ECE 513, University of Arizona

Heart Rate Monitor System

Project Documentation / User Manual

By
Amber Parker, Seven Gilbert, Samantha Perry

Table of Contents

1. Project Overview

1.1 Project Description

This project is a secure platform designed to monitor heart rate and SpO_2 in a simple and accessible way. Patients use the system to record and view their vital signs, while physicians can monitor their assigned patients and review both daily readings and longer-term weekly trends. Admins and operators are responsible for managing device deployment and overall system setup. The platform also allows measurement frequency to be adjusted as needed, making it flexible for different patient monitoring needs.

The project can be found at this GitHub repository:
https://github.com/samnperry/ECE513FinalProject.

1.1.1 Intended Users

Patients can register their device, take measurements, review weekly averages and daily charts, and set measurement frequency to what they want.

Physicians can log into a separate portal to view assigned patients, see 7-day summaries, look at daily readings, and override measurement frequency.

1.1.2 System Summary

1.1.2.1 Frontend (React)

The frontend is a responsive web app built around certain user roles. Patients can create accounts, log in, register their device, and view their heart-rate and SpO_2 data through daily and weekly dashboards, along with basic account management. Physicians have access to an all-patients view, individual patient summaries, detailed daily readings, and the ability to override how often a patient's device takes measurements when needed.

1.1.2.2 Backend (Node / Express / MongoDB)

The backend provides REST APIs that handle authentication, account updates, device registration, and incoming sensor measurements. It also manages physician-to-patient assignments and holds both patient and physician dashboards. Each device has a unique API key, and physician-defined measurement overrides are stored and sent back to devices through a configuration endpoint (GET /api/device/config/:deviceId).

1.1.2.3 Embedded Device (Photon)

The Photon reads heart-rate and SpO_2 data using the MAX30102 sensor, checks the backend for its current measurement interval, and sends readings to the server using its device-specific API key. If the device loses connectivity, it temporarily stores the data locally and automatically uploads it once the connection is restored.

2. System Architecture and Implementation

2.1 System Architecture Overview

The system is designed as a simple, end-to-end flow that connects a wearable device to a web-based monitoring platform. At a high level, the embedded device (a Particle Photon 2 with a MAX30102 sensor) collects heart rate and SpO_2 data and communicates with a Node/Express backend over HTTP(S). The backend stores and processes this data in MongoDB and makes it available to users through a React-based frontend for both patients and physicians.

The data flow starts on the device, which samples heart rate and SpO_2 values and checks in with the backend to see how often it should be taking measurements. This configuration is fetched from a device-specific endpoint using an API key. Once a valid reading is captured, the device sends the measurement to the backend, again using its API key for authentication. The backend validates the device, stores the data using MongoDB, and exposes it through REST APIs. The frontend then retrieves this information securely and displays it through dashboards and charts tailored to each user role.

The system uses standard web technologies throughout, including HTTPS over Wi-Fi for device communication, JWT-based authentication for users, and a REST-based architecture connecting the backend to the frontend. This approach keeps the system modular, easy to understand, and scalable.

2.2 Backend Implementation

The backend is built using Node.js with the Express framework and MongoDB for data storage, using Mongoose models for users, devices, and measurements. It handles authentication, device management, incoming sensor data, and all patient and physician views. User passwords are hashed with bcrypt, and JWTs are used to secure web-based routes. CORS is configured to allow requests only from the frontend origin.

Authentication routes allow both patients and physicians to sign up and log in, with JWT middleware protecting all authenticated endpoints. Device registration is handled through dedicated device routes, which generate a unique API key for each device. This API key is required for all device communication. Incoming measurements are sent to POST /api/measurements with the device's x-api-key, and each payload includes the device ID, heart rate, SpO_2 value, and timestamp. The backend validates the API key, stores the data in MongoDB, and makes it available through patient and physician dashboard routes.

The backend also manages relationships between patients, physicians, and devices. Patients can assign a physician to their account, and physicians can view summaries for all assigned patients, drill down into daily measurement data, and override how

frequently a device takes measurements. These overrides are stored with the device and returned to the device through the configuration endpoint GET /api/device/config/:deviceId, allowing the embedded system to dynamically update its measurement interval.

Environment variables are used to manage sensitive configuration values, including the database connection string (MONGO_URI), JWT secret (JWT_SECRET), and frontend API base URL (REACT_APP_API_BASE).

For deployment, the React frontend is built locally using npm run build, producing a static build/ directory. This directory is deployed to the AWS EC2 instance under /var/www/heartapp, where it is served by Nginx. The Node.js/Express backend runs on the same EC2 instance on port 5001 and is managed using PM2 to ensure it stays running across logouts and system reboots.

Nginx is configured as a reverse proxy for the application. It serves the static frontend and forwards all /api requests to the backend at 127.0.0.1:5001. HTTPS is enabled using Let's Encrypt via Certbot, which generates and manages SSL certificates for the domain. All traffic is encrypted over port 443, and certificate renewal is handled automatically.

The full deployment was verified using HTTP and HTTPS requests with curl and by reviewing Nginx logs to confirm that authentication and API requests were successfully reaching the backend. This setup provides a complete end-to-end deployment where the frontend is served through Nginx, the backend is reliably managed by PM2, and all communication is secured using HTTPS..

2.3 Frontend Implementation

The frontend is built in React using TypeScript and is structured around role-based access. Patients and physicians see different views and features based on their role, but the overall interface remains consistent and easy to navigate.

Patients can create an account, register or remove devices, assign a physician, and view their data through weekly summary dashboards and detailed daily charts. They also have access to basic account management features, such as updating their password.

Physicians use a separate portal that provides an overview of all assigned patients, including seven-day statistics such as averages, minimums, and maximums. From there, they can view individual patient summaries, drill down into daily measurement details, and adjust the measurement frequency for specific devices when needed.

The frontend communicates with the backend using secure HTTPS requests and includes JWTs in the Authorization header. Data visualization is handled with Chart.js, using line charts for weekly trends and bar charts for daily details, along with tables for

clear presentation of statistics and lists.

- Tech stack/UI: “React + TypeScript with React Router for routing, localStorage for tokens/role, and Pages.css for consistent cards/forms across login/signup/dashboard. Protected routes redirect unauthenticated users.”
- Role routing: “Header/nav hides patient-only links for physicians and vice versa; login redirects physicians to /physician-dashboard and patients to /dashboard.”
- Data flow: “All API calls use fetch with Authorization: Bearer <JWT> and the deployed base https://sfwe513.publicvm.com/api; errors surface as inline banners.”
- Components: “Dashboard cards show weekly stats, measurement frequency controls, and device filters; forms use shared styles; physician tables list patients with action links to summary/daily views; device and physician selectors are populated from live API calls.”
- Charts: “Chart.js line charts for weekly trends, daily detail overlays heart rate vs. SpO2; tables provide min/max/avg and counts for the last 7 days; daily view charts update with date filters.”

2.4 Embedded Device Implementation

The system uses a Particle Photon 2 microcontroller using the Particle Device OS, (Particle.h), with built-in Wi-Fi and two onboard LED’s used for visual output indicators. The microcontroller connects to a MAX30102 Heart-Rate / SpO2 sensor via I2C. The sensor is also powered using the voltage output of the Photon 2. The EEPROM hardware storage on the Photon 2 is used for offline data collection.

To collect data, the particle board samples the MAX30102 sensor to read current IR values during the measurement period. When the IR value hits a threshold that determines the presence of a finger, the Photon 2 will start recording data from the sensor with help from the spo2_algorithm.h file included. Data is filled into a rolling buffer that, when populated, is used to calculate the users heart rate and SpO2. These measurements are then recorded and verified by the microcontroller. Once a sufficient number of valid measurements have been taken, the Photon 2 can package and send the data.

The sensor is set to a sampling rate of 25 Hz, or every 40 ms. This data is stored into a rolling buffer of 100 recordings. The data is then used to calculate the heart rate and SpO2 values. Additionally, variables for if those calculations represent valid outputs are set. These need to remain true for a specified number of sequential measures for data to be deemed safe for sending. Time stamps are used for data and event tracking; unix seconds are used to represent our time stamps.

The data is transmitted via WiFi to the Particle Cloud Server. This is done when the custom webhook made in the online Particle Console is triggered using the private particle event, “Photon2_SendEvent”. The microcontroller then uses webhooks to verify whether or not the data was sent successfully. Similarly, the Photon 2 sends a GET request to the webhook using the event “Photon2_Config_Request” in order to obtain the most recent configuration for frequency of measurement times. This information is collected by the front end and saved in the server.

Local processing and filtering is performed on sensor data and recorded measurements. A threshold is set for the incoming IR values to filter out any irrelevant data. Once the sensor begins to make valid measurements, the program waits until the data has stabilized before sending. Measurements made are stored into local memory for up to 24 hours where they can be later uploaded when able.

3. File and Code Structure Description

3.1 Backend Files

3.1.1 Models

Path: server/models

The user.js model stores all user-related information, including email, hashed password, role (patient or physician), registered devices, and any assigned physician. The device.js model represents each registered device and includes the device ID, the associated user, a unique API key, an optional measurement frequency override, and a nickname for easier identification. The measurement.js model stores incoming heart rate and SpO_2 readings along with the device ID and timestamp.

- user.js: Stores users (email, password hash, role: patient/physician, devices[], assignedPhysician).
- device.js: Stores devices (deviceId, user ref, apiKey, measurementFrequencySeconds, nickname).
- measurement.js: Stores heart rate/SpO_2 readings (deviceId, heartRate, spo2, timestamp).

3.1.2 Routes

Path: server/routes

Authentication and account-related functionality is handled in authRoutes.js, which supports signup, login, password updates, and JWT generation. Device-related actions are handled in deviceRoutes.js, including device registration (which returns a unique API key), listing devices, and providing device configuration through /api/device/config/:deviceId. Incoming sensor data is handled by

measurementRoutes.js, which accepts POST requests from devices using an API key and allows recent measurements to be retrieved by device. Physician-specific functionality lives in physicianRoutes.js, covering physician registration and login, patient assignment, all-patient statistics, per-patient summaries, daily views, and device frequency overrides. An optional accountsRoutes.js file is used for additional account-related helpers when needed. Most of the business logic is implemented directly inside the route files. This includes password validation, API key generation during device registration, measurement validation and storage, physician-only data aggregation, saving frequency overrides, and serving configuration data back

- authRoutes.js: Signup/login, account update (password), JWT issuance.
- deviceRoutes.js: Register device (returns per-device apiKey), list devices, get device config /api/device/config/:deviceId (includes measurementFrequencySeconds).
- measurementRoutes.js: POST measurements with x-api-key, GET recent measurements by device (enforces apiKey/device match).
- physicianRoutes.js: Physician register/login, list physicians, assign physician to patient, all-patients 7-day stats, patient summary/daily, override device frequency.

3.1.3 Controllers / Services (if applicable)

The backend does not use separate controller or service layers. Instead, all business logic is handled directly inside the route files, which keeps the project easier to follow and debug. This includes authentication and password validation, device registration with automatic API key generation, validation, and storage of incoming measurements, and aggregation of patient statistics for physician views. Physician-only functionality, such as overriding device measurement frequency, is also handled here, along with serving updated configuration data back to devices when they request it.

3.2 Frontend Files

3.2.1 Public Files

The public/index.html file serves as the entry point for the React single-page application and includes the landing page introduction shown before the app loads. Static assets such as the favicon, manifest, robots.txt, and team photos are stored directly in the public/ directory and are loaded by the browser as needed.

3.2.2 JavaScript / Components

Utilizing React app is initialized in src/index.tsx, with App.tsx defining the main layout and all route definitions. Authentication components handle patient and physician login and signup. Patient-facing components include the main dashboard, which shows

weekly summaries and daily charts, along with the home page and device registration flow. The account page allows patients to update their password, add or remove devices, and assign a physician. Physician functionality is grouped into its own set of components, including an all-patients dashboard with seven-day statistics, individual patient summaries, detailed daily views, and controls to adjust how often a patient's device takes measurements. Shared components such as the header and navigation bar handle role-based navigation so each user only sees the pages relevant to them. State management is handled mostly with local useState and useEffect hooks. All API communication uses standard fetch calls to the backend defined by REACT_APP_API_BASE, with JWTs passed in the Authorization: Bearer header. Device API keys are never exposed to or used by the frontend.

3.3 Embedded Device Files

The embedded device code is centered around the main firmware file, 513Photon2.ino (with a matching .cpp file generated to contain the same logic). This firmware implements a state machine that controls user prompting, sensor sampling, and data transmission. Device-specific values such as the API key and backend host information are set before building the firmware.

Sensor handling is implemented using the MAX30102 driver and supporting algorithm files, which handle heart rate and SpO_2 calculations. MAX30105.cpp is the SparkFun MAX3010x driver that wraps all I^2C register definitions, configuration, FIFO access, LED pulse parameters, etc. It gives the firmware low-level control over the IR/Red photodiode hardware., The file heartRate.cpp implements a Maxim's PBA heart-beat detector: it filters incoming IR samples, tracks AC/DC components, and emits beat events. The spo2_algoryihm.cpp file provides the Maxim SpO_2/heart-rate computation routine (maxim_heart_rate_and_oxygen_saturation) that the firmware calls after filling its sample buffers.

The device uses HTTP networking to request its current configuration from the backend and update its measurement interval dynamically; frequency-webhook.json. Measurements are sent to the backend over HTTP, and the device includes local EEPROM buffering so readings are not lost during temporary network outages. Once connectivity is restored, any queued measurements are automatically sent. 513Photon2.ino handles the implementation of these actions.

4. Results and Demonstration

The system was demonstrated through screenshots showing key functionality for both patient and physician roles. These include login and registration pages, the patient dashboard with weekly summary statistics and daily detail charts, and heart rate and SpO_2 visualizations created using Chart.js. Additional screenshots show device

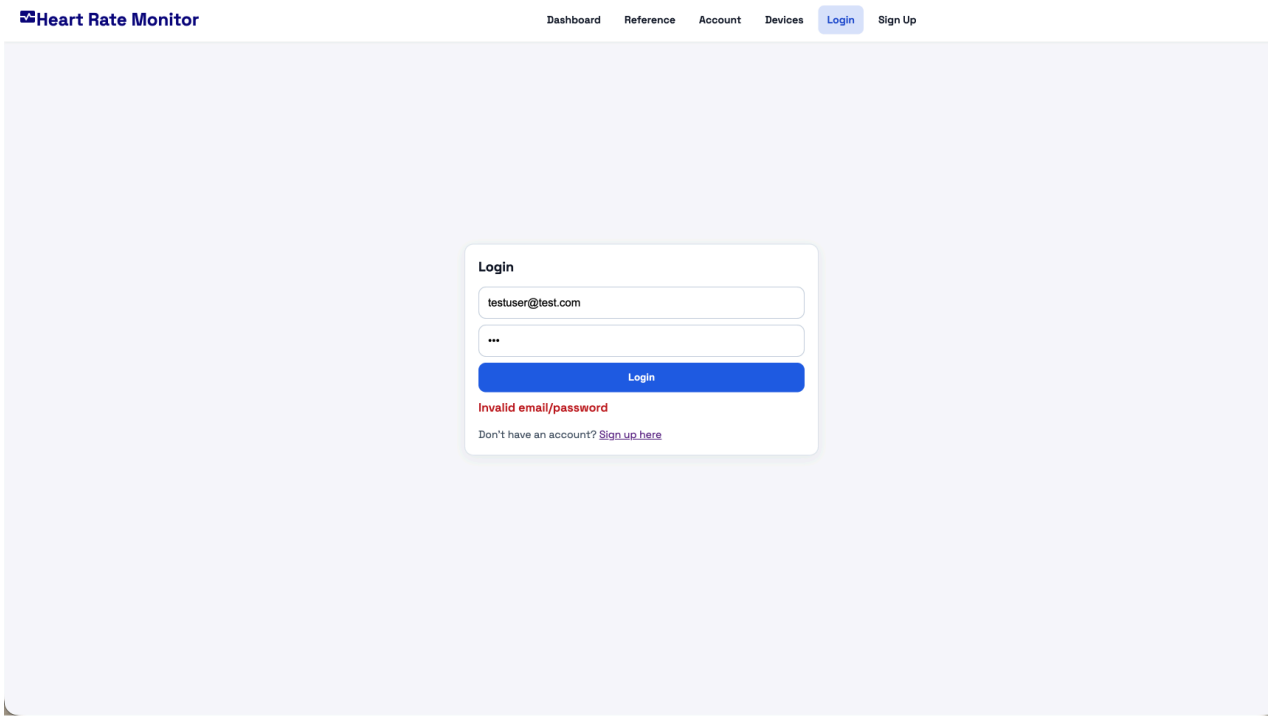
management from the patient account page, including adding and removing devices, as well as physician management features. The physician portal screenshots highlight the all-patients overview with seven-day averages, minimums, and maximums, individual patient summaries, detailed daily views, and the ability to override device measurement frequency.

