CONNEC+

Sprint 2 Retrospective

Connec+ (ConnectPlus)

Team 9

Cameron Glass, Calvin Henry, Victoria Oldson, Nirali Rai, and Joey Vinyard

What Went Well

User Story 1

- ➔ **Implement services in the front end to request the profile data from the backend.**
  - ◆ Used Angular's HTTP client to make GET requests for user profile data to our Node server.
- ➔ **Implement a route in the backend to read the user data from the database and return it to the client.**
  - ◆ Built a function in the Node server that pulled user information from the database and returned it to the client in an HTTP response.
- ➔ **Create unit tests confirming that the database returns an accurate snapshot of the user data.**
  - ◆ We created unit tests to confirm that the server routes generate the correct output from the database upon querying

User Story 2

- ➔ **Implement services in the front end to request deletion of the user in the backend.**
  - ◆ Used Angular's HTTP client to make DELETE requests for user profile data to our Node server.
- ➔ **Implement a route in the backend to delete the user in the database, and return a success state to the client.**
  - ◆ Built a function in the Node server that deleted the user profile information from the database and returned a copy of it to the client in an HTTP response.
- ➔ **Create unit tests confirming that the database reflects the deletion of the user.**
  - ◆ We created unit tests to confirm that the server routes generate the correct output from the database upon deleting

User Story 3

- ➔ **Implement services in the front end to send updated user information to the backend**
  - ◆ Used Angular's HTTP client to make PUT requests for user profile data to our Node server.
- ➔ **Implement a route in the backend to update the user information in the backend and return a success state to the client.**

- Built a function in the Node server that took an updated user object, and updated the database, then returned a copy of the updated user to the client in an HTTP response.

➔ **Create unit tests confirming that the database reflects the accurate state of the user after a put request has been issued.**

- We created unit tests to confirm that the server routes generate the correct output from the database upon updating

User Story 4

➔ **Implement getting a user's location from the web application**

- We used the browser geolocation permission in order to request a location from the user. This data is then handled in a later task.

➔ **Implement sending the user's location from the web client to the backend.**

- I took the location data received from the browser and wrote the necessary functions to send this data from the front end to the backend which is implemented below.

➔ **Implement server routes to send the user's location to the backend.**

- Implemented a function in the backend that

➔ **Implement unit tests showing that the users location is accurately stored.**

- We created unit tests to confirm that the server correctly updated the database when given a users location

User Story 5

➔ **Add a range selector to the front end that adjusts the map or list accordingly.**

- I added a slider scale with an update button that will adjust the range value of the map in order to zoom it in or out with the user's pin centered.

➔ **Implement finding users based on range from current location in the backend.**

- I added filters in the backend that took into account current location and distance from that location.

➔ **Implement filters based on basic saved profile information.**

- I added more to the filters that took into account users' saved interests.

➔ **Implement filters based on information pulled from social media sources.**

- We query the database to get the nearby users social media data, and then we use hashmaps to find similarities between them in a time efficient manner, removing users that have no similarities.

➔ **Add unit testing for whether or not the correct users are pulled from the database to be displayed.**

- We created unit tests to confirm that the http requests between the server, front end and database were working successfully to pull the nearby users from the database

User Story 6

➔ **Design user component which can be populated with information from the selected user's profile in the database**

- I wrote a popup that positions itself in front of the map or list that displays the selected user's profile image, mood status, and distance away from yourself.

➔ **Refine the list view to have interactive profile cards that can be clicked to display the user profile card**

- In the last sprint, I made a working list view, however it was not set up well enough to implement setting user data. I reworked all of the user cards so that they fit better with populated information, and also implemented the profile cards when clicked.

➔ **Implement unit tests to confirm that the list view shows valid users.**

- I wrote four unit tests to confirm that list view showed to the correct users and information.

User Story 7

➔ **Use Users location to create a pinpoint on the map at the specified location.**

- I used the user's stored location to create a marker on the map at that location, which is then the central point on the map. From then, I created.

➔ **Implement clicking on pinpoints on the map to view the specified users profile.**

- I added a click event that then positions the profile card I previously created in front of the map and populates with the selected user's data.

➔ **Implement a distance filter for the map view, so it does not try to load every user in the database.**

- Calculated the distance from other user's locations in the backend, then returned only users within 3 miles.

➔ **Implement unit tests to confirm that the map view shows valid users.**

- I wrote four unit tests to confirm that list view showed to the correct users and information.

