

HEART RATE MONITORING PROJECT

ECE 413/513 FINAL TERM PROJECT
UNIVERSITY OF ARIZONA

DECEMBER 16TH, 2024

TEAM MEMBERS

AARON COLIN

ASHISH KHADKA

TARON BASHAR

Contents

Project Overview.....	4
Introduction.....	4
Frontend.....	4
Html.....	4
CSS.....	5
JavaScript.....	5
Responsive design.....	5
Backend.....	6
Node.js.....	6
Express.....	6
MongoDB.....	6
API logic.....	7
Device.....	7
Firmware.....	7
State machine.....	8
Server interaction.....	8
Secure implementation using LLMs.....	8
File Overview.....	10
Public.....	10
Static files (HTML, CSS, JS).....	10
Routes.....	11
API endpoints.....	11
Models.....	12
MongoDB schemas.....	12
Firmware.....	12
IoT scripts.....	12
Config.....	13
.env.....	13

app.js.....	13
package.json.....	14
Results.....	14
Website.....	14
Navigation.....	15
Team Members Page.....	17
Patient Login and Sign-Up Pages.....	17
Physician Login and Sign-Up Page.....	19
Devices.....	21
Charts.....	21
Portal.....	21
IoT Data.....	22
Tables.....	22
Trends.....	22
Key Takeaways.....	22
Challenges.....	23
Team Contribution.....	24
References.....	25

Project Overview

Introduction

The Heart Rate Monitoring project, developed as part of the ECE 413/513 final project at the University of Arizona, integrates IoT technology, web development, and embedded systems to create a platform for monitoring heart rate and blood oxygen saturation levels in real time. The system allows users and physicians to track critical health metrics conveniently through a user-friendly web interface hosted on an AWS EC2 instance.

The backend, built with Node.js and Express, manages user authentication, device management, and database interactions using MongoDB. The hardware leverages a Particle Argon IoT device equipped with a MAX30102 sensor to collect and transmit physiological data via GET and POST requests, with offline storage for resilience.

Key features include:

- **Data Visualization:** Interactive charts display weekly trends and daily time-series data.
- **Device Management:** Patients can add or remove devices, while physicians (ECE 513) manage patient data.
- **Secure Communication:** JWT-based API authentication and HTTPS ensure data protection.

This project highlights the practical application of IoT in healthcare, showcasing the seamless integration of hardware, software, and cloud technologies to create a scalable and multidisciplinary system.

Frontend

The frontend of the Heart Rate Monitoring project provides an intuitive and responsive user interface, allowing users to interact seamlessly with the system. It is designed to visualize data, manage devices, and ensure accessibility across a range of devices, including desktops, tablets, and mobile phones. The frontend implementation uses HTML, CSS, and JavaScript for structure, styling, and interactivity, respectively, with responsive design principles ensuring a consistent user experience.

Html

The structure of the frontend is implemented using HTML, which serves as the foundation for the user interface. Key components include:

- **Page Layout:** The HTML files define the layout of the web application, including navigation bars, content sections, and forms. For example:
 - o The **home page** introduces the application, presenting a brief description and navigation options.
 - o The **device management page** allows users to add or remove IoT devices using simple forms.
 - o The **data visualization page** displays heart rate and oxygen saturation trends using embedded charts.
- **Integration with Backend:** Dynamic content is generated by the backend using template engines like EJS or Pug, enabling the server to populate HTML pages with real-time data.

CSS

Cascading Style Sheets (CSS) were used to style the application, ensuring that it is visually appealing and user-friendly. The styling is consistent across pages to maintain a cohesive look and feel. Key styling features include:

- **Custom Design:** CSS provides branding elements like consistent color schemes and typography to give the application a professional appearance.
- **Grid and Flexbox Layouts:** CSS layouts ensure that elements like charts, forms, and buttons are aligned and responsive, regardless of screen size.
- **Thematic Elements:** CSS is used to style key sections, such as the navigation bar, buttons, and tables displaying device and user data.

JavaScript

JavaScript is used to add interactivity and dynamic behavior to the frontend. Key functionalities implemented include:

- **Dynamic Content Loading:** Fetching real-time data from backend APIs to display on charts and tables without requiring a page reload.
- **Form Validation:** JavaScript validates user input (e.g., device IDs) before sending it to the server, ensuring errors are caught early.
- **Chart Rendering:** Libraries like Chart.js are used to render interactive graphs for visualizing heart rate and oxygen saturation trends.

- **Event Handling:** JavaScript handles user actions such as button clicks (e.g., adding/removing devices) and updates the DOM accordingly.

Responsive design

Responsive design ensures that the web application provides an optimal user experience across different devices and screen sizes. This is achieved through:

- **Media Queries:** CSS media queries adjust the layout, font sizes, and element visibility based on the screen size.
- **Fluid Grid Layouts:** Elements are arranged using percentage-based widths, ensuring they resize dynamically as the viewport changes.
- **Mobile-First Design:** The application is designed to work seamlessly on smaller screens first, with additional styles added for larger screens.

