HelpMeOut

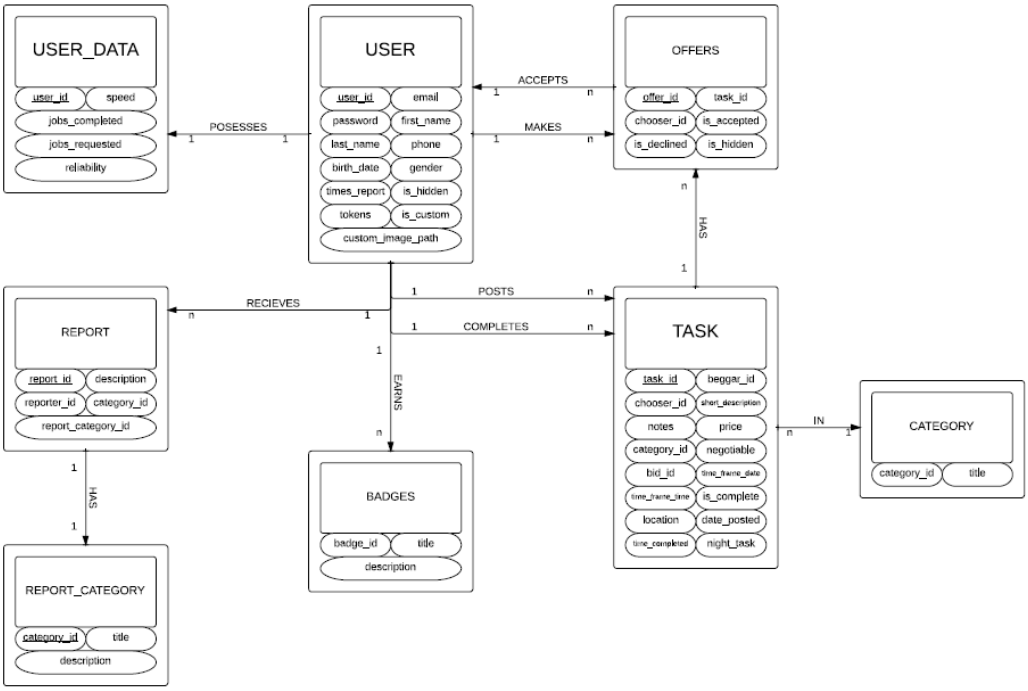
for CSE 3330/CSE 3345 Semester Project

HelpMeOut is a website developed by LabLabs, our team is made up of Austin Wells (Android), Chris Dunn (GUI), Jordan Silver (GUI), Charlie Albright (DB), Spencer Kaiser (DB), and John Wilson (DB). The website allows a user to request for help and fulfill job request for people and earn real world cash in return. All a user has to do is make an account. If they want to post jobs, we've integrated PayPal in our website for ease of transaction (and so any potential users don't have to worry about giving their credit card info out to ANOTHER company).

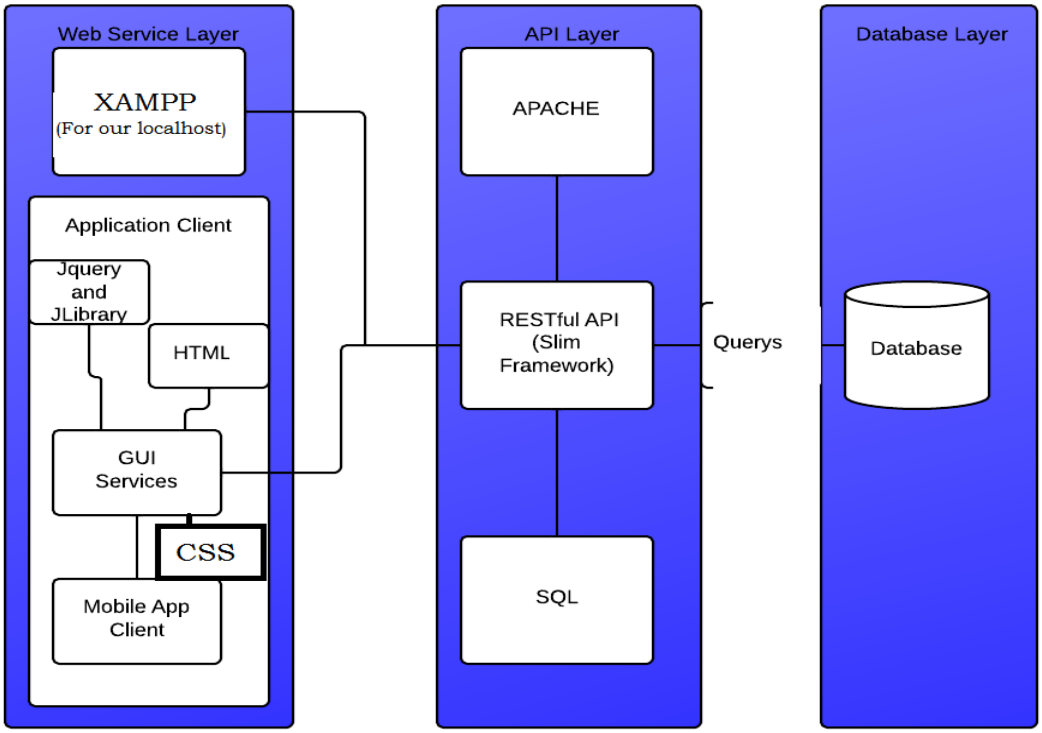
When a user post a job, they get placed on the main page in several different categories such as food, laundry, rides, groceries, cleanings, tech support, and more! When someone offers help, the user can view the helper's profile info to check out their reliability and speed ranking. When the user finds someone they are happy with they choose that person to complete their task. That user then does the task and when the job is done. The poster can push complete job and both users rate each other. It's a simple as that!

