ECE 413/513 Final Project

Team Members: Todd Peterson, Eli Jacobson, Andres Galvez, Tej Scott

Rev: 1.0

Table of Contents

1. Project Description

This section provides a detailed narrative of how each component of the project was implemented.

1.1 Backend Implementation

1.1.1 Backend Architecture and Tools Used

- **Node.js**: JavaScript runtime for non-blocking, event-driven architecture.
- **Express.js**: Web framework for handling routes, middleware, and requests.
- **MongoDB**: NoSQL database for storing user data and sensor readings.

1.1.2 API Design, Endpoints, and Data Flow

- **API Design**:
    - Handles user data and sensor readings.
    - Supports communication between frontend and backend.
- **Endpoints**:
    - POST /signUp
        - Description: Sign up a new user (either Patient or Physician). Includes validation of input and password hashing.
        - HTTP Method: POST
    - POST /logIn
        - Description: Log in an existing user. Returns a JWT token and user role if credentials are correct.
        - HTTP Method: POST
    - GET /status
        - Description: Get user details (email, role, name, patients, measurements, lastAccess) based on a valid JWT token in the X-Auth header.
        - HTTP Method: GET
    - GET /patient
        - Description: Get patient information based on a provided patient query parameter. Requires valid JWT in X-Auth header.
        - HTTP Method: GET
    - PUT /updatePassword
        - Description: Update the user's password. Requires the current password and the new password.
        - HTTP Method: PUT
    - POST /validatePassword
        - Description: Validate the current password for a user.
        - HTTP Method: POST
    - PUT /updateDevices

        - Description: Update a user's list of devices.
        - HTTP Method: PUT
    - GET /getDevices
        - Description: Get a user's list of devices by their email.
        - HTTP Method: GET
    - GET /getUserEmail
        - Description: Retrieve the email of the authenticated user from the JWT token.
        - HTTP Method: GET
    - GET /getPhysicians
        - Description: Fetch a list of physicians. Access is restricted to users with the role of Patient or Physician.
        - HTTP Method: GET
    - PUT /assignPhysician
        - Description: Assign a physician to a patient. Requires both email (patient) and physicianName.
        - HTTP Method: PUT
    - GET /getAssignedPhysician
        - Description: Get the assigned physician for a given patient based on their email.
        - HTTP Method: GET
    - POST /submitMeasurement
        - Description: Submit measurement preferences for a user, including time range and frequency.
        - HTTP Method: POST
    - GET /getSensorReadings
        - Description: Fetch sensor readings for the authenticated user or a specified patient.
        - HTTP Method: GET
    - POST /webhook
        - Description: Handled the data sent by Particle's webhook and recorded the data for the correct user account, after verifying the correct API key and device id.
        - HTTP Method: POST
- **Authentication**:
    - JWT tokens for secure user authentication, passed via x-auth header.
- **Data Flow**:
    - **Frontend to Backend**: Frontend sends AJAX requests for data retrieval and submission.
    - **Backend to MongoDB**: Backend interacts with MongoDB to store and retrieve data.
    - **Backend to Embedded Devices**: Retrieves sensor data for processing and display.

1.1.3 Key Functionalities Implemented

- **User Authentication**:
  - Uses JWT tokens for secure user authentication.
- **Data Processing**:
  - Processes sensor data (heart rate, oxygen saturation) and calculates statistics (average, min, max).
- **Communication with Embedded Devices**:
  - Backend fetches sensor data for real-time processing and analysis.
- **Chart Generation**:
  - Calculates min/max values for heart rate and oxygen saturation for chart display.
- **Measurement Tracking**:
  - Tracks measurement settings (start time, end time, frequency) and updates the UI.
- **Communication with Embedded Devices**:
  - Handled using CORS middleware for JSON object handling.
  - Utilizes node-fetch to send data to Particle REST API.

1.2 Frontend Implementation

1.2.1 Frontend Design Overview

- **HTML**: Used for the basic structure of the web pages (e.g., forms, tables, charts).
- **CSS**: Styled the page elements and ensured responsive design.
- **JavaScript**: Implements interactive features, form validation, and AJAX requests.
- **Bootstrap**: Utilized for UI components (e.g., buttons, modals, form controls) and responsive design.