Backend

The backend of the Heart Rate Monitoring project serves as the primary intermediary between the IoT devices, the database, and the frontend. It is responsible for handling all API requests, managing data storage and retrieval, and ensuring secure communication. The backend implementation utilizes Node.js for the runtime environment, Express for server and route management, and MongoDB for data persistence.

Node.js

In this project, Node.js powers the backend, enabling the efficient processing of requests and responses. It handles the communication between the Particle Argon IoT device, the MongoDB database, and the frontend interface. Node.js modules like `.env` are used to load sensitive configuration details such as database credentials and API keys from environment variables.

Express

The server uses Express for managing API endpoints and middleware integration. The project structure includes a modular organization of routes:

- **patients.js:** Handles patient operations such as registering and managing devices.
- **physicians.js:** Implements physician-specific functionalities, including viewing aggregated data for patients.
- **devices.js:** Manages device registration, removal, and data synchronization with the database.

Middleware is used extensively for tasks like request parsing, cookie handling, and CORS configuration. Error-handling middleware ensures consistent responses for invalid or unauthorized requests.

MongoDB

MongoDB is the database used to store users, devices, and measurement data. It is integrated into the backend using the Mongoose library for schema management and database interactions.

- **Schemas:**
 - **User Schema:** Includes fields for storing patient and physician information, login credentials, and device associations.
 - **Device Schema:** Tracks IoT device metadata, including device IDs and ownership.
 - **Measurement Schema:** Stores timestamped heart rate and blood oxygen readings for each registered device.
- **Cloud Hosting:** MongoDB Atlas is used to host the database, ensuring high availability and secure access via SSL/TLS.

API logic

The backend exposes RESTful APIs to manage all interactions between the system components. Key endpoints include:

- **User Management:**
 - `/api/register`: Registers new users.
 - `/api/login`: Authenticates users and returns JWT tokens for secure API access.
- **Device Management:**
 - `/api/devices`: Supports adding, updating, and deleting IoT devices.
- **Measurement Data:**
 - `/api/measurements`: Handles fetching and aggregating measurement data for visualization on the front end.

Each API endpoint includes authentication middleware that validates JWT tokens to ensure authorized access. POST endpoints process data from IoT devices, while GET endpoints retrieve data for visualization and analysis.

Device

The device implementation for the Heart Rate Monitoring project leverages a **Particle Argon IoT device** equipped with a **MAX30102 sensor**. The device is responsible for collecting real-time physiological data, including heart rate and blood oxygen saturation levels, and transmitting

this data securely to the backend server. The implementation is divided into three key components: firmware, state machine logic, and server interaction.

Firmware

The firmware, written in C++, runs on the Particle Argon IoT device and handles all operations related to data collection, processing, and communication. Key functionalities of the firmware include:

- **Sensor Initialization:** The firmware configures the MAX30102 sensor to read heart rate and SpO2 levels. The sensor is periodically activated based on the state machine's schedule.
- **Data Acquisition:** Using I2C communication, the firmware retrieves data from the sensor and prepares it for transmission to the server.
- **Error Handling:** The firmware monitors sensor and network statuses, retrying operations in case of failures.

State machine

A state machine is implemented within the firmware to manage the device's various operational states efficiently. The state machine ensures the device is energy-efficient and reliable by controlling the behavior of the sensor and communication modules.

Key states include:

1. **Idle State:** The device remains idle, conserving energy until the next measurement cycle.
2. **Measurement State:** The device activates the MAX30102 sensor, collects heart rate and SpO2 data, and processes it for transmission.
3. **Transmission State:** Data is sent to the server via HTTP POST requests.
4. **Error State:** If a sensor or network error is detected, the device retries operations or logs the error.

Server interaction

The device communicates with the backend server to transmit collected data and receive configuration updates. Interaction with the server is facilitated through secure HTTP requests.

Key components of server interaction include:

- **Data Transmission:** The firmware sends heart rate and SpO2 readings as JSON objects to the server using HTTP POST requests.
- **Secure Communication:** API keys are embedded in the firmware to authenticate device requests with the server.

- **Offline Handling:** If the device cannot reach the server, data is stored locally in a buffer and retransmitted when the connection is restored.

Secure implementation using LLMs

The secure implementation of this project involved identifying and mitigating key security vulnerabilities (CWEs) in the codebase. This was accomplished using Large Language Models (LLMs) like ChatGPT to detect weaknesses and provide recommendations for secure code practices. The identified weaknesses included **CWE-352 (Cross-Site Request Forgery)**, **CWE-311 (Missing Encryption of Sensitive Data)**, and **CWE-400 (Uncontrolled Resource Consumption)**. The table below summarizes the issues, and the mitigation strategies implemented.

