SomnoHealth Final Report

Client: Chris Crowley, Dave Tobler
Advisor: Dr. Wendy D. Fisher
Team Members: Gustavo Alves, Giovanni Paladino, Trevor Lewis, Jaren Peckham, Matthew Kline
6/16/2021

Introduction

SomnoHealth is a company stationed in Golden that produces the EverSleep device. This device is a wearable wrist-strap device that is used to track information regarding the user's sleep throughout the night. This device will measure things such as blood oxygen levels, heart rate during sleep, apneas that occurred during the night, etc. This data is presented to the end-user using a web portal.

Product Vision

- Make improvements and enhancements to the web portal which allows users and wellness coaches to more accurately and easily read the information presented.
- Create a group statistic that will allow wellcare providers to see how users in their group are doing, and seeing if any of them are making improvements.
- Create an individual statistic that will show the user if they are making any improvements in regards to their sleeping habits (i.e. spending more time in bed, getting more sleep, using a CPAP machine, etc.).

Requirements

Functional Requirements

- Enhancing the current wellness coaches' side of the application. Work with the different parts of the portal that does not work exactly as intended as well as making the data and graphs easier to read.
- Create an HTTPS domain for the application/website to run on for security purposes.
- The web portal needs to have more accessibility as well as be able to show the database trends.
- The client end application shows the database, the current tables divided by day for analysis. It also contains the survey information for each of the users and will be able to see trends in their whole group, as well as for each individual. Each day will have different coaching information for how to get better sleep results.
- The user end application only shows the user their information, trends, and coaching for each night they record data for. They are also able to answer survey questions based on their repeated daily activity as well as a survey based on that day's events to better enhance the coaching provided.
- Ensuring that wellcare professionals have access to all new features that fall within their permissions.

Non-Functional Requirements

- Making sure that all modifications and enhancements to the portal are readable and intuitive.
- All elements are aesthetically pleasing.

Potential Risks

- Crashing the web portal for some extended time.
- Deleting the database through a bad query.
- Security and privacy breaches, ending with the release of private user information.
- Reducing the readability or clarity of the data.

System Design

This diagram shows how data travels from the user and their EverSleep device to the EverSleep Portal.

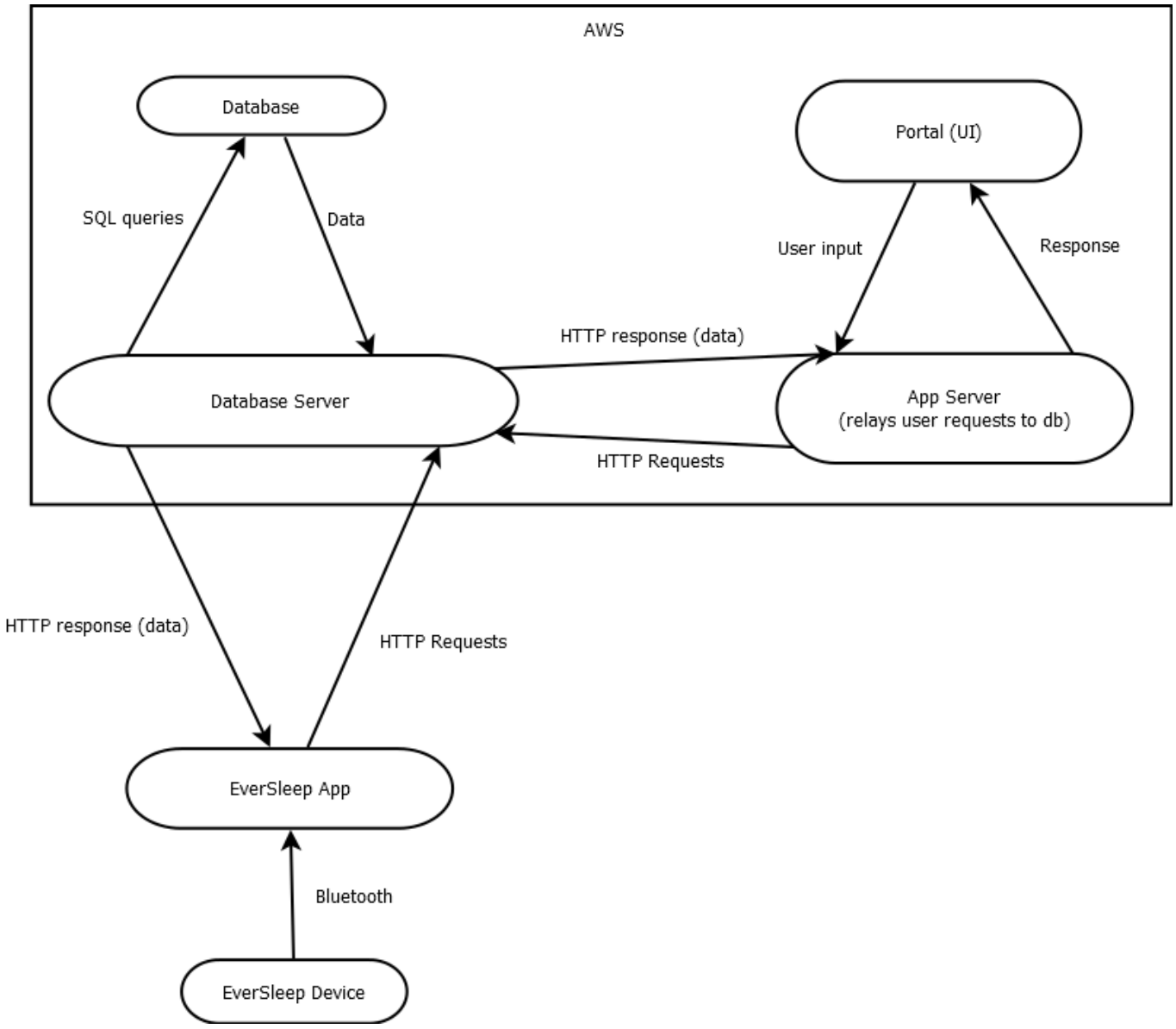

Figure 1: General Overview of the Structure of EverSleep

As Figure 1 shows, the EverSleep device communicates with the EverSleep App, which then transfers the information to a database server. The database server then transfers that data to a

database using SQL queries, and it uses HTTP responses to transfer information to the app server, which is what is used to relay user requests to the database itself. The app server is the connection point that the web portal uses to present any information to the users. All of the servers, including the web portal, are all hosted on AWS.

