Documentation

Getting Familiar with the Code:

1. **Login, Register, Logout**

   When you first run the website, you should be greeted with a Login screen. As you may not have an account yet, you can simply press “Create new account.” This will bring you to the registration page. You will enter an email, username, password and will also be asked to confirm your password. There will be alerts on the screen if any of these fields do not meet the guidelines. These can be found in /src/RegisterPage /RegisterPage.js and include ensuring the email is valid (using a built-in JS email validation package), ensuring the username is between 3-20 characters, ensuring the password is between 6-40 characters and that the passwords match. Any of this criteria can easily be changed if necessary. Once you hit create account, an API call will be made to register your account and save your information in the database. You can also coordinate with the back-end team to require certain formatting for the database entry as well. Right now, the only restriction the backend has is that no emails can be repeated as they are the primary key.

   After creating an account, the website will automatically take you to the login screen and display a message “Registered, please login.” You will also notice that there is a “Forgot Password” button on the login screen. If you click it, the web portal will take you to the password reset page where you can enter an email address associated with an account. Note that if the entry is not a valid email address, a warning message will be displayed. This functionality has not been fully implemented yet. It needs to be integrated with the backend. Now, you can sign in using the email and password you registered your account with. Another API call will be made to validate your input. If unsuccessful, you will see a message that you entered invalid credentials. Note: you should never reveal which field is incorrect on a web-app as it will make it easier to hack into accounts by narrowing down the errors. If your login attempt is successful, you will be greeted by the “Dashboard Home” page. Note that if you try to access the landing page without logging in, say by typing into your address bar “localhost:3000/admin”, you will be re-directed to “localhost:3000 /login,” as this is a restricted route. This was implemented using the PrivateRoute component found in /src/components/PrivateRoute.js.

   Once the dashboard redirects to the home page, you will be greeted by a welcome message and a logout button. Pressing the logout button will redirect the browser back to the login page. As for now, there are issues with the CSRF token provided by the login API expiring. Therefore, the logout button right now does not make an API call but simply removes the user from the Local Storage. This logout function can be found in /src /services/AuthService.js and should be fixed as soon as the back-end problem is fixed.

   So now that we know what is happening in regards to the user experience, what is happening at the code level? The login, register and logout functionalities were implemented using Redux. Redux is a pattern and library for managing application state, using events called “actions.” It serves as a centralized store for state that needs to be used across the entire application, with rules that the state can only be updated in a predictable fashion. In addition to Redux, we also use Local Storage, React Router, Axios and React.js hooks. Local Storage is a type of web storage that allows JavaScript sites and apps to store and access data right in the browser with no expiration date. React Router is the standard routing library for React. It keeps your UI in sync with the URL. Axios is a library that helps us make http requests to external resources. Rather than explain in detail every file I added for this functionality (as it is a lot), I will provide a links to very similar examples that I found explained this topic very well: Example 1 and Example 2.

2. **Upload**

   If you navigate to the upload tab, you will see a Select File(s) button that allows the user to select a file from their computer. Note that only csv files are supported. The goal of this component is to allow researchers to upload their own observations collected in a CSV to the database. The web portal will then display these observations on the Visualize Page by making a GET call from the database.

3. **Visualize**

   After selecting “Visualize Data” on the navigation bar, you will see a table with two tabs: Sample and Baby. These tables have been implemented using the “react-table” package and can be found in the /src/components/Tables folder. The data displayed in these tables currently is being pulled from a database. The API contains the following data from the survey responses: BabyID, SampleID, Color, Sick Days, GI Stress and many more. Let’s further discuss each table individually:

   a. **Sample:** The sample table displays data about each specific sample collected by mothers. The color values are extracted from the stool sample by researchers. There is an arrow on the left side of every row that allows the user to expand the row. In the expanded component, the user can see more detailed responses collected from survey forms. There is also a download button for each of the individual samples as well as a bigger button on the top right to download all the sample data.
   b. **Baby:** The baby table consolidates the sample data by Baby ID. The two columns displayed are Baby ID and Sample Count (number of samples for this particular baby). The expandable row for the table displays time plots of the metabolite and microbiome data. The x-axis for these graphs is the Sample ID for the corresponding baby. The y-axis remains the percent composition. Each line on the metabolite graph represents how that metabolite has changed from sample to sample while the microbiome graph represents how that microbe has changed from sample to sample. Since there are numerous lines on each graph, we thought it would be helpful to be able to eliminate them, one at a time. The user can simply click on the labels in the legend and the line will disappear. It can be pressed again to be shown once again. The label will be crossed out in the legend if it is not currently present on the graph. There is also a download button for each of the individual babies as well as a bigger button on the top right to download all the baby data.
   c. **Demographics:** The demographics table joins the results from the Infant Nutrition Onboarding and Infant Nutrition After Birth surveys from Qualtrics. To access these surveys, contact Prof. Johnson to add you to the survey and login to Qualtrics with through your Cornell credentials. The survey data is joined by baby ID which is retrieved from the backend given the email that is used in the survey. Not all of the Infant Nutrition Onboarding surveys have a corresponding Infant Nutrition After Birth survey.
   d. **Summary Demographics:** The summary demographics table takes some of the demographic information from the Qualtrics questions used in the surveys from the Demographics tab and gets the count of each answer choice.

