# Project Description

****Note - The AWS Hosted Website has limited features as we had difficulties getting the MongoDB to work on the cloud, so our database with MongoDB is not working and HTTPS is not on the AWS server, however all of it is working locally as shown in the demo video****

The Heart Track project is an innovative, low-cost, IoT-enabled system for monitoring heart rate and blood oxygen levels. Designed to help users track their health metrics in real time, it also enables seamless data sharing with healthcare professionals. The system integrates an IoT device, a robust backend, and a responsive web application to deliver a comprehensive and user-friendly solution.

The backend system is built using Node.js, Express, and MongoDB, providing scalability, reliability, and security. A RESTful API manages user authentication, device registration, data retrieval, and storage, with all endpoints documented for clarity. Token-based authentication using JSON Web Tokens (JWT) secures access to the web application, and IoT devices require an API key for data submissions, ensuring data integrity and preventing unauthorized access. MongoDB stores user profiles, device information, and health metrics, with schemas optimized for time-series data. The backend logic also allows users to configure measurement alerts, including frequency and time range.

The web application is designed to offer a seamless and intuitive user experience across all devices. It features a responsive design built with modern HTML, CSS, and JavaScript frameworks, ensuring compatibility with desktops, tablets, and mobile devices. Users can create accounts, manage devices, and configure measurement settings through a user-friendly interface. Data visualization is implemented using the third-party charting library Chart.js, presenting weekly summaries and detailed daily views of heart rate and blood oxygen levels. For ECE 513, a dedicated physician portal allows healthcare professionals to view and manage patient data, providing both summary and detailed views of health metrics.

The IoT component is built on the Particle Argon development board and paired with a MAX30102 sensor module for heart rate and blood oxygen measurements. The firmware uses a synchronous state machine approach to manage tasks such as requesting measurements, storing data, and transmitting data to the backend. The MAX30102 sensor captures accurate readings using built-in stabilization algorithms. Data is transmitted over Wi-Fi when available; if offline, data is stored locally for up to 24 hours and synced once connectivity is restored. The onboard RGB LED provides visual alerts, flashing blue to prompt measurements, green for successful data uploads, and yellow to indicate offline storage. Users can configure measurement frequency and time range via the web application, with default settings of every 30 minutes between 6:00 AM and 10:00 PM.

Integration and testing were key to ensuring a robust and reliable system. The IoT devices communicate with the backend using secure RESTful APIs, validated using tools like Postman. AJAX and Fetch API facilitated smooth communication between the frontend and backend, and end-to-end testing ensured consistent performance across all features.

Heart Track is a comprehensive system that seamlessly integrates IoT, backend, and frontend technologies to deliver an effective and user-friendly health monitoring solution. By leveraging affordable hardware and scalable software, the project addresses critical gaps in the market for real-time cardiovascular health monitoring.

## File Description

Our files are organized into 3 sections: Our frontend, backend, and embedded device code.

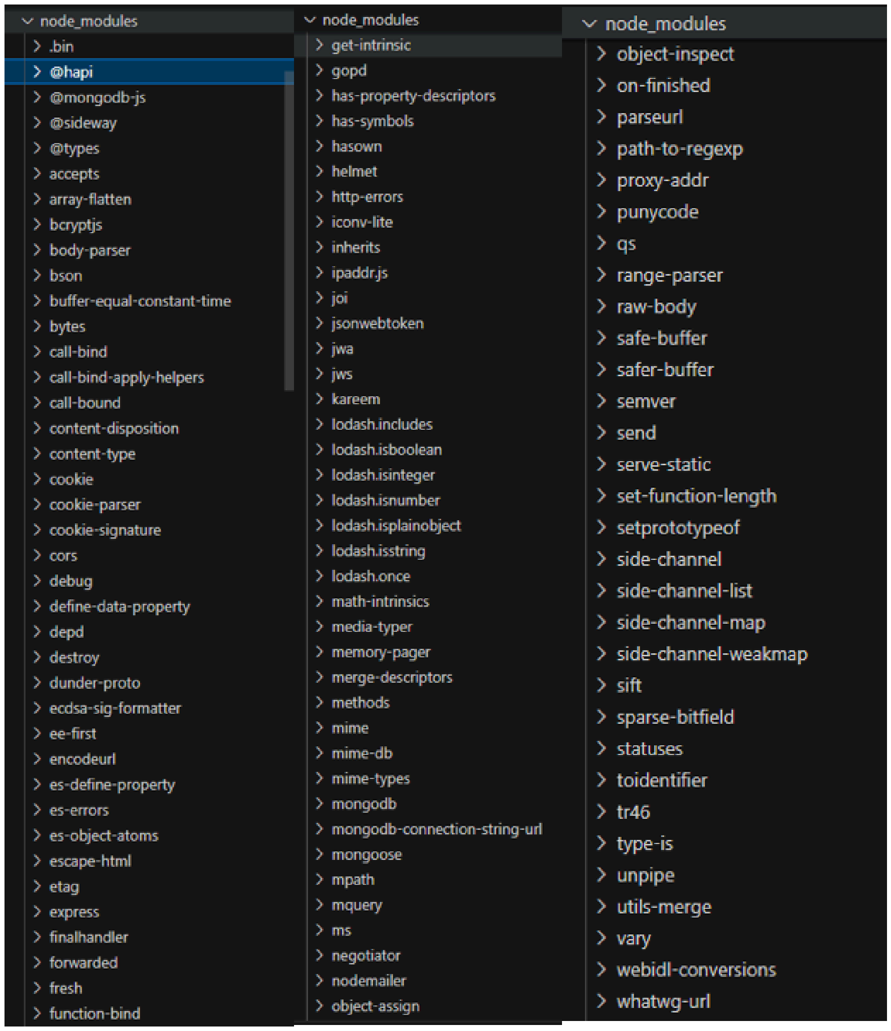
Starting with our **frontend**, we have:

- Assets/icons folder: Contains our team photos for our index page and our profile icon folder that patients can click on the top right of the website to access their account.
    - **1734383939156.jpg, Anand image.jpg, malcolm.jpg, profile icon.jpg**
- ArgonKit.png - Image for the Particle Argon IoT device
- README.md - Markdown file for our description of the project and how to implement on a local machine
- Account-details.css - CSS rules for the account details page for styling
- Account-details.html - HTML for the webpage for the account details, patients can see their name, edit their password and email. A list of physicians can be selected from a dropdown to allow the patients to connect their data to their provider.
- Account-details.js - Javascript for loading the physician selection and saving the patient details
- Account.html - Login HTML page where patients can login or click a link to be redirected to a page to create an account
- Account.js- Javascript for loading the login and signup functionality. Contains logic to send POST requests to the backend and to verify password strength.
- Add-device.css - CSS rules and responsive design elements for the device management page (for patients)
- Add-device.html, Add–device.js - HTML and javascript for the device management page: patients can add a new device, where they input a device name, device ID, and description. The devices added to the patient's account are listed.
- anand.JPG, dalian.JPG - photos for index.html page for team member descriptions
- Dashboard.html, dashboard.js - HTML and JS for the dashboard of the website, where patients can view their weekly summary, and detailed daily view for their data.
- Index.html - Home page of the website where team members and the project description are listed
- Patient-summary.html, patient-summary.css, patient-summary.js - HTML, CSS, and JS for our patient summary page. This allows the patient to see their detailed daily view as

well as adjust measurement frequency, so they can change how many measurements they do per day.
- Physician_portal.html - HTML page that allows physicians to see all of their patients' data from the device.
- Physician-registration.html - HTML page that allows physicians to create a heart track account, where they enter their name, email, years of experience, and password.
- Reference.html - HTML page that shows all of the references used in the project, such as 3rd party embedded system firmware git repos.
- Styles.css - Our master CSS rule set document that applies to most of our website, creates our responsive design that has a white and blue theme.

