Hometown Heroes

CS 361 - 400, Group 6, Homework 5

Aug 1, 2017

Jon Austin
developer
austinjo@oregonstate.edu

Valerie Chapple
developer
chapplev@oregonstate.edu

Kenny Lew
developer
lewke@oregonstate.edu

Charlotte Murphy
developer
murpchar@oregonstate.edu

Gregory Niebanck
developer
niebancg@oregonstate.edu

Benjamin Rodarte
client
rodartec@oregonstate.edu

1. INTRODUCTION

Hometown Heroes is a web-based app designed to increase volunteerism. Users can publish service opportunities, search for events, register for events, and participate in service events. A Hometown Hero is a user who volunteers his/her time supporting a Hometown Heroes service event.

Our goal is to write up a formal plan for the next two weeks' work using an Agile method. We will need to obtain several user stories from our customer. With those stories in hand, we will divide those stories into concrete tasks and estimate the amount of effort it will take to accomplish those tasks. If we are unsure how much effort it will take to perform a task, we will perform a spike to research and report back with a more educated estimate. After dividing the tasks into a logical order, we will be able to decide which tasks we will need to do next week and which tasks we can do the following week. We will also be able to decide which user stories will require an unreasonable amount of effort, then cancel or delay the implementation of those stories. This method will help us organize our activities for the next two weeks in order to accomplish as much as possible in an efficient manner.

2. USER STORIES

Our client produced 15 user stories that he envisioned being useful for *Hometown Heroes* (aka *HH*). Some of the user stories have many working parts, which could be broken down into additional user stories or tasks. We discussed these user stories with our client, coming up with a list of doable user stories, based on frontend and backend. Below is the preliminary list of user stories, including our thoughts about each of the user stories prior to sorting priorities.

2.1 User Story #1

As a first-time user, I can create a new account using my email address or using login credentials from social media sites such as FB/Twitter/IG.

This user story can be broken down into two separate user stories. That is, the first-time user can create a new account with an email address, and also the first-time user can create a new account using social media sites.

2.2 User Story #2

As a first-time user, I can fill out a profile which includes the types of volunteering events I am interested in and the types of skills I possess so that the most relevant opportunities can be suggested to me.

Completing a profile is very much related to user story #1. It takes a natural progression from account creation to account customization for the user.

2.3 User Story #3

As a first-time user, I can elect to take a short survey which allows me to identify skills I could include in my profile.

User story #3 is a specialization of user story #2. It is an alternative strategy for the user to complete the skills section of the profile.

2.4 User Story #4

As a user, I can edit my profile and update my preferences/skills so they remain relevant.

After the user has created an account and completed a profile, they may want to change their details. This editing interface for user story #4 can be the same interface that a user uses to create their profile. The only difference is how the user gets to the interface and that data will be pre-populated from their current profile.

2.5 User Story #5

As a user, I can see upcoming volunteering opportunities around me in a map view or list view so that I can find ones that I am interested in attending.

This is another user story that can be divided into sub-stories. That is, the user can view the events through a list view and the user can view the events through a map view. While the user will be able to complete similar actions in both views, the visual interaction is dramatically different.

2.6 User Story #6

As a user, I can register for an upcoming volunteering opportunity which automatically puts the event in my in-app calendar so that I remember to attend.

This user story contains multiple stories. That is, a user can register for upcoming events. The user can view registered events in a calendar view.

2.7 User Story #7

As a user, I can register for multiple upcoming events even if they conflict but I should be notified of the conflict so that I can attend multiple events for partial durations.

This story requires the usage of a trigger that checks the currently focused event times with the user's registered events. This story could be implemented with a number of notification methods, which would need to be clarified (push notifications, email, HH inbox, or social media).

2.8 User Story #8

As a user, I can unregister for an event either on the event view page where I registered in the first place or in my in-app calendar.

This user story can be implemented with a single interface call to the event to unregister a user. However, the function call will need to be used in two different visual places: the event

profile page and the event display page inside the calendar. This could produce two stories.

2.9 User Story #9

As a user, I can elect to have appointments on my in-app calendar pushed to another calendar (such as Google Calendar).

Pushing to a calendar can mean a few things; however, the easiest and most reliable method to push events to a Google Calendar would be to create an ICAL calendar feed for the user to inject into their Google account. This has advantages in that a user is able to view the event with their Google account, but they will not be able to delete events. This is important, given that it is preferable for the user to unregister for an event using the Hometown Hero app. Alternatives include using the calendar database through iOS or Android.

