Hometown Heroes

CS 361 - 400, Group 6, Homework 7
Aug 16, 2017

Jon Austin
developer
austinjo@oregonstate.edu

Charlotte Murphy
developer
murpchar@oregonstate.edu

Valerie Chapple
developer
chapplev@oregonstate.edu

Gregory Niebanck
developer
niebancg@oregonstate.edu

Kenny Lew
developer
lewke@oregonstate.edu

Benjamin Rodarte
client
rodartec@oregonstate.edu

ABSTRACT

In this paper, we describe the process of completing the assigned user story for this week, and our final status.

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.

The purpose of this document is to describe the second week of the implementation of our system. This includes a link to our software and a description of how to install and use it. We will also describe the work we did on the user story we assigned ourselves this week, our interaction with spikes, refactoring, integration testing, and our interaction with our customer. We will also provide a burndown diagram of tasks completed.

2. SOFTWARE REPO AND INSTALLATION

Our team used a GitHub repo for collaboration. It can be found here: https://github.com/vchapple17/hometown-heroes.git. To run the software, users must have Node and mySQL installed on the server. After the repo is cloned to the appropriate machine, users are to navigate via command line to the folder called `server`. The user must set their database connection parameters in `dbcon.js`, which is in the folder called `db`. The file `hometown_heroes_data.sql` must be imported into the user's database to create the necessary tables. In the `server` folder, the user must run `npm install` to download dependency packages. After dependencies are installed, the user starts the server by typing `node ServerCode.js`. To view the main page, open a web browser and go to http://localhost:16661/.

3. USER STORIES

Our work in week two included finishing integration for modules from user story 1, and programming user stories 2 and 4, which include user account profile setup and editing. User story #2 and #4 are closely related to each other. Since both of these user stories were being implemented this week, we chose to combine tasks that would be useful for both user stories.

4. 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.

4.1 Pair Programming

Programming pair for the rest of User Story #1 was Valerie and Greg. Their goal was to finish the backend database interface required to create and retrieve basic social media user account information as it relates to the Hometown Hero user information.

4.2 Initial Status and Final Goal

At the end of last week, User Story #1 included a basic route structure for a user to navigate to a signup page. Within that page, a user could sign up for an account with a username or password, or utilize Facebook or Twitter social media authorizations. The front end development and operation of these tasks were completed, along with a successful backend setup of recording a user's social media ID within the browser's session. The implementation of the authorizations utilized a stub for a database call.

To finish User Story #1, we must implement an interface for the Hometown Heroes app to effectively communicate with the database, which will hold all account information. Specifically, we will be writing unit tests for actual database queries involved in account creation when logging in with Facebook and Twitter for the first time. These queries must also account for future login sessions of returning users. Lastly, we must test the integration of the database queries

into the previously tested sections of the server code, making adjustments as needed.

4.3 Unit Tests

Previously, we implemented the authentication process using PassportJS returns a profile object containing various attributes, such as the social media provider, social-media-issued user ID (profile ID), the user's name, and other provider specific information. Information is available on the PassportJS documentation (link), as well as looking at the source code of the two strategies. The user information is first queried through the user's profile ID. If the user already has an account, it will need to be loaded. If there is not account for that social media user, a new account will be created.

For the newly implemented interface to the database, there were a couple parts to test. First of all, the actual MySQL queries to the database were relatively simple, as there were no complex queries to design. In a relational table, the user's profile ID is stored with the name of the social media provider and a reference to the Hometown Hero user ID. The automated unit test program passes a hard-coded profile object to the database interface function along with a generic callback function which displays the queried entity from the hh_user table. The Facebook and Twitter database modules check for previously existing social media users and determine whether to create a new Hometown Hero user. If a new Hometown Hero user is required, the interface creates a new Hometown Hero user in the database. Then, the user's Hometown Hero ID and profile ID is added to the relationship table along with the name of the social media provider. The Facebook version also sets the username attribute to the displayName attribute from the profile object. This feature was not tested in the Twitter module. Testing the database interface in isolation went smoothly.