Next, we have our **backend** code:

- **controllers/authController.js**: - Handles authentication logic, such as user login, registration, and token issuance. It interacts with services like password hashing and database models to validate user credentials.
- **middlewares/identification.js**: - Middleware for identifying and verifying user requests, such as decoding and validating tokens to ensure authorized access.
- **middlewares/sendMail.js**: - Provides functionality to send emails, such as verification emails or password reset links, using an email service.
- **middlewares/validator.js**: - Middleware for validating input data (e.g., email format, password strength) to ensure that only valid requests are processed.
- **models/userModel.js**: - Defines the MongoDB schema for user data, including fields like username, email, password, and any other user-related information.
- **routers/authRouter.js**: -Sets up routes for user authentication, such as login, registration, logout, and token refresh. Connects routes to controller methods.
- **utils/hashing.js**: -Provides utilities for hashing and verifying passwords using libraries like bcrypt to ensure secure password management.
- **index.js**: - The entry point for the Node.js backend application. It sets up the server, connects to the database, and configures middlewares and routers.
- **Package.json**: - Contains metadata about the project, including dependencies, scripts, and configuration for the Node.js backend.
- **Package-lock.json**: - Automatically generated file that locks the exact versions of installed dependencies to ensure consistency across environments.
- **.gitignore**: - Specifies files and directories (e.g., `node_modules`, environment files) to be excluded from version control.
- **Node_modules folder** - contains all of the external modules used by our Node.js implementation. A screenshot of all of the modules used is below:

○ 

```
⌄ node_modules
  > .bin
  > @hapi
  > @mongodb-js
  > @sideway
  > @types
  > accepts
  > array-flatten
  > bcryptjs
  > body-parser
  > bson
  > buffer-equal-constant-time
  > bytes
  > call-bind
  > call-bind-apply-helpers
  > call-bound
  > content-disposition
  > content-type
  > cookie
  > cookie-parser
  > cookie-signature
  > cors
  > debug
  > define-data-property
  > depd
  > destroy
  > dunder-proto
  > ecdsa-sig-formatter
  > ee-first
  > encodeurl
  > es-define-property
  > es-errors
  > es-object-atoms
  > escape-html
  > etag
  > express
  > finalhandler
  > forwarded
  > fresh
  > function-bind
```

```
⌄ node_modules
  > get-intrinsic
  > gopd
  > has-property-descriptors
  > has-symbols
  > hasown
  > helmet
  > http-errors
  > iconv-lite
  > inherits
  > ipaddr.js
  > joi
  > jsonwebtoken
  > jwa
  > jws
  > kareem
  > lodash.includes
  > lodash.isboolean
  > lodash.isinteger
  > lodash.isnumber
  > lodash.isplainobject
  > lodash.isstring
  > lodash.once
  > math-intrinsics
  > media-typer
  > memory-pager
  > merge-descriptors
  > methods
  > mime
  > mime-db
  > mime-types
  > mongodb
  > mongodb-connection-string-url
  > mongoose
  > mpath
  > mquery
  > ms
  > negotiator
  > nodemailer
  > object-assign
```

```
⌄ node_modules
  > object-inspect
  > on-finished
  > parseurl
  > path-to-regexp
  > proxy-addr
  > punycode
  > qs
  > range-parser
  > raw-body
  > safe-buffer
  > safer-buffer
  > semver
  > send
  > serve-static
  > set-function-length
  > setprototypeof
  > side-channel
  > side-channel-list
  > side-channel-map
  > side-channel-weakmap
  > sift
  > sparse-bitfield
  > statuses
  > toidentifier
  > tr46
  > type-is
  > unpipe
  > utils-merge
  > vary
  > webidl-conversions
  > whatwg-url
```

Lastly, we have our **IoT / Embedded System** code:

- IoT Code/ECE413.cpp - This is our main embedded system code programmed by us. This code implements the required functionalities for the project such as measuring heart rate and blood oxygen, flashing LEDs depending on what state the device is in (no WiFi, time to measure, etc.). We start with a setup function that sets our values to their initial state. We have functions programmed in the code that takes a measurement, checks the time to decide if it is time to take a measurement, change LED color, sending data to the cloud, and setting delay.
- IoT Code/ECE413.ino - This is very similar to the ECE413.cpp file, however it is in a slightly different format that can be read by the embedded device.
- IoT Code/heartRate.cpp - Firmware for the device that implements the heart
- IoT Code/heartRate.h - Header file for heartRate.cpp - defines functions used
- IoT Code/MAX30105.cpp - Library used from SparkFun for the MAX30105 that is also compatible for our MAX30102 sensor, written by Peter Jansen and Nathan Seidle. This

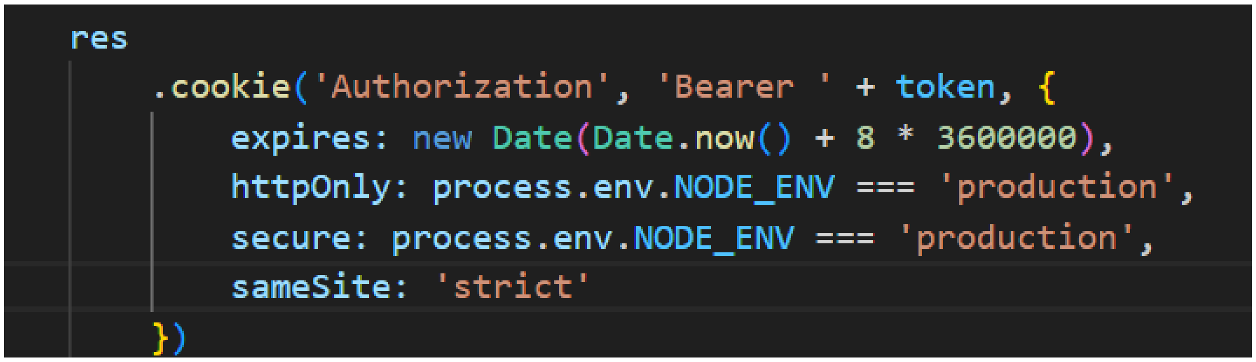
firmware defines registers on the device to be easily referenced and defines functions to manipulate the registers on the Particle Argon.

- IoT Code/MAX30105.h - Header file for the SparkFun library, where functions and constants are defined
- IoT Code/spo2_algorithm.cpp - Firmware for using the MAX30102 along with the SparkFun Library. This code calculates the heart rate and blood oxygen levels from the MAX30102 sensors by implementing the spo2 algorithm.
- IoT Code/spo2_algorithm.h -Header file for the spo2_algorithm.cpp file, defining functions and constants.

## Use of LLM for CWE

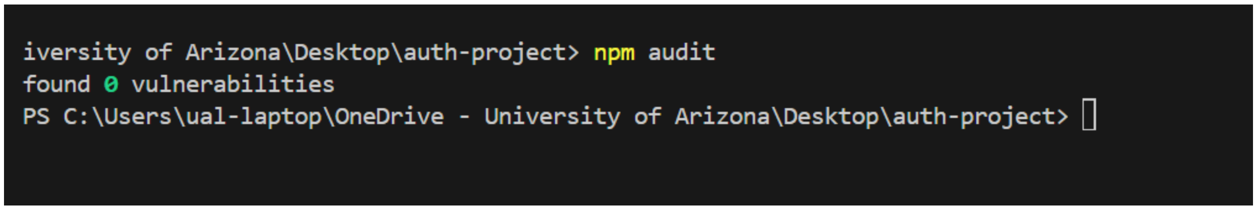
| **CWE-ID (Web Link)** | **Description** | **Domain (HTML/CSS/JS/Firmware)** | **Detected/Mitigated** |
|---|---|---|---|
| **CWE-1104** | Use of Unmaintained Third-Party Components. | JS | **Mitigated** (Used npm audit to check the vulnerabilities ) |
| **CWE-319**: | Cleartext Transmission of Sensitive Data – If the device transmits data without encryption. | IOT | **Mitigated** As HTTPS Protocol is implemented in the Backend |
| **CWE-523**: | Unencrypted Communication – Failing to enforce HTTPS. | API(JS) | **Mitigated** Enforce HTTPS for all communication (using TLS 1.2 or higher). |

|  |  |  |  |
|---|---|---|---|
| **CWE-352:** | CSRF allows an attacker to perform unauthorized actions on behalf of an authenticated user. | Server | **Mitigated**<br><br>Require SameSite cookies to block cross-origin requests:<br><br>javascript<br><br>Copy code |



```
res
    .cookie('Authorization', 'Bearer ' + token, {
        expires: new Date(Date.now() + 8 * 3600000),
        httpOnly: process.env.NODE_ENV === 'production',
        secure: process.env.NODE_ENV === 'production',
        sameSite: 'strict'
    })
```

Here is an attached screenshot of us protecting against CWE-352.

