Project Documentation

To implement this project, there are three main components:

* Backend - this section of the code links the other components to the MongoDB database, where our data is stored in a customers schema. The routes go to each endpoint where data is posted, retrieved, deleted, etc. in the database.
* Frontend - this section consists of html, javascript, and css files and focuses more on how the website looks and provides functionality for the buttons along with a user interface and displays graphs.
* Embedded - this section provides functionality for the particle device, utilizing state machines to control the LEDs and measurements. It also looks for WiFi connectivity to determine whether to transmit data to the server or store locally.

Results:

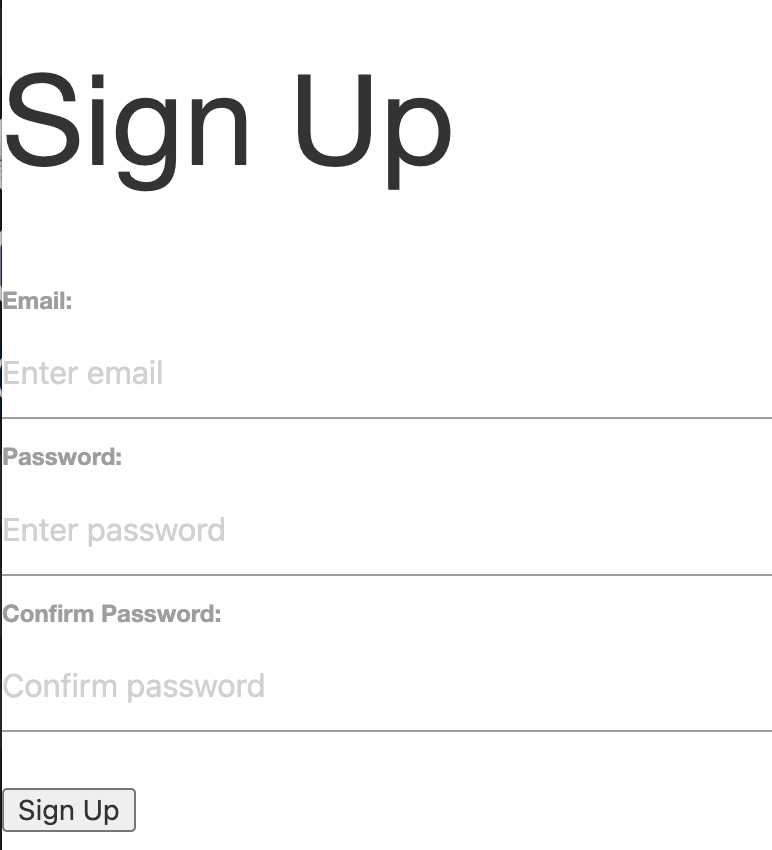


Figure 1: Sign up page - User enters email address and **strong** password that is **hashed and salted** inside database.