4.4 Problems

While working on the rest of User Story #1, we discovered a few issues with our database schema. That is, we had placed an attribute called access_token, which we thought was necessary to store per user. At this point in time, that is not necessary. With this attribute no longer needed, we discussed that a future refactoring of our code could include the removal of the relationship table between the social media provider table and the user table (as well as the removal social media provider table). This could be replaced by adding attributes to the Hometown Hero user table since there are so few social media providers we will be using.

We also encountered issues with merging code with other programming pairs. That is, express provides the ability to add middleware, which makes for a very flexible framework. However, because there were three different configurations of PassportJS strategies, some errors occurred while we added the Facebook and Twitter interfaces. In addition, these issues with integrating all the different Passport strategies into the same code required a fair amount of research, which detracted from meeting our goals this week.

4.5 Time Required

Below are the tasks and time required for User Story #1. The times listed are the actual time required to implement.
Note: 1 Unit of Time = 1 Programmer Hour

1. 2 units test programming
2. 3 units Facebook database interactions programming and passing unit tests.
3. 4 units Twitter database interactions programming and passing unit tests.
4. 3 units hand tracing and researching passport.js and express sessions for integration testing.

4.6 Current Status

Facebook database interactions are working in isolation but not yet integrated into the working server branch. The Twitter database interactions are function in the server. The rest of the app will rely on the Hometown Hero user ID to access user specific information.

4.7 Remaining Tasks

The Facebook database interactions need to be successfully integrated with the working server branch. The database interface concerning Twitter is currently working on the master branch. However, the Twitter module relies on having three tables to represent the relationship between a social media provider and a Hometown Hero User. If the integration of the Facebook database requires changing the Hometown Hero user table to include social media attributes, then the Twitter interface will need to be updated as well.

5. 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.

5.1 Pair Programming

Jon and Kenny worked on the frontend of the profile forms. Valerie, Charlotte, and Greg worked on creating an interface for accessing and modifying user data, Interest Tags, and Skill Tags on the database.

5.2 Unit Tests

After a user has created an account, they should be presented with their account profile page. This page will have a form which the user can input their account profile information as well as skills and interests. To test that this works, we first tested that the form loads correctly by verifying that the form input boxes/check boxes display on the page and function as intended. We then check that the interest and skill tags appear correctly in a checkbox group. Once we ensured that the form inputs work, we prevent incorrect input by adding validation functionality and testing that it works appropriately, depending on the input form. Now that the form works, we test that the form submits the input values correctly via POST by outputting the sent data to console upon receiving at the server. Finally, we test that the data is inserted into the database correctly by the server by pulling the data back from the server to display on the account information page.

5.3 Problems

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. With that, most tasks regarding the profile UI went smoothly. We had a little difficulty making the profile fit into the iPhone mockup we have on our web page, but the problem was resolved after talking to the template programmer who was not in our pair. There are also obvious UI improvements to be made, but these would be addressed after the system implementation was functional. Problems occurred with the completion of backend implementation due to time.

5.4 Actual Time Required

Below are the tasks and time required for User Story #2. The times listed are the actual time required to implement.
Note: 1 Unit of Time = 1 Programmer Hour

1. Backend - API to read Interest Tags and Skill Tags (Not implemented)
2. Frontend - Form to collect user profile information, including Interest and Skill Tags (Time: 6 units)
3. Backend - API to update user profile (Time: 2 units, not fully implemented)

5.4 Current Status

User Story #2 required both the completion of frontend and backend implementation, which did not fully occur. The frontend aspect of a user seeing a form with appropriate inputs was successful. Users can input their first and last name, mobile phone number, address, interests and skills. However, none of this information was implemented with the database. That is, hard-coded information was used to list interests and skills for the user.

5.5 Remaining Tasks

User Story #2 still requires implementation with the database. There are a few things that must get done. First, the profile edit form must pull Interest and skill tags from the database. Therefore there needs to be an interface to fetch those tags, once as interests and again as skills. Once the database interface has been tested, the server must utilize that information to build the form with the correct interests and tags for the user.