On the User's profile page, their contact info, name, custom profile image, speed and reliability rankings, and Badges they've earned are displayed. The user can change any of their contact at any time and upload a profile picture in our easy settings page.

<u>Use Case Diagram:</u> Located in Design Folder as "Final DB Model"



<u>Software Architecture Diagram:</u>

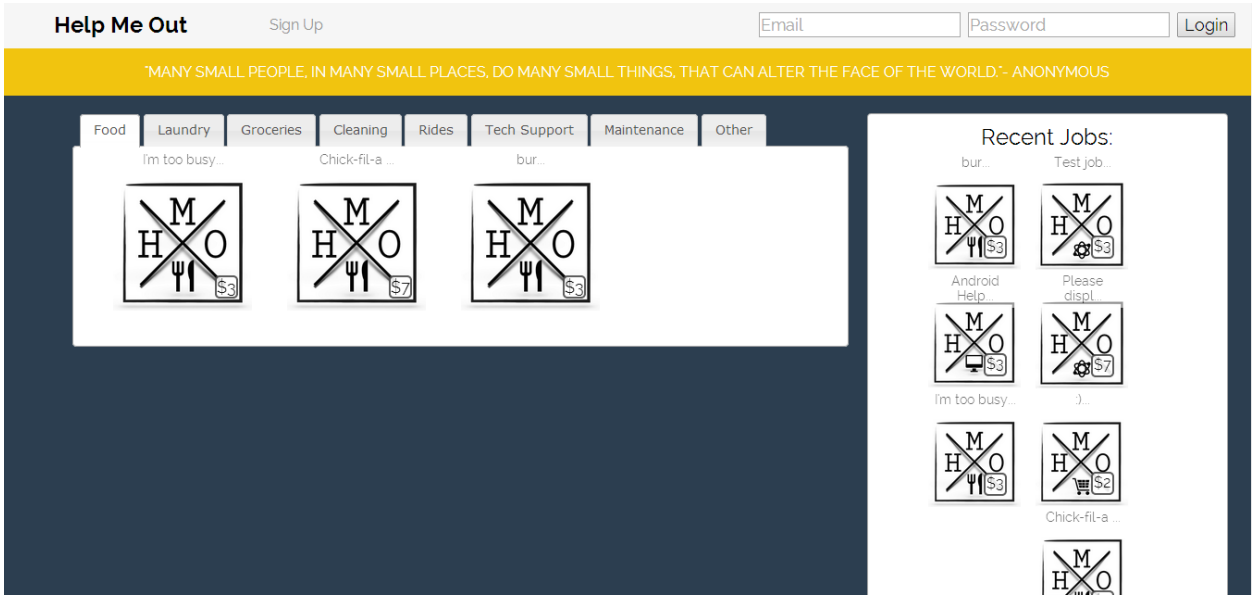


- HTML: HTML stands for **H**yper **T**ext **M**arkup **L**anguage, and is a markup language that are a set of tags used to describe a page's content. HTML is more commonly known by non-programmers as "web pages".
- CSS: **CSS** stands for **C**ascading **S**tyle **S**heets, these "styles" tell the website how to display the HTML elements that have been coded. This was added in HTML4 and can be described to non-programmers as, "what makes a website look good."
- JavaScript: This language is the most popular programming language in the world right now. With Javascript you can: change HTML attributes, HTML styles, HTML Elements, and more. If you want to be a web programmer this is one of the 3 you have to learn (the other two being HTML and CSS discussed above).
- PHP: Is a programming language intended for server side programming but is also a popular scripting language used with web development. While this was the only language our DB guys coded in, it was used minimally by our GUI team.
- Jquery: Jquery is a java library that makes it "easier to use javascript" on a website. jQuery is able to take commonly performed task by web development teams and puts them into methods that you can call with a single line of code. Many of today's biggest companies use Jquery so learning how to program in it is not only useful, but most certainly a must.
- Slim Framework: Allows the user to write simple queries that can accomplish a lot. The queries we used in this website our get and post request, A get request retrieves info about a collection of resources. An easy way to remember this is a get "gets you data". A Post request creates a new resource. An easy way to remember this is a post "creates a

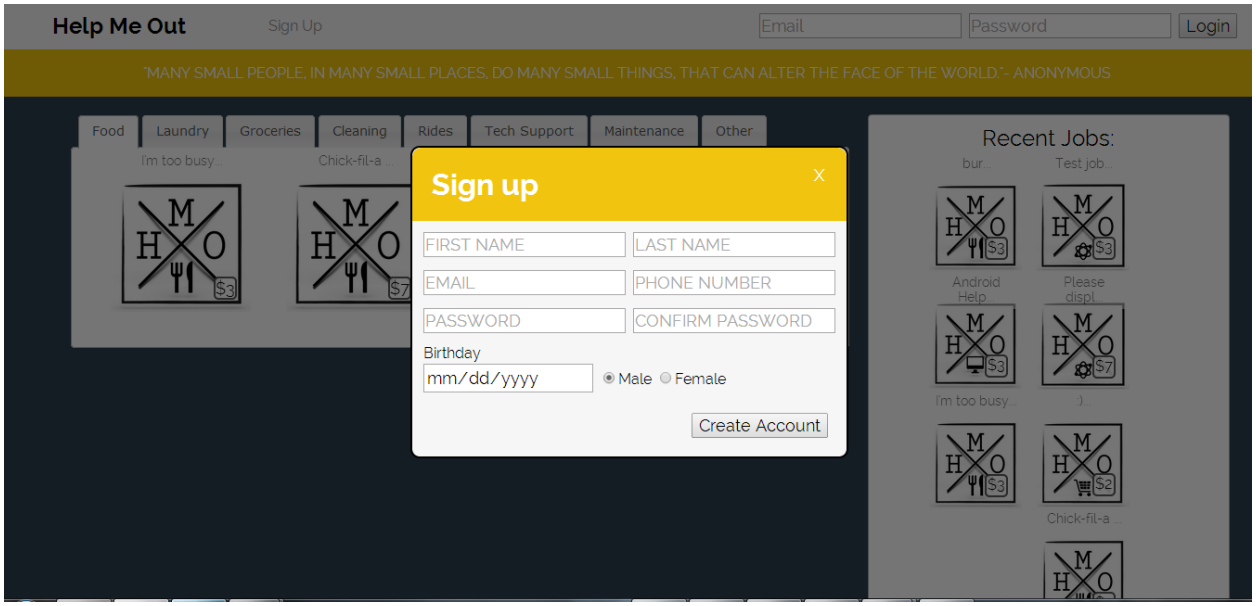
new user/job/etc…”

