Christopher Bremser

Aryaman Nath

Rusty Rinehart

Rine Heart Monitoring: ECE 413 Final Project Documentation

Front End

As the front end specialist it was my job to create the full interaction for the customer on the website. To begin with the list of files that I have created and modified are as follows:

- ➔ Index.html
- ➔ Index.css
- ➔ Index.js
- ➔ Account.html
- ➔ Account.css
- ➔ Account.js
- ➔ Signup.html
- ➔ Signup.css
- ➔ Signup.js
- ➔ Login.html
- ➔ Login.css
- ➔ Login.js
- ➔ Navbar.html
- ➔ Navbar.css
- ➔ Navbar.js
- ➔ Particle.html
- ➔ Particle.css
- ➔ Particle.js
- ➔ Reference.html
- ➔ Reference.css
- ➔ reference.js

Going through each file:
Index.html:

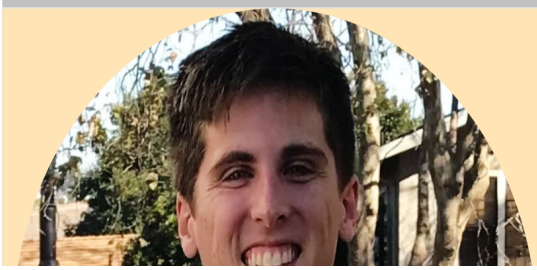
```
public > index.html > html > body > div.container > div#name1.row.border > p#info.blockquote.text-right.col.border-secondary
30   src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"></script>
31   test compiled JavaScript -->
32   src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/js/bootstrap.bundle.min.js"></script>
33
34
35   v id="nav-placeholder"></div>
36   ass= "container" >
37   v class="row border" id = "missionStatement">
38    <p id="mission" class="col strong em font-weight-bold font-italic text-danger">
39        Welcome to Rine Heart Monitoring, our premiere blood oxygen saturation level and pulse instrument empire!
40        We use a proprietary technology to get the most precise measurements in the business. We are a family company based on making sure that we protect our family, so you can protect yours.
41        Come <span id = "team">meet some of our team!</span> Enjoy your experience.
42        We use the Particle Argon Blood Sensor for users to calculate heart rate and blood oxygen(o2) saturation.
43        Those who have the right aptitude are invited to join our coven.
44
45    </p>
46   iv>
47   v class = "row border" id="name1">
48   g src ="images/rusty.jfif" alt = " Rusty Rinehart Our CFO " id = "rusty" class = "rounded-circle float-left col img-fluid" >
49   class = "blockquote text-right col border-secondary" id="info"> Dr. Rusty Rinehart graduated from Harvard Summa Cum Laude in 1956 and begun to emmerge as the world's leading Hematologist.
50    He began to catalog the world's blood, and in the midst of the Vietnam conflict, his knowledge in blood transfusions catapulted him to the world stage. He began research on cataloging the world's blood type in a regi
51    In 1985, he joined the Nosferatu Coven with his new registry. His research shifted into advancing human potential with an advanced gene he found in his thesis research.
52    In 1990, he founded Rine Heart Monitoring company to engineer new devices to help create a better human. In 2015, Peter Thiel's mentee billionaire power duo of Chris Bremser and Ary Nath invested in Rinehart Heart Mo
53     <br>Contact: rustyrinehart@arizona.edu</p>
54   iv>
55
56   v class = "row border" id="name2">
57    <ing src = "images/ary.jfif"  alt = "Aryaman Nath Genius billionaire playboy philanthropist"  id = "ary" class = "rounded-circle float-right col img-fluid ">
58    <p class="blockquote text-left col border-secondary" id="info" >
59        Ary Nath immigrated here in 2000, and was able to get a prestigous scholarship to Stanford. There, he accelerated to the top of his class and eventually joined up with friend Chris Bremser.
60        Ary and Chris became an inseparable team. They invested early into Facebook in 2004. With that they were contacted by Peter Thiel to join his team. Under this new mentor, the team pushed themslves to the top of t
61        investment game. Within 1 year they were able to become multi-billionaires by investing heavily in the tech market. Ary is the strategist to the team. Some say it is Ary who targets the companies to take down and
62        This may have happened when they had bought a small tech firm "Etsy" to provide a platform for journeymen to sell their wares. Famously, it was Ary's idea to buy out any competitor, keep the ideas, keep the usefu
63        <br>Contact: anath@arizona.edu
64    </p>
65
66   iv>
67
68   v class="row border" id="name1">
```

A Bremser & Nath Holdings Company Production

Rine Heart Monitoring

Home | References | Physician's Portal | Account Settings | Logout

Welcome to Rine Heart Monitoring, our premiere blood oxygen saturation level and pulse instrument empire! We use a proprietary technology to get the most precise measurements in the business. We are a family company based on making sure that we protect our family, so you can protect yours. Come meet some of our team! Enjoy your experience. We use the Particle Argon Blood Sensor for users to calculate heart rate and blood oxygen(o2) saturation. Those who have the right aptitude are invited to join our coven.



Dr. Rusty Rinehart graduated from Harvard Summa Cum Laude in 1956 and begun to emmerge as the world's leading hematologist. He began to catalog the world's blood, and in the midst of the Vietnam conflict, his knowledge in blood transfusions catapulted him to the world stage. He began research on cataloging the world's blood type in a

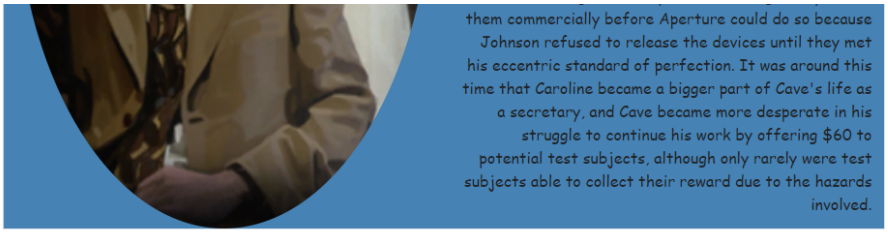
Dr. Rusty Rinehart graduated from Harvard Summa Cum Laude in 1956 and begun to emmerge as the world's leading hematologist. He began to catalog the world's blood, and in the midst of the Vietnam conflict, his knowledge in blood transfusions catapulted him to the world stage. He began research on cataloging the world's blood type in a registry. In 1985, he joined the Nosferatu Coven with his new registry. His research shifted into advancing human potential with an advanced gene he found in his thesis research. In 1990, he founded Rine Heart Monitoring company to engineer new devices to help create a better human. In 2015, Peter Thiel's mentee billionaire power duo of Chris Bremser and Ary Nath invested in Rinehart Heart Monitoring, and catapulting it to the top of the NASDAQ.

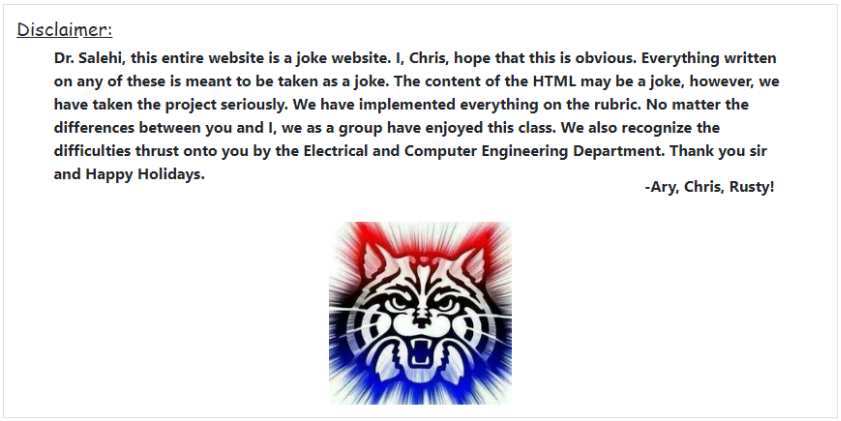
Ary Nath immigrated here in 2000, and was able to get a prestigous scholarship to Stanford. There, he accelerated to the top of his class and eventually joined up with friend Chris Bremser. Ary and Chris became an inseparable team. They invested early into Facebook in 2004. With that they were contacted by Peter Thiel to join his team. Under this new mentor, the team pushed themslves to the top of the investment game. Within 1 year they were able to become multi-billionaires by investing heavily in the tech market. Ary is the strategist to the team. Some say it is Ary who targets the companies to take down and Chris is the fangs that does so. This may have happened when they had bought a small tech firm "Etsy" to provide a platform for journeymen to sell their wares. Famously, it was Ary's idea to buy out any competitor, keep the ideas, keep the useful's, and terminate everybody else.
Contact: anath@arizona.edu

Chris Bremser went to Stanford on family money and had spent much of his life in halls with very wealthy people. He grew up making mult-million dollar deals, when his father demanded that he would go get a college degree. Stanford business was an easy, yet boring endeavour for the young investor. This was true until he met Ary Nath. Ary was the only person who was smarter than him for years. Immediately, they became a power team. In 2004, the team invested early into Facebook. It was a family contact that connected Chris and Ary with Peter Thiel. Peter Thiel loved the viciousness of the team. However, Peter Thiel underestimated the duo. Though it was Ary's idea, Chris manipulated Peter into losing 2 billion dollars in 2007. This allowed the pair to weather the Great Recession. In 2015, the team invested in Dr. Rinehart's heart company to build a better tomorrow.