CWE-ID (Web Link)	Description	Domain (HTML/CSS/JS/Firmware)
CWE-352: Cross-Site Request Forgery (CSRF)	CSRF allows attackers to trick users into making unintended requests to the server without their consent.	HTML, JavaScript
Detected: Yes Mitigated: Yes		
Vulnerable Code: <pre>app.use(function (req, res, next) { // Set up CORS res.setHeader('Access-Control-Allow-Origin', '*'); res.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, PATCH, DELETE'); res.setHeader('Access-Control-Allow-Headers', 'X-Requested-With,content-type'); res.setHeader('Access-Control-Allow-Credentials', true); next(); });</pre> LLM Prompt: <i>"How can I protect this Node.js application from CSRF attacks? Suggest a solution and provide an example implementation."</i> Modification Approach: <ul style="list-style-type: none"> • Implemented CSRF protection middleware using csrf. • Added CSRF tokens to forms to ensure requests originate from authenticated users. 		
CWE-311: Missing Encryption of Sensitive Data	Missing encryption of sensitive data	JavaScript, Firmware
Detected: Yes Mitigated: Yes		
Vulnerable Code: <pre>var http = require('http'); var server = http.createServer(app);</pre> LLM Prompt:		

"How can I enforce HTTPS on an Express.js server to ensure sensitive data is transmitted securely?"

Modification Approach:

- Used the ngrok tool to expose the local server over a secure HTTPS tunnel (port forwarding).
- Ensured all data transmitted between the client and server was encrypted via the ngrok HTTPS endpoint (encryption).

[CWE-400: Uncontrolled Resource Consumption \(DoS\)](#)

Uncontrolled resource consumption (DoS)

JavaScript, Middleware

Detected: Yes **Mitigated:** No

Vulnerable Code:

```
// Middleware setup
app.use(logger('dev')); // Show HTTP requests in the console
app.use(express.json()); // Parse JSON bodies
app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser()); // Parse cookie headers
app.use(express.static(path.join(__dirname, 'public'))); // Serve static files
```

LLM Prompt:

"How can I implement rate limiting in an Express.js server to prevent Denial of Service (DoS) attacks and control resource consumption?"

Potential Modification Approach:

- Integrate rate-limiting middleware like express-rate-limit.
- Restricts number of requests an IP can make within a specified time frame.
- Prevents attackers from overwhelming the server with excessive requests.

File Overview

Public

The **public folder** contains static files that define the structure, styling, and functionality of the web interface.

Static files (HTML, CSS, JS)

- **index.html:** The entry point of the website, integrating the navigation bar and linking other pages.
- **home.html:** Provides an overview of the system with user-friendly navigation.
- **login.html & signup.html:** Contains forms for user authentication (login and registration).

- **devices.html:** Implements functionality for adding/removing IoT devices.
- **heart-summary.html:** Includes the simulation tab to display real-time and historical health data.
- **plot-sensor-data.html:** Renders graphs and visualizations for sensor data collected from IoT devices.
- **user-profile.html & physician-profile.html:** Displays user/physician-specific data and controls.
- **physician-login.html & physician-signup.html:** Provides dedicated login and registration pages for physicians.
- **faq.html:** A static page answering frequently asked questions for users.
- **reference.html:** Template page used for formatting additional components.

Custom stylesheets are implemented to ensure a clean and responsive user interface:

- The styles in `template_index.html` and other HTML files define the layout of buttons, forms, graphs, and navigation.
- CSS rules standardize background colors, margins, and responsive scaling using media queries.

The JavaScript files in the public folder handle interactivity and data fetching:

- **Dynamic Content:** `plot-sensor-data.html` and `devices.html` use JS to interact with the server using API endpoints.
- **API Integration:** Sends HTTP GET/POST requests to display sensor data or manage devices.
- **Hardcoded Credentials:** Some JS files (e.g., `devices.js`) currently contain placeholders for testing IoT commands but require security fixes.

Routes

The **routes** folder implements RESTful APIs using Express. Each route file corresponds to a specific feature.

API endpoints

- **index.js:**
 - Implements the main route for fetching and serving the homepage.
 - Manages basic interactions like redirects and rendering of static pages.
- **devices.js:**
 - Handles API endpoints for device management (e.g., add, remove, update IoT devices).

- o Includes hardcoded placeholders for sending IoT commands, which require credential cleanup.
- **patients.js:**
 - o Implements functionality for patient-specific data, such as retrieving heart rate and oxygen saturation data.
 - o Supports adding/removing devices and visualizing measurement data.
- **physicians.js:**
 - o Provides APIs for physician-specific features, including viewing patient data and managing patients.
- **sensor.js:**
 - o Integrates sensor data processing logic. Handles interaction between backend APIs and the IoT devices.

Models

The models folder uses **Mongoose** to define the database structure for MongoDB.

MongoDB schemas

- **user.js:**
 - o Defines the schema for user accounts, including fields for name, email, hashed passwords, and user roles (patient or physician).
- **device.js:**
 - o Represents IoT device details, including device IDs, patient ownership, and synchronization statuses.
- **sensor.js:**
 - o Tracks heart rate and blood oxygen saturation measurements, storing timestamped data for analysis.