The other aspect that needs to occur is the actual persistence of the profile onto the database. Again, a backend interface needs to be created to update the details of the user's profile. This would build upon the interface started with account creation, as it will be modifying the same model: User.

6. 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.

6.1 Unit Tests

Relevant tests for the update profile screen include verifying that the values are pre-populated with the User's current information. It is also important to verify the appearance of the interest and skill tags when more than one is selected at a time. Once the pre-populated data and interest and skill tags are verified, testing the functionality of the form input boxes and tag check boxes is the next step.

Then, after data, appearance, and functionality are verified, the next unit test should validate user input to prevent incorrect data stored to the User's profile. And the final unit test verifies that form input indeed updates the User's profile in the database through a POST.

6.2 Problems

Most tasks regarding updating the profile UI went smoothly. Much of the code from the account profile creation was re-used for updating the profile. Interest and Skill Tags have not been implemented because they reside in a separate table from the other user data, which makes pulling in their data a little more complicated. Therefore, they were not able to be implemented this week.

6.2 Actual Time Required

Below are the tasks and time required for User Story #4. The times listed are the actual time required to implement.
Note: 1 Unit of Time = 1 Programmer Hour

1. Frontend - Prepopulate profile creation form, including Interest and Skill Tags (Time: 3 units, not fully implemented)
2. Backend - API to create new or update user profile (Time: 4 units, not fully implemented)

6.3 Current Status

The account profile form is prepopulated with the user's personal information, such as mobile, email, address, etc, but it does not pre-populate the form with user's Interest and Skill tags. The API updates the user's personal information, such as mobile, email, address, etc, but it does not update the user's Interest and Skill tags.

6.4 Remaining Tasks

The user's skill and interest tags need to be prepopulated in the profile form, and the profile form needs to update skill and interest tags based on user input from the update profile screen.

7. SPIKE AND UML DIAGRAM DISCUSSION

We utilized the Twitter and Facebook spikes and diagrams from last week. However those spikes were updated, as we needed to know how long it would take to integrate different passport strategies into the same system. This took longer than expected due to the various configurations possible with middleware in ExpressJS.

We also encountered an issue with our structure of our database. That is, we originally planned to have a separate table represent the relationship between a User entity and a Social Media entity. This is shown in an ER diagram below.

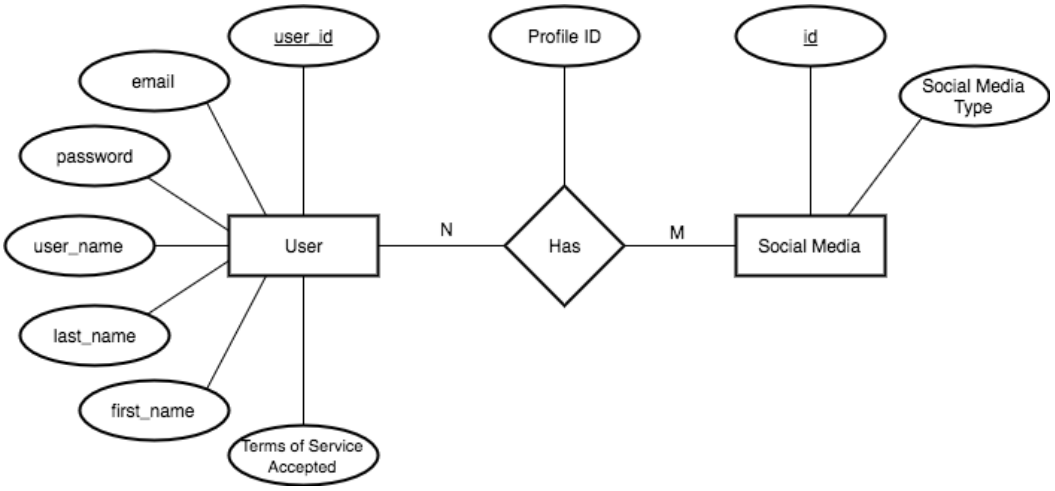

Figure 1. ER Diagram

However, after encountering a few problems with saving the social media information to the database, we discussed the possibility of changing the social media provider from being an entity to being an attribute of the user. At this point, that is, we do not see the merit for creating a social media entity that simply has a name. Below represents the ER diagram of this possible restructuring.