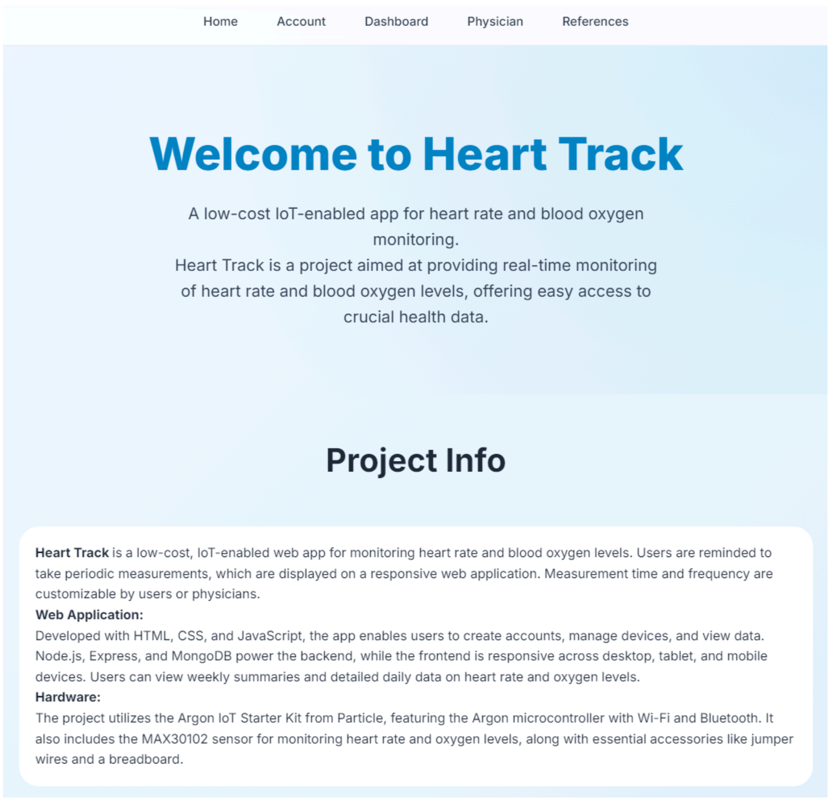


```
iversity of Arizona\Desktop\auth-project> npm audit
found 0 vulnerabilities
PS C:\Users\ual-laptop\OneDrive - University of Arizona\Desktop\auth-project> 
```

Here, we can see how CWE-1104 was mitigated.

# Results

In this section, we will show a set of various screenshots showing the functionality of our website.



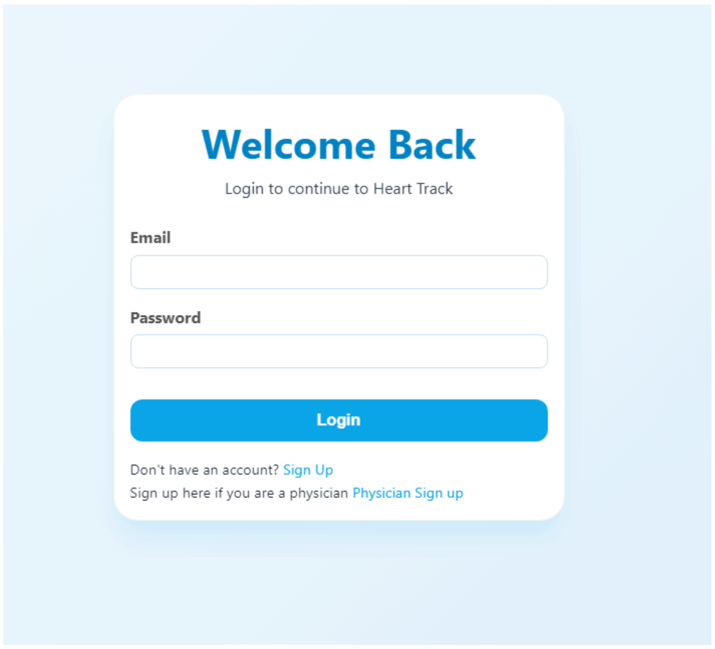
Welcome to Heart Track

A low-cost IoT-enabled app for heart rate and blood oxygen monitoring.
Heart Track is a project aimed at providing real-time monitoring of heart rate and blood oxygen levels, offering easy access to crucial health data.

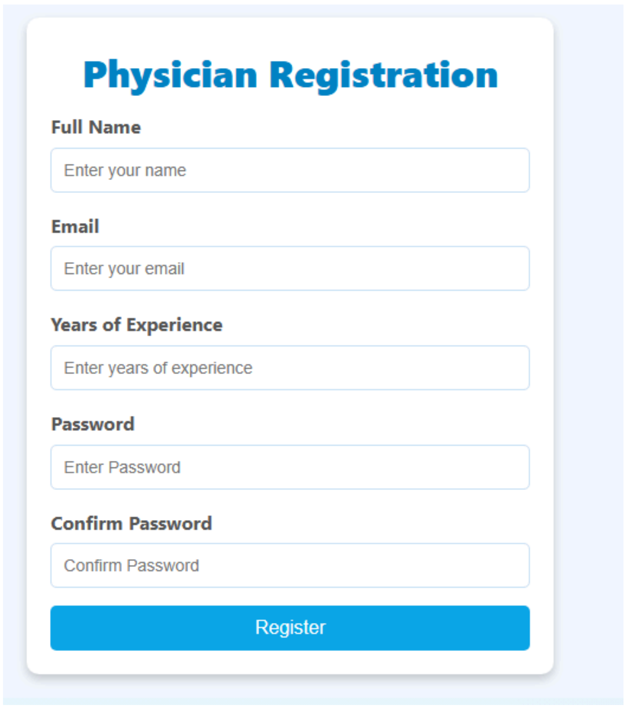
Project Info

**Heart Track** is a low-cost, IoT-enabled web app for monitoring heart rate and blood oxygen levels. Users are reminded to take periodic measurements, which are displayed on a responsive web application. Measurement time and frequency are customizable by users or physicians.
**Web Application:**
Developed with HTML, CSS, and JavaScript, the app enables users to create accounts, manage devices, and view data. Node.js, Express, and MongoDB power the backend, while the frontend is responsive across desktop, tablet, and mobile devices. Users can view weekly summaries and detailed daily data on heart rate and oxygen levels.
**Hardware:**
The project utilizes the Argon IoT Starter Kit from Particle, featuring the Argon microcontroller with Wi-Fi and Bluetooth. It also includes the MAX30102 sensor for monitoring heart rate and oxygen levels, along with essential accessories like jumper wires and a breadboard.

Here we can see a screenshot of our index.html of our website, which implements our responsive design, and has menu buttons at the top of the page that allow the patients and physicians to easily navigate through the website.


