ECE 413/513 Project – Heart Rate Monitoring

Final Project Due: Monday, December 16, 11:59 PM

Last day to submit (Late): Wednesday, December 18, 11:59 PM for 10 points deduction

Content

Heart Rate Monitoring Project Overview

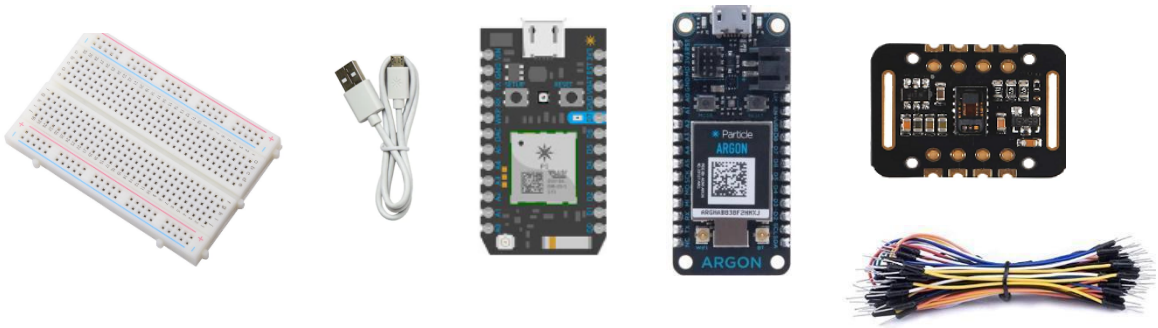
- The Heart Track application is a low-cost IoT-enabled web app for monitoring heart rate and blood oxygen saturation level throughout the day at a user (or physician-defined, for ECE 513) rate.
- The IoT device uses a heart rate and oxygen sensor and will periodically remind users to take heart rate and blood oxygen saturation level measurements.
- The IoT device will transmit measurements to a web application through which users (or physicians, for ECE 513) can view their data.
- The time of day and rate at which measurement should be taken will be configurable by the user (or physician, for ECE 513)
- The Heart Track web application allows the user to monitor their heart rate and blood oxygen saturation level.
- The web application uses a responsive design to allow users to view the application seamlessly on desktop, tablet, and mobile devices.

Requirements and Clarifications

- Some of the project requirements are open-ended, allowing individual groups to be creative in how the requirement is implemented. Intentionally open-ended requirements are underlined below.
- If any requirement is not clear, you should send an email to the TA to clarify the requirements. Any clarifications needed will be added to this document and marked as Clarification.

Hardware Requirement

- IoT development board
  - Photon: Link or Argon: Link
  - Heart Rate Sensor (Module MAX30102 Pulse Detection Blood Oxygen) Link
  - Micro USB cable for data transfer: Link
  - Mini breadboard: Link
  - Jumper wires: Link



Requirements (Both ECE 413 and 513)

- **System Requirements Overview:**
  - The web application should have a navigation menu.
  - The web application should use a responsive design to be viewable on desktops, tablets, and smartphones.
  - The web application should have an index.html page to introduce your team and project.
  - The web application should have the reference.html page to list your third-party APIs, libraries, and code.

- **Account Creation and Management:**
  - A user must be able to create an account, using an email address as the username and a strong password, and register at least one device with their account.
  - A user should be able to update any of their account information, except their email.
  - A user should be able to add and remove devices in their account.
  - A user should be able to have more than one device.

- **Server:**
  - Your server ***must*** be implemented using Node.js, Express, and MongoDB.
  - Your server's endpoints must use RESTful APIs. Each endpoint must have documentation that describes the behavior, the expected parameters, and responses.
  - Recommended HTTP response code (https://developer.mozilla.org/en-US/docs/Web/HTTP/Status)
    - 200: OK (request succeeded), 201: Created, 400: bad request, 401: Unauthorized, 404: Not found, 500: Internal server error
  - Access to the web application should be controlled using token-based authentication.

