BIA Business and Information Systems Architecture MGMT-6134-(31)-25F Capstone Project

HEARTLINK: Heart rate Tracking and Analyst

Milestone 2

Professor Vinnie Moraes

Name – Student ID	**Email**
Guan, Ivan – 1289006 (Captain)	y_guan251701@fanshaweonline.ca
Diden, Gbubemi - 1205960	g_diden@fanshaweonline.ca
Dinh, Patrick - 1260577	n_dinh240521@fanshaweonline.ca

Version 1.0 Oct 18, 2025

Index

• Milestone Summary

During this milestone, research activities were conducted to support decision-making regarding the selection of technologies, methodologies, software tools, system architecture, and the data dictionary. Additionally, the initial website prototype was developed, and detailed requirements were gathered and documented through UML diagrams.

Event	Description	Start date	Due date	Status
Milestone 1	Planning & Requirements	2025-09-03	2025-09-29	Completed
Milestone 2 & 3	System Design	2025-09-30	2025-10-10	Completed
Milestone 4	Development	2025-10-09	2025-10-31	Processing
Milestone 4	Testing & Validation	2025-10-31	2025-11-19	Not Started
Final	Finalization & Reporting	2025-11-19	2025-12-05	Not Started

Table 1: Milestone summary and status

At this stage of the capstone project, all required steps for Milestone 2 completion have been performed and documented, which are:

- Web Search
- Draft of:
    - System Architecture
    - UML Diagrams
    - Entity-Relationship Diagram
    - Data Dictionary
    - UI/UX Prototypes

1.Web Search

Following comprehensive research and analysis, our team identified the most suitable technologies and methodologies for developing the capstone project system. The objective is to ensure that the solution is reliable, efficient, and user-friendly while maintaining scalability and cost-effectiveness. The selected approaches are designed to optimize system performance, enhance data security, and improve the overall user experience.

1.1 Hardware Development Technologies and Frameworks

Our project relies on the ESP32 microcontroller and the MAX30102 pulse oximeter/heart rate sensor. Together, they form the core hardware layer for real-time physiological data acquisition, processing, and communication. The hardware development is supported by microcontroller programming frameworks, peripheral libraries, and testing tools to ensure accurate data collection and efficient device operation

1.1.1 Sensor Integration Technologies

- **Arduino Core for ESP32**: The Arduino Core provides a simplified development environment for programming the ESP32. It offers high-level APIs for GPIO, I²C, SPI, WiFi, and sensor integration. We chose Arduino Core because it speeds up development through pre-built libraries, including support for the MAX30102 sensor, and integrates easily with the Arduino IDE or PlatformIO. In our project, Arduino Core is used for rapid prototyping, handling sensor data acquisition, and transmitting results via WiFi or Bluetooth.

- **ESP-IDF (Espressif IoT Development Framework)**: ESP-IDF is the official framework from Espressif, offering low-level hardware control, multitasking via FreeRTOS, and optimized networking stacks. While it is more complex than Arduino Core, ESP-IDF provides greater flexibility and fine-grained resource management. In the current phase, we will not rely on ESP-IDF to avoid overcomplicating development. However, if additional time is available in later project stages, we will consider migrating selected modules (e.g., real-time multitasking or advanced power optimization) to ESP-IDF to enhance system robustness.

1.1.2 Sensor Integration Technologies

- **I²C Communication Protocol**: The MAX30102 communicates with the ESP32 through the I²C protocol. We chose I²C because it is efficient, requires only two data lines (SDA and SCL), and supports reliable communication between microcontrollers and low-power sensors. In our project, I²C is used to transmit raw infrared and red-light readings from the MAX30102 to the ESP32 for further processing.

- **MAX30102 Sensor Library**: Open-source MAX30102 libraries are used to simplify the interaction with the sensor. These libraries provide functions for initializing the sensor, reading raw photoplethysmography (PPG) signals, and calculating heart rate and SpO_2 levels. Using these libraries allows us to focus on algorithm implementation instead of low-level sensor configuration.

1.1.3 Development Tools

- **Arduino IDE / PlatformIO:** Arduino IDE is used for quick development and deployment, while PlatformIO (inside Visual Studio Code) is used for more advanced debugging and dependency management. Both tools allow uploading code to the ESP32, monitoring serial output, and testing sensor data in real time.

- **Serial Monitor:** For debugging I^2C signals and verifying communication between ESP32 and MAX30102, the built-in Arduino Serial Monitor is used. This helps ensure the reliability of data transfer and validates sensor initialization.

1.1.4 Wireless Communication Frameworks

- **WiFi (IEEE 802.11 b/g/n):** WiFi is integrated into the ESP32 and is used to transmit health data to a remote server or web application. We selected WiFi because it provides high data throughput and supports direct integration with cloud platforms for real-time monitoring.

- **Serial Monitor:** For debugging I^2C signals and verifying communication between ESP32 and MAX30102, the built-in Arduino Serial Monitor is used. This helps ensure the reliability of data transfer and validates sensor initialization.

1.1.5 Algorithmic Processing

- **Signal Processing Libraries (C/C++):** The raw data from the MAX30102 requires filtering (e.g., moving average or FIR filters) and peak detection algorithms to calculate accurate heart rate and SpO_2. C/C++ libraries for digital signal processing are integrated within the ESP32 codebase to ensure real-time performance.

1.2 Maintenance and Version Control

To ensure stability, reliability, and long-term usability of the project, we will implement clear maintenance procedures and a structured version control strategy.

1.2.1 Maintenance

- **Bug Fixes and Updates:** Any errors identified during development or operation will be documented, fixed, and validated through a structured process to maintain system reliability.

- **Dependency Management:** Regular updates of ESP32 SDK, Arduino Core, or ESP-IDF libraries will be performed to minimize security risks and prevent outdated dependencies.

- **Hardware Maintenance:** The ESP32 development board and MAX30102 sensor will be periodically tested to ensure stable performance. Damaged modules will be replaced promptly.

- **Documentation Updates:** Technical documentation, including system architecture, interface specifications, and user guides, will be continuously updated to support both current and future team members.

1.2.2 Version Control

- **Git Version Control System:** The project source code will be hosted on GitHub to ensure transparency, collaboration, and traceability of changes.