1.2.2 Key Features Implementation

- **Dynamic Rendering**:
  - Device Entry Fields: Dynamic form creation using jQuery when adding new device fields (#btnAddDevice).
  - Chart Updates: Dynamically renders heart rate and oxygen saturation charts using Chart.js.
- **Form Submission**:
  - **Signup Form**: Validates inputs (email, password, name, and devices) before submitting data using POST requests.
  - **Measurement Form**: Submits the patient's measurement settings (timeRangeStart, timeRangeEnd, frequency) via POST to the backend.
- **AJAX Requests**:
  - Used for submitting and fetching data (e.g., submitting form data, retrieving patient info, and sensor readings).
  - **Error Handling**: Alerts displayed on form submission or data fetch failures using $.fail method.

- **User Interactions**:
    - **Input Validation**: Ensures form inputs (email, password, name) meet required formats using JavaScript functions.
    - **Device List**: Allows adding multiple devices dynamically, validating the entered device ID and access token before submission.

1.2.3 Integration with the Backend

- **Data Exchange**:
    - Data is sent to and received from the backend using AJAX requests.
    - User data, sensor readings, and measurement settings are sent as JSON to backend routes.
- **Error Handling**:
    - Errors such as invalid email, password, and missing devices are displayed on the UI using the showError function.
    - Server errors (e.g., 404, 401) are handled using $.fail, displaying appropriate messages (e.g., "Server could not be reached," "Email already used").
- **Input Validation**:
    - Ensures that user input (email, password, name, and devices) is validated before sending to the backend.
    - The isValidEmail, isStrongPassword, and isValidName functions validate inputs and prevent submission of invalid data.

1.3 Embedded Device Implementation

1.3.1 Devices Used

- **Particle Argon:**
    - Wifi and bluetooth enabled device that was used to receive and send data to the website and database. This device also told the user when to take a reading and if the reading was sent to the server or was stored locally, due to connection issues.
- **Max 30102:**
    - Pulse oximeter sensor that took readings from the patients.

1.3.2 Device Communication

- **Argon and Max 30102:**
    - The Argon collected the readings from the Max 30102 using an I2C protocol.
- **Argon and the server:**
    - Data was sent to MongoDB, using a webhook set up on the particle website. The data was received by the server and sent to the user account in MongoDB that had the correct device registered to it.
    - Measurement parameters were sent back to the device using Particle's REST API method, specifically Particle.function().

1.3.3 Functions and Descriptions

- **setup():**
    - Handled the initialization of the Argon device.
        - Set up the REST API functions, connected the device to Wifi and Particle's server.
        - This function also synced the time on the device and did the initialization of the Max 30102.
- **loop():**
    - This function was the loop that continued to run for the functionality of the device. It implemented a synchronous state machine with seven different states (Idle, wait, measure, process, OffLineUpload, error, default).
    - This function continued to loop and execute code based on the state it was in.
- **timeValid():**
    - This function was used to determine if the current time was in the range of time the user had specified to take measurements.
    - The function returned true or false based on the results.
- **updateWait(String waitTemp):**
    - This function updated the wait time in between measurements based on input from Particle's REST API.
    - This function received a string, turned it into an integer and updated the variable in the device, and then returned a 1 to the REST API.
- **updateStart(String startTemp):**
    - This function updated the start time for the measurements based on input from Particle's REST API.
    - This function received a string, turned it into two integers (one for hour and one for minute) and updated the variables in the device, and then returned a 1 to the REST API.
- **updateEnd(String endTemp):**
    - This function updated the end time for the measurements based on input from Particle's REST API.
    - This function received a string, turned it into two integers (one for hour and one for minute) and updated the variables in the device, and then returned a 1 to the REST API.

---

2. File Descriptions (1 point)

2.1 Public Files

- **index.html**: The main entry point for the website, introduces the team, and has login and sign up options.
- **login.html**: The page for when you log into the website.