Firmware

The **firmware** scripts written for the Particle Argon device manage sensor communication and server interaction:

IoT scripts

- **main.cpp**
 - o Acts as the entry point of the firmware, managing the overall execution flow.
 - o Implements start and stop functionality to control when the sensor reads and transmits data.

- o Integrates the state machine logic to manage operational states efficiently.
- **MAX30105.cpp & MAX30105.h**
 - o Driver files for the MAX30102 sensor to collect raw heart rate and SpO2 data.
 - o Implements I2C communication to initialize the sensor and retrieve readings.
 - o Support posting the processed data to the server via HTTP requests.
- **heartRate.cpp & heartRate.h**
 - o Implement algorithms to process raw sensor data and calculate heart rate.
 - o Outputs the calculated heart rate in beats per minute (BPM).
- **spo2_algorithm.cpp & spo2_algorithm.h**
 - o Contains the algorithm for calculating SpO2 (blood oxygen saturation) from the sensor readings.
 - o Works in conjunction with the heart rate logic to ensure accurate results.

The firmware scripts handle the critical functionalities of the IoT device:

- **Sensor Initialization:** Configures the MAX30102 sensor for accurate measurements.
- **Data Processing:** Uses algorithms to calculate heart rate and SpO2 from raw readings.
- **Network Communication:** Sends processed data to the server using HTTP POST requests.
- **Error Management:** Logs errors locally and retries failed operations.

Config

The configuration files in the project are essential for managing system settings, sensitive credentials, and server behavior. These files ensure the application runs smoothly across different environments and simplify tasks like environment management, server setup, and dependency tracking.

.env

The .env file stores sensitive information and environment-specific variables, such as API keys, database connection strings, and server configurations. Using this file ensures sensitive data is not hardcoded into the codebase and can be easily modified without changing the source code.

Key Variables in .env:

- **MongoDB Connection String:** Configures the connection to the MongoDB database.
- **API Keys:** Stores secure keys for IoT device communication and server access.
- **Port:** Specifies the port on which the server runs.

app.js

The app.js file serves as the **main configuration file** for the backend server. It initializes the Express application, integrates middleware, defines routes, and manages error handling.

Key Sections in app.js:

- **Middleware Setup:** Integrates middleware for tasks like parsing JSON data, managing cookies, and handling CORS.
- **Routes:** Configures API endpoints by linking route files (e.g., devices.js, patients.js).
- **Error Handling:** Implements custom error-handling middleware to return consistent error responses.
- **Static File Serving:** Serves frontend files from the public directory.

package.json

The package.json file acts as the **metadata and dependency manager** for the project. It defines the libraries and tools required to run the application and provides scripts for server management.

Key Sections in package.json:

- **Dependencies:** Lists of the required libraries (e.g., express, mongoose, jsonwebtoken) and their versions.
- **Scripts:** Provides commands to run, test, and manage the server.

Results

The results section shows the functionalities of the Heart Rate Monitoring project, including screenshots of the website, its various components, and the results captured by the IoT device. This section highlights how users interact with the system, view health trends, and manage devices. Additionally, tables and charts display the real-time and historical physiological data collected from the embedded device.

Website

The web application is hosted on an **AWS EC2 instance**, providing accessibility across devices. The user interface is designed to be intuitive and responsive, enabling seamless navigation and interaction for both patients and physicians.

Navigation

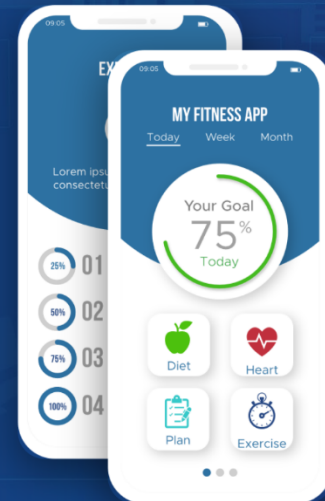
The **navigation bar** allows users to access key features of the application, including device management, data visualization, and user profiles. It includes options for login, logout, and switching between different user roles (e.g., patient or physician).

Saguaro Heart Rate Monitor

Modern heart rate monitor at your fingertips

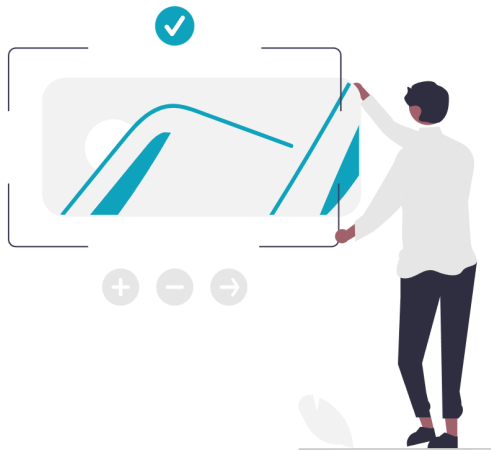
Login

Sign Up



Features

Our system offers a comprehensive suite of features to empower you in taking control of your health effortlessly and efficiently.



