Device Controller-Monitor (DCM) Documentation

MECHTRON 3K04 Assignment 2
Group 4
Andrew De Rango, Ali Hussin, Ethan Otteson,
Marco Tan, Rafael Toameh
Friday, November 29, 2024

Table of Contents

1 Scope

This document provides a comprehensive overview of HeartFlow, a desktop application designed for healthcare providers to manage and control their patients' pacemaker settings effectively. It outlines the program's functionality, implementation, design decisions, and justifications made throughout the development process, enabling users and developers to have a clear understanding of the application's features and software architecture.

2 Terms and Definitions

Pacemaker Firmware - All software that is embedded into the pacemaker microcontroller.

DCM - Device controller-monitor, the software and graphical user interface that is used to interact with the pacemaker board and heart.

Off-p Pacemaker mode in which no sensing or shocking occurs.

AOO - Pacemaker mode in which the atria are shocked rhythmically, without considering the natural atrial pulse.

VOO - Pacemaker mode in which the ventricles are shocked rhythmically, without considering the natural ventricular pulse.

AAI - Pacemaker mode in which the atria are shocked rhythmically unless inhibited by natural atrial pulses.

VVI - Pacemaker mode in which the ventricles are shocked rhythmically unless inhibited by natural ventricular pulses.

ARP - Atrial refractory period.

VRP - Ventricular refractory period.

BPM - Beats Per Minute (heart rate).

UI - User interface, the part the user interacts with.

IPC - Inter-process communication is the communication protocol that links the main process and the renderer process together.

Main process - The process in an Electron app responsible for handling operating system-level tasks such as window creation and resource management.

Session - Period that the user is logged in and is interacting with the HeartFlow dashboard.

Renderer process - The process in an Electron app responsible for handling the rendering of the UI.

Electron - The application development framework used to make the application on the operating system level.

React - The JavaScript framework used to develop the UI.

State - In the context of React, it is data that is internal to a component and can change over time.

Props (properties) - In the context of React, they are values that are passed from parent components to child components, behaving similarly to the arguments of a function.

DOM - Document Object Model. It is the area where HTML components are rendered and can be interacted with. This is also the area where the user interacts with the application, as the DOM is rendered in the window that is created.

3 Purpose

The purpose of the HeartFlow application is to provide healthcare providers with a comprehensive tool for monitoring telemetry data, adjusting pacemaker parameters, and ensuring optimal device performance. Designed with a focus on usability, HeartFlow enables users to navigate seamlessly between various functionalities, empowering them to deliver high-quality care to their patients. By allowing for real-time and secure management of pacemaker settings, the application enhances the ability of healthcare providers to respond promptly to patient needs, improving overall patient outcomes. Additionally, HeartFlow aims to facilitate accurate data handling and device control, thereby increasing the reliability of pacemaker operations. This ensures that healthcare providers can make informed decisions based on up-to-date telemetry information and patient-specific configurations.

4 Requirements

The following outlines all the recognized DCM requirements. These requirements come from either the PACEMAKER requirements document provided as part of the project instruction package or from the project team directly. Not all requirements presented in the PACEMAKER document are included as many were considered out of scope for the implementation required for this project. This section includes all of the requirements given in natural language with many also provided in tabular expressions and class diagrams for clarity, and a note regarding the expected future requirement changes.

4.1 Requirements Overview

The home page should provide a welcoming interface with options for user registration and login. The user registration page must facilitate the creation of accounts with secure input fields for username, password, and pacemaker serial number. The user login page should enable existing users to authenticate their credentials while ensuring password confidentiality. The DCM dashboard should serve as the primary interface for managing pacemaker settings, and displaying critical information such as programmable parameters, device serial numbers, and telemetry status. It should support user interactions like updating parameters, terminating connections, and storing user data securely. Together, these pages ensure a cohesive and user-friendly experience for managing pacemaker functionality within the HeartFlow application. In addition, the DCM must be expanded to include all required modes and parameters while implementing serial communication to transmit and receive information between the DCM and the Pacemaker. The updated system should also be able to set, store, and transmit programmable data, and ensure it is stored in an appropriate and organized manner within the Pacemaker device. The system should be able to display electrogram data for the ventricle, atrium, or both. The DCM should be able to achieve this by receiving electrogram data from the Pacemaker through a serial communication link to be able to display the data. Further, this electrogram data, along with serial logs and user activity history should be available to be downloaded from the DCM via a reports generation feature.