- **dashboard.html**: At the top of most pages, provides the buttons to navigate to other pages.
- **display.html**: Handles the display of the 401 unauthorized message for when the user is not authorized to visit a certain webpage.
- **editPatient.html**: Page for patients to edit their account details.
- **editPhysician.html**: Page for physicians to edit their account details.
- **patientAccount.html**: Start page for patients once they log in.
- **physicianAccount.html**: Start page for a physician once they log in.
- **references.html**: Page that contains all the references to libraries, APIs, and other code used.
- **signupPatient.html**: Page for patients to sign up on the website, gets all their necessary information.
- **signipPhysician.html**: Page for physicians to sign up.
- **viewPatientinfo.html**: Page for physicians to view their patients.
- **site.css**: Contains basic styling for the site.
- **Style.css**: Contains more specific styling for different pages, classes, ids, and addresses responsive design.

3.2 Routes

- **/routes/users.js**: Handles all the routing in between pages, the communication between the website and the database through HTTP requests, and includes middleware.
- **/routes/particle-webhook.js**: Handles the webhook form the Particle website and also has the function to send parameter updates to the device.

3.3 Models

- **/models/users.js**: Defines the schema for the user, both physician and patient.

3.4 JavaScript Files

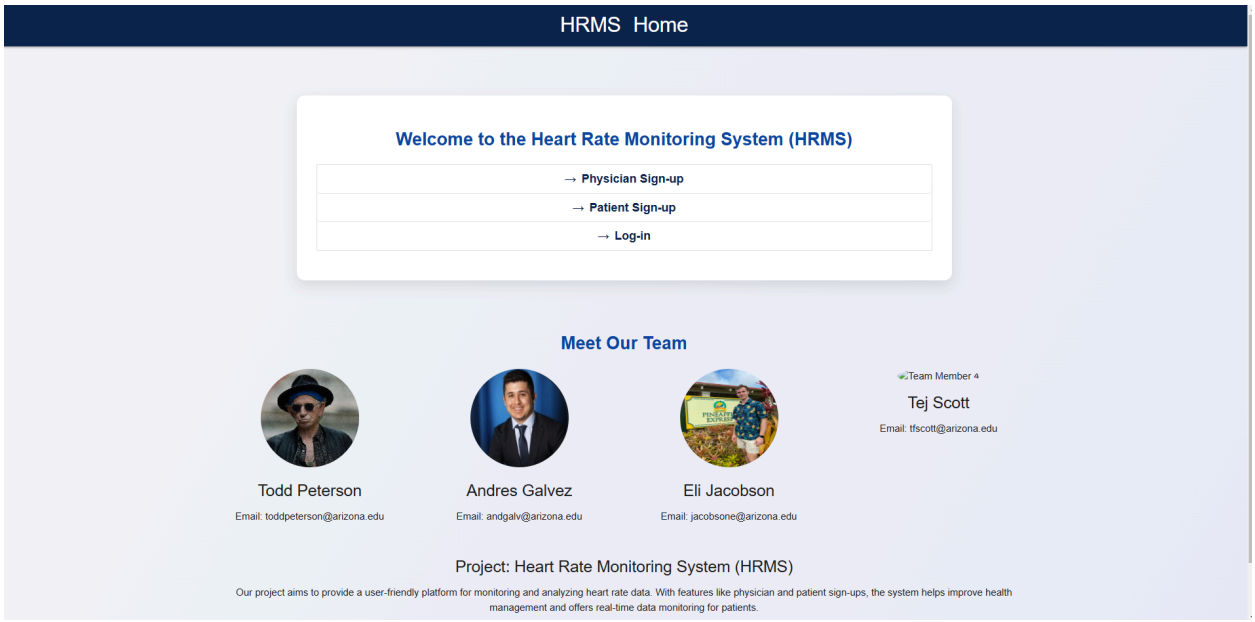
- **app.js**: Page that sets up and configures the server, defines routes, and initializes the middleware.
- **db.js**: Page that connects the website to MongoDB.
- **login.js**: The page that handles the dynamic actions when you log into the website.
- **dashboard.js**: Dynamically fetches user information to display at the top of the website.
- **editPatient.js**: Page that retrieves patient data from the database.
- **editPhysician.js**: Page that retrieves physicians data from the database.
- **patientAccount.js**: Page that retrieves patient readings to display and updates patient measurement parameters.
- **physicianAccount.js**: Page that retrieves patient data from the database.
- **signupPatient.js**: Page that checks if the patient has signed up correctly and sends the data to the database.

