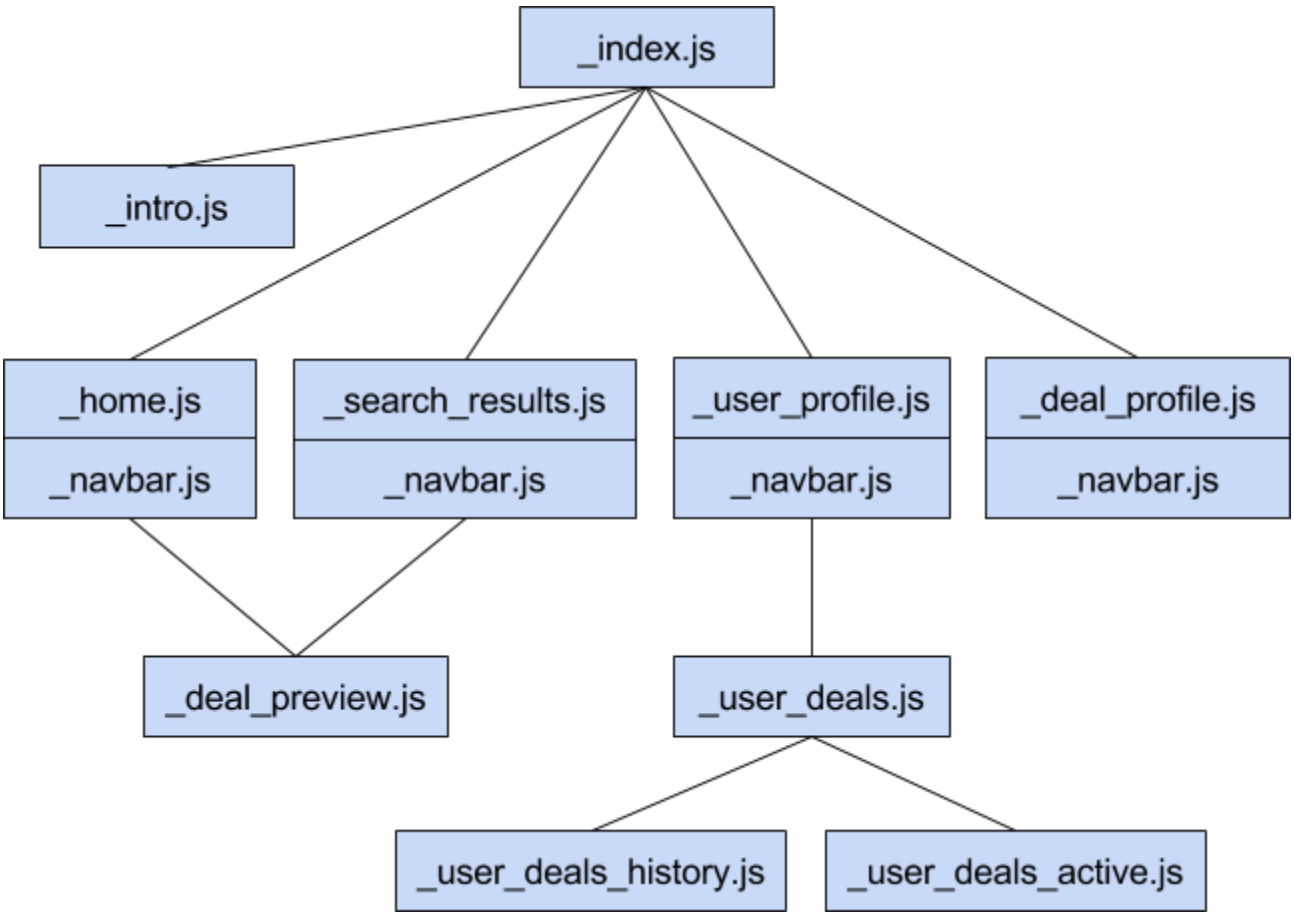
Updated Structure

After much deliberation, Ella, Katie, and Helena restructured the project to effectively manage all components. We originally thought about making Pair'd a single page app (without routing), in which case, we would use feature.js as a placeholder for any component to be in, depending on what 'page' we wanted to show to the user (home, search results, user profile, etc.), but now with our new structure, we have a newly revised DOM with the accurate filenames reflected below. The underscore in front of the filenames is because we were refactoring and didn't want to mess with our previous files (which have since been relocated into the new directory components/archive/).

The reason why navbar.js is present (in the image below) under the home.js, search_results.js, user_profile.js and deal_profile.js, is because in our index.js file, we have class App as the primary component in our application and it includes navbar. All of the pages, except intro.js, have navbar at the top. The image below attempts to



explain which pages/routes will direct to what. For example, the user profile includes

user deals, which is the widget that holds both the user's active deals and deals history (all within the same page and url).

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."*

Below is a list of all the server routes we plan on implementing.

GET:

- getDealData should retrieve deal data from the db using GET /deal/:dealid returns 200 status code and resolved Deal object in body
- getUserData should retrieve user data from the db using GET /user/:userid returns 200 status code and resolved User object in body
- getSearchResult GET /tag/:tagid returns 200 status code and an array of JSON objects containing deals with tags that match the input tag.

POST:

- postDeal should post a new deal and uses POST /deal/:dealid returns 200 status code and resolved Deal object in body
- Upload image

PUT:

- updateUserData should update the specified user info and uses PUT /user/:userid, returns 200 status code and resolved User object in body
- updateDealData should update the specified deal and uses PUT /deal/:dealid, returns 200 status code and resolved Deal object in body
- Pair two users on a deal should update the specified deal with the paird user and uses PUT /deal/:dealid/dealpair

- Add to active deals should update the specified user active deals and uses PUT /user/:userid/dealsparticipated, returns 200 status code and resolved User object in body
- Add to deal history should update the specified user deal history and uses PUT /user/:userid/dealsposted, returns 200 status code and resolved User object in body

DELETE:

- deleteDealData should remove a specified deal and uses DELETE /deal/:dealid returns 200 status code and a blank response for success.
- deleteUserData should remove a specified user and uses DELETE /user/:userid returns 200 status code and a blank response for success.

