Report: HTTP Requests / Routes

In the report, include a list of HTTP routes implemented in the server, and the mock server method that they replace. For each, briefly describe:

- The *verb*, *target*, and *body* of the request.
  - Example: POST /feeditem [feeddata], where [feeddata] is a JSON object describing a Feed Item.
- What the HTTP request does, and what mock server method it replaces.
- What resources the HTTP request creates/modifies/etc.
  - Examples: "Updates /user/:userid", "Creates a new /user"…
- Who is authorized to use the request
  - Example for PUT /user/:userid userdata: "A user with the specified :userid can issue this request. Administrators can also make this change on any :userid."
  - Example for POST /feeditem feeddata: "The body of the HTTP request contains an "author" field containing a user ID. The requester must have the same user ID."
  - If you have a hard time explaining it generally, feel free to include examples. Example: "A user with ID 4 can issue a PUT /user/4, but a user with a different ID cannot."

Report: Special Server Setup Procedure

If your server has any advanced features that require additional setup besides npm install and node src/server.js, include a section in the report that describes how to set up these features. Include details on how to tell if the feature is working properly.

Report: Individual Contributions

Include a section that describes each startup founder's contribution. Each startup founder should be responsible for at least one product feature and its HTTP route(s). Name the feature and the HTTP routes that the founder implemented / was responsible for.

Report: Lingering Bugs / Issues / Dropped Features

If your application has any lingering bugs, issues, or missing/dropped features, include a section listing these.

Navbar - All
BUG: The navbar search functionality is still not implemented but it redirects the user to the job board with all job listings.

Sidebar - Nicholas Achin

We have properly linked the user's associated projects in the sidebar. Now the projects listed on their profile page are the same as the sidebar for the single user Jane.

Sidebar HTTP Requests

- GET /users/:userid/sidebar-projects [projects], where [projects] is a JSON object describing the associated project objects for a user.
  - The request pulls all of the projects from the associated user and renders the projects from the "project" JSON object. This request replaces the emulated return of getProjectPillData(userid, cb) in the mock server. This data is used to make the project pills in the Sidebar.
  - This request does not create or modify any resources since it is simply a GET request.
  - Only the user with the specific ":userid" can access that user's associated projects. An individual project could be in multiple people's sidebars since multiple people can work on one project, but each user has their own projects in the sidebar.

Main Feed - Nicholas Achin

For this submission, I fixed all of the rendering issues I was previously having with the Main Feed and can now properly render job and notification feed items into the main feed for an associated user.
DROPPED FEATURE: We have removed the ability to filter out feed items and removed the possibility of following other projects and having relating notifications for them.

Main Feed HTTP Requests

- GET /feed/:userid/notificationsitems [notificationItems], where [notificationItems] is a JSON object describing the Notification Feed Items for a user.
  - The request pulls all of the notificationItems for the associated user to the user's Main Feed. This request replaces the emulated return of getNotificationFeedData(user, cb) in the mock server.
  - This request does not create or modify any resources since it is simply a GET request.

    - Only the user with the specific ":userid" can access that user's associated notificationItems. An individual notificationItem could be in multiple people's feeds, but each user has a unique feed.
- GET /feed/:userid/jobitems   [jobItems], where [jobItems] is a JSON object describing the Job Feed Items for a user.
    - The request pulls all of the jobItems for the associated user to the user's Main Feed. This request replaces the emulated return of getJobFeedData(user, cb) in the mock server.
    - This request does not create or modify any resources since it is simply a GET request.
    - Only the user with the specific ":userid" can access that user's associated jobItems. An individual jobItem could be in multiple people's feeds, but each user has a unique feed.

Profile Page - Ryan Minichiello

The profile page now has the routing requesting (GETing) Jane's profile information from the server, and it being served back to the page via user/:userid. The profile also has the ability to send messages from the logged in user to the profile being observed by adding a new messaging object to the chat messages array.

HTTP Requests

- GET '/user/:userid' pulls the user json object, containing all the information needed for the user's profile page.
- Does not modify the database, and all registered users have access to the page.

