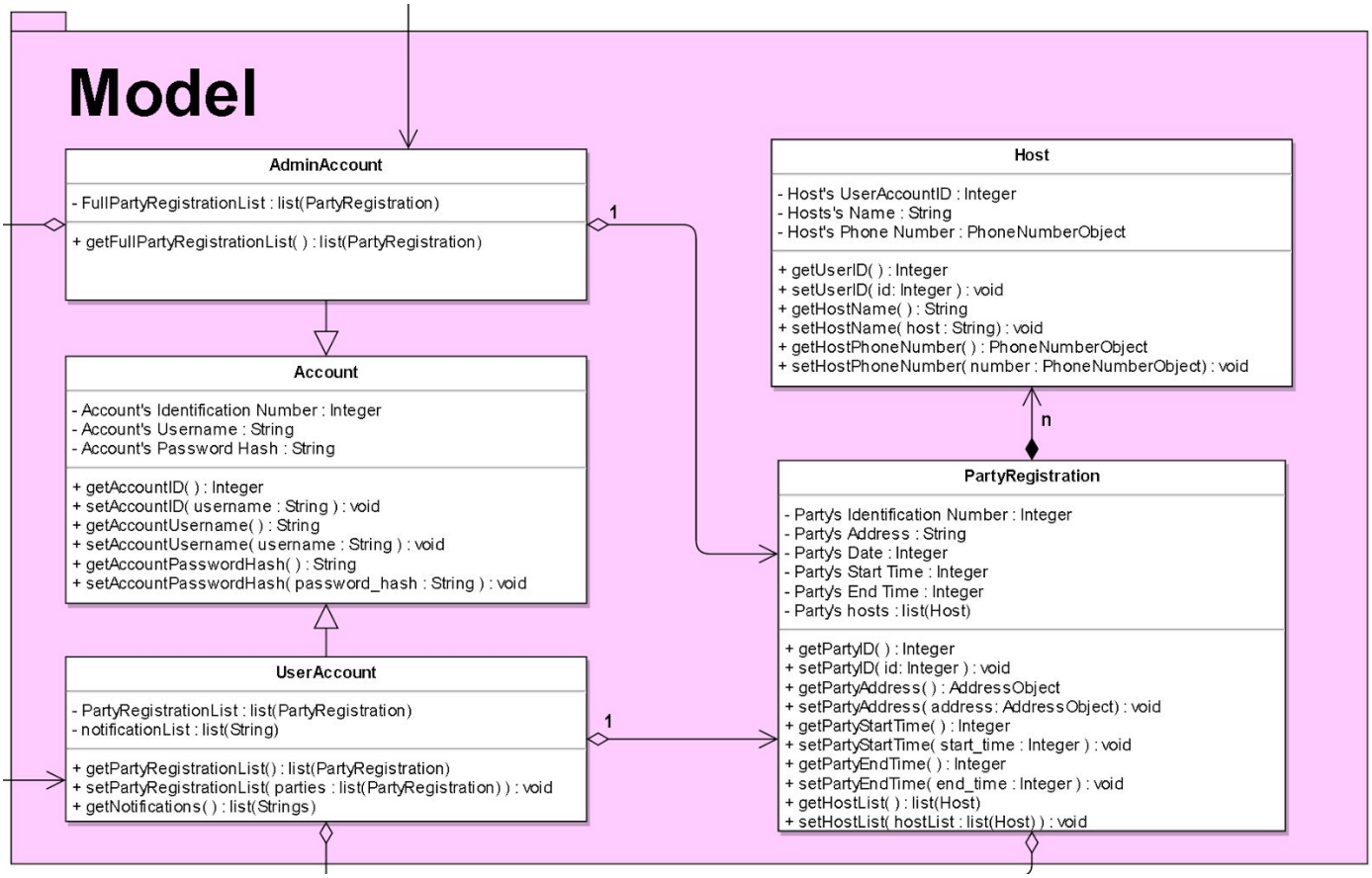
2 - Software Design & Architecture

2.1 - Universal Modeling Language Diagram

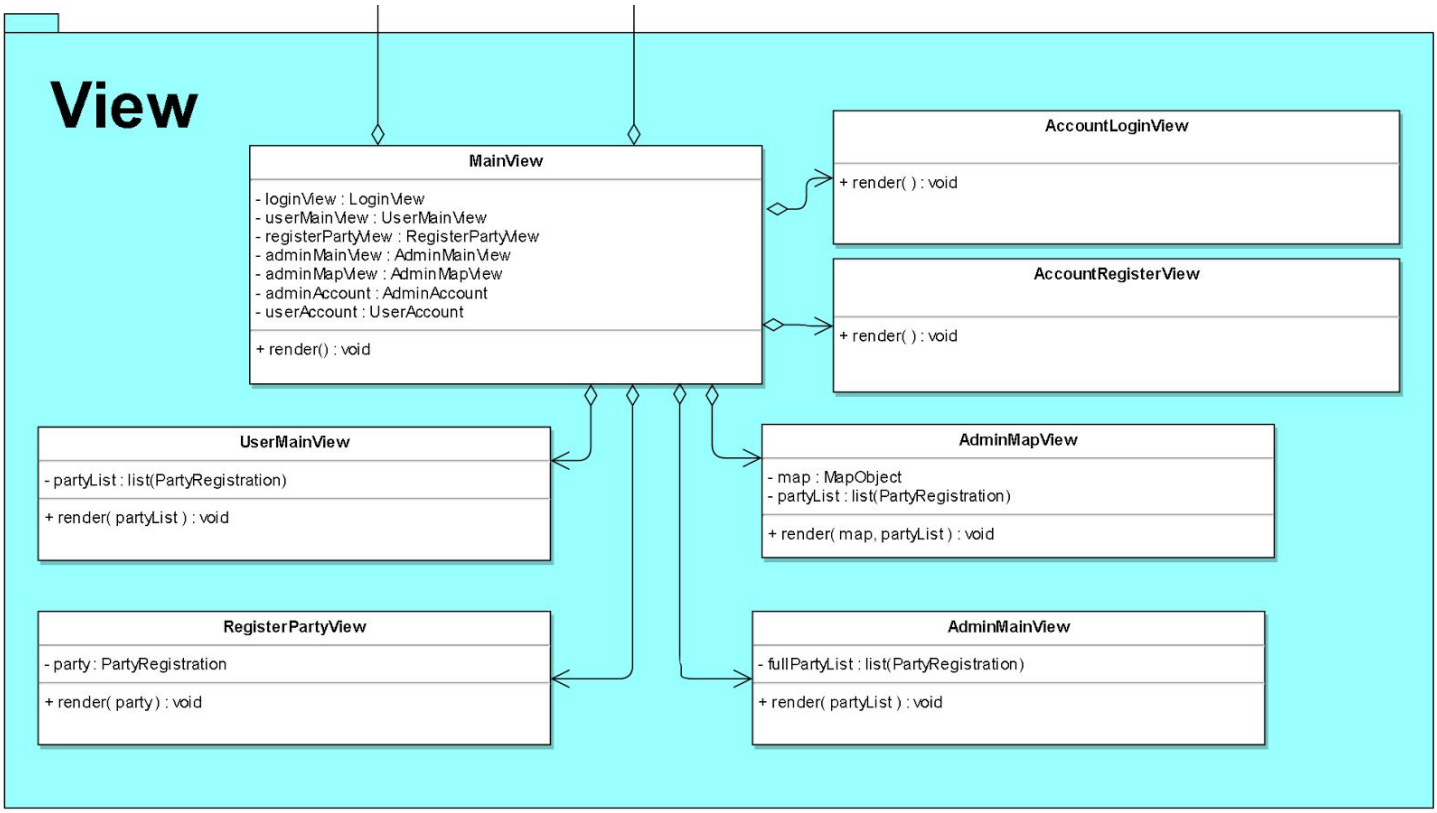


This diagram is a “big picture view” that shows a general overview of our uml diagram. There are snippets of each individual part below describing what each part of the UML diagram does and what its purpose will be in our system.