4.2 Home Page Requirements

The home page should be the default page opened upon launching of the application. The home page should permit the user to either log in to their existing account or register a new account. The home page should convey the logo branding and a short slogan, with the software release version.

4.3 User Registration Requirements

Users should be able to register with a unique username that is no less than 3 characters long. Passwords must meet minimum security standards, including at least 8 characters with one special character, and must be confirmed correctly in the "Confirm Password" field. The "Pacemaker Serial Number" is required and cannot be left blank. If all fields meet these conditions, the system should register the user and store the data in its local database. Otherwise, an error should indicate which input field needs correction, such as a username that is too short, a weak password, or a mismatched confirmation password.

Table 4.3.a: Tabular Expression for user registration.

Condition	Username	Password	Confirm Password	Pacemaker Serial Number	Existing Users	Result
Username unique and ≥ 3 characters	Valid	≥ 8 characters, special character	Matches password	Non-null	< 10	Successful registration

Username < 3 characters	Invalid	Any	Any	Any	Any	Error: Username is too short
Username already exists	Invalid	Any	Any	Any	Any	Error: User already exists
Password < 8 characters	Any	Invalid	Any	Any	Any	Error: Password is too short
Password contains no special characters	Any	Invalid	Any	Any	Any	Error: Password is too weak
Confirm Password ≠ Password	Any	Any	Mismatch	Any	Any	Error: Password mismatch
Pacemaker Serial Number is null	Any	Any	Any	Null	Any	Error: Null serial number
Max user limit reached	Any	Any	Any	Any	≥ 10	Error: User limit reached

4.4 User Login Requirements

The DCM should allow registered users to log in using their previously created credentials. A valid username and password must be entered to access the system. If the username exists and the correct password is provided, the user should access the dashboard. If the password is incorrect, an error message should prompt the user to try again. If the username does not exist, the system should display an error notifying the user that the account is not found. The login process ensures that only authorized users can access pacemaker settings.

Table 4.4.a: Tabular Expression for user login.

Condition	Username	Password	Result
Username exists, correct password	Valid	Correct	Successful user log-in
Username exists, incorrect password	Valid	Incorrect	Error: Incorrect password
Username does not exist	Invalid	Any	Error: User not found

4.5 DCM Dashboard Requirements

The DCM dashboard should serve as the main interface within the HeartFlow software, providing users with access to essential functions and device information. At a minimum, the dashboard should display all programmable pacemaker parameters, enabling users to view and adjust the settings of the connected device directly from the interface. When a pacemaker is

connected, the device's serial number should be prominently displayed, ensuring users can verify the correct device connection. Additionally, the dashboard should show the current software release version, date, and time, offering clear context for the software environment and ensuring users know the precise system status.

The dashboard should also reflect user information, identifying the currently signed-in user and, where applicable, facilitating easy and secure multi-user access, with a capacity for storing login credentials for up to ten users locally. This feature allows multiple users to retain access settings specific to their profiles. A visual notification should inform the user if a different pacemaker device is detected than the one previously connected. In the event of telemetry disconnection, the dashboard should indicate both the connectivity status and specific causes of connection loss, such as high noise levels or a failed connection, allowing users to troubleshoot quickly. The dashboard should also be able to display reconnection to the device if connected after 10 seconds. Once reconnected, the electrogram display should continue on after the disconnection and continue the necessary functions.

The dashboard should support essential user actions, such as adjusting and saving pacemaker parameters and safely logging out of their account as needed. To enhance user control, the system should permit users to terminate the telemetry connection with the pacemaker at any time during use. For data continuity, user-specific configurations should save automatically after each session, so programmable settings are retained when users return.