2.10 User Story #10

As a user, I can modify the default search settings and specify distance, skills, duration of events, etc. for the opportunities shown to me by default in map view or list view.

Customizing the search results is an extension of user story #5. User story #5 already filters events using the current user's distance and skills. User story #10 makes that feature editable and can gradually offer more filter settings.

2.11 User Story #11

As a user, I can submit a request to create a new event so that I can mobilize my local HH users for a worthy cause in our community.

Creating an event is one of our use cases we focused on for this project. Creating an event requires gathering enough information about the event for other users to be able to approve or deny the event for Hometown Heroes.

2.12 User Story #12

As an administrator, I can review submitted event creation requests and approve or deny them so that I can ensure only events which meet HH criteria are published.

Extending the process of user story #11, an administrator will need to approve an event in the early stages of the app. This user story requires a single admin to see the submitted event information as well as provide feedback on its status.

2.13 User Story #13

As a user hosting an event, I can begin and end the event on its specified date so that I can start early or end late if necessary.

This user story requires an event to exist, where only the organizer can start or end the event. It is limited to the specified dates on the event profile page.

2.14 User Story #14

As a user attending an event, I will be automatically checked in when I am in range of the host while the event is running so that I can earn HH points as I volunteer.

A user wants to be able to earn points for volunteering without the hassle of signing in. Interaction with the GPS on a phone is necessary. A native app is appropriate for this function.

2.15 User Story #15

As a user, I can opt to have HH post to my social media whenever I register for an event, increase in HH level, or create an event.

This user story can be divided into separate functions. That is since the user is opting into HH posting to social media, that can occur during the profile creation stage. The user can select their default settings for HH sharing. Another function of this story is to develop the information to include in the post and actually access the user's social media accounts.

3. PRIORITIZING USER STORIES

Upon discussing the user stories with our client, we chose to focus on splitting the user stories into frontend development and backend development. The user stories have been sorted by order of importance, starting with those stories due week 1, then stories due for week 2, and stories that will not be implemented. Below each user story is a list of tasks, not all tasks will be implemented in that week, as some user stories are quite large and could be broken into separate user stories. Time for each task is estimated in *units*, which are defined as one person-hour.

3.1 Stories Due Week 1

User Story #1

As a first-time user, I can create a new account using my email address or using login credentials from social media sites such as FB/Twitter/IG.

1. iOS spike (Time: 58-60 units, see iOS spike below)
2. Facebook spike (Time: 4.5 units)
    a. Setup basic Node.JS server
    b. Acquired documentation from developers.facebook.com
    c. Acquired documentation from passport middleware http://passportjs.org/docs/facebook
    d. Referenced youtube tutorial https://www.youtube.com/watch?v=OMcWgmkMpEE
    e. Amended example code from GitHub
    f. Tested working example
3. Twitter spike (Time: 5 units)
    a. Setup basic Node.JS server

    b. Acquired documentation from https://dev.twitter.com/web/sign-in/implementing
    c. Acquired documentation from passport middleware http://passportjs.org/docs/twitter
    d. Create Twitter prototype login
4. Full Stack - Account creation option page for Email, Facebook, Twitter (Time: 2 units)
    a. UI storyboard - 1 unit
    b. Express: Add routes for rendering email, Facebook, or Twitter login pages
    c. Handlebars: HTML buttons for Email, Facebook, or Twitter login
5. Frontend - Account creation form for email address with validation and submission (Time: 5 units)
    a. UI storyboard - 1 unit
    b. Handlebars:
        i. Email Login Page
        ii. Form validation with NPM validator.
6. Backend - API to create new account login info (email address) in database (Time: 6 units)
    a. Node JS server-side code
        i. Routes for login page
    b. MySQL database
        i. ER diagram
        ii. Table definitions for Account Information

3.2 Stories Due Week 2

User Story #2

As a first-time user, I can fill out a profile which includes the types of volunteering events I am interested in and the types of skills I possess so that the most relevant opportunities can be suggested to me.

1. Backend - API to read Interest Tags and Skill Tags (Time: 5 units)
2. Frontend - Form to collect user profile information, including Interest Tags and Skill Tags (Time: 6 units)
3. Backend - API to create new or update user profile (Time: 4 units)

User Story #4

As a user, I can edit my profile and update my preferences/skills so they remain relevant.