Contact:
cwbremser@arizona.edu

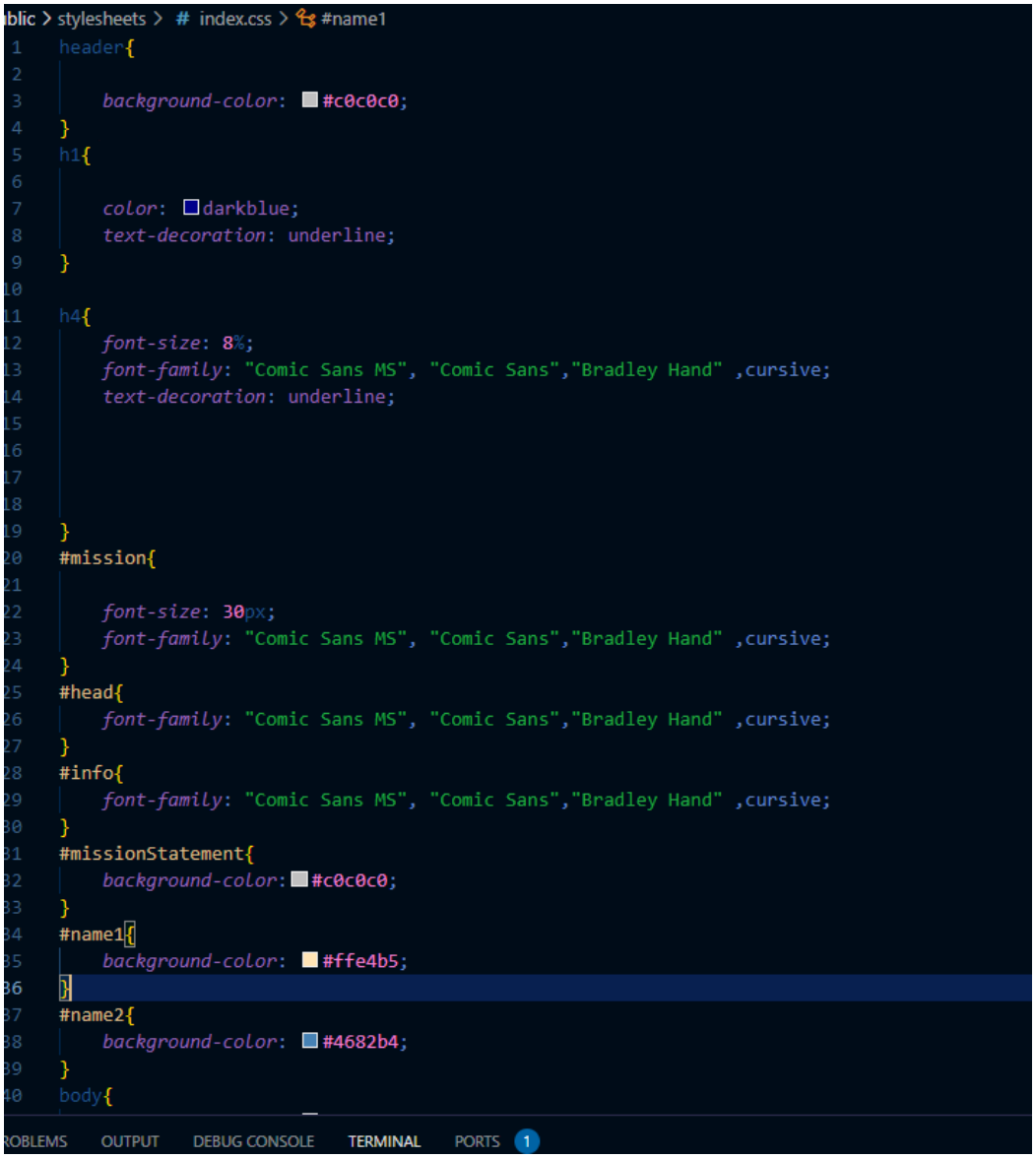
In 1943, Cave Johnson created Aperture Fixtures, a shower curtain manufacturer. The precursor to Aperture Science, he selected the name because it 'makes the curtains sound more hygienic'. He became a billionaire after winning contracts to manufacture shower curtains for all branches of the U.S. Military except the Navy, winning the '1943 shower curtain salesman of the year' award. At the time of its creation, Aperture Science Innovations had developed the Aperture Science Portable Quantum Tunneling Device, a prototype to the modern Handheld Portal Device. As such, it seems that Aperture had already started to perfect portal technology, but was not ready to release it to the world, under the belief that much more could be done than just create portals. Of course, the technology was also impractical and bulky, and was delayed by Cave's wild innovations. During the 1970s, Aperture Science had become known as just 'Aperture', and the rise of Black Mesa and its success in bidding for contracts pushed Cave's temper to the limit. It is also implied that Black Mesa stole designs from Aperture and began to produce them commercially before Aperture could do so because Johnson refused to release the devices until they met his eccentric standard of perfection. It was around this time that Caroline became a bigger part of Cave's life as a secretary, and Cave became more desperate in his struggle to continue his work by offering $60 to potential test subjects, although only rarely were test subjects able to collect their reward due to the hazards involved.





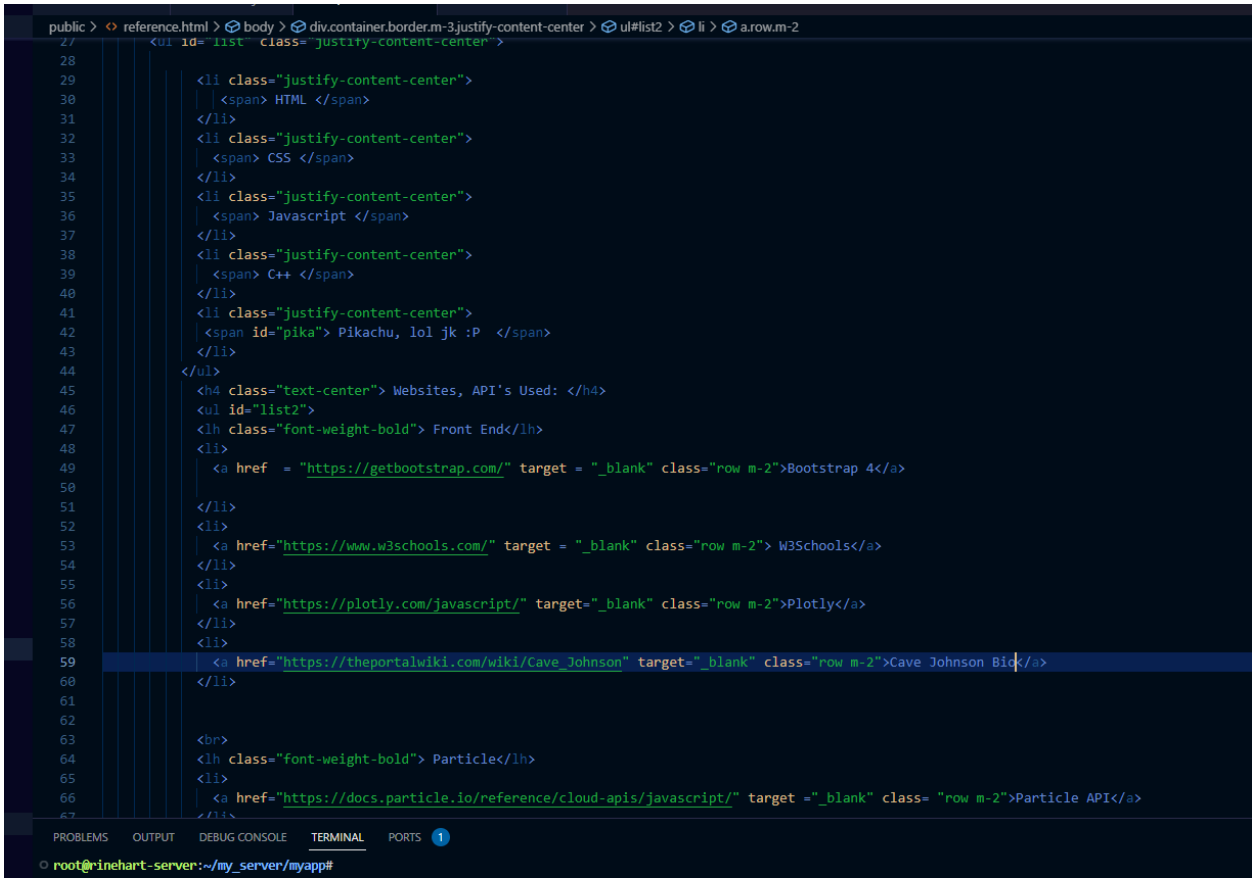
So this file is to begin by describing our website. When I read through the documentation, I found that we had to make a video. This video was intended to pitch to investors as if this was a real company. I figured since we were to make a company, I wanted to have some fun. I created a joke company. My idea was a vampire company that uses these sensors to find specific blood type people. With Rusty being the doctor that created the first vampire(Himself). Since this is an “evil” corporation we had to include Cave Johnson from the portal universe. One of the requirements was we had have our CSS be variable for both mobile and desktop browsers.

I learned how to use bootstrap4 for this reason. I used it with all of the front in our website. Bootstrap uses a class system to make changes. These classes will automatically fit no matter what device you look at the website on. This is why we were so interested in using it. I used the container class a lot, which made a box that I could play in. Using rows I could make sure the y axis were the same. With col (columns), I could play within the same x axis. Columns would split the available real estate and make sure it displayed quite nicely. No matter what device you look at this website it will display everything quite nicely. With this in mind I will not discuss bootstrap much on each page discussion. However the css for each is very very simple.



The only thing I really used CSS for was background colours, as a hex code doesn't change for mobile devices. And we wanted a playful font. However on apple devices comic sans ms isnt available (Microsoft Product). So I added Bradley Hand for apple devices.
With index.js is only for our Navbar trick. Which I'll talk about when i get to navbar.

References.html,reference.css,reference.html:


public > reference.html > body > div.container.border.m-3.justify-content-center > ul#list2 > li > a.row.m-2
<ul id="list" class="justify-content-center">
<li class="justify-content-center">
 HTML

<li class="justify-content-center">
 CSS

<li class="justify-content-center">
 Javascript

<li class="justify-content-center">
 C++

<li class="justify-content-center">
 Pikachu, lol jk :P

<h4 class="text-center"> Websites, API's Used: </h4>
<ul id="list2">
<lh class="font-weight-bold"> Front End</lh>

Bootstrap 4

 W3Schools

Plotly

Cave Johnson Bio

<lh class="font-weight-bold"> Particle</lh>

Particle API
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 1
root@rinehart-server:~/my_server/myapp#

References

Home | References | Physician's Portal | Account Settings | Logout

Code Languages Used:

- HTML
- CSS
- Javascript
- C++
- Pikachu, lol jk :P

Websites, API's Used:

Front End

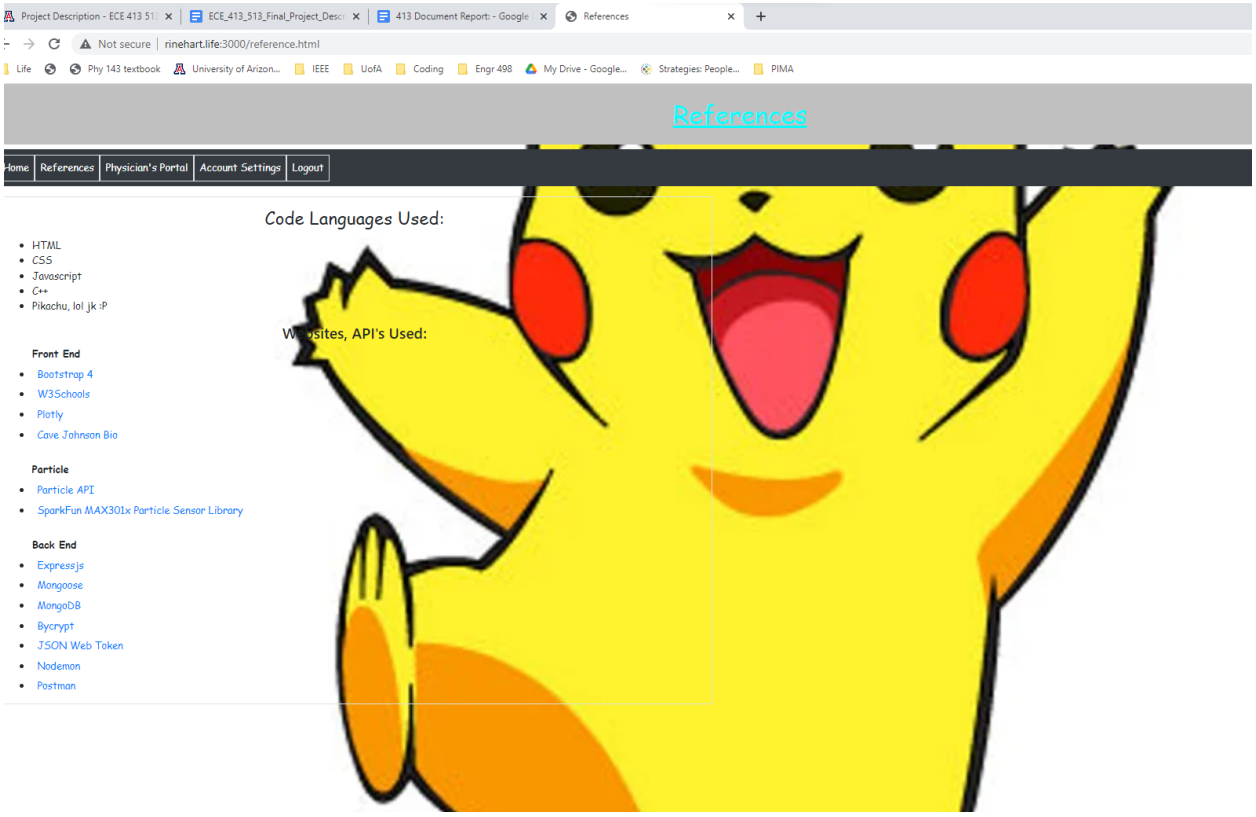
- Bootstrap 4
- W3Schools
- Plotly
- Cave Johnson Bio