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.

N/A

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.

Ella: Made _user_profile and _deal_profile toggle edit buttons when user is view their own profile vs viewing another user's profile. On _user_profile the user can now successfully edit their name which changes the database. A lot of paired programming was done with Helena regarding debugging, merging branches and brainstorming. I also added key functions to the src/server.js like reset database and JSON error translation.

Katie: Pair programming with Ella to work on pulling information from the database into fields on the user profile, updated deal information in the database (they previously all had the American Eagle deal info) so further references to the database would be correct, added initial unixToTimeString function to convert Unix time to human readable format in util.js file, for the active deals part of the user deals widget dealTitle pulls from the database, corrected routes and syntax between intro and homepage (making sure to use onClick and the fat arrow for all buttons, instead of the Link tag), added functionality to the wordmark and glyphicons in the navbar and implemented various

onClick methods, incorporated much of John's example code for implementing search from Facebook (but search results is currently not completely functional), additional pair programming with Helena on fixing some routes according to information from previous course workshops and debugging when information on the dealProfile wouldn't pull from the database.

Helena: I worked with Katie and Ella to refactor the project, which happened twice: once when we made a mistake and thought we should make a single page app with no routing, only components re-rendering, and the second time when we talked to John and realized why this was such a bad idea. I debugged a lot of problems to do with React, which often involved rewriting components to make sure they used React correctly, so I have touched/refactored most of the files in this submission. They are still in progress. My other major contribution was in rewriting a few of the server methods to pay attention to (a)synchronicity, because I have experience with this concept from a past internship and was realizing that there were errors being caused by our inattention to this detail.

Brent: I worked on the getSearchResult function which is mainly used to find deals associated with a given tag. This function sends a get request to the server with the tag as the parameter. The server then looks for the deals associated with the given tag and returns an array of all deals that match the specification. I added routing for the search result page which allows for the navbar to pass 5 tags over to the search result page using the url. When the search result page renders, it calls getSearchResult and receives an array of JSON objects with deals that match the first tag on the tag bar (eventually all tags on the tag bar). Due to time restraints the search result page only prints the dealTitle attribute of the deal JSON object into the console (eventually this page will dynamically render deals from an image path in the JSON object.

Guan (**Raymond)**: I worked on some of the issues from last submission, mainly trying to get the post deal function to work to add a deal to the database and render it, and also worked on some of the rendering and routing issues we were having. In regards to the work pertaining to this submission, I updated the git repository to have client and server, and created a server.js file that uses the express server and listens on port 8080. I added authorization to the server, and implemented the getUserData HTTP route with authorization.

Wesley: I worked on issues from the last submission, mostly involved with pulling and updating the database. I added routing for home, dealpreview and dealprofile. I let deal profile, user active deals, user active history, and deal preview pull its information from

the database. I also rewrote how user profile pulled from the database to pull more efficiently(was caught in a loop). I implemented the Pair Me! button in the deal profile which attempts to write to the database the newly paired user for that deal. I added the edit overlay for deal profile which attempts to write to the database the edited deal. In server, I implemented the getDealData HTTP route and postDealData HTTP route. I added the deal schema for posting a new deal and postDealData uses validation to check that schema. In util.js, I rewrote unixTimeToString to return and display the appropriate date format dependent on input date. Similarly, I wrote calculateExpirationDay to return and display the different expiration date formats dependent on input date.

Report: Lingering Bugs / Issues / Dropped Features

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

We decided to replace the ability to delete a single tag from the tag bar with a button that clears all of the tags. (problem occurred when trying to delete an individual item)

The first search input that routes to /search/ does not get recorded. (Hope to fix this by next submission)

Search result page currently prints the search results into the console (retrieves these results from the server). Will eventually need to render these results onto the screen, but due to time restraints this was not possible.

Tagbar renders on all pages after the first search. This should only be rendered while on the search result page.

Only the first tag is used to decide the deals being searched for. The next version should include all tags in deciding the deals to render.

Our current submission does not include the intro page, unless you click on the logout glyphicon in the top right corner. We will deal with this in the next submission, because the intro page (Pair'd title, two buttons to continue as guest or login, and blurb with pears background) is a special case without the navbar.

Since we are still catching up on restructuring and introducing routing again, a lot of work was put into how components should be communicating and handling data. Features, such as editing the profile data and deal data, which was at a halt before, is now able to be implemented. Many things are left to be done however, such as finishing

editing all of the user's information, payment information, and the information of each deal.

We have removed a few fields from the database file in client, such as specific properties of deals, that could be implemented in Pair'd 2.0 (not in this course).

When a user tries to pair a deal and a modal popups up, we have removed the expectation that the user will be emailed with further details about how to claim their deal. For our current Pair'd submission, the user will just be notified that they have been paired with the modal and then the deal will reflect this change on screen (no email will be sent to either party).

After pairing a deal, the deal profile is updated with "Currently paird with userID". Instead of userId, this should be the user's full name.

We do not have something like ErrorBanner implemented in Pair'd yet, so if any errors occur, they are just in the console of Chrome Developer tools.

Some http requests are not implemented such as Pair two users on a deal, Add to active deals, Add to deals history, Upload Image. (hopefully fixed by next submission)

XML http requests are not done because not all of our server functions are fully functional and implementing them will cause more problems.

Deal profile's tag buttons do not go to the appropriate search pages.