1. Frontend - Prepopulate profile creation form, including Interest and Skill Tags (Time: 6 units / 3 units if reusing code)
2. Backend - API to create new or update user profile (Time: 4 units)

3.3 Stories Not Implemented Within Two Weeks

User Story #3

As a first-time user, I can elect to take a short survey which allows me to identify skills I could include in my profile.

1. Backend - API to read skill survey questions (Time: 3 units)
2. Frontend - GUI to have user answer 20 skill survey questions, one at a time (Time: 6 units)
3. Frontend - Analysis package to tally appropriate results of survey (set of skill tags) (Time: 5 units)
4. Backend - API to update user profile (Time: 5 units)

User Story #5

As a user, I can see upcoming volunteering opportunities around me in a map view or list view so that I can find ones that I am interested in attending.

1. Frontend - Display table view of events (Time: 5 units)
2. Frontend - Create event detail view (Time: 4 units)
3. Backend - API to read events with given filter settings (Time: 5 units)
4. Backend - API to read event detail given event id (Time: 3 units)

User Story #6

As a user, I can register for an upcoming volunteering opportunity which automatically puts the event in my in-app calendar so that I remember to attend.

1. Frontend
    a. Create Registration Button on Event Page (Time: 2 units)
    b. Create Calendar View and Clickable dates (Time: 6 units)
2. Backend - API for registering for events (Time: 6 units)

User Story #7

As a user, I can register for multiple upcoming events even if they conflict but I should be notified of the conflict so that I can attend multiple events for partial durations.

1. Frontend
    a. Create Registration Button on Event Page (Time: 2 units)
    b. Create calendar View and Clickable dates (Time: 6 units)
2. Backend
    a. API for registering for events (Time: 5 units)
    b. API for collecting registered events (Time: 3 units)

User Story #8

As a user, I can unregister for an event either on the event view page where I registered in the first place or in my in-app calendar.

1. Frontend
    a. Toggle Register button with Unregister button on Event Detail Page (Time: 3 units)
    b. Create Calendar View of Events to direct to Event Detail Page (Time: 4 units)
2. Backend
    a. API to unregister for events (Time: 3 units)
    b. API for collecting registered events to populate calendar view (Time: 3 units)

User Story #9

As a user, I can elect to have appointments on my in-app calendar pushed to another calendar (such as Google Calendar).

1. Frontend
    a. Create UI button in settings/profile to supply ICAL URL for user (Time: 2 units)
2. Backend
    a. Create ICAL feed for users to follow events in their native calendars (Time: Spike needed)

User Story #10

As a user, I can modify the default search settings and specify distance, skills, duration of events, etc. for the opportunities shown to me by default in map view or list view.

1. Frontend
    a. Create filter button to load filter settings view (Time: 2 units)
    b. Create filters view and form to select settings (Time: 5 units)
    c. Create ListView (Time: 3 units)
    d. Create MapView (Time: 6 units)
2. Backend
    a. API to query event database with a given event filter (Time: 6 units)

User Story #11

As a user, I can submit a request to create a new event so that I can mobilize my local HH users for a worthy cause in our community.

1. Frontend - Forms to collect event name, date(s), tags, and location information (Time: 8 units)
2. Backend - API to create new event (Time: 4 units)

User Story #12

As an administrator, I can review submitted event creation requests and approve or deny them so that I can ensure only events which meet HH criteria are published.

1. Frontend
    a. Create separate page to display list pending events (Time: 3 units)
    b. Create detail view of pending events with approval form attached (Time: 3 units)
2. Backend
    a. API to call all pending events for a user/admin (Time: 5 units)
    b. API to call single pending event based on id (Time: 1 unit)
    c. Route to process approval form and change event status if necessary (Time: 3 units)

User Story #13

As a user hosting an event, I can begin and end the event on its specified date so that I can start early or end late if necessary.

1. Frontend
    a. Add Run Events section to show an organizer's events for that day (Time: 4 units)
    b. Add detail view of event with button to start or stop event (Time: 2 units)
2. Backend
    a. API to query an organizer's available events that day (Time: 1 unit)
    b. API to update event start and stop time logs (Time: 4 units

User Story #14

As a user attending an event, I will be automatically checked in when I am in range of the host while the event is running so that I can earn HH points as I volunteer.

1. Frontend
    a. Check that app is available for location services (Time: Spike Needed)
    b. Complete Push notification to turn on location services for closed apps (Time: Spike Needed)
    c. Query Location services on the minute and check against event range (Time: Spike Needed)
2. Backend
    a. API to trigger location services on volunteer devices after event starts. (Time: 4 units)
    b. API to update user timestamps of being within the event range (Time: 4 units)

User Story #15

As a user, I can opt to have HH post to my social media whenever I register for an event, increase in HH level, or create an event.