Particle

- Particle API
- SparkFun MAX301x Particle Sensor Library

Back End

- Expressjs
- Mongoose
- MongoDB
- Bycrypt
- JSON Web Token
- Nodemon
- Postman

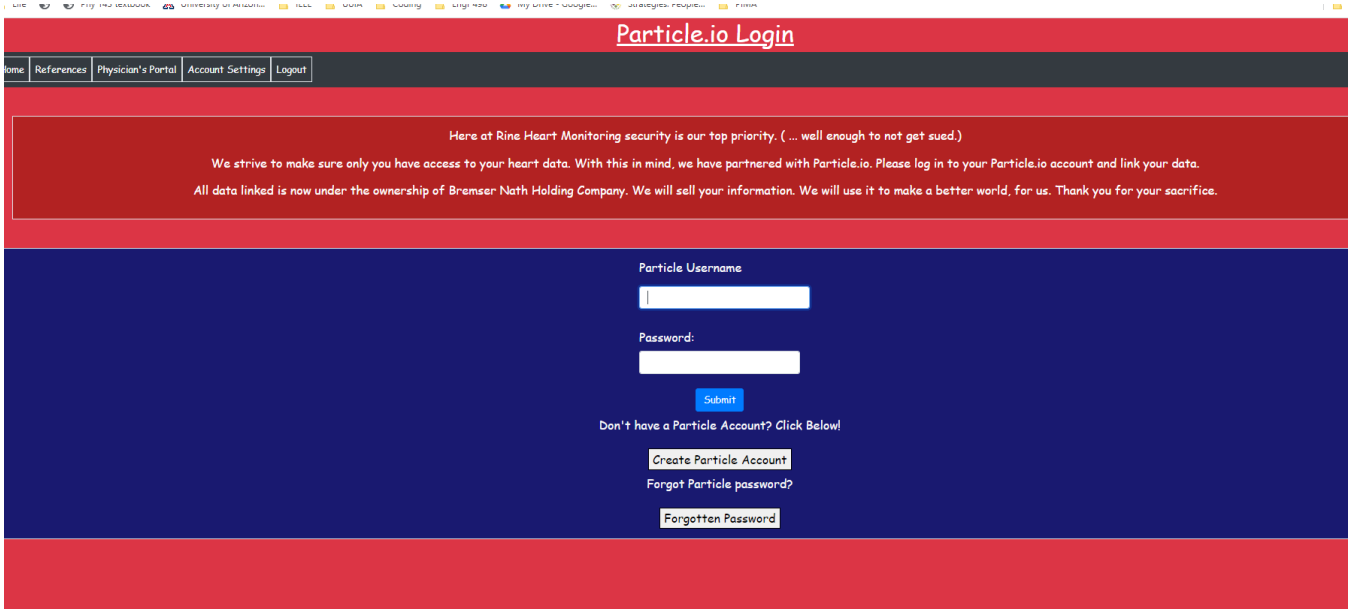


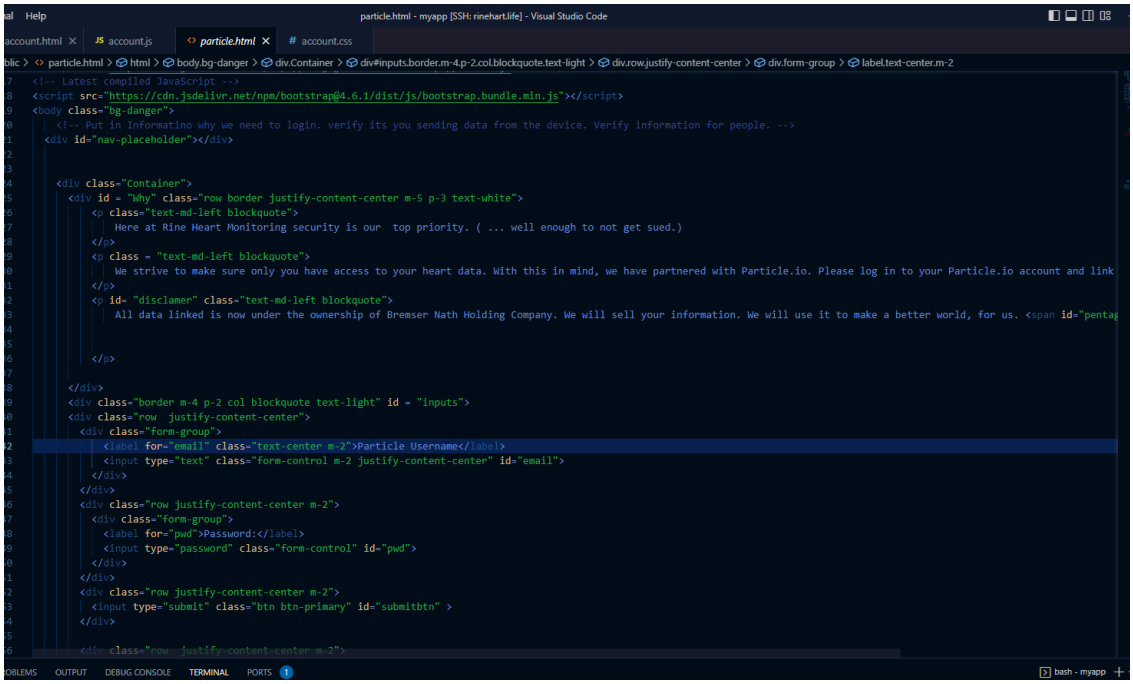
For References, we have posted exactly what languages and links we have used. All of these are split up between front end, back end, and particle. The only thing of note here is that Particle API was used in both the back end and particle. Other than the links open up to their respective websites that we used, there is a fun easter egg i have screenshot here.

There is a famous image that has a resume with a bunch of coding languages with random pokemon mixed in, this is an homage to that meme. If you hover over pikachu it changes the background image to pikachu.

Particle

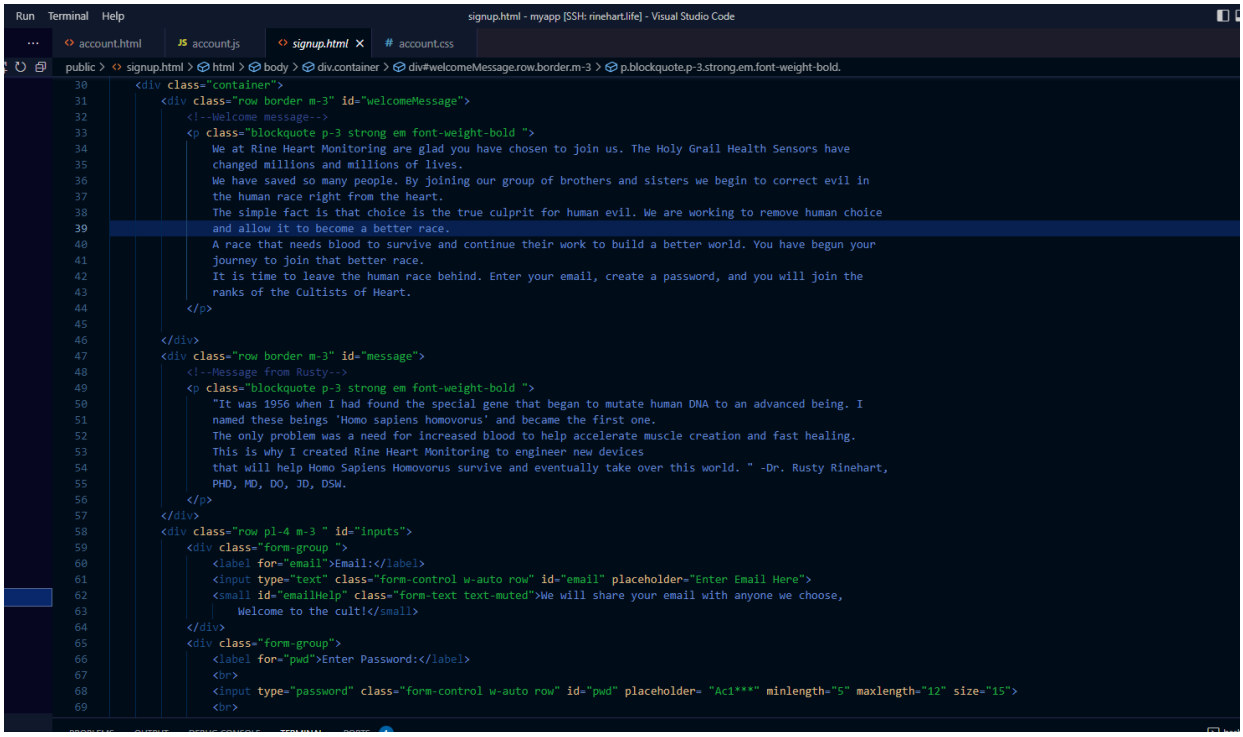
We embedded a connection between particle and our website. This was a clever way we found to link and remove devices from the website using the particle website.





What you will see here is Particle Login, and if the customer does not have a particle login, it will direct them to particle.io to create an account. As we are a vampire company out to be the most evil company in the world. There is always an easter egg if you ever hover over a sacrifice word on our website ;).
This is automatically directed after a customer logs into our website and doesn't have a connected particle account. If the customer has already logged into particle then it will not direct the user to this page.

Sign up:


Run Terminal Help
signup.html - myapp [SSH: rinehart.life] - Visual Studio Code
account.html
account.js
signup.html
account.css
public > signup.html > html > body > div.container > div#welcomeMessage.row.border.m-3 > p.blockquote.p-3.strong.em.font-weight-bold.

