Homework 7

Mohammed Algadhib
algadhim@oregonstate.edu
Johnny Chan
chanjoh@oregonstate.edu

Riley Kraft
kraftme@oregonstate.edu

Diego Saer
saerd@oregonstate.edu

1. SOFTWARE

URL:
https://github.com/cs361-su18-group8/FinalProject

OR

http://flip2.engr.oregonstate.edu:4856/

INSTRUCTIONS:

Pre-conditions: In order to run and test this web application the user should have Node and a web browser installed.

Download the application: To download the application there are two options, the first is by cloning the repository using git by executing this command `git clone <URL>. The second method is by downloading a .zip file of the software from GitHub, this can be done by navigating to the software's URL and then clicking on `Clone or download` green button, then clicking on `Download ZIP`.

Install dependencies and packages: In order to run the application successfully, dependencies and required packages should be installed first. This can easily be done using node package manager. To install the missing packages run the command `npm install` in the home directory of the project using any command line interface (PowerShell, CMD, Bash, etc.). After executing this command all required dependencies will be downloaded.

Run the app: After installing all the required dependencies, now it is possible to run the application. To run the application simply execute the command `node app.js` in the home directory of the project. If there are no error messages, the app is running.

Start using the app: The app can be used now by visiting http://localhost:3000

OR

Preconditions: In order to run and test the is web application, the user must be able to connect to the OSU VPN before attempting to go to the URL. https://oregonstate.teamdynamix.com/TDClient/KB/?CategoryID=6889

Run the app: Log in to the OSU VPN and go to the URL provided above on the flip server.

2. USER STORIES DUE

2.1 USER STORY A: DAILY HEALTH QUESTIONS

a. TEAM

Mohammed (pilot) and Diego (co-pilot)

b. TASKS

i. A2: Come up with survey questions and add them to the database

1. Unit Tests
   - Survey questions were able to be retrieved when connecting to database through node.js with mySQL.
2. Problems
   - Could not figure out how exactly to add questions to the database.
3. Time

   2 hours
4. Status

   Completed
5. What's Left?
   - 11 Default questions set (sources used to find them in references). The survey questions list will be continually expanded with questions.

ii. A3: Create a homepage icon for the survey.

1. Unit Tests
   - Check the "take survey" icon would appear correctly when the homepage loads.
2. Problems
   - N/A
3. Time

   15 minutes
4. Status

   Completed
5. What's Left?
   - N/A

iii. A5: Create the web page user interface to answer survey questions.

1. Unit Tests
   - The survey instructions loads and has the correct styling when the survey is entered from the homepage.
   - The survey questions page loads and has the correct styling when entered from the survey instructions page.
   - The slider moves and interacts as expected in the survey questions page.
   - The return to homepage link works as expected.
2. Problems
   - N/A
3. Time

   3 hours
4. Status

   Incomplete
5. What's Left?
   - Implement the numbers that correspond to the rating scale of the slider.

iv. A7: Write code to retrieve questions from database.

1. Unit Tests
   - Check to see if a proper connection to the database could be established by printing status to console.
   - Check to see if queries were retrieving questions from database by outputting data to console.
   - Check to see if code was returning questions in the correct format by outputting data to console.
   - Check to see if returned query results were rendered on the website.
   - Check to see if a random question would appear each time the survey was started or each time the next question link was selected.
2. Problems
   - A connection to the database could not be established from node.js
   - SQL queries to database were not returning the desired data.
   - Code was not being returned in the correct format.
   - Questions were not being rendered on website.
   - Many methods were tried to present random questions to the user. For example, having a randomly generated number sent through a GET request query string proved to be more complicated than originally thought and revealed information about the database. In the end a number randomly generated on the server proved to be simpler and more secure.
3. Time

   3 hours
4. Status

   Incomplete
5. What's Left?
   - Store the question ID of randomly retrieved question so that when an answer to a question is received, so that the returned answer is complete with a question ID.
   - Make sure questions following any previously answered questions are not repeated.
   - Decide how many questions a daily survey will ask.

2.2 USER STORY K: MANUALLY ENTER MEDICATIONS

a. TEAM

Mohammed (pilot) and Diego (co-pilot) - K2

Johnny (pilot) & Riley (co-pilot) - K3, K5, K6

b. TASKS

i. K2: Create add prescription and add over the counter homepage icons.