- XAMPP: XAMPP allows us to host a person local server on their computer as well as create, edit, and store a database (phpmyadmin). This is crucial when building a website and allows us to test changes or even see if what we’re coding works before we push it to the live server.
- GitHub: GitHub was not a tool used in the website but did allow my group to share code for the website as we were able to post any changes made and other people could pull the newest changes. This powerful tool also allows us to share our code with anyone, meaning that not only could a test team or professors see it, but future employers too.

UI WALKTHROUGH OF HELPMEOUT:

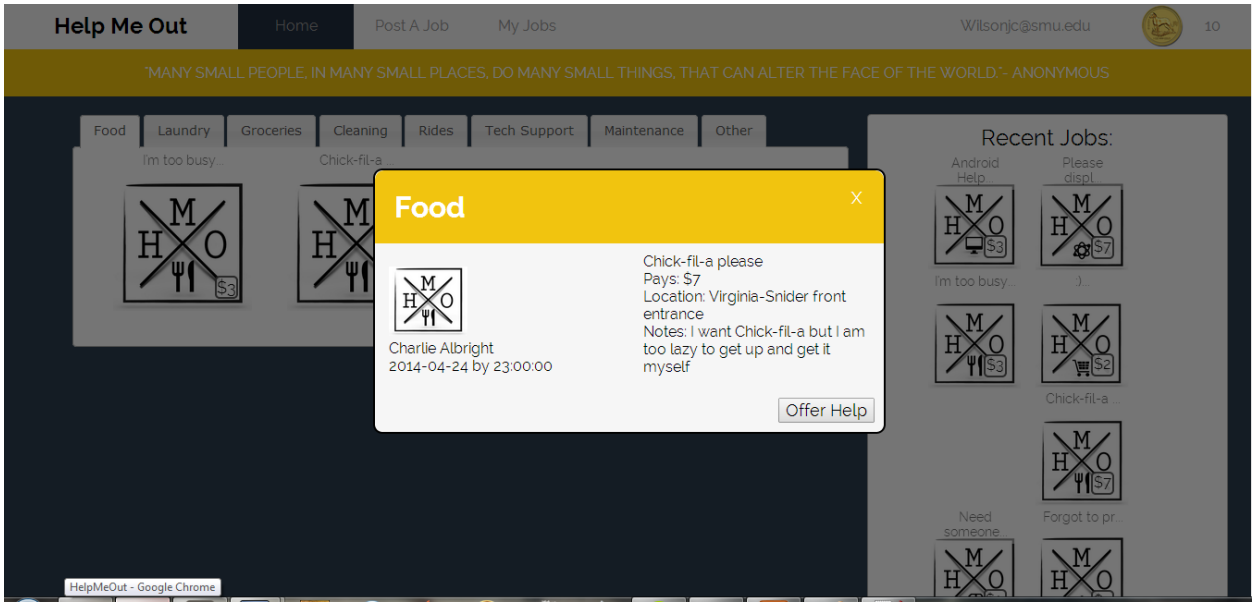


Hey, this is Wilson, and he just discovered this site called HelpMeOut! He’s going to be taking a walkthrough the website and see all the features it has to offer. This is the home page when a user first goes on our website. They can see jobs posted by category and the most recent jobs, but they can not offer help on these jobs. Look like I have to click on the “Sign Up” button.

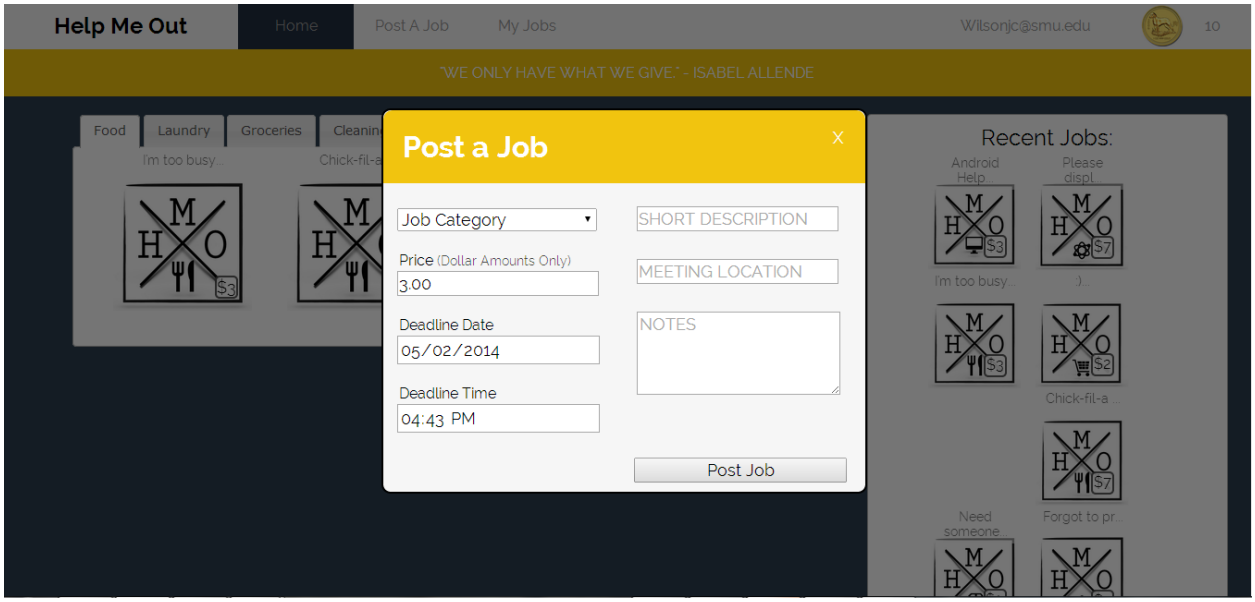


The new user must create an account, here a create account pop-up appears asking for the user's first and last name, their email, phone number, password, birthday, and their gender. Once this is entered in the homepage changes slightly.