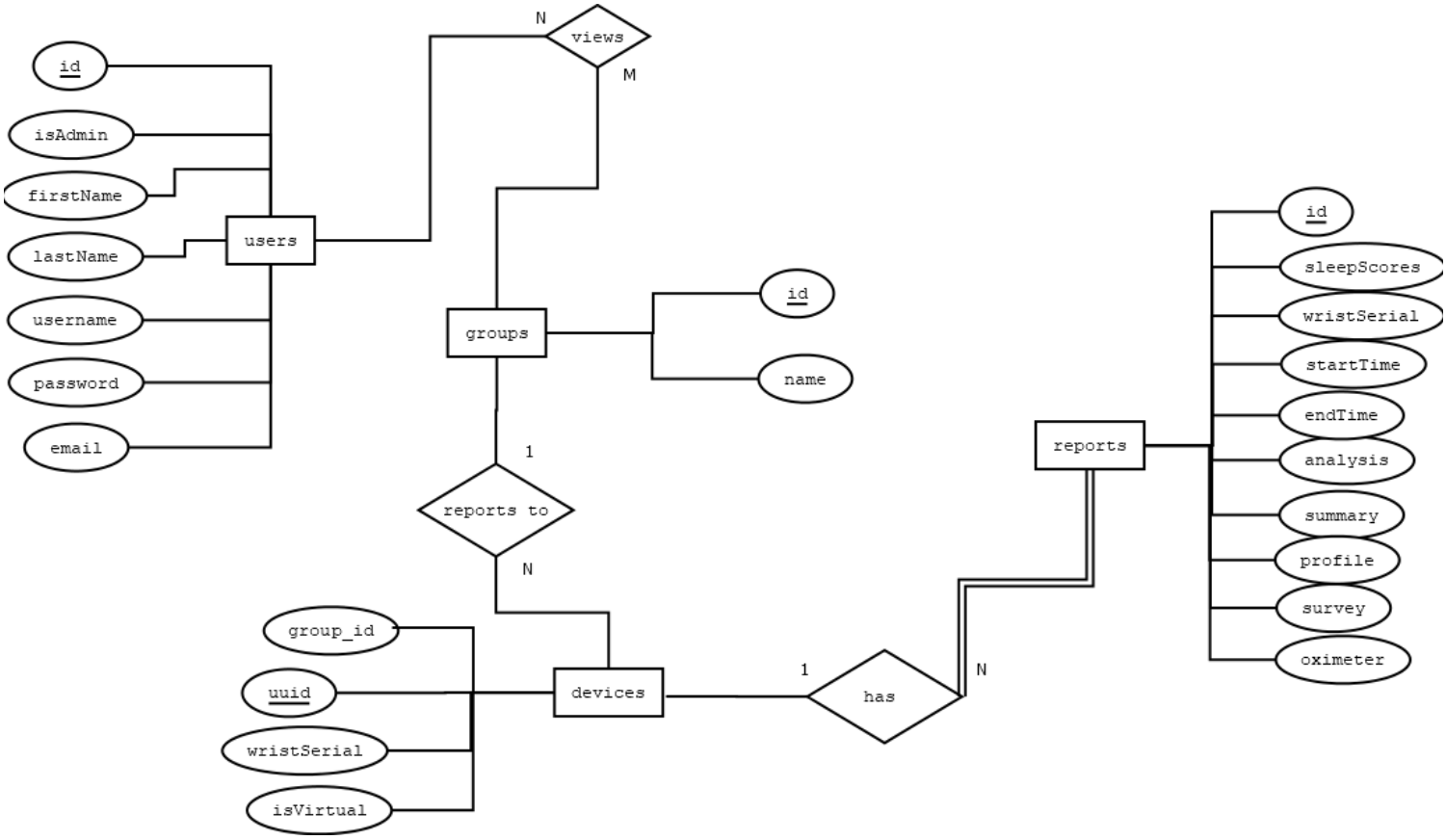

Figure 2: Overview of Data Included in JSON File

The figure above expresses the relationship between each entity in the data being collected as well as describes the structure of the database itself. The database focuses mostly on the reports entity and relates other entities to this central point, such as using this entity's sleep score values for the group's entity. The database allows for many reports to one device, many devices to one group, and many users being able to view many groups. The database collects a lot of different information but the most information funnels into the reports and users entities.

Technical Design

Technology Used

The EverSleep web portal, as mentioned before, is hosted on AWS and it uses a SQL database to store any user information. When it comes to the web portal itself, EverSleep uses mainly Node.js, Express.js, and Jade to do all of its programming. Node.js is used mainly for rendering web pages with routes. Express.js is mainly used for querying the database and making any calculations required to be shown on the page itself. The main library that is used is Chart.js. Chart.js allows for the quick creation of a variety of different graphs, which is crucial to our client. Finally, Jade is used for all HTML files. Jade is a templating engine that reduces the redundancy of characters found in the HTML language, making it so the code is easier to read and follow. Templates also make it easy to pass variables directly from the server.

Group Statistic

One of the main goals for our project was to create a way for users to see how their sleep is progressing over time. To do this, we created a metric called a “sleep score” that was calculated using different categories of the user’s sleep for that night.