<!--Welcome message-->

We at Rine Heart Monitoring are glad you have chosen to join us. The Holy Grail Health Sensors have
changed millions and millions of lives.
We have saved so many people. By joining our group of brothers and sisters we begin to correct evil in
the human race right from the heart.
The simple fact is that choice is the true culprit for human evil. We are working to remove human choice
and allow it to become a better race.
A race that needs blood to survive and continue their work to build a better world. You have begun your
journey to join that better race.
It is time to leave the human race behind. Enter your email, create a password, and you will join the
ranks of the Cultists of Heart.

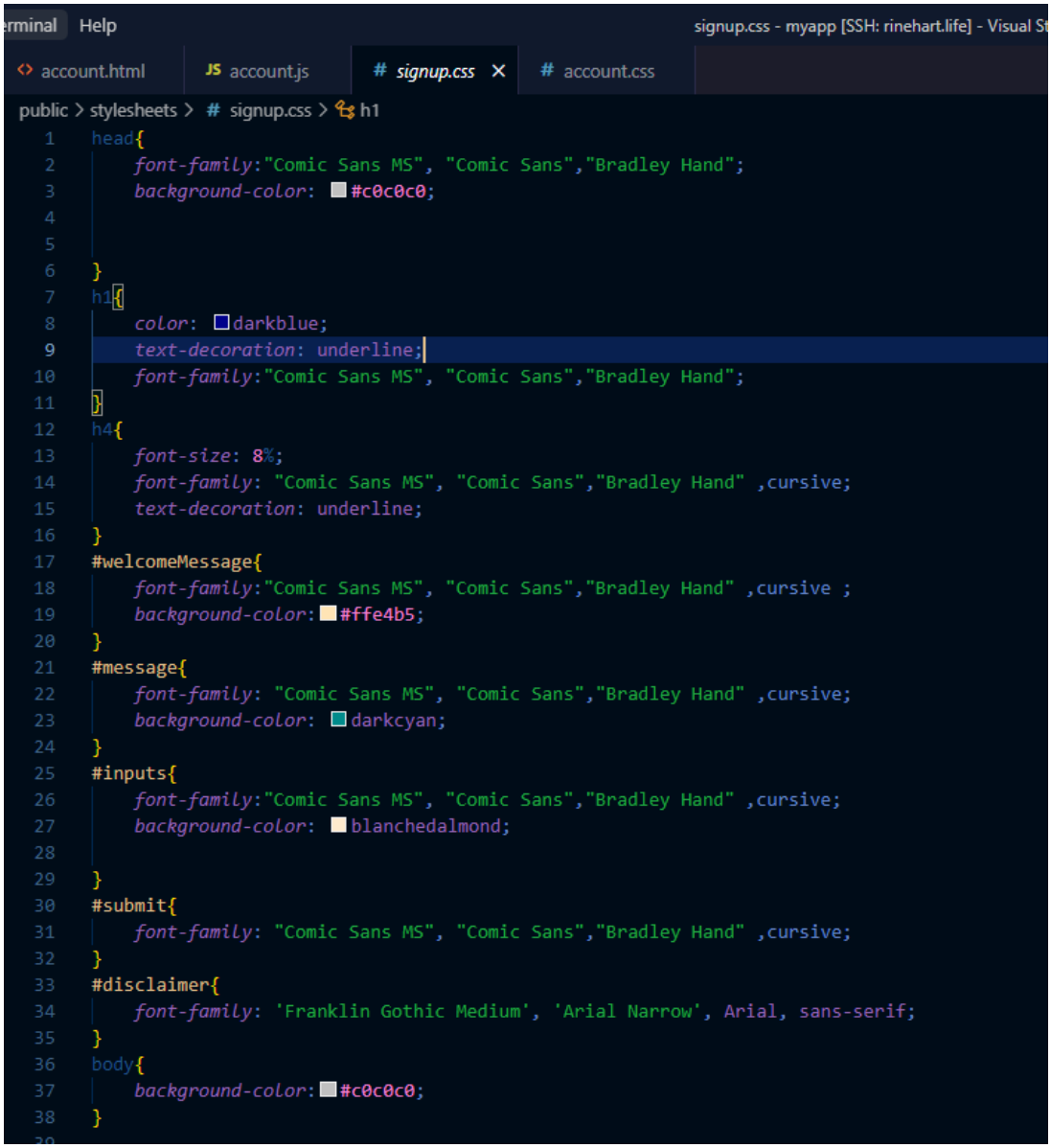
<!--Message from Rusty-->

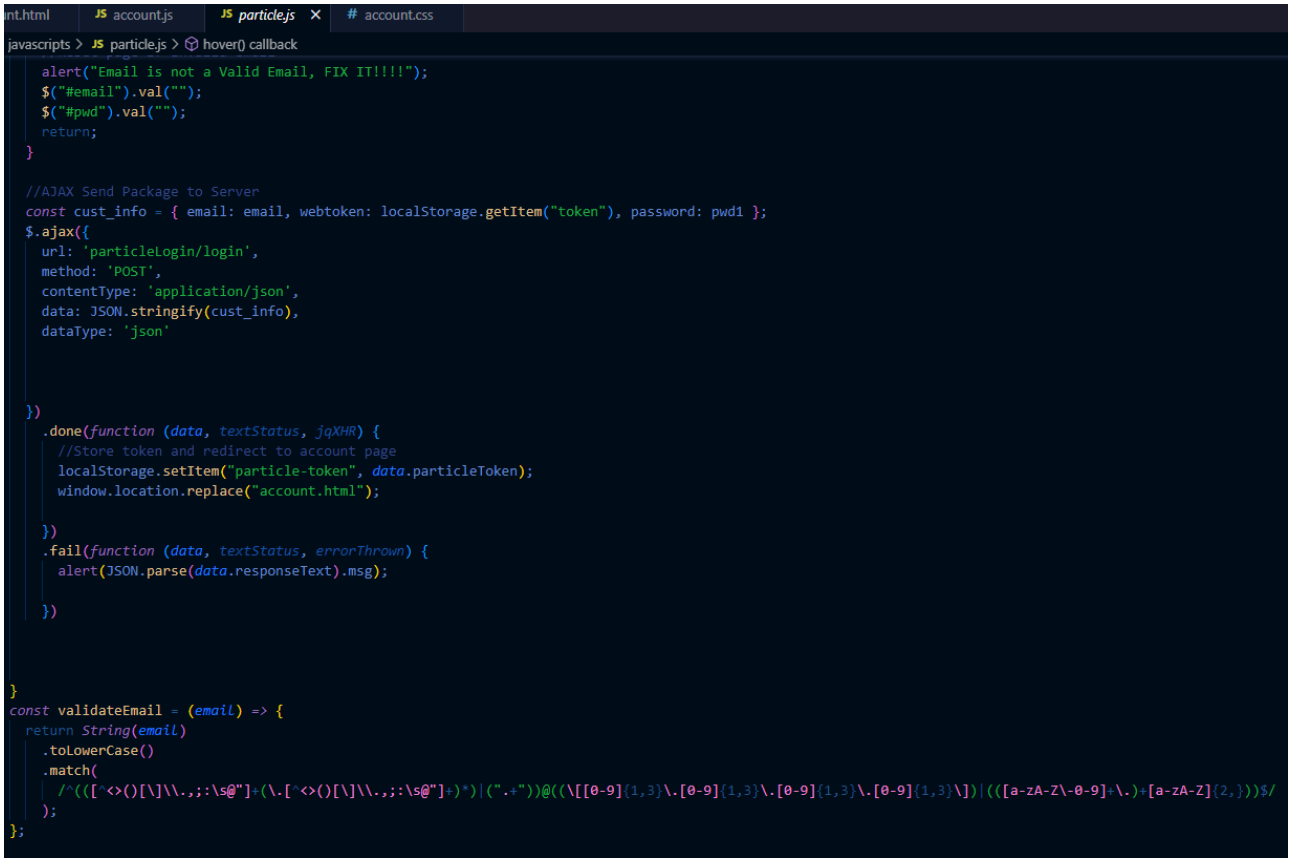
"It was 1956 when I had found the special gene that began to mutate human DNA to an advanced being. I
named these beings 'Homo sapiens homovorus' and became the first one.
The only problem was a need for increased blood to help accelerate muscle creation and fast healing.
This is why I created Rine Heart Monitoring to engineer new devices
that will help Homo Sapiens Homovorus survive and eventually take over this world. " -Dr. Rusty Rinehart,
PHD, MD, DO, JD, DSW.

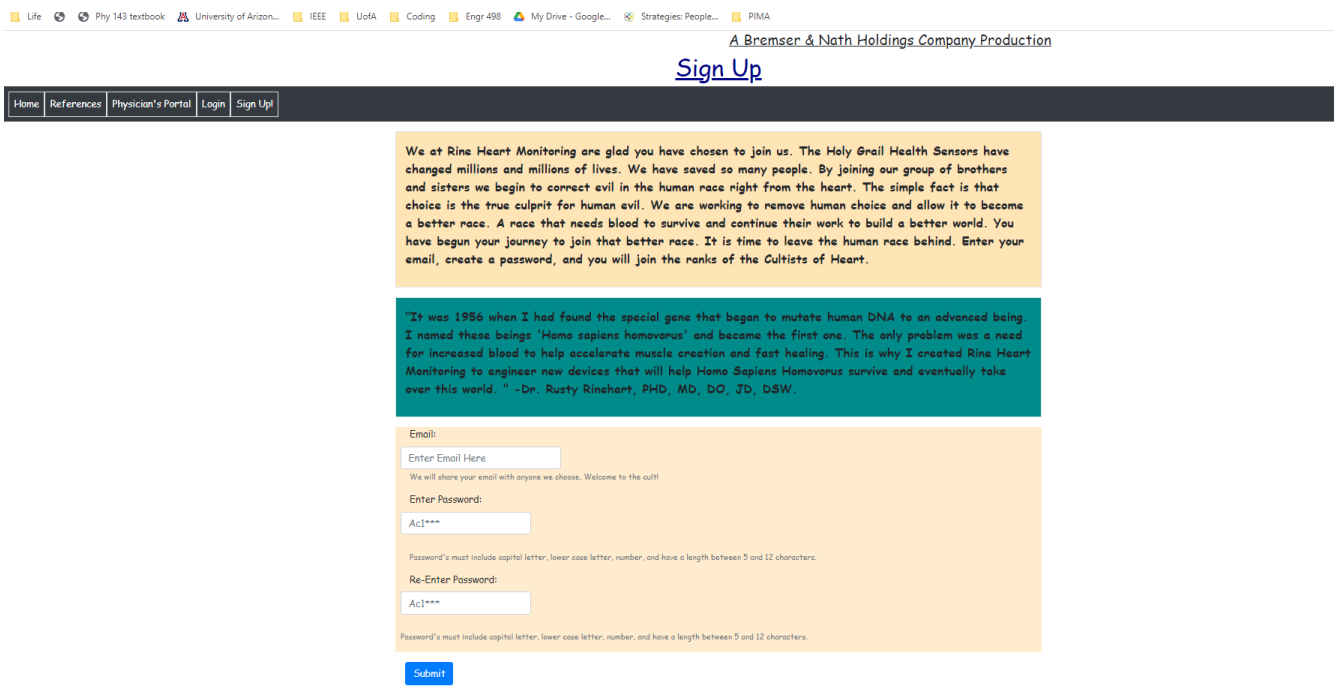
<label for="email">Email:</label>
<input type="text" class="form-control w-auto row" id="email" placeholder="Enter Email Here">
<small id="emailHelp" class="form-text text-muted">We will share your email with anyone we choose,
Welcome to the cult!</small>