- **Code Review Process:** All contributions will undergo peer review before merging into the main branch to ensure code quality, maintainability, and consistency.

1.3 Frontend Technologies

HeartLink's web dashboard integrates frontend web technologies, cloud platforms, and API connections to visualize real-time biomedical data. The dashboard serves as the main interface where users can view live readings, trends, and alerts received from the ESP32 through cloud synchronization.

1.3.1 HTML5, CSS3, and JavaScript

- Structure and style dashboard components.
- Implement interactive features such as real-time charts and alerts.
- Ensure responsiveness and cross-browser compatibility.

1.3.2 React.js (Frontend Framework)

- Display real-time heart rate and PPG data from the cloud.
- Manage UI components such as graphs, status indicators, and alert panels.
- Handle state updates efficiently for continuous data refresh.

1.3.3 Chart.js / D3.js (Visualization Libraries)

- Display real-time heart rate data trends.
- Visualize pulse variations and time-based patterns.
- Offer clear and engaging data presentation for users.

1.3.4 API and WebSocket Communication

- Instant updates of heart rate data.
- Reliable synchronization with minimal delay.

- Continuous data flow for live visualization.

1.4 User Interface (UI) and User Experience (UX) Design

HeartLink's dashboard design emphasizes clarity, simplicity, and accessibility for users viewing health data in educational and non-clinical contexts.

1.4.1 UI/UX principles

- Clean layout with clearly labeled sections for heart rate, signal strength, and system status.
- Use of color coding (e.g., green for normal, red for high readings).
- Responsive design for accessibility on laptops, tablets, and desktops.
- Easy navigation with intuitive menus and icons.
- Consistent typography and contrast for readability.

1.4.2 Usability features

- Clean layout with clearly labeled sections for heart rate, signal strength, and system status.
- Use of color coding (e.g., green for normal, red for high readings).
- Responsive design for accessibility on laptops, tablets, and desktops.
- Easy navigation with intuitive menus and icons.
- Consistent typography and contrast for readability.

1.5 Data Engineering Pipeline

The data engineering pipeline for this project involves several key stages: Data Acquisition (from the sensor), Ingestion (transferring data to the cloud), Storage & Processing, and finally Visualization.

1.5.1 Data Ingestion (Sensor to Cloud)

- **Google Cloud Pub/sub (Real-time Data Ingestion):** Pub/Sub is a fully managed, serverless, real-time messaging service. It acts as a buffer, reliably ingesting high-volume, continuous streams of data from your ESP32/Raspberry Pi. It decouples the data sender (sensor) from the data receiver (database/pipeline), ensuring that even if your device sends bursts of data, no data is lost during pipeline processing.

- **Google Cloud IoT Core (Cloud-Side Authentication):** IoT Core provides secure, direct connections for global devices. It handles device authentication (crucial for capstone security), connection, and communication, pushing all data directly into a Pub/Sub topic.

1.5.2 Data Storage and Processing

- **Google BigQuery (Cloud Data Warehouse):** BigQuery is a **highly scalable, serverless data warehouse** that is optimized for fast, complex SQL queries on massive datasets. Crucially, **it offers native, high-performance integration with Looker Studio (formerly Data Google Studio)**, making it the ideal final storage layer before visualization. It handles the high volume and velocity of time-series sensor data perfectly.

- **Google Cloud Dataflow or BigQuery SQL (Data Transformation):** BigQuery is a **highly scalable, serverless data warehouse** that is optimized for fast, complex SQL queries on massive datasets. Crucially, **it offers native, high-performance integration with Looker Studio (formerly Data Google Studio)**, making it the ideal final storage layer before visualization. It handles the high volume and velocity of time-series sensor data perfectly.

1.5.3 Visualization and Reporting

- **Google Looker Studio (Automation Dashboard):** As requested, this tool is free to use and provides an intuitive, web-based interface for creating interactive, automated dashboards. It has a native, high-speed connector to BigQuery, allowing for real-time visualization of the ingested heart pulse and SpO_2 data without needing to move or copy data.