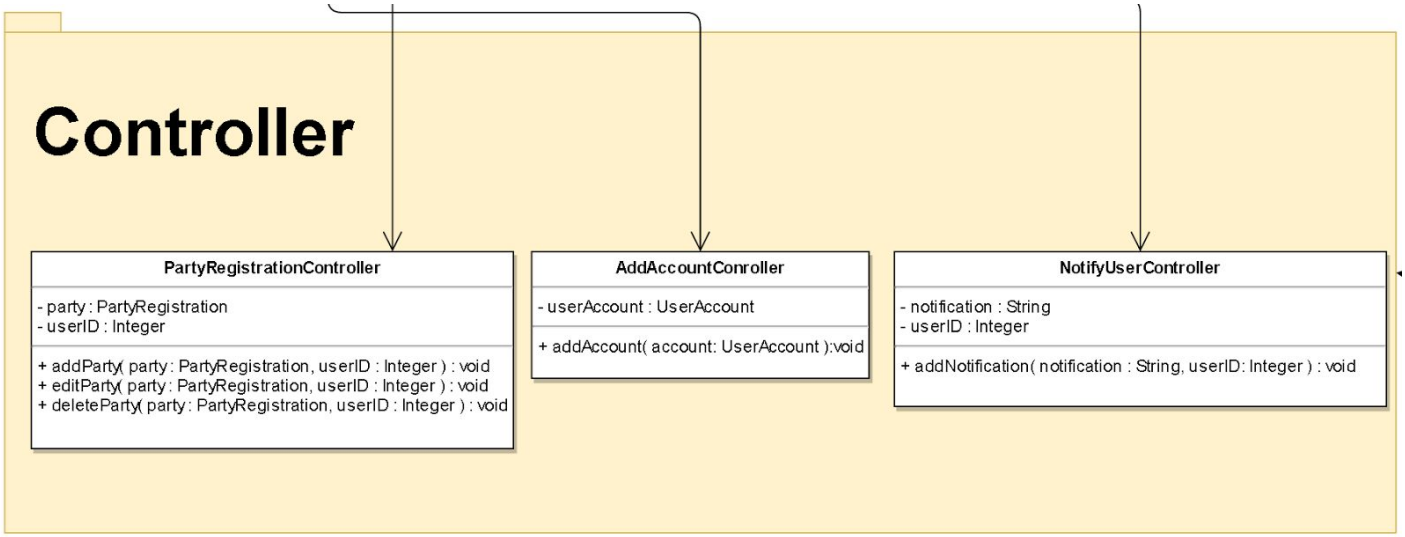
2.1.1 - Model


Model
AdminAccount
- FullPartyRegistrationList : list(PartyRegistration)
+ getFullPartyRegistrationList( ) : list(PartyRegistration)
Account
- Account's Identification Number : Integer
- Account's Username : String
- Account's Password Hash : String
+ getAccountID( ) : Integer
+ setAccountID(username : String) : void
+ getAccountUsername( ) : String
+ setAccountUsername(username : String) : void
+ getAccountPasswordHash( ) : String
+ setAccountPasswordHash(password_hash : String) : void
UserAccount
- PartyRegistrationList : list(PartyRegistration)
- notificationList : list(String)
+ getPartyRegistrationList() : list(PartyRegistration)
+ setPartyRegistrationList(parties : list(PartyRegistration)) : void
+ getNotifications( ) : list(Strings)
Host
- Host's UserAccountID : Integer
- Hosts's Name : String
- Host's Phone Number : PhoneNumberObject
+ getUserID( ) : Integer
+ setUserID(id: Integer) : void
+ getHostName( ) : String
+ setHostName(host : String) : void
+ getHostPhoneNumber( ) : PhoneNumberObject
+ setHostPhoneNumber(number : PhoneNumberObject) : void
PartyRegistration
- Party's Identification Number : Integer
- Party's Address : String
- Party's Date : Integer
- Party's Start Time : Integer
- Party's End Time : Integer
- Party's hosts : list(Host)
+ getPartyID( ) : Integer
+ setPartyID(id: Integer) : void
+ getPartyAddress( ) : AddressObject
+ setPartyAddress(address: AddressObject) : void
+ getPartyStartTime( ) : Integer
+ setPartyStartTime(start_time : Integer) : void
+ getPartyEndTime( ) : Integer
+ setPartyEndTime(end_time : Integer) : void
+ getHostList( ) : list(Host)
+ setHostList(hostList : list(Host)) : void
1
1
n