1. Unit Tests
   - Check if the homepage present icons for 'add prescription' and 'add over the counter' links to respective pages
   - Check that the icons are sized properly to fit on the page layout.
   - check if the color of the icons match the website scheme.
2. Problems
   - There were no technical problems encountered. There biggest hurdle was in deciding whether to use an icons from a front end framework, such as Bootstrap, or free pictures off an image search.
3. Time
   - 3 hours
4. Status
   - Find and/or create icons and replace the linked text. - complete
5. What's Left?

   Complete

ii. K3: Create medication form webpage.

1. Unit Tests
   - Connect to OSU VPN, launch web app, and

refresh webpage after every edit to the code to check for rendered template
- Only apply changes to the Over-the-Counter form in order to isolate and resolve mistakes. After features have been finalized, with customer approval, transfer findings to Prescription form.

2. Problems
- Tried multiple styling options to arrange checkboxes and radio buttons as side-by-side, but they remain stacked. Had to implement individualized float styling within the HTML code to correct the behavior.

3. Time

Riley: 6+ hours of coding and testing styling features

4. Status
- Integrate homepage template - complete
- Adjust styling (significant) - complete
- Implement collapsible fieldset feature - removed from this user story in the interest of simplicity; will be added to a new user story of a lower priority
- Implement edit icons and subsequent features - removed from this user case in the interest of simplicity; will be added to a new user story of a lower priority
- Over-the-Counter form - complete

5. What's Left?
- Implement findings into prescription form

iii. K5: Write code for website to connect to database.

1. Unit Tests
- Comment out qparams in otc.js file to eliminate data format as source of recording error.

2. Problems
- None. First unit test confirmed source of last week's database problems as located in the formating of the data being written.

3. Time

Riley: 15 min

4. Status

Complete

5. What's Left?
- N/A

iv. K6: Write code to record data to database.

1. Unit Tests
- Connect to OSU VPN, open to database, launch web app, and refresh database after every edit to the code to check for a valid data record in the database.
- Comment out individual lines of variables in the qparams variable to isolate the problem data.
- Change all names and ids of variable fields to reflect names in database.

2. Problems
- SQL code would stop recording at date_entered field, no matter the format that was changed. To resolve, the date_entered attribute was moved to the last column in the table so that the js code would not even need to reference it. Field was then set to default to the current timestamp, rather than receive the current date from the application.
- Each data field being recorded to the database has certain criteria it needs to meet to be able to be recorded successfully. So far, we have only been able to figure out the formatting for a little over half of the fields.

3. Time

Riley: 4+ hours of coding and testing

4. Status
- Incomplete
- Transfer functionalities from otc.hbs over to prescription.hbs - removed from this user story in the interest of simplicity; the Prescription form requires more features that the Over-the-Counter form, and must thus have it's own user story (now known as User Story U)
- Find and correct bugs in the code preventing the writing of records to the prescription table in the database - removed from this user story in the interest of simplicity; the Prescription form requires more features that the Over-the-Counter form, and must thus have it's own user story (now known as User Story U)

5. What's Left?
- Find and correct bugs in the Javascript code preventing recording of all fields in a medication record

2.3USER STORY L: USER FRIENDLINESS

a. TEAM

Mohammed (pilot) and Diego (co-pilot)

Johnny (pilot) & Riley (co-pilot)

b. TASKS

i. L2: Create homepage as a template for remaining web pages.

1. Unit Tests

- Setup web app to run node forever on the OSU flip server, and refreshed the webpage after every edit to ensure desired result.
- Edited code in app.js, main.hbs, homePage.hbs, homepageTemplate.css, and created style.css, to cut down on repeated code for each web page file.
- The homepage body and footer icons were tested to see if they would show up on the homepage in the correct size and layout.