Welcome Back
Login to continue to Heart Track
Email
Password
Login
Don't have an account? Sign Up
Sign up here if you are a physician Physician Sign up

Here we see our patient login page, where they can enter their email and password to login to access their account. They can hit the sign up button to switch to the sign up page, as well as switch to the physician sign up page. The physicians using the website will be in the minority of users, thus the login page does not default to bringing the user to a physician login portal.


Physician Registration

Full Name

Enter your name

Email

Enter your email

Years of Experience

Enter years of experience

Password

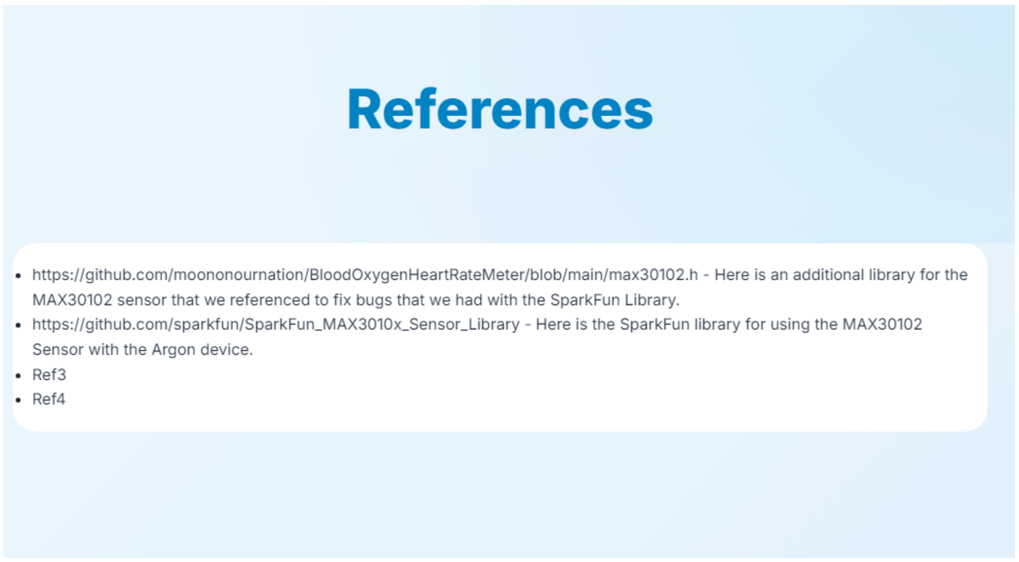
Enter Password

Confirm Password

Confirm Password

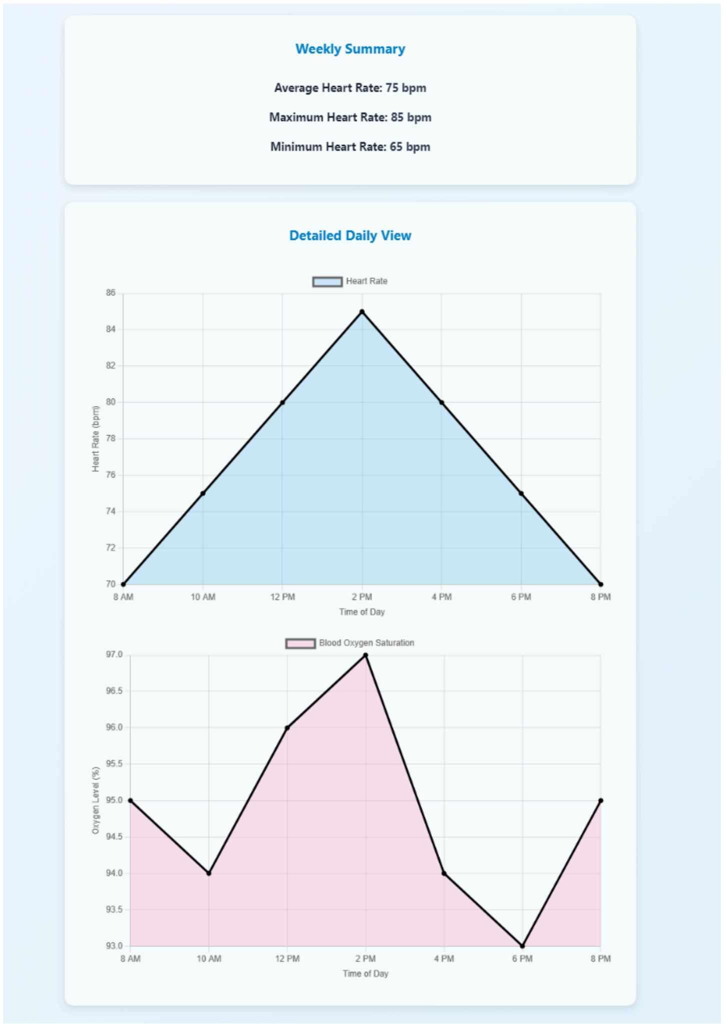
Register

Here you are able to see the physician registration portal, where they enter all of the required information to access the physician portal.


References

- https://github.com/moononournation/BloodOxygenHeartRateMeter/blob/main/max30102.h - Here is an additional library for the MAX30102 sensor that we referenced to fix bugs that we had with the SparkFun Library.
- https://github.com/sparkfun/SparkFun_MAX3010x_Sensor_Library - Here is the SparkFun library for using the MAX30102 Sensor with the Argon device.
- Ref3
- Ref4

Here is the reference.html page, where we list our 3rd party libraries from GitHub that we used for our IoT device firmware.