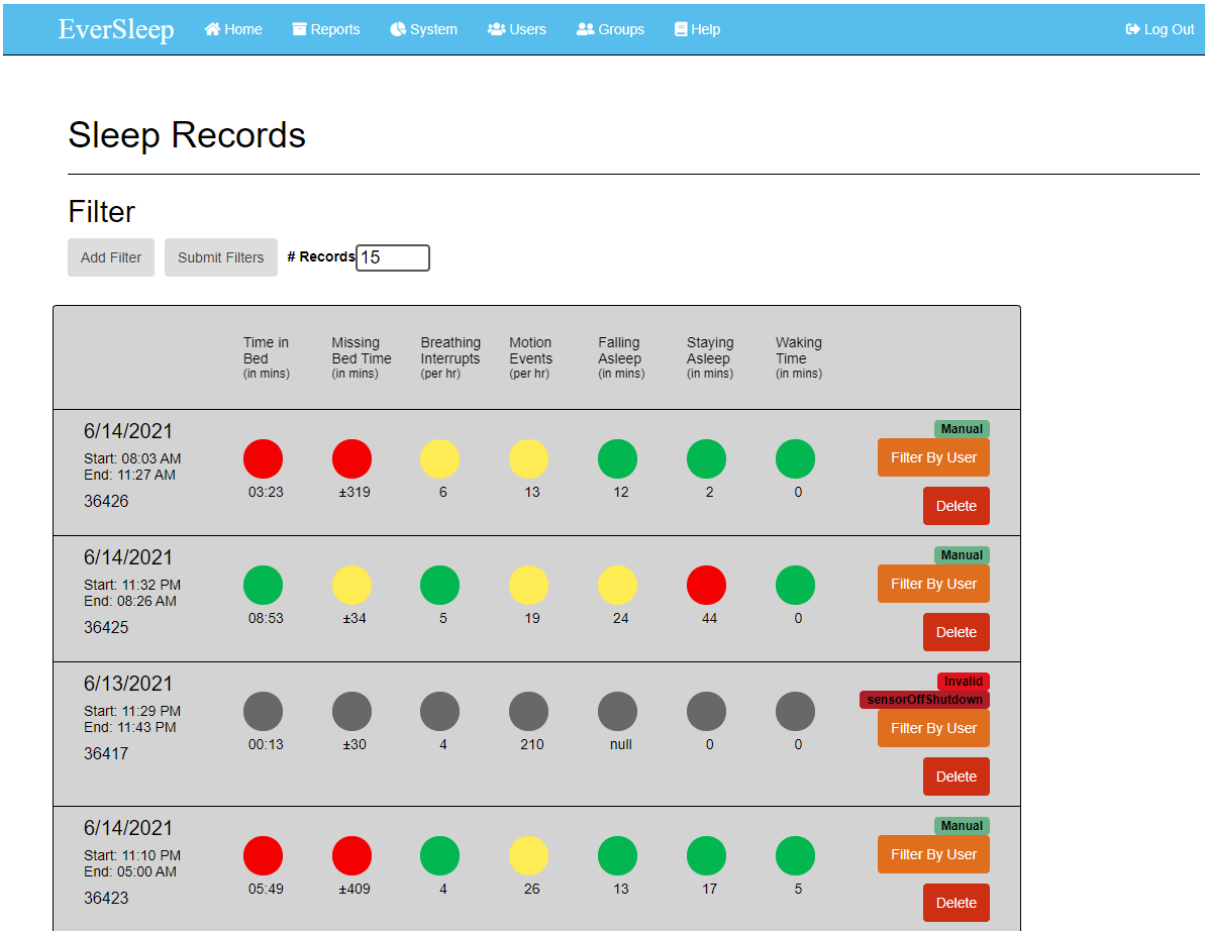

Figure 3: Sleep Records Page - EverSleep Portal

We based the sleep scores for individuals off of the colors of the circles of their report as seen in Figure 3. Each circle is a different metric from the night before. For Example, Staying asleep is the time that the user was having trouble staying asleep and falling asleep is the time the user had trouble falling asleep. Each circle color was given a certain number of points, and all those points were added up to calculate the overall sleep score for the night. The red color was worth 1 point, the yellow was worth 2 points, and the green was worth 3 points. This point-based system was based on what the colors represent, where red corresponds to doing a "bad" job, yellow corresponds to an "okay" job and green corresponds to a "good" job. This would serve as a metric for the overall sleep score.

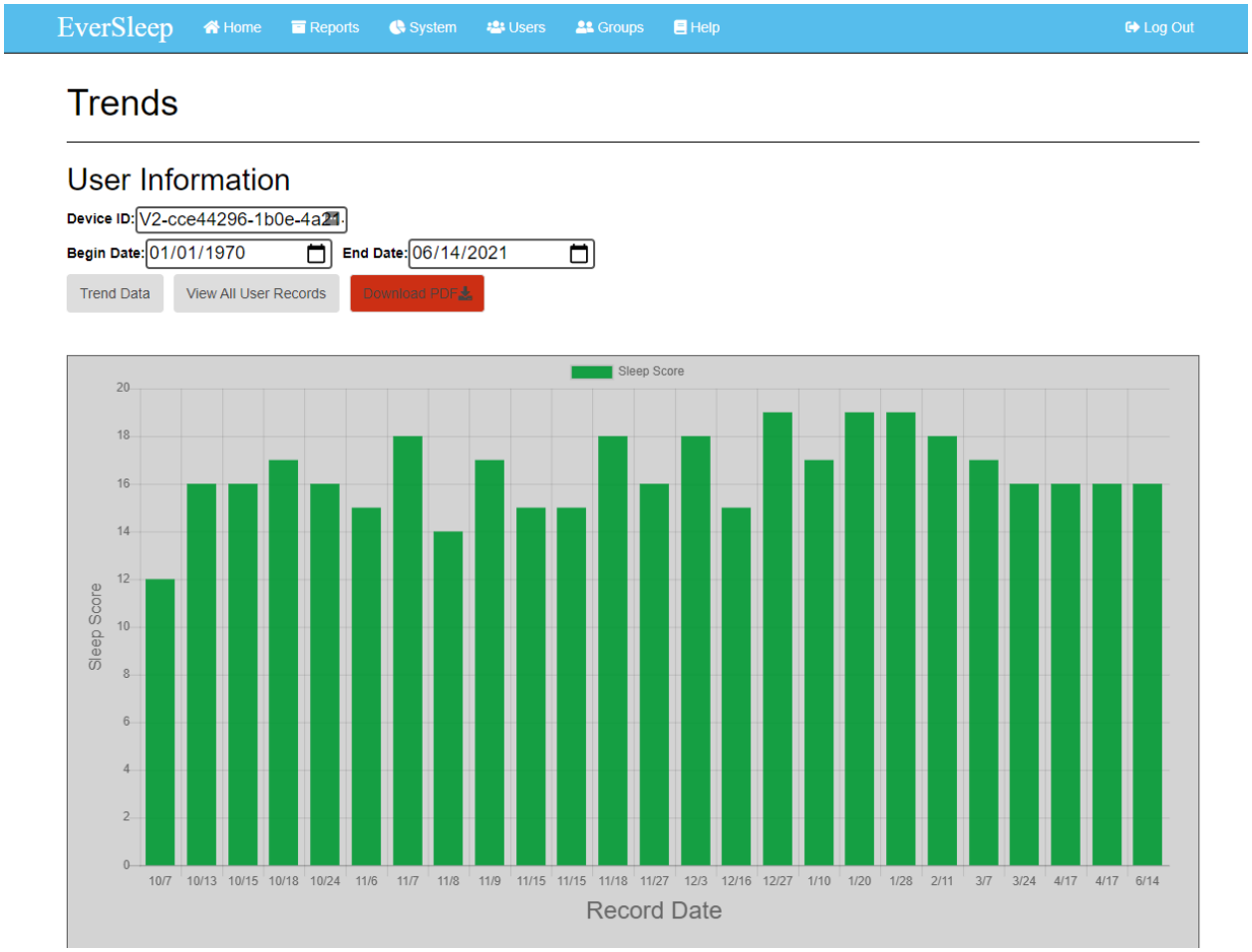

Figure 4: User Trends Page - EverSleep Portal

Using the user's sleep scores that were calculated from each individual record, we created a chart showing their scores over time seen in Figure 4. This allows the individual to see if they are making any improvements regarding their sleepover time. The next step was to apply this same feature to the Groups feature of the website.