2. Problems
   - main.hbs template was not rendering for the other webpages when the app was run. With some additional testing of the app.js setup code, we were able to establish main.hbs and its associated style.css as the header and footer template for all webpages.
   - Logo image was not rendering when the app was run. After some research and code editing, it was found that the image file was in the wrong directory for the app, and the href code was incorrect for the logo's path.
   - The template requires a display of the username and date, as well as a feature to change the size of the webpage font. We wanted these two features on the same line but on opposite sides of screen. Float fields were implemented, but this caused the features to overlap the body of the page. With some tinkering, we were able to move the body of the page below these features. However, the workaround for this has cause undesirable page formatting for other pages in the web app.
3. Time

   Riley: 4+ hours of coding and testing

   Mohammed and Diego: 3 hours to find icons, modify them, and integrate them into the website.
4. Status
   - Finish styling header - completed
   - Integrate colors specified by customer - completed
   - Implement logo - competed
   - Style footer - completed
   - Create icons for homepage body and footer links - completed
   - Adjust formatting for body section of webpage - completed
   - Implement font change functionality - removed from this user story in the interest of simplicity; will be added to a new user story of a lower priority
5. What's Left?
   - Adjust display of today's date to be a long-date

ii. L3: Integrate homepage design into design of other pages of web site.

1. Unit Tests
   - Run node app.js on local machine and view in web browser to check every edit the code for desired results
   - Connect to OSU VPN, launch web app, and refresh webpage after every edit to the code to check for rendered template
2. Problems
   - Simply running node app.js from the IDE environment would not render the styling features from the style.css file. Thus, we had to connect to the OSU VPN and run the app on the flip server.
   - app.js requirements for main.hbs were not rendering the template for every web page.
   - After main.hbs requirements were figured out in testing and the template rendered for every web page, the requirements were not valid after uploading to the GitHub repository. Thus, requiring redefinition of the requirements again.
3. Time

   Riley: 1 hour formating and testing app.js requirements setup
4. Status
   - Enable homepage template to wrap around all other web pages created for the app - complete
5. What's Left?

   N/A

3. SPIKES & SEQUENCE DIAGRAMS

3.1 USER STORY A, TASKS 2,3,5,7

Was the diagram useful? Why or why not?

The sequence diagram was not very useful. The tasks for this week revolved around retrieving the survey questions which was pretty clear for the team working on user story (A message had to go from the website, to the server, to the database, back to the server, and finally back to the user).

Were there any diagrams that you wish that you had? Why or why not?

A step by step diagram that shows what information is necessary to properly implement a question would have been useful. For example, if the website fetches a question with a question ID generated on the server, where should this question ID be kept? Should it be kept on the server somewhere? Should a session be made to store this question ID? Should the returned question ID be stored as a hidden attribute in an HTML element? A diagram that could help answer questions like this would be helpful.

3.2 USER STORY K, TASKS 2,3,5,6

Was the diagram useful? Why or why not?

This week the team responsible for this user story worked mainly on the styling of this form, and correcting coding errors for writing medication records to the database server. In this regard, the sequence diagram for this user case was not particularly helpful this week. It was solved last week what order of events we needed to accomplish in order to test our code, for which the diagram was useful. However, the diagram did not provide any additional information for the tasks this team accomplished this week.

Were there any diagrams that you wish that you had? Why or why not?

The team did not necessarily need a diagram in order to resolve the issues with the database for this week. As was mentioned last week, an ERD would be useful in designing the tables and relationships between those tables when working with databases. On one occasion this would have been useful this week, as we needed to create two additional fields in the medication table that we had not added last week. Our UML diagram was useful in resolving this. Otherwise, we were simply correcting formatting errors in the code, for which a diagram is not useful. Perhaps a spike on how to appropriately format different kinds of data to be written to a database from a Javascript file would have been more useful. This way we could have gotten the research done earlier and would not have to expend so much time research and utilizing trial-and-error to get our code to work properly.

In regards to the styling of the Over-the-Counter page, while our prototypes from HW2 were invaluable, perhaps a website wireframe would have been more useful for the web browser implementation of our application. This way the group could present their ideas to each other in a visual format before investing the time to come up with our individual thoughts on presentation of information, navigation, and interface design. There are still some features between the medications and survey forms regarding navigation interface that differ greatly.

3.3 USER STORY L, TASKS 2-3

Was the spike useful? Why or why not?

With regard to selecting icons, the spike was very useful as the team had to ensure the icons had high contrast and had sufficient spacing between each icon and the elements around it. The team chose icons that could be coloured in a way to provide high contrast with the background. When placing the icons on the website, extra care was taken to ensure adequate spacing between each icon.
With regards to further designing the Over-the-Counter form, this spike was again very useful. To improve the interface of the form, the team implemented more organized space, larger text, contrast between elements, and contrasting colors in order to help the user identify different functionalities.