4.1 Frontend Results

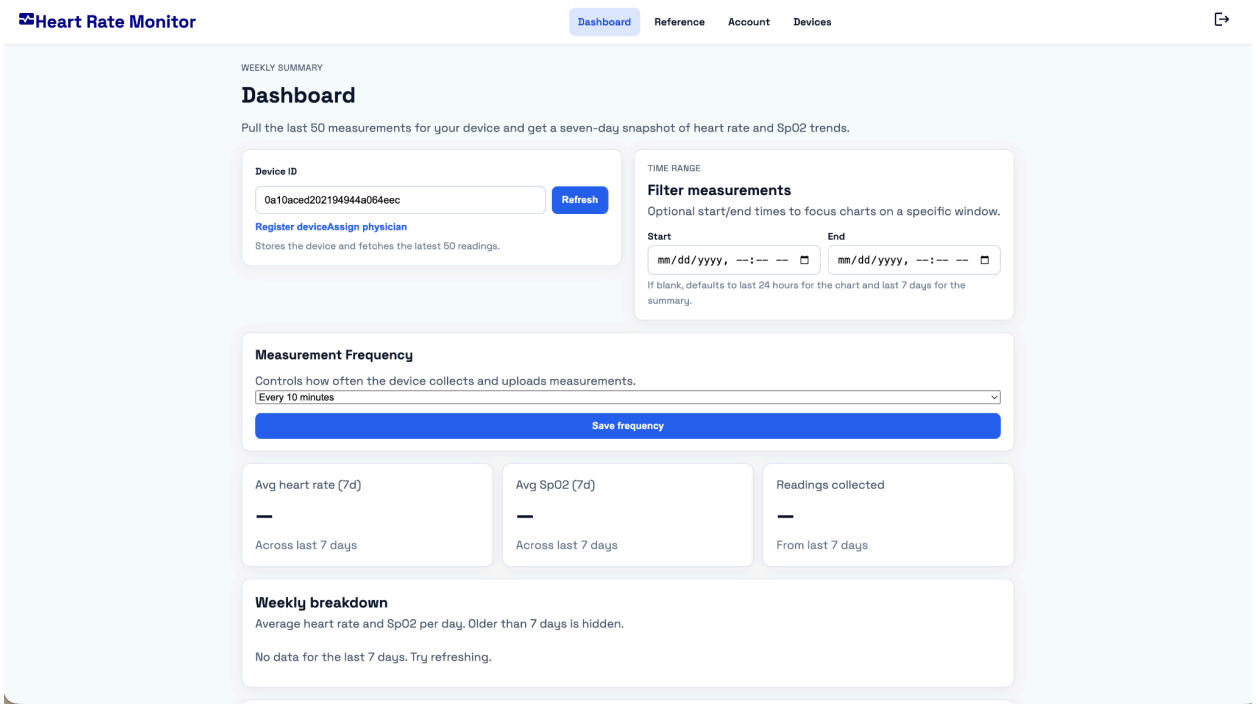
4.1.1 Login and registration pages

The frontend authentication flow was validated using both the login and registration pages. The login page shows standard email and password inputs along with clear error messaging when invalid credentials are entered. The registration page enforces password requirements and displays a success message once an account is created, confirming that the authentication flow and form validation are working as expected.

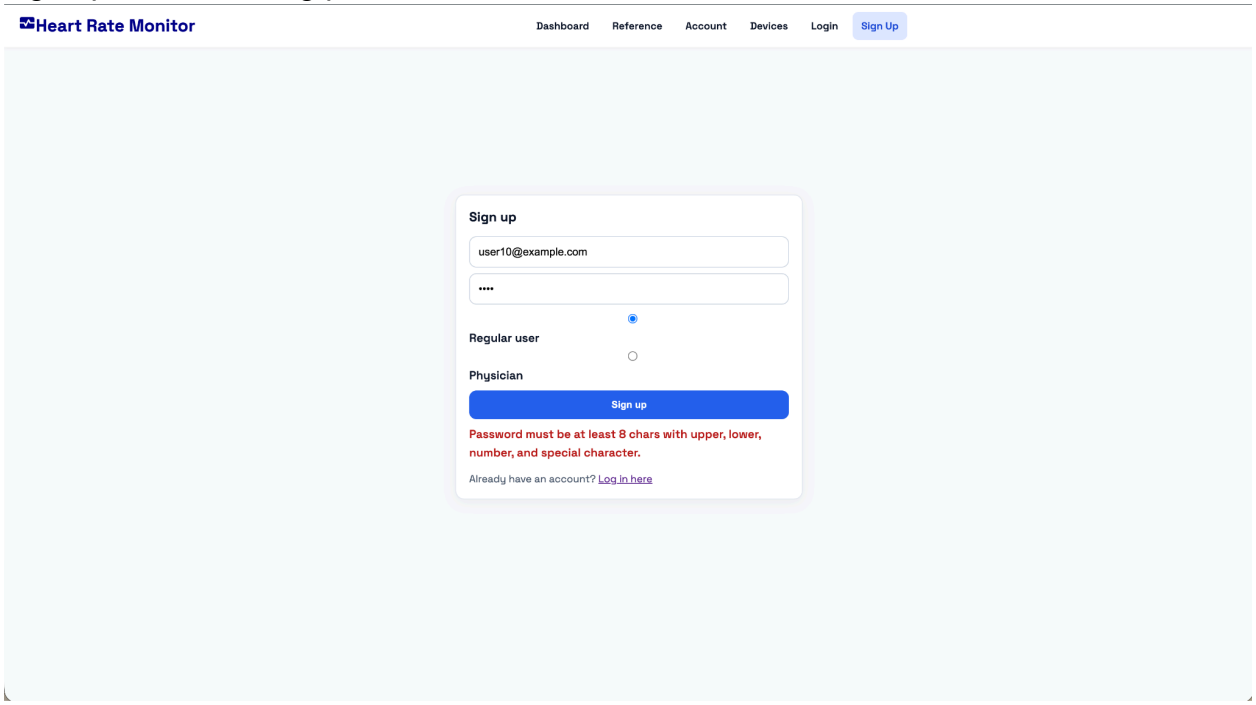
Invalid Login:



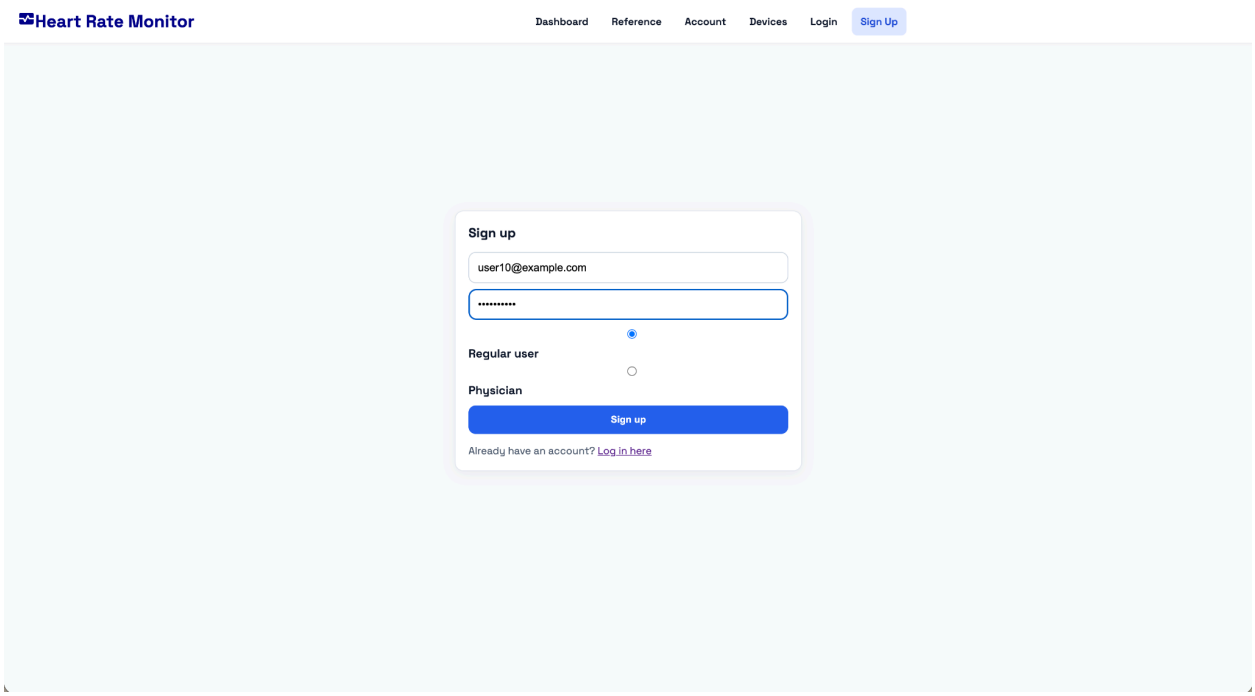
Valid Login that directs to Dashboard:

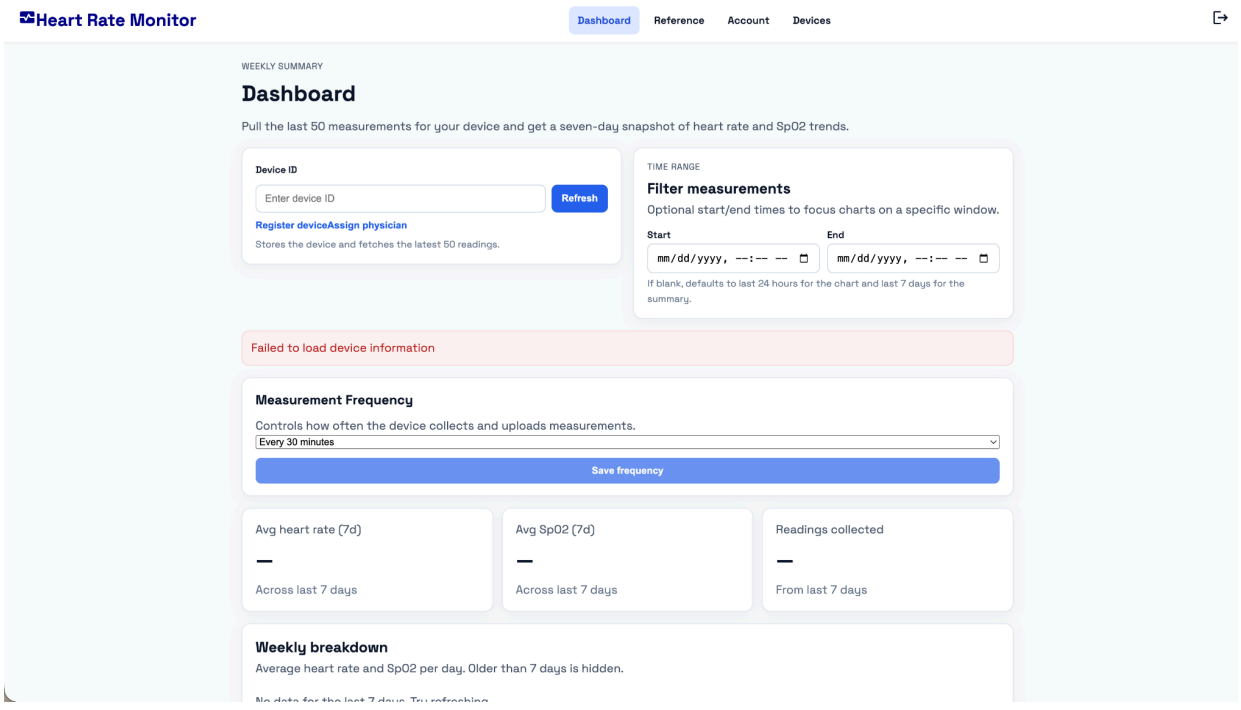


Sign up without strong password:

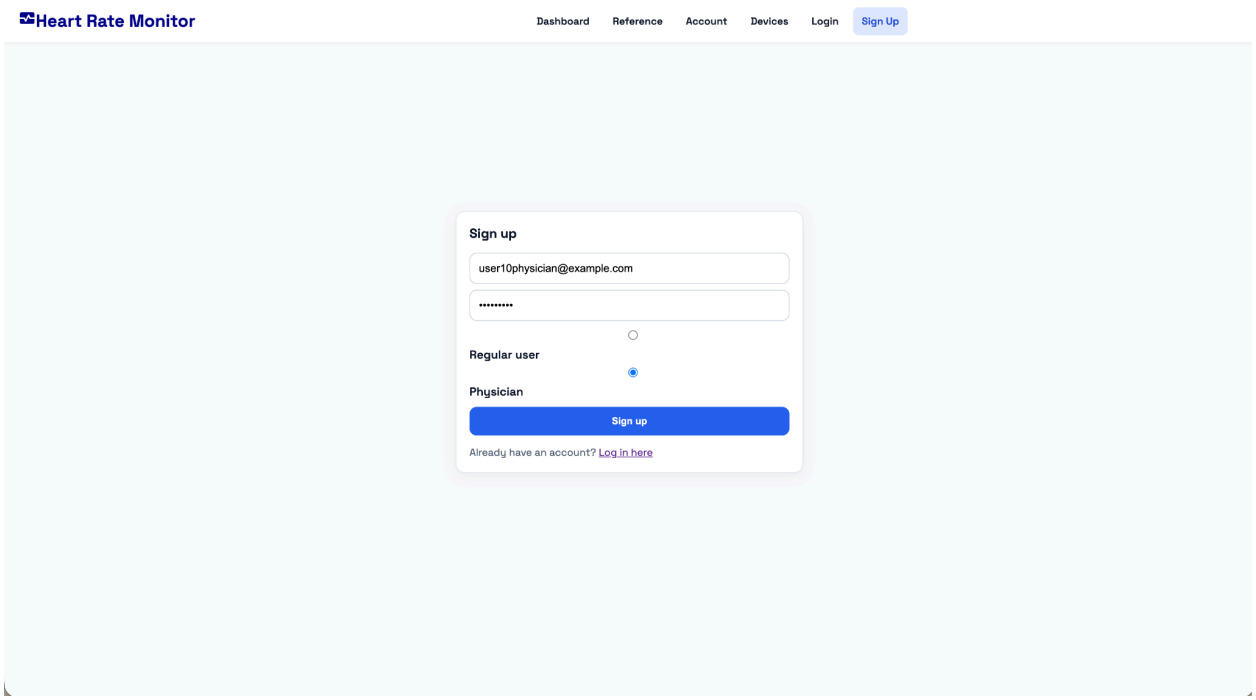


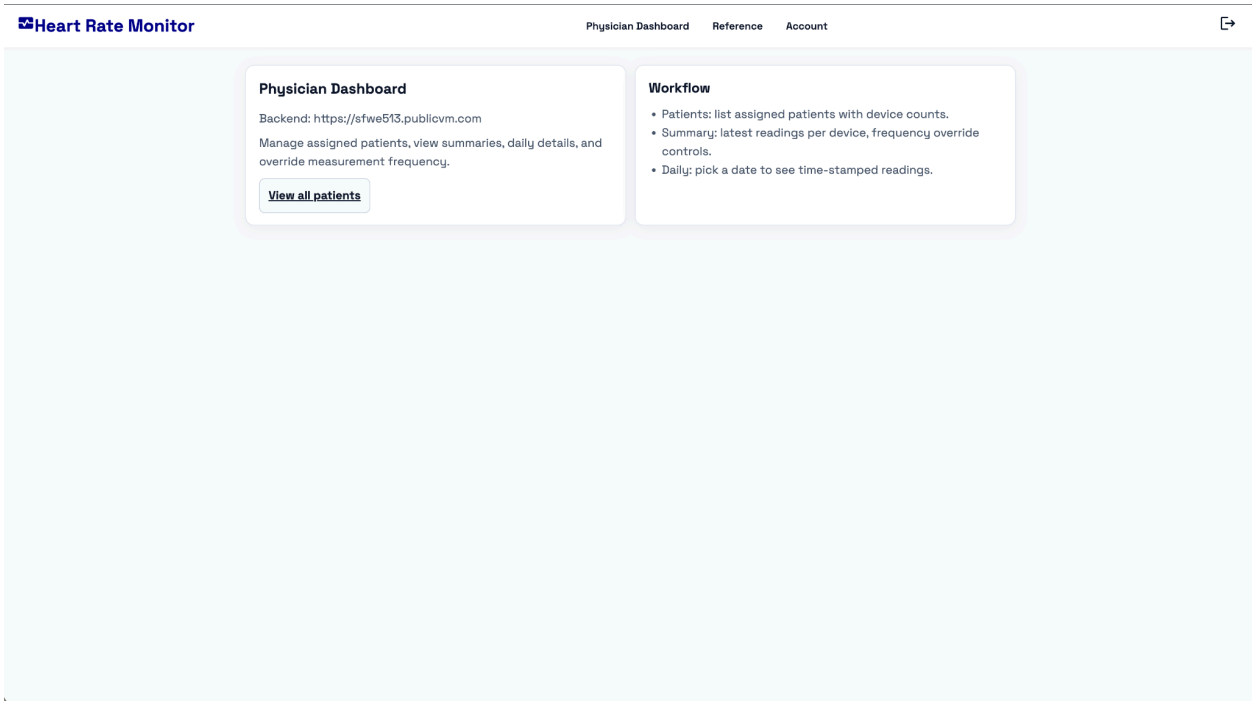
Sign up with strong password to dashboard (shows failed to load device information as there is no device registered to the user):





Sign up for physician to physician dashboard:



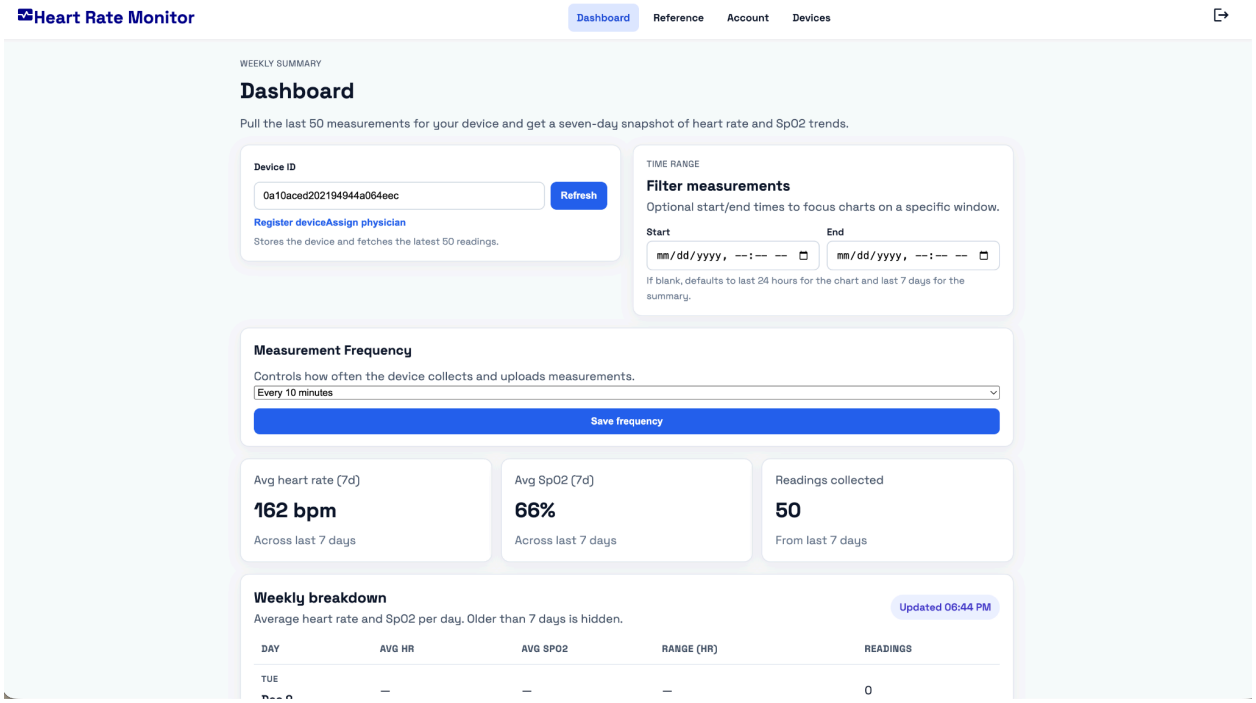


4.1.2 Patient dashboard

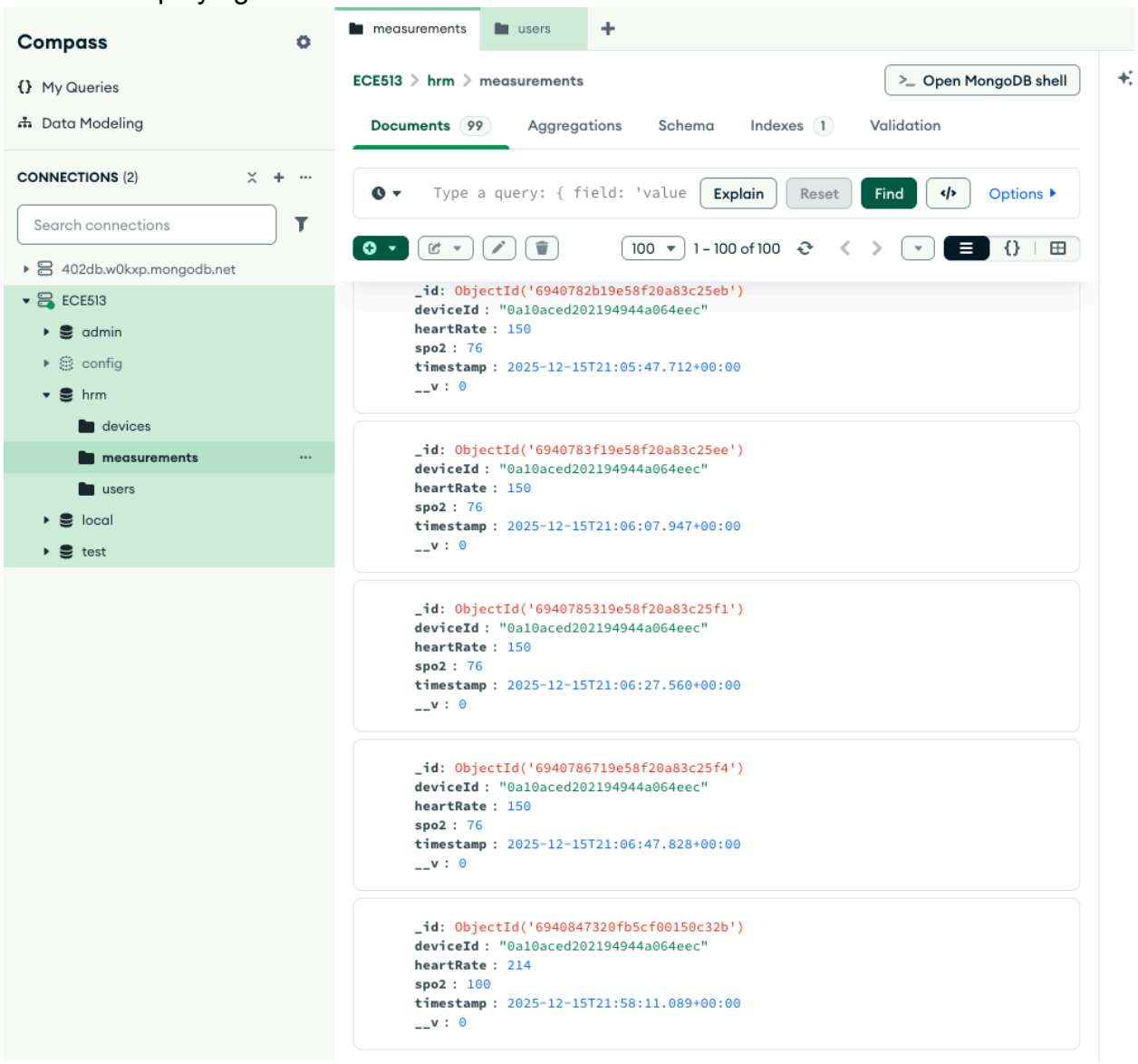
After logging in as a patient, the dashboard displays a weekly summary view that includes average heart rate, average SpO_2, and the total number of readings collected. The dashboard also shows when the data was last updated and provides a seven-day

breakdown of measurements for the registered device. This view confirms that data is being correctly pulled from the backend and summarized over the expected time window.

Dashboard:



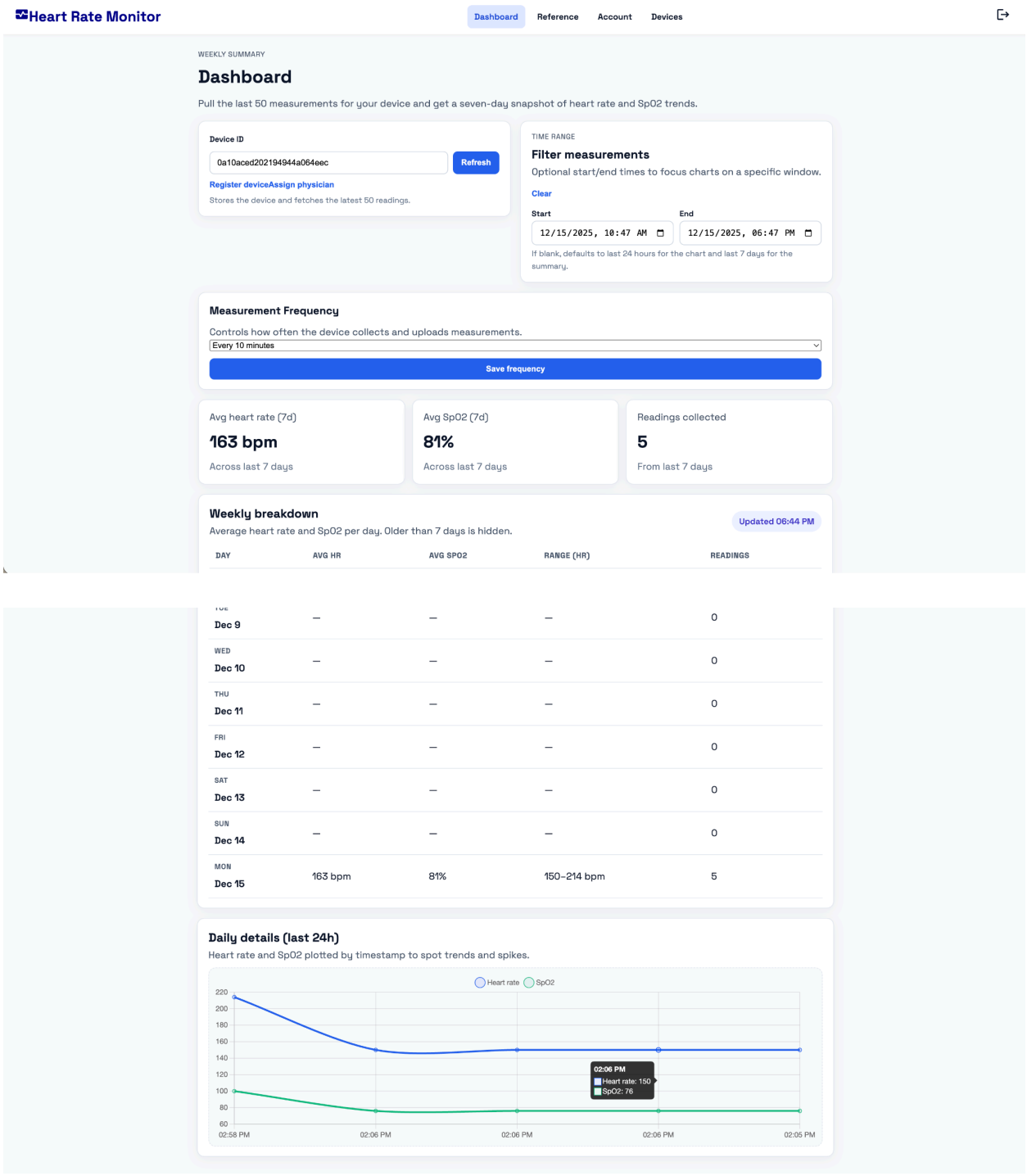
Backend displaying measurement data:



4.1.3 Heart rate and SpO_2 visualization

Daily charts were generated using Chart.js to visualize heart rate and SpO_2 over time for a selected date. The charts clearly show changes throughout the day, with labeled axes for time and measurement values. These visualizations confirm that individual readings are stored correctly and rendered accurately in the frontend.