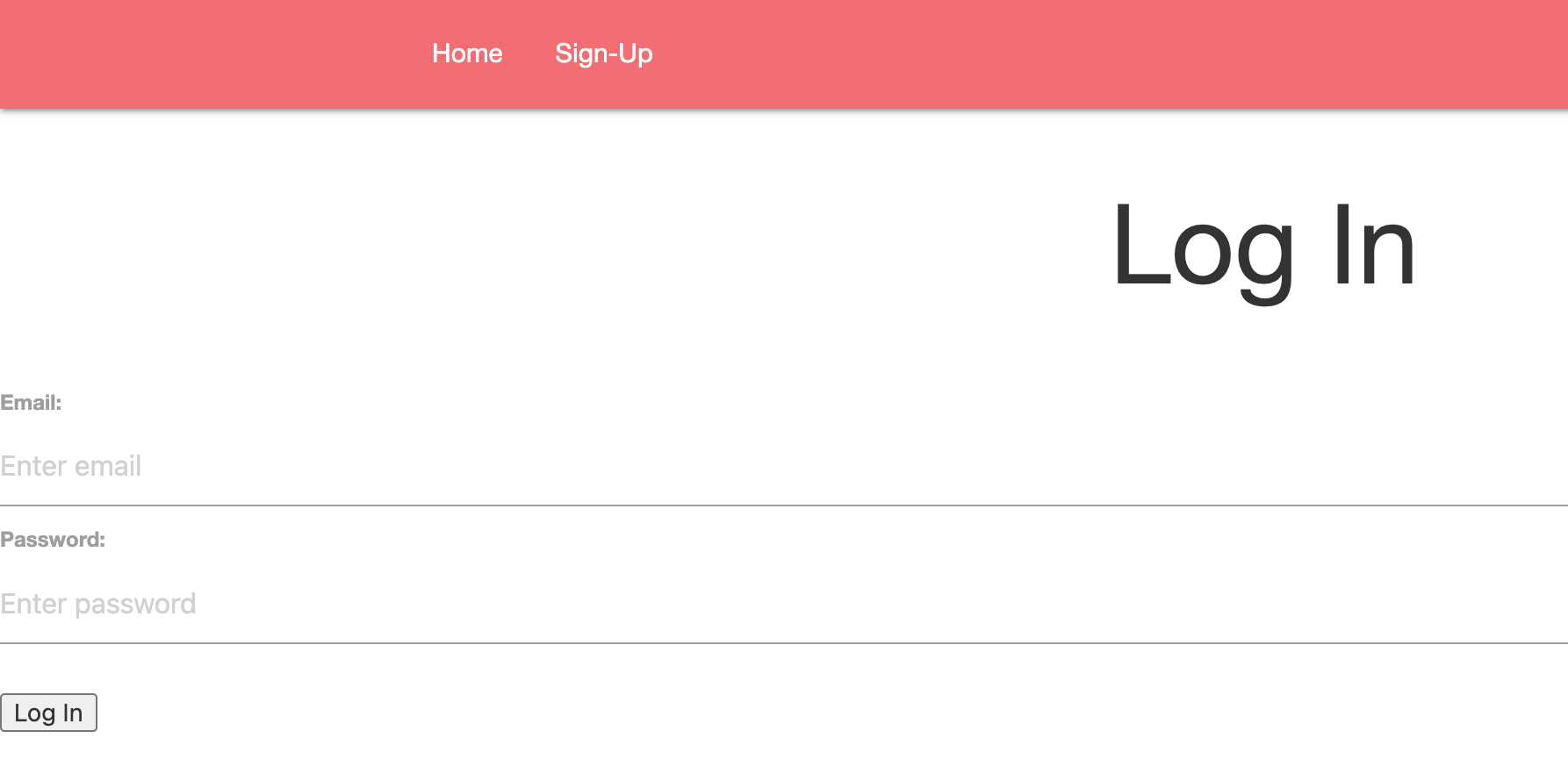


Figure 2: Login page - User logs in with credentials and receives a **token** that is used for authorization throughout the session.

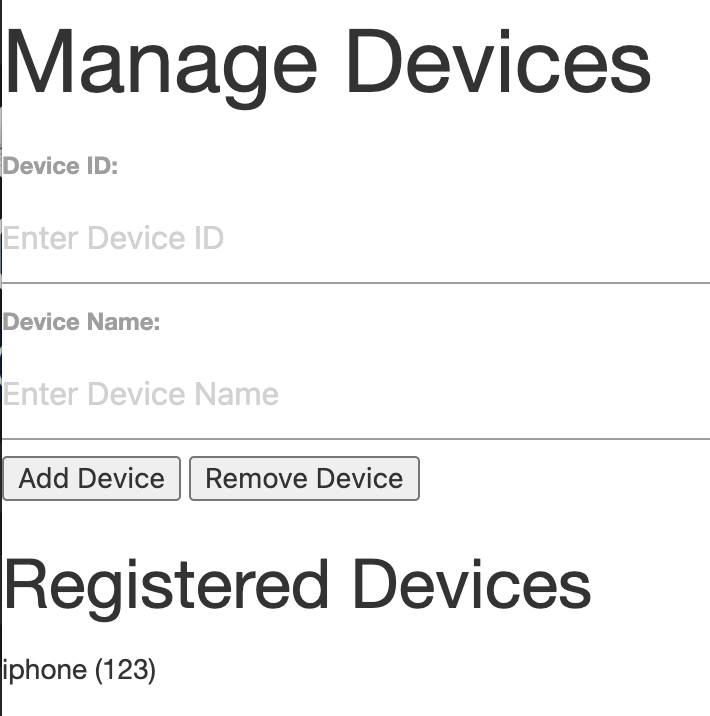


Figure 3: Manage Devices page - User can add or remove devices to their account.