Were there any diagrams that you wish that you had? Why or why not?

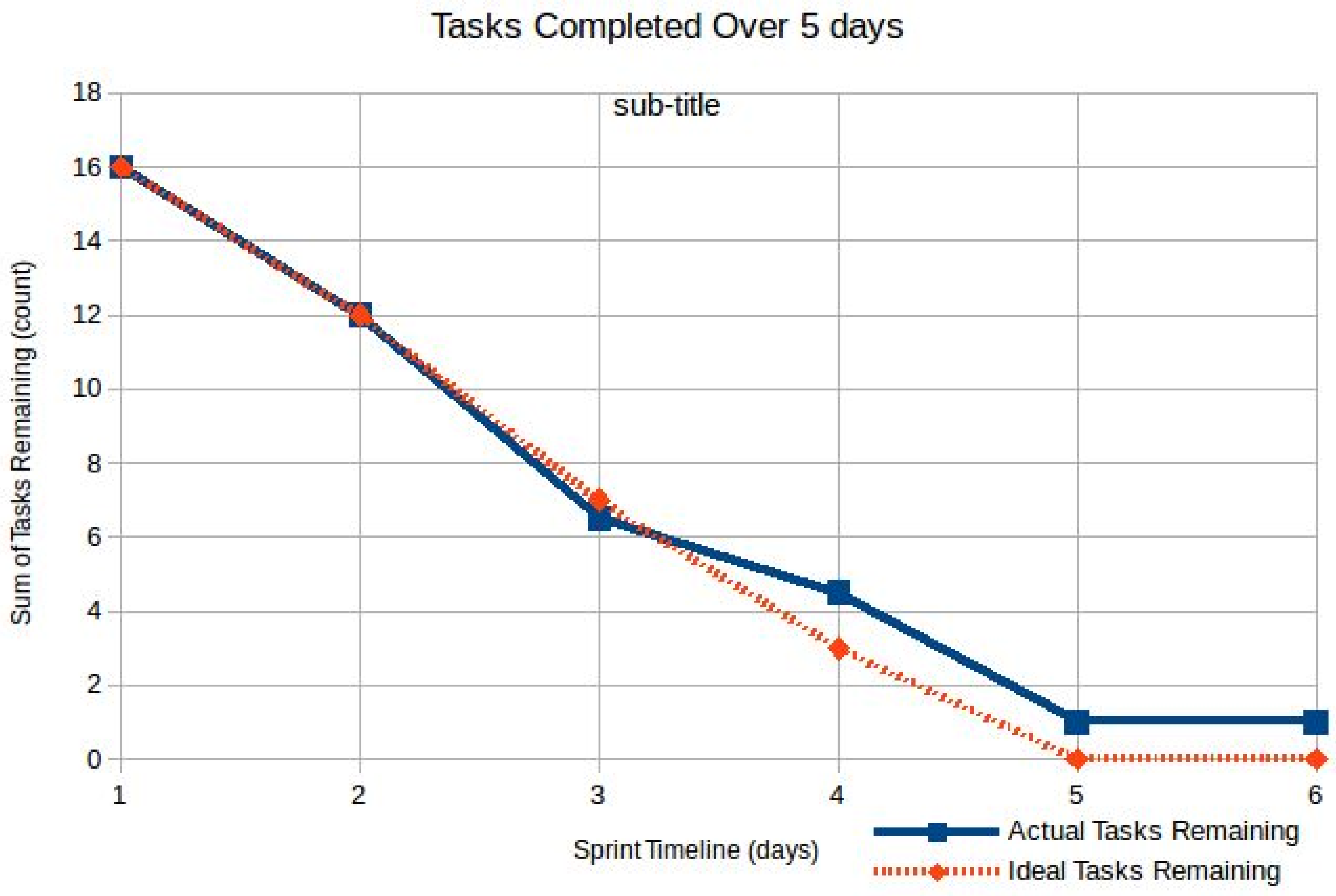
When selecting and implementing the icons there were no diagrams that were deemed necessary. The guidelines from the spike provided enough information to properly implement them.
When arranging the layout and styling code for the main.hbs template, as well as the Over-the-Counter form, the spike was useful, but as mentioned above a website wireframe would also have been useful. This way to group could have agreed on a template design upfront and not expended the time altering our design from the HW6 version to the current HW7 version.
Apart from the styling, the spike was not useful in helping the group organize our actual program files. It took us quite a bit of trial and error to figure out how best to make our separate user stories integrate into one functioning application. In this case, a site file and directory structure would have been very useful. This kind of visual structure would have laid out exactly where each type of file needed to be located, and we could have defined the URL needed to access those files from any other directory in the program. Even now, we have images stored in different directories. To ensure a clean setup for maintainability and reusability, it is important for this team to have a singular structure to our documents, which a site file and directory structure diagram would have been very useful for.