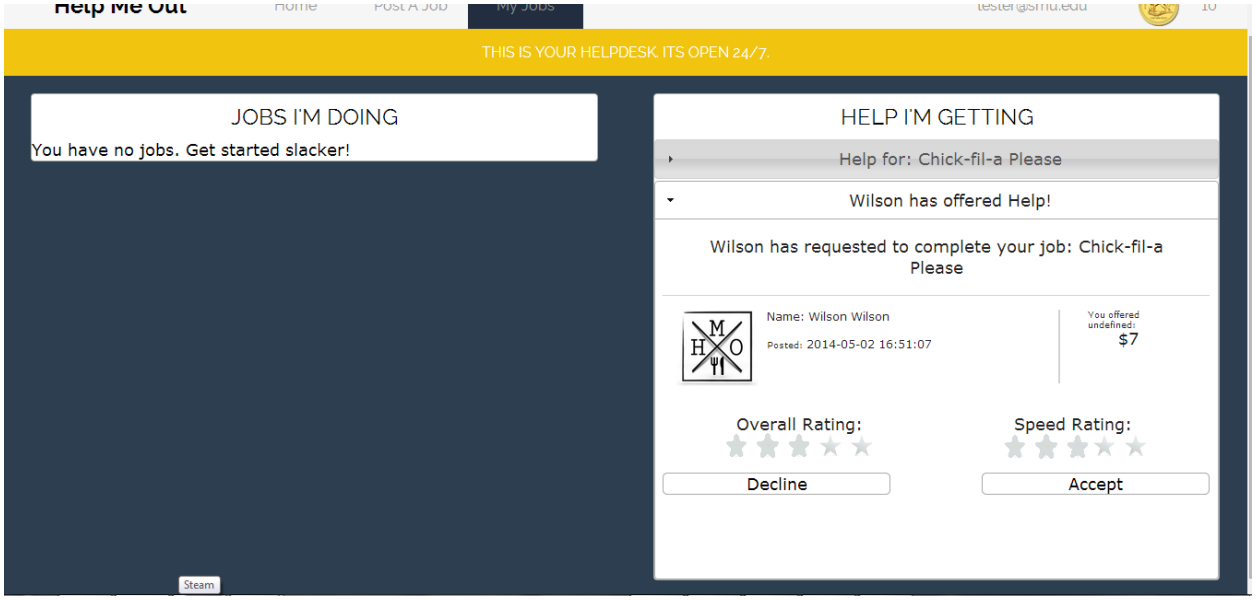
The new user has created an account and can now offer help. We even gave them 10 tokens to start out with (10 jobs for free!) So they can click on any to start offering help!



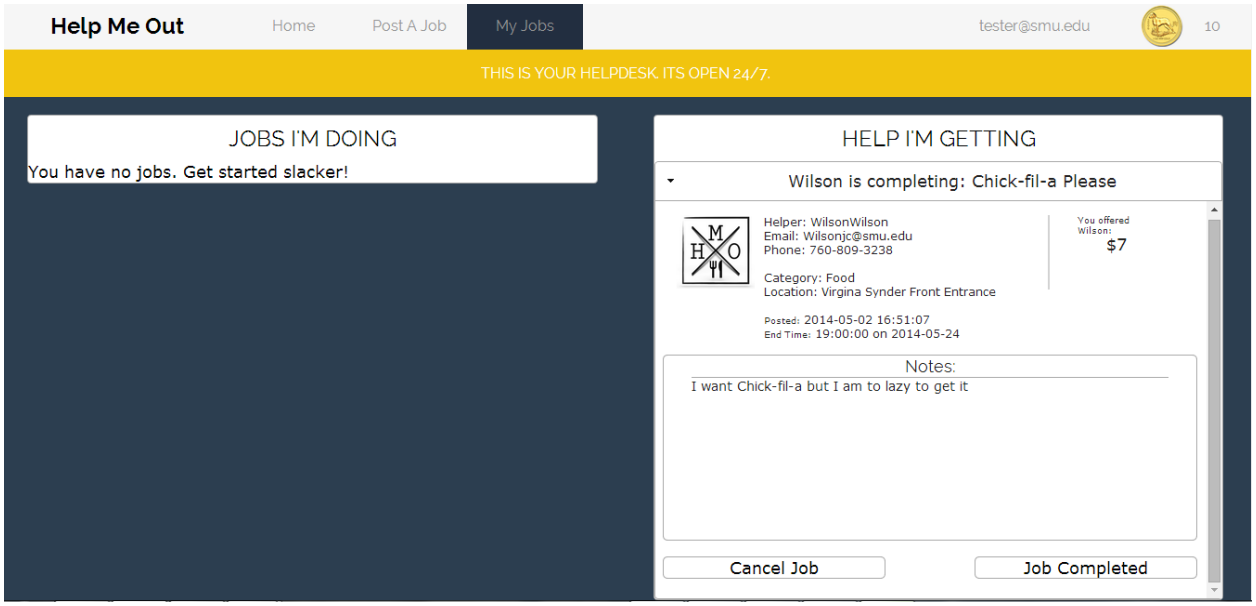
Now the new user is offering help to a user named Charlie Albright. He wants Chick-fil-a delivered to him because he's to lazy to go himself. Well looks like his laziness is going to being paying us $7! We click offer help and now must wait for him to accept.