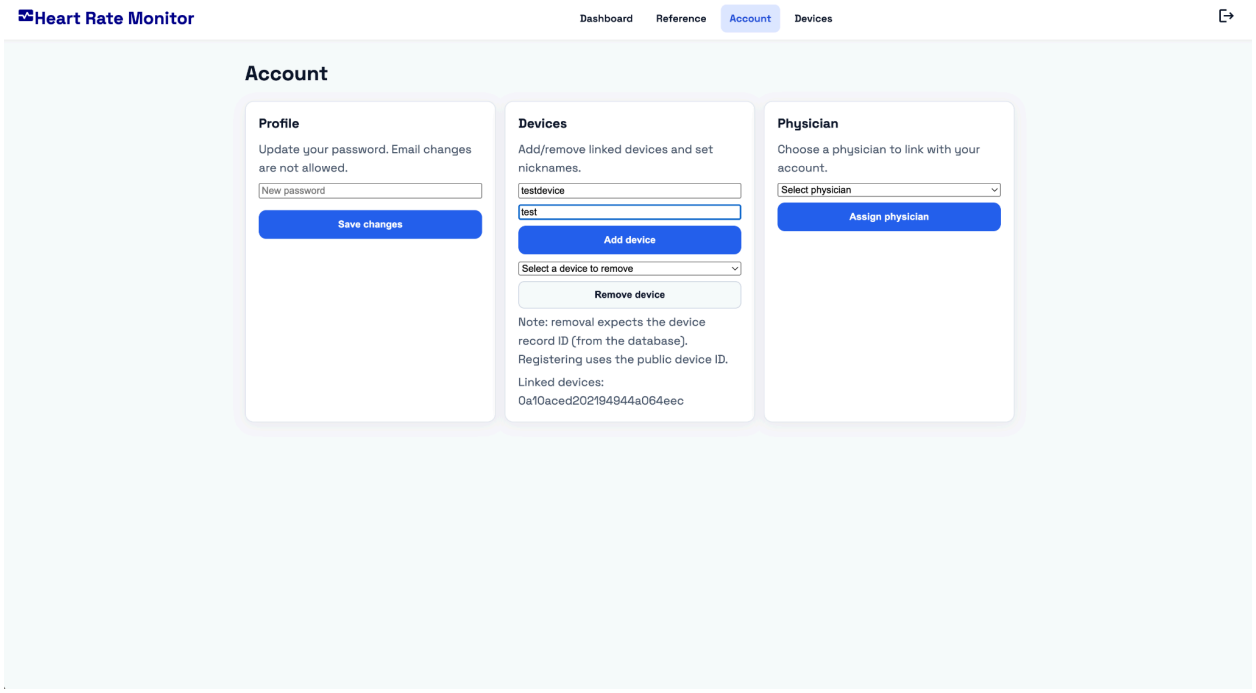
Charts:

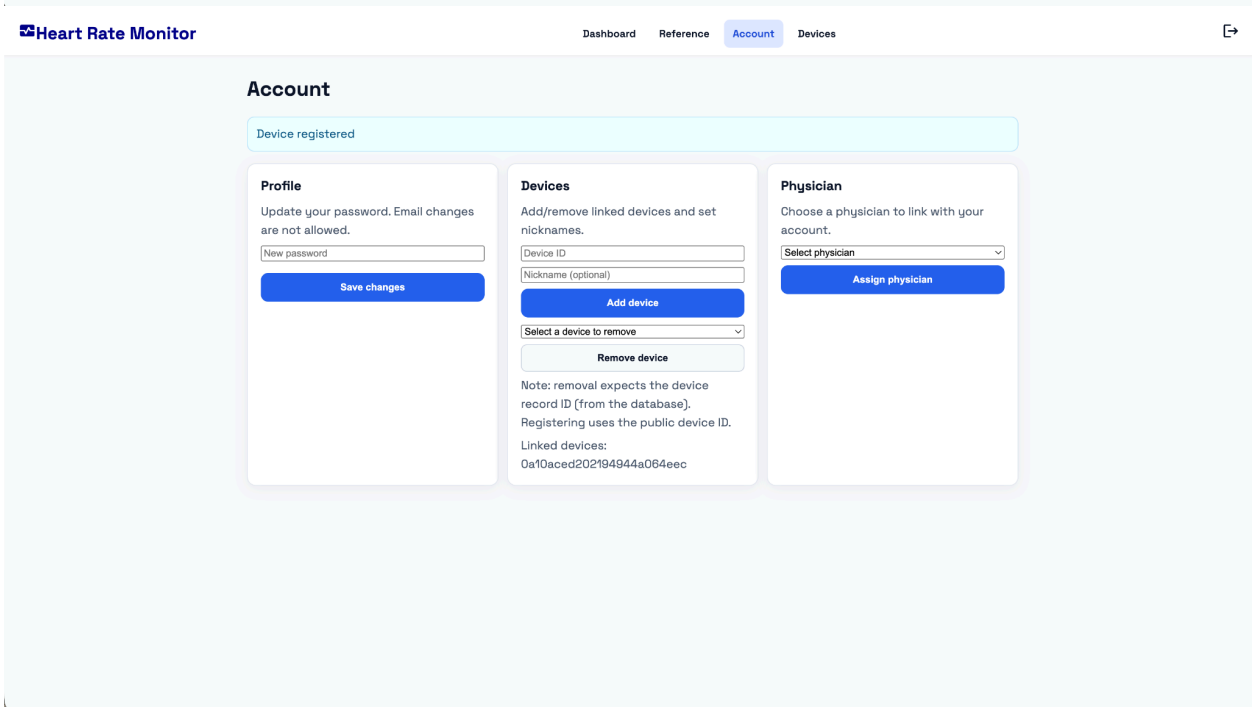


4.1.4 Device management (add/remove device)

The Account page allows patients to register new devices by entering a device ID and nickname, as well as remove devices that are no longer in use. The linked devices list updates immediately after these actions, confirming that device registration and removal are handled correctly by both the frontend and backend.

Add a device (it is added to the device list):

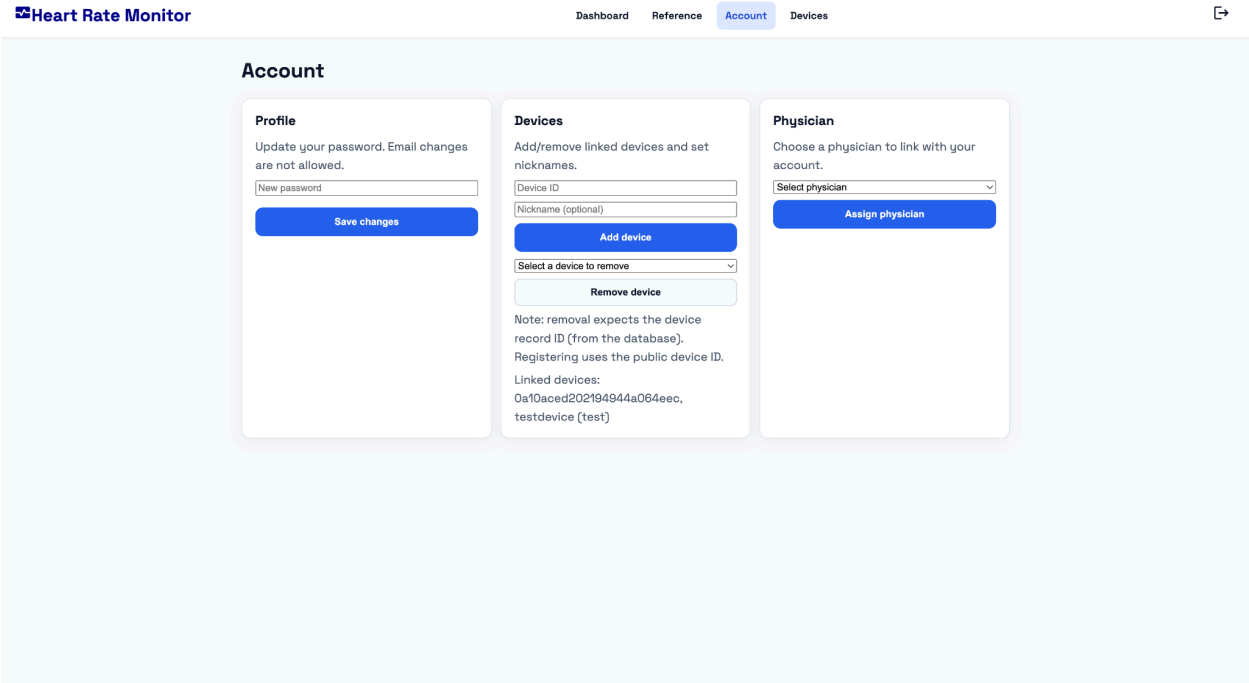




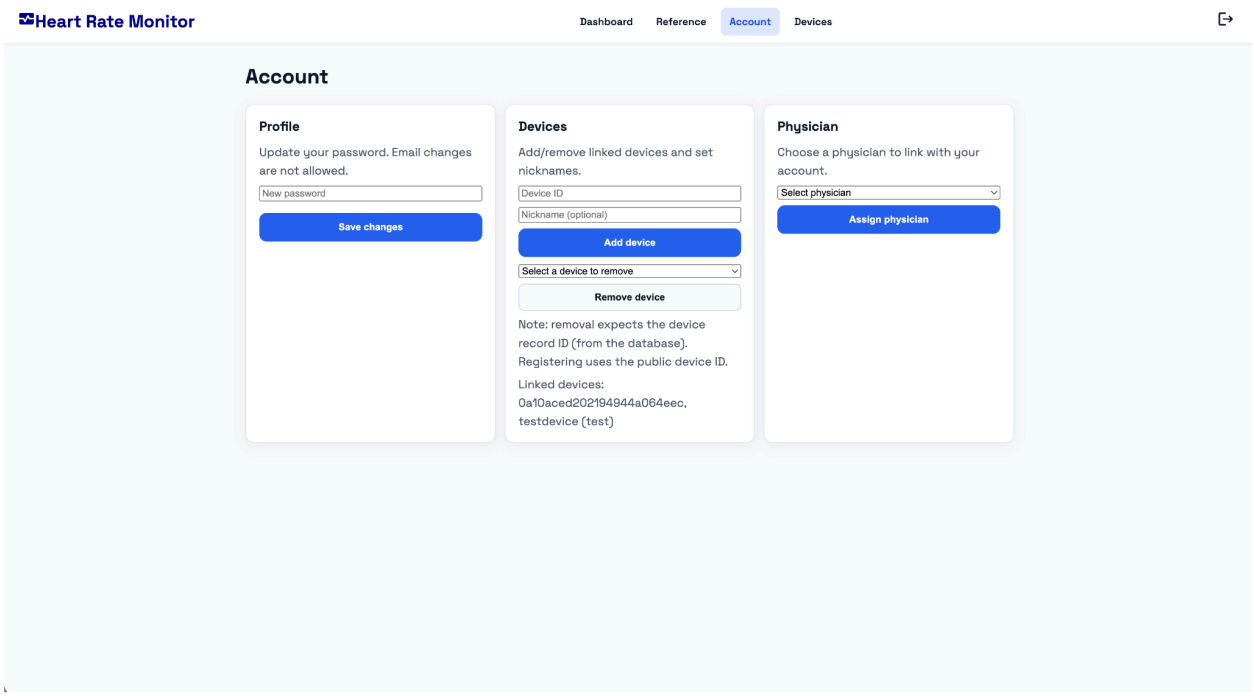
```
_id: ObjectId('693dbfc6db9dc580be50339b')
deviceId : "0a10aced202194944a064eec"
user : ObjectId('69364cb0ff8c94a67406005e')
apiKey : "3cf562803c98eee6f2df540bdc3b45a61e2b93200790ca78dbbfe4c3ce47a38c"
__v : 0
measurementFrequencySeconds : 600
```

```
_id: ObjectId('6940bba3b46a9bb595641a76')
deviceId : "testdevice"
nickname : "test"
user : ObjectId('69364cb0ff8c94a67406005e')
apiKey : "dbb4c25eb9b800c289236430ebf6f233e2dffadb77a4f1d2b928fe699a871fe0"
measurementFrequencySeconds : 1800
__v : 0
```


After refresh of page:



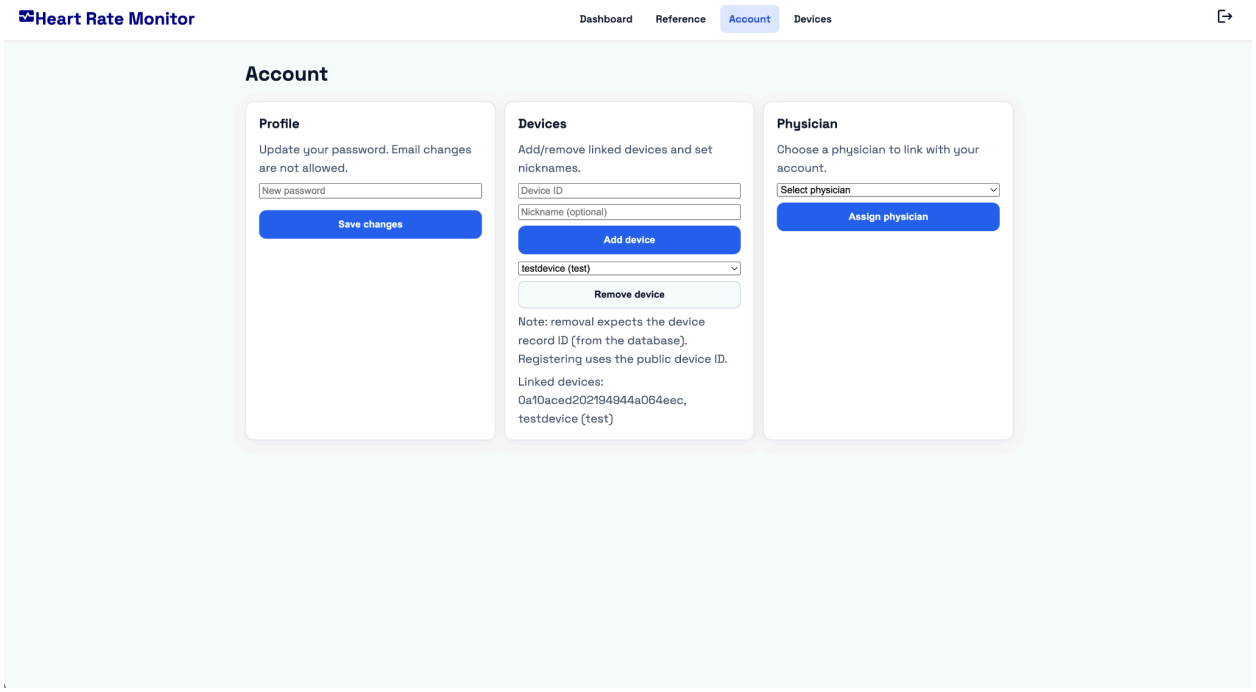
Device removal page:

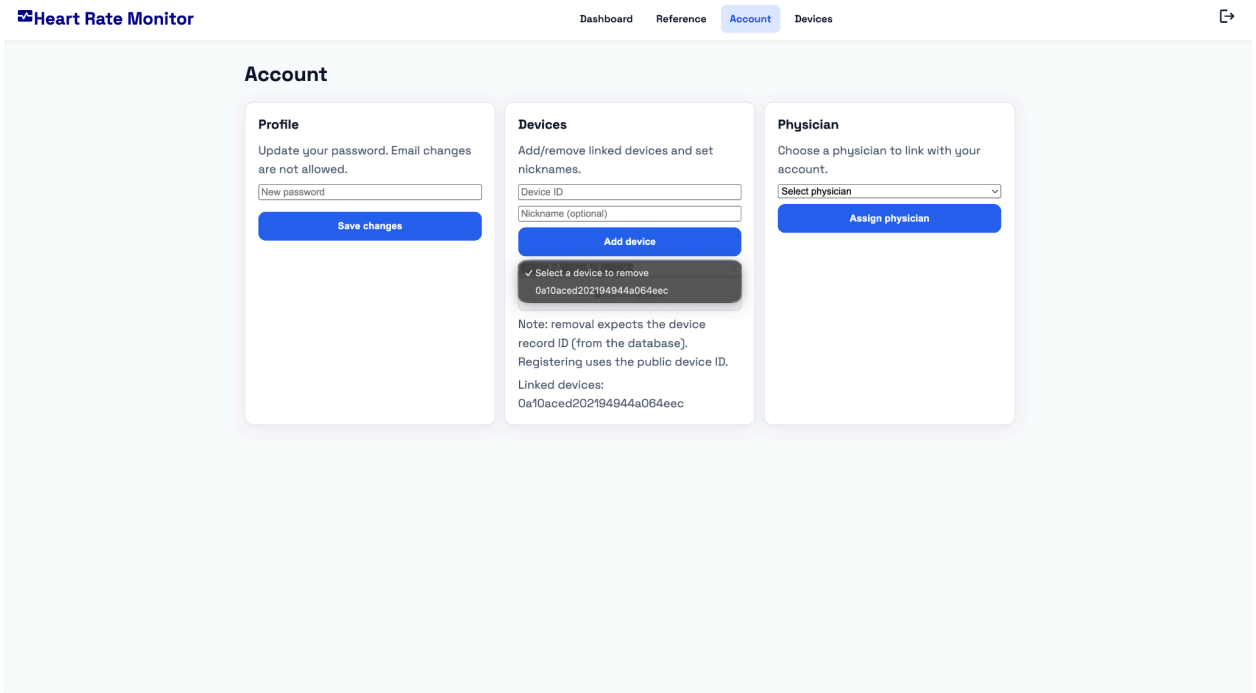


Removing a device (It is removed from the list of linked devices):


```
_id: ObjectId('693dbfc6db9dc580be50339b')
deviceId : "0a10aced202194944a064eec"
user : ObjectId('69364cb0ff8c94a67406005e')
apiKey : "3cf562803c98eee6f2df540bdc3b45a61e2b93200790ca78dbbfe4c3ce47a38c"
__v : 0
measurementFrequencySeconds : 600
```

```
_id: ObjectId('6940bacdb46a9bb595641a69')
deviceId : "testdevice"
nickname : "test"
user : ObjectId('69364cb0ff8c94a67406005e')
apiKey : "046f714f65118ff587b4ab34038ed75428b926a0474464a334db4032598ed0de"
measurementFrequencySeconds : 1800
__v : 0
```