2.Draft of UML Diagrams

2.1 System Architecture

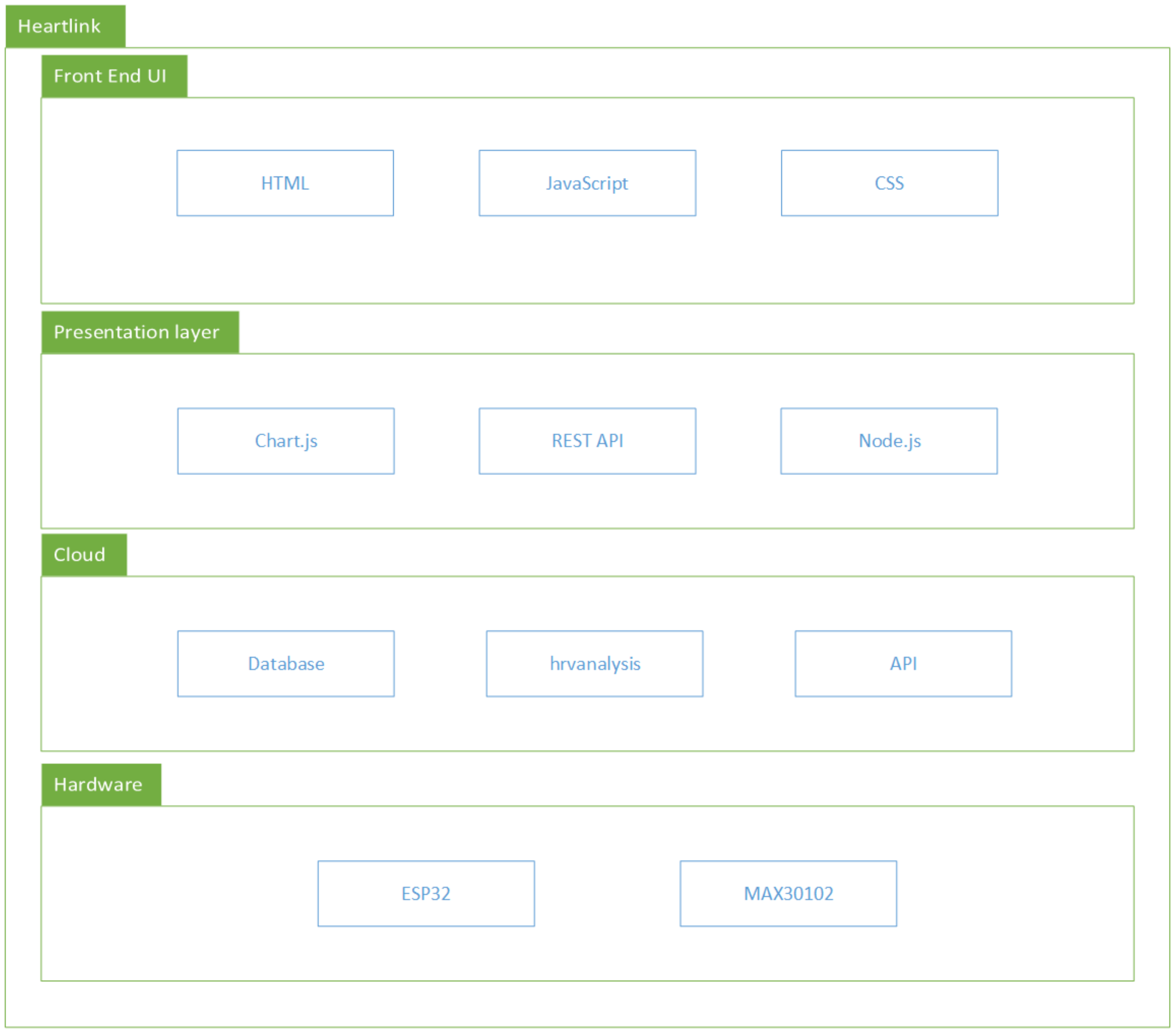

Figure 1: System Architecture

The diagram provides an overview of the HeartLink's system architecture. The proposed system architecture is organized into four primary layers: the Hardware Layer, the Cloud Layer, the Presentation Layer, and the User Interface Layer. Each layer plays a distinct role in ensuring the system's functionality, scalability, and reliability.

- **Hardware Layer:** This layer is responsible for data acquisition and initial signal processing. It consists of the ESP32 microcontroller and the MAX30102 pulse oximeter sensor, which work together to measure heart rate and blood oxygen saturation (SpO_2)

levels. The ESP32 handles sensor communication, data sampling, and transmission to the cloud. It also manages interrupt-driven operations to ensure efficient, real-time data collection.

- **Cloud Layer:** The cloud layer serves as the data management and processing center. It receives physiological data from the hardware via Wi-Fi, performs computations such as heart rate variability (HRV) analysis, and stores results in a secure database. This layer ensures scalability, data persistence, and availability for future analytics or visualization.

- **Presentation Layer:** The presentation layer acts as a bridge between the cloud and the user interface. It handles data retrieval, formatting, and visualization logic to ensure that processed data is presented meaningfully to the end user. It also provides APIs that enable seamless communication between the backend and the frontend.

- **User Interface (UI) Layer:** The UI layer delivers processed information to users through an intuitive and responsive web interface. Users can view real-time heart rate and SpO_2 readings, analyze historical trends, and monitor overall health status. This layer emphasizes accessibility, usability, and visual clarity to enhance the user experience.