1. Frontend
    a. Create section in settings where users can checkmark what types of posts HH can complete (Time: 3 units)

b. After registration, trigger social media post (Time: 4 units)
c. After HH level increase, trigger social media post (Time: 4 units)
d. After event approval, trigger social media post (Time: 4 units)
e. Create reauthorization page for each social media, if needed. (Time: 5 units)

2. Backend
   a. API to store access tokens for various social media (Time: 2 units)
   b. API to read access tokens for post authorization (Time: 2 units)

4. WEEK 1 PREPARATIONS

4.1 User Story #1: iOS Spike

Since none of us have any iOS development experience, we performed a spike on the amount of effort it would take to implement this story in iOS. We are assuming that the backend work will be done to be interoperable with both a web app and iOS client, so the spike focuses mainly on the development of the client itself. Development activities include:

1. Setting up the Xcode development environment
   a. On macOS: 2 units, non-parallel, for each developer
   b. On PC, including installing and configuring VirtualBox: 4 units, non-parallel, for each developer
2. Language Familiarization: 16 units, non-parallel, for each developer
3. Basic UI Implementation: 8 units
4. Database I/O: 8 units
5. Social Networking Integration: 8 units
6. Unit Testing: 16 units

Since development environment and language familiarization cannot be done in parallel, it would take both developers at least half of the week before they would even start coding. Since we have a week to implement at least two user stories, the risk involved to try and develop an iOS app is unacceptable. After talking to our customer about this, he suggested that we focus on a web only version, then add iOS functionality later.

Adding Facebook to an iOS app can be done using the Facebook SDK for Swift. The Facebook SDK for Swift allows us to program Facebook Login and enable sharing of content. However, after conducting the iOS spike, we determined that a web app will be the most appropriate place to start our project.

4.2 User Story #1: Facebook, Passport, and Node.js Spike

Intro

We amended our first user story to emulate the mobile account creation using a web application using node JS with CSS to model a mobile interface. Much of the vision for HH requires social media integration to promote the usage and circulation. In addition to semi- automated posts (Facebook does not allow applications to make fully automated posts), we wanted to lower the effort for account creation by allowing new users to create an HH account using their Facebook, Google+, or Twitter credentials. We wanted to know how much effort this element of social media integration would require so that we could better estimate the unit time required for this user story. We decided to do some research, and implement a spike that utilizes Facebook login credentials to create an account on the HH site.

Setup

The first step was to set up a working web server using Node.JS as a skeleton. This site uses Node.js with Express. The server contains two routes. The first route is a GET route used to render an HTML page consisting of a simple form with fields for name, password, and a submit button. A custom JavaScript event handler will simplify AJAX requests, which will be directed to the second route on the server. A POST route allows the client to send JSON to the server with corresponding attributes for username and password.

App Registration

To utilize this login feature, developers must first register their app with Facebook. The process was short, it required creating a developer account, giving some basic information about the application platform and some contact information. The Facebook for developers site creates an application page where settings for the application can be viewed and changed. Among these settings are an application ID and an application Secret. The app ID and secret are required by the server-code middleware to implement login functionality.

Passport

"Passport is authentication middleware for Node.js. Extremely flexible and modular, Passport can be unobtrusively dropped into any Express-based web application. A comprehensive set of strategies supports authentication using a username and password, Facebook Twitter, and more." This middleware seemed like an obvious choice to get practice with considering that any knowledge gained would be usable when implanting later stories for other social media. The Facebook module for this middleware can be installed using 'npm install passport-facebook'. Documentation is available for this module at http://passportjs.org/docs/facebook. [2]