Heart Rate Monitor
Dashboard
Reference
Account
Devices
Account
Profile
Update your password. Email changes are not allowed.
New password
Save changes
Devices
Add/remove linked devices and set nicknames.
Device ID
Nickname (optional)
Add device
testdevice (test)
Remove device
Note: removal expects the device record ID (from the database). Registering uses the public device ID.
Linked devices: 0a10aced202194944a064eec, testdevice (test)
Physician
Choose a physician to link with your account.
Select physician
Assign physician

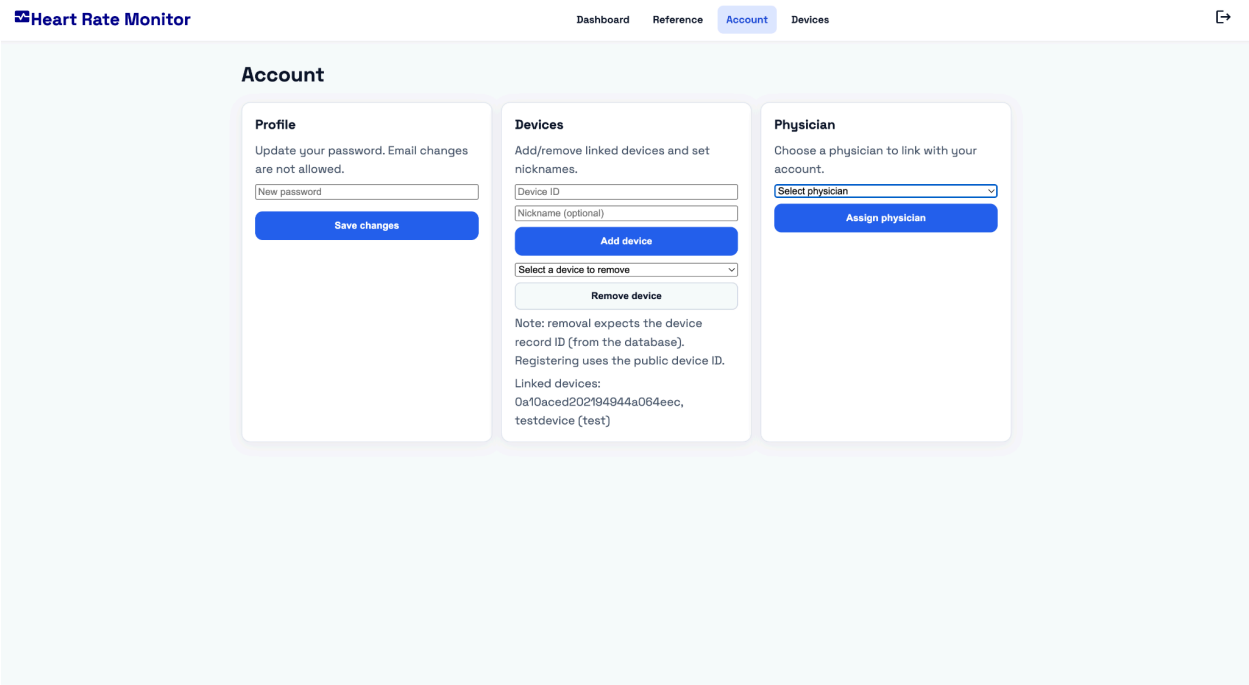

Heart Rate Monitor
Dashboard
Reference
Account
Devices
Account
Profile
Update your password. Email changes are not allowed.
New password
Save changes
Devices
Add/remove linked devices and set nicknames.
Device ID
Nickname (optional)
Add device
✓ Select a device to remove
0a10aced202194944a064eec
Note: removal expects the device record ID (from the database). Registering uses the public device ID.
Linked devices: 0a10aced202194944a064eec
Physician
Choose a physician to link with your account.
Select physician
Assign physician

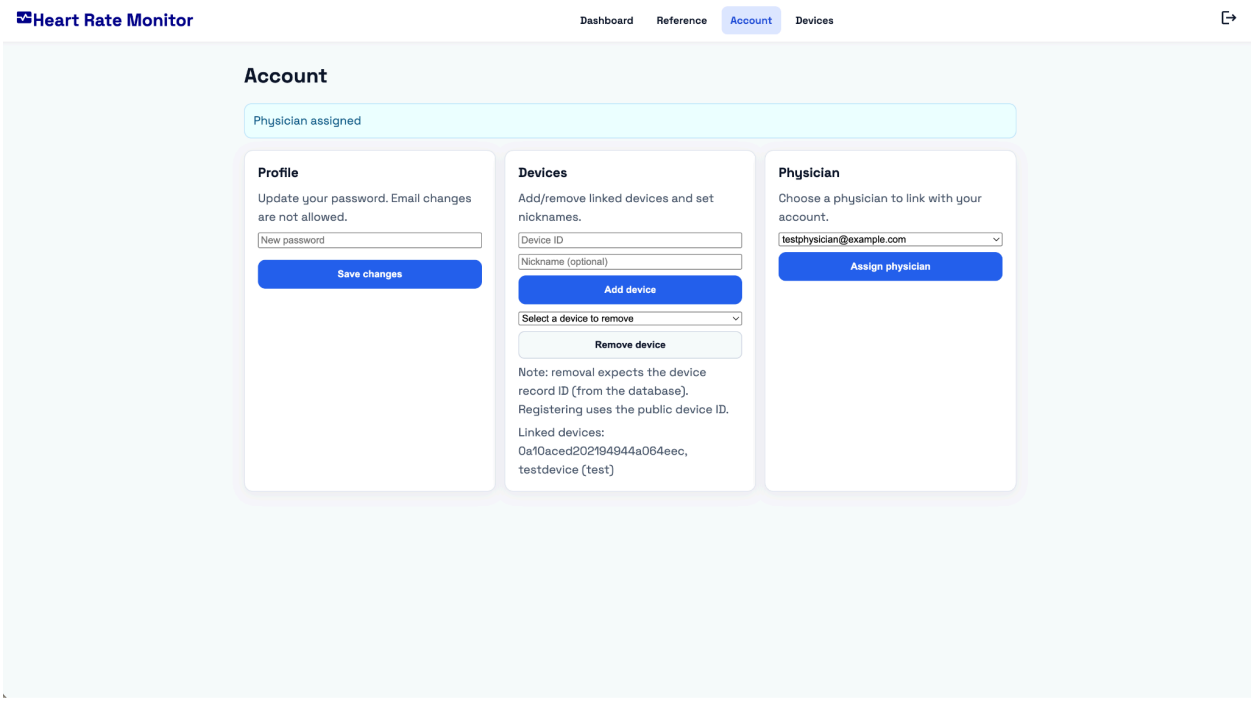
```
_id: ObjectId('693dbfc6db9dc580be50339b')
deviceId : "0a10aced202194944a064eec"
user : ObjectId('69364cb0ff8c94a67406005e')
apiKey : "3cf562803c98eee6f2df540bdc3b45a61e2b93200790ca78dbbfe4c3ce47a38c"
__v : 0
measurementFrequencySeconds : 600
```


4.1.5 Physician management

From the patient account page, a physician can be assigned using a dropdown menu, with a confirmation message displayed after a successful assignment.

Assign a physician:





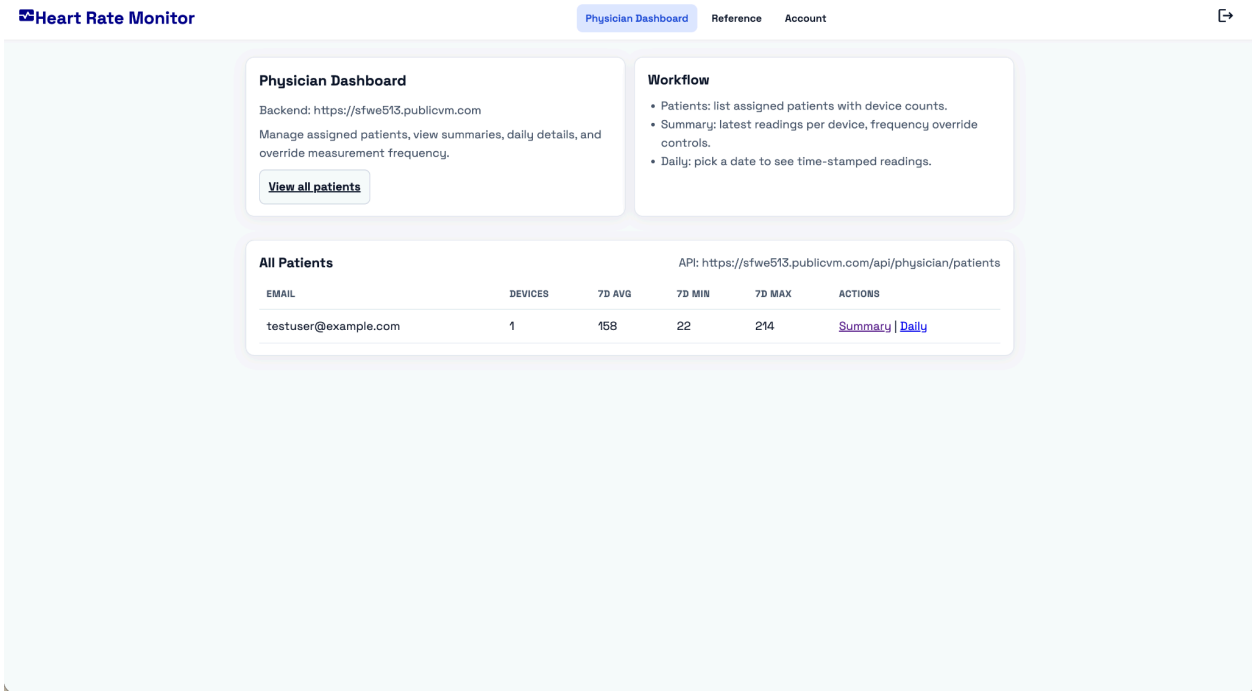
In the database, the assignedPhysician is the ObjectId to the selected physician:

```
_id: ObjectId('69364cb0ff8c94a67406005e')
email : "testuser@example.com"
passwordHash : "$2b$10$/Xq/awL/ijrYWm5ye0dnre.7iqqj7ssDiEJLgyWZy/nXpyYq0yeem"
role : "user"
▸ devices : Array (1)
assignedPhysician : ObjectId('69364ccbff8c94a674060061')
__v : 4
```

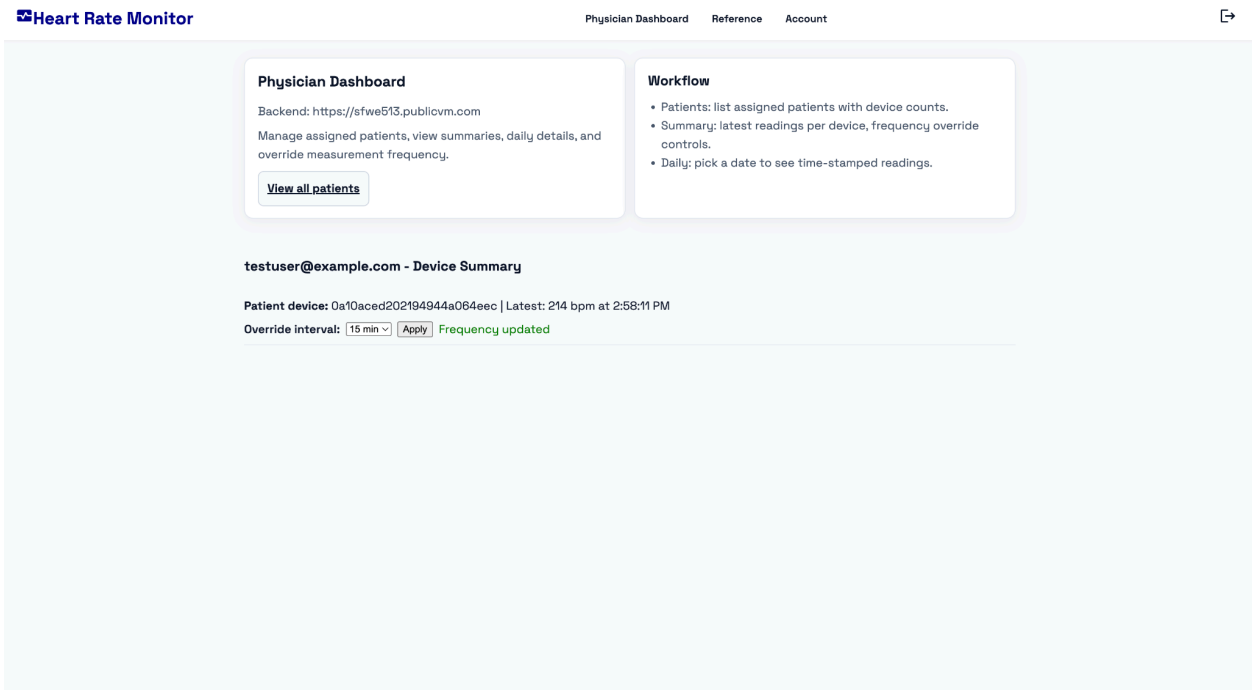
```
_id: ObjectId('69364ccbff8c94a674060061')
email : "testphysician@example.com"
passwordHash : "$2b$10$u29ovhrJq4wzTUWv92gR5uDhYICxqBxq8vEviSN9wrP8slE6vaGYi"
role : "physician"
▸ devices : Array (empty)
assignedPhysician : null
__v : 0
```

The physician portal provides an overview of all assigned patients, including seven-day averages, minimums, and maximums for each patient, along with device counts. Physicians can select an individual patient to view a summary of recent readings, access detailed daily charts, and adjust the device measurement frequency when needed. These views confirm that role-based access and physician-specific functionality are working correctly.