- **Web Application:**
  - Login/logout interface
  - The web application supports a weekly summary view and a detailed daily view.
  - The ***weekly summary view*** will show the user's average, minimum, and maximum heart rate in the past 7 days.
  - The ***detailed daily view*** (for a selected day) will plot heart rate and blood oxygen saturation level readings for the selected day on separate charts.
    - The horizontal axis for the charts should be the time of day, and the vertical axis should be the measurement.
    - The minimum and maximum values should be visually indicated on the chart.
    - Third-party plotting libraries can be used for the plots.
    - The web application should allow the user to define the time-of-day range and frequency at which measurements will be requested.

- **Heart Track IoT Device:**
  - The IoT device will periodically ask the user to take a pulse and blood oxygen saturation level reading by flashing the RGB LED on the device blue.
  - The LED will flash until the user takes a measurement or 5 minutes without a measurement has elapsed.
  - Once a measurement is taken:
    - If the device is connected to Wi-Fi, the measurement should be immediately recorded, and the RGB LED should briefly flash green once the server confirms the data was recorded in the DB.
    - If the device is not connected to Wi-Fi, the IoT device should briefly flash the RGB LED yellow, and locally store the data for up to 24 hours, and submit the data when later connected.
  - The time-of-day range of measurements should be configurable by the user through the web application. The IoT will only ask the user for measurements during this time period. By default, measurement should be requested between 6:00 AM and 10:00 PM.
  - The frequency of measurements should be configurable by the user through the web

application. By default, measurements should be requested every 30 minutes.
  - The heart rate sensor can take a while before the measurement stabilizes (i.e., initial measurements may be inaccurate). You must implement an approach/algorithm that seeks to maximize the accuracy of the reported measurements.
  - The server should require an API KEY from the IoT device for posting data.
  - The software for the IoT device must be implemented using synchronous state machines.

- **Third-Party APIs, Libraries, and Code:**
  - You may use additional third-party APIs.
  - You may use open-source JavaScript libraries.
  - You may use open-source CSS libraries.
  - You may use code from any examples presented in the course
  - You should prepare a reference page (i.e., references.html) to list your third-party APIs, Libraries, and Code

- **Index.html:**
  - Introduce your team and your project
  - You may use each student's name, email, and a photo
  - You may present your project based on text, images, videos, or others
  - You should use responsive design

- **Use of LLMs (Large Language Models) for secure implementation (+5pts Extra Credit)**
  - Requirement for 413 vs 513:
    - For students in 413, you have to identify 1 CWE and mitigate it.
    - For students in 513, you have to identify 2 CWEs and mitigate them.
  - You can use LLMs (ChatGPT, CodeLLaMa, Github Co-Pilot, etc.) that suit your project stack and team's familiarity, to detect and mitigate the CWEs within your HTML, CSS, and JavaScript codebase.
  - You can refer to the below websites for more information:
    - Detailed description of vulnerabilities (CVEs): https://nvd.nist.gov/vuln
    - Detailed description of weaknesses (CWEs): https://cwe.mitre.org/
  - Identify weaknesses, such as code injection, that have a high severity score according to the CVEs in your project, find the CWEs related to them, and mitigate them (Please refer to **Appendix A** at the end of this document).
  - In your project report, create a table with columns as below to identify which weaknesses and vulnerabilities were detected and mitigated:

CWE-ID (Web Link)	**Description**	**Domain (HTML/CSS/JS/Firmware)**
□ Detected; □ Mitigated;		
• Please provide the code snippet including the vulnerability and explain your approach for mitigating the weakness. • If you are using LLM to do it, please provide the prompt and explain how you make LLM work.		

Additional Requirements (for ECE 513 students only)

- The server must use **HTTPS**.
- The web application should support a separate physician portal that:
  - Allows a physician to register an account using a separate registration page
  - Allows users, on their account information page, to select one of the registered physicians as their physician.
  - The physician portal has a patient view, a patient summary view, and a patient's detailed daily view.
  - The *all-patient view* will list all patients by name with their 7-day average, maximum, and minimum heart rate.
  - The physician will be able to select a patient to view the patient's summary view and the patient's detailed view.
  - The *patient's summary view* is similar to the weekly summary view for a user but also includes controls that allow the physician to adjust the frequency of measurement.
  - The *patient's detailed day view* will present the same information as the detailed day view for the user.