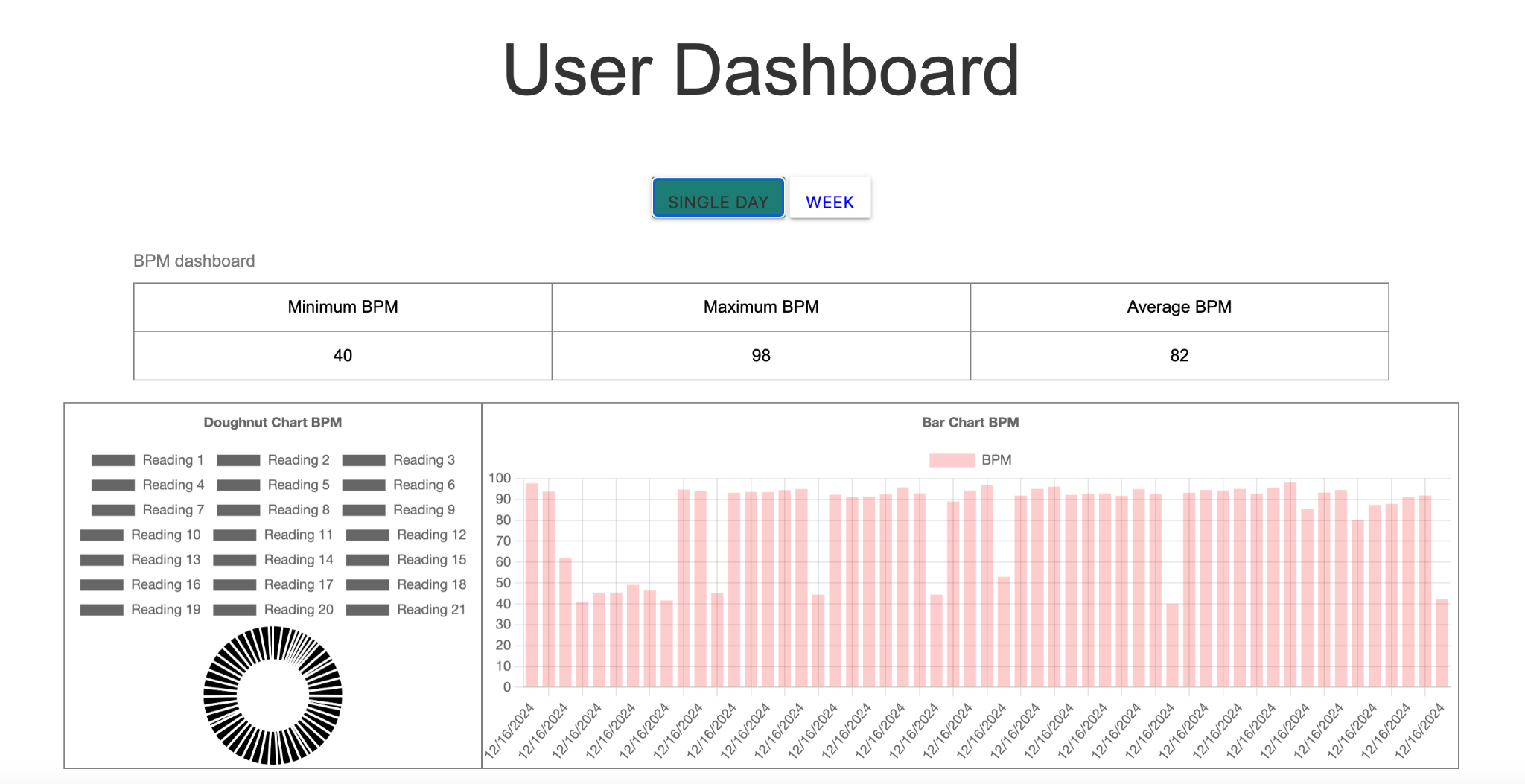


Figure 4: User Dashboard - User can view sensor data by day or week, they can view a bar graph along with minimum bpm, maximum bpm, and average bpm.

File Description

Public: (html)

account.html: account.html contains a Navbar along with buttons whose ids connect back to their .js counterparts taking you to their appropriate functions.

changePassword.html: changePassword.html contains a Navbar and various buttons and input for email, password and new password. At the bottom it also features a text box to provide the user with a response to their action.

Dashboard.html: dashboard.html is heavily based on the Covid example in class, it features many of the same style and script calls. Functions are changed to allow for either a “day” or “week” option to then trigger its associated .js to update the same doughnut and bar graphs along with the table up top.

display.html: display.html is a 404 error page, this contains a single Navbar option to take you back to the index page.

Index.html: index.html is the home page to our website, this holds our first iteration of the Navbar. Additionally there are containers and div’s for our teams images, name and email. Further below are four additional divs for holding the title of our teams videos along with the videos itself.

Login.html: login.html holds the input fields for the user prompting them for an email and password after which pressing the login button triggers the event listener in its accompanying .js file to make a POST request with this data, if it succeeds a token is generated and the user is taken to account.html

manageDevices.html: Allows the user to add a device and remove the device using two buttons. It will list all devices that the user currently has. The page will display messages regarding whether or not the user was successful in either removing or adding a device. If the user could not achieve their goal the message will display an error and explain what had occurred that prevented the user.