2.1.1 - View

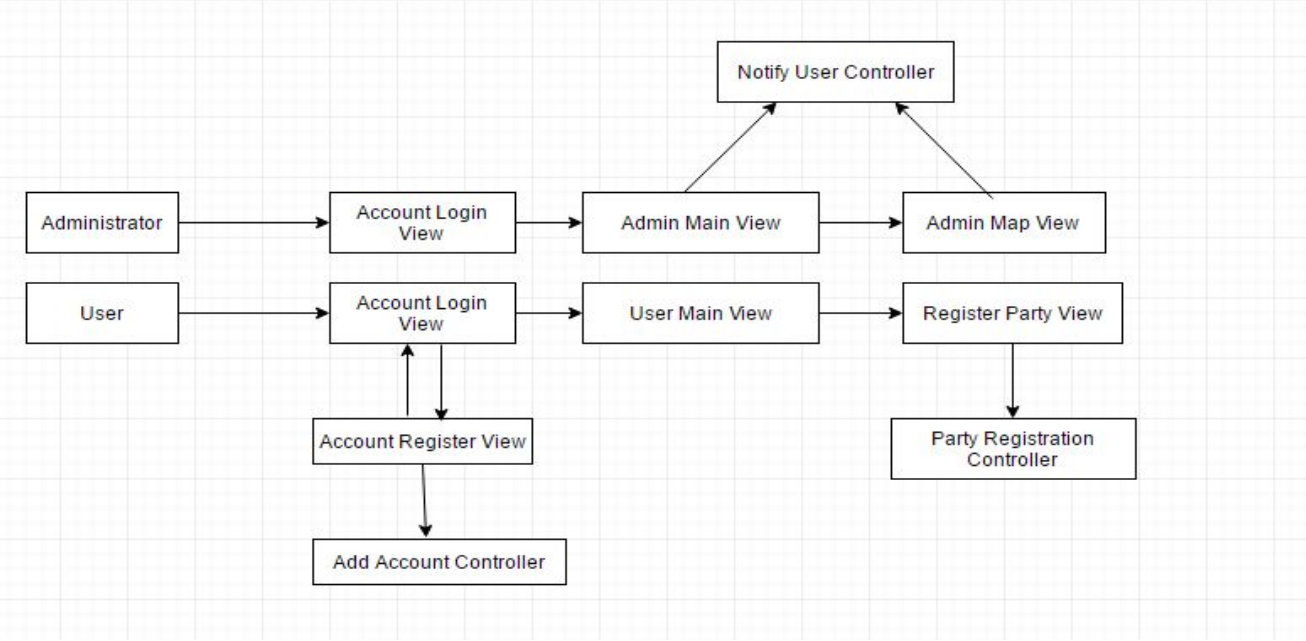


2.1.1 - Controller

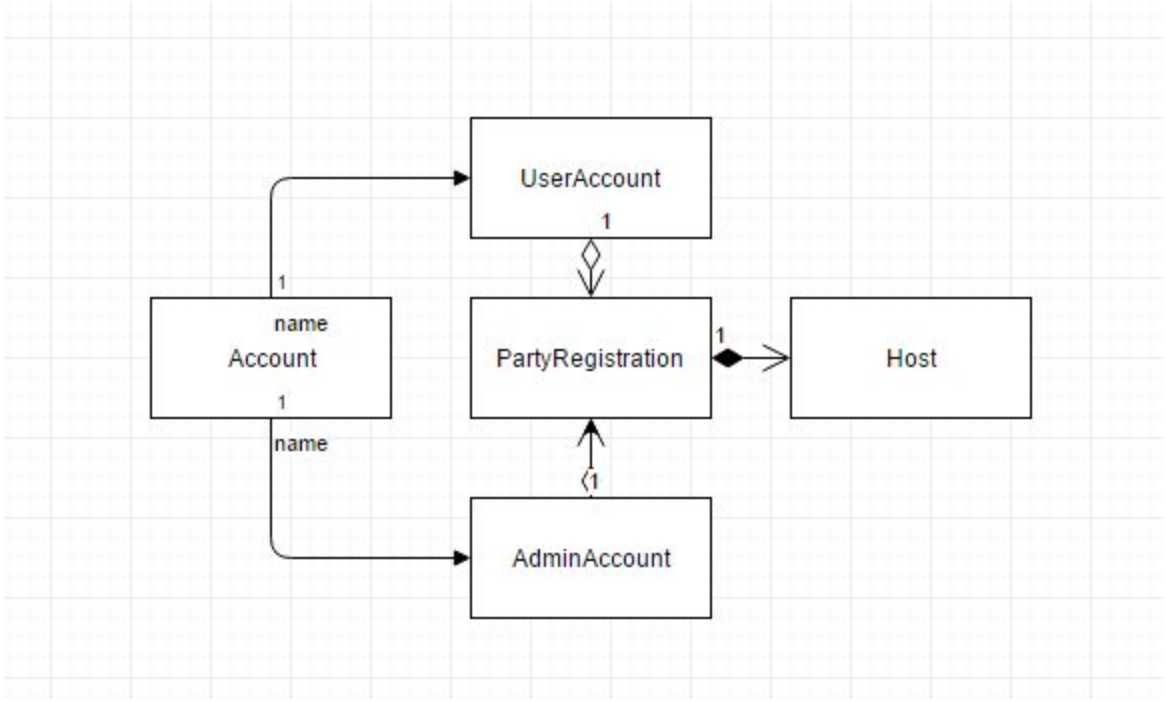


2.2 - Inter-Component Interfaces

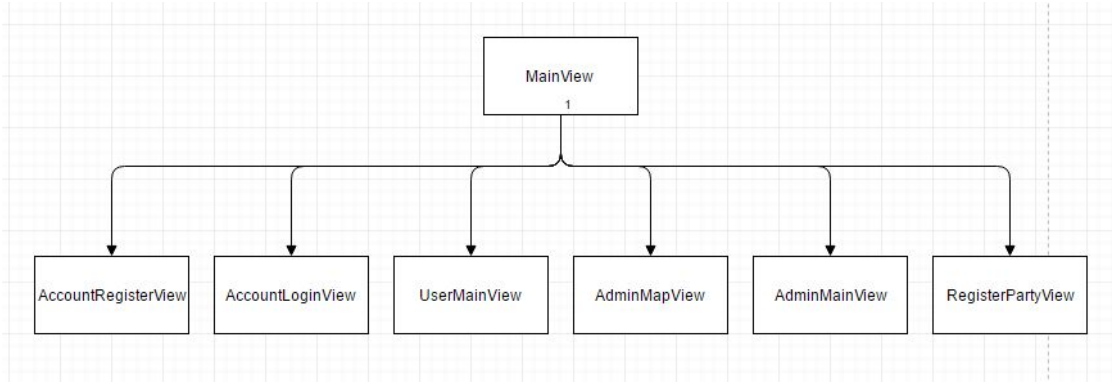
Represented in our model is two different accounts to login which are administrator and user. A user will be able to see the parties that their Account ID is associated with, while an administrator will be able to see all of the registered parties. The view component will present the user with an empty MainView, to be populated by one of the other views depending on the use of the app. The model and view components are connected in that both accounts will see and be able to navigate between six different screens (UserMainView, AdminMainView, RegisterPartyView, AdminMapView, AccountRegisterView and AccountLoginView). The different user and administrator information will also be displayed on the view screens. From the AdminMainView and the AdminMapView the administrator account can send a warning to the user through the NotifyUserController. When in the AccountRegisterView the user can register an account which utilizes the AddAccountController. This can't be used to make an administrator account as that needs special privileges. This controller is used to just make a user account. In the RegisterPartyView the user can register a party using the PartyRegistration controller.