- **signupPhysician.js**: Page that checks if the physician has signed up correctly and sends it to the database.
- **viewPatientinfo.js**: Page that pulls data from the database for the patient to view.

---

4. Results (0.5 points)

4.1 Website Screenshots

- 

Home Log In

Patient Sign Up

Email:

Enter email

Password:

Enter password

Name:

Enter name

Register a Device

Device Id:

Enter device id

Device Access Token:

Enter device access token

Add Another Device

Sign Up

Home

Heart Rate Monitoring System Log In

Email:

Enter email

Password:

Enter password

Log In

Home | Edit Profile | Log Out

Logged In as: John Pork

Physician Dashboard

John Pork's Patients:

Patients List

carter bryant
7-Day Average: 75 bpm
Max Heart Rate: 90 bpm
Min Heart Rate: 60 bpm
View More Information

Joe Craddock
7-Day Average: 75 bpm
Max Heart Rate: 90 bpm
Min Heart Rate: 60 bpm
View More Information

Home | Edit Profile | Log Out

Logged In as: John Pork

Edit Physician Portal

Change Password

Current Password:
Enter current password

New Password:
Enter new password

Change Password

Home | Edit Profile | Log Out

Logged In as: John Pork

carter bryant's Patient Dashboard

Current Measurement Time Range and Frequency

Start Time: 6:30 PM
End Time: 10:10 PM
Frequency: 10 minutes

Set Measurement Time Range and Frequency

Start Time:

--:-- --

End Time:

--:-- --

Frequency (minutes):

Submit

Weekly Summary | Detailed Daily View

Weekly Summary

Average Heart Rate: 75 bpm
Minimum Heart Rate: 60 bpm
Maximum Heart Rate: 90 bpm

Home | Edit Profile | Log Out

Logged In as: carter bryant

Patient Dashboard

Current Measurement Time Range and Frequency

Start Time: 6:30 PM
End Time: 10:10 PM
Frequency: 10 minutes

Set Measurement Time Range and Frequency

Start Time:

--:-- --

End Time:

--:-- --

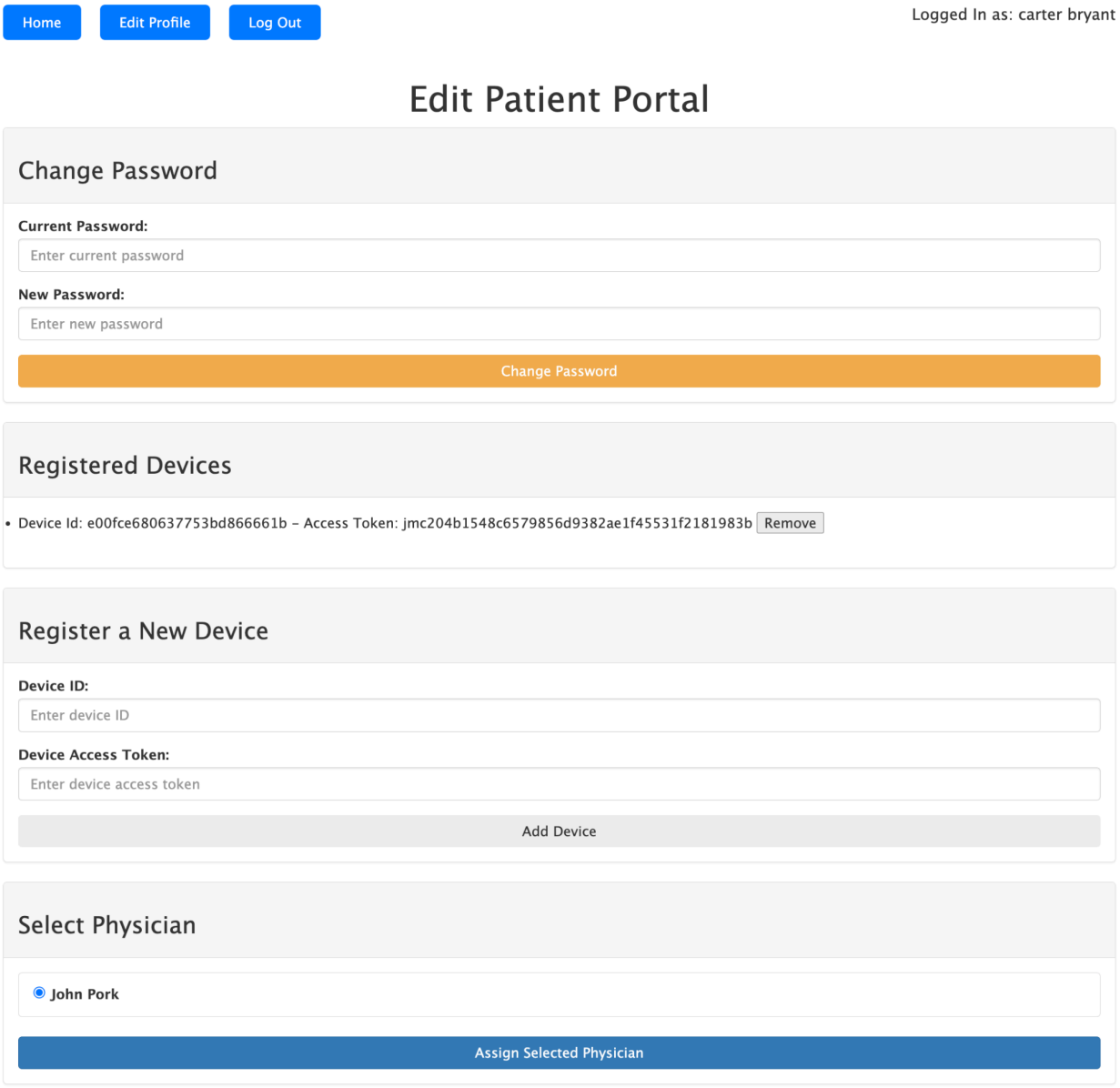
Frequency (minutes):

Submit

Weekly Summary | Detailed Daily View

Weekly Summary

Average Heart Rate: 75 bpm
Minimum Heart Rate: 60 bpm
Maximum Heart Rate: 90 bpm



4.2 Embedded Device Outputs

Home | Edit Profile | Log Out

Logged In as: carter bryant

Patient Dashboard

Current Measurement Time Range and Frequency