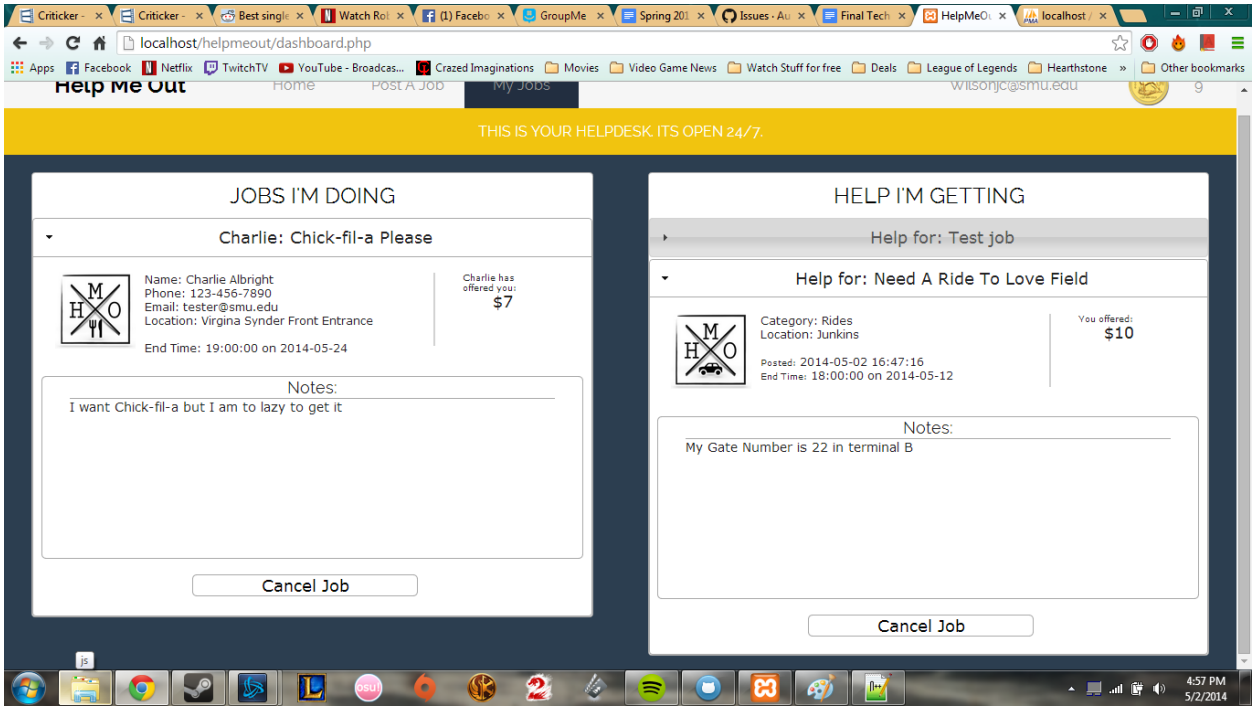
While we wait for Charlie to accept our offer, let's post a job! By clicking on the post a job category in the top nav bar, we our given this popup. As you can see creating a job is pretty easy. All we need to do is pick a category, enter how much we'll pay the person to do the job, when's the deadline or when do we NEED this done by, a title, a meeting location, and an optional notes section for the user to put down any details about his/her job.



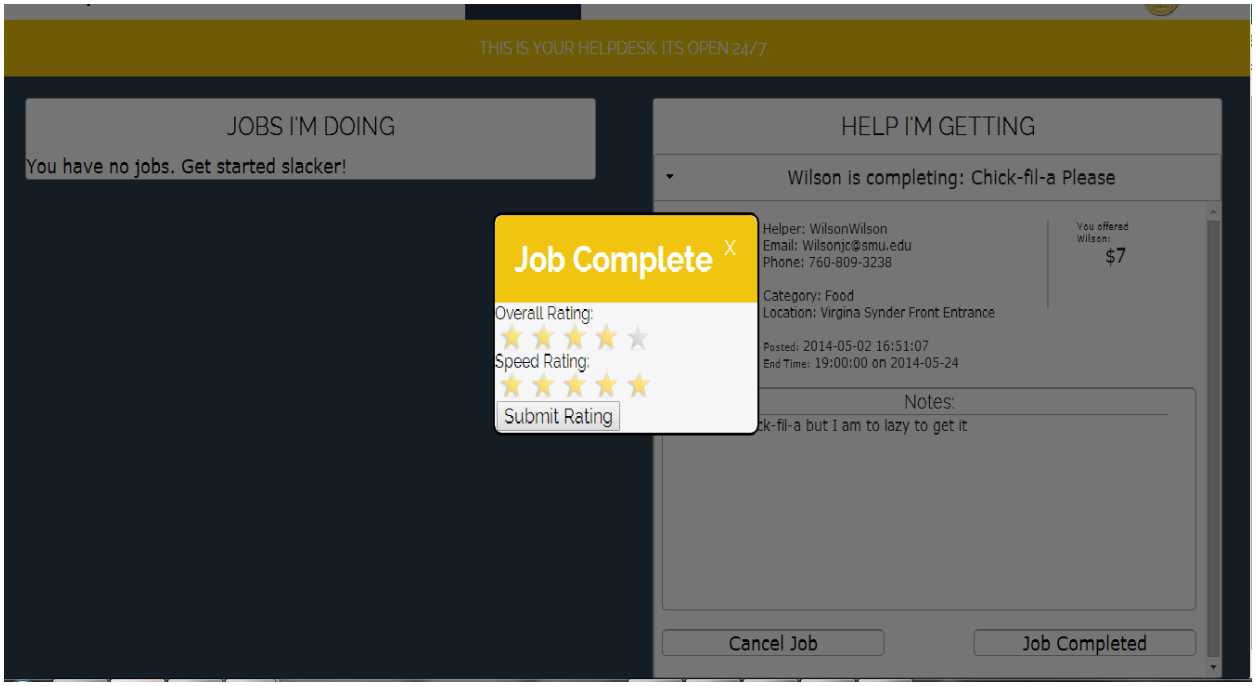
Over on Charlie's profile, he can see that Wilson has offered him help! Since Wilson hasn't done a job yet he has no rating, but Charlie still accepts him anyway by clicking the accept button!