Designed for your health, the Saguaro Heart Rate Monitor delivers real-time insights and reliable data to keep you proactive about your well-being.

Whether you're tracking daily health metrics or aiming for long-term wellness goals, our innovative features ensure a seamless and personalized monitoring experience incorporating:

- ✓ Continuously tracks heart rate and blood oxygen levels with high precision, ensuring reliable health insight.
- ✓ Provides instant notifications for irregular readings, helping users respond promptly to potential health concerns.
- ✓ Allows users to set personalized schedules for measurements, tailored to their daily routines.
- ✓ Automatically syncs with the cloud, enabling easy access to data across devices.
- ✓ Features an intuitive interface optimized for both desktop and mobile devices, ensuring seamless user interaction.

How it Works

Discover how the Saguaro Heart Rate Monitor seamlessly integrates into your daily routine, making health tracking simple, intuitive, and effective.

01

Sign Up

Create an account with your email to access your personalized health dashboard.

02

Record

The device prompts you to measure your heart rate and oxygen levels at set intervals. The data is stored and synced with your account.

03

Enjoy

View your health trends via interactive charts. Identify patterns, set wellness goals, and take informed actions.

About the project

Saguaro Heart Rate Monitor

The Saguaro Heart Rate Monitor App enables you to keep track of your heart rate and blood oxygen levels right from home, offering valuable awareness of your health on a daily basis. With frequent, real-time data at your fingertips, you can observe patterns and monitor your progress, empowering you to make proactive adjustments to your wellness routine.

[Learn More →](#)



Mobile

Access your health data anywhere, anytime, with the convenience of mobile monitoring to support on-the-go lifestyles.



Monitoring

Track heart rate and oxygen levels continuously to gain a clear picture of your daily health patterns and changes.



Interconnected

Syncs seamlessly with other devices and health apps, creating a cohesive health profile for more comprehensive wellness insights.



Data

Provides real-time and historical data, helping you recognize trends, set goals, and make informed health decisions over time.

Team Members Page

This page showcases the project team members and provides an overview of their contributions to the system.

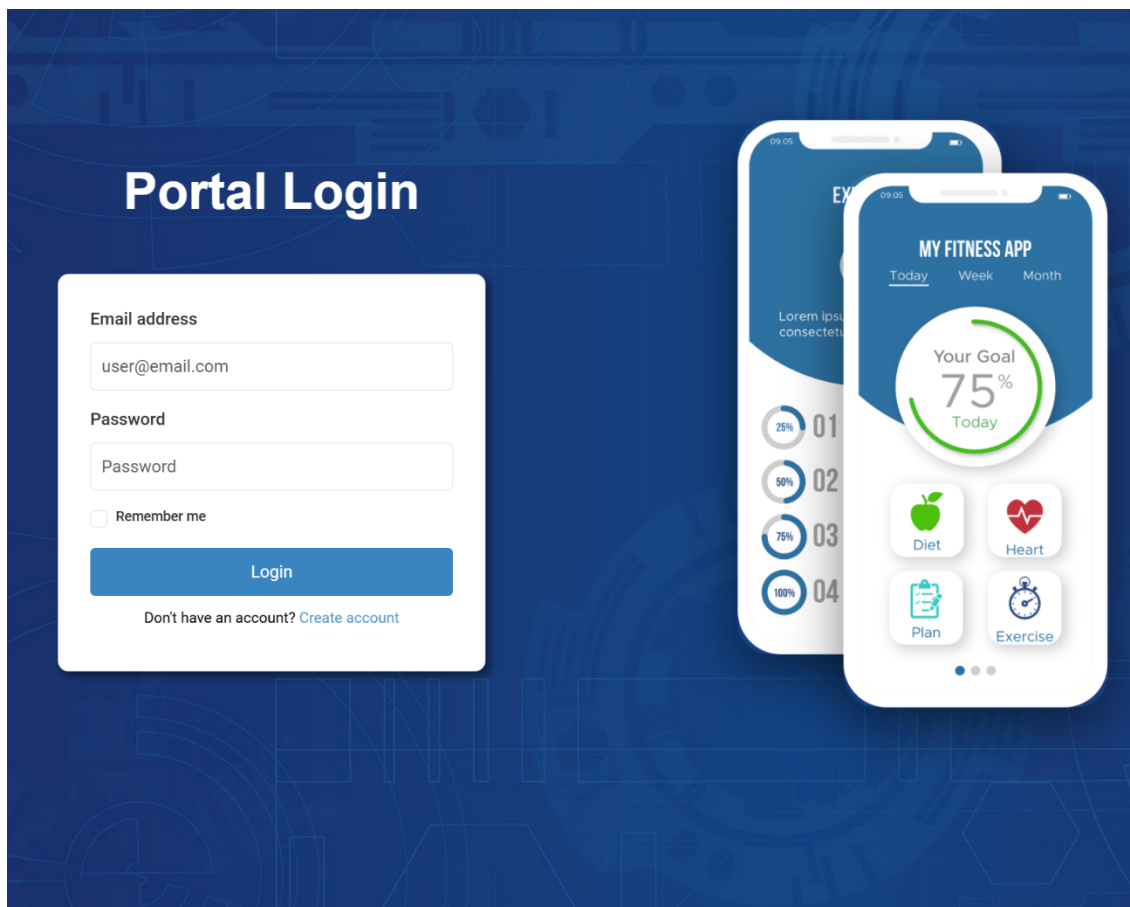
Patient Login and Sign-Up Pages

Patient Login Page (login.html):

- Allow patients to securely log in using their credentials.
- Includes input fields for username/email and password.