2.3 - Intra-Component Interfaces

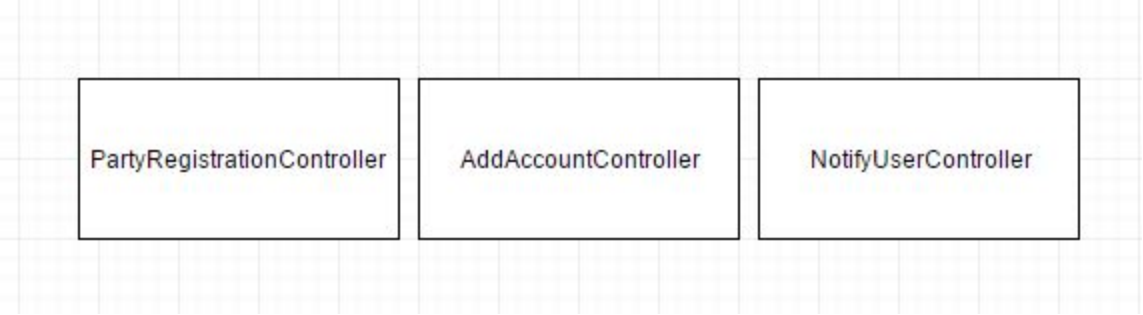


Model:In our model there are 2 types of accounts used for logging in. Both types of accounts will need to be defined by their username and password (hashed for security). Both accounts will be assigned an account ID that is unique to each individual account. Specific to the administrator account will be the ability to view all parties in the PartyRegistration list. The Users will be able to view all parties their account ID is associated with, as well as notifications that are sent to them. These users will be able to create a party defined by address, date, start and end time, and a list of hosts. The party itself will be assigned a party ID that is unique to the individual party. The Hosts are defined by either a user ID or a combination of the host's name and phone number.



View: There will be a view interface that has a single method to render the screen. The MainView will manage six screens: UserMainView, AdminMainView, RegisterPartyView,

AdminMapView, AccountRegisterView, and AccountLoginView. The MainView will render the correct view based on the state of the program.



Controller: The individual controllers do not interact with one another, and are completely unique to one another. PartyRegistrationController will allow the creation and modification of party. AddAccountController will allow creation of accounts. NotifyUserController will enable an administrator to notify a user.

3 - Model

3.1 - PartyRegistration

Summary:

- Contains the data for the party, including the variables below. This will be added to a user's list of PartyRegistrations to later be viewed/modified.

Variables:

- Party ID : Integer - The identifying number of the party. Stored as an Integer.
- Party Address : String - The location of the party. Stored as a String.
- Party Date : Integer - The date of the party. Stored as an Integer.
- Party Start Time : Integer - The time the party starts. Stored as an Integer.
- Party End Time : Integer - The time the party ends. Stored as an Integer.
- Host List : List - A list of hosts associated to the party. Each is a Host object.

Methods:

- getPartyID( ) : Integer
    - Gets the Party ID integer
- setPartyID(id: Integer) : void
    - Sets for Party ID integer
- getPartyAddress( ) : AddressObject
    - Gets Party Address object
- setPartyAddress(address: AddressObject) : void
    - Sets Party Address object
- getPartyStartTime( ) : Integer
    - Gets Party Start time integer

- setPartyStartTime(start_time : Integer) : void
    - Sets Party Start Time integer
- getPartyEndTime( ) : Integer
    - Gets Party End Time integer
- setPartyEndTime(end_time : Integer) : void
    - Sets Party End Time integer
- getHostList( ) : list(Host)
    - Gets Host List
- setHostList(hosts : List) : void
    - Sets Host List, takes in a list containing all hosts for party

3.2 - Account

Summary:

- Contains the data for the account the user is on, including the variables below. This account class can be implemented as a user account or an administrator account.

Variables:

- Account's ID : Integer - The identifying number of the Account. Stored as an Integer.
- Username : String - The username for the account. Stored as a String.
- Password : String - The password the account is associated with. Hashed using an encryption algorithm and stored in a table for retrieval by entering a username.

Methods:

- getAccountID( ) : Integer
    - Gets the account ID of the user
- setAccountID(username : String) : void
    - Sets the the account ID for the user
- getAccountUsername( ) : String
    - Gets the username for the account
- setAccountUsername(username : String) : void
    - Sets the account username for the account
- getAccountPasswordHash( ) : String
    - Get account password for the account
- setAccountPasswordHash(password_hash : String) : void
    - Set the account password for the account

3.2.1 - AdminAccount

Summary:

- Contains the data for the administrator account, including the variables below. This class will be used to view the list of all registered parties and warn party hosts.

Variables:

- FullPartyRegistrationList : List - The list of all PartyRegistrations.

Methods:

- getFullPartyRegistrationList( ) : list(PartyRegistration)
    - Gets all registered parties

3.2.2 - UserAccount

Summary:

- An account which holds the data for the user account, which also includes the variables which are listed below. User can register for a party and view the parties they are registered for.

Variables:

- PartyRegistrationList : List - The list of PartyRegistrations assosciated to that user.
- NotificationsList : List - The list of all notifications sent to the user from the administrator.

Methods:

- getPartyRegistrationList() : list(PartyRegistration)
    - Gets all of the parties associated with the specific user
- setPartyRegistrationList(parties : list(PartyRegistration)) : void
    - Update the party registration list associated with the user
- getNotifications( ) : list(Strings)
    - Gets all of the notifications that are available to the user

3.3 - Host

Summary:

- Contains the data associated with each host. The host object will store the user's ID if they have an account, or the host's name and phone number if not.

Variables:

- User ID : Integer - The UserAccount's ID of the host.
- Host Name : String - The name of the Host.
- Host Phone Number : PhoneNumberObject - The Host's phone number.