4.5.1 Serial Communication Status

The DCM should indicate the communication status with a "Connected", "Disconnected", or "Reconnecting" label. The default state upon first logging in should be Disconnected, forcing the user to manually initiate communication between the DCM and pacemaker. When the pacemaker is connected and the handshake has occurred and a mode is selected, the display should show an animated, beating heart to confirm the live connection, the option to switch between all 11 modes should be available, and the electrogram should be updating live. Conversely, when the pacemaker is not actively communicating with the DCM due to no hardware connection or the user having yet to click Connect, the heart image should remain static, each mode should be disabled, and the electrogram should be static as well. Users should see the current status displayed on the dashboard so they can monitor if the connection is active.

The active serial communication status should dictate the options for the user to either connect or disconnect from the pacemaker. When in the "Connected" communication status, the user should be given the option to disconnect to terminate telemetry to terminate any accidental and/or potentially harmful active settings. When the handshake has not yet been completed or the DCM re-enters the "Disconnected" state due to an extended period in the "Reconnecting" state, the user should have the option to manually attempt to initiate communication using a "Connect" button.

Table 4.5.1.a: Tabular Expression to display telemetry status, heart animation, and electrogram.

Communication Status	Condition Triggering Status Change	Heart Animation	Live Electrogram	Displayed Status
Connected	The handshake is completed. The pacemaker should be plugged and the user should click Connect.	Animated (beating)	Updating live	Connected (Green)
Reconnecting	Handshake was completed during the active session but the most recent poll request did not receive a response.	Static (paused)	Static (paused)	Reconnecting (Yellow)
Disconnected	DCM was reconnecting for an extended period or the handshake was never completed.	Static (paused)	Static (paused)	Disconnected (Red)

4.5.2 Parameter Requirements

The DCM should offer all operational modes: AOO, VOO, AAI, VVI, AOOR, VOOR, AAIR, VVIR, DDD, and DDDR, each with its corresponding set of adjustable parameters. Additionally, users should be able to turn the mode OFF at any time when logged into their account by terminating the telemetry signal. Each programmable parameter has a specific valid range, and the user should be prompted to change their inputs outside of a defined range. Fields that are not relevant to the selected mode should be disabled, preventing users from mistakenly adjusting unnecessary settings. If a value is inputted that is not of the valid range, an error message will appear at the bottom right side of the screen, notifying the user to provide a valid input. Upon making parameter adjustments, users should be able to store the changes by pressing the "Submit" button. Successfully validated data should be stored securely in the system's local database, enabling telemetry to function with the latest configurations. The adjustable parameters, their applicable modes, and their ranges are shown below.

Table 4.5.2.a: Tabulation of all programmable parameters that should be available for modification, and their purpose.

Parameter	Types	Summary
Rate Limit	Upper, lower	Establishes the minimum/maximum heart rate allowed before the pacemaker initiates a corrective pulse.
Amplitude	Atrium, ventricle	Defines the strength of the electrical pulse delivered to the atrium or ventricle.
Pulse Width	Atrium, ventricle	Sets the electrical pulse duration for the atrium or ventricle.
Refractory Period	Atrial, ventricular	Specifies the minimum period after a heartbeat during which no new pulse can be initiated.
Sensitivity	Atrial, ventricular	Determines the threshold at which the pacemaker detects electrical signals from the heart.
Reaction Time	–	Defines the time required for the pacemaker to respond to changes in heart rhythm or input signals.
Recovery Time	–	Specifies the time taken for the pacemaker to reset to normal operating conditions after a disturbance.

Rate Factor	–	Modifies the pacing rate based on specific conditions or patient requirements, such as activity levels.
Atrioventricular Delay	–	Sets the delay between atrial and ventricular contractions to ensure synchronized heart activity.
Activity Threshold	–	Defines the level of activity required to trigger a response or change in pacing settings.

The units, acceptable ranges, and acceptable increments are summarized below.

Table 4.5.2.b: Tabular Expression for acceptable programmable parameters and their types, units, and ranges.