<label for="pwd">Enter Password:</label>

<input type="password" class="form-control w-auto row" id="pwd" placeholder= "Ac1**" minlength="5" maxlength="12" size="15">


erminal Help
signup.css - myapp [SSH: rinehart.life] - Visual St
account.html
account.js
signup.css
account.css
public > stylesheets > # signup.css > h1
1 head{
2 font-family:"Comic Sans MS", "Comic Sans","Bradley Hand";
3 background-color: #c0c0c0;
4
5
6 }
7 h1{
8 color: darkblue;
9 text-decoration: underline;
10 font-family:"Comic Sans MS", "Comic Sans","Bradley Hand";
11 }
12 h4{
13 font-size: 8%;
14 font-family: "Comic Sans MS", "Comic Sans","Bradley Hand" ,cursive;
15 text-decoration: underline;
16 }
17 #welcomeMessage{
18 font-family:"Comic Sans MS", "Comic Sans","Bradley Hand" ,cursive ;
19 background-color: #ffe4b5;
20 }
21 #message{
22 font-family: "Comic Sans MS", "Comic Sans","Bradley Hand" ,cursive;
23 background-color: darkcyan;
24 }
25 #inputs{
26 font-family:"Comic Sans MS", "Comic Sans","Bradley Hand" ,cursive;
27 background-color: blanchedalmond;
28
29 }
30 #submit{
31 font-family: "Comic Sans MS", "Comic Sans","Bradley Hand" ,cursive;
32 }
33 #disclaimer{
34 font-family: 'Franklin Gothic Medium', 'Arial Narrow', Arial, sans-serif;
35 }
36 body{
37 background-color: #c0c0c0;
38 }

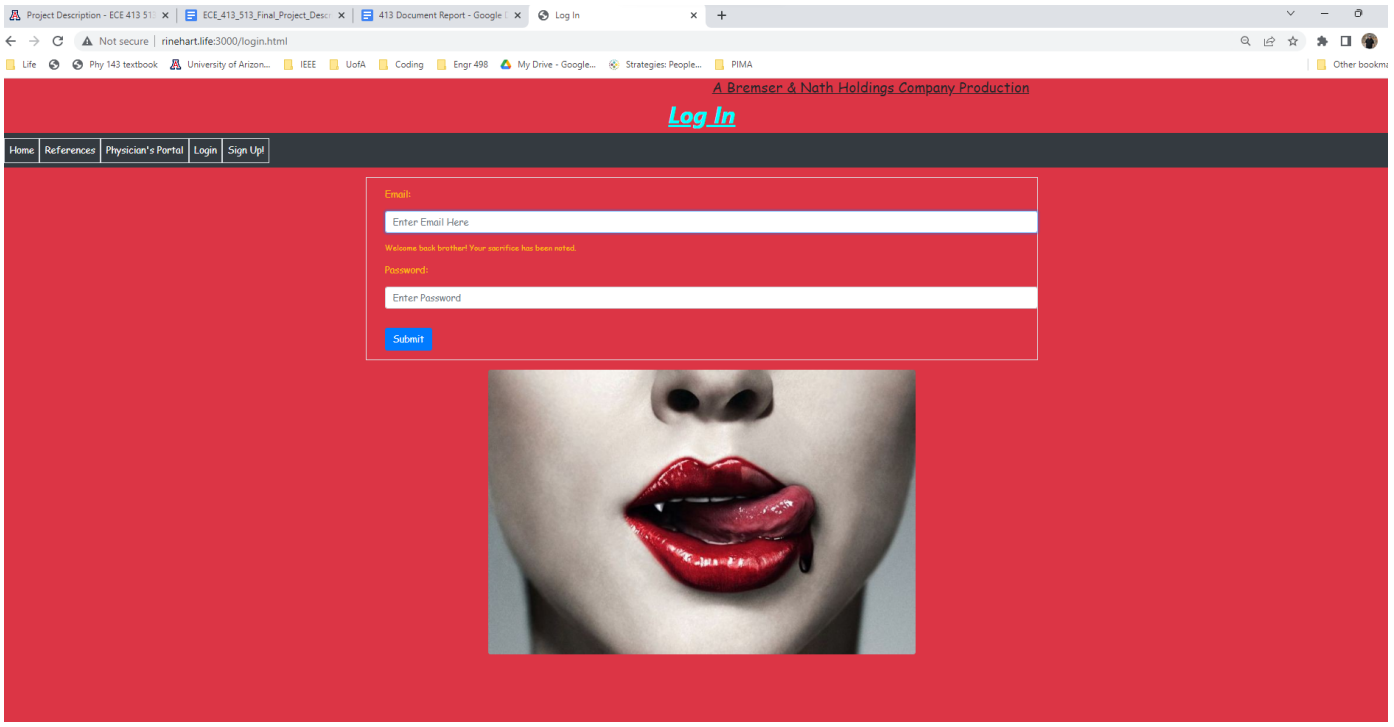




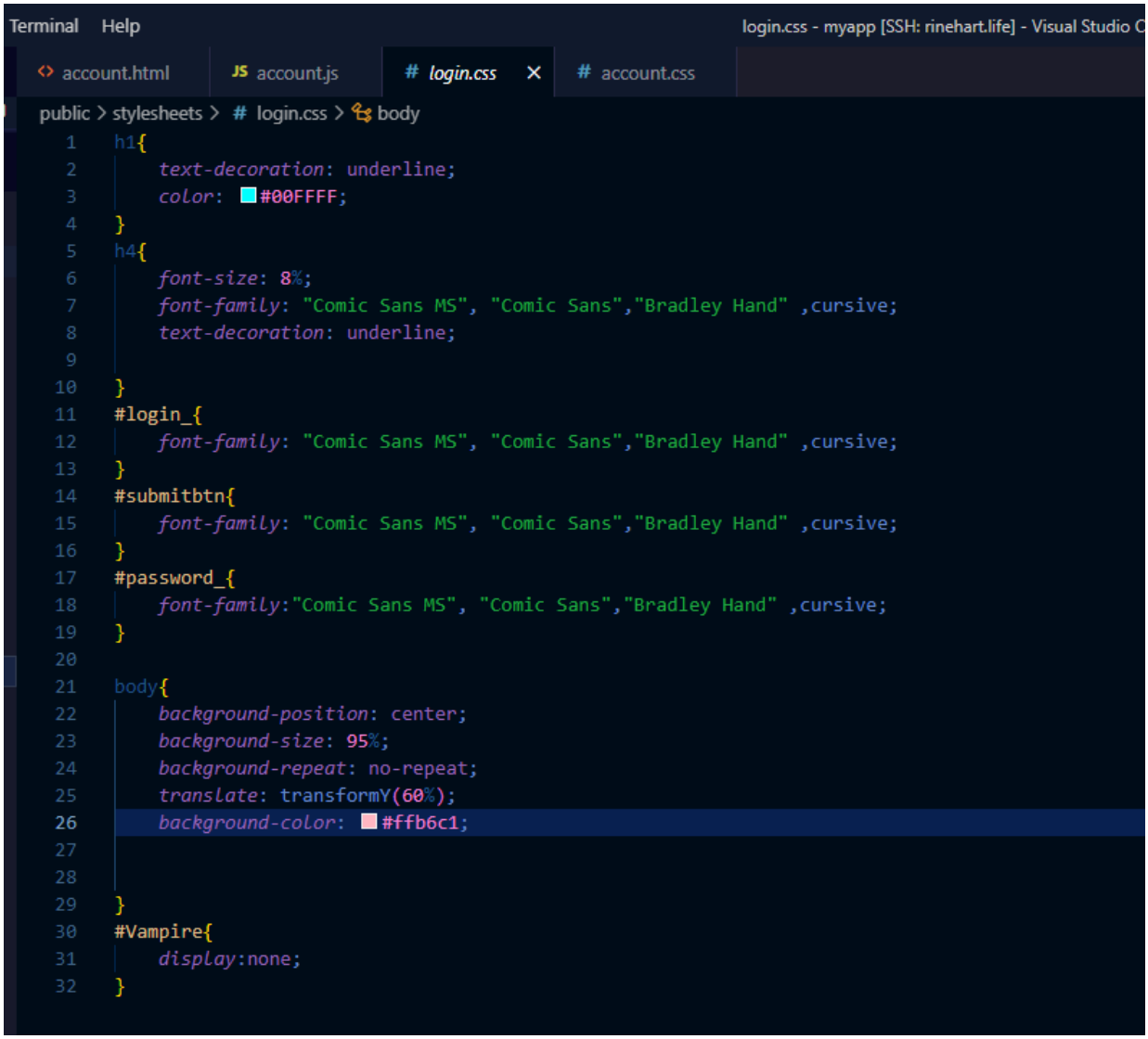
This is the Signup screen. Its pretty simple. We continue our narrative of an evil corp with our messaging. And we take an email and a strong password. Passwords must include capital letter, lower case letter, number, and have a length between 5 and 12 characters. We use regex to find this value in our js file. This email is sent to be stored on the server. The password is