Adapted Code Samples [1] [2]

```
//Importing the passport module
var passport = require('passport');
var Strategy = require('passport-
facebook').Strategy;
//The Facebook Strategy
// credentials export file
module.exports = {
    'facebookAuth' : {
         d'clientID': '(insert from
         developers.facebook settings page)',
         'clientSecret': '(insert from
         developers.facebook settings page)',
        'callbackURL':
         'http://flip2.engr.oregonstate.edu:16661/lo
         gin/facebook/callback' }
     }
passport.use(new FacebookStrategy(
{ clientID: configAuth.facebookAuth.clientID,
 clientSecret: configAuth.facebookAuth.clientSecret,
 callbackURL: configAuth.facebookAuth.callbackURL
}, function(accessToken, refreshToken, profile,done)
   { User.findOrCreate({ facebookId: profile.id },
      function(err, user) {
        if(err) { return done(err);}
        done(null, user);
      });
    }
));
//The user object returned from successful
authentication
//contains the user's public facebook info
{ Profile:
        Provider: String, facebook twitter, etc.
        Id: (String) this is the client token given
from the provider
        Name:{
        FamilyName: string,
                givenName: string,
                middleName:string}
        Emails {Array}[n]
                Value:string(the email address
                Type:string (home work etc)
        Photos{Array} [n]
                Value: String URL for images
}
//The server routes
// route for login screen
app.get('/login',function(req,res)
{
    res.render('login_screen');
});
// route after successful facebook login
app.get('/success',function(req,res)
{
       res.render('success');
});
// route for failed facebook login
app.get('/fail',function(req,res)
{
    res.render('fail');
});
app.post('/login', function(req,res,next)
{
       var data_object = req.body.data;
       console.log(data_object);
       res.send(data_object);
});
// at this point passport & Facebook does the work.
app.get('/login/facebook',
passport.authenticate('facebook'));
app.get('/login/facebook/callback',
passport.authenticate('facebook',{
successRedirect: '/success',
    failureRedirect: '/fail' })
);
```

Conclusion

This type of web server functionality is highly used and very well documented. Implementing this functionality in our project would take considerable effort for the first-time installation, but once implemented would become a common routine for setting up web applications. Considering our group has no prior experience with this middleware, I estimate the back end for this segment of the story at 4.5 hours for a programmer pair.

4.3 User Story #1: Twitter Spike

Intro

Adding Twitter to an iOS app can be done using the Twitter Kit for iOS. However, we decided after the iOS spike that for maximum usage of our time, we will be working on a web app. Therefore, there are some tasks to be done to implement Twitter using JavaScript. Most of this setup and implementation is similar to the Facebook login.

Setup & Passport

As in the Facebook Spike, the first step is to set up a working web server using Node.JS and Express. The server-side requires two routes: a GET route that renders an HTML page consisting of a simple form with fields for name, password, and a submit button and a corresponding POST route to accept the username and password.

Passport is a node module that operates as middleware to easily transport authentication information. The node module for Twitter using Passport is can be installed using *npm install passport-twitter*. Documentation is available at http://passportjs.org/docs/twitter [3].

App Registration

To utilize this login feature, developers must first register their app with Twitter. The process requires a developer to register a Twitter account and the application at the website: https://apps.twitter.com/ [4] . The app secret must remain

confidential. To do this, it must be stored on the server, preferably encoded.

Configuration

Once the Node server is set up and Passport installed, the Passport module must be configured to accept sign in requests and redirect users. To configure Passport, select the Twitter strategy from the module, and program passport to use the Twitter strategy with the app's key and secret that is securely stored on the server. Lastly, configure the server to have two additional routes: an authenticate route and a callback route for after authentication. A simple Twitter button can be installed that links to the authentication route.

If not using the Passport-Twitter node module, additional research of the Twitter documentation is required for sign-in implementation and can be read at https://dev.twitter.com/web/sign-in/implementing [5].

Conclusion

Twitter login can easily be achieved through the server by using Node.js and passportjs. Additionally, as social media options are added to Hometown Hero, the organization of the server code will need to be modularized and the database will need to store the *user_id* that is supplied from Twitter to identify the user signing into Hometown Heroes.

4.4 User Story #1: Facebook or Twitter-Based Account Creation UML Sequence Diagram

For implementing user sign-in using Facebook or Twitter, the user needs to authenticate their login credentials. Then, we will request an access token from the social media service. Once access is granted, the server will create a new user account record in the database. Once the account is created, it sends a new account confirmation email to the user.