Programmable Parameter	Units	Type	Min	Max	Applicable Modes
Lower Rate Limit	bpm	Integer	30	175	All except OFF
Upper Rate Limit	bpm	Integer	50	175	All except OFF
Atrium Amplitude	V	Float	0.5	5	AOO, AAI, AOOR, AAIR, DDD, DDDR
Ventricle Amplitude	V	Float	0.5	5	VOO, VVI, VOOR, VVIR, DDD, DDDR
Atrium Pulse Width	ms	Integer	1	30	AOO, AAI, AOOR, AAIR, DDD, DDDR
Ventricle Pulse Width	ms	Integer	1	30	VOO, VVI, VOOR, VVIR, DDD, DDDR
Atrial Refractory Period	ms	Integer	150	500	AOO, AAI, AOOR, AAIR, DDD, DDDR
Ventricular Refractory Period	ms	Integer	150	500	VOO, VVI, VOOR, VVIR, DDD, DDDR
Atrial Sensitivity	V	Float	0	5	AAI, AAIR, DDD, DDDR
Ventricular Sensitivity	V	Float	0	5	VVI, VVIR, DDD, DDDR
Reaction Time	s	Integer	1	50	AOOR, AAIR, VOOR, VVIR, DDDR
Recovery Time	s	Integer	1	240	AOOR, AAIR, VOOR, VVIR, DDDR
Rate Factor	None	Integer	1	16	AOOR, AAIR, VOOR, VVIR, DDDR
Atrioventricular Delay	ms	Integer	30	300	DDD, DDDR
Activity Threshold	None	Nominal	Very Low	Very High	AOOR, AAIR, VOOR, VVIR, DDDR

The requirements for applying new parameters are summarized below.

Table 4.5.2.c: Tabular Expression for entering valid programmable parameters.

Condition	RL	AMP	PW	RFP	SN	RXNT	RCVT	RF	AVD	Result
User applies new parameters	Valid	Valid	Valid	Valid	Valid	Valid	Valid	Valid	Valid	Database successfully updated and parameters applied
Upper/lower rate limit out of range	Invalid	Any	Any	Any	Any	Any	Any	Any	Any	Error: Rate limit out of valid range
Atrium/ventricle amplitude out of range	Any	Invalid	Any	Any	Any	Any	Any	Any	Any	Error: Amplitude out of valid range
Atrium/ventricle pulse width out of range	Any	Any	Invali d	Any	Any	Any	Any	Any	Any	Error: Pulse width out of valid range
Atrial/ventricular refractory period out of range	Any	Any	Any	Invalid	Any	Any	Any	Any	Any	Error: refractory period out of valid range
Atrial/ventricular sensitivity out of range	Any	Any	Any	Any	Invalid	Any	Any	Any	Any	Error: sensitivity out of range
Reaction time out of rage	Any	Any	Any	Any	Any	Invalid	Any	Any	Any	Error: Reaction time out of range
Recovery time out of range	Any	Any	Any	Any	Any	Any	Invalid	Any	Any	Error: Recovery time out of range
Rate factor out of range	Any	Any	Any	Any	Any	Any	Any	Invalid	Any	Rate factor out of range
Atrioventricular delay out of range	Any	Any	Any	Any	Any	Any	Any	Any	Invalid	Error: AVD out of range
User clicks OFF and submits	Any	Any	Any	Any	Any	Any	Any	Any	Any	Mode switched to OFF
User clicks Disconnect	Any	Any	Any	Any	Any	Any	Any	Any	Any	Connection is terminated

Legend: RL is Rate Limit, AMP is amplitude, PW is pulse width, RFP is refractory period, SN is sensitivity, RXNT is reaction time, RCVT is recovery time, RF is rate factor, and AVD is atrioventricular delay.

4.5.3 Serial Integration Requirements

The DCM must establish a robust and efficient mechanism for serial communication to facilitate seamless interaction between the desktop application and the pacemaker hardware. The integration must support a custom UART protocol which is to be developed and will be

defined in the Serial Protocol Documentation. The system must ensure bidirectional communication for telemetry data retrieval, real-time electrogram updates, mode and parameter adjustments, and other critical interactions.

The user must initiate the serial communication explicitly through a manual connection process. This requires the DCM to support a handshake mechanism to establish a reliable connection, with the application only commencing data exchange upon a successful handshake. The handshake process must be user-driven to prevent unintended device interactions and to give users control over when communication begins.