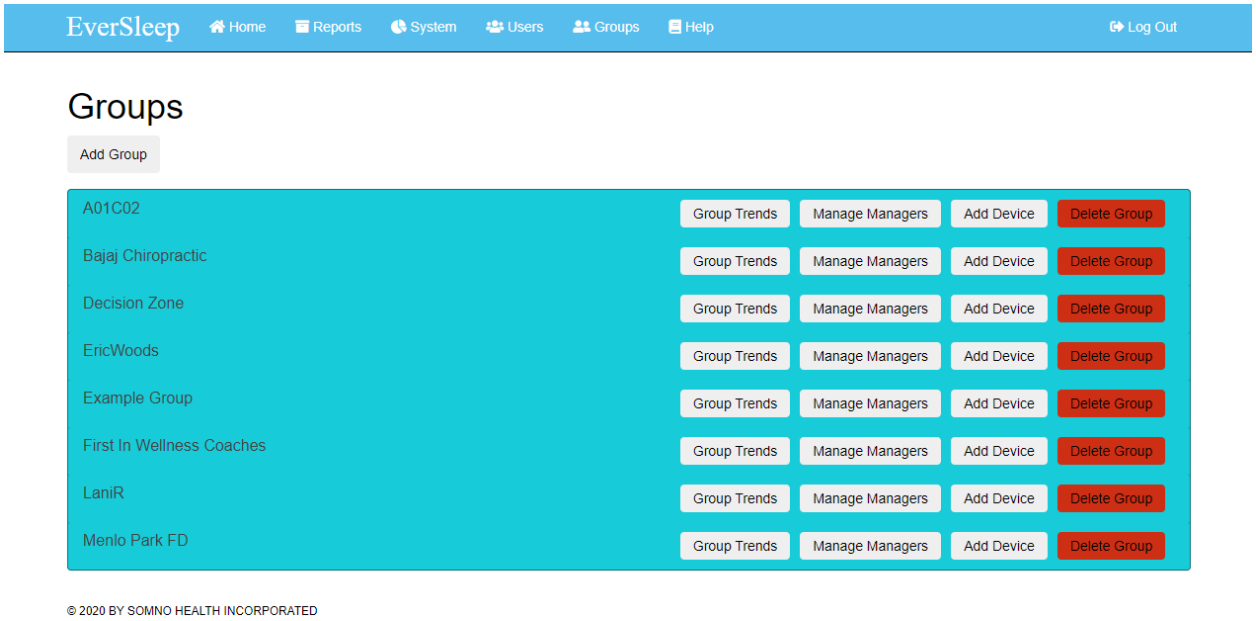

Figure 5: Groups Page - EverSleep Portal

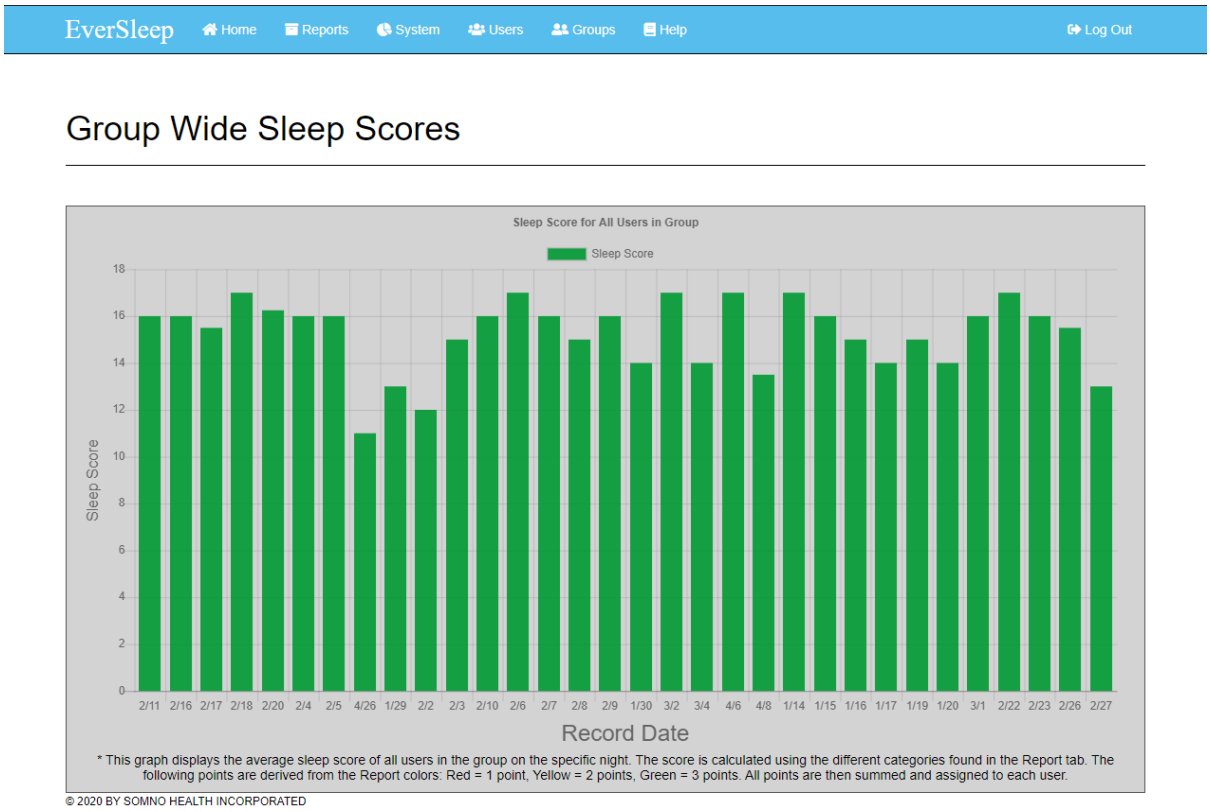

Figure 6: Group Wide Sleep Scores- EverSleep Portal

We used the sleep score data calculated for each individual of the group and created a page that allowed the group manager to see how all users in that group are doing over time. To get to this new page, only group managers can click on the “Group Trends” button which can be seen in Figure 5. Since there were multiple users, and since some users had data for an individual night, the sleep score was calculated by taking the average of all the sleep scores for a single night. After all the sleep scores were calculated, this data was presented in a chart as it can be seen in Figure 6.

The feature works as intended and will hopefully give a good picture to well care providers to see if their patients are improving or worsening over time with regards to their sleep patterns. In the case that one of the elements that are used in the score is more or less important, the score weighting can be changed to allow for certain aspects of the score to be more impactful.