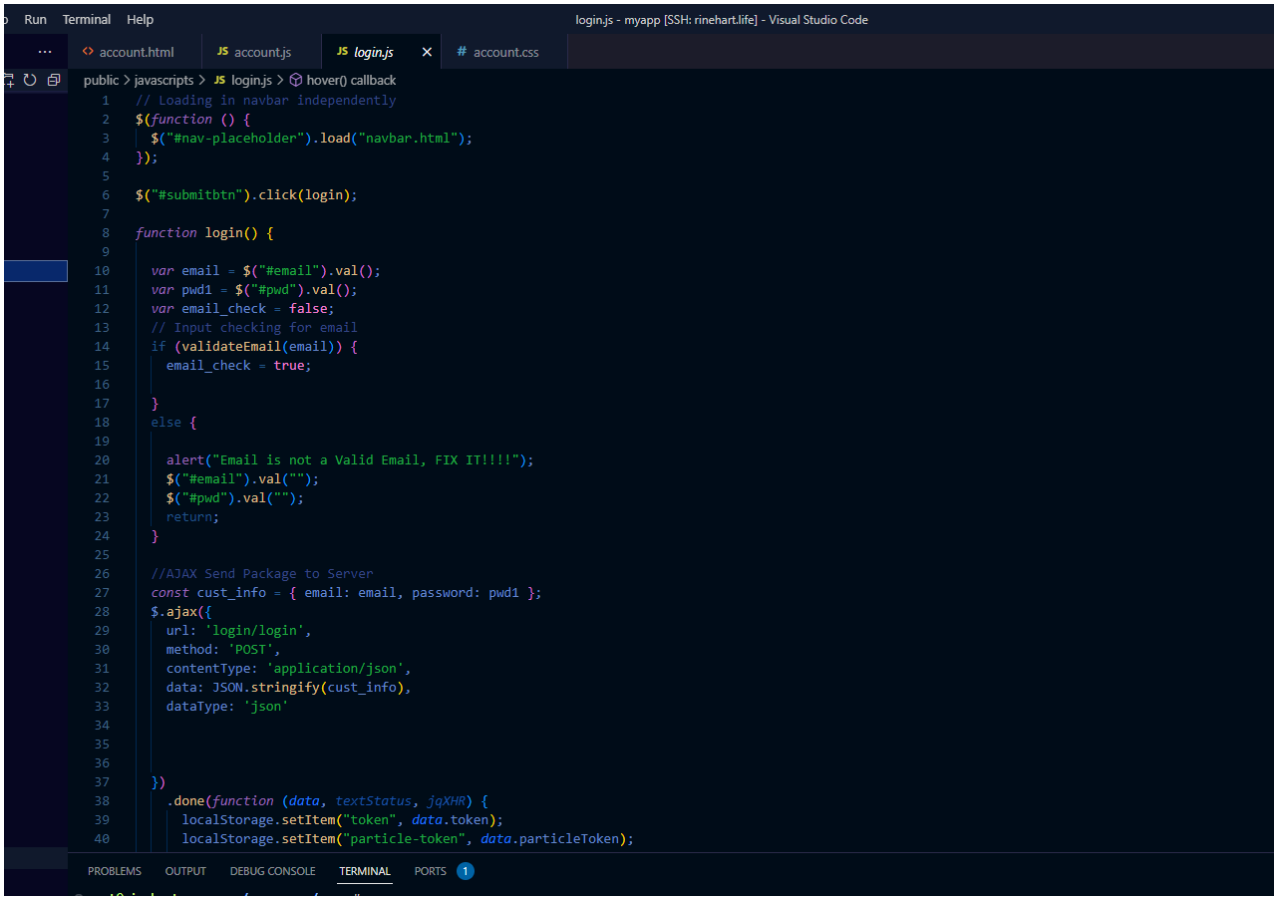
encrypted via a salted hash with a specific keyword. If the server sends any errors back they are alerted to the customer.

Login:



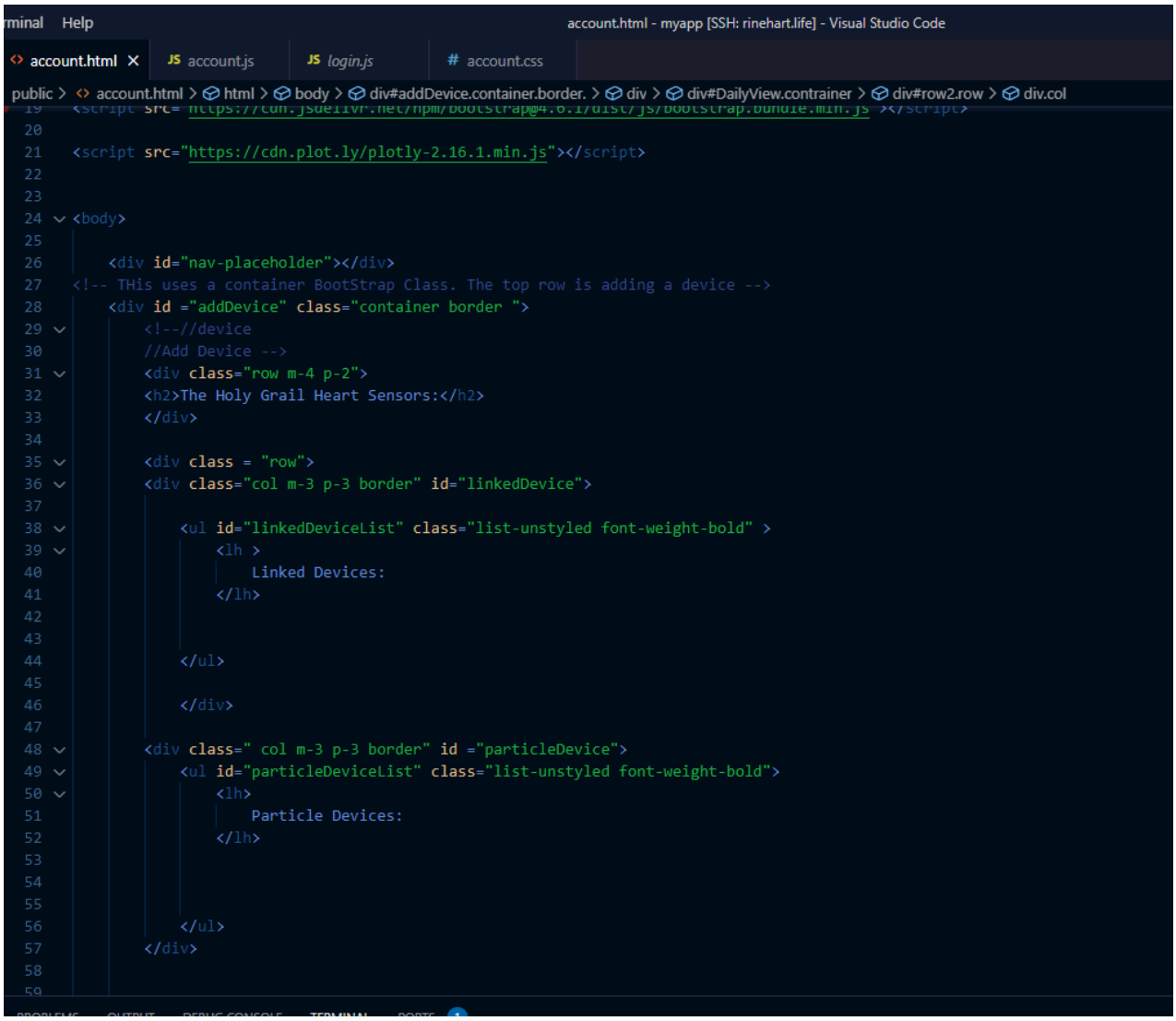

```
account.html | account.js | login.html | account.css
public > login.html > html > body.bg-danger > div > img#Vampire.img-responsive.rounded.mx-auto.d-block.my-3.

<!-- Latest compiled and minified CSS -->
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/css/bootstrap.min.css">
<!-- jQuery library -->
<script src="https://code.jquery.com/jquery-1.10.2.js"></script>
<!-- Popper JS -->
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"></script>
<!-- Latest compiled JavaScript -->
<script src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/js/bootstrap.bundle.min.js"></script>
<body class="bg-danger">
  <div id="nav-placeholder"></div>
    <div class="container border mt-3">

      <div class="form-group" id="login_">

        <label for="email" class="text-warning justify-content-left row m-3">Email:</label>
        <input type="text" class="form-control  justify-content-center row-auto m-3" id="email" placeholder="Enter Email Here">
        <small id="emailHelp" class="form-text text-warning justify-content-center row m-3"> Welcome back brother! Your <span id="sacrifice">sacrifice has been noted.</span></small>
      </div>


      <div class="form-group" id="password_">
        <label for="pwd" class="text-warning justify-content-left row m-3">Password:</label>
        <input type="password" class="form-control justify-content-center row-auto m-3" id="pwd" placeholder="Enter Password">

      </div>
      <input type="submit" class="btn btn-primary justify-content-center row m-3" id="submitbtn" >
    </div>
      <div>
        <img src="/images/blood_1.jpeg" alt="Sexy Vampire" id= "Vampire" class="img-responsive rounded mx-auto d-block my-3 ">
      </div>


</body>
<script src="javascripts/login.js"></script>
</html>
```


Terminal Help
login.css - myapp [SSH: rinehart.life] - Visual Studio C
account.html
account.js
login.css
account.css
public > stylesheets > # login.css > body
h1{
text-decoration: underline;
color: #00FFFF;
}
h4{
font-size: 8%;
font-family: "Comic Sans MS", "Comic Sans","Bradley Hand" ,cursive;
text-decoration: underline;
}
#login_{
font-family: "Comic Sans MS", "Comic Sans","Bradley Hand" ,cursive;
}
#submitbtn{
font-family: "Comic Sans MS", "Comic Sans","Bradley Hand" ,cursive;
}
#password_{
font-family:"Comic Sans MS", "Comic Sans","Bradley Hand" ,cursive;
}
body{
background-position: center;
background-size: 95%;
background-repeat: no-repeat;
translate: transformY(60%);
background-color: #ffb6c1;
}
#Vampire{
display:none;
}



Login is another super simple html. It just asks for the email password combination to login to the website. There is another easter egg on this page that will happen if you hover over something specific. The email and hashed password are then sent to the server. Once the server gives the go ahead that this is indeed the person logged in, it will then move the customer to account.html. We threw a vampire image on the bottom to really embrace the evil vampire organization. I really hope you guys find it as humorous as I did creating it.
CSS is the same as we used Bootstrap4 for everything, the only thing we had to do was change the font family and the background color. Everything else was done inside HTML using Bootstrap.

Account.html


rminal Help
account.html - myapp [SSH: rinehart.life] - Visual Studio Code
account.html
account.js
login.js
account.css
public > account.html > html > body > div#addDevice.container.border. > div > div#DailyView.contrainer > div#row2.row > div.col
<script src="https://cdn.plot.ly/plotly-2.16.1.min.js"></script>
<body>

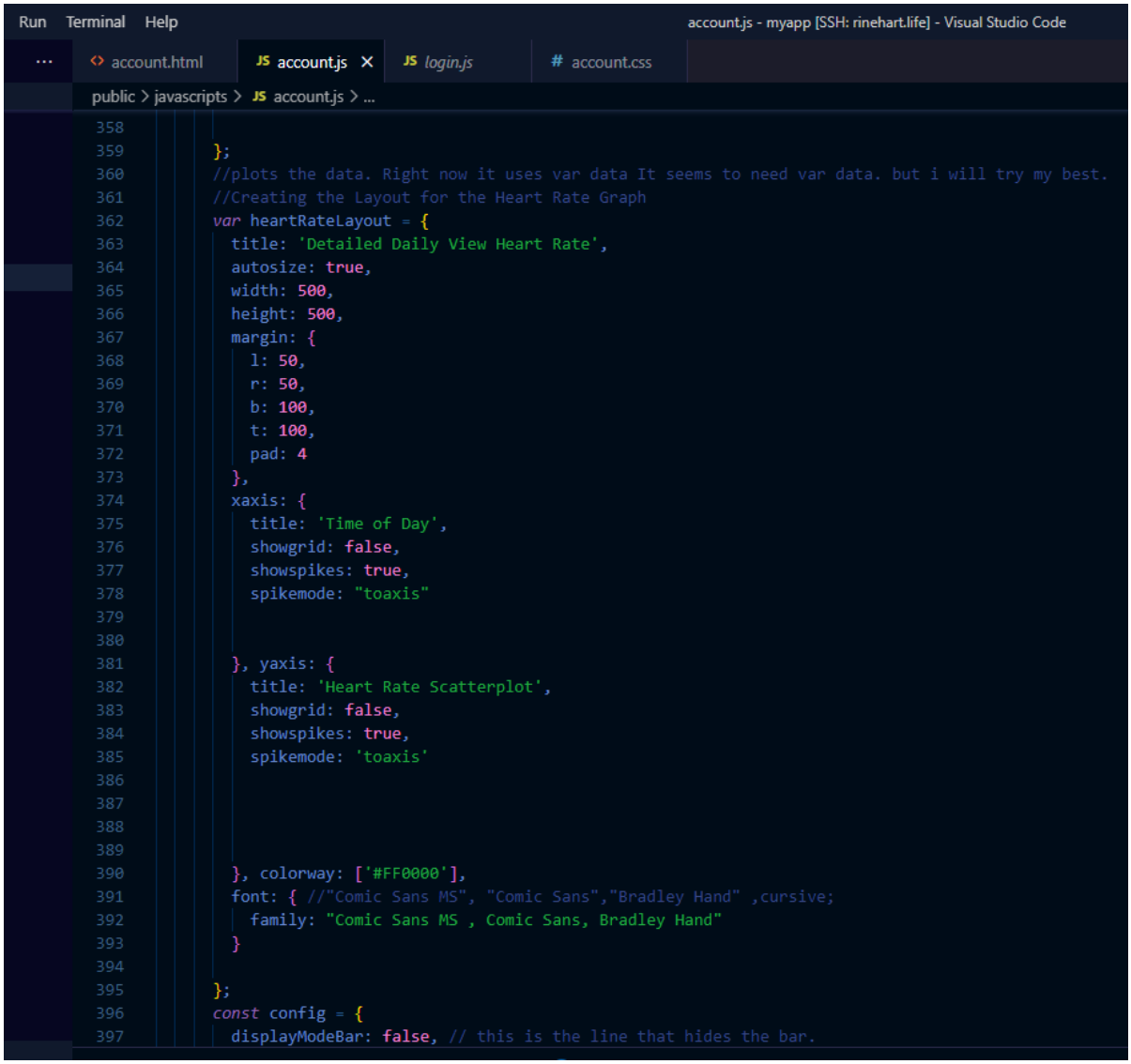
<!-- THis uses a container BootStrap Class. The top row is adding a device -->