Once connected, the system must implement a continuous polling mechanism to monitor the connection status. Poll requests should be sent at a configurable rate, ensuring the balance between timely detection of disconnection and minimizing resource load. The DCM must also handle poll responses asynchronously, resetting the disconnection timer with each valid response. If no response is received for a configurable time period, the system should terminate the connection gracefully and notify the user.

The DCM must support the transmission of parameter and mode change requests, triggered only when users explicitly submit changes via the interface. The application must transmit these requests even if the new parameters match the current configuration, providing a fail-safe mechanism to resynchronize the pacemaker and the DCM in cases of discrepancy.

Electrogram data must be streamed continuously after a successful connection and handshake, using a subscription-based model. The DCM should listen asynchronously for incoming electrogram packets, process them in real-time, and map them onto the live electrogram display without interruption. The data should also be incorporated into session metadata for reporting and analysis purposes.

Serial integration must prioritize data integrity and low latency. All incoming and outgoing data must be validated against protocol specifications to ensure consistency. The system should be designed for asynchronous operation to prevent blocking tasks or introducing bottlenecks, maintaining smooth performance even under high data loads.

In scenarios of termination, whether due to user action, device disconnection, or signal noise, the DCM must handle the event gracefully. It must update the UI to reflect the termination, log the disconnection event, and provide users with actionable feedback where applicable.

Finally, the serial integration must be scalable to support potential enhancements, such as multi-device communication or variable telemetry rates. It must be robust enough to handle future protocol changes while maintaining its core functionality. These requirements aim to ensure a reliable and user-friendly communication interface that meets the demands of a medical application.

4.5.4 Electrogram Requirements

The real-time electrogram must provide a dynamic and continuous graphical representation of the atrial and ventricular voltage data received from the pacemaker via UART serial communication. The graph should plot voltage on the y-axis and time on the x-axis, ensuring a clear and comprehensible display of cardiac electrical activity. It should dynamically update to reflect the most recent telemetry data, maintaining a minimal latency to deliver near-instantaneous feedback.

The electrogram must support a variable sampling frequency and accommodate a variable number of points widthwise. It should be capable of showing electrogram data in two series, displaying both the atrial and ventricular signals at any given time, ensuring a resolution that is high enough to render smooth, detailed waveforms without overwhelming the program's processing capabilities. The graph should depict heartbeat patterns, pacing spikes, and voltage trends, enabling users to identify rhythm irregularities or unexpected behaviours.

The design must prioritize usability and adaptability. The interface should be responsive, automatically adjusting to different screen sizes and resolutions while maintaining the clarity and readability of the waveforms. The electrogram must allow users to distinguish atrial and ventricular activity through visual differentiation, such as distinct colours or line styles. The graph should refresh seamlessly without noticeable flicker or delay.

Diagnostic support must be integrated into the electrogram, enabling users to detect anomalies such as pacing artifacts, irregular voltage spikes, or signal noise. While the specific method for anomaly detection is flexible, the system must present the data in a manner that facilitates quick identification of potential issues.

Additionally, the electrogram should be capable of functioning consistently across all pacemaker modes and parameters, regardless of the specific telemetry data being transmitted. This ensures a universal and reliable diagnostic tool for any pacemaker configuration. The implementation must be robust and efficient, leveraging the system's real-time processing capabilities to maintain performance even under varying telemetry conditions.

4.5.5 Reports Requirements

The DCM must provide a robust and user-friendly Reports feature that enables users to generate, view, and download detailed data summaries related to pacemaker operations and system interactions. Reports must be designed to serve diverse stakeholders, including clinicians, patients, and technicians, offering insights into telemetry data, system usage, and operational changes. Reports should be accessible through a dedicated "Reports" tab in the application and downloadable directly to the user's device in a standardized and secure CSV format. The filename convention must include the user's username and a descriptive report name in snake case to enhance traceability and reduce the risk of miscommunication or mismanagement of data.

Each report must focus on a specific aspect of the system's operation. The Serial Log Report must comprehensively document all UART serial communication events during active sessions, including timestamps, transmitted data, and headers. This is essential for debugging and verifying protocol compliance. The Electrogram Report must offer a detailed tabular record of atrial and ventricular voltage readings since login, complete with timestamps and session metadata, providing clinicians with a tool for detailed diagnostic analysis.