**Weekly Summary**

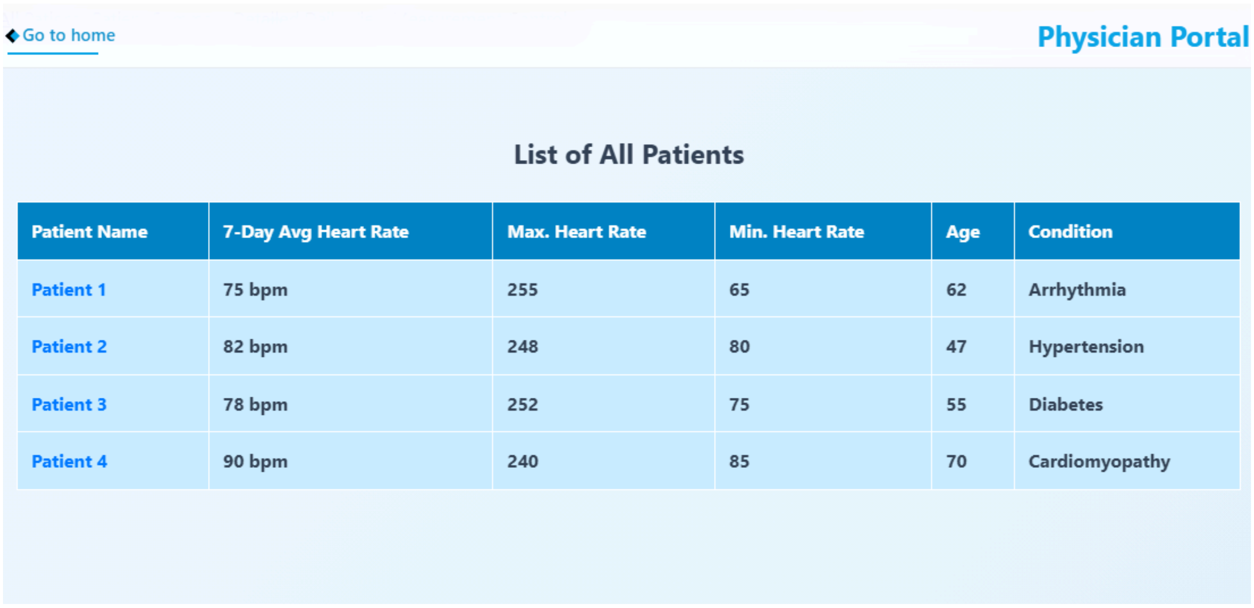
**Average Heart Rate: 75 bpm**

**Maximum Heart Rate: 85 bpm**

**Minimum Heart Rate: 65 bpm**

**Detailed Daily View**

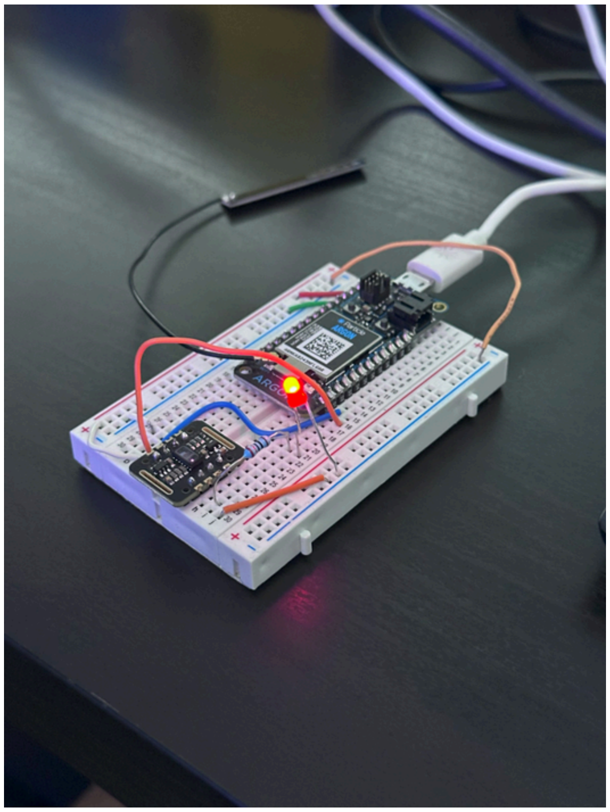
Here is the page that a patient will see when they successfully login to the dashboard, where they are able to see their weekly summary and detailed daily view.



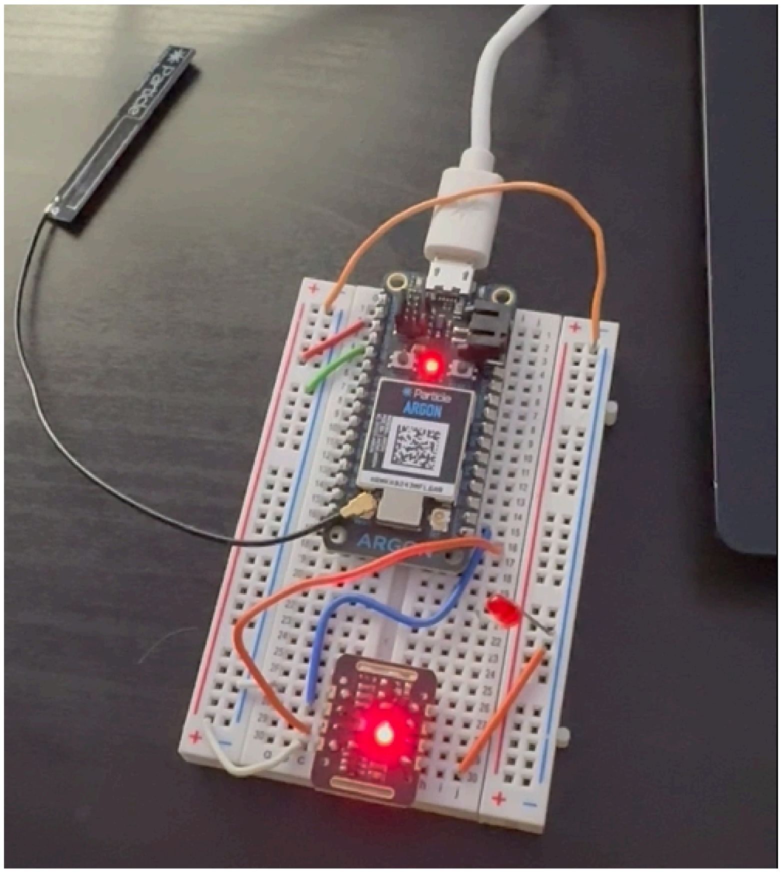
List of All Patients

| Patient Name | 7-Day Avg Heart Rate | Max. Heart Rate | Min. Heart Rate | Age | Condition |
|---|---|---|---|---|---|
| Patient 1 | 75 bpm | 255 | 65 | 62 | Arrhythmia |
| Patient 2 | 82 bpm | 248 | 80 | 47 | Hypertension |
| Patient 3 | 78 bpm | 252 | 75 | 55 | Diabetes |
| Patient 4 | 90 bpm | 240 | 85 | 70 | Cardiomyopathy |

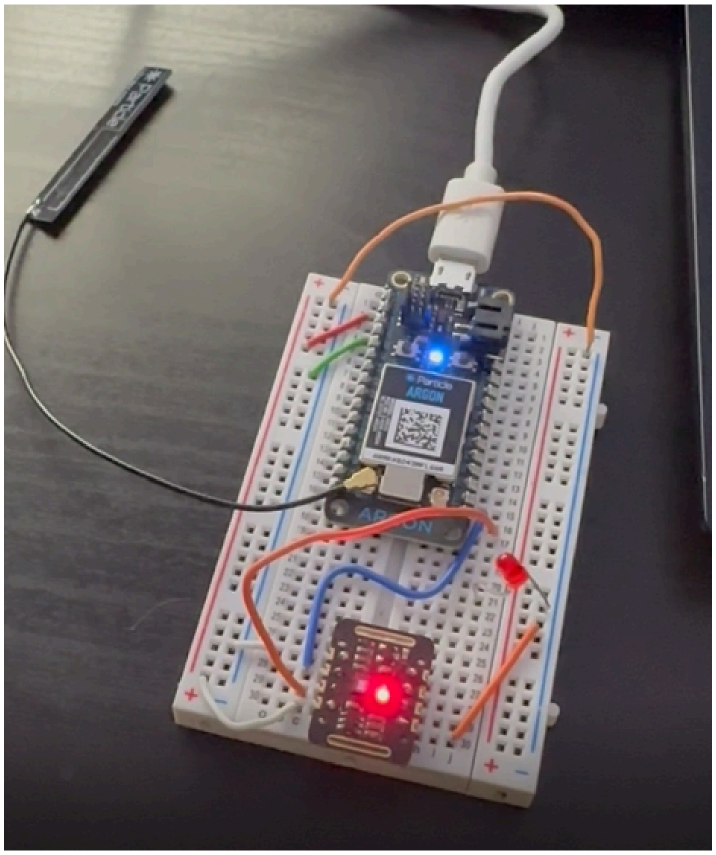
Once the physician logins in through the physician login/signup page, they are able to see the list of the patients in the system, where they can see data inside of the table about each patient.