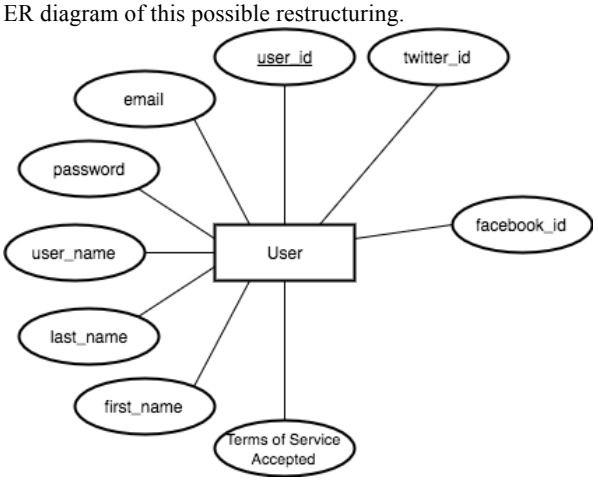

Figure 2. Proposed ER Diagram Update

In addition to the above ER diagrams (instead of UML diagrams), we also utilized the table of routes created from Homework 5 to ensure our team was on the same page. Knowing what we know now about PassportJS, it would be worthy to document the data flow necessary within each social media strategies. Our old documents follow data as it flows through our whole app in general, but those documents leave out a lot of inner details on how to structure smaller modules. Lastly, a state diagram in the future might be necessary, as we have incorporated not only a database, but a user sessions, which can easily change but may not persist without the correct interface. A more detailed dataflow diagram in addition to a state diagram will be helpful in the future.

8. BURNDOWN DIAGRAM

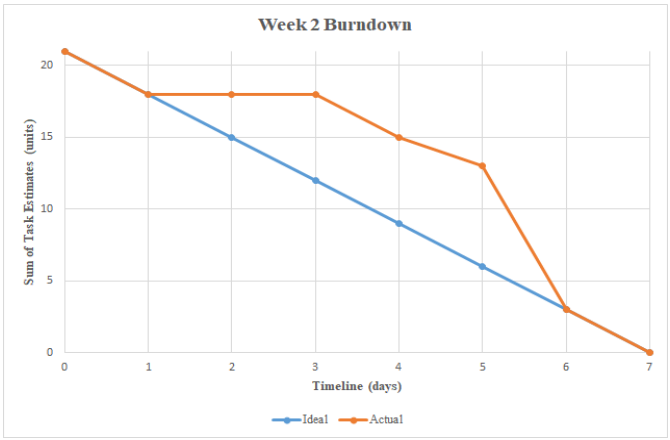

Figure 3. Burndown Diagram

9. REFACTORING

Our group completed some smaller refactoring when renaming pages to better represent their purpose, as well as renaming variables to improve readability. Future refactoring needs to include clear definitions of functions, such as the exact output of the Twitter and Facebook interface functions. That is, are those functions returning an ID, a JSON object, or an array of objects.

One refactoring that occurred after the interface to save a user's Twitter login was successfully implemented. This refactoring did not change the function of the interface; however, the interface was modularized into key functions that were needed for obtaining a user's information instead of one large function.

Further refactoring is possible in the future with this same file, as it is too long. Factoring can occur by separating the Twitter specific functions, such as *findByTwitterUserId*(), from the User interface. Additionally some reorganization could occur to create a base class for social media API's to inherit from. That is, a Twitter interface and Facebook interface could each have an inherited *findBySocialUserId*() function to override.