Patient Sign-Up Page (signup.html):

- Enables new patients to register by providing their details.
- Input fields include username, email, password, and confirmation password.
- Patients can create an account using their **email address as the username** and a strong password. Input validation ensures the password meets security requirement



Sign Up

First Name

Last Name

Email address

Password

Confirm Password

Sign Up

Already have an account? Login [here](#)

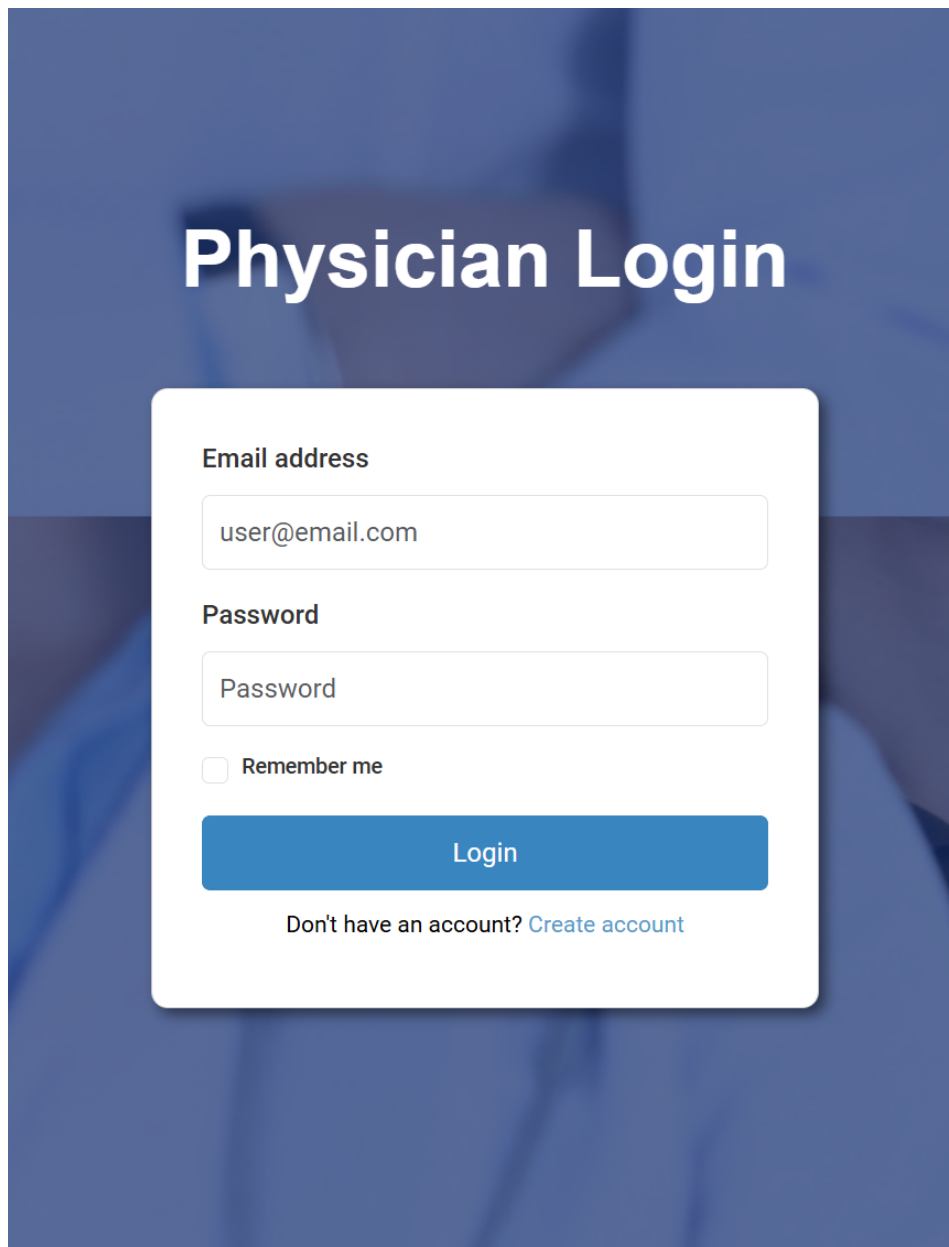
Physician Login and Sign-Up Page

Physician Login Page (physician-login.html):

- Provides a secure login interface for physicians to access patient data.