The Activity Log Report must maintain a complete history of user login events, including usernames and timestamps, starting from user registration. This ensures long-term accountability and facilitates auditing and traceability. Similarly, the Parameter Log Report must track all pacemaker setting adjustments, recording timestamps, previous and updated parameter values, and the applied modes. This log ensures clinicians can review historical configurations, confirm adherence to medical guidelines, and identify trends or anomalies.

All reports must prioritize accuracy, security, and accessibility, supporting clinicians and technicians in making informed decisions while offering patients a clear view of their device's operation. Reports should also be designed for scalability, allowing for future enhancements, such as additional report types or integration with external analytics systems, without disrupting existing functionality.

4.6 Complete and Expected Changes

Due to the finite scope of this product, there will inevitably be requirement changes in future iterations of the product. Many of these requirements can be predicted. The expected changes for each of the DCM subsystems are outlined below.

4.6.1 Completed Changes

The DCM's user interface (UI) was initially designed to handle basic functionality, such as user input for programmable parameters for four pacemaker modes, saving and loading these parameters, and login and registration. However, significant enhancements and new features have been implemented during the project's progression.

One of the key changes was the addition of new pacemaker modes. This required extending the current UI to accommodate additional modes and their associated programmable parameters. The expansion also necessitated an overhaul of the main process logic to seamlessly integrate these new modes into the DCM's operation.

Another major enhancement was the inclusion of real-time pacemaker telemetry in the DCM. This allows users to view live data, such as a continuously updated electrogram and averaged values like heart rate. To achieve this, graphing functionality was implemented, enabling the visualization of critical values such as cardiac muscle potential, pacemaker output voltage, and sensed voltages. This real-time telemetry capability required developing a robust module to read data from the pacemaker via serial input and output. Since the pacemaker and

DCM share the same communication protocol, this feature had to be developed in tandem with the pacemaker to ensure compatibility.

The DCM was also improved to better handle connection status and failure modes. Previously, the UI included only a placeholder for the connection indicator. This was replaced with a dynamic display that reflects telemetry status, including user-initiated termination, physical disconnection, or signal noise exceeding acceptable levels. To support these functionalities, separate modules were implemented alongside the serial integration module. These modules handle tasks such as noise filtering, hardware identification, parameter monitoring, and manual termination.

Several UI elements were refined or added to enhance the user experience. For instance, the Reports feature was introduced, allowing users to generate and download summaries of telemetry data, user activity logs, and parameter adjustments. Additionally, data squares were added to the dashboard, displaying critical real-time metrics such as heart BPM, current pacemaker mode, and telemetry rate.

Together, these changes significantly enhance the DCM's functionality, usability, and ability to provide comprehensive insights into pacemaker operation, aligning with both clinical needs and user expectations.

4.6.2 Expected Changes

In future iterations, the DCM will need to evolve to meet new requirements and further enhance its functionality. Key anticipated changes include:

One significant update involves enabling each user to interface with multiple pacemakers, allowing the creation of individual profiles for each device. Currently, DCM is one pacemaker per user. This new feature will require several UI changes, including an interface for switching between pacemaker profiles and another for creating new profiles. Supporting multiple pacemakers will necessitate an overhaul of user registration and data-saving modules. The current schema, which ties a single pacemaker to each user, must be restructured to store and manage multiple profiles under a single user account.

The increase in system complexity due to multiple pacemaker profiles also means that the current method of data storage in the renderer and main processes will need a comprehensive overhaul. A more robust data storage solution will be required, ensuring seamless retrieval and management of profile data through improved APIs. Additionally, client-side data storage methods must be enhanced to handle the new requirements efficiently and securely.

Another future enhancement involves localizing the DCM for specific regions and languages. This feature will allow users from diverse linguistic backgrounds to interact with the application in their preferred language. Implementing localization will require the UI to support multiple languages and regional customizations. The development process will need to integrate

internationalization frameworks and manage dynamic content translations while maintaining a consistent user experience across all supported languages.