Due Dates and Submission Requirements

Final Project Submission, Pitch, and Demo (Due: December 16, 11:59 PM)

- The project must be available through a Git repo (You can use GitHub or GitLab)
- You need to clean this repo without the module you installed in the project. It just needs to contain the code that you develop along with a well-described "Readme.md" file written in Markdown.
- Tutorials:
  - GitHub Tutorial - Beginner's Training Guide: https://www.youtube.com/watch?v=iv8rSLsi1xo
  - Using the GitHub Web-Based Editor: https://www.youtube.com/watch?v=d7jHUh1PGwU
  - Guide to using Markdown: https://www.markdownguide.org/basic-syntax/
  - Git Repo Template: https://gitlab.com/yuzhenglin/ece413_513_project_template
- All code for the project, including the device, server, front-end web pages, README, etc., must be submitted as a single archive to the D2L assignment drop box.
- The README should include:
  - Links to your server, the pitch, and demonstration videos.
  - Login credentials for an existing user account with recently collected data
  - **ECE 513 only**: Login credentials for a physician account
  - Create a 5-minute video that pitches the Heart Track project to potential investors (e.g., angel investors, VCs, Kickstarter, etc. See https://kickstarterguide.com/2012/06/13/examples-of-great-pitch-videos/ for examples of good pitch videos.)
- Create a **15-20-minute video (20 minutes is maximum length)** demonstrating your project implementation. This video should demonstrate both the user experience and discuss the actual implementation. The suggestion is to split the video into 7-10 minutes for the user experience and 7-10 minutes to discuss your code implementations. This video will be used during the grading of your project and should mention how you addressed each requirement.
- The pitch video and demonstration video should either be hosted on your server or hosted on some other publicly accessible service (e.g., YouTube)
- The project is due **Monday, December 16 by 11:59 pm**, the D2L assignment drop box will remain open until **Monday, December 18 by 11:59 pm**. **Your server must be accessible until**

at least December 20.

- **Submit on or before December 18 by 11:59 pm for 10 points deduction**
- **No submission will be accepted after December 18 by 11:59 pm**

Project Milestone and Demo (Due: November 22 by 11:59 PM)

- Set up a web server that allows users to create accounts, register a Heart Track device, and view basic data collected from the device.
- Implement code for the IoT device that
  - Takes basic measurements for heart rate and blood oxygen saturation.
  - Minimally uses a fixed schedule to ask the user every 30 minutes to take a measurement.
- Submit a **5-minute video** demonstrating your IoT device asks the user to take a measurement, takes pulse and blood oxygen saturation measurements, transmits that data from the device to your web server, stores that data in the database, and has a front-end webpage for viewing that data.
- All code for the Project Milestone progress, including the device, server, front-end web pages, README, etc., must be submitted as a single archive to the D2L assignment.
- **Submit on or before 11/22 by 11:59 pm for 3 points extra credit**

Project Documentation (out of 5):

- A brief grading rubric for the project documentation is listed below. The project documentation must act as a user's manual. In particular, a full report describing what has been done is required including the main items below:
  - A project description section covering the detailed narrative of how each component of the project was implemented (including subsections describing backend, frontend, and embedded device implementations). **(2 points)**
    - For those who do the secure implementation, provide a section describing the secure code using LLMs, a table similar to the template shown above, and a description of how each weakness was identified and addressed in your code.
  - A file description section to elaborate what each file entails using a few full sentences/short paragraphs for each file (including subsections for each of the public, route, model, JavaScript, etc. files). **(1 point)**
  - A results section with screenshots of your website and various functionalities as well as diagrams and tables showing your results captured by your embedded device (for example, screenshots for each HTML component and describing each functionality implemented, screenshots demonstrating how the patient can add/remove devices and visualize the results (for 513 demonstrate how a patient can add/remove physician), etc.). **(0.5point)**
  - A lessons learned section describing the main lessons learned from implementing this project (at least 5 bullet points describing in full sentences what you have learned). **(0.5point)**
  - A challenges section including the challenges you faced during the implementation of each component and how you resolved those issues (at least 5 bullet points describing in full sentences what challenges you faced and how you resolved them). **(0.5point)**
  - A table showing each team member's contribution (in percentage out of 100) to each component of the project (for example, a table with three rows for the three team members and three columns for the three components of frontend, backend, embedded device implementation, documentation, demos, etc.). **(0.25point)**
  - A reference section where you properly cite any resources used. **(0.25point)**