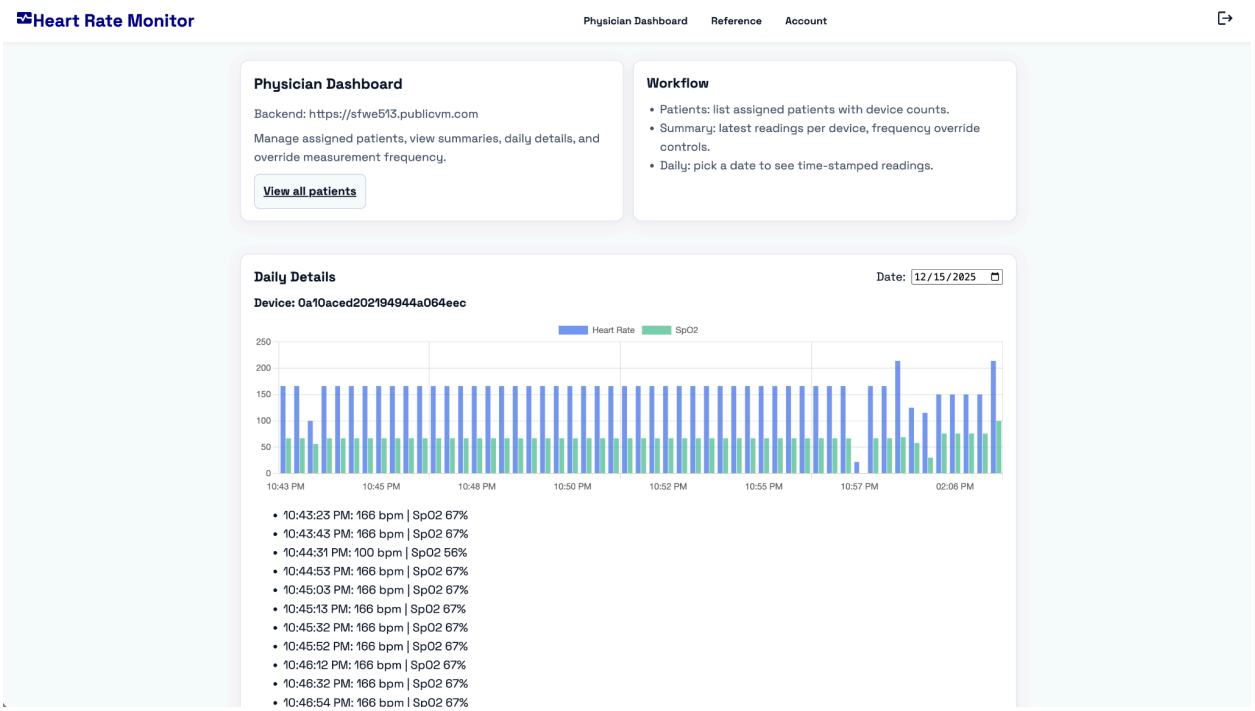
Physician dashboard:

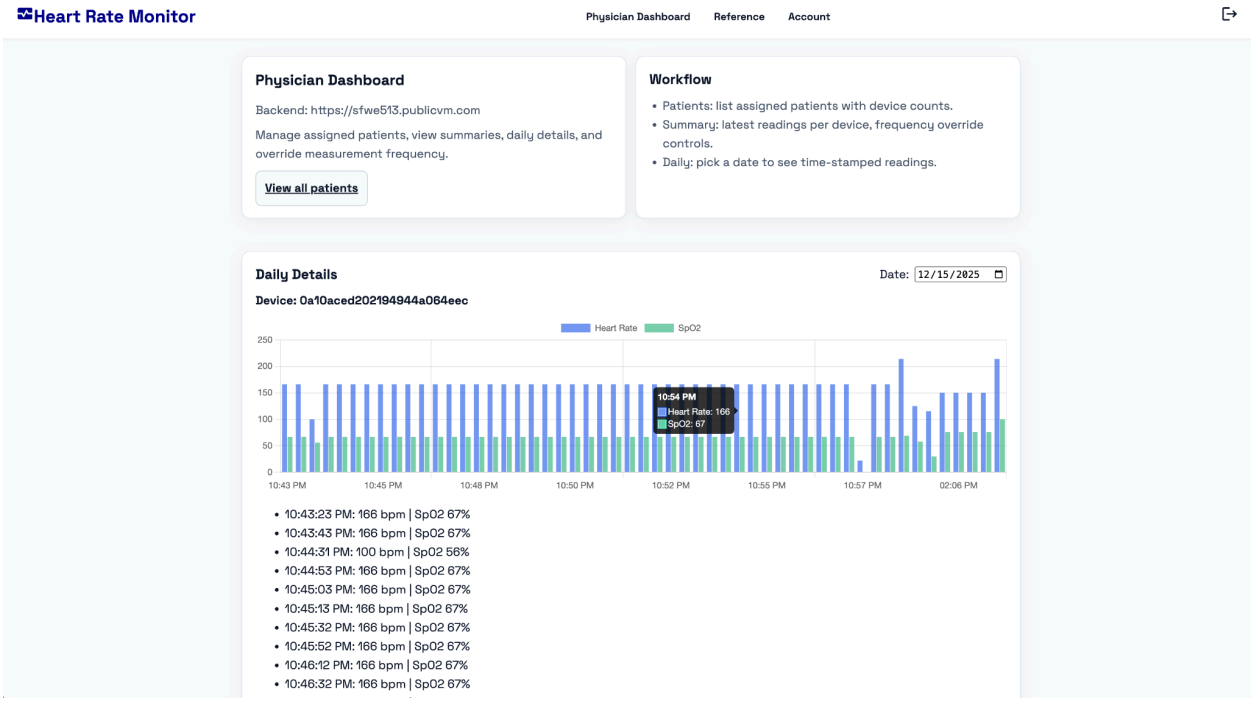


Device Summary and Frequency Adjustment:



Daily view:





4.2 Embedded Device Results

4.2.1 Collected data

Heart rate and SpO_2 readings collected by the device were verified by viewing stored records in the database. Each record includes the device ID, timestamp, heart rate, and

SpO_2 value. The timestamps in the database align with the times reported by the device, confirming that measurements are being transmitted and stored correctly.

Serial monitor of data being read:

```
[MAX30102] IR=420  RED=312  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=2
[MAX30102] IR=34840  RED=12519  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=2

[STATE] STATE_ACQUIRE
[MAX30102] IR=36956  RED=32979  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37114  RED=33254  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37186  RED=33527  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37098  RED=33337  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37067  RED=33317  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37062  RED=33335  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37045  RED=33314  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37058  RED=33353  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37040  RED=33330  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37038  RED=33351  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37073  RED=33391  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37032  RED=33363  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37017  RED=33394  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37060  RED=33436  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37044  RED=33445  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37012  RED=33420  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37019  RED=33438  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37035  RED=33491  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37038  RED=33502  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37012  RED=33459  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=36974  RED=33445  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=36952  RED=33446  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=36952  RED=33450  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=36961  RED=33490  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=36910  RED=33462  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=36904  RED=33478  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=36904  RED=33486  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=36928  RED=33542  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=36926  RED=33540  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=36902  RED=33509  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=36893  RED=33510  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=36885  RED=33515  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=36865  RED=33546  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3

[STATE] STATE_MEASUREMENT_READY

[STATE] STATE_SEND_PENDING
Published measurement event (waiting for server ACK)...

[STATE] STATE_WAIT_SERVER_ACK
Server ACK received (DB recorded).

[STATE] STATE_FLASH_GREEN
```

```
_id: ObjectId('6940ca0eb46a9bb595641b51')
deviceId : "0a10aced202194944a064eec"
heartRate : 214
spo2 : 100
timestamp : 2025-12-16T02:55:10.513+00:00
__v : 0
```

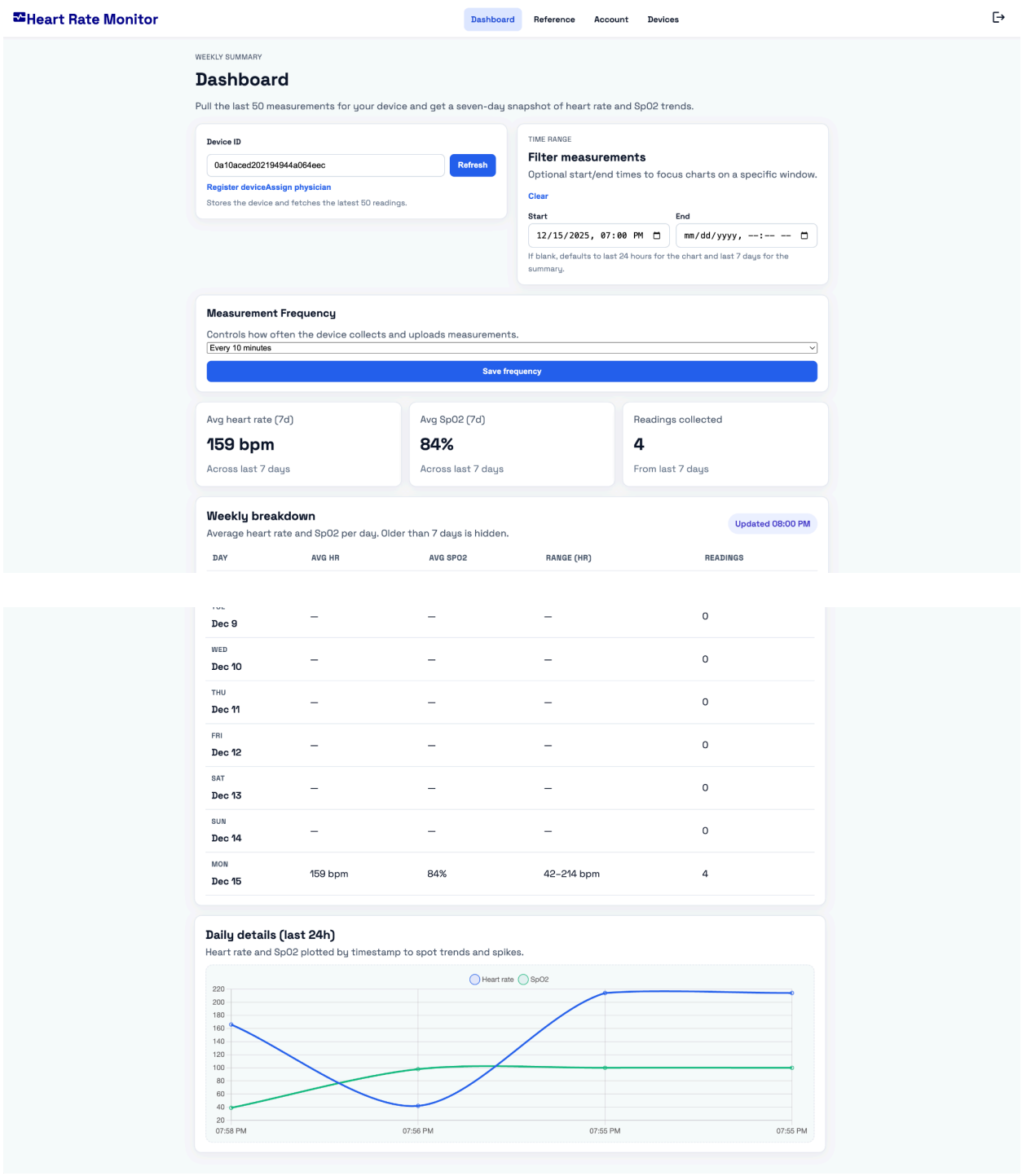
```
_id: ObjectId('6940ca18b46a9bb595641b54')
deviceId : "0a10aced202194944a064eec"
heartRate : 214
spo2 : 100
timestamp : 2025-12-16T02:55:20.517+00:00
__v : 0
```

```
_id: ObjectId('6940ca77b46a9bb595641b58')
deviceId : "0a10aced202194944a064eec"
heartRate : 42
spo2 : 98
timestamp : 2025-12-16T02:56:55.009+00:00
__v : 0
```

```
_id: ObjectId('6940caecb46a9bb595641b5b')
deviceId : "0a10aced202194944a064eec"
heartRate : 166
spo2 : 39
timestamp : 2025-12-16T02:58:52.180+00:00
__v : 0
```


In addition, the same readings shown in the database appear in the frontend dashboards and charts, demonstrating consistency across the system.

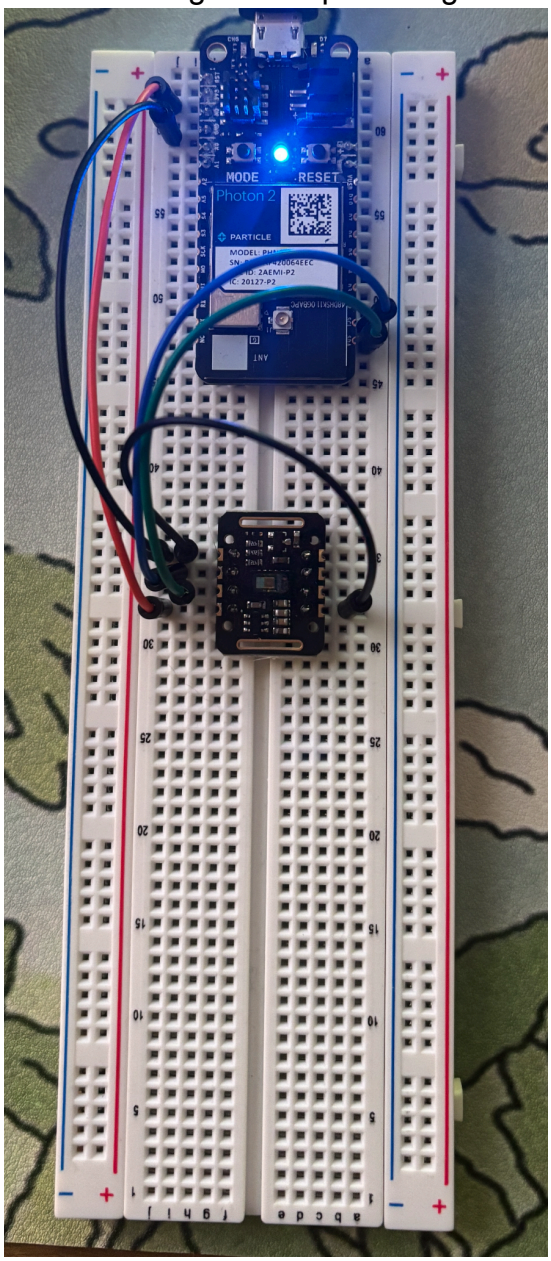
Frontend data:



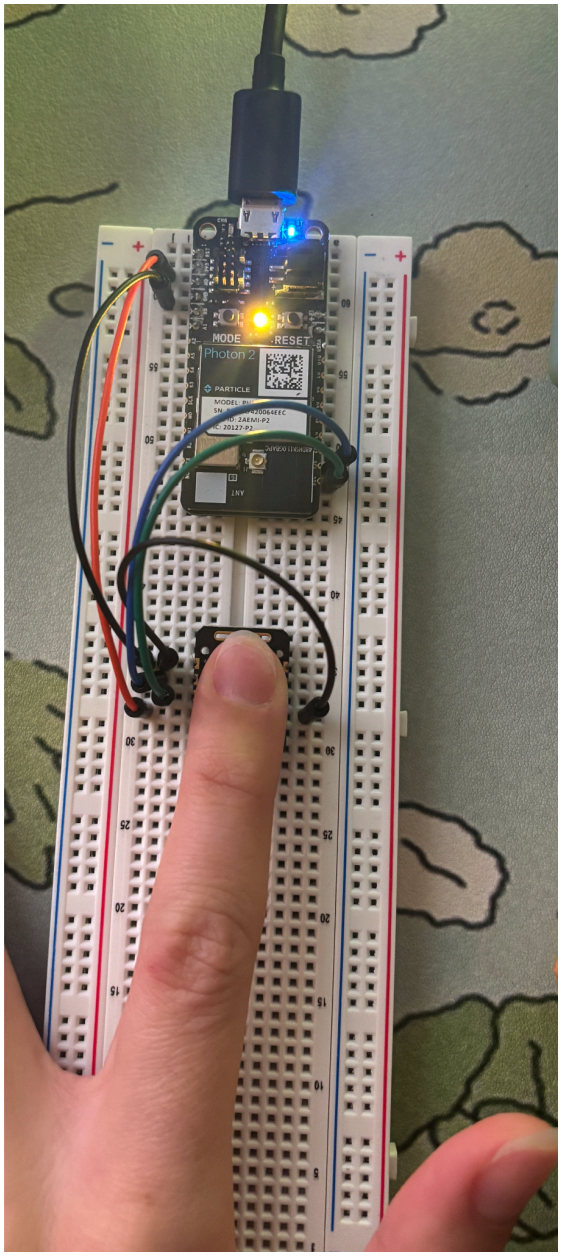
4.2.2 Device in operation

The assembled device consists of a Particle Photon 2 connected to a MAX30102 sensor. LED indicators on the device provide feedback during prompting, measurement, and transmission. Serial monitor output shows the device fetching its configuration, successfully capturing readings, and publishing measurements to the backend.