A potential future improvement involves introducing a variable telemetry rate, defined during the handshake between the DCM and the pacemaker. This feature would provide users with greater flexibility, allowing them to adjust the telemetry rate to match their specific needs or address limitations in the pacemaker's capabilities.

Visualizing direct muscle potential and/or pacemaker output is another future improvement that can be implemented. As of now, the pacemaker only visualizes the voltage sensed. Visualizing pacemaker output will provide the healthcare provider with even more data to be able to care for the patient and provide a greater insight into the pacemaker’s performance and functionality.

Lastly, enhancements to the Reports feature are anticipated, such as incorporating flags to highlight abnormalities, such as low heart rates or other significant events. These flags will improve report usability by drawing attention to critical data points, and aiding clinicians, technicians, and patients in their analyses.

These planned changes aim to expand the DCM's versatility, improve its user interface, and enhance its functionality to meet the evolving needs of its users. Through these developments, the DCM will become a more robust and accessible tool for managing pacemaker devices.

5 Design

The Design section outlines the overall structure, functionality, and behaviour of the DCM pages and its various components. It details how each element is designed to enhance user interaction and streamline functionality, ensuring a seamless experience for users managing their pacemaker settings. Furthermore, this section justifies design decisions made with a focus on usability and practicality, emphasizing how these choices facilitate efficient navigation, accurate data handling, and effective device control.

5.1 Overall Design

The overall design of the DCM is centred around a modular architecture that promotes efficient organization and behaviour across its various components. Each page and feature is structured to perform specific functions, allowing users to intuitively manage their pacemaker settings.

The DCM pages are organized into distinct sections, each responsible for a particular aspect of application and device control, such as telemetry monitoring, parameter adjustment, and user account management. This modular approach enables easy updates and

maintenance, as individual components can be modified or replaced without impacting the entire system.

Behaviorally, the application is designed to respond promptly to user inputs, with clear feedback mechanisms in place. For instance, actions like submitting changes or logging in trigger immediate system responses, enhancing user confidence and engagement. The application also maintains state persistence, ensuring that users can navigate seamlessly between different functionalities without losing their progress or data.

Overall, the design prioritizes an organized and behaviour-driven user experience, ensuring that all elements work harmoniously to facilitate efficient interaction with the pacemaker. The dashboard GUI can be seen below, encompassing the application's interface style and display.

5.1.1 Design Considerations

Several key considerations influenced the design of the HeartFlow application to ensure it meets both functionality and usability requirements. First, user experience was prioritized by designing intuitive navigation and clear interface layouts, minimizing the need for extensive training or technical expertise. The home page is simple and direct, guiding users to either log in or register, ensuring easy access from the beginning. Security considerations were also crucial, leading to password hiding, encryption, and strength validation, ensuring user data protection.

For the DCM Dashboard, real-time feedback was emphasized through the use of visual elements such as telemetry status indicators and heart-beat animations to confirm successful device connectivity. The design ensures that key actions, like adjusting pacemaker parameters or changing modes, are straightforward, with submit and discard buttons to confirm or revert changes easily.

Furthermore, continuity was maintained by automatically recalling previous configurations upon login, allowing users to quickly return to their preferred settings. The use of toast notifications provides non-intrusive feedback, ensuring that important updates, errors, or successful actions are communicated without disrupting workflow. Finally, the system ensures data integrity by updating the database upon submission and telemetry termination, even if the program exits unexpectedly, preventing data loss. These considerations create a user-focused design that prioritizes both ease of use and reliability.

5.2 Page Navigation

The Page Navigation system in HeartFlow is designed for intuitive and efficient movement between different sections of the application. Users are guided seamlessly from one page to another, ensuring that key functionalities such as logging in, registering, adjusting pacemaker parameters, and viewing telemetry status are easily accessible.

The navigation structure follows a clear hierarchy, beginning with the Home page, which offers straightforward access to the Login and Registration pages. Once logged in, users are directed to the DCM Dashboard, the primary interface for controlling and monitoring their pacemaker. The structure is highlighted below in a page flow diagram.