   There are also several components that are standard on the tables:

i. **Filter Data:** The filter feature is a checkbox that is present in the header of both tables. By checking the box, a new row will appear in the table that allows users to search by one or more columns at a time. The search will update the table for every character that is typed in the search bars. The feature was added using a component from “react-table.” There is no filtering in the Summary Demographics Tab.
ii. **Sort Data:** The sorting feature that is present in both tables was also imported from “react-table.” Next to each column, there are two arrows that can be clicked to sort the table. The double arrow means the table is not sorted, the up arrow means the column is ascending and finally, the down arrow means the column is descending. One can be sorted at a time.
iii. **Expand Row:** Both tables have a small arrow on the left side of each row. Clicking on this arrow will expand the given row and it can be pressed again to close it. Any number of rows can be expanded at once on a given page. These rows are implemented with renderSubComponent functions found in the /src/views/Visualize.js file. There is no expanded row in the Demographics tab.
iv. **Download:**
    1. For the Sample Table, the downloading feature allows users to download a stool image from the database by inputting the sample id for the sample.
    2. On the Sample and Baby pages, there's a download button to the right of the filter, which allows users to download a . csv file for all data in the database. (It isn't affected by the filter. If you see some weird record in the CSV, those are records inputted for testing(we haven't dealt with that yet, please ask Prof. Johnson for further instruction on how to do that). The Download button on the Demographics tab only downloads the Qualtrics data. On the Summary Demographics tab, each dropdown has a Download button the download the CSV for the corresponding bar graph.
v. **Pagination:** The pagination component seen being used for both tables is again part of the “react-table” package. It is combined with Pagination components from “react-strap” to create the UI. The current capabilities include: changing the number of rows per page, displaying the current and total number of pages, jumping to the first and last page, going back or forward one page (if possible), and finally jumping to any page using an input box.

Tasks that Remain

1. **Fix Logout:** As aforementioned, the logout function currently does not make a call to the backend, but rather just updates the Local Storage. Once the CSRF token issue is fixed, this can be changed. This will be an easy change as the code is already written, I just have commented it out. The existing code may have to be changed if any changes are made to the back-end API. It can be found in file /src/services/AuthService.js on line 36.
2. **Improve Security:** An extra layer of protection should be added for the login/register functions, such as encrypting the passwords before sending them to the database

Tips and Tricks

1. **API Documentation:** Backend-Offline is the team that supports the dashboard. The documentation for their APIs can be found on their confluence page.
2. **React Stateless Components:** Every component that I added to the dashboard, I made a stateless functional component. This tutorial compares stateless and stateful React components and explains the advantage of stateless. Note: there are some components in the dashboard that already existed as stateful components and have not been changed.
3. **React Hooks:** A tutorial can be found here.
4. **React-Table:** Documentation and examples for react-table can be found here.
5. **React-CSV:** The GitHub including documentation for this package is linked here.
6. **React-ChartJS-2:** This package was used for the graphs seen on the Visual page, so I will link the documentation.
7. **Argon Dashboard:** The framework of the dashboard was built using Argon Dashboard components.
8. **Production:** Once the dashboard is ready to be put into production, there is an existing domain here: https://diapercohort.com/admin/upload
9. **Need further help?** I am happy to help answer any questions that might still remain. My school email is: mmd272@cornell.edu and my personal email is: meredith.dobrzynski@gmail.com. You can also contact Eric Perez [Spring 2022] at eric9650@gmail.com for anything related to Qualtrics.

File Structure

Here's the file structure tree for the project. Note, the structure starts at the src folder found in the dashboard-frontend folder.

src

actions

auth.js Login/register/logout functions

message.js Response from API calls

types.js Type of actions being performed

components

Graphs

BabyGraph.js Generate metabolite and microbiome line graphs

SampleGraph.js Generate metabolite and microbiome bar graphs

SummaryDemographicGraph.js Generate summary demographic bar graphs

Tables

BabyTable.js Generate table for samples grouped by baby

DemographicTable.js Generate table for Qualtrics Demographics data

SampleTable.js Generate table for sample data

SummaryDemographicTable.js Generate table for Qualtrics Summary Demographics data

PrivateRoute.js Limit access when user is not logged in

reducers

auth.js Updates isLoggedIn and user state

index.js Combines message and auth reducers

message.js Updates message state when a message action is dispatched

services

AuthService.js API calls to login/register/logout

store.js Bring actions and reducers together and hold the app state

views

Index.js Home page for dashboard

LoginPage.js Page for user to login to dashboard

RegisterPage.js Page for user to register for an account

ResetPasswordPage.js Page for user to reset the password

Upload.js Page for user to upload a file

Visualize.js Page for Sample, Baby, Demographics, and Summary Demographics tables

index.js Root that acts as a container for all other components

routes.js Define routes to keep the URL in sync with the UI