Signup.html: signup.html similarly to login.html allows the user to input an email and password additionally it requests a confirmation of this password. After pressing the sign up button the associated .js file is triggered and the email checked to ensure its valid, next the password complexity is checked ensuring it meets our specification of a strong password, and lastly the password and confirm password fields are checked against one another to ensure they match. A POST request with this is sent to our customers/signUp webhook and as long as this passes the user is taken to the login page.

viewData.html: viewData.html is an old file we kept for debugging receiving user data. This html page has no formal nav option and the user would need to manually type it into the url to access the page.

Images:

This folder holds the teams images to be displayed in our index.html

Javascripts:

account.js: account.js handles unique functions for the user such as GET request for customer status and a POST request to change the password and its associated function.

dashboard.js: dashboard.js handles button listening from the html side, depending on the button pressed either the fetchUserDataDay or fetchUserDataWeekly is executed. Within these function blocks the day current day is grabbed and in the case for weekly measurements the day 7 days prior is also found. In both a filter array is made checking against either the day or date range. Matching data is then averaged along with the min and max is calculated. Lastly the graph/table functions are updated. These functions are below and are heavily reused from prior class labs.

login.js: login.js handles the user logging in, it makes a POST request with a provided email and password.

manageDevices.js: manageDevices.js makes a GET request for the localStorage token and lists the devices found in the token. There is also an addDevices function generating an API key for the user's device then making a POST of the token containing the new device API/id/name. Lastly there is a removeDevices function that similarly makes a POST request for the token, getting the device id and sending the request through the removeDevice webhook.

signup.js: signup.js allows the user to create an account. Within the user must make a strong password based on being 8-20 characters, at least one uppercase letter and at least one number. This is sent throughout customers/signUp webhook.

viewData.js: viewData.js is an old file we decided to keep for the use of debugging receiving customer data using the customer/getuserData webhook

Routes:

customers.js: Provides routes for signing up, logging in, managing devices, and retrieving user data from the database.

webhook.js: Provides an endpoint to receive particle data from the webhook configured in particle and saves this data to the database under bpmReading.

index.js: Gets home page to display.

Models:

customer.js: customer schema for email, passwordHash, date of access, devices composed of a device id, device name and apikey, bpm readings composed of the date and their bpm number.

Lessons Learned

* One of main lessons learned is the use of Serial Prints and error handlers. This allowed us to debug our code. With this, we can see where the IoT device is currently at in our code which allowed us to correct the issue.
* Parsing and validating JSON responses on both the backend and frontend taught us how to manage data consistency. We learned how to extract specific fields like bpm and date while handling edge cases such as missing or malformed data.
* We learned how to properly integrate the frontend, backend, and the IoT device. This learning experience will be helpful if we continue working on IoT devices after college.
* We learned that rapid communication and having multiple in person meetings really help speed the project forward. Working in collaboration allowed someone else to easily look through the code and find issues where somebody else could not.
* We learned a bit more about security and privacy especially with learning the salted hash for the password.

Challenges

* One of the main challenges was getting the IoT to run through all the states. In order to fix the issue we used Serial Prints to see where it gets stuck in the code. This allowed us to solve these issues and allowed IoT to run without issue.
* Another issue was the remove device button and add device button. We did not realize until testing that we forgot to check if there was already a device with the same name or ID and we implemented code to check for that.
* Pathing issues: the code was worked on by separate people which meant that initially the code was not integrating itself when combined. We had to go through after adding new code to make sure that it was interacting with the existing code. For example, the view data tab was invisible because while the code was there, there was no path to it on the website.
* Another challenge was the api key. It was unable to get the key and transmit it to the server. We realized that it was taking the incorrect key then it was supposed to have caused the data to be sent to like the particle server but not to our aws.
* Another problem is that the IoT keeps disconnecting. Using the Iphone hotspot is a bit difficult due to it constantly turning off to save power. To fix this we implemented in our loop to check to see if the IoT device can connect to the wifi if disconnected.

Team Participation:

|  | Sebastian Robinson | Hannah Spallas | Tyler Thursby |
| --- | --- | --- | --- |
| Frontend | 70 | 15 | 15 |
| Backend | 15 | 70 | 15 |
| Device Implementation | 15 | 15 | 70 |
| Documentation | 33 | 33 | 33 |
| Demos | 33 | 33 | 33 |

References:

SparkFun Electronics. *MAX30105 Particle and Pulse Ox Sensor Library.* GitHub,<https://github.com/sparkfun/SparkFun_MAX3010x_Sensor_Library>.

"W3Schools" *W3Schools*, <https://www.w3schools.com>.