Other Fixes/Features

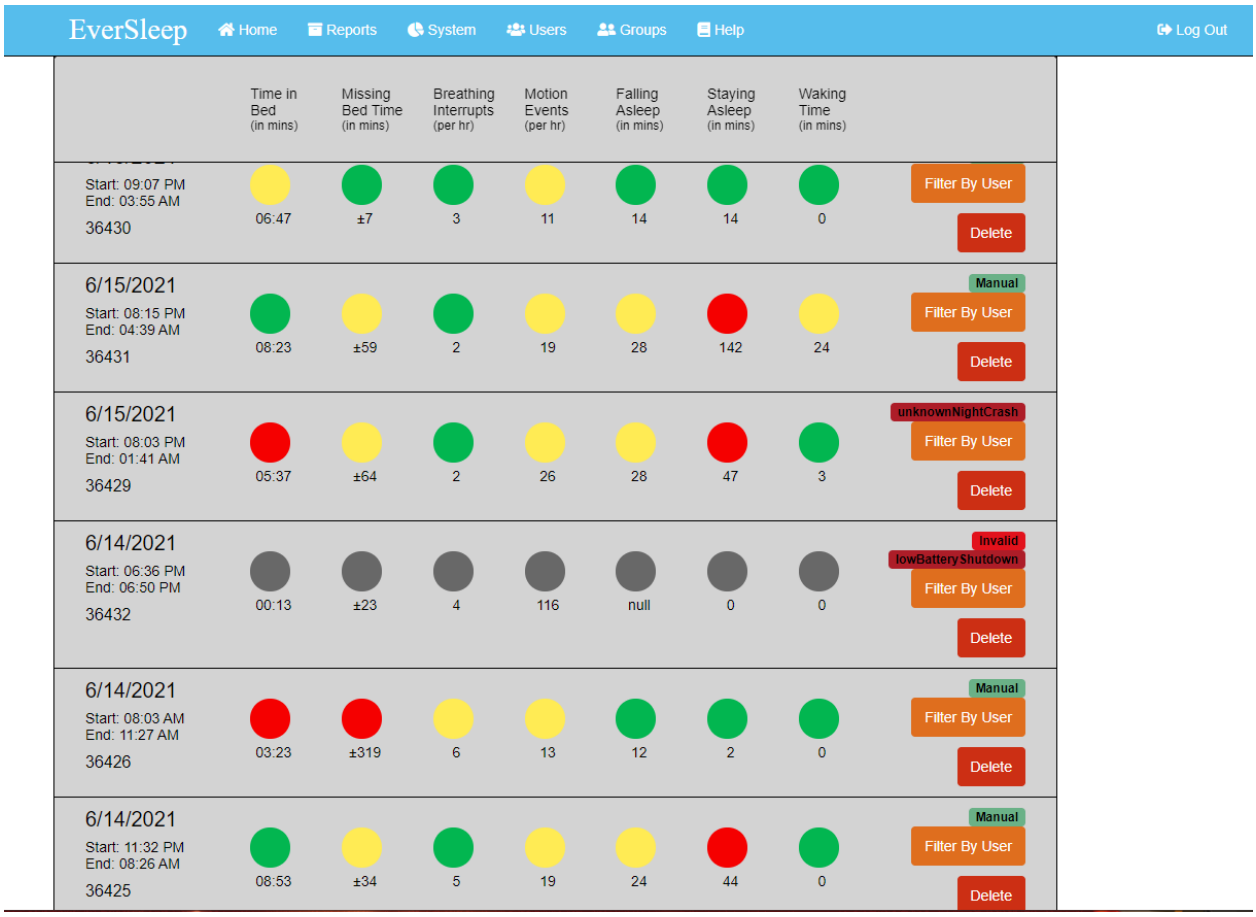

Figure 7: Reports Tab - EverSleep Portal

We added more features to this page such as the “Filter by User” button which allows you to see all of that user's reports. We also made it so the top bar sticks to the top of the page so you can see the labels as you scroll through reports as it can be seen in Figure 7.

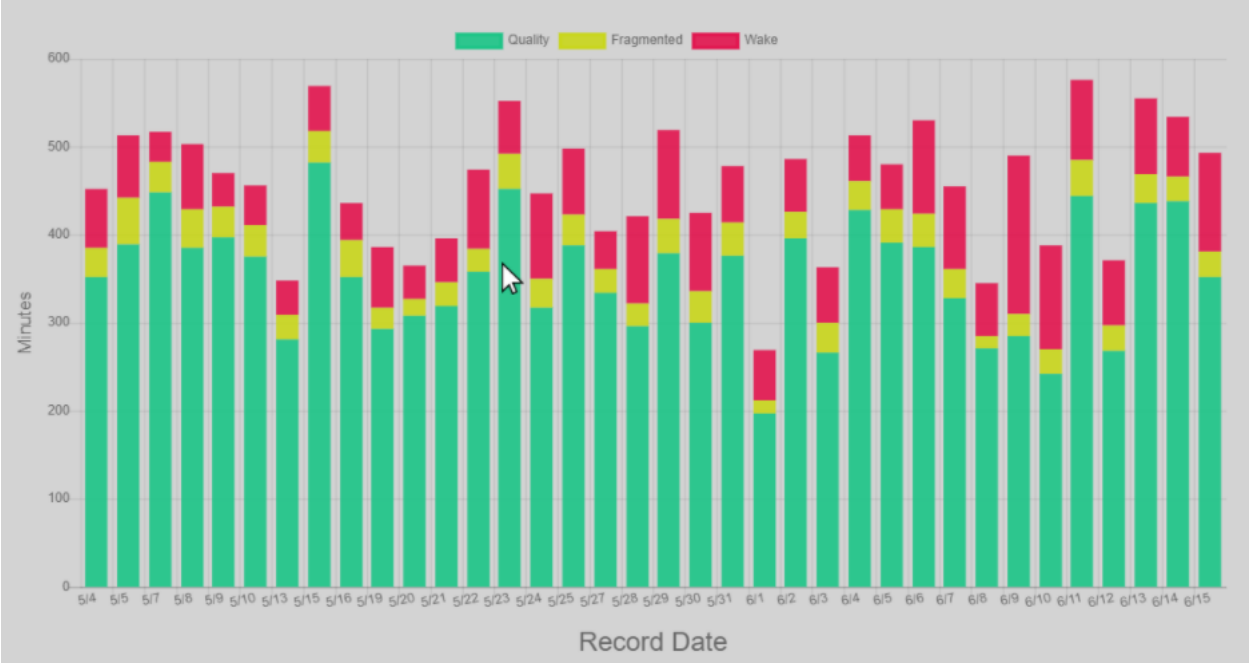

Figure 8: Quality Sleep Trends Graph - EverSleep Portal

Figure 8 represents the different levels of sleep with green being good quality sleep, yellow being fragmented not perfect sleep, and red being awake time during the night. We also made it so any bar found in the graphs of the Trends tab could be clicked and would redirect the user to the specific report for that night. One of the graphs that can be clicked on is shown in Figure 8.

In addition to the features described above, our group developed other “quality-of-life” features that enhanced the web portal.

Quality Assurance

Workflow

To manage and define the work that needs to be accomplished during each sprint, several different resources were used. For example, the main program used for workflow was Jira. This was used to keep a backlog of all features that need to be implemented and it helps the team keep track of the scrum tasks. It also helps organize issues into sprints which are completed every week. Additionally, it helps set the priority and complexity of assigned tasks to assure good time management. Figure 9 shows an example of the Jira Backlog.

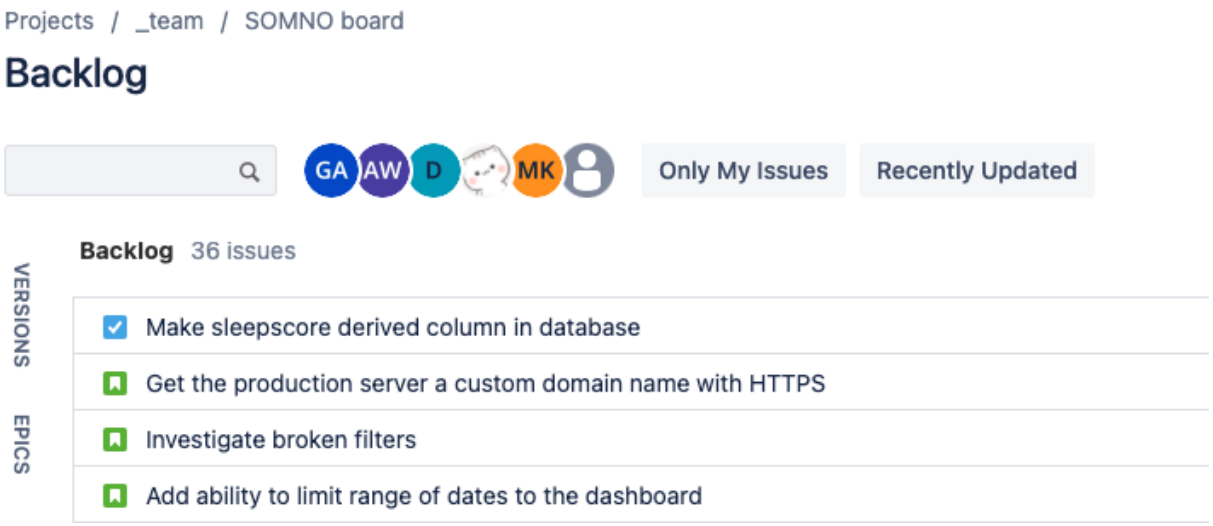

Figure 9: Photo showing the Jira Backlog

To ensure that any changes that were made would not interfere with pre-existing functionality, our group used GitLab. GitLab allowed us to work on different branches to develop features, which allowed us to handle any bugs or errors before we merged onto the main development and main production branch. If anything goes wrong on a branch, then reverting to a previous version to prevent issues was relatively easy. Figure 10 below shows some branches located in the GitLab used for this project.