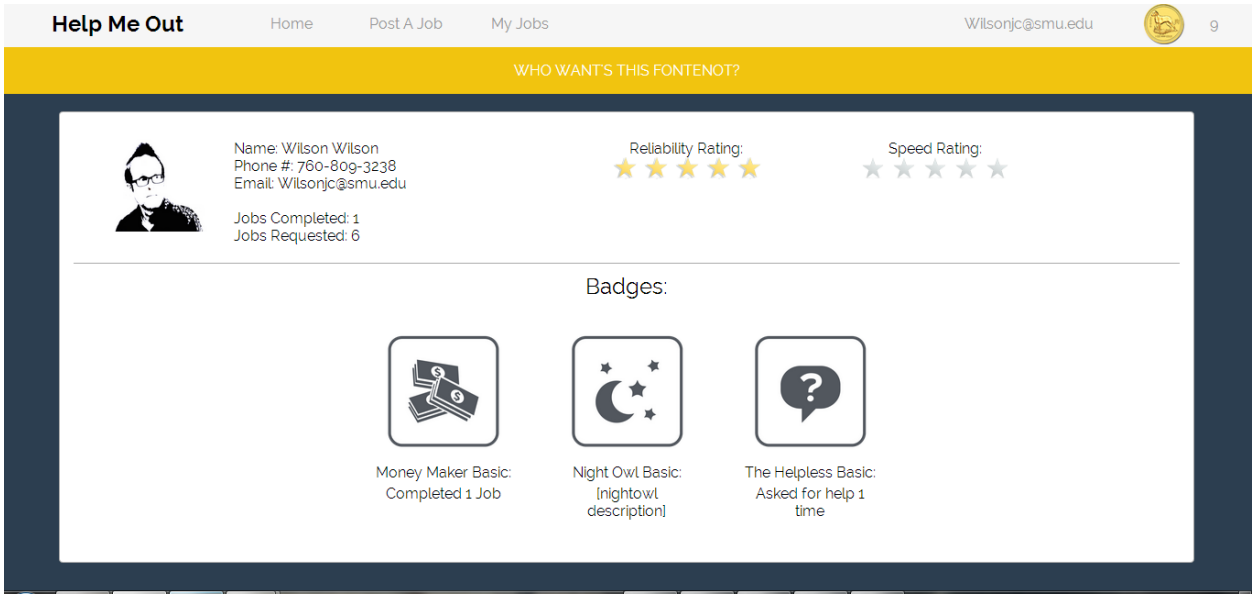
Now Charlie just has to wait while Wilson does the job and then can hit job complete. All of Wilson's contact info is up so he can contact Wilson about the job. If something comes up and Charlie has to leave, all he has to do is hit cancel job.



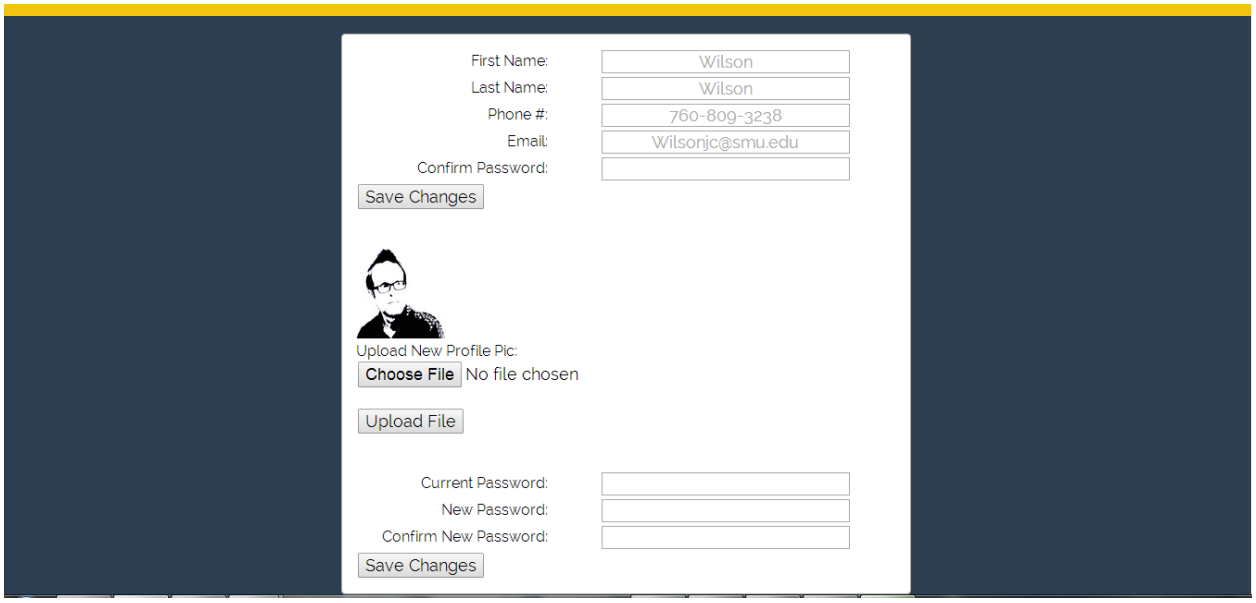
On Wilson's screen is two bars, one is for the job he is now doing and the other is for the job he needs help on. For the job he's doing Charlie's contact info is displayed so the two can talk and go over the order. On the right, info about wilson's job is posted so he remembers what job he made.