2.2 Hardware UML Diagrams

2.2.1 Flow Chart

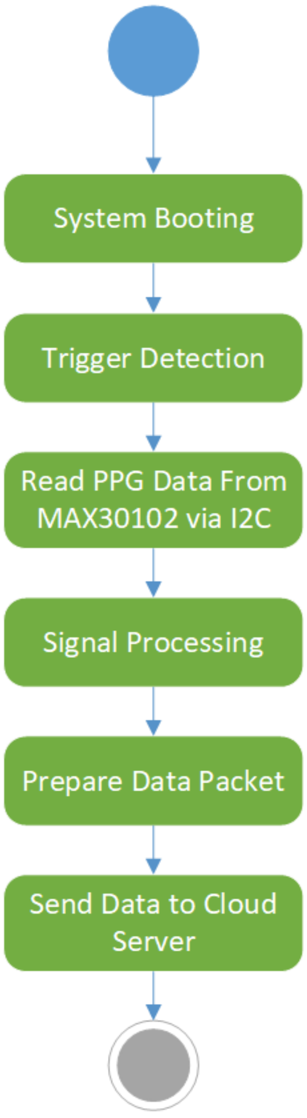

Figure 2: Flow Chart

2.2.2 Sequence Diagram

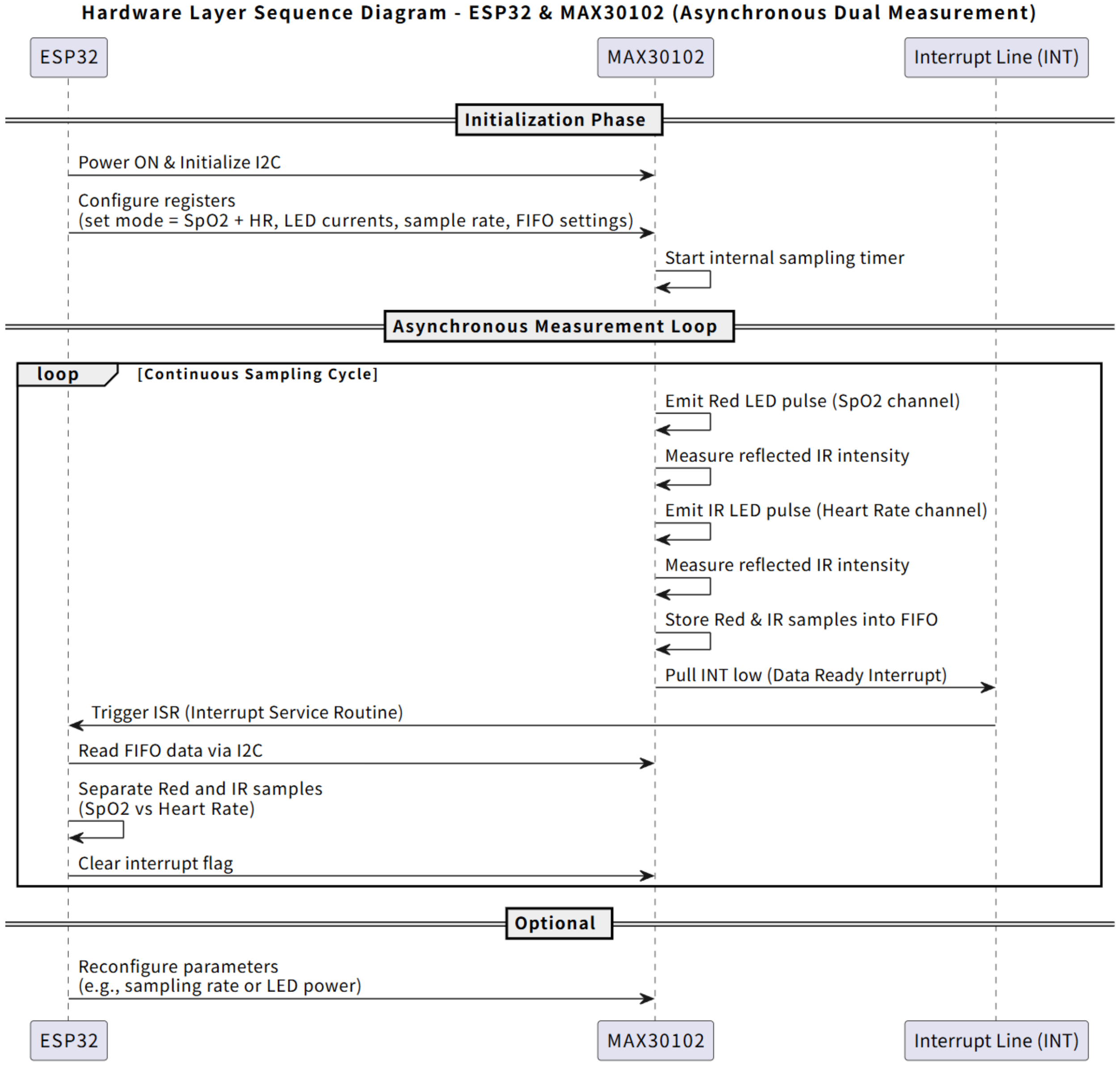
Hardware Layer Sequence Diagram - ESP32 & MAX30102 (Asynchronous Dual Measurement)
ESP32
MAX30102
Interrupt Line (INT)
Initialization Phase
Power ON & Initialize I2C
Configure registers (set mode = SpO2 + HR, LED currents, sample rate, FIFO settings)
Start internal sampling timer
Asynchronous Measurement Loop
loop [Continuous Sampling Cycle]
Emit Red LED pulse (SpO2 channel)
Measure reflected IR intensity
Emit IR LED pulse (Heart Rate channel)
Measure reflected IR intensity
Store Red & IR samples into FIFO
Pull INT low (Data Ready Interrupt)
Trigger ISR (Interrupt Service Routine)
Read FIFO data via I2C
Separate Red and IR samples (SpO2 vs Heart Rate)
Clear interrupt flag
Optional
Reconfigure parameters (e.g., sampling rate or LED power)