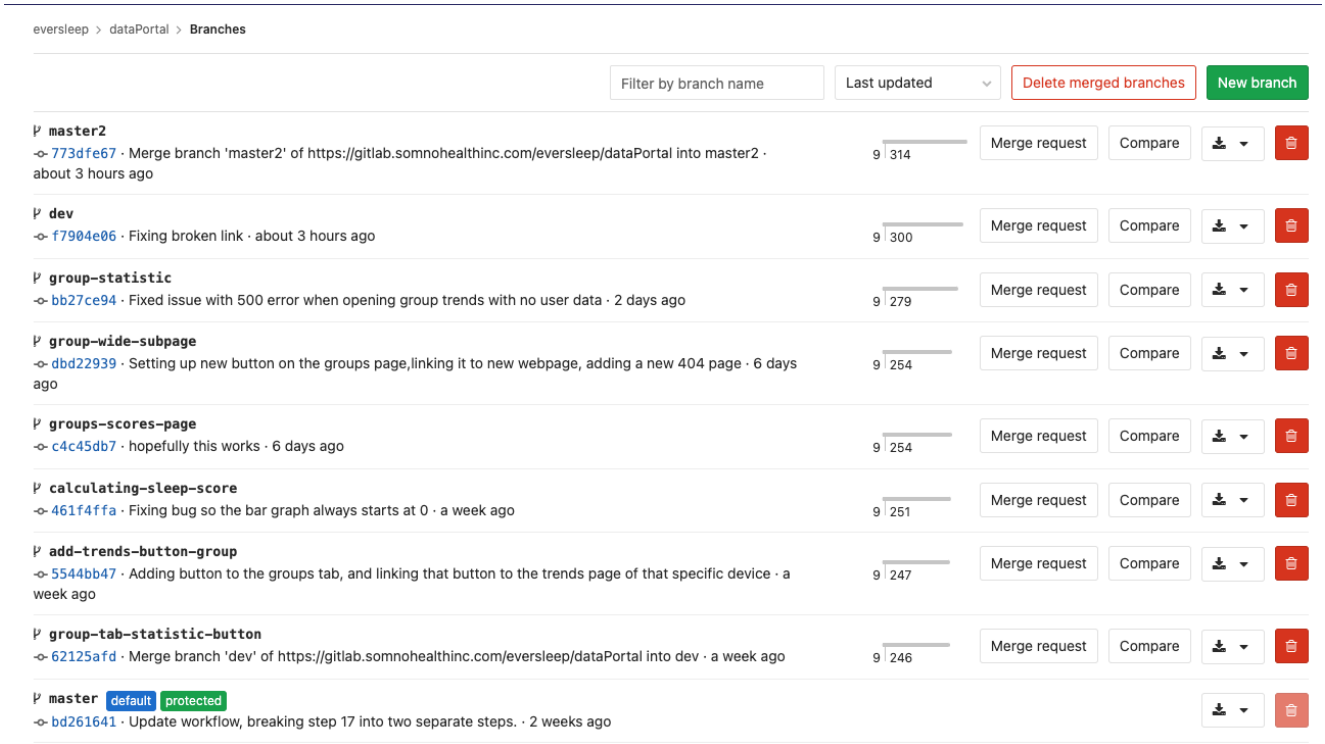

Figure 10: Git Branches

The final part of the workflow for this project was using pair programming to develop different features. When a good stopping point was reached we would switch the driver and navigator. This allowed us to do continuous code reviews and allowed us to catch any bugs before pushing to development and production. Almost every feature in our sprints was pair programmed as they made for good problems to solve together.

For each sprint, a reasonable number of features/fixes were grabbed from the backlog. These features were then assigned to teams, then a new branch was created to develop. Teams would utilize pair programming until the task was completed, then we would begin testing.

Testing

Testing was a crucial part of the process for our group since all changes deployed to the production server would affect paying customers. Testing ensured that all features were bug-free and did not break the functionality of preexisting features. The main method used was code reviews. Before merging a feature's branch with the development branch, other group members would review the code written to make sure everything looked correct. After code reviews, user interface testing was done. A non-admin user would log in and ensure that all features were working as expected and there were no bugs. Integration testing was also used. Our group would locally test on a development branch, then it would test using the production database.

The testing for this project was entirely manual, with the main source of testing being user interface testing. No automated tests were created since all features implemented did not require this. The testing was done in three stages: (1) testing on a locally hosted development server, (2) testing on a development server hosted by AWS, (3) pushing to the production server and running through one last round of tests. Before any changes were pushed to production, we ensured that the features met the standards of the client. The usability tests were not a high priority for this project as there already are many users using the application and their feedback on our changes are still being collected.

An example of one of the edge cases tested was the group statistic calculations. We had to test to make sure if there was no data for a group that it wouldn't error out, as well as make sure it calculated it correctly for many different users in one group. Afterward, the changes were pushed to the development server and tested, and after getting approval from the client for our changes, the changes were pushed to production, and it was tested to ensure that there were no bugs.

Results

This project was intended to make the user interface easier to read as well as create more useful information and clarity on the EverSleep site. Their website was already up and running so our main goal was to implement new features as well as update the user-friendliness of the website itself. All functional and non-functional requirements have been met including the far reach goals that the client wanted to achieve if we had extra time. These goals included different "quality-of-life" features such as creating new charts, moving chart locations, delete buttons, etc.

Over the four weeks, we have been able to work on this project, we mainly learned how extremely helpful code reviews and pair programming can be when implemented correctly. At first, they felt like they slowed down the pace of the project when in reality they saved many hours attempting to find bugs in the code. Another lesson we learned was how to better judge the complexity of projects being assigned in agile frameworks. A few projects were easier than expected and a few took much longer than anticipated. When we started with the agile software Jira we did not have a good idea of how long different tasks would take as previous groups did not make use of the priority feature.

Future Work

One of the largest pieces of work to be completed in the future would be updating the scripts being utilized in the Jade. One example of this would be updating the Chart.js script as we were unable to make the exact graph we wanted to since this was outdated. When we attempted to update this feature it broke in many places expressing how difficult this update will prove to be. Another future extension we believe should be implemented would deal with the security of the company's web portal as well as creating an HTTPS server for their data and updating their encrypting software. Currently, encryptions are lacking and even the smallest security changes

could benefit the company so this should not be too hard of a fix. Different strategies like password salting could help immensely with security. We made sure to comment on the code in all difficult areas to allow future coders to be able to more easily make changes and understand what we created. Adding more features to the current group statistics page such as allowing the bar graph to be clicked on and bringing you to a different page that shows each of the group's users and their scores for the night would be a great addition to the work we accomplished. Additionally, the current math to calculate the sleep score for groups and individuals is not incredibly accurate since it assumes all categories are weighted equally. A future fix would be to add weighting to the features, but for this to be accomplished, a clinician needs to be contacted. Something like this would not be very difficult but would have a lot of different pieces which would need to be accounted for.