Methods:

- getUserID( ) : Integer
    - Gets the unique integer ID associated to the User
- setUserID(id: Integer) : void
    - Sets the unique integer ID associated to the User
- getHostName( ) : String
    - Gets the party's Host Name string
- setHostName(host : String) : void
    - Sets the party's Host Name string
- getHostPhoneNumber( ) : PhoneNumberObject
    - Gets the party Host's Phone Number
- setHostPhoneNumber(number : PhoneNumberObject) : void
    - Sets the party Host's Phone Number

4 - View

4.1 - MainView

Summary:

- The main view is a skeleton that all other views will be rendered with. The MainView will manage the views and decide which view to display to the user.

Variables:

- loginView : LoginView
- userMainView : UserMainView
- registerPartyView : RegisterPartyView
- adminMainView : AdminMainView
- adminMapView : AdminMapView
- adminAccount : AdminAccount
- userAccount : UserAccount

Methods:

- render(): void
    - Renders each possible view by calling the render method in the specific class for the page that the user is on.

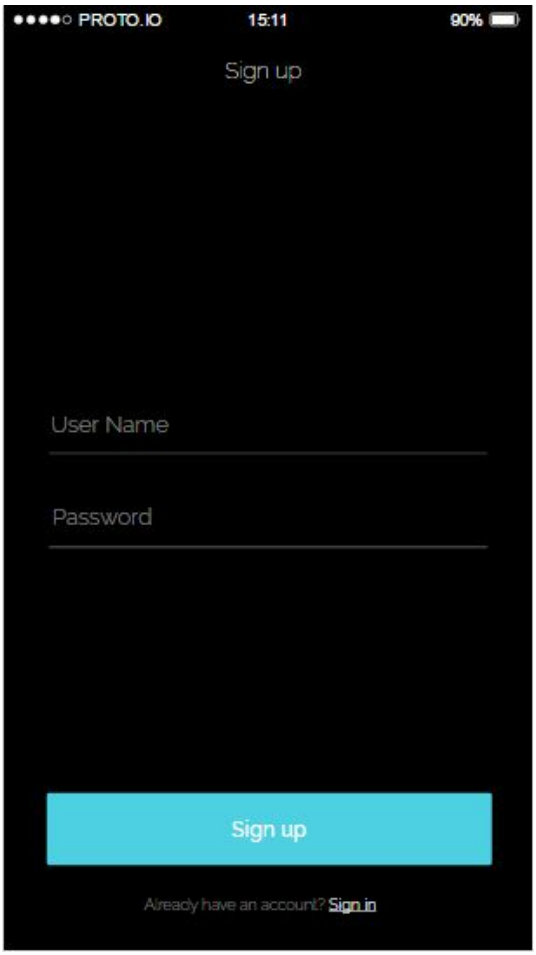
4.2 - Account Register View

Summary:

- This view allows the user to register a user account with our app if they have not already done so. From there the user is brought to the User Main View. The Username and Password fields allow the user to enter their Username and password when logging in. The signup button allows the user to submit their account information when they are done filling out the form. The sign in button allows the user to switch to the sign in view.

Methods:

- render(): void - renders the account register view on the screen



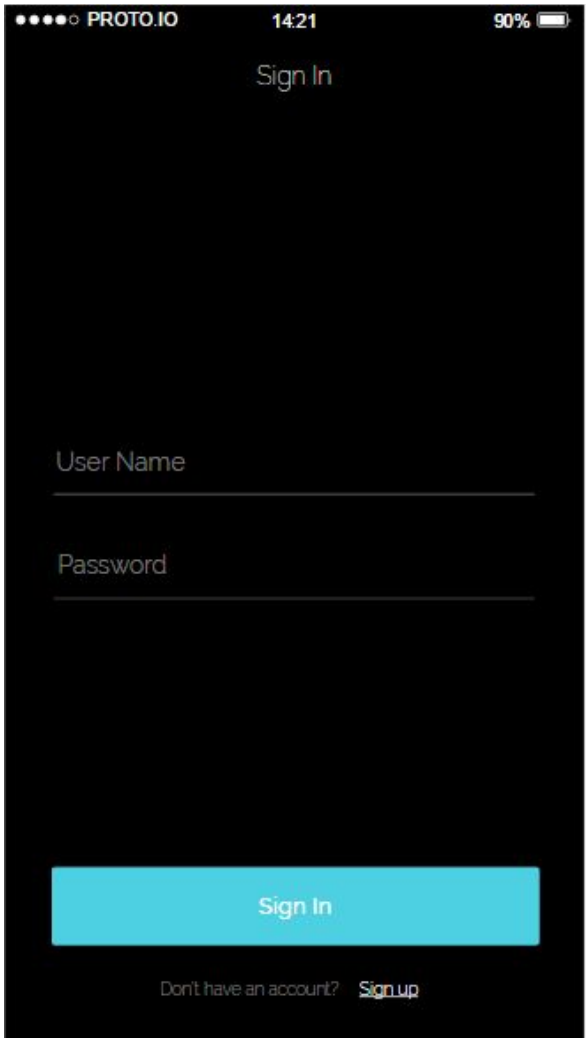
4.3 - Account Login View

Summary:

- This view allows the user to login if they have an existing account. The User enters in their unique username and their password. Each account is flagged as a user or an administrator. From there you are brought to your respective account interface. The view also allows you to enter the Account Register View through the signup link at the bottom.