Figure 3: Sequence Diagram

2.3 Front-End UML Diagrams

2.3.1 Use Case Diagram

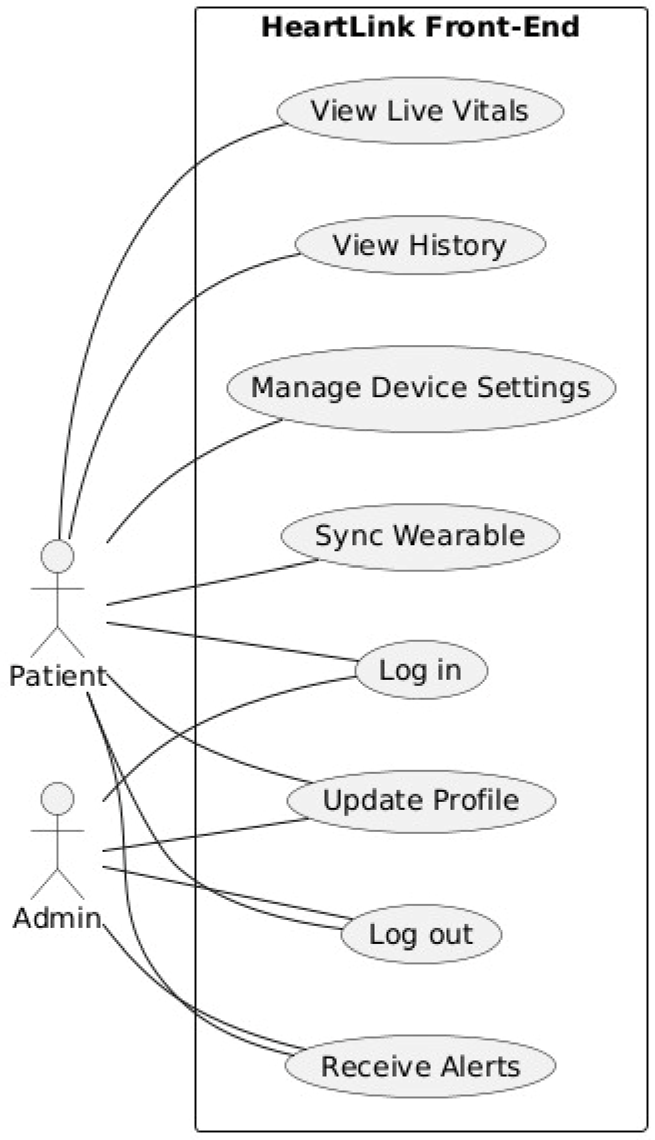

Figure 4: Use Case Diagram

2.3.2 Activity Diagram

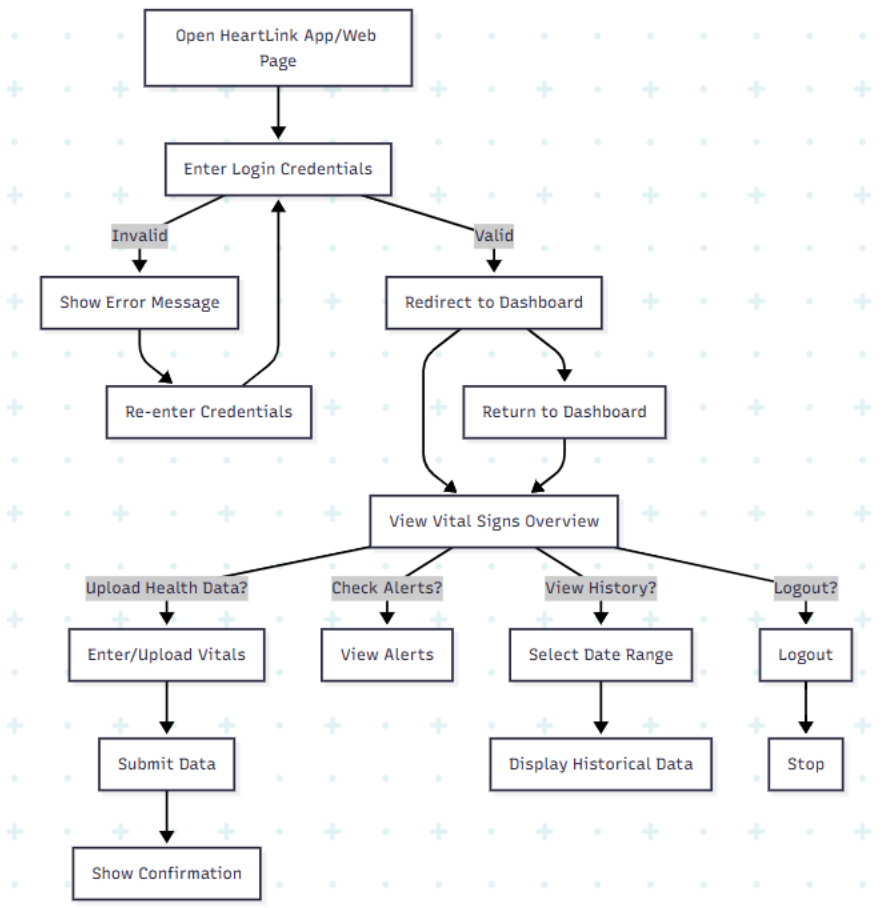

Figure 4: Activity Diagram

2.4 Cloud Layer UML Diagrams

2.4.1 Class Diagram

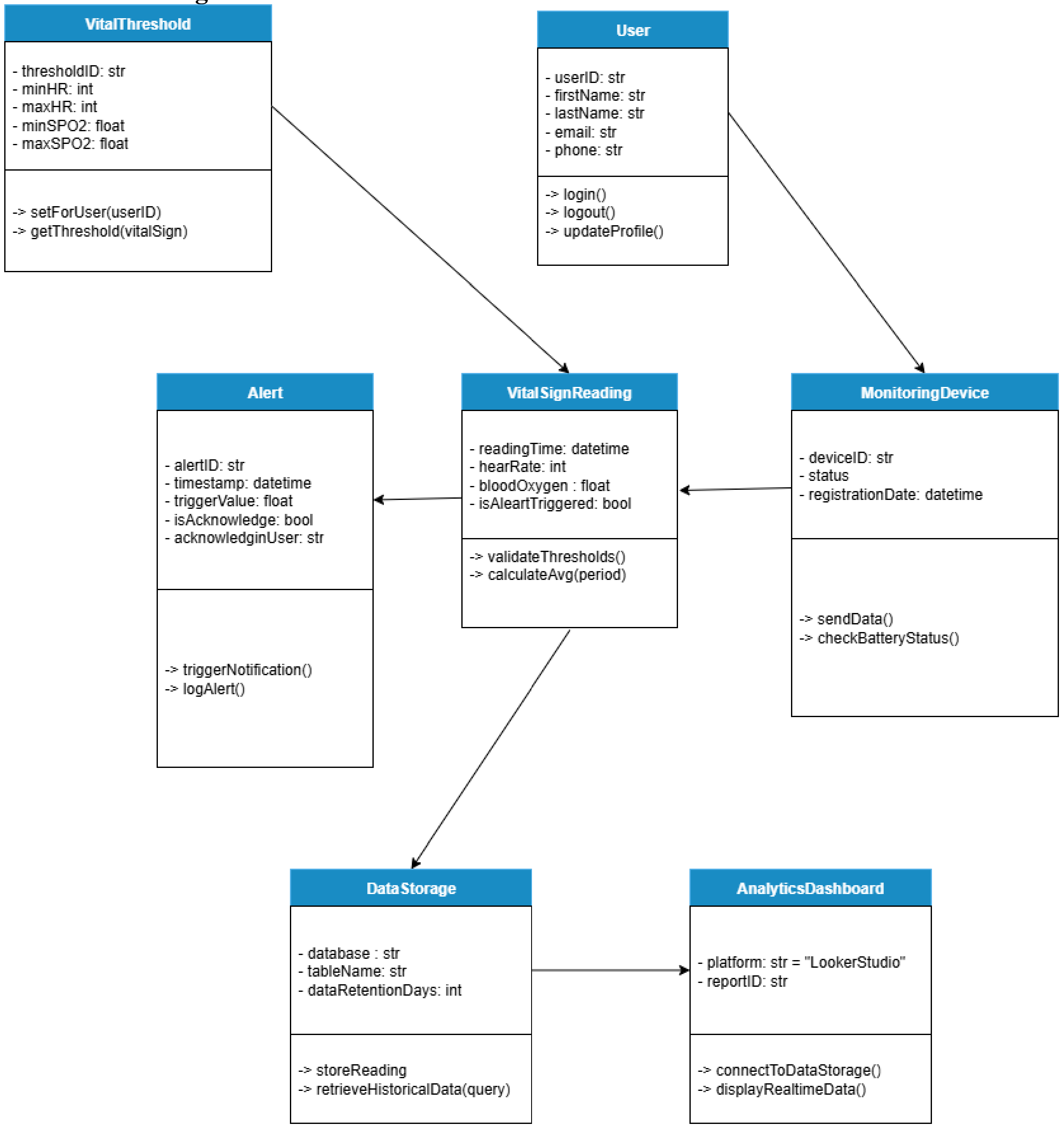

Figure 5: Class Diagram

2.4.2 Flow Chart

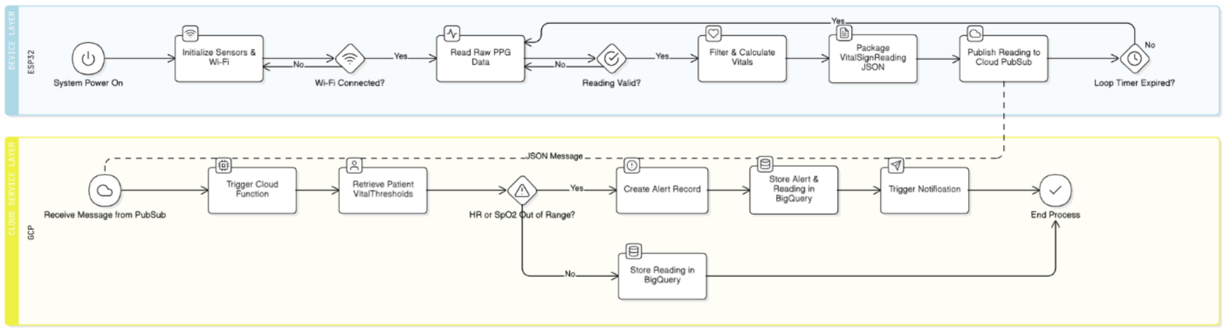

Figure 6: Flow Chart

3.Draft of Data Dictionary

3.1 Hardware Layer Data Dictionary

Varibles	Type	Unit/Format	Description
timeStamp	string	YYYY-MM-DD HH:MM:SS	Timestamp (YYYY-MM-DD HH:MM:SS) marking the exact time of each sensor reading, generated by the ESP32 clock.
deviceId	string	-	Unique identifier assigned to the ESP32 board for tracking data source during cloud synchronization.
redAdc	int	ADC count (18bit)	Raw ADC reading from the MAX30102 red LED channel, representing reflected light intensity for calculation.
irAdc	int	ADC count (18bit)	Raw ADC reading from the MAX30102 infrared LED channel, used to detect pulse waveform and assist estimation.
fifoPointer	int	0-31	Current position pointer in MAX30102 FIFO register, used to track unread sensor samples.
overflowCount	int	0-31	Counter indicating how many times the FIFO buffer overflowed due to delayed data reading.
heartRate	float	Beats per minute	Calculated heart rate (in beats per minute) derived from processed red and IR signals using pulse detection algorithms.
ibi	float	ms	Inter-beat interval measured in milliseconds, representing the time difference between consecutive heartbeats.
spo2	float	%	Estimated blood oxygen saturation percentage, computed from the ratio of red and IR signal amplitudes.
ledCurrentRed	float	mA	Current drive level (in mA) applied to the red LED, configurable via the MAX30102 register.
ledCurrentIr	float	mA	Current drive level (in mA) applied to the IR LED, adjustable to optimize signal strength.

signalQuality	int	0-100	Computed quality score of the PPG signal, indicating noise level or motion artifact presence.
uploadStatus	bool	true/false	Indicates whether the current data packet has been successfully uploaded to the cloud database
wifiRssi	int	dBm	Received Signal Strength Indicator (RSSI) value from the ESP32 Wi-Fi connection, used to assess network quality.
sessionId	string	UUID	Unique session identifier generated for each monitoring session to group related data records.

Table 2: Data Dictionary of Hardware Layer

3.2 Front-End Layer Data Dictionary

3.2.1 User Interface Data

Varibles	Type	Description
screen_name	string	Identifies the current view (e.g., Dashboard, Settings, History).
deviceId	enum	Determines layout appearance (light/dark modes).
redAdc	int	Tracks the active UI route.