Grading Rubric (out of 35):

- The grading rubric for all submissions is provided in the table below.
- To evaluate your implementation, we will check your video demo, code, and other documents

	Item	Point(s)		Brief Description
		ECE413	ECE513	
1	AWS running	1	1	Server-side successfully running.
2	Index.html	1	1	Main page
3	Information about project	1	1	Basic information about the project
4	Team information	1	1	Picture, email, other info.
5	Sign in/ Sign up	2	1	Interface to sign in/sign up
6	Strong password	3	2	Salted Hash
7	Device registration	1	1	Able to register a device
8	Reading Data	1	1	Sensor data
9	Periodic reading every 30 mins	2	2	Using a state machine
10	README file	2	2	Describing how to run the project from scratch. The documentation for the endpoint can be included in the README. Links to your server, the pitch, and demonstration videos. Login credentials for an existing user account with recently collected data.
11	Git Repo	2	2	The entire project must be uploaded to a Git Repo, which can be Gitlab or Github, and includes a well-described file “Readme.md” written in Markdown. The readme requirement is the same as above. Also, include ECE 513 only: Login credentials for a physician account information. Refer to the submission guideline.
12	Video pitch submission	3	3	Demo pitch
13	Coding style	2	2	Comment code (LLM use allowed)
14	Responsive pages	1	1	Support mobile devices
15	Video demo submission	1	1	Demo project correct functionality
16	Code submission	1	1	Frontend, Backend, and Particle codes

17	Store data in the device	3	2	Local storage of data up to 24 hours if server is unreachable or WiFi connection is unsuccessful
18	Charts	1	1	Visualization of the data on the client side.
19	Localhost running	1	1	Evidence of code running on your PC in demo.
20	HTTPS implementation	-	3	The server must use HTTPS.
21	Project Documentation	5	5	As elaborated in project documentation guide.
	Total Points	**35**	**35**	**Submission deadline 12/16 11:59pm**
22	Extra Credit (Secure code using LLMs)	5	5	Refer to the description provided regarding using LLMs to identify vulnerabilities (CVEs and CWEs) in your HTML, CSS, JS, and firmware.
23	Extra credit (Milestone)	3	3	Milestone submission by 11/22 11:59pm
24	Penalty (Late submission)	-10	-10	Late submission by 12/18 11:59pm

Appendix A – Common Weaknesses Enumeration Example List

CWE-ID	CWE Description	Domain
CWE-352: Cross-Site Request Forgery (CSRF)	When a web server is designed to receive a request from a client without any mechanism for verifying that it was intentionally sent, then it might be possible for an attacker to trick a client into making an unintentional request to the web server which will be treated as an authentic request. This can be done via a URL, image load, XMLHttpRequest, etc. and can result in exposure of data or unintended code execution.	HTML, JS
CWE-232: Improper Handling of Undefined Values	The product does not handle or incorrectly handles when a value is not defined or supported for the associated parameter, field, or argument name.	JS
CWE-20: Improper Input Validation	Input validation is a frequently-used technique for checking potentially dangerous inputs in order to ensure that the inputs are safe for processing within the code, or when communicating with other components. When software does not validate input properly, an attacker is able to craft the input in a form that is not expected by the rest of the application. This will lead to parts of the system receiving unintended input, which may result in altered control flow, arbitrary control of a resource, or arbitrary code execution.	JS, Firmware
CWE-311: Missing Encryption of Sensitive Data	The lack of proper data encryption passes up the guarantees of confidentiality, integrity, and accountability that properly implemented encryption conveys.	JS, Firmware

CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting' - XSS)	Cross-site scripting (XSS) involves injecting malicious scripts into web applications, leading to unauthorized execution of code in victims' browsers, potentially compromising their data and actions, with three main types: Reflected, Stored, and DOM-Based XSS. The product does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other users.	HTML, CSS
CWE-94: Improper Control of Generation of Code ('Code Injection')	When a product allows a user's input to contain code syntax, it might be possible for an attacker to craft the code in such a way that it will alter the intended control flow of the product. Such an alteration could lead to arbitrary code execution.	JS, Firmware