Once the job is done, Charlie can rate Wilson based off of how fast he was and how reliable (accurate) the order was. Looks like Wilson did pretty good!



After completing the job Wilson goes to his profile to check it out. Looks like He earned some badges for everything he's done! "That's pretty sweet" says Wilson, I should keep using this website to unlock all the tiers of the badges!" (Note: I accidently pressed 0 stars for the speed rating after the photo was taken, this is not a bug. Just user error)



Here is the setting page, where Wilson can change anything from his phone number to uploading a new profile picture. It's important to keep your info up to date on the site because contacting people while performing jobs is very important!

TESTING

We showed this website to people on the engineering floor after iteration 2, to get a feel for what they thought about the aesthetic looks of the site and whether or not they were confused by any of the processes on the website. We let them have free reign over the website and just watched and observed them. Alot of people thought the general idea of the site was "cool",

which is what we want in a target market. We were hoping for college kids to think the idea of HelpMeOut was “cool” not just that it was “cool” because we made this for a school project. Testers also overwhelming liked the ability to see the other user’s info after they accepted the job, and liked how it provided an increased sense of reliability. Also since it didn’t show their info prior to accepting the job, more people felt comfortable with giving this info up.

The test team that looked over our program was incredibly useful in finding bugs. Jordan Kayse found so many helpful bugs after iteration two, ones that we hadn’t found yet or wouldn’t have even thought of looking for. For both iterations she was constantly giving us suggestions and pointers on how to make things clearer to the user. She has definitely helped us shape our final product into what we hoped it could be. A easy, user friendly website.

As a test team, our team was really good at finding bugs , especially first iteration where for a group meeting we all just made accounts and saw what we could break. Making breaking the website into a game helped thoroughly search for bugs. By iteration 2, we mostly were giving suggestions as the website we tested was nearly done and thoroughly checked for bugs before being given to us. I haven’t checked out their website yet to see how it looks, but judging by the GitHub comments, it seems the team we tested has taken many of our idea’s into consideration or even implementation.

TEAM REFLECTION

During our first iteration, the DB team got behind when they debated whether or not to use Laravel framework over Slim framework. This ended up costing us precious time as after reading through documentation, they decided that it would be to much work and too powerful for the website we were setting out to create. We also had a problem with our server during the first iteration that caused us to not be able to log in or create an account (even though the functionality was there).

By iteration 2, our team was focused, scheduling mini-hackathons during the weekend to accomplish a lot of work and by changing how we ran our group meetings. Austin Wells suggest we started each meeting with a “stand-up” which involved standing up in front of the group and saying the following things:

- What you had accomplished that week (I.E.” I created the UI for the settings page”)
- What were your setbacks (I.E.” I had to research and then asked for help on how to make a drop down menu”)
- And what were your goals for next meeting (I.E. “By next meeting I want to have the UI for the profile page finished.”)

This created accountability in the group, as we knew who was not holding their weight, what was accomplished, and which DB/GUI guys needed to work together to accomplish their goals for the next meeting. By iteration 2, most of our website was done and ready to present, only missing a couple of features, that while not important for the website to function, were still important for the user (such as badges that give a gamification process to the website and keep users engaged in our website for longer than they would).

Outside of class we attempted to learn Laravel, but found that way more complex than what was needed. But, taking a class devoted to that, since it is one of the most used framework might be useful for preparing us for a job in the field (even though like we discovered, it is much more complex than for what is needed by our’s or anyone else’s in the class website and would

need to be taught on the GUI side to in order to have the full effect). We also learned that their is a data type called “blob” and that it helps store a binary data type (such as pictures). Our group also learned how to incorporate outside code bases (such as the jquery/css for the job tabs & the accordions for my jobs)

If we started this project again, we would definitely have started the “stand up” meetings a lot earlier.We would also have started using GitHub a lot sooner too. Both of these tools increased our productivity exponentially. Pretty much most of what we would have done differently would have helped out productivity, it would have allowed us to avoid the problems we had during the first iteration while still creating the quality website we ended up with. I don’t think there are more ways to stress how much work this project is then from actually doing it, it let’s you know “Oh, this IS a lot of work”.

CLOSING-WHAT WOULD WE DO NEXT?

In our next iteration the two biggest features we’d add would be a notification system that let’s the user know when a job they have posted was offered help or whether a job they offered help to was canceled or accepted. This would help the user out a lot and solve potential problems of the user never knowing their task was canceled or their offer was accepted. The other big feature we’d add is a report system, the reporting system is already set up in the DB, but to make it work the way we envisioned it would require some work on the middle and front end. If a user gets reported enough time they would be effectively be shadow banned from the website. They could post jobs or offer help and to them it would appear as though everything is normal. But, their post would be hidden and their offers would never reach another user. Thus the reported person wouldn’t make another account and continue causing problems, they would believe that everyone would see the problems they were causing.

All considering, this project was a great way to not only learn GUI or DB but to test ourselves with rigours deadlines and constantly working through problems on our own. It’s extremely beneficial to any comp sci student and we’re all excited to tackle the other side of the project next semester (after a much deserved break of course).