Table 3: Data Dictionary of User Interface

3.2.2 Vital Signs Data

Varibles	Type	Description
heart_rate	string	Representing BPM (beats per minute).
spo2_level (optional)	float	Blood oxygen saturation percentage.

timestamp	string/date time	The time of recording.

Table 3: Data Dictionary of Vital Signs Data

3.2.3 Device and Connectivity Data

Varibles	Type	Description
device_id	string	Unique identifier for the ESP32 unit.
connection_status	enum	Indicates the strength of the signal.
last_sync_time	string/date time	The time of the last synchronous.

Table 4: Data Dictionary of Device and Connectivity Data

3.2.4 Alert and Notification Data

Varibles	Type	Description
alert_message	string	Display the alert messages to users.

Table 5: Data Dictionary of Hardware Laye

3.2.5 User and Settings Data

Varibles	Type	Description
user_id	string	Unique identifier for the user.
preferred_units	enum	Choose the display unit (e.g. BPM).

notifications_enabled	bool	Turn on/off the notifications.
cloud_sync_enabled	bool	Boolean flags synchronized to the cloud.
Language/locale	string	String for regional formatting.
theme_preference	enum	Choose the theme (e.g. Light, Dark, System).

Table 6: Data Dictionary of User and Settings Data

3.2.6 Historical and Trend Data

Varibles	Type	Description
history_records	Arrays of heart_rate, spo2_level and timestamp	Arrays of heart_rate, spo2_level and timestamp.
date_range_filter	string	The range of date.
chart_data	Arrays for graph plotting	Arrays for graph plotting.

Table 7: Data Dictionary of Historical and Trend Data

3.3 Cloud Layer Data Dictionary

3.3.1 User Entity

Attribute Name	Data Type	Key Type	Description	note

User_ID	STRING	PK	Unique identifier for any actor	Assigned UUID.
First_Name	STRING		User's first name.	Required.
Last_Name	STRING		User's last name.	Required.
User_Role	STRING		The actor's role in the system.	Enum : "Admin".
Email	STRING	Unique	User's primary contact and login.	Must be unique.
Phone_Number	STRING		User's mobile number for notifications.	

Table 8: Data Dictionary of User Entity

3.3.2 Device Entity

Attribute Name	Data Type	Key Type	Description	note
Device_ID	STRING	PK	Unique identifier for the physical IoT device.	Unique per hardware unit.
Patient_User_ID	STRING	FK	The ID of the patient to whom the device is assigned.	Links to USER.User_ID.
Model	STRING		Hardware model of the monitoring unit.	e.g., "ESP32/MAX30102".
Registration_Date	TIMESTAMP		The date the device was first activated.	

Table 9: Data Dictionary of Device Entity

3.3.3 Reading Entity (time-series Data)

Attribute Name	Data Type	Key Type	Description	note
Reading_ID	STRING	PK	Unique identifier	Generated UUID.
Device_ID	STRING	FK	The device that generated this measurement.	Links to DEVICE.Device_ID.