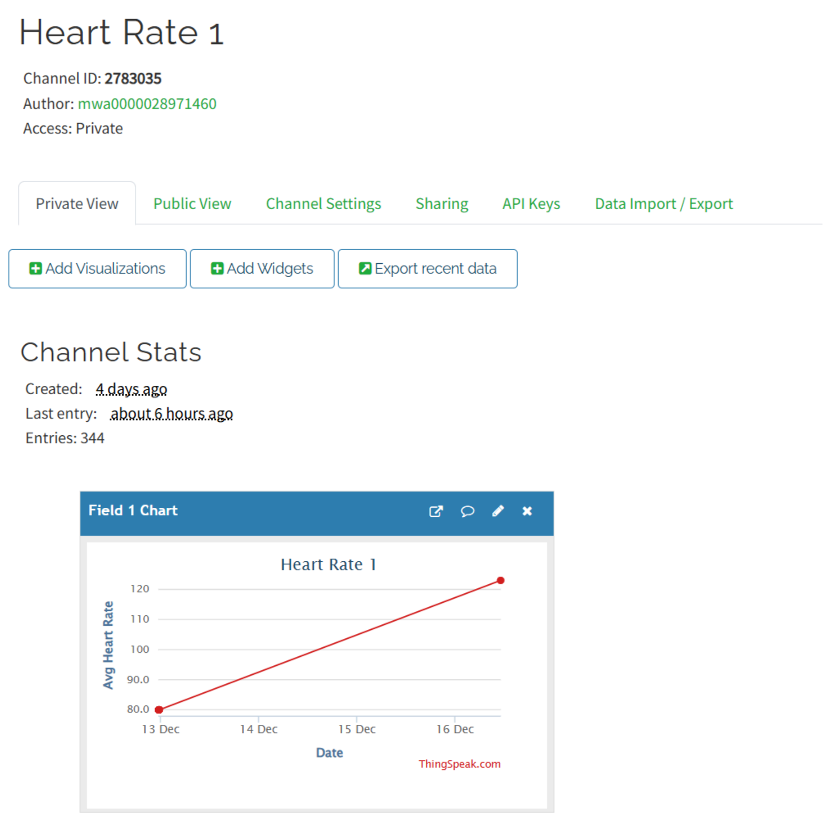
Here, we can see the setup of our Particle Argon IoT device on the breadboard hooked up with the MAX30102 sensor. We can see our Red LED light up, showing that our wiring is setup correctly.



Here, we can see the red LED functionality properly showing that our device is not currently not connected to the internet.



Here, we see the Argon board is properly lighting up blue, which is properly happening after a 30 minute interval of waiting. The blue light signifies that the patient needs to place their finger on the MAX30102 sensor to get a measurement.



Heart Rate 1

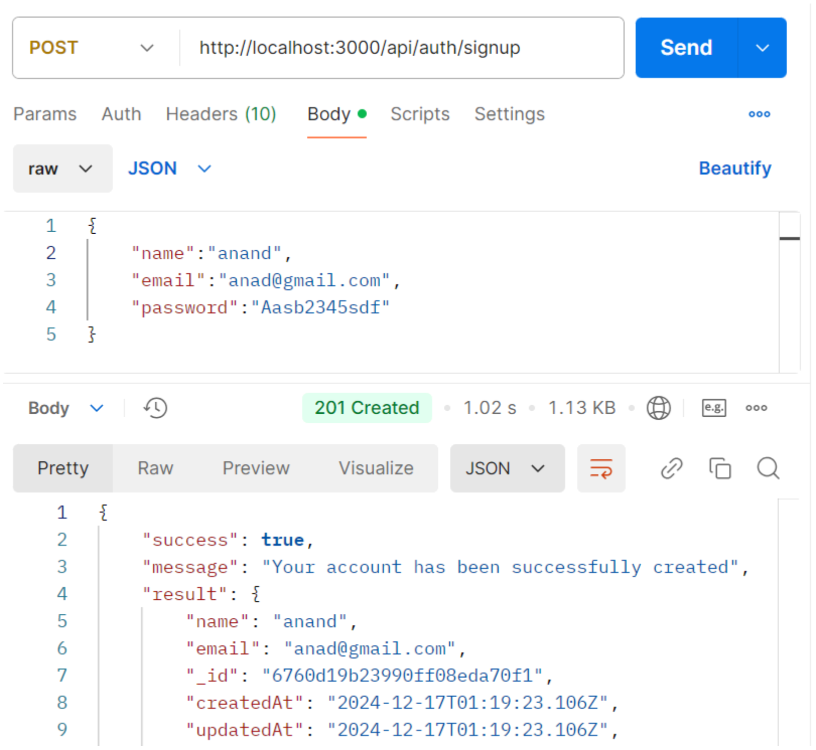
Channel ID: **2783035**
Author: mwa0000028971460
Access: Private

Channel Stats

Created: 4 days ago
Last entry: about 6 hours ago
Entries: 344

The following screenshot showcases the visualization of heart rate data collected from a patient using MATLAB ThingSpeak. The IoT device, equipped with the Particle Argon microcontroller and a MAX30102 sensor, measures the heart rate data and transmits it to the backend server. From the server, the data is forwarded to the ThingSpeak cloud platform for real-time storage and visualization. The graph displayed in the screenshot provides a clear representation of individual heart rate measurements collected at regular intervals, allowing for an intuitive understanding of the patient's cardiovascular trends over time.

This visualization highlights the effectiveness of the system's data flow: the IoT device captures the heart rate readings, securely transmits the data to the backend, and integrates with the ThingSpeak API for analysis and monitoring. The graph serves as a critical feature, enabling users and healthcare professionals to observe patterns and anomalies in heart rate behavior over a specified period. By leveraging ThingSpeak's capabilities, the project successfully delivers real-time insights into health metrics, bridging the gap between raw sensor data and meaningful visualization.



http://localhost:3000/api/auth/signup

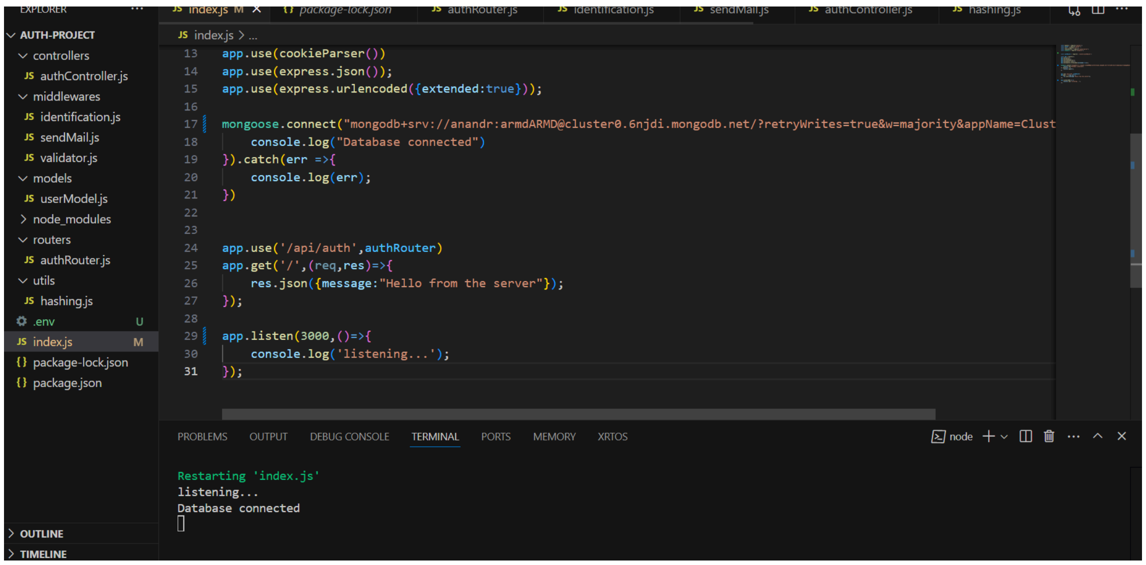
```
{
    "name":"anand",
    "email":"anad@gmail.com",
    "password":"Aasb2345sdf"
}
```

```
{
    "success": true,
    "message": "Your account has been successfully created",
    "result": {
        "name": "anand",
        "email": "anad@gmail.com",
        "_id": "6760d19b23990ff08eda70f1",
        "createdAt": "2024-12-17T01:19:23.106Z",
        "updatedAt": "2024-12-17T01:19:23.106Z",
```

The screenshot demonstrates the successful implementation of the user signup functionality using the **POST** method via Postman. The endpoint `http://localhost:3000/api/auth/signup` is used to send a JSON payload containing user details, including the name, email, and password. In this example, the request body contains the following user information: `"name": "anand"`, `"email": "anad@gmail.com"`, and `"password": "Aasb2345sdf"`. The server processes this request, validates the input, and creates a new user account in the MongoDB database.