APPENDIX A. DATABASE DICTIONARY

1. Category: Jobs a user post are split into categories. The Job’s only get a category_id. In this table we explain what each category_id means.
    a. Category_id-int(3)-The Primary key of the table.
    b. Title-varchar(30)-What each category is, matches up with the different category’s a user can post his job under: food delivery, rides, grocery, cleaning, laundry, maintenance, tech support, and other.
2. Offers: This table stores all the offers made on the website. it contains info on whether they are accepted, declined, or the user making the request is hidden from the website (not implemented feature).
    a. Offer_id-int(11)-The primary key of the table.
    b. task_id-int(11)-The Id number of the job posted on the website (the primary key from the Task table)
    c. chooser_id-int(11)-The user_id of the person offering help on a job.

    d. is_accepted-tinyint(1)- Once a user selects who they want to do their job, this column is set to 1 for the corresponding “chooser” thanks to a trigger set up in the DB.
    e. is_declined-tinyint(1)-Every other user who wasn’t selected to do a job gets a 1 in this column (set up automatically by a trigger or when the user just declines an offer from a user on the website)
    f. is_hidden-tinyint(1)-A feature not implemented in any of our iterations, this would notify this table that the user is hidden and their offers would not count.
3. Task: These are the jobs posted on our website. The most important table for our website. This contains all the info for the jobs.
    a. task_id-int(11)-The primary key of the website.
    b. beggar_id-int(11)-The user_id of the person requesting help on the website (who post the job?)
    c. chooser_id-int(11)-Initially set to null, once a “beggar” selects a person to do the job, their user_id fills this spot on the table.
    d. short_description-varchar(36)-The title of the job.
    e. notes-varchar(200)-Additional details for the job
    f. price-int(5)-How much money you’re paying to a person for completing your tasks
    g. category_id-int(3)-Which category is this job posted in?
    h. negotiable-tinyint(1)-A feature that was taken out but still left in DB side in case we want to add in upon user request, if this is set to 1 then a user
    i. bid_id-int(11)-Same deal as negotiable, would allow us to collect to the bid table (discussed in Appendix B, Unused Tables in the DB.)
    j. time_frame_date-date-The Deadline of when this job needs to be done.
    k. time_frame_time-time-The Deadline of what time this job needs to be done by.
    l. is_complete-tinyint(1)-If it’s set to 1, then the job is done.
    m. location-varchar(100)-Where is the person asking for help located?
    n. date_posted-datetime-When was the job posted?
    o. time_completed-time-When was the job done
    p. night_task-tinyint(1)-If a job is done at night according to the time_completed value, this value is set to 1 (for the night owl badge)
4. User: Here is where most of the user’s data that he/she enters when they create an account is posted.
    a. user_id-int(11)-The primary key of this table
    b. email-varchar(100)-The email of the user.
    c. password-varchar(32)-The password the user selects to log in to the website. This is hashed
    d. first_name-varchar(40)-The user’s first name
    e. Last_name-varchar(40)-The user’s last name
    f. phone-varchar(14)-the user’s phone number
    g. birth_date:date- The user’s brithday
    h. gender-enum('M', 'F')-The user’s gender
    i. time_reported-int(2)-A feature that was not implemented yet, this contains the

amount of time's the user was reported.

j. is_hidden-tinyint(1)-A feature that was not implemented yet, Once a user gets reported enough the user is hidden on the website. And their post and job offers are hidden but they don't know that. ("shadow ban")
k. tokens-int(11)-The amount of tokens the user has, once a job is accepted a user loses one token.
l. is_custom-tinyint(1)-Does the user have a custom profile pic? If so this is set to 1.
m. custom_image_path:varchar(100) Where the custom image is located.

5. user_data: Stats about the user as they use the website
    a. User_id- int(11)- The primary key of the table
    b. jobs_completed-int(11)- How many jobs the user has completed
    c. jobs_requested-int(11)- How many jobs the user has created
    d. speed-int(100)- How fast was the person who completed the job? Rated by the user who they completed the job for
    e. reliability-int(100)- How reliable, accurate, and well-done was the job? Rated by the user who they completed the job for.

EXPLANATION OF RELATIONSHIPS: Starting at User, when they create a user creates an account they set most of the attributes discussed above, the only ones they don't set are user_id, and their custom_image and custom_image_path (done in the settings page) that **posseses** user_data in a "1 to 1" relationship.

The user can **accept and create** offers with the offers table as well as **post and complete** jobs on the task table. The user also **earns** badges from the badge table and **recieves** reports from the report table. All these relationships discussed in this paragraph are "1 to many relationships"

A task belongs **in** one category so that is a "many to 1" relationship. A task also **has** "1 to many" offers from the offer table. Finally a report **has** a "1 to 1" relationship with it's report category.

APPENDIX B: Unused Features in the DB

The following tables mentioned are features we wanted to implement in the next iteration or were just worked at in another way (but left in, in case this website gets big and we need more efficient ways to expand)

1. Badges: The gamification aspect of our website. This will encourage users to keep being active in the community as they will want to earn additional badges. This table was created at the beginning when we thought there would be a lot of badges made. Right now we only have basic badges implemented, but if we expand the badge aspect of our website then we will have to use this table.
    a. badge_id-int(3)-The primary key of this table
    b. Title-varchar(30)-The name of the badge
    c. Description-varchar(200)-What does the badge say? How'd you earn it?
2. Badges_earned:Currently an empty table, but this would connect our badge table to our user table much in the same way offers connects task and users together.
3. Bids:A feature that was originally going to be a huge focus on our website. This would

have allowed users to bid a price on the task they wanted to accomplish. This feature was scrapped though because we figured the user wanted this to be a quick and easy process. If this website's traffic picks up and users post a lot of jobs that are not needed to be done immediately we might ask the community if they want this feature.

4. report:Currently an empty table, this is where reports on users would go. It would function similary to the task table, with time stamps and short descriptions from the user. It even has a category table connecting to it just like task has a category table connecting to it!
5. report_category: The user can pick why they are reporting a user. The user can pick from the following categories: Incomplete Task, Unreasonably Late, Minimal Completion, Fake Profile, Spam, Harassment, Offensive Profile, Illegal/Inappropriate Request.
    a. category_id-int(3)- The primary key for this table
    b. title-varchar(30)- the title for the report. (titles mentioned above)
    c. description-varchar(200)-Describes the title in more detail so the user understands what the categories mean.