4. BURNDOWN DIAGRAM

The following diagram depicts the expected burdown of slated tasks for this week based on the task outline we formulated in HW6. The timeline for this diagram is slightly shorter for this week since our assignment is due on Friday, rather that Sunday. The expected burndown is represented by the red line in the sum of remaining tasks per day, over the course of the 5 working days of the week. The blue line depicts the actual burndown of the tasks for this week. Note how much similar to the red line the blue line is compared to last week. Last week, the team greatly underestimated the time it would take us to complete our scheduled tasks. After recalculating our estimates for this week, it is apparent from the diagram below that our estimates are quite a bit more accurate for this past week.

We completed most of the user stories that we started last week, save for a few remaining bugs centered around communicating with the database. Results from our customer's review will also determine if the team may officially close out the tasks we've deemed "completed".

For next week, in order to maintain the steady momentum and nearly accurate estimates, we will take the data from the diagram below to guide the limits of the plan outline for next week.



5. REFACTORING

5.1 USER STORY A, TASKS 2,3,5,7

During the implementation of story A this week there was some minor code reuse from previous tasks due to the similarity between the code. However, some tasks didn't require any refactoring due to their simplicity, like task A2.

To accomplish task A2, the team members wrote 11 questions and entered them manually into the database via the portal. However, we don't believe this is the best approach in the long run. In the future, we highly recommend writing a script that reads a question list from a file and then queries them to the database for storage, thus the process is much faster and more efficient.

Task A3 and task K2 are mostly required the same approach. As a result, these tasks were implemented by the same pair of programmers to ensure consistency in design. The code and html tags were reused in all cases of these tasks as they are pretty much the same. The only differences were the actual icons and the URL's associated with each icon.

During the implementation of the survey page, task A5, the template used to implement this was the same as used to create the survey instruction page. As a result these two pages created duplicate code in the application source code. Since this is still just a prototype for the application, in the final version it would be highly recommended to use the same file to render both the survey and its instruction by utilizing the express and handlebars framework. The body of the page is then rendered as the instruction, survey, or really any other page that uses the same template.

Finally in task A7, the mysql library was used to handle most of the database related requests, which resulted in simpler, cleaner, and easier code to read. Moreover, the handlebars library was utilized to render each question with its number on the page without the need to create a Javascript file to modify the 'dom', resulting in more complexity to the source code of the application.

5.2 USER STORY K, TASKS 2,3,5,6

In order to achieve the desired styling for much of the Over-the-Counter form, sections of code were borrowed from reputable CSS tutorial sites (cited within our source code). After completing the styling for the Over-the-Counter form, we had to consider how to implement the same styling for the Prescription form for next week. Originally, the styling code was placed in the oct_style.css file, which is the CSS file exclusive to the Over-the-Counter form. In the interest of preventing duplicated code, and allowing for future features to have access to the exact same styling options, the CSS code for the checkbox and radio buttons was moved from the otc_style.css file to the style.css file, which is the styling file for main.hbs. main.hbs is applied to all .hbs web page files in our project. Thus, these sections of code will not have to be recreated! Rather, the corresponding fields in the Prescription form, and in any future forms, will only need to include the ID and or Class definitions that correspond to the desired styling.

Several fields within the Over-the-Counter form needed isolated styling applied to their features. However, these could be grouped by their input type. Text fields are a certain percentage width of the form. Date fields were reduced to the same size. Number fields were reduced to the same size. Drop down menus were reduced to the same size. Though, each of these field's styling features was handled individually within the otc.hbs file. In order to remove and prevent future duplicated code, the team intends to create classes and/or id definitions for each type of input field and define the size options within the style.css file. This way, the styling for all these types of fields will be consistent throughout the web app, and will not require copying code.