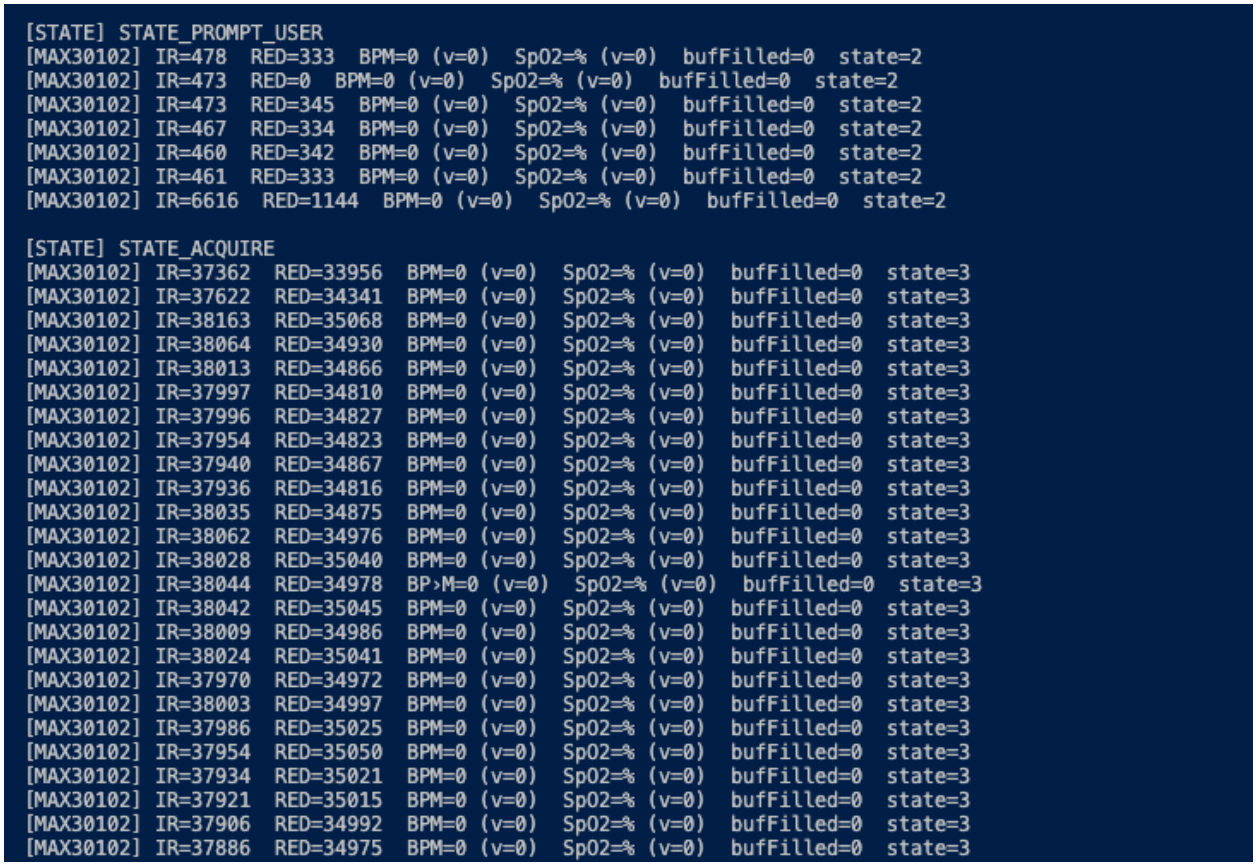
LED indicating user to place finger:



LED indicating data is being collected:



Serial Monitor of data being collected:



Database with published data:

```
_id: ObjectId('6940ca0eb46a9bb595641b51')
deviceId : "0a10aced202194944a064eec"
heartRate : 214
spo2 : 100
timestamp : 2025-12-16T02:55:10.513+00:00
__v : 0
```

```
_id: ObjectId('6940ca18b46a9bb595641b54')
deviceId : "0a10aced202194944a064eec"
heartRate : 214
spo2 : 100
timestamp : 2025-12-16T02:55:20.517+00:00
__v : 0
```

```
_id: ObjectId('6940ca77b46a9bb595641b58')
deviceId : "0a10aced202194944a064eec"
heartRate : 42
spo2 : 98
timestamp : 2025-12-16T02:56:55.009+00:00
__v : 0
```

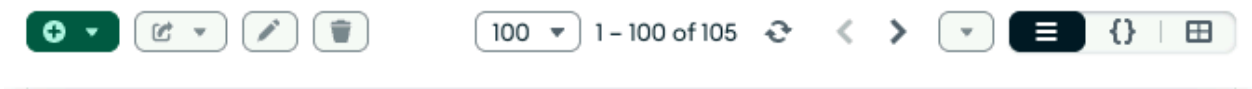
```
_id: ObjectId('6940caecb46a9bb595641b5b')
deviceId : "0a10aced202194944a064eec"
heartRate : 166
spo2 : 39
timestamp : 2025-12-16T02:58:52.180+00:00
__v : 0
```

```
_id: ObjectId('6940cc22b46a9bb595641b62')
deviceId : "0a10aced202194944a064eec"
heartRate : 250
spo2 : 92
timestamp : 2025-12-16T03:04:02.774+00:00
__v : 0
```


4.2.3 Data correspondence

Each POST request sent by the device to the /api/measurements endpoint results in a stored record in MongoDB. The device ID and timestamp shown in the serial logs match the corresponding database entry and the data point displayed in the frontend charts. This confirms end-to-end data flow from the device to the user interface.

Database with 100+ collected data:




```
_id: ObjectId('693fa6703357bddf4a2bab12')
deviceId : "0a10aced202194944a064eec"
heartRate : 115
spo2 : 30
timestamp : 2025-12-15T06:10:56.046+00:00
__v : 0
```

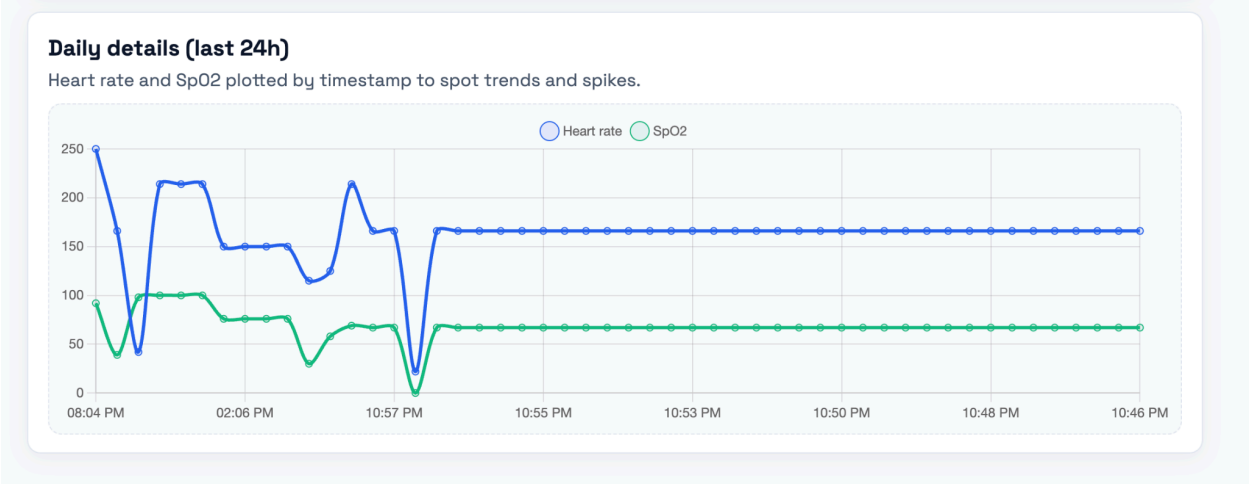
```
_id: ObjectId('6940782b19e58f20a83c25eb')
deviceId : "0a10aced202194944a064eec"
heartRate : 150
spo2 : 76
timestamp : 2025-12-15T21:05:47.712+00:00
__v : 0
```

```
_id: ObjectId('6940783f19e58f20a83c25ee')
deviceId : "0a10aced202194944a064eec"
heartRate : 150
spo2 : 76
timestamp : 2025-12-15T21:06:07.947+00:00
__v : 0
```

```
_id: ObjectId('6940785319e58f20a83c25f1')
deviceId : "0a10aced202194944a064eec"
heartRate : 150
spo2 : 76
timestamp : 2025-12-15T21:06:27.560+00:00
__v : 0
```

```
_id: ObjectId('6940786719e58f20a83c25f4')
deviceId : "0a10aced202194944a064eec"
heartRate : 150
spo2 : 76
timestamp : 2025-12-15T21:06:47.828+00:00
```


Chart from frontend:



4.3 System Workflow Demonstration

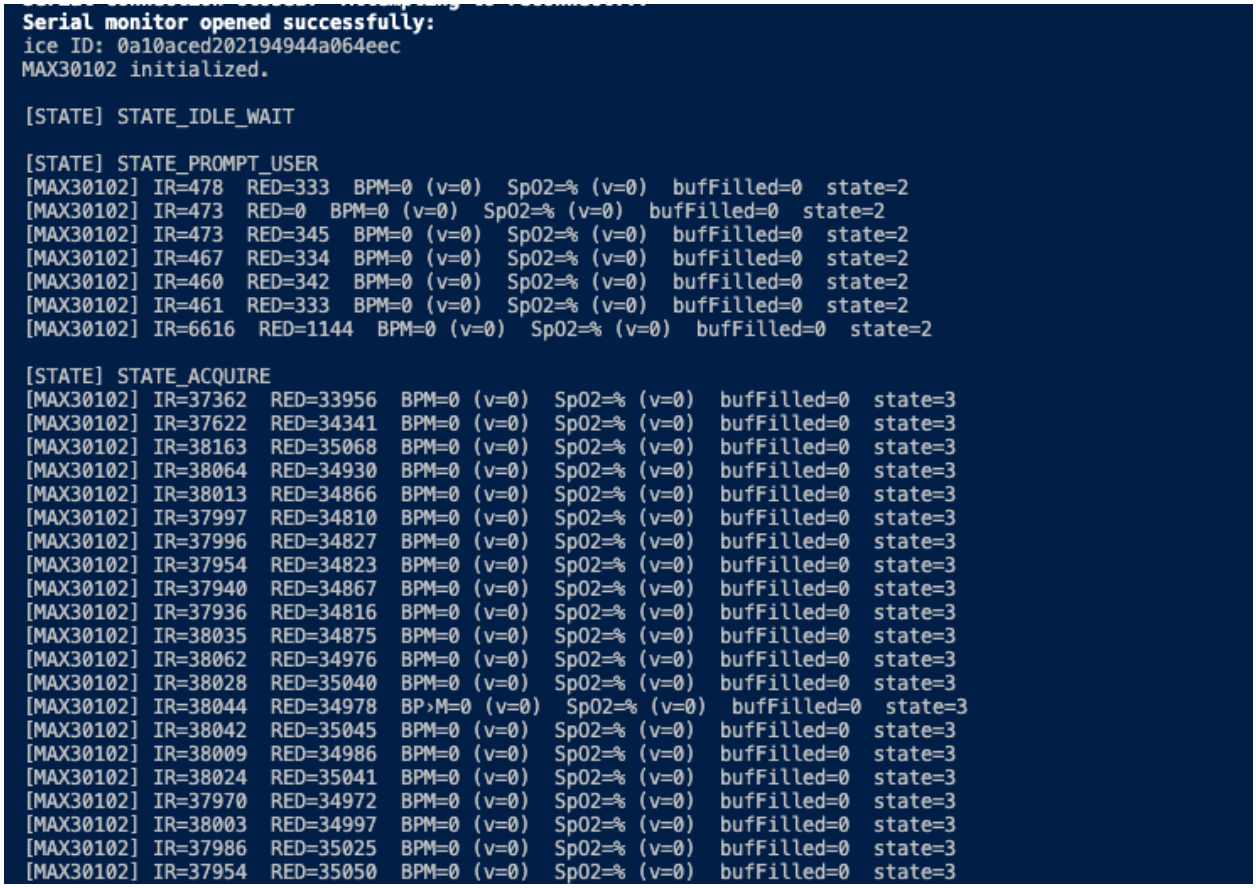
4.3.1 Device boot and configuration

When powered on, the Photon 2 requests its configuration from the backend using its API key. The device receives the current measurement interval and updates its internal timing accordingly. This update is confirmed through serial log messages indicating the new measurement interval.

4.3.2 Measurement capture

The device prompts the user to place their finger on the sensor and samples data from the MAX30102 at approximately 25 Hz. To ensure accuracy, the firmware requires multiple consecutive stable samples before accepting a measurement. The serial logs show clear state transitions as the device moves from idle, to prompting, to acquiring data, and finally to a ready measurement state.

State transition:



Multiple stable samples:

```
[STATE] STATE_PROMPT_USER
[MAX30102] IR=478  RED=333  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=2
[MAX30102] IR=473  RED=0  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=2
[MAX30102] IR=473  RED=345  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=2
[MAX30102] IR=467  RED=334  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=2
[MAX30102] IR=460  RED=342  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=2
[MAX30102] IR=461  RED=333  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=2
[MAX30102] IR=6616  RED=1144  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=2

[STATE] STATE_ACQUIRE
[MAX30102] IR=37362  RED=33956  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37622  RED=34341  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=38163  RED=35068  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=38064  RED=34930  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=38013  RED=34866  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37997  RED=34810  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37996  RED=34827  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37954  RED=34823  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37940  RED=34867  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37936  RED=34816  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=38035  RED=34875  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=38062  RED=34976  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=38028  RED=35040  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=38044  RED=34978  BP›M=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=38042  RED=35045  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=38009  RED=34986  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=38024  RED=35041  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37970  RED=34972  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=38003  RED=34997  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37986  RED=35025  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37954  RED=35050  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37934  RED=35021  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37921  RED=35015  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37906  RED=34992  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37886  RED=34975  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
```

Publish measurements:

```
[STATE] STATE_ACQUIRE
[MAX30102] IR=34217  RED=31420  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34355  RED=31612  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34470  RED=31717  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34457  RED=31742  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34596  RED=31894  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34597  RED=31978  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34757  RED=32192  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34691  RED=32078  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34661  RED=32044  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34650  RED=32042  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34667  RED=32053  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34651  RED=32078  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34619  RED=32077  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34634  RED=32068  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34626  RED=32121  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34664  RED=32173  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34631  RED=32125  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34626  RED=32131  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34606  RED=32143  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34596  RED=32143  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34654  RED=32231  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34680  RED=32249  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34616  RED=32206  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34600  RED=32214  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34585  RED=32204  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34593  RED=32211  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34568  RED=32228  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34570  RED=32226  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34611  RED=32240  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34510  RED=32201  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34466  RED=32143  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34488  RED=32165  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34536  RED=32264  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3

[STATE] STATE_MEASUREMENT_READY

[STATE] STATE_SEND_PENDING
Published measurement event (waiting for server ACK)...

[STATE] STATE_WAIT_SERVER_ACK
Server ACK received (DB recorded).

[STATE] STATE_FLASH_GREEN

[STATE] STATE_IDLE_WAIT
```

4.3.3 Sending data to the backend

Once a valid reading is captured, the device sends the measurement to the backend as a JSON payload containing the device ID, heart rate, SpO_2, and timestamp. The backend validates the API key, stores the data in MongoDB, and returns a success response. The device logs confirm receipt of the server acknowledgment.