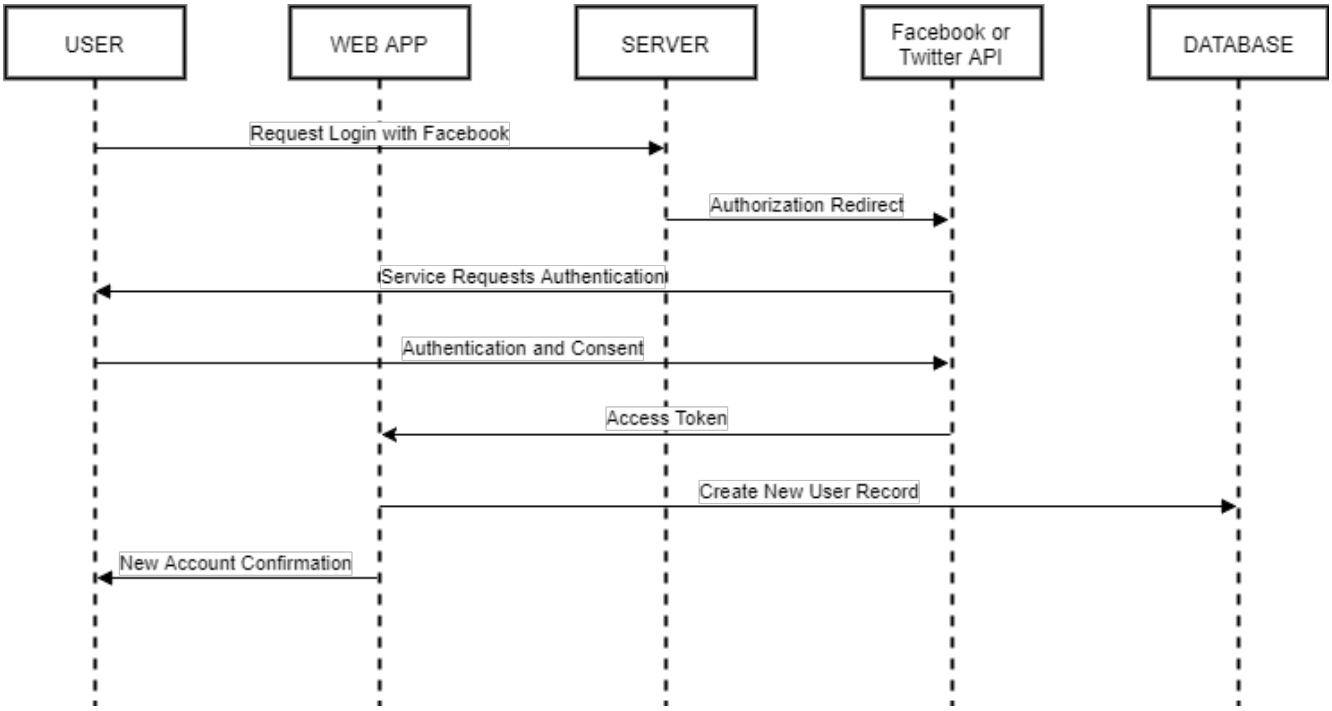

Figure 1: Facebook or Twitter-Based Account Creation UML Sequence Diagram [6]

4.5 User Story #1: Email-Based Account Creation UML Sequence Diagram

For implementing user sign-in using email, we will need to securely get the email and password for the user. The server will need to validate the information and possibly re-render the login screen with validation errors. Once the email and password are valid entries, the server will store the account information in the database and render a page that confirms account creation, such as a custom homepage for that user.