5.3 USER STORY L, TASKS 2-3

After the code to implement main.hbs as the template for all webpages in our application became functional, we were able to refactor the code for all the other webpage files. With this functionality, we could remove all duplicate code between the .hbs and .css files regarding the header and footer content of the web app. Therefor, each additional .hbs file, besides main, and their subsequent .css files now only have to include code for the body of their respective web pages. This significantly reduced duplicate code and cleaned up the remaining code in the webpage files for each page of the app.

6. CUSTOMER QUESTIONS

The customer did not indicate the need for any requirement changes, thus there were no surprises in response to the work we completed this week.

- What do you think of the color scheme? We wanted a high contrast with the white and dark blue, but also found opportunities to use the colors you requested.
- Is the new template for the web pages (header and footer) clean/modern/usable/etc. enough?
- Do you have any changes you would want to see in the Over-the-Counter form?
- Do you have any changes you would want to see in the Survey Questions form?

Our customer did not provide detailed feedback, nor any objections to the direction our team has taken the project in. In this case, the team will proceed with our current version of the plans for this project.

7. INTEGRATION & USABILITY TESTS

The following table is an outline of the integration and usability tests performed on the current version of our web application. We performed many regression tests to ensure any new code did not alter the functionality and style of the old code. The integration tests detail the specifics of how well the different components are working together, and how to successfully run and navigate the system. The chart also includes the positive and/or negative findings of performing the tests, and any changes that need to be made to the system in order to run the same test flawlessly again in the future. The usability tests detail the functionalities and feel of the features that were implemented this past week, any positive and/or negative findings from the tests, and any changes recommended by the tester to improve the user stories and system as a whole.

TEST	RESULTS	CHANGES
Verifying that the questions are in the database: From the database portal verify that the database contain a questions table the contain questions under the description column with a unique id under the ID column. .	The database should contain a table that have two columns, description and ID. the description column will contain the question text and the ID will have a unique id for each question.	One possible change that will be effective with a large set of questions is to add a category column to group similar questions together to help while searching the database.
Verify that the app is able to connect to the database to retrieve questions: first run the app by using `node app.js`, then verify that a message is printed to the console confirming that the connection was established with the database. After a successful connection to the database, navigate to ‘take survey page and then continue to take the survey. Now, verify that in the survey page there is a numbered questions in the middle of the page. Finally, refresh the page and verify that there is another question displayed on the page.	The app should print a message to the console after establishing a successful connection to the database. and the survey page should print a question (currently the survey page queries a random question from the database)	In the future to implement the final product, the survey page should implement a counter that counts how many questions were answered by the user to be able to present the appropriate questions and know exactly when to display the confirmation page to end the survey.
Homepage icons verification: Run the app using the command `node app.js` and verify that the app was launched successfully with no error messages. After that navigate to the app homepage using the link printed on the console. In the homepage of the app verify that there are icons associated with the action list and the navigation list. After that, verify that the color of the icons matches the colors of the page. finally, verify that each icon is presenting the link associated with it and test the functionality of each link.	The icons in the homepage should be clear and unambiguous represent the associated links.	To enhance the usability of the app the size of the icons could be adjusted with the size of the text based on the user's desire.
Verify the survey web page functionality: Run the app using the command `node app.js` and verify that the app was launched successfully with no error messages. After that navigate to the app homepage using the link	The survey page should present the question with its number. Also it should render a 10 steps slider to enter the answer under the questions.	One improvement that would improve the usability and the user experience is to display the value of the slider on the page so the user know exactly what is the current answer. In addition, it would be helpful to display the total number