Physician Sign-Up Page (physician-signup.html):

- Allow physicians to create accounts with additional verification fields, if required.
- Physicians can also create an account using their **email address as the username** and a strong password. Input validation ensures the password meets security requirements.

The image shows a 'Physician Login' form centered on a dark blue background with a blurred image of a person in a white lab coat. The form is a white rounded rectangle with a subtle drop shadow. At the top, the title 'Physician Login' is written in a large, bold, white sans-serif font. Below the title, the form contains two input fields: 'Email address' with the placeholder text 'user@email.com' and 'Password' with the placeholder text 'Password'. Both fields have light gray borders. Below the password field is a checkbox labeled 'Remember me'. At the bottom of the form is a solid blue button with the word 'Login' in white. Below the button, the text 'Don't have an account?' is followed by a blue link that says 'Create account'.

Sign Up

First Name

Last Name

Specialty



Email address

Password

Confirm Password

Sign Up

Already have an account? Login [here](#)

Devices

The **Devices** section enables patients to add, remove, and manage IoT devices. Each registered device is associated with a unique ID, and the server updates the status of the devices (e.g., online/offline).

- **Adding Devices:** Users can input a device ID to register new IoT devices.
- **Removing Devices:** Patients can remove devices from their account using a simple interface.
- **ECE 513:** Physicians can manage patient devices, allowing oversight of multiple users' data.

Charts

The **Charts** section visualizes heart rate and SpO2 data collected by the IoT device. Interactive graphs display the following data trends:

- **Weekly Trends:** Average, minimum, and maximum heart rate and SpO2 levels over the past 7 days.
- **Daily Trends:** Time-series graphs showing real-time heart rate and oxygen saturation readings throughout the day.

These charts are implemented using libraries like **Chart.js**, providing interactive visualizations that allow users to hover and view precise data points.

Portal

The **Portal** section differentiates functionalities for patients and physicians:

- **Patient Portal:**
 - Allow patients to view their device data, add/remove devices, and monitor health trends.
- **Physician Portal:**
 - Provides physicians with access to aggregated data from multiple patients.
 - Enables physicians to manage patient devices and monitor individual health metrics.

IoT Data

The IoT device (Particle Argon with the MAX30102 sensor) collects and transmits heart rate and SpO2 data to the backend server. The data is processed, stored in the MongoDB database, and visualized in the web application.

Tables

Tables display the raw heart rate and oxygen saturation data retrieved from the server. Each entry includes the following fields:

- **Device ID**
- **Heart Rate (BPM)**
- **SpO2 (%)**
- **Timestamp**

Trends

The data collected is analyzed to identify trends in heart rate and blood oxygen levels:

- **Weekly Trends:** Average heart rate and SpO2 calculated daily for the past 7 days.
- **Daily Trends:** Graphs show fluctuations in heart rate and oxygen saturation at different times of the day.

The trends are displayed as line graphs and bar charts for clear visualization.

Key Takeaways

The development of the Heart Rate Monitoring project provided valuable insights into implementing a multidisciplinary system that integrates IoT devices, web development, and backend services. The following key takeaways summarize the lessons learned during the project:

1. Integration of IoT Devices with a Web Application

- o Successfully combining an IoT-based physiological data monitoring system with a user-friendly web interface required careful synchronization between hardware, firmware, backend logic, and the database. Understanding the interaction between

these components was critical for achieving seamless communication and data flow.

2. Secure Communication and Data Management

- o Implementing **JWT-based authentication** and HTTPS protocols ensured secure communication between the IoT device, backend server, and frontend. Proper data encryption and secure API endpoints protected sensitive health data, highlighting the importance of security in healthcare systems.

3. State Machine Design for IoT Firmware

- o Designing an efficient **state machine** on the Particle Argon device allowed the firmware to optimize performance, conserve energy, and handle errors effectively. This approach ensured reliable data collection and transmission without overloading the device.

4. Dynamic Data Visualization

- o Leveraging libraries like **Chart.js** enabled the creation of interactive and responsive visualizations for heart rate and SpO2 trends. This demonstrated the importance of presenting complex data in an accessible and meaningful way for users.

5. Scalability and Modular Design

- o Structuring the project into modular components, such as routes, models, and frontend sections, made the system scalable and easy to maintain. The modular design also allowed for future extensions, such as adding more physiological metrics or user roles.

Challenges

The implementation of the Heart Rate Monitoring project posed several challenges across different components of the system. Below are the key challenges encountered during the development process and the solutions applied to overcome them:

1. Challenge: Synchronizing IoT Device Data with the Server

- o **Issue:** Ensuring reliable communication between the Particle Argon device and the backend server, especially when network connectivity was intermittent.