<!--//device
//Add Device -->

<h2>The Holy Grail Heart Sensors:</h2>

<ul id="linkedDeviceList" class="list-unstyled font-weight-bold" >
<lh >
Linked Devices:
</lh>

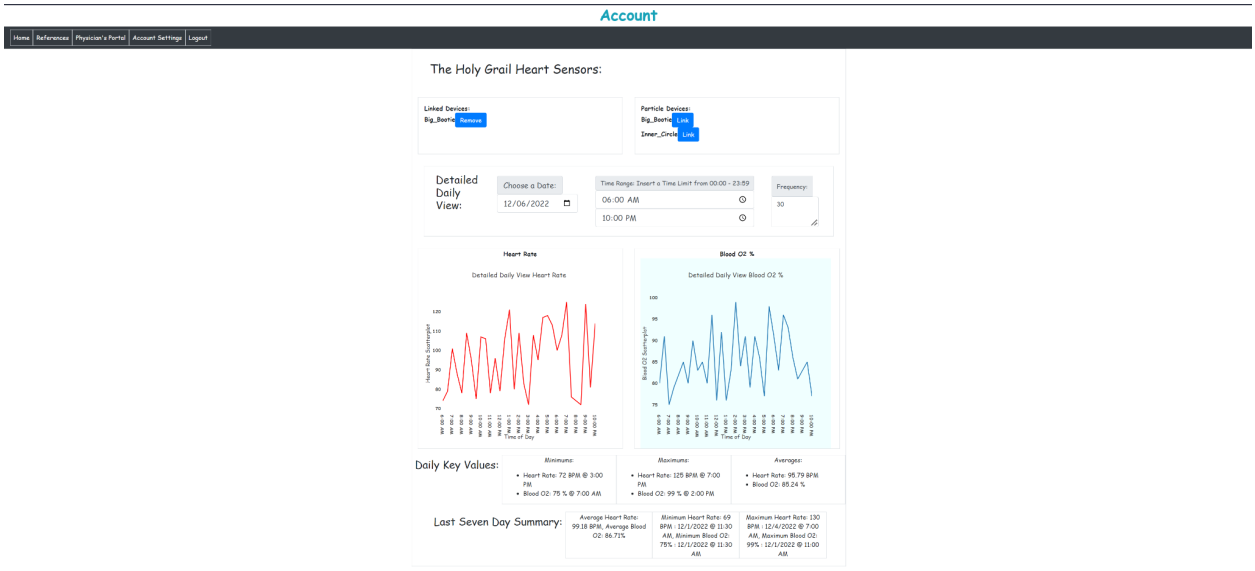
<ul id="particleDeviceList" class="list-unstyled font-weight-bold">
<lh>
Particle Devices:
</lh>

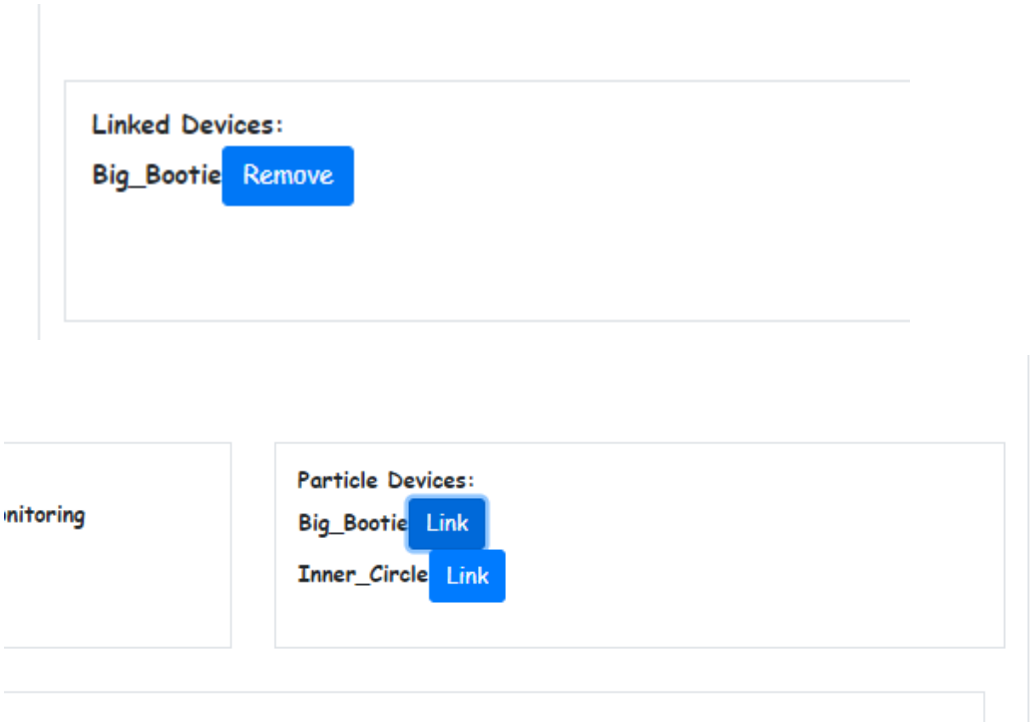

Run Terminal Help
account.js - myapp [SSH: rinehart.life] - Visual Studio Code
account.html
account.js
login.js
account.css
public > javascripts > account.js > ...
358
359 };
360 //plots the data. Right now it uses var data It seems to need var data. but i will try my best.
361 //Creating the Layout for the Heart Rate Graph
362 var heartRateLayout = {
363 title: 'Detailed Daily View Heart Rate',
364 autosize: true,
365 width: 500,
366 height: 500,
367 margin: {
368 l: 50,
369 r: 50,
370 b: 100,
371 t: 100,
372 pad: 4
373 },
374 xaxis: {
375 title: 'Time of Day',
376 showgrid: false,
377 showspikes: true,
378 spikemode: "toaxis"
379
380
381 }, yaxis: {
382 title: 'Heart Rate Scatterplot',
383 showgrid: false,
384 showspikes: true,
385 spikemode: 'toaxis'
386
387
388
389
390 }, colorway: ['#FF0000'],
391 font: { //"Comic Sans MS", "Comic Sans","Bradley Hand" ,cursive;
392 family: "Comic Sans MS , Comic Sans, Bradley Hand"
393 }
394
395 };
396 const config = {
397 displayModeBar: false, // this is the line that hides the bar.

account.html | account.js | login.js | account.css

public > stylesheets > # account.css > #detailed_daily_view

```css
#detailed_daily_view{
    font-family: "Comic Sans MS", "Comic Sans","Bradley Hand" ,cursive;
}
#title{
    font-family: "Comic Sans MS", "Comic Sans","Bradley Hand" ,cursive;
}
#linkedDeviceList{
    font-family: "Comic Sans MS", "Comic Sans", "Bradley Hand", cursive;
}
#particleDeviceList{
    font-family: "Comic Sans MS", "Comic Sans", "Bradley Hand",cursive;

}
#DailyView{
    font-family: "Comic Sans MS", "Comic Sans","Bradley Hand", cursive;
}
#Weekly_View{
    font-family: "Comic Sans MS", "Comic Sans", "Bradley Hand", cursive;
}
#addDevice{
    font-family: "Comic Sans MS", "Comic Sans", "Bradley Hand";
}
```





Account is the most important of our pages. To begin with we have cleverly connected our particle.io account with the website. You can simply hit a link button to link that device to our website. Once the device is linked, you can update the date you are looking at, frequency at which the particle device updates, and time interval at which it will ask you to take measurements. After this you will see our Plotly graphs. Plotly takes in two arrays and can plot. Both are interactive if we only have 5 data points the graphs will plot if we have 100 points the graph will plot. I made some improvements to the plotly graphs by removing the grids. And then also making it so that when you look at different values on the graph, dotted lines go to each of the axes of the graph. Plotly does not have the functionality to show a box inside the graph but I created a box that shows avg, min, and max values for the day. I also created the last seven day summary for min, max and avg.

Back End

I, Ary Nath, was in charge of the expressjs routes, generating the API for our website, and creating the database for our website and all the models relating to that. Additionally, I also handled serving data to the Particle Cloud and receiving data from the Particle Cloud. Here are files that I created:

- models

    - account.js
- routes
    - account.js
    - argon.js
    - login.js
    - particleLogin.js
    - Signup.js
- app.js
- db.js

Backend largely consisted of 2 actions: serving data to and from the database, and serving data to the Particle cloud. Mongoose provided services to interoperate between mongodb data structures and Javascript data structures, allowing us to serve data to the server as well as to the website. In addition to these functions, we also ran basic calculations on the server side, such as finding minimums, maximums, and averages.

Backend implementation was largely smooth, with the exception of Particle promises and utilizing their API. I will go into more detail in the later sections.