Appendices

Read Me

Eversleep Dashboard Operating Procedures

Welcome to the Eversleep Dashboard! This was initially created as a part of the Colorado School of Mines Field Session Project in the Summer of 2018. Since then, at least six different field sessions have contributed to this project. Below is an overview of the structure of this project and you can visit the links below for more documentation on how to set up your development environment and how to deploy.

If there are any questions, feel free to reach out to the original developer at lvalentine@mymail.mines.edu, for inquiries within reason of course.

Infrastructure

AWS

AWS is very complicated and has a ton of features, but for this project, we mainly work with two of its services: Amazon Elastic Compute Cloud servers (EC2) and Amazon Relational Database Service (RDS). You'll be able to visit the links below once you get credentials from Dave.

EC2 Servers

This is where you can see the different EC2 "instances" where different codebases are deployed.

Instance	IP Address	Info
EverSleep-dataPortal-production (34)	34.209.240.196	This is the current production server so it is in use clients. This has master2 deployed to it and is linked to the eversleep production database.
EverSleep-dataPortal-development (35)	35.163.0.159	This is the current development server. This has dev deployed to it and is linked to the eversleep-clone development database. You an use this server for the last round of testing before you deploy new code to the production server.
EverSleep-dataPortal (54)	54.214.193.247	The branch master is deployed here. This is an old production server. It is still connected to the production database, but its APIs are not fully compatible with the production database.
EverSleep Dashboard Server	34.215.210.39	The eversleep-dashboard repo is deployed here. We suspect that this is some kind of relay server and even though this isn't being used, data might be passing through it and so the production server is dependent on this. Please don't shut this down.

Figure 11: EC2 Instances

RDS

This is where you can see the different databases that the data portals are pulling data from. Both databases are PostgreSQL databases.

- eversleep: This is the production database. This is a live database that stores over 30,000 nights of data from clients. You should be really careful when make changes to this database, if any at all. [link to info about missing devices table]
- eversleep-clone: This is a development database that stopped being live sometime in 2020. It is nearly identical to eversleep in structure. If you plan to make any database changes, you should test it out on this database first.

Main Branches

- master: Old production branch that is deployed on 54.214.193.247.
- master2: Current production branch. Only tested features should be merged into this branch. Please don't develop directly on this branch.
- dev: Current development branch. After testing individual features, you should merge them into this branch. Please don't develop directly on this branch either.

Interface

- There are two types of users that can log into the portal presently. There is an admin and manager. If you login as an admin, you can see who has an admin check or not in the Users Tab. The admins can see all of the data in the portal and managers can only see data depending on who is in their group and have limited functionality. There are some users that are a manager of only theirself, so they can only see their own data. You should login as both an admin and a manager to see the difference in views.
- The Home Tab shows the number of new records/reports that came in last night (if you are connected to the production database).
- The Reports Tab shows the 15 most recent reports that the signed-in user has permission to see.
    - You can click on the record id (5-digit number) to view more info about that specific person.
    - The displayed start and end times are based on the logged-in user's timezone.

Code Structure

Below is a very brief overview of the structure of the codebase. One thing to note is that throughout the whole app, but especially between app_api and app_server, there are a lot of variables that have the same name or very similar names. This can be very confusing because some are related and some are not. Not all user or users variables refer to the same thing and same with group and groups. Sorry! That's going to be confusing and you'll just have to figure it out.

Tip: User shift-ctrl-f to search for all instances of a term in VS Code.

dataPortal/app_server/views

- This project uses Jade/Pug.js templating. If you haven't worked with Jade, you can find resources online and you'll pick it up pretty quickly.

dataPortal/app_api/controllers

- The JS scripts in this folder are split into one for the Home Tab (home.js), one for the System Tab, Users Tab, and Groups Tab (other.js), one of the Reports Tab and an individual record (record.js), and lastly, the Trends Tab (trends.js)
- These scripts are for making API requests to the database
- Read more about how promises work to understand the structure of this code.

dataPortal/app_server/controllers

- These scripts provide server-side functionality for the web app.
- This is where the data returned from API requests in the app_api folder is 'caught' and assigned to variables that can be accessed within the Jade views.
- Ex. In the following snippet from dataPortal/app_api/controllers/others.js, you can see that in groups.jade, the view uses information that is saved in variables groups, users, devices_id, roster, and admin.

```
var renderGroupPage = function(req, res, responseBody){
if(responseBody.length == 0) {
message = "No groups found";
}

res.render('groups', {
    groups: responseBody.groups,
    users: responseBody.users,
    devices_id: responseBody.devices_id,
    roster: responseBody.groups_roster,
    admin: responseBody.admin
  });
}
```

Figure 12: Groups Coding

dataPortal/public/scripts

- These scripts provide functionality more typical of a client-side JavaScript that adds interactivity to the web app.
- You will see that at the bottom of these Jade files, many have these scripts imported.
- Many of the views use the same scripts (like modal.js).

dataPortal/public/bootstrap & dataPortal/public/stylesheets

- You've probably noticed that there are some funky colors in the portal. We have had issues using the amelia bootstrap theme that a previous team built this portal with. The bootstrap is 'broken'. Best practice suggests that you overload the styling in the the stylesheets folder instead of editing the bootstrap folder directly, but styling is very messy in this project, so it's really up to your best judgement.

Group Statistic Information

A general setup for the group statistic was completed in the Summer 2021 Field Session. The math for it is pretty generalized, which means that it might not be completely accurate. If changes need to be made, this section will explain what files will need to be changed.

dataPortal/app_api/controllers/trends.js: The function getDataWithSleepScore(data) is what calculates the sleep score itself. This is done in the "backend" of the site and it's stored along with the other information that is obtained in the JSON file for trends. The calculations rely on the colors that are assigned on the Reports page. If the color for that category is red, we assign 1 point, if the color is yellow, 2 points, and if the color is green, 3 points. These points are

calculated for all categories, and are then summed. Since the calculations right now rely on just addition, scaling a specific category should be easy since you can just multiply the numerical value by the scaling factor. The numerical value is defined in the calculateSleepScore function.

Going Forward

This project has been touched by a lot of different students, many of which have not had any web development experience. Because of this, you will see many inconsistencies, janky solutions, and poorly named variables. For the sake of future teams, please consider the following suggestions:

- Please do not delete the 2020_03_FinalReport.pdf. If you delete it, it can, for some reason, make your GitLab freak out and it won't let you checkout to a different branch, or merge to another branch.
- Add lots of comments as you develop! This is for you and the people who come after you. You may be surprised at how quickly code loses its meaning only a week or two after you wrote it.
- Test. Please test. So much of our time was spent changing a single line of code that had a million dependencies and then we would discover that several major features had been broken for weeks with no idea what broke it. And with a messy codebase like this, that's hard. During Spring 2021, we tried to work with developing some testing using the Selenium IDE and using GitLabs built-in CI/CD features. This is something to look into.
- Please refactor where you can. Code and poorly chosen variable names are littered throughout the codebase and database.
- Whenever you notice a bug, add a bug task to Jira. You may not have the capacity to fix the bug right now, but please add it to the Jira so the bug is at least documented somewhere.