User Story 8

- ➔ **Add missing required text inputs**
  - ◆ I added the missing text inputs that were required for the user to better change their profile information, often times improving the security of the process.
- ➔ **Split content into appropriate sections**
  - ◆ I took all of the settings sections and split them into separate divs. This allows the content to be more organized, and allowed me to use these divs to expand and collapse later on.
- ➔ **Make sections expand and collapse**
  - ◆ I took the divs previously created and added ngIfs to them that would allow them to collapse and expand. This makes the page less overwhelming when opened, and the user only needs to see what they want to adjust.
- ➔ **Improve UI Experience**
  - ◆ I made it so that the expanding and collapsing does not require a certain button to be clicked, the user can click anywhere on the div. I also improved the scalability of many sections to look better on different sized displays.
- ➔ **Frontend Testing**
  - ◆ I did manual testing of each component that was changed to ensure that they are functioning properly. If they were not, a note was made for fixes later in the sprint.

User Story 9

- ➔ **Add Mood Status Editor to Map and List views**
  - ◆ I added an edit button on the mood status that brings up a simple dropdown interface to allow the user to edit their mood status and click update to send it to the database.
- ➔ **Add Invisibility selection to Map and List views**
  - ◆ I added an invisibility section that provides multiple options for invisibility. When an option is selected, the updated selection is sent to the database and updated on all other user's profile.
- ➔ **Implement services to send a change in Map or List view to the backend.**
  - ◆ We took the user input from the buttons they clicked and stored the results in the backend. We were also able to show on the client what they had selected as well as have the information changed based on what the user changes.
- ➔ **Implement backend services to store these settings in the database.**
  - ◆ We wrote a server route to use http requests to store data that is sent to the firebase database.
- ➔ **Implement services to read and store these settings in the local browser storage.**
  - ◆ I wrote code that stored changes in local browser storage and got

information from there when it was available.

- ➔ **Implement services to request user settings from the backend if local storage does not contain them.**
  - ◆ In the event that nothing had been stored in local browser storage, I made the information be fetched from Firebase.
- ➔ **Implement invisibility in maps**
  - ◆ I added a section of code that checks for the invisibility of the user before they are added to the nearbyUser array. This prevents these users from appearing on the map.
- ➔ **Create unit tests to confirm that the database accurately shows changed settings.**
  - ◆ We created unit tests to confirm that the server correctly changed the database

User Story 10

- ➔ **Add drop down menu to navigation bar**
  - ◆ I added a small button that showed all available pages when hovered over.
- ➔ **Implement functionality for dropdown menu buttons to navigate pages**
  - ◆ I used basic http routing features to allow the navigation menu to work.
- ➔ **Implement logout button**
  - ◆ I added a logout button to the routing and fixed the function to allow the user to effectively log out. I also updated the routing so that a logged-in user sees different options than those who are logged out. This improves security and user experience.
- ➔ **Add unit tests to confirm that routing works.**
  - ◆ Used Angular's RoutingTestModule to confirm that attempts to navigate to other pages were successful

User Story 11

- ➔ **Implement a route in the backend to store data from the Facebook API in the database**
  - ◆ We wrote code to query the Facebook graph api using a node module, and then use http requests from our server to store them in the database
- ➔ **Implement a route in the backend to allow the client to request Facebook data**
  - ◆ We implemented a route the allowed the client to query the server and get the facebook friends using http requests.
- ➔ **Display the user's Facebook login status in the web client**
  - ◆ I added "Connected" and "Not Connected" messages on all social media sections that show the current status of the user. These swap with an ngIf based on the facebook login method return.

- ➔ **Implement the ability for a user to disconnect Facebook from the app**
  - ◆ When the user clicks disconnect, it logs out from the facebook app
- ➔ **Create tests to confirm that user's Facebook login status is correctly displayed**
  - ◆ Used Angular's Testing modules to confirm that the status properly changed when the login status of the user changed.

User Story 12

- ➔ **Create Frontend Interface for choosing classes**
  - ◆ I used the basic functionality of the Blackboard implementation in order to create an interface that populates with the API information in order to prevent the main Blackboard section from becoming too cluttered.
- ➔ **Implement receiving class lists from Purdue API**
  - ◆ Used the Purdue.IO API to receive a list of classes that a user could possibly be enrolled in, then displayed them in the frontend interface.
- ➔ **Implement adding classes to User profile**
  - ◆ Made a POST request to the backend containing the data for a class that the user selected, then stored it in the database.
- ➔ **Implement removing classes from User profile**
  - ◆ Made a DELETE request to the backend that removed the selected class from the user's list of classes.
- ➔ **Implement backend route to send Blackboard data to firebase**
  - ◆ Created functions in the backend to listen for the GET/POST/DELETE requests from the client, and modify the database appropriately.
- ➔ **Create unit test confirming user's classes are updated**
  - ◆ We created unit tests to confirm that the server provided the correct response, and correctly updated the database when it was given a users classes