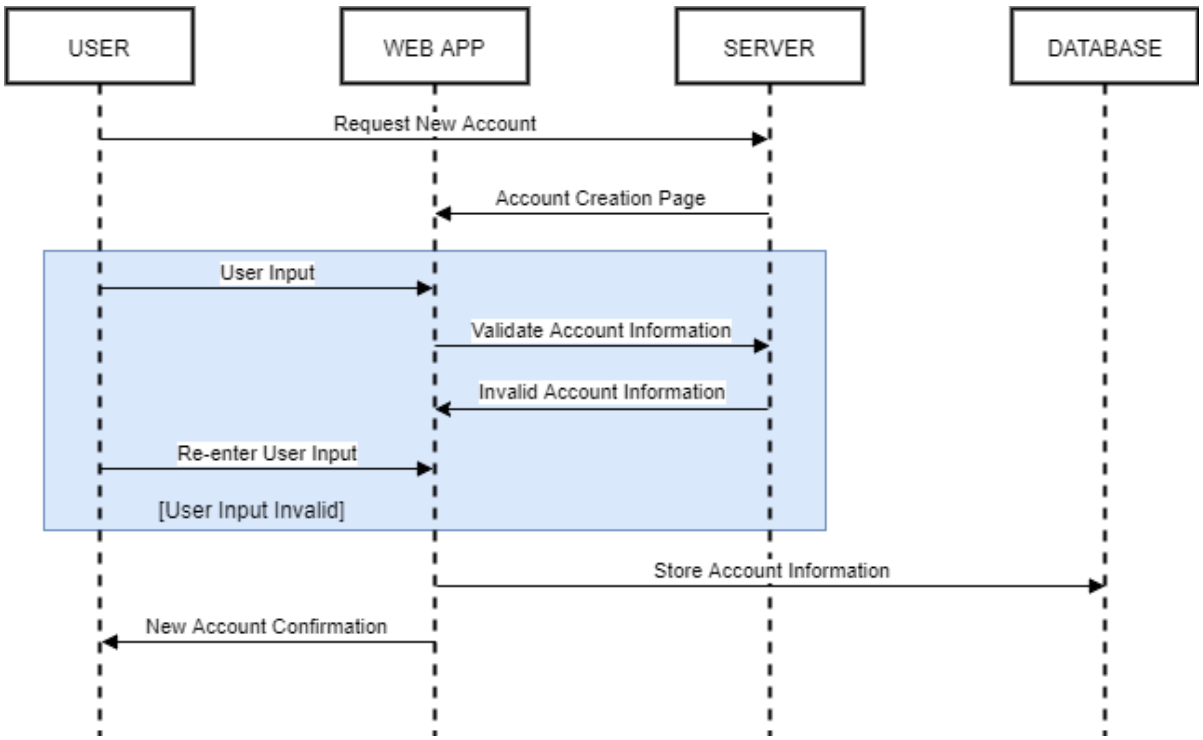

Figure 2: Email-Based Account Creation UML Sequence Diagram

4.6 User Story #1: Database to Store New Account Information

All user information will be stored in a database where data can be added or retrieved by the server. The database will store the new account login information into a table that holds all login information for all other users. For those users who create an account via email, the database will store their password. Additionally, the user may create an account by linking their desired social media service. In this case, the database links the social media access credentials to the user.

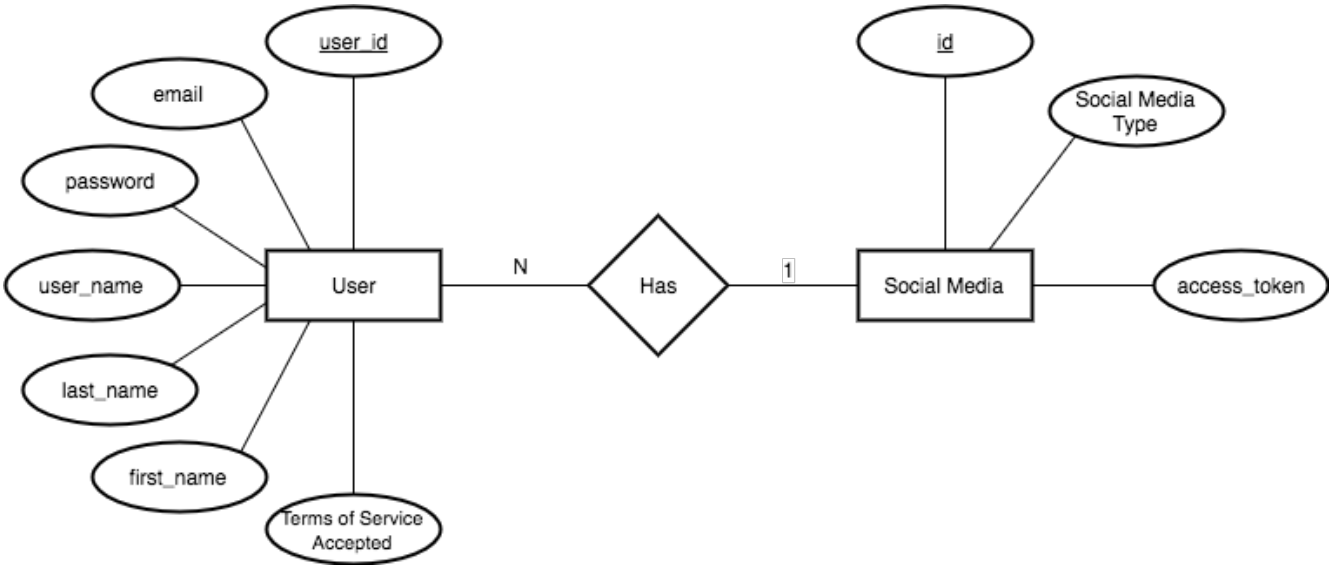

Figure 3: Account Information Entity-Relationship Diagram

4.7 User Story #1: Routes Server Must Handle

User Views

GET	/	Returns home page based on if a user is logged in
GET	/signup	Return Page of Options for Account Creation 1. Email - button links to /signup/email 2. Facebook - button links to /auth/facebook 3. Twitter - button links to /auth/Twitter
GET	/signup/email	Returns Account Creation Page using email address
GET	/profile	*Accessible after account creation or verification.* Returns Profile Editing Page (empty or pre-populated)
GET	/login	Return Page of Options for Account Login. 1. Email - Form embedded in this page 2. Facebook - Button to page /auth/facebook 3. Twitter - Button to page /auth/Twitter