Charts

The charts are created with chart js v2.7.2, some of the newer documentation might not work https://www.chartjs.org/docs/2.7.3/charts/bar.html

Development

Workflow

For the most part, don't touch the master branch. We created master2 as our main branch and developed each feature on an independent branch, then merged into the dev branch, then merged into master2. Here's a general workflow for reference:

1. Get new features from Chris and Dave
2. Add features into Jira backlog
3. Prioritize most important tasks in the backlog
4. Launch a sprint
5. Grab a task and assign it to to yourself and move to in-progress on Jira
6. Make a branch for your feature
7. Develop on that branch
8. Set upstream to GitLab
9. Commit often and push often
10. Test your branch locally
11. When finished with the feature, push the branch and move the task to the review column for peer review
12. After review, someone should merge the feature into dev.
13. Test dev locally
14. Deploy dev to 35 after a couple of features have been merged into it.
15. Test deployed dev
16. Merge into master2
17. Test master2 locally, then deploy master2 to the 34 server
18. Move task to done column.

How to Set-up a Local Development Environment

Run the web app locally

1. If you don't have a text editor yet, this is a great time to get one. We recommend Visual Studio Code.
2. Get access to a Linux-based OS
    a. The pathnames in the project assume you are using a linux-based machine. If you are using Linux or Mac, you should be ok. For Windows, here are some options:
    b. Install Linux Bash Shell
    c. SSH into Mines Linux Server
3. Clone the project.
    a. Navigate to a location on your machine where you would like to store the project in your terminal.
    b. Clone the project with
    git clone https://gitlab.somnohealthinc.com/eversleep/dataPortal.git
4. Switch the branch to dev
    a. Go into the dataPortal folder, and type git branch -a
    b. Switch to the dev branch by typing git checkout dev
5. Create a branch to develop on
    a. Create a new branch by typing git checkout -b $name
    b. You should be moved to the new branch, you can check this by typing git branch
6. Install Dependencies
    a. Make sure you have node and npm installed. You can google how to check if you have them installed or how to install them. We recommend a node version between v8 and v12. If you don't have the right version of node, checkout how to use nvm.
    b. Install dependencies for the project running npm i in the project.
7. Run the project locally
    a. Run export NODE_ENV=development; npm start in your terminal
    b. The terminal should confirm that you were able to connect to the database successfully and the address that your project is hosted on. If you go to localhost:3000 in your browser, you should be able to visit the locally hosted project. You can use NODE_ENV=production to connect to the production database.
    c. To shutdown the instance, use ctrl-c.

How to View the Databases

Database Clients: Database Clients
The database clients allow you to read and write to the databases hosted on the AWS server. You can run queries with them to make changes to the database.

1. Visit the link above for a couple of possible clients. In the past, some teams have used MySQL Workbench, but the the team that is writing this did not have success using it. We recommend Squirrel SQL and TablePlus (pretty GUI). The following instructions assume you are following the instructions for using SQuirreLSQL.
2. Ask Dave for the credentials to the databases. You will need them for when you add an alias to SQuirreLSQL.
3. Find the database URL for the database you want to connect to.
    a. There are multiple databases that are hosted on the database. For more info, visit the main README.
    b. The databases currently being hosted can be found here: AWS RDS
    c. Each database has its on address. If you click on the database name, you will be able to see the address labelled as the endpoint for the database.
4. Connect to a database
    a. If you are following the instructions from the CSCI 403 Website , you will need to add a URL to add an alias.
    b. Preface the endpoint with jdbc:postgresql://.
    c. For example, to connect to the eversleep-clone database, the URL looks like jdbc:postgresql://eversleep-clone.cjvc6lbtyjwi.us-west-2.rds.amazonaws.com:5432/eversleep
5. You may want to add an alias for each database for convenience. Please be very careful of touching the production database.
6. The databases store a lot of json files. To be able to view jsons instead of <Other> in SQuirrel SQl, use these instructions.

Tip: If you are working on updating or adding a query, try running the query directly in your database client before trying to parse it into string form in ./app_api/controllers.

Troubleshooting

- For debugging help:
    - Make use of console.log() in the code. This will print whatever you pass it in the terminal you are running your code in or the browser console (use your browser's inspector to see this console). However, when you are finished using the console.log(), please remove it before committing to keep the codebase clean.
- When you try to deploy on the server or locally, the deployment fails and you get a bunch of red text
    - It might be the version of node. Sometimes the node version of the servers automatically changes. You can change it back to something like v8 with nvm.
    - Sometimes you need to reinstall your node packages. Make sure you pulled the repo and npm i again.
    - Sometimes your package-lock.json gets messed up on the server or locally. You can try to go to the GitLab and copy the package-lock.json file and paste it over your server or local package-lock.json.
    - Sometimes web app gets confused with credentials when deploying locally. Try using a private window.
- When you try to visit the GitLab it will not load or you can't push your code to it.
    - If this happens, give it a couple of minutes and try again.

Deployment Notes

How to Deploy to AWS Server

Deploying to an AWS server is similar to running the project locally.

SSH to AWS EC2 Instance

This is how you access the AWS server to deploy code to it.

1. Please get the csm-group.pem file from Chris or David. This will be needed to login to the EC2 and for security reasons, cannot be stored on the git repo. PLEASE MAKE SURE THIS STAYS PRIVATE. It is the only security this EC2 instance has.
2. Change the access rights to the pem file once you have a copy of it saved locally.
   chmod 600 csm-group.pem
3. Identify the IP address of the instance that you want to connect to.
4. At the time of writing, there is a production (34.209.240.196) and a development server (35.163.0.159). In the case that the IP address or the instances change, you can check the IPv4 addresses on the AWS server here.
5. Then, you need to SSH into the domain using the appropriate IP address with the following command (with the csm-group.pem file somewhere local on your machine): ssh ubuntu@<IP Address> -i <path to local csm-group.pem>. At this point, you should be logged into the EC2 instance.

Make sure the code on the server is up to date

1. You need to pull the latest changes before redeploying. This is pretty simple. You just need to go into the directory and pull down your changes.
   ```
   cd ~/dataPortal
   git pull
   npm install
   ```

At this point, your changes are pulled down and any new dependencies were installed.

Kill the old deployment

1. The website is currently running on a detached process. We need to kill that old process before we can start a new one.

The following command will generate list of the current processes that are running and their ids.

ps -eaf | grep node

Kill all the process that are listed there (except for the color-related one). You can also learn how to kill these processes on Google.

kill <pid>

Redeploy

You can specify the node by typing NODE_ENV=development; npm start directly in the terminal, but this is temporary. When you close your terminal, the deployment will shutdown too. To permanently deploy, we use tmux. tmux is used to permenantly deploy by setting the bash profile of the instance. Note from original team: Ideally the project should use something like elastic beanstalk for deployment.

Start a tmux session:

1. tmux new
2. ctrl+b (nothing changes, you probably typed it right)
3. export NODE_ENV=development; npm start

Here are the current NODE_ENV environments:

Environment	
development	Connects to eversleep-clone.
Test	Not defined yet.
production	Connects to eversleep. Only for deployment. Ignores all packages listed under devDependencies in package.json file.

Figure 13: Node Environments

For more information, see dataPortal/config/config.json.

1. Hit the enter key
2. ctrl+b then type d. It should exit the mux shell.
3. Travel to the IP address of the EC2 server you SSHed into (http://:3000) and the site should be live. You can now exit the shell.