Alternatively, we can solve this same issue of having a large interface file for social media integration by incorporating the social media profile ID into the Hometown Hero user table on the database. This would reduce the number of queries required to verify a user exists and/or create a new user who is using social media authentication. This refactoring would be very straight forward, and at this moment would only affect the Twitter interface.

10. CUSTOMER QUESTIONS/REQUIREMENT CHANGES

We did not need to meet with the client this week. However, he did contact us via Slack to see if we had any questions. We shared the current version of the project with him to demonstrate our progress. He didn't have any questions or concerns about our work so far.

Overall, the client, Ben Rodarte, had a clear vision for what he expected, and he clearly communicated those expectations to us as a group. He was responsive, prompt, and professional when answering questions and providing feedback.

11. INTEGRATION TESTS

We focused on two integration tests, which can be further broken down. First, we must verify a user can login via email and access account information no matter the login choice. We also focused on a user being able to create/edit and view their profile information. Integrating the frontend with the backend development was quite the task.

From User Story #1, integration testing revealed a few changes that needed to be implemented in our database SQL definitions. One minor change was the removal of the attribute for an access_token associated with a user of a social media account. While this attribute may have its use in the future for helping users move between browsers, we found it to not be necessary at this time with our use of the modules PassportJS and express-session. This inefficiency was highlighted in our integration testing. The Twitter interface removed the access_token and added a profile ID attribute.

A larger issue we encountered during integration testing involved the complexity of our database. In our original schema, the database kept track of social media accounts with three tables, as there was a many-to-many relationship between a Hometown Hero user and a social media object. To represent the relationship, we created a social media service table, which identified a service name (e.g. Facebook), a Hometown Hero user table, and a relational table used to connect the service ID, a user's social media profile ID (delivered through the authentication process), and the Hometown Hero user ID from the main user table. While this was a valid schema to relate the user ID and social ID, it required extra function calls in our database interface to query the three tables. Ultimately our group decided to add the social media profile identifications directly to the main user table to simplify database interactions. The addition of columns to a table is not ideal for a many-to-many relationship; however, the number of different social media providers will be limited to a small number.

Integration testing during all the user stories revealed some problems created by separate programmer pairs implementing the ExpressJS middleware differently. As discussed in our previous write up, the discovery of PassportJS seemed to streamline our code by implementing the same middleware for multiple types of authentication. As the database interaction modules for different kinds of authentication were integrated, new errors were occurring involving passport and express-sessions. This was an issue in that the majority of the time allocated for the new modules for this weeks stories went to further research into understanding the passport.js authentication flow. While this research experience was valuable in terms of education, there were not enough units to implement every commitment in the schedule.

12. CONCLUSION

Our team did not complete our assigned tasks for the week. However, we continued with good planning and constant interaction with each other as programmers, our programming tasks went fairly smoothly. We completed the majority of the assigned tasks. Relative to our tasks last week, we almost doubled our workload this week. We had originally thought that the first user story would take the greatest effort. And once the first user story was completed, each of the remaining user stories would then take less time each because they would incrementally add on to the first. However, the assigned tasks for week two took longer than anticipated, thus delaying the completion of User Stories #2 and #4. Moving forward, we will more accurately assess the effort required to complete each set of requirements.

13. CUSTOMER AVAILABILITY

We did not schedule a meeting with the client this week, however, he was available to respond to questions via Slack.

14. TEAM CONTRIBUTIONS

14.1 Jon Austin

1. Frontend profile screens
2. Collaborated on homework document

14.2 Valerie Chapple

1. Backend Social Media integration framework
2. Collaborated on User Story and diagram discussions

14.3 Kenny Lew

1. Frontend profile screens
2. Email login functionality

14.4 Charlotte Murphy

1. Backend update profile screen
2. Collaborated on homework document

14.5 Gregory Niebanck

1. User story #1 Facebook DB interactions unit testing
2. User story #1 Facebook DB integration
3. User story #1 write up