printed on the console. From the homepage navigate to ‘take survey’ and then continue to take survey. Now the survey page is displayed. In the survey page verify that the question and the question numbers are displayed cleary on the screen. Also verify that the answer slider is visible and sized properly to fit the page.		of questions to the user.
Verify web page template is consistent for all current pages: Access node forever app.js on flip server. Take note of all header and footer features on home page. Click on each active home page icon. For each respective web page, verify all content for header and footer from home page appears here as well.	Logo, title, navigation bar, size change icons, footer icons, and copyright are all consistent. Username and Date data are not visible on Prescription or Over-the-Counter forms. Username and Date data are consistent for Survey Instructions and Questions.	Find and correct error preventing username and date from appearing in prescription and over-the-counter forms.
Verify use of radio buttons on Over-the-Counter form: Access node forever app.js on flip server. Click on all radio buttons, tab between fields and hit Enter. Confirm color and function consistency.	All radio buttons are functional. Can only select one radio button for each form section. All radio buttons have same hover-highlight color and selection color. Colors are consistent with web app theme.	Performs as expected. Design flows with web app theme colors and style.
Verify use of checkboxes on Over-the-Counter page: Access node forever app.js on flip server. Click on all boxes, leave all boxes unchecked, check only some boxes. Confirm color and function consistency.	All checkboxes are functional. Can check and and all checkboxes at the same time. Can uncheck any and all checkboxes at the same time. All checkboxes have same hover-highlight color and selection color. Colors are consistent with radio buttons and web app theme.	Performs as expected. Design flows with web app theme colors and style.
Verify use of text fields on Over-the-Counter page: Access node forever app.js on flip server. Click on field, type in a variety of different text, delete text, tab out of field, click out of field. Confirm color and function consistency.	Input text is smaller than labels. Text is black and labels are dark blue. The fields take all text options possible. Able to tab in and out of the fields.	The slight difference in color and size for the input text helps to differentiate user text vs hard-coded labels. This style is consistent for all inputs on the form, thus the slight differences do not detract from the web app theme.
Verify use of number fields on Over-the-Counter page: Access node forever app.js on flip server. Tab to and/or click on field, enter number by keyboard, use up and down arrows, try to enter alphabet characters. Confirm color and function consistency.	Number text is black and smaller than form labels. Size of input field is more appropriate for form layout. Can enter numbers by keyboard. Can use up and down arrows to increase or decrease current number in field. Default value is 0. Cannot enter alphabet characters.	Performs as expected. Style is consistent with other input fields. The slight difference in color and size helps to differentiate where the user is meant to input data.
Verify use of date fields on Over-the-Counter page: Access node forever app.js on flip server. Enter date by the keyboard, use up and down	Size of date input is more appropriate for layout of form. Text is black and smaller than form labels, just like the text and number fields. All fields	The slight difference in color and size for the date text is consistent with the text and number fields to better differentiate where the user is meant to

arrows, use calendar GUI, click on X. Confirm color and function consistency.	perform the same functions in the same fashion. Tab automatically moves to next section of the date when finished typing in month, then day, then year. Up and down arrows only work on individual fields of the date, not the entire date. Calendar GUI is very user friendly. X icon clears the date field.	input data. No change needed.
Verify use of dropboxes on Over-the-Counter page: Access node forever app.js on flip server. Click on the down arrow and select a unit, move up and down the list with the keyboard arrows, type a unit into the field. Confirm color and function consistency.	Can tab to field and use up and down keys to scroll through the dropdown lists. Can click on down arrow, expand list, and select item. Cannot enter freeform text. Text is dark blue, but not as large as the labels.	Color and text size are appropriate to reflect user input, like the text and number fields. Put drop down menu items in alphabetical order. Allow for free-form text, up to a limited size.
Verify consistency of radio button and checkbox styling among web pages: Access node forever app.js on flip server. Start from home page and check each linked web page for radio buttons and checkboxes. Confirm color and function consistency of features between web pages.	Only Prescription and Over-the-Counter forms have radio buttons and checkboxes right now. These features are not consistent between the two forms. Prescription foram has not been updated since last round of integration and usability tests.	Alter styling of Prescription form to reflect that of the Over-the-Counter form. Not only the radio buttons and checkboxes, but the styling, colors, and layout of the forms must be consistent.
Verify use of Submit button in Over-the-Counter form: Access node forever app.js on flip server. Access database on phpMyAdmin. Fill out Over-the-Counter form, and click Submit.	Submit form remains highlighted. Form fields to not clear. There is not confirmation message. Record is created in database, but with some field data missing.	Submit button is functional. Enable button to render confirmation page after medication record is written to the database. The user needs an indication that their record was or was not recorded.
Verify that the app is able to connect to the database to record medication records: Access node forever on flip server. Open database repository on phpMyAdmin and open to the medications table in order to view new records. Fill out Over-the-Counter form and click the Submit button. Go to database and refresh medications table to view new record.	New records for each Submit are created. More than half of the fields are recording, however several are still being recorded as Null.	Fix bugs in code to record all fields from form to the database for a medication record.