Payload sent to backend with measurement:

```
samanthaperry@Samanthas-MacBook-Pro photon % particle subscribe hook-response/Photon2_SendEven
t
Subscribing to "hook-response/Photon2_SendEvent" from my devices

{"data":"{\"deviceId\":\"0a10aced202194944a064eec\",\"heartRate\":115,\"spo2\":45,\"_id\":\"69
40cf4fb46a9bb595641b66\",\"timestamp\":\"2025-12-16T03:17:35.300Z\",\"__v\":0}","ttl":60,"publ
ished_at":"2025-12-16T03:17:35.322Z","coreid":"particle-internal","name":"hook-response/Photon
2_SendEvent/0"}
```

```cpp
bool publishMeasurement(const MeasurementRecord &rec) {
    // Build JSON payload
        String payload = String::format(
        "{"
            "\"deviceId\":\"%s\","
            "\"heartRate\":%d,"
            "\"spo2\":%d,"
            "\"timestamp\":%lu"
        "}",
        deviceId.c_str(),
        (int)rec.heartRate,
        (int)rec.spo2,
        (unsigned long)rec.timestamp
    );

    return Particle.publish(MEAS_EVENT, payload, PRIVATE);
}
```

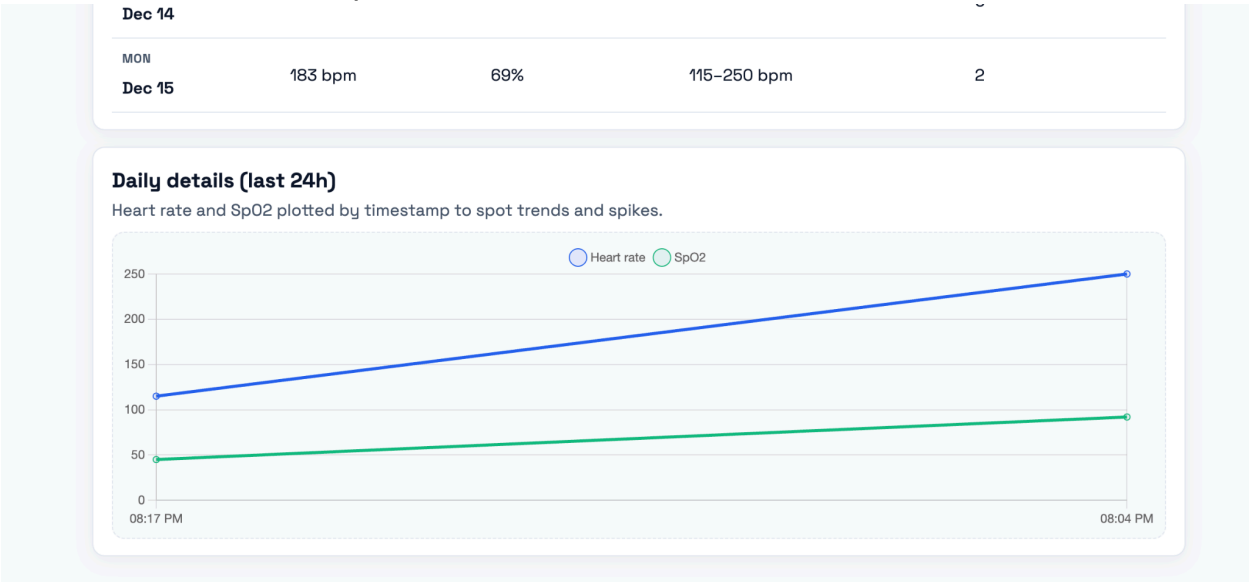
Device logs with server acknowledgement:

```
[STATE] STATE_ACQUIRE
[MAX30102] IR=34217  RED=31420  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34355  RED=31612  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34470  RED=31717  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34457  RED=31742  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34596  RED=31894  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34597  RED=31978  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34757  RED=32192  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34691  RED=32078  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34661  RED=32044  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34650  RED=32042  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34667  RED=32053  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34651  RED=32078  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34619  RED=32077  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34634  RED=32068  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34626  RED=32121  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34664  RED=32173  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34631  RED=32125  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34626  RED=32131  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34606  RED=32143  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34596  RED=32143  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34654  RED=32231  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34680  RED=32249  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34616  RED=32206  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34600  RED=32214  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34585  RED=32204  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34593  RED=32211  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34568  RED=32228  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34570  RED=32226  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34611  RED=32240  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34510  RED=32201  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34466  RED=32143  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34488  RED=32165  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3
[MAX30102] IR=34536  RED=32264  BPM=187 (v=0)  SpO2=81% (v=0)  bufFilled=0  state=3

[STATE] STATE_MEASUREMENT_READY

[STATE] STATE_SEND_PENDING
Published measurement event (waiting for server ACK)...

[STATE] STATE_WAIT_SERVER_ACK
Server ACK received (DB recorded).

[STATE] STATE_FLASH_GREEN

[STATE] STATE_IDLE_WAIT
```

4.3.4 Frontend consumption

Both patient and physician views retrieve measurement data through secured REST endpoints using JWT authentication. The dashboards update automatically to reflect new readings, and the time of the posted measurement matches the data point shown in the daily charts.

After data collected from previous section:



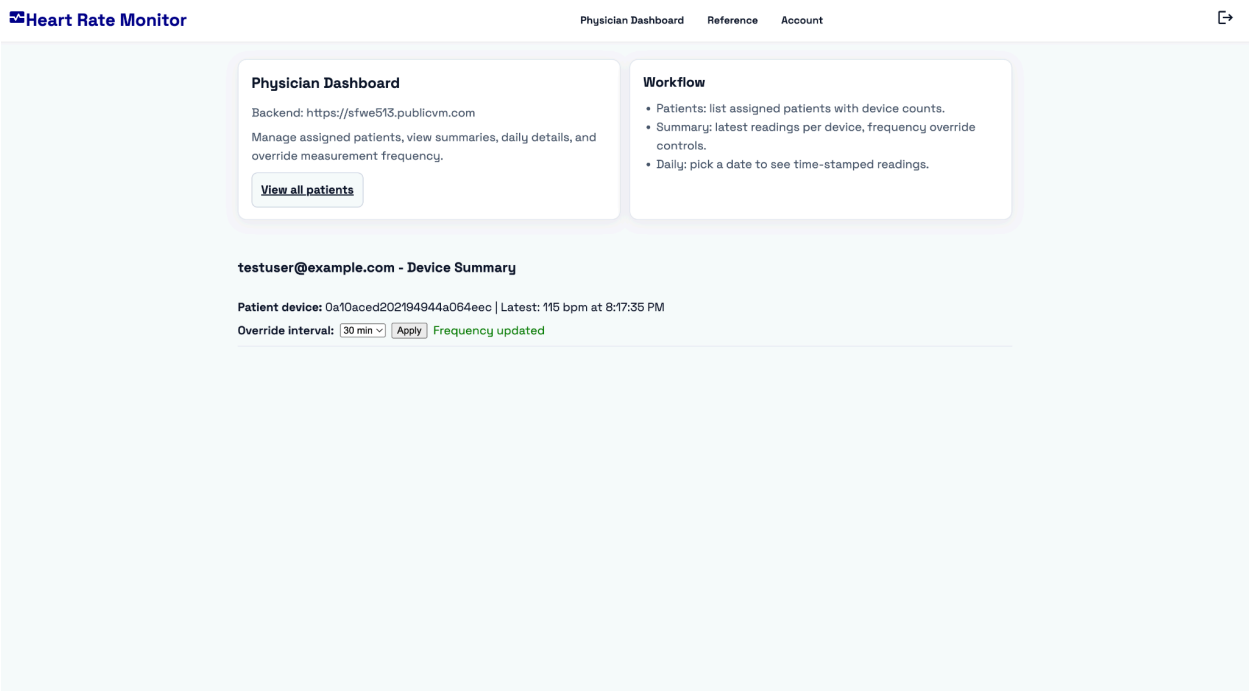
4.3.5 Physician override

Physicians can adjust how often a device takes measurements through the frontend interface. When a new frequency is set, the backend stores the updated value and provides it to the device on the next configuration fetch. The device then logs the updated interval and begins using the new measurement cadence.

Original frequency in database:

```
_id: ObjectId('6940c488b46a9bb595641b16')
deviceId : "0a10aced202194944a064eec"
nickname : "test2"
user : ObjectId('69364cb0ff8c94a67406005e')
apiKey : "59613fa10ce3e7d9db64a668216a1aaf306939fc08e29baf819b6165ac273e26"
measurementFrequencySeconds : 600
__v : 0
```

Physician changes frequency:



Frequency changes in the backend:

```
_id: ObjectId('6940c488b46a9bb595641b16')
deviceId : "0a10aced202194944a064eec"
nickname : "test2"
user : ObjectId('69364cb0ff8c94a67406005e')
apiKey : "59613fa10ce3e7d9db64a668216a1aaf306939fc08e29baf819b6165ac273e26"
measurementFrequencySeconds : 1800
__v : 0
```

Device recognizing frequency update and sending GET request to retrieve new frequency:

```
[MAX30102] IR=37864  RED=34782  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=38042  RED=34812  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=39223  RED=36102  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=39627  RED=36643  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=38861  RED=35574  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=39171  RED=36273  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=39052  RED=35577  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=38281  RED=34378  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=38045  RED=34080  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37706  RED=33765  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3

[STATE] STATE_MEASUREMENT_READY

[STATE] STATE_SEND_PENDING
Published measurement event (waiting for server ACK)...

[STATE] STATE_WAIT_SERVER_ACK
Server ACK received (DB recorded).

[STATE] STATE_FLASH_GREEN

[STATE] STATE_IDLE_WAIT
Config response received: {"measurementFrequencySeconds":1800}
Updated measurement interval: 1800 sec
```

4.3.6 Offline handling

If the device temporarily loses connectivity, measurements are stored locally instead of being discarded. Once the connection is restored, the queued readings are sent to the backend and appear in both the database and the frontend dashboards. Serial logs confirm when data is stored locally and when it is successfully flushed.

Data collection when WiFi is turned off, then turned back on:

```
[MAX30102] IR=36929  RED=33519  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=36954  RED=33619  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=36931  RED=33648  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=36967  RED=33638  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=39529  RED=33117  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=40257  RED=37503  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=40074  RED=37204  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=38751  RED=35903  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37219  RED=34508  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37257  RED=34233  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=37235  RED=34472  BPM=0 (v=0)  SpO2=% (v=0)  bufFilled=0  state=3
[MAX30102] IR=38967  RED=35485  BPM=150 (v=1)  SpO2=21% (v=1)  bufFilled=1  state=3

[STATE] STATE_MEASUREMENT_READY

[STATE] STATE_STORE_OFFLINE
Stored measurement offline in EEPROM queue.

[STATE] STATE_FLASH_YELLOW

[STATE] STATE_IDLE_WAIT

[STATE] STATE_FLUSH_BACKLOG

[STATE] STATE_SEND_PENDING
Published measurement event (waiting for server ACK)...

[STATE] STATE_WAIT_SERVER_ACK
HOOK ERROR event=hook-error/Photon2_SendEvent/0 data=queryAaaa ETIMEOUT sfwe513.publicvm.com
Server returned hook-error.

[STATE] STATE_IDLE_WAIT

[STATE] STATE_FLUSH_BACKLOG

[STATE] STATE_SEND_PENDING
Published measurement event (waiting for server ACK)...

[STATE] STATE_WAIT_SERVER_ACK
Server ACK received (DB recorded).

[STATE] STATE_FLASH_GREEN

[STATE] STATE_IDLE_WAIT
```

5. Lessons Learned

- We learned that a small mismatch in REACT_APP_API_BASE (HTTP vs HTTPS, localhost vs domain) leads to mixed-content blocks and failed API calls; rebuilding with the correct base and redeploying the static build fixed the issue.

- We learned how important it is to set up gitignore files super early on to avoid pushing large amounts of unnecessary data especially when it comes ot photon files. We made the mistake of pushing some of these files and had to revert back.

- We learned that having a domain name was essential. It allowed us to obtain valid TLS certificates for the web app and gave the Photon a stable host to call without IP changes. Using the domain mapped to the Elastic IP meant we could terminate HTTPS for browsers while keeping port 80 open for the device, avoiding hardcoded IPs and certificate mismatches.

- We learned that serving a React SPA behind Nginx requires proper proxy and try_files setup; incomplete server blocks or Certbot edits led to TLS/proxy errors until we replaced them with full 80/443 configs.

- We learned that PM2 + systemd for the backend and Nginx for static assets is a simple, reliable deployment model on EC2. It keeps services running after logout/reboot.

6. Challenges and Resolutions

- Challenge 1: Mixed-content errors when the HTTPS page called HTTP APIs. Resolution: Hardcoded the API base to https://sfwe513.publicvm.com and rebuilt the frontend.

- Challenge 2: Certbot left partial Nginx blocks causing ERR_SSL_PROTOCOL_ERROR. Resolution: Replaced with complete 80/443 server blocks including root/proxy and cert paths, then reloaded Nginx.

- Challenge 3: Backend stopping after SSH logout. Resolution: Ran PM2 with pm2 startup and pm2 save to auto-start and persist the Node process.

- Challenge 4: Photon needed a stable host and per-device API key. Resolution: Set API_HOST to the domain after Nginx was set up, kept port 80 open, added /api/device/config/:deviceId for frequency, and enforced apiKey/device checks in ingestion routes.

- Challenge 5: Photon attempted to request configuration before establishing a

cloud connection, resulting in missed webhook triggers. Resolution: Added a cloud-connection check and delayed configuration requests until Particle.connected() returned true.

7. Team Contributions

Team Member	Frontend (%)	Backend (%)	Embedded Device (%)	Documentation (%)	Demos/Testing (%)
Amber	45	10	5	25	15
Seven	5	10	55	15	15
Samantha	10	45	10	15	20

8. References

8.1 Framework Documentation

- React Framework

8.2 Libraries

- heartRate.cpp
- MAX30105.cpp
- spo2_algotithm.cpp
- React – Frontend JavaScript library for building user interfaces
- React DOM – DOM rendering for React applications
- React Router DOM – Client-side routing for React
- React Scripts – Build and development tooling for React applications
- TypeScript – Typed superset of JavaScript used across the project
- Express.js – Backend web framework for building REST APIs
- Mongoose – Object Data Modeling (ODM) library for MongoDB
- MongoDB – NoSQL database used for storing user and device data
- CORS – Middleware for enabling cross-origin requests
  bcrypt – Library for hashing and securing user passwords
- jwt-simple – Lightweight JSON Web Token (JWT) authentication library
- Material UI (MUI) – React component library for UI design
- MUI Icons – Icon set for Material UI components
- Emotion – CSS-in-JS styling library used with Material UI

- Chart.js – Library for rendering charts and visualizing heart-rate data
- Testing Library (React, DOM, User Event) – Tools for testing React components and user interactions
- Web Vitals – Library for measuring frontend performance metrics

8.3 Datasheets

- Particle Photon 2 Datasheet
- MAX30102 Sensor Datasheet