The server responds with a **201 Created** status code, indicating that the account has been successfully created. The response includes a success message and the details of the newly created user account, such as the name, email, and the unique `_id` generated by MongoDB. Additionally, the `createdAt` and `updatedAt` fields are automatically generated to record the timestamps for when the account was created and updated. The response structure confirms that the backend successfully handles account creation while returning the appropriate information to the client.

This result validates the integration of the **backend API**, **MongoDB database**, and **Express server**. It demonstrates that the system is capable of securely receiving user input, storing it in the database, and returning a clear and organized response. Overall, the implementation successfully showcases the key functionality of account creation, ensuring proper communication between the client, server, and database.

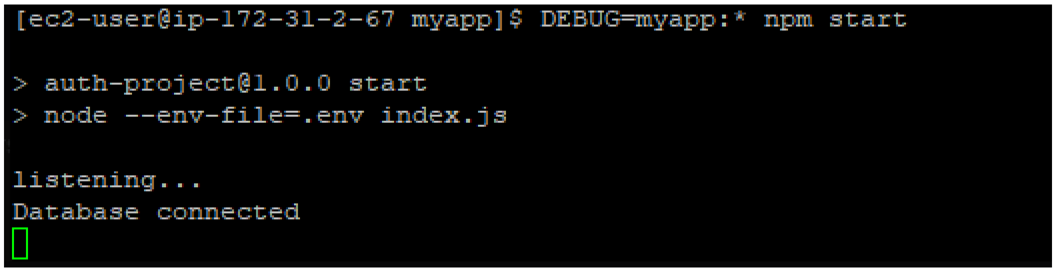


```javascript
app.use(cookieParser())
app.use(express.json());
app.use(express.urlencoded({extended:true}));

mongoose.connect("mongodb+srv://anandr:armdARMD@cluster0.6njdi.mongodb.net/?retryWrites=true&w=majority&appName=Clust
    console.log("Database connected")
}).catch(err =>{
    console.log(err);
})


app.use('/api/auth',authRouter)
app.get('/',(req,res)=>{
    res.json({message:"Hello from the server"});
});

app.listen(3000,()=>{
    console.log('listening...');
});
```

```
Restarting 'index.js'
listening...
Database connected
```

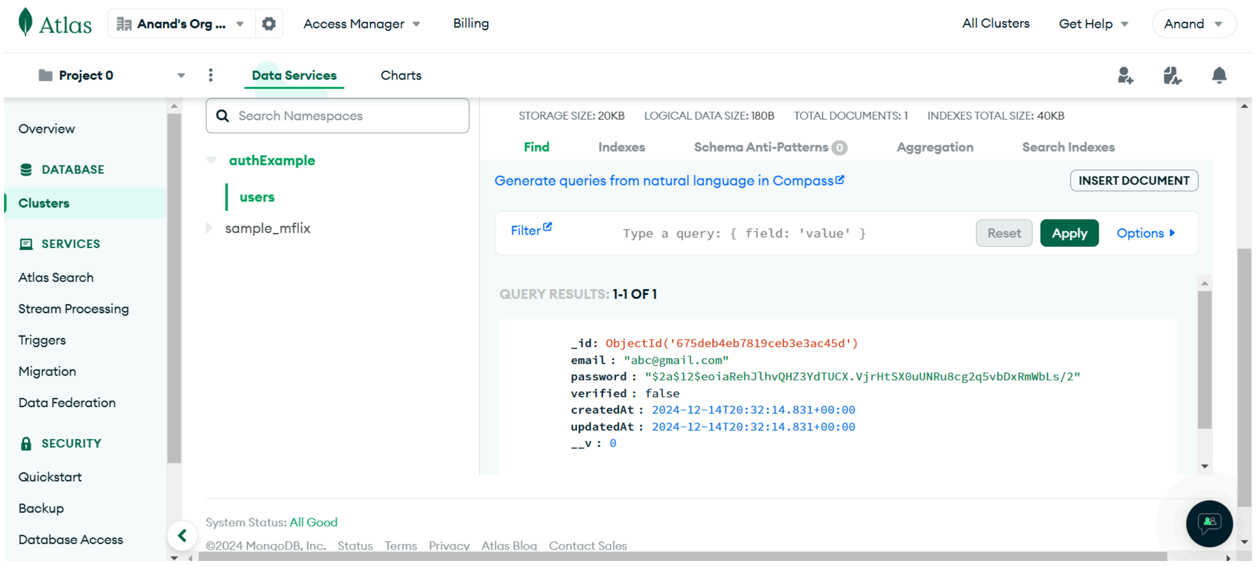
The subsequent screenshot highlights the use of the **Node.js terminal** within **VSCode** to launch the localhost server. The output in the terminal confirms that the backend server has successfully started and is listening on **port 3000**. Additionally, the terminal output verifies that the connection to the **MongoDB database** has been established without errors. This output signifies the successful configuration and integration of the Node.js application with the database, ensuring smooth communication between the backend server and the data storage system.

The terminal feedback serves as an important checkpoint during development, as it confirms that both critical components — the server and the database — are operational. This step ensures that the backend environment is ready to handle incoming API requests, process user data, and interact with the MongoDB database for data storage and retrieval. The successful server launch and database connection reflect the robustness of the backend infrastructure and provide a solid foundation for the system's functionality.



```
[ec2-user@ip-172-31-2-67 myapp]$ DEBUG=myapp:* npm start

> auth-project@1.0.0 start
> node --env-file=.env index.js

listening...
Database connected
```

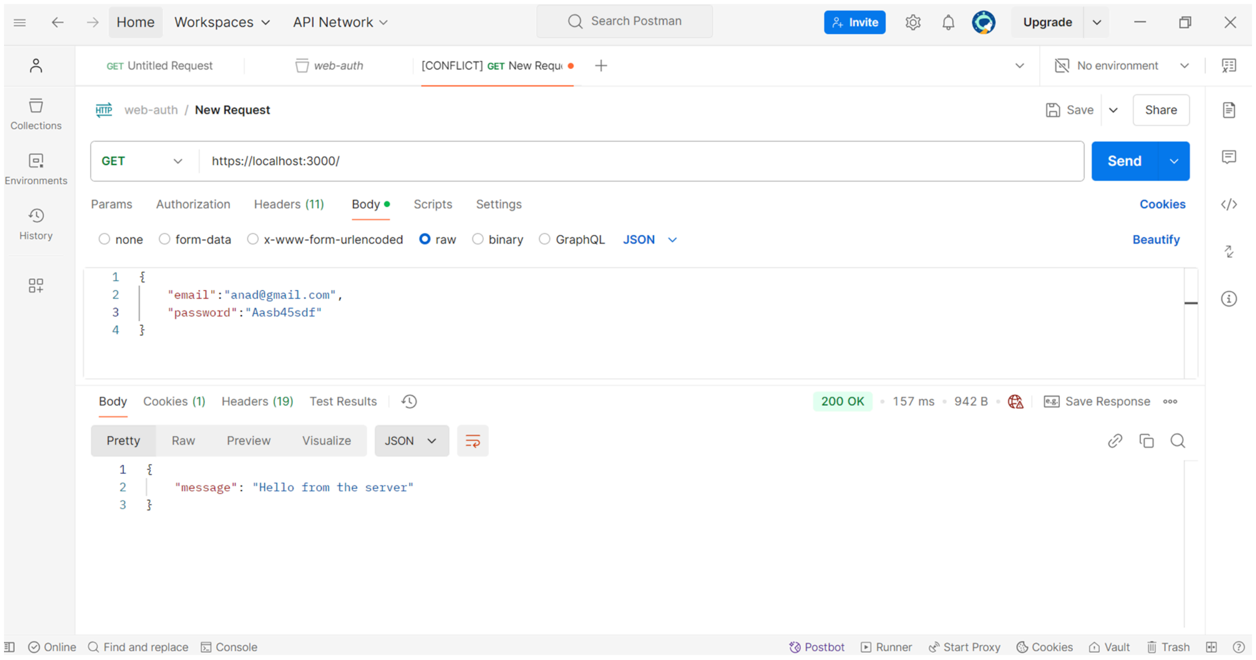
Similarly to the prior screenshot that shows the localhost server starting up from VSCode, here is the output showing the server running in AWS.