Methods:

- render(): void - renders the account login view on the screen



4.4 - User Main View

Summary:

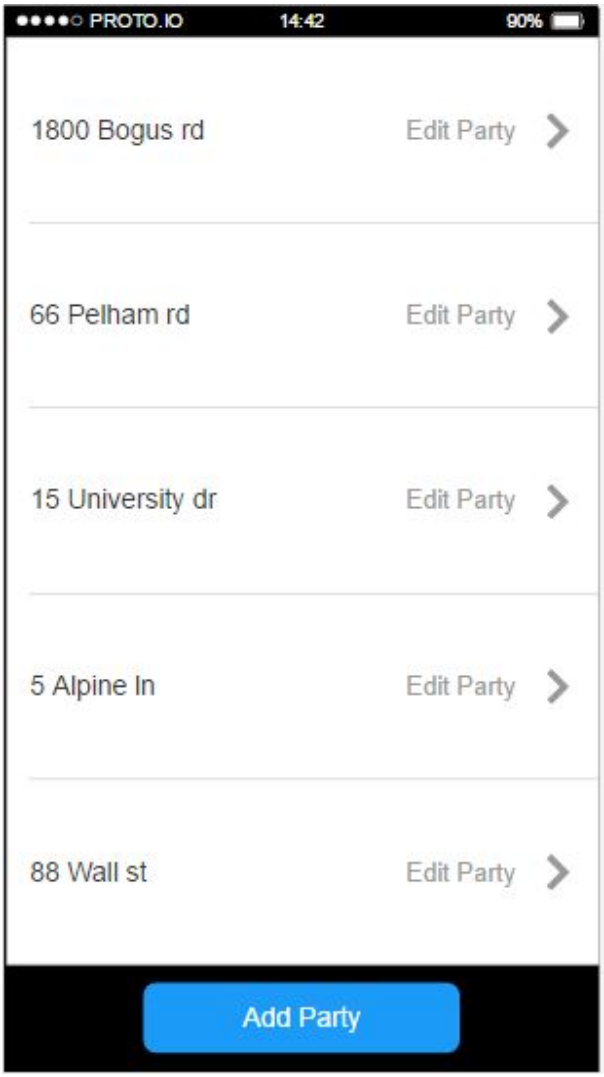
- This view lets users see a list containing the different addresses of the parties registered under their account. From there you can edit any of the existing parties or click the “Add Party” button on the bottom of the screen to add to the existing list

Variables:

- partyList : list(partyRegistration) - The list of parties

Methods:

- render(partyList) : void - renders the party list onto the screen



4.5 - Register Party View

Summary:

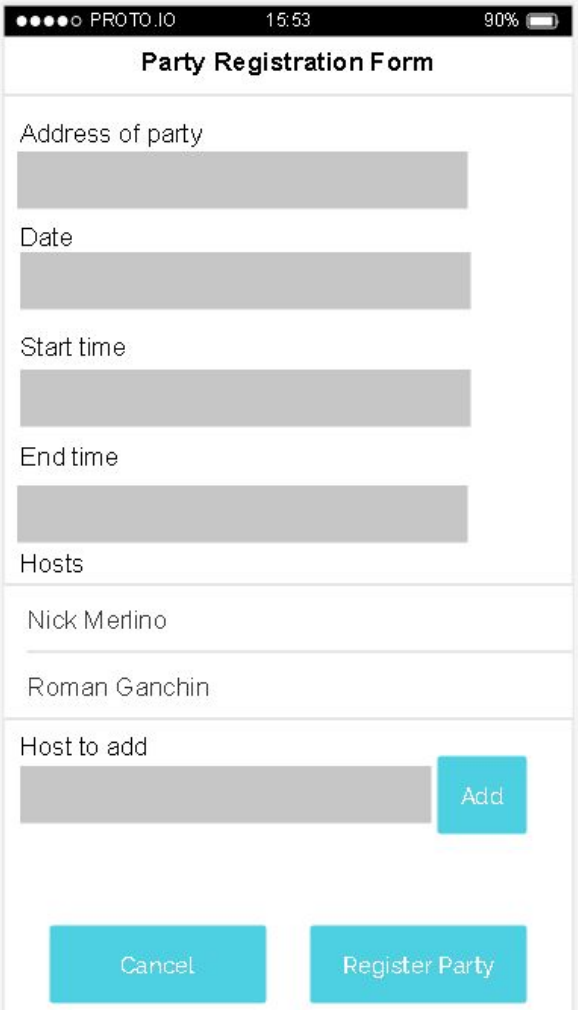
- This view lets the User create and register a party. The user enters in the relevant information to their party such as the address of party, data, start time, end time, and hosts. The user is able to add in multiple hosts for the administrators to contact if needed. The registration itself can be canceled with the cancel button on the bottom of the page or finalized with the register party button.

Variables:

- party : partyRegistration - the party registration data.

Methods:

- render(party) : void - renders the party registration data onto the screen.