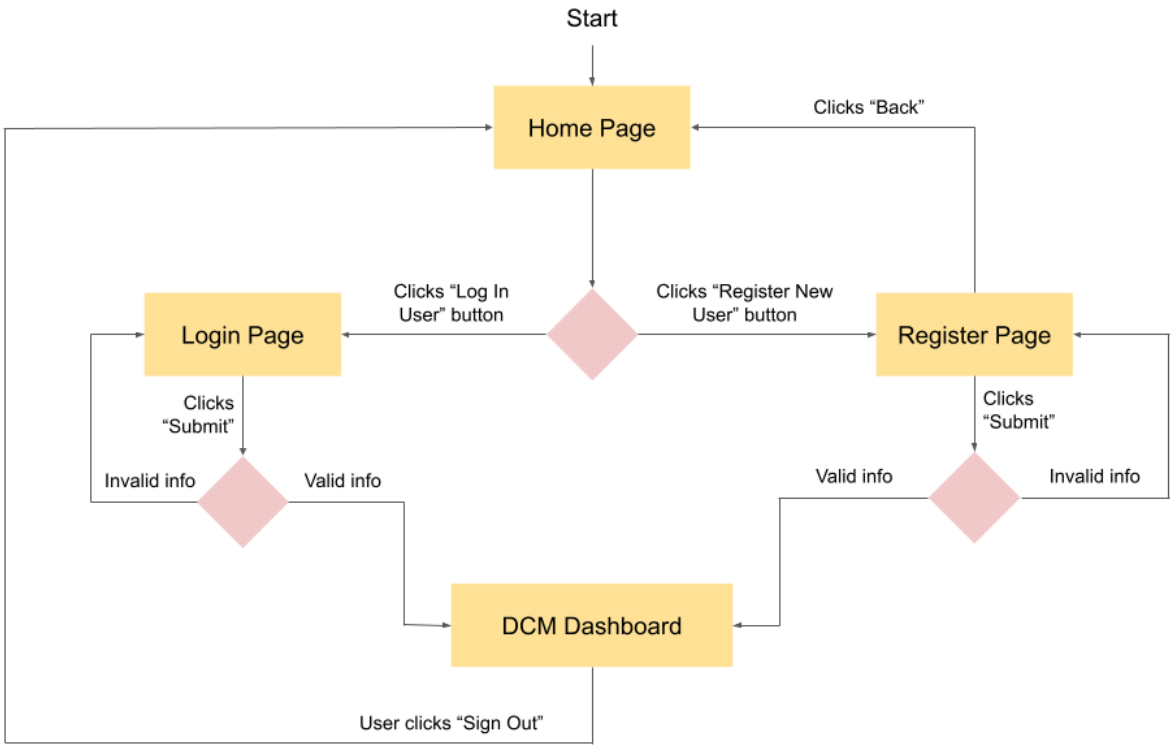

Figure 5.2.a: Page flow diagram for the HeartFlow application.

Navigation is supported by consistent button placement and labelling across the application, such as "Back" buttons for returning to previous screens and "Submit" buttons for confirming actions. This design ensures a logical flow, minimizing the learning curve for users and making interactions as efficient as possible.

Special attention has been given to the navigation flow between critical functionalities, such as transitioning from telemetry monitoring to parameter adjustment so that users can interact with the pacemaker smoothly without unnecessary steps or confusion. The simplicity and clarity of the navigation system are key to maintaining user focus on the essential task of managing their pacemaker settings.

5.3 Home Page

The home page of the HeartFlow desktop application serves as the entry point for users, as it is the page open upon the program's launch. Its design focuses on simplicity and clarity, ensuring that both returning users and first-time users can navigate the system effortlessly.

Users are greeted with a welcoming message, positioned prominently underneath the HeartFlow logo, which reinforces the application's branding. Below the logo and welcoming message is a brief description outlining the goal of the application. At the bottom of the page, the version number of the application (e.g., "HeartFlow v2.0.0") is displayed. This small but crucial detail allows users to easily identify the current version they are using, which is essential for technical support and updates.

Directly beneath the welcome message are two interactive buttons:

- **Login User**: This button directs returning users to the login page, where they can enter their credentials to access the application.
- **Register New User**: This button is for first-time users who have not yet created an account. It directs them to the registration page, which collects necessary user information and credentials for future access.