STORAGE SIZE: 20KB LOGICAL DATA SIZE: 180B TOTAL DOCUMENTS: 1 INDEXES TOTAL SIZE: 40KB

QUERY RESULTS: 1-1 OF 1

```
_id: ObjectId('675deb4eb7819ceb3e3ac45d')
email : "abc@gmail.com"
password : "$2a$12$eoiaRehJlhvQHZ3YdTUCX.VjrHtSX0uUNRu8cg2q5vbDxRmWbLs/2"
verified : false
createdAt : 2024-12-14T20:32:14.831+00:00
updatedAt : 2024-12-14T20:32:14.831+00:00
__v : 0
```

The following screenshot displays the **MongoDB Atlas** interface, where a query has been executed to retrieve the list of users stored in the database. The query result confirms that the user data has been successfully stored, showing key fields such as `email` and `password`. This demonstrates the proper functioning of the user registration feature, where data submitted through the signup endpoint is validated, processed, and securely stored in the database.

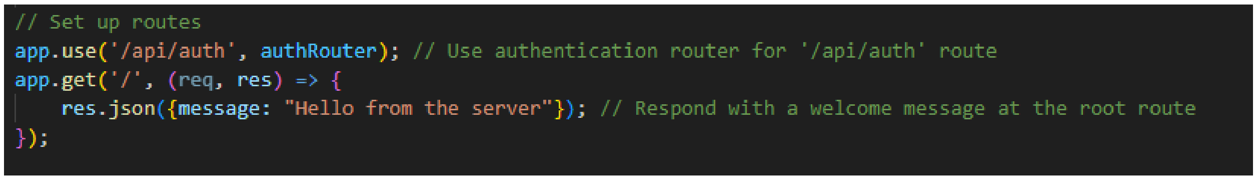

GET https://localhost:3000/

```
{
    "email":"anad@gmail.com",
    "password":"Aasb45sdf"
}
```

200 OK · 157 ms · 942 B

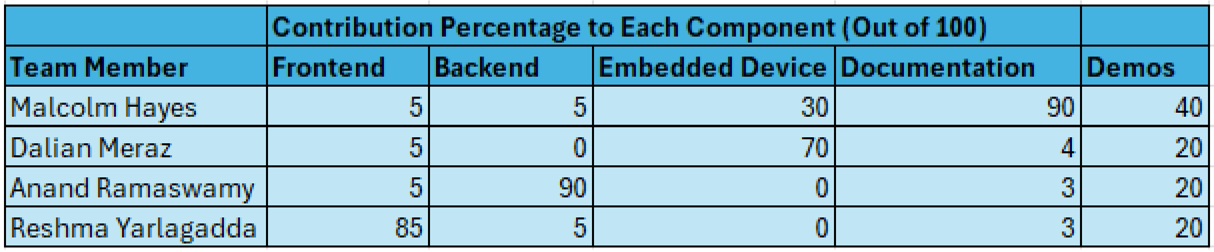
```
{
    "message": "Hello from the server"
}
```

The following screenshot from Postman shows the correct implementation of HTTPS on the localhost. A GET request is sent to [https://localhost:3000/](https://localhost:3000/) with our email and password that has been registered on our website. Then, we are properly seeing our output that we are able to properly receive GET requests sent via HTTPS. This message is sent by our index.js file, which is running the following lines of code (which verifies website functionality).



```
// Set up routes
app.use('/api/auth', authRouter); // Use authentication router for '/api/auth' route
app.get('/', (req, res) => {
    res.json({message: "Hello from the server"}); // Respond with a welcome message at the root route
});
```

## Contribution Table



| | Contribution Percentage to Each Component (Out of 100) | | | | |
|---|---|---|---|---|---|
| **Team Member** | **Frontend** | **Backend** | **Embedded Device** | **Documentation** | **Demos** |
| Malcolm Hayes | 5 | 5 | 30 | 90 | 40 |
| Dalian Meraz | 5 | 0 | 70 | 4 | 20 |
| Anand Ramaswamy | 5 | 90 | 0 | 3 | 20 |
| Reshma Yarlagadda | 85 | 5 | 0 | 3 | 20 |

## Lessons Learned

1. Developing a responsive web application emphasized the importance of accessibility and user experience across various devices, such as desktops, tablets, and smartphones.
2. Implementing token-based authentication and secure APIs taught us how to safeguard sensitive user data and ensure the integrity of communications between the IoT device and the backend.
3. Working with the Particle Argon and MAX30102 sensor highlighted the challenges of achieving accurate and reliable measurements, necessitating stabilization algorithms and thorough testing.
4. Building a robust backend using Node.js, Express, and MongoDB demonstrated the need for scalable architecture to handle real-time data storage and retrieval efficiently.
5. Conducting integration and end-to-end testing reinforced the value of iterative development and early feedback from users to refine both functionality and usability.
6. Managing offline data synchronization for the IoT device provided insights into designing fault-tolerant systems that maintain functionality even when connectivity is lost.
7. Collaborating as a team across frontend, backend, and IoT development emphasized the importance of clear communication and effective project management to ensure all components work seamlessly together.

## Challenges

1. Achieving Accurate Sensor Measurements: Initially, the MAX30102 sensor produced

inconsistent readings due to issues with our firmware code. To address this, we introduced a delay for each reading so that the server was not overloaded with data readings. We also changed the code to output our readings as JSON so that our outputs were more organized

2. Ensuring Seamless Backend and IoT Integration: During testing, we faced issues with dropped data packets during transmission from the IoT device to the server. We resolved this by adding retry logic on the IoT device firmware and validating data receipt on the backend.
3. Responsive Web Application Design: Creating a consistent user experience across different devices presented layout challenges, particularly for data visualization. We addressed this by using the responsive charting library Chart.js.
4. Offline Data Synchronization: Ensuring data integrity when the IoT device was offline required careful design. We implemented local storage for up to 24 hours and designed a synchronization protocol to upload data when connectivity was restored.
5. Secure User Authentication: Implementing robust security for user accounts and IoT devices required learning about and integrating token-based authentication. By using JSON Web Tokens (JWT) and API key validation, we ensured secure access control for users and devices.
6. Debugging State Machine Transitions: Programming the IoT device's state machine to handle various scenarios, such as data transmission and user prompts, revealed edge cases that caused unintended behavior. We resolved these issues by thorough testing and refining the state transition logic.
7. Team Collaboration Across Components: Coordinating development across the IoT device, backend, and frontend was initially challenging due to dependencies. Regular team meetings and using tools like Git for version control helped streamline collaboration and integration efforts.

# References

We have used 3rd party device drivers to interface with the MAX30102 heart monitoring device. We have based our drivers from Sparkfun Electronics, who have provided the open source drivers via GitHub (https://github.com/sparkfun/SparkFun_MAX3010x_Sensor_Library)

We also looked at other drivers for the MAX30102 sensor as we had issues with debugging for our firmware. These issues made us branch out and look at several libraries to see which one worked best. The git repository is available at https://github.com/moononournation/BloodOxygenHeartRateMeter/blob/main/max30102.h.