Start Time: 6:30 PM
End Time: 10:10 PM
Frequency: 10 minutes

Set Measurement Time Range and Frequency

Start Time:

--:-- --

End Time:

--:-- --

Frequency (minutes):

Submit

Weekly Summary | Detailed Daily View

Weekly Summary

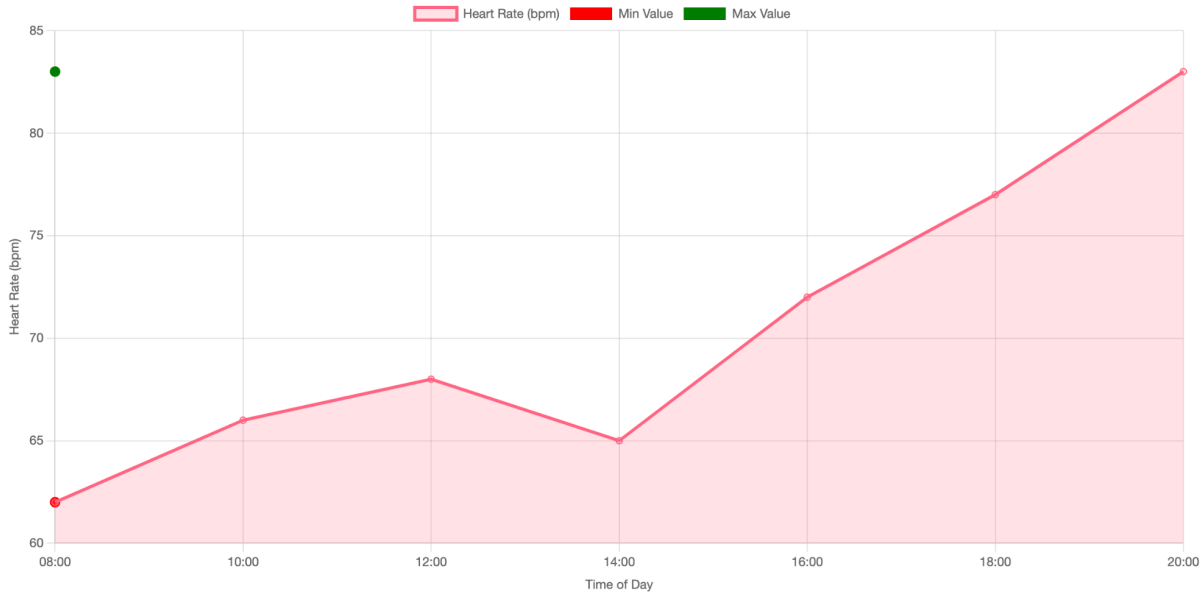
Average Heart Rate: 75 bpm
Minimum Heart Rate: 60 bpm
Maximum Heart Rate: 90 bpm