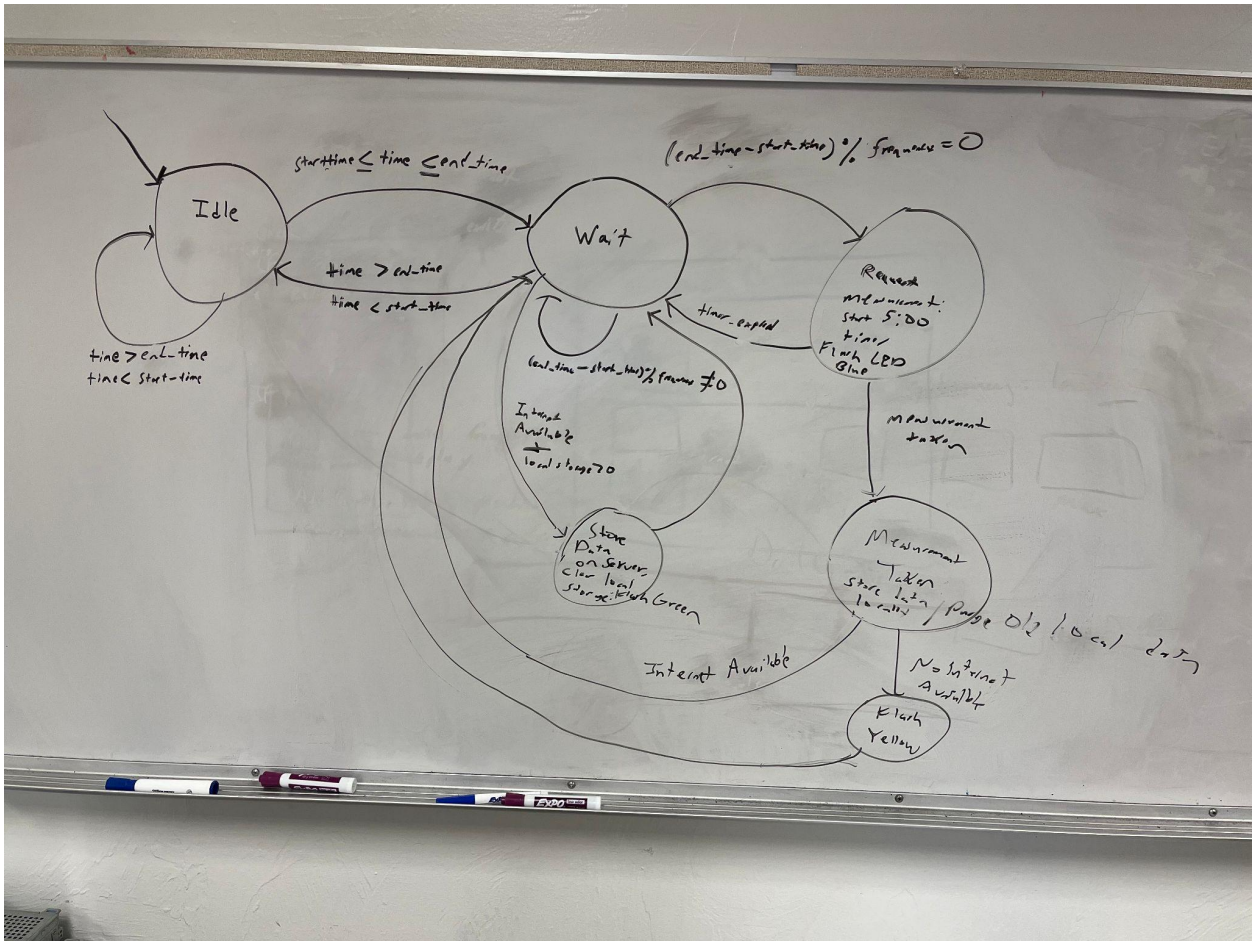
Going through each file in detail:

- models
    - account.js
        - This is the file where we generate our schema that our database stores for each person. We store basic information such as email and password as well as arrays of our recorded data and arrays of devices attached to each account.
            - Email: String and password: String
            - startTime: hours (number) and minutes (number)
            - endTime: hours (number) and minutes (number)
            - recordedValues (Array): timestamp (Date), pulse (number), saturation (number)
            - lastAccess: Date
            - devices (Array): name: String and key: String
            - particleToken: String
            - apikey: String
- routes
    - account.js
        - This is the file with the most routes.
        - /addDevice
            - Requires the JWT web token and the name of the device in JSON form.
            - Returns an array of the devices attached to the account in JSON form.
        - /removeDevice

            - Requires the JWT web token and the name of the device in JSON form.
            - Returns an array of the devices attached to the account in JSON form
        - /getDevices
            - Requires the JWT web token and the particle token in JSON form
            - Returns an array of devices attached to the particle account in JSON form
        - /getLocalDevices
            - Requires the JWT web token in JSON form
            - Returns an array of devices associated with the local account in JSON form
        - /getFrequencyAndTimes
            - Requires the JWT web token, particle token, and name of the device in JSON form
            - Returns the frequency, start time, and end time in JSON form
            - Also sends the frequency, start time, end time, and generated API key to the Particle Cloud
        - /getDailyData
            - Requires a date and a JWT web token in JSON form
            - Returns an array of pulses, oxygen saturations, and timestamps in JSON form
        - /getWeeklyData
            - Requires JWT web token in JSON form
            - Returns average heart rate, maximum heart rate, maximum heart rate timestamp, minimum heart rate, minimum heart rate timestamp, average oxygen saturation, maximum oxygen saturation, maximum oxygen saturation timestamp, minimum oxygen saturation, minimum oxygen saturation timestamp in JSON form
    - argon.js
        - /store
            - This route is webhooked to the Particle cloud
            - Requires event as a String and data as a string in JSON form
                - Data is formatted as “apikey mm/dd/yyyy hh:mm pulse spo2”
            - Returns message stating data received
            - Stores data entry into database
        - /sendFrequency
            - Requires particle token, particle device name, JWT web token, and the newFrequency in JSON form
            - Returns nothing to website
            - Stores new frequency in database for user and sends updated frequency to the Particle cloud

        - /sendStartEnd
            - Requires particle token, particle device name, starting time hour, starting time minute, ending time hour, and ending time minute in JSON form
            - Returns nothing to website
            - Stores new start time and end time into server
            - Sends new start time and end time into Particle cloud
    - login.js
        - /login
            - Requires user email in JSON form
            - Returns JWT token and particle token on success
    - particleLogin.js
        - /login
            - Requires particle email and password in JSON form
            - Returns particle token on success
    - signup.js
        - /signup
            - Requires email and password in JSON form
            - Returns message stating that account has been created
- app.js
    - Standard express.js setup with necessary routers and databases linked
- db.js
    - Links to mongo.db server on the machine

Particle



We implemented a state machine in our Particle C++ code in order to simplify all of the various interactions. The particle starts in the Idle stage where it constantly checks to see if the current time falls in the window specified by the user between the start and the end time. If the user gave no specification, then it will default to checking between 6AM and 10PM. If the device is in the correct window, it will transition to the wait state.

The particle will remain in the wait state until the time falls out of the window. We have a function that checks to see if we are at a certain frequency that needs to be checked. If we are at this time, we transition into the request state. The time is logged when the request is started, and the device will remain in request until the 5 minutes expires. In addition, the device will flash blue as it prompts the user to take a measurement. We added a helper function that uses IR to tell if the user has his or her finger presses against the sensor. We used the MAX30105 library that minimized the complexity of reading heart rate and blood saturation from the sensor.

After the value is read, the data is stored into a vector, and the device transitions back to the wait state. However, if there is no internet, the device will go to a brief “yellow” state where it will flash yellow. The wait state has another check if the particle device is connected to the internet and the local storage vector has elements inside of it. We have a loop that goes

through each element in the vector and send that off to the Particle webhook we set up. All of these states continue to cycle while the particle device is on.

Lessons Learned

- It is crucial to split up the project into reasonable pieces. We made sure to split it up by our strengths so that we were each confident with what we needed to finish.
- We made sure to have after-meetings after each “sprint,” or work session. This allowed us to coalesce and figure out what was done and what still remained.
- Each aspect of the “stack” on a full-stack website was crucial in creating a functional website. Front-end had to account for all of the user interaction. Back-end had to communicate with both the front and the Particle cloud and device and serve the necessary data to both entities. Particle had to take the data from the back structured in a certain way and change its properties accordingly.
- Having a clear line of communication between all team members was crucial to finishing this project. We both had a discord and a text group chat to share ideas and check in on when we needed to finish the project.
- Using libraries such as Bootstrap or JQuery accelerated development and resulted in creating a better website. Using vanilla HTML, CSS, and JS would’ve made creating the website much harder or responsive.

Challenges

- Particle code on the device often crashed on initial builds. This was due to the use of memory intensive data structures such as Strings or other std library methods and structures. We mitigated this by utilizing the Particle String data structure and using simpler data structures to keep the memory used by our firmware low.
- Particle API was difficult to work with. One of our errors was with the publishEvent() promise which inexplicably crashed every time we tried to use it to update the frequency and data even though we were using the correct API key to communicate with the Particle Cloud. We mitigated this through debugging and finding out that using the word “particle” in our API parameters in publishEvent() consistently crashed the API. We changed the offending parameter and the API went back to normal.
- Particle interfacing with the heart sensor was difficult. There were a lot of unknowns with the library we were planning to use due to the library provided by SparkFun and the Particle documentation being optimized for Arduinos and not the Particle device. This made us concerned that the memory constraints of the device would make us unable to utilize the library provided by Particle’s own documentation. We mitigated this by surgically implementing the library block by block and debugging as needed to utilize the library properly.
- Particle firmware often bootlooped the device. On our devices, after flashing the code, the device often remained in DFU mode and couldn’t get out of DFU mode without a

complete reset of the firmware. This usually fixed the issue and Particle's documentation indicated that this cleaned up the firmware to allow for a fresh flash.

- Particle firmware often had inexplicable behavior. On one of our devices, after flashing the device with our firmware, the device kept flashing white and neither wasn't responding to our interactions with the sensor nor going through the state machine. After reading the Particle documentation and troubleshooting guide, we learned that the white LED flashing signified "WiFI off." The documentation stated that this mode triggers if in our code we change a macro indicating the WiFi state of the device. We never even knew this macro existed when writing our code. Firmware restore and reflash did not resolve this issue. However, we were still able to keep going due to one of the teammates having a spare Particle Argon that functioned normally.

Table

	Front End	**Back End**	**Particle**	**Documentation, etc**
Chris Bremser	33.33	33.33	33.33	33.33
Ary Nath	33.33	33.33	33.33	33.33
Rusty Rinehart	33.33	33.33	33.33	33.33

References

- Code Languages Used:
  - HTML
  - CSS
  - Javascript
  - C++
- Websites, libraries, and API's Used
  - Bootstrap: https://getbootstrap.com/
  - W3Schools: https://www.w3schools.com/
  - Plotly: https://plotly.com/javascript/
  - Cave Johnson Bio: https://theportalwiki.com/wiki/Cave_Johnson
  - Particle API: https://docs.particle.io/reference/cloud-apis/javascript/
  - SparkFun MAX301x Particle Sensor Library: https://github.com/sparkfun/SparkFun_MAX3010x_Sensor_Library
  - ExpressJS: https://expressjs.com/
  - Mongoose: https://mongoosejs.com/
  - MongoDB: https://www.mongodb.com/
  - Bcrypt: https://www.npmjs.com/package/bcrypt
  - JSON Web Token: https://www.npmjs.com/package/jsonwebtoken

- Nodemon: https://www.npmjs.com/package/nodemon
- Postman: https://www.postman.com/