Project Page - Thomas Cloutier

The project page now pulls project data for each individual project and allows the individual user interact with the page by applying by clicking the apply button.

HTTP Requests

- GET '/user/:userid/open/pos_id/:pos_id' [positions], where positions is a json object containing all the open positions for the project
  -Pulls out all the projects associated open positions to display on the project page
  -Does not modify the existing database in any way
  -All registered users have access
- GET '/user/:userid/filled/pos_id/:pos_id' [positions], where positions is a json object containing all the filled positions for the project

-Pulls out all the projects associated filled positions to display on the project page
-Does not modify the existing database in any way
-All registered users have access

- GET '/user/:userid/projectid/:projectid' [notifications], where notifications is a json object containing all of the recent updates for the project
  -Obtains all of the most recent project notifications for this specific project
  -Does not modify the existing database in any way
  -Access available to all registered users
- GET '/user/:userid/project/:projectid' [project], where project is a json object containing all the information for the specific project

Create Project Page - Ryan LeCours
Bug Fixes:
-Maximum number of tags has been set to 5
-Maximum number of positions has been set to 5
-Information provided in each field now saves to and updates the database
-Create Project button now sends user to the project page
-Added Remove Button in tag section so the user can remove unwanted buttons
-Tags now show inline, in their own respective divs

Bugs:
- Project page does not yet retrieve data from database yet. When this bug is fixed the user will be brought to the project page of the project they just created.
- I will be adding a remove tag button that will remove an added tag

Inbox - Caroline Kim

Bugs

There has been issues with passing a function as a prop for child component and making onClick work from that. Due to that issue, I've only limited the inbox page to have one chat on there. Making onClick function work and being able to process messages that is associated with the certain chat that is clicked on will be fixed ASAP.

When send message button is clicked, it wasn't creating a new message. Instead, it pulled the next message from the messages that was in the database. I tried implementing the POST for it, but couldn't get it to work. Once I can get the function in, I'll try implementing POST again.

HTTP Requests

- GET '/user/:userid/inbox/:inboxid'
    - Retrieves data for the user's inbox with certain user id and the inbox id that is associated with that user
    - This replaces the emulated return of getInboxData method
- GET '/user/:userid/chats'
    - Retrieves the array of chat objects that is associated with a user with that user id
    - This replaces the emulated return of getChatListItems
    - getChatListItems method was originally getChatArrData which had parameters for array of chat id. But now it receives the user id instead.
- GET '/user/:userid/messages/:chatid'
    - Retrieves the array of message objects that is associated with the certain chat id
    - This replaces the emulated return of getMessageListItems
    - getMessageListItems method was originally getMessageArrData which had parameters of array of message id. But now it receives the user id and chat id instead.
- GET '/user/:userid/message/:messageid'
    - Retrieves the message object that is associated with the certain message id
    - This replaces the emulated return of getMessageData
- GET '/user/:userid/chats/:chatid' [chats]
    - Retrieves the chat object that is associated with the certain chat id
    - This replaces the emulated return of getChatData

- Only the user with specific userid can access the object being returned from GET

Job Board - Nicholas Achin

Prior to this submission, the Job Board was in tough shape so I reformatted some of the CSS and HTML to use more of the stable elements from the Main Feed. The filter bar now uses the Main Feed's filter bar styling, but the ability to actually filter has not been implemented into our application. In order to render all of the job items into the Job Board feed, I added a field to the users that links to "allJobItems" that contains an array of every "jobItem" in our database.js. This allows me to easily render all job items into the feed.

Job Board HTTP Requests

- GET /job-board/:userid/jobitems [allJobItems], where [allJobItems] is a JSON object containing all the “jobItems” in the entire system.
    - The request pulls all of the jobItems in the database and populates the UI. This request replaces the emulated return of getAllJobs(userid, cb) in the mock server.
    - This request does not create or modify any resources since it is simply a GET request.
    - All users can access these jobItems since all jobItems are open to all users.