- o **Solution:** Implemented a buffer mechanism in the firmware to store sensor data locally when offline and retransmit it once the connection was restored. This ensured no data was lost during downtime.
- 2. **Challenge: Accurate Measurement from the MAX30102 Sensor**
 - o **Issue:** The MAX30102 sensor required precise calibration to produce accurate heart rate and SpO2 readings. External factors like noise and improper sensor placement affected data accuracy.
 - o **Solution:** Applied signal processing techniques and fine-tuned the sensor algorithms (spo2_algorithm.cpp) to filter out noise and improve measurement reliability.
- 3. **Challenge: Managing Data Storage and Retrieval in MongoDB**
 - o **Issue:** With multiple devices transmitting data simultaneously, ensuring efficient storage and querying of data in MongoDB was challenging.
 - o **Solution:** Used **schema indexing** and optimized queries in Mongoose to retrieve specific records quickly. Aggregation pipelines were implemented to calculate weekly trends and historical data summaries.
- 4. **Challenge: Ensuring Secure API Communication**
 - o **Issue:** Exposing API endpoints for data retrieval and management posed risks of unauthorized access or tampering.
 - o **Solution:** Added **JWT-based authentication** and middleware to validate API requests, ensuring only authorized users and devices could interact with the server. HTTPS was used for secure data transmission.
- 5. **Challenge: Responsive Design Across Devices**
 - o **Issue:** Creating a frontend that worked seamlessly across desktops, tablets, and mobile devices required consistent testing and adjustments.
 - o **Solution:** Applied **responsive design principles** using CSS media queries and Bootstrap components to ensure proper scaling and layout adjustments for various screen sizes.

Team Contribution

The contributions for each of the major sections of the project for each team member is shown in the table below:

Team Member	Frontend (%)	Backend (%)	Firmware (%)	Documentation (%)
Aaron Colin	60	40	20	20
Ashish Khadka	30	40	60	20
Taron Bashar	10	20	20	60

References

1. Chart.js. (n.d.). *Simple yet flexible JavaScript charting library*. Retrieved from <https://www.chartjs.org/>
2. Express. (n.d.). *Fast, unopinionated, minimalist web framework for Node.js*. Retrieved from <https://expressjs.com/>
3. MongoDB. (n.d.). *NoSQL Database for modern applications*. Retrieved from <https://www.mongodb.com/>
4. Mongoose. (n.d.). *Elegant MongoDB object modeling for Node.js*. Retrieved from <https://mongoosejs.com/>
5. Node.js. (n.d.). *JavaScript runtime built on Chrome's V8 JavaScript engine*. Retrieved from <https://nodejs.org/en/>
6. Particle. (n.d.). *Particle CLI documentation*. Retrieved from <https://docs.particle.io/>
7. Bootstrap. (n.d.). *Build responsive, mobile-first projects on the web with the world's most popular front-end component library*. Retrieved from <https://getbootstrap.com/>
8. MAX30102 Datasheet. (n.d.). *Pulse Oximeter and Heart Rate Sensor*. Retrieved from <https://datasheetspdf.com/pdf-file/1398007/MAXIM/MAX30102/1>
9. OpenAI. (2024). *Large Language Model for Code Assistance*. Retrieved from <https://openai.com/chatgpt>
10. CWE-352: Cross-Site Request Forgery (CSRF). (2024). *Common Weakness Enumeration*. Retrieved from <https://cwe.mitre.org/data/definitions/352.html>
11. CWE-311: Missing Encryption of Sensitive Data. (2024). *Common Weakness Enumeration*. Retrieved from <https://cwe.mitre.org/data/definitions/311.html>
12. CWE-400: Uncontrolled Resource Consumption. (2024). *Common Weakness Enumeration*. Retrieved from <https://cwe.mitre.org/data/definitions/400.html>
13. Ngrok. (n.d.). *Secure tunnels to localhost*. Retrieved from <https://ngrok.com/>
14. JWT.io. (n.d.). *Introduction to JSON Web Tokens*. Retrieved from <https://jwt.io/introduction>
15. AWS EC2. (n.d.). *Elastic Compute Cloud (EC2)*. Retrieved from <https://aws.amazon.com/ec2/>
16. MDN Web Docs. (n.d.). *Responsive Web Design Basics*. Retrieved from https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Responsive_Design
17. Postman. (n.d.). *API Testing and Development Tool*. Retrieved from <https://www.postman.com/>

18. Particle CLI. (n.d.). *Compile and flash IoT firmware*. Retrieved from <https://docs.particle.io/reference/device-os/cli/>
19. Spo2 Algorithm. (n.d.). *Algorithm implementation for SpO2 and heart rate calculation*. Retrieved from project code files: spo2_algorithm.cpp & spo2_algorithm.h
20. MAX30102 Integration. (n.d.). *Sensor initialization and measurement*. Retrieved from project code files: MAX30105.cpp & MAX30105.h
21. Express Route Implementation. (n.d.). *API Endpoint Design*. Retrieved from project files: devices.js, patients.js, and physicians.js
22. Signal Processing for Noise Reduction. (n.d.). *Implementation in firmware for MAX30102 sensor data*. Retrieved from project files: heartRate.cpp & spo2_algorithm.cpp