User Story 13

- ➔ **If I have already connected to my account, fill Twitter Username Box with valid username**
  - ◆ We stored the twitter screen name upon connection, and upon loading the settings page we query the database for the name.
- ➔ **Implement retrieving user data from the Twitter API**
  - ◆ We wrote database routes to allow the front end to query for the twitter friends list
- ➔ **Display message confirming the validity of the given Twitter Username**
  - ◆ If querying the twitter API resulted in an error, we displayed an error to the console.
- ➔ **Implement backend route to send Twitter data to Firebase**
  - ◆ Upon successfully querying the twitter api for friends, we used our server to store the friends list in the database.

- ➔ **Create unit test confirming user's login status is correctly displayed**
  - ◆ We created unit tests confirming that the routes between the server and front end worked as intended.

User Story 14

- ➔ **Implement error handling for when a user provides invalid input when attempting to change their email or password**
  - ◆ We checked if the given email and password was valid for the user, and then checked if the new email was the same as the old one, if it was not valid and/or the emails were the same, we sent an error message to the client.
- ➔ **Implement error handling for when a user provides invalid data when attempting to change their password**
  - ◆ We checked if the given password was valid for the user, and then checked if the new password was the same as the old one, if it was not valid and/or the passwords were the same, we sent an error message to the client.
- ➔ **Implement error handling for when a user provides an incorrect email or password when deleting account**
  - ◆ We checked to see if the email and password were valid for the user, if not, we sent an error message to the client.
- ➔ **Implement error handling for when a user provides invalid data during profile creation**
  - ◆ We checked to see if the required fields were filled as well as if the information provided was valid, if not, we sent an error message respective to the error caused.
- ➔ **Create unit tests to confirm that error messages appear**
  - ◆ We created unit tests to confirm that the proper messages appear, and that they correspond to the action that the user performed.

User Story 15

- ➔ **Implement success messages for when a user successfully deletes their account**
  - ◆ We checked if the email and the password was valid for the user, if it was valid we use the delete user method from firebase as well as one generated specifically to delete the user that was created
- ➔ **Implement success messages for when a user successfully changes their account email address**

- We checked if the given email and password was valid for the user, and then checked if the new email was the same as the old one, if not then we changed the email using a firebase method.

➔ **Implement success messages for when a user successfully changes their password**

- We checked if the given password was valid for the user, and then checked if the new password was the same as the old one, if not then we changed the password using a firebase method.

➔ **Implement success messages for when a user successfully creates a profile**

- We check if certain fields are filled out and that the contain valid information, if so we call a create user method that creates the user and sends the information to the database

➔ **Create unit tests to confirm that success messages appear**

- We created unit tests to confirm that the proper messages appear, and that they correspond to the action that the user performed.

What Did Not Go Well

For this sprint, we did a much better job at determining the workload for the sprint, and better assigned hour estimates so that each member actually had a fair workload. However, although we were able to better complete our tasks, we failed to allocate the appropriate amount of time to thoroughly review our Acceptance Criteria, and therefore missed some errors that were caught during our Sprint Review. Those small errors are described below.

User Story 2

- ➔ **Implement a route in the backend to delete the user in the database, and return a success state to the client.**
  - ◆ We forgot to include a route in this task to delete social media information of users that had been logged in.

User Story 12

- ➔ **Implement error handling for when a user provides invalid data when attempting to change their profile data**
  - ◆ We forgot to check if the first and last name fields were blank and if the age was a negative number. To fix this we need to modify the existing checks to account for these issues.
- ➔ **Implement success message for when a user successfully updates their profile information**
  - ◆ We had an inaccurate check for name fields and age fields, causing the success message to be sent even when it should not have been. To fix this, we need to modify checks to account for potential issues.

How To Improve

- ➔ The main issue that caused us to not get things complete in this sprint was failing to thoroughly check our acceptance criteria. This caused an error in user stories 11 and 12, as well as 2. In the future, we plan on making sure that we confirm our acceptance criteria are met before the sprint review.
- ➔ This sprint we also had some miscommunication with Github. We did not deal with all of our merge conflicts as well as we could have and this resulted in us losing some code that had already been written.