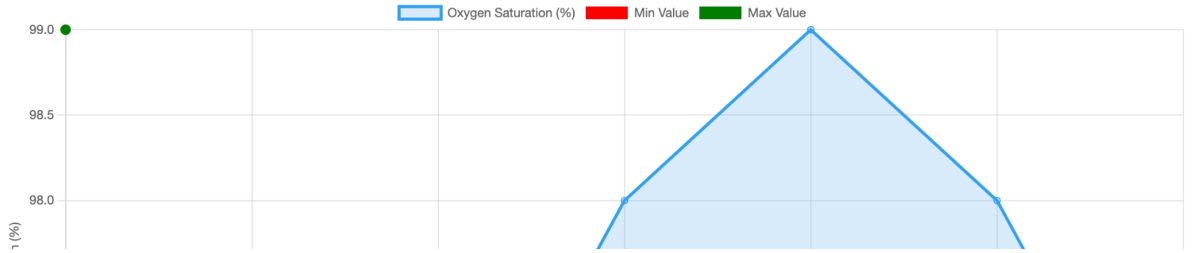
Weekly Summary | Detailed Daily View

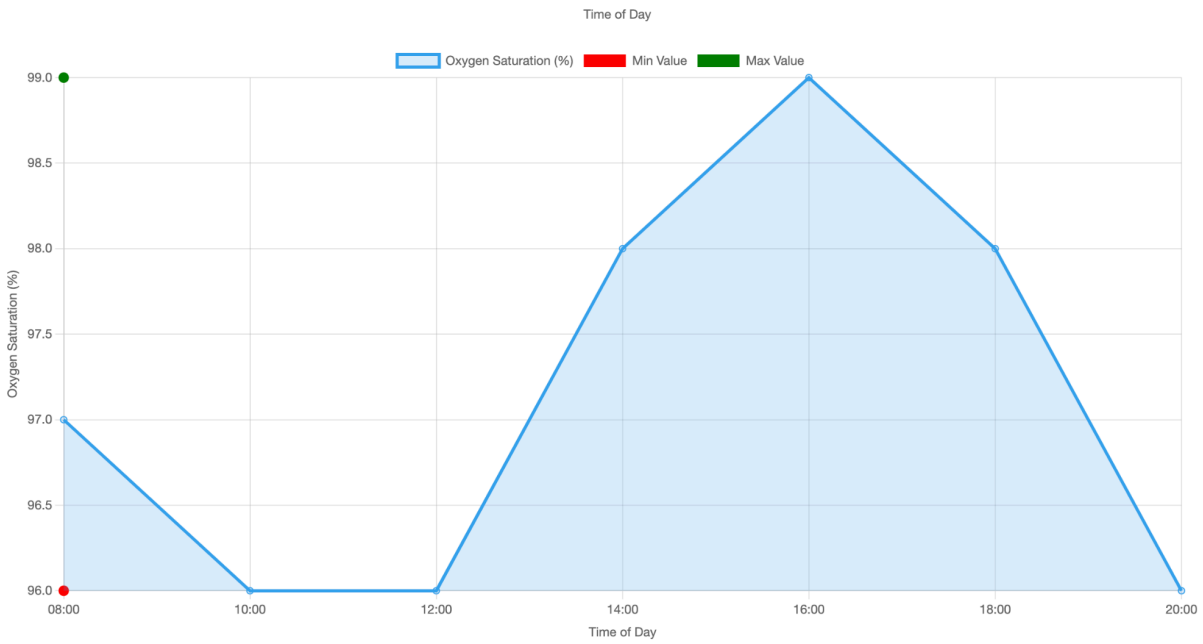
Detailed Daily View

Select Day:

2024-12-04







---

5. Lessons Learned (0.5 points)

Provide at least five lessons learned from the project implementation:

- **Lesson 1**: One lesson learned was learning how to use Particle's REST API to connect a web application to a internet enabled device
- **Lesson 2**: Another lesson learned was how to set up a webhook, and how to verify the data, and send it to the correct place in the database.
- **Lesson 3**: Additional lesson learned was the process of setting up the database schema when there are attributes that are dependent on the role of a user.
- **Lesson 4**: The process of debugging API calls. Whether using debugging console statements or stepping through the code, we learned how to properly fix API issues depending on the error code.
- **Lesson 5**: The use of outside libraries outside the class such as Chart.js. We learned how to implement and display functional charts from a third-party library.

---

6. Challenges (0.5 points)

Describe at least five challenges faced and how you resolved them:

- **Challenge 1**: One challenge we faced was the whole team being able to collaborate on the project. We solved this by emailing each other, using microsoft teams, and sharing/updating code on github.
- **Challenge 2**: Another challenge we faced was soldering the Max 30102 sensors. We solved this by learning how to solder and then soldering the devices at the Engineering Design Center.
- **Challenge 3**: A third challenge the team faced was two team members working on the same file unknowingly. We resolved this by having a call to discuss our changes, what our goals for the file were, and utilizing GitHub branches.
- **Challenge 4**: A challenge occurred in the process of having team members access someone's AWS instance via SSH. This was resolved by creating additional .pem files in AWS and sharing the proper private key with them to access it.
- **Challenge 5**: When some members are using macOS, when trying to add files or update files to the AWS instance terminal, they have to add files manually. To resolve this issue, we assigned the team members that use Windows to use WinSCP to easily drag files from their local machine to the instance terminal.

---

7. Team Contributions (0.25 points)

Provide a table showing each team member's contribution to various components of the project:

Team Member	Frontend (%)	Backend (%)	Embedded Device (%)	Documentation (%)	Demo (%)
Todd Peterson	15	15	25	33	50
Eli Jacobson	15	20	75	33	25
Andres Galvez	70	65	0	33	25
Tej Scott	0	0	0	0	0

---

8. References (0.25 points)

Third-party APIs and Libraries

Materialize CSS - A modern front-end framework based on Google's Material Design principles.
Website: https://materializecss.com

Google Icon Font - A library of Material Design icons provided by Google. Website: https://fonts.google.com/icons

Node.js Express - A popular server-side framework for building web applications. Website: https://expressjs.com

Body-Parser - A Node.js middleware for parsing incoming request bodies. Website: https://www.npmjs.com/package/body-parser

http-errors - A Node.js library for creating HTTP error objects. Website: https://www.npmjs.com/package/http-errors

CORS Middleware - The server-side middleware for handling json objects. Website: https://www.npmjs.com/package/cors

Node-Fetch API - The server-side code for connecting to Particle REST API. Website: https://www.npmjs.com/package/node-fetch

MAX30102 Sensor - A cpp library of code that can be used to read values from the Max 30102 sensor. Website: https://github.com/sparkfun/SparkFun_MAX3010x_Sensor_Library

Code

Node.js Code - The server-side code to handle routing, requests, and responses for the application. Source: Custom code used for the Heart Rate Monitoring System (HRMS).

HTML/CSS Code - The front-end code for structuring and styling the Heart Rate Monitoring System (HRMS) interface. Source: Custom code used for the user interface of HRMS.