Reading_Timestamp	TIMESTAMP	PK	The exact time the measurement was taken (UTC).	Crucial for ordering and partition key.
Heart_Rate_BPM	INTEGER		The calculated heart rate in beats per minute.	Expected range: 30 - 250.
SpO2_Level	FLOAT		The calculated blood oxygen saturation level.	Expected range: 80.0 - 100.0.

Table 10: Data Dictionary of Reading Entity

3.3.4 Threshold Entity

Attribute Name	Data Type	Key Type	Description	note
Threshold_ID	STRING	PK	Unique identifier for this set of patient thresholds.	
Patient_User_ID	STRING	PK, FK	The patient this threshold applies to.	Links to USER.User_ID.
Min_Heart_Rate	INTEGER		Minimum acceptable Heart Rate (BPM).	
Max_Heart_Rate	INTEGER		Maximum acceptable Heart Rate (BPM).	
Min_SpO2	FLOAT		Minimum acceptable SpO2 level.	
Set_By_User_ID	STRING	FK	The Doctor/User who configured this threshold set.	Links to USER.User_ID.

Table 11: Data Dictionary of Threshold Entity

3.3.5 Alert Entity (Event Data)

Attribute Name	Data Type	Key Type	Description	note

Alert_ID	STRING	PK	Unique identifier for a triggered alert instance.	
Reading_ID	STRING	FK	The specific reading that triggered this alert.	Links to READING.Reading_ID.
Patient_User_ID	STRING	FK	The patient who triggered the alert.	Links to USER.User_ID.
Alert_Timestamp	TIMESTAMP		The time the system recognized the alert condition.	
Type	STRING		The condition that caused the alert.	Enum: "HR_High", "HR_Low", "SpO2_Low".
Is_Acknowledged	BOOLEAN		Flag if a professional has closed the alert.	Default: FALSE.
Acknowledging_User_ID	STRING	FK	The ID of the User who closed the alert.	Links to USER.User_ID. (Nullable if not acknowledged).

Table 12: Data Dictionary of Alert Entity

4.Draft of ER Diagram

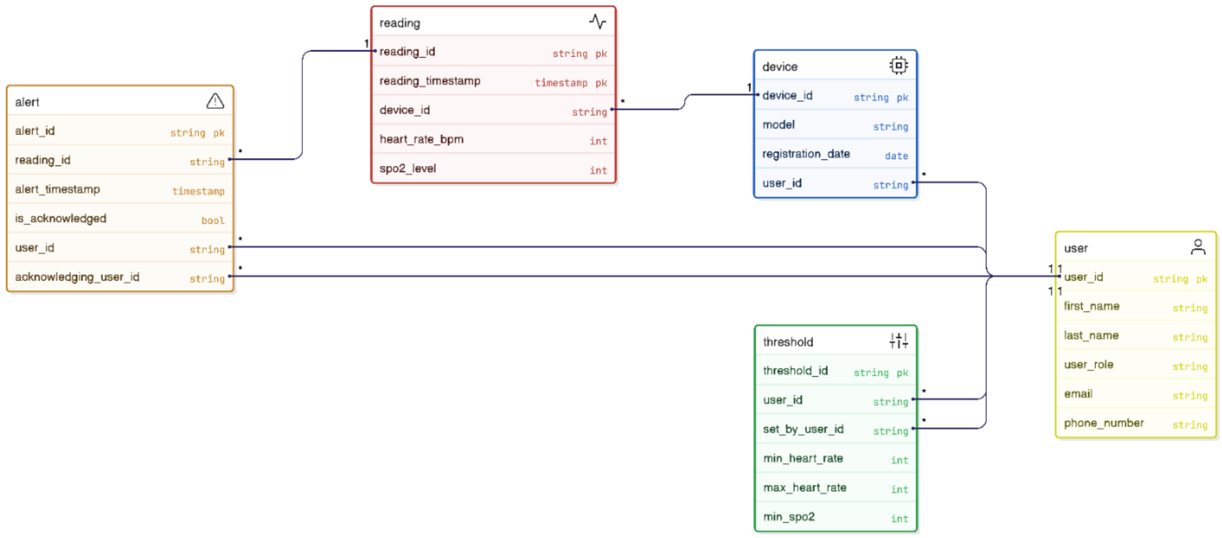

Figure 7: ER Diagram

5.Draft of UI and UX Prototypes

https://www.figma.com/make/rU63BB8kvgExZDKW5ixYQa/Wearable-Health-Monitor?node-id=0-1&t=g6IvCl59lmOZ5bmX-1

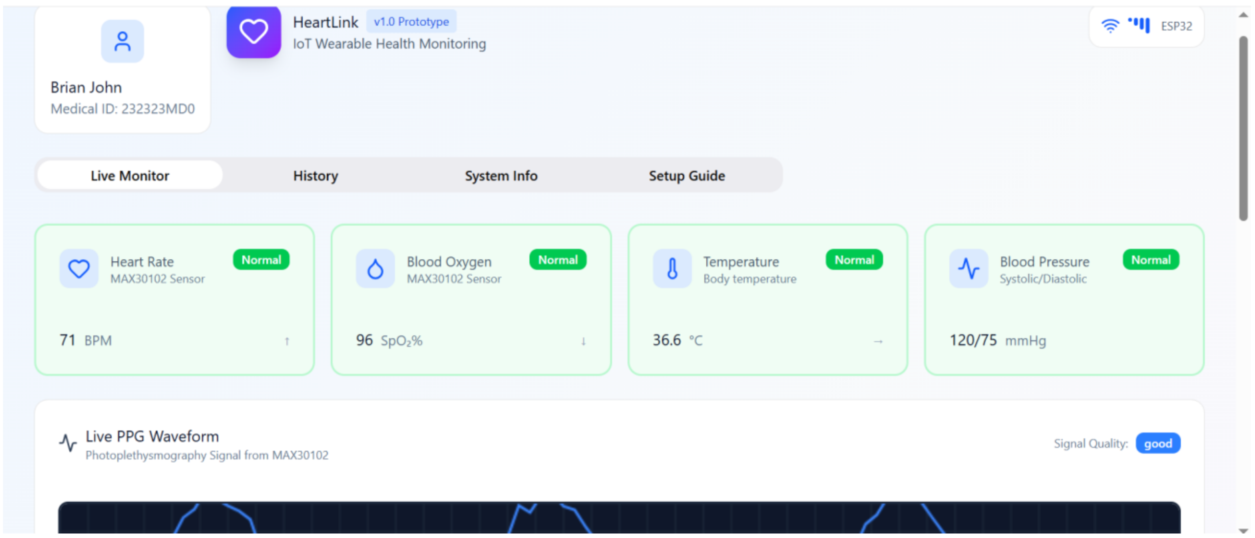

Figure 8: Draft of UI and UX Design

5.1 Overview of Structure

The webpage structure includes the following components:

- **State:** Controls whether the device is connected (simulated with a timeout).
- **VitalSignsMonitor:** Shows live heart rate & vitals.
- **LiveWaveform:** Real-time waveform/ECG graph.
- **HistoricalData:** Past vitals.
- **AlertsPanel:** Active alerts.
- **DeviceStatus:** Connection status.
- **SystemInfo:** Device/system info.
- **SetupGuide:** Guide for pairing/setup.
- **DisclaimerBanner:** Legal disclaimer.
- **Footer:** Webpage footer.
- **UI Framework:** Uses TailwindCSS for layout and styling.
- **Tabs Navigation:** Live Monitor, History, System Info, Setup Guide.

5.2 UI/UX Flow Based on Code

- **Header:**
    - Logo + Heart icon
    - Website name + version
    - Short description
    - Device status (connection indicator)

- **Main Tabs:**
    - **Live Monitor**
        - VitalSignsMonitor → Real-time readings
        - LiveWaveform → Graph of heart signals
        - AlertsPanel → List of alerts
    - **History**
        - HistoricalData → Past readings and trends
- **System Info**
        - SystemInfo → Device info, connectivity
- **Setup Guide**
        - SetupGuide → Step-by-step pairing instructions

- **Other UI Elements:**
    - DisclaimerBanner → Always visible below header
    - Footer → Site footer with links/info

5.3 Data Flow and Usage in the Frontend

Cloud-based retrieval from services such as Firebase or AWS, if sync is enabled. Data is typically stored in state variables or local models within the application. JSON objects are commonly used for internal handling and API exchange. UI components (cards, charts, text fields) map directly to these data structures to ensure consistent rendering.

5.4 UI Integration and Component Mapping

Each frontend component (dashboard card, connection badge, alert modal, history list) references fields from the data dictionary. Examples:

- heart_rate → Large text element in the dashboard center.
- connection_status → Icon indicator in the screen header.
- alert_message → Notification box or popup.
- history_records → Scrollable list or chart component.

Design tools like Figma can reference these fields when labeling sections, setting placeholders, or creating prototypes. This alignment improves consistency between UI mockups and implementation.

5.5 Consistency and Scalability Benefits

A structured data dictionary helps:

- Align frontend components with backend APIs or cloud services.
- Simplify future integrations (e.g., AI analytics or mobile app scaling).
- Enable efficient debugging, updates, and collaboration.
- Maintain compatibility with version control and documentation practices (e.g., GitHub repositories).

By formally defining and adhering to frontend data structures, HeartLink can maintain a flexible, scalable, and user-friendly interface while supporting real-time data monitoring and alert-based workflows.

5.6 Suggested UI/UX Enhancements

- **Header:**

    - Add connection animation for isConnected instead of static checkmark

- **Live Monitor:**
    - Use card layout for vital signs (heart rate, oxygen, BP)
    - Add color-coded indicators: green (normal), yellow (warning), red (critical)

- **Historical Data:**
    - Allow time range filters: 1h, 24h, 7d
    - Add export button (CSV/PDF)

- **Alerts Panel:**
    - Add acknowledge & resolve buttons
    - Show timestamp & type of alert

- **Setup Guide:**
    - Step-by-step checklist with visual indicators for completion

- **Responsive Design:**
    - Ensure mobile/tablet view collapses sidebar and tabs properly

6.References

MAX30102. (n.d.). https://www.analog.com/media/en/technical-documentation/data-sheets/max30102.pdf

Instructables. (2023, June 3). *Guide to Using MAX30102 Heart Rate and Oxygen Sensor With Arduino*. Instructables. https://www.instructables.com/Guide-to-Using-MAX30102-Heart-Rate-and-Oxygen-Sens/?utm_source=chatgpt.com

UML Use Case Diagram Tutorial. (2025, August 22). Lucidchart. https://www.lucidchart.com/pages/tutorial/uml-use-case-diagram

SmartDraw. (2019). *SmartDraw - Create Flowcharts, Floor Plans, and Other Diagrams on Any Device*. Smartdraw.com. https://www.smartdraw.com/

Espressif Documentation. (2025). Espressif.com. https://documentation.espressif.com/esp32_datasheet_cn.pdf

Introduction to GitHub. (2022, October 18). GitHub. https://github.com/skills/introduction-to-github

Figma. (2021). *Figma*. Figma. https://www.figma.com/community

(2025). Fanshaweonline.ca. https://www.fanshaweonline.ca/d2l/le/content/2001609/viewContent/17909532/View

All APIs and references. (2025). Google Cloud. https://cloud.google.com/pubsub/docs/apis

Node.js client library. (n.d.). Google Cloud. https://cloud.google.com/nodejs/docs/reference/iot/latest

APIs and reference. (2025). Google Cloud. https://cloud.google.com/bigquery/docs/reference

Welcome to Looker Studio! (2025). Google Cloud. https://cloud.google.com/looker/docs/studio

Sullivan, H., & Lin, M. (2021). *Cloud-Centric IoT Data Processing: A Multi-Platform Approach Using AWS, Azure, and Snowflake*. *2*, 12–23. https://doi.org/10.63282/3050-9416.ijaibdcms-v2i1p102

(2022). Amazon.com. https://docs.aws.amazon.com/iot/

Borra, P. (2024). Impact and Innovations of Azure IoT: Current Applications, Services, and Future Directions. *International Journal of Recent Technology and Engineering (IJRTE)*, *13*(2), 21–26. https://doi.org/10.35940/ijrte.b8111.13020724