8. PLAN OUTLINE

The new estimated times mostly reflect the time it took our group to accomplish this past week's tasks. LAst week we re-evaluated our time requirements and how many tasks we thought we could accomplish in a week. As we predicted, the team was a bit more comfortable working with web design code again, and thus out work was much more efficient. Though, we did not fully complete all of our tasks this week, we did work on all of them and completed most of them. See our brun down diagram above for an overview.

Our aim was to complete the main styling of the web app to give the customer a good idea of what the final product will look like for their users. As well, we were also aiming to complete at least one additional user story, which we chose as the medication forms, since the rest of the system is dependent on the user creating records of medications. we were able to complete the Over-the-Counter form, save for a couple remaining database code bugs. It was decided that due to the number of differences between the Over-the-Counter and Prescription medication forms, the additional features and database code required of the Prescription form would best be handled in its own user story. Hence, the team responsible for user story K completed the layout of the Over-the-Counter form in order to send to the customer for approval before doing further work on the Prescription form in the event the customer required any changes. This way, all the mistake could be made in one form and the team would save time and facilitate simplicity and efficiency. The new Manually Record Prescription user story tasks are as follows:

O. MANUALLY RECORD PRESCRIPTION

High priority. This is a convenience for the user when user's mobile camera is not functioning, or the user is utilizing a web browser to record prescription data.

1. Setup database to record prescription objects - complete
2. Create Add Prescription and icon on homepage - complete
3. Create prescription form webpage - needs styling
4. Write code to take user to prescription form from homepage - complete
5. Write code for web app to connect with database - complete
6. Write code to record data to database - fix bugs
7. Design a window for the confirmation from the user. - incomplete

Estimated Time: 30 hours

Due: Thursday, August 23rd.

For the tasks and times assigned below, due to the improvement of our burn down diagram compared to last week, the team is maintaining our estimated times for similar tasks. Estimated times for some coding tasks, particularly regarding the database coding, have been slightly increased.

	Team		
Day	Riley/Johnny	Mohammed/Diego	Notes
M	* L2: Get date to display as long date (1 hour) * K6: Get medications to properly record (2 hours)	* A5: Add Scale to slider (2 hours) * A7: Store displayed question's ID (1 hour)	
Tu	* K7: Design window for the confirmation of user (2.5 hours)	* A7: Make sure survey is x questions long (1 hour) * A7: Ensure random questions are not repeated (2 hours)	x will be determined in meeting with the client
W	* O3: Style prescription form to resemble the over the counter form (3 hours)	* A9: Write code to record answers in DB (1.5 hours) * A10: Create confirmation page (2 hours)	
Th	* O6: Write code to record prescription data (4 hours)	* B1: Design table to record symptoms in database (1 hour) * B2: Design web site to record symptoms (2.5 hours)	
F	* Meet with customer (1 hour) * Integeration and usability testing (1hour)	* Meet with customer (1 hour) * Integeration and usability testing (1hour)	

9. REFERENCES

[1] Henderson. Nov 2015. *10 things you need to stop hiding from your doctor*. [Online] https://www.netdoctor.co.uk/healthy-living/wellbeing/advice/a26010/10-things-to-stop-hiding-from-your-doctor/

[2] Horton, Sarah and Lynch, Patrick H.. Web Style Guide. *Presenting Information Architecture*. [Online] https://webstyleguide.com/wsg3/3-information-architecture/4-presenting-information.html

[3] Nelson. 2010. *The Top 8 Secrets You Keep from Your Doctor*. [Online] https://www.webmd.com/women/features/talking-with-your-doctor-what-to-say#1

[4] Rankin. May 2015. *10 Crucial Questions Your Doctor Should Ask You (But Probably Doesn't)*. [Online] http://lissarankin.com/10-crucial-questions-your-doctor-should-ask-you-but-probably-doesnt

[5] Wikipedia. *Website Wireframe*. [Online] https://en.wikipedia.org/wiki/Website_wireframe