API Routes

GET	/login/facebook	Redirects client to Facebook for authentication (done by passport-facebook)
GET	/login/twitter	Redirects client to Twitter for authentication (done by passport-twitter)
GET	/login/facebook/callback	URL that Facebook redirects to after authentication. The callback in Passport redirects based on successful login or not: 1. successRedirect: / 2. failureRedirect: /login
GET	/login/twitter/callback	URL that Twitter redirects to after authentication. The callback in Passport redirects based on successful login or not: 3. successRedirect: / 4. failureRedirect: /login

5. WEEK 1 TASK ASSIGNMENTS

Jon Austin

1. Frontend Account Creation

Valerie Chapple

1. Node.js full stack for Twitter Login
2. Frontend/UI focus

Kenny Lew

1. Backend Account Creation

Gregory Niebanck

1. Account Signup Page presenting options to log in
2. Node.js full stack for Facebook Login

Charlotte Murphy

1. Create table definitions for MySQL database
2. Backend/data focus

6. CONCLUSION

After reviewing, prioritizing, and analyzing the user stories submitted by the client, we determined that we will implement user story #1, Account Creation, this week and user stories #2 and 4, Create Profile and Edit Profile, next week. We gave user story #4, Edit Profile, precedence over user story #3, Complete Profile Questionnaire, because editing profile creates additional functionality rather than just providing another method of creating a profile. The remaining twelve user stories will not be implemented within the two weeks based on the amount of effort required to complete the remaining stories.

After completing an iOS spike, we determined that the effort involved in getting the entire team up to speed on iOS was not

reasonable for the proposed timeframe. The client agreed that we should develop a web-based version of the app instead.

The spike results for implementing account creation via Facebook and Twitter login determined that the workload required to implement those features is reasonable, and we will complete those tasks in week 1.

Further analysis of user stories #1, 2, and 4 allowed us to detail specific requirements for each implementation and then divide those tasks among the team members. We will work in two sub-groups, frontend focused and backend focused.

7. CLIENT AVAILABILITY

Our client was able to produce user stories and meet with us to discuss those stories on Sunday, July 30.

8. TEAM CONTRIBUTIONS

Jon Austin

1. Attended group conference Sunday, July 30
2. Introduction
3. iOS Development Spike

Valerie Chapple

1. Attended group conference Sunday, July 30
2. Annotate User Stories section
3. Collaborated on Prioritizing User Stories
4. Twitter Login Spike

Kenny Lew

1. Attended group conference Sunday, July 30
2. Database to Store New Account Information

Gregory Niebanck

1. Attended group conference Sunday, July 30
2. Conducted Spike: using passport middleware to use Facebook authentication as HH log in credentials
3. Assisted in user story decomposition

Charlotte Murphy

1. Facebook or Twitter-Based Account Creation UML sequence diagram
2. Collaborated on Email-Based Account Creation UML sequence diagram
3. Conclusion

9. REFERENCES

1. Facebook for Developers. Account Kit for Web (Javascript). Retrieved from https://developers.facebook.com/docs/accountkit/webjs
2. Jared Hanson. Passport Middleware Documentation. Retrieved from http://passportjs.org/
3. Jared Hanson. Passport Twitter Middleware Documentation. Retrieved from http://passportjs.org/docs/twitter
4. Twitter. Twitter Application Management. Retrieved from https://apps.twitter.com/
5. Twitter. Implementing Sign in with Twitter. Retrieved from https://dev.twitter.com/web/sign-in/implementing
6. Nick Pinheiro. Facebook Login. 2015. Retrieved from https://blogs.msdn.microsoft.com/nickpinheiro/2015/02/28/facebook-login-with-asp-net-web-forms/
7. ACM Word Template for SIG Site. Retrieved from https://www.sigapp.org/sac/sac2018/authorkit/2018_SAC_Word_Template.pdf
8. ACM. Citation Style and Reference Formats. Retrieved from https://www.acm.org/publications/authors/reference-formatting