4.6 - Administrator Main View

Summary:

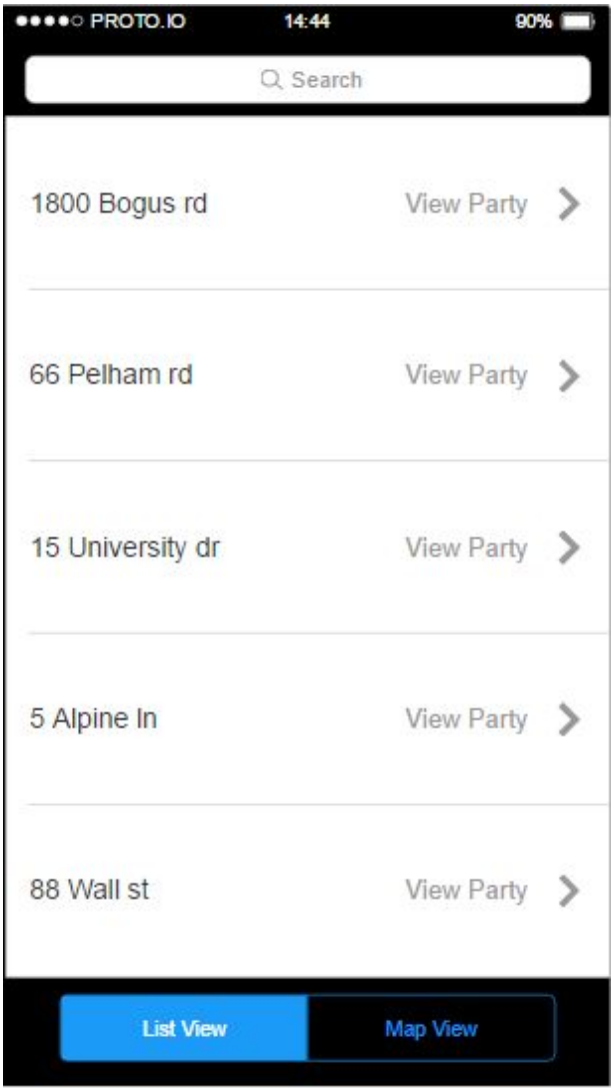
- This view lets the administrator see a list of all the registered parties. From this main view you can use the buttons at the bottom of the screen to switch from the list to a map view. Also the administrator is able to tap on each party individually to view the party's information and to able to send warning messages to the party's hosts.

Variables:

- fullPartyList : list(partyRegistration) - contains the list of registered parties.

Methods:

- render(fullPartyList) : void - renders the list of registered parties on the screen.



4.7 - Administrator Map View

Summary:

- This view lets the administrator see the pinpoints of all the registered parties on a map. On the bottom of the view, the administrator can switch to List View to see a list of the registered parties if desired. The administrator is also able to tap on each pin to bring up the party's information.

Variables:

- map : MapObject - contains the map.
- partyList : list(partyRegistration) - contains the list of registered parties.

Methods:

- render(map, partyList) : void - adds the registered parties to the map and renders the map on the screen.

5 - Controller

5.1 - PartyRegistrationController

Summary:

- The PartyRegistrationController allows for the manipulation of the PartyRegistration and will allow a user to edit, add, and delete his parties.

Variables:

- Party : PartyRegistration - A PartyRegistration object that will be modified.
- userID : Integer - The ID of the user making a party.

Methods:

- addParty(party : PartyRegistration, userID : Integer) : void

        - Allows a user to add a new party.
    - editParty(party : PartyRegistration, userID : Integer) : void
        - Allows a user to edit a party.
    - deleteParty(party : PartyRegistration, userID : Integer) : void
        - Allows a user to delete a party.

5.2 - AddAccountController

Summary:

- The AddAccountController will allow a new user to create an account in the database.

Variables:

- userAccount : UserAccount - The account to be added to the database.

Methods:

- addAccount(account: UserAccount) : void
    - Enables a user to create an account.

5.3 - NotifyUserController

Summary:

- The NotifyUserController is the toolset used by administrators to send notifications to general users about their parties.

Variables:

- Notification : String - Message string to be sent to users.
- User ID : Integer - User ID of message recipient.

Methods:

- addNotification(notification : String, userID: Integer) : void
    - Creates a new notification given the message string, and the recipients ID

6 - Testability Design

We would separate our app into three components which follow the MVC architecture. The three components are View, Model, and Controller. Multiple classes are separated into each of the three components. The components can all exist without the the others allowing for separation of concerns and better testability as you can test the model, view and controller separately. Furthermore within the model, view, and controller we separate everything even further which allows better testability as we can test small separate parts and then slowly aggregate them together once we know all of other separate components are working. We will look to use the Jasmine testing framework along with Karma which will serve as the runner for the tests. Both of these work together and are meant for Angular JS code. Karma is a direct product made by AngularJS due to the fact they struggled making efficient tests with the tools that were available at the time. Karma will provide us with greater flexibility in testing our code as it gives us the option to test out code on different browsers and devices which is perfect for our application. We will aim to test all of the functionality of the application like adding, deleting and editing parties. Also, things like logging in, the warning notification and registering accounts will be accounted for in the testing. We will look to cover all possibilities to make sure that the application runs to the best of it's ability.

7 - References

meteor.com
ionicframework.com
cordova.apache.org
Angularjs.org

8-Glossary

AngularJs - A structural framework for dynamic web apps. Lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly.

Cordova - A platform to build Native Mobile Applications using HTML5, CSS and Java Script

Controller - A software program that manages or directs the flow of data between two entities.

Model-View-Controller - A software architectural pattern for implementing user interfaces on computers. It divides a given application into three interconnected parts in order to separate internal representations of information from the ways that information is presented to and accepted from the user.

Ionic Framework - A complete open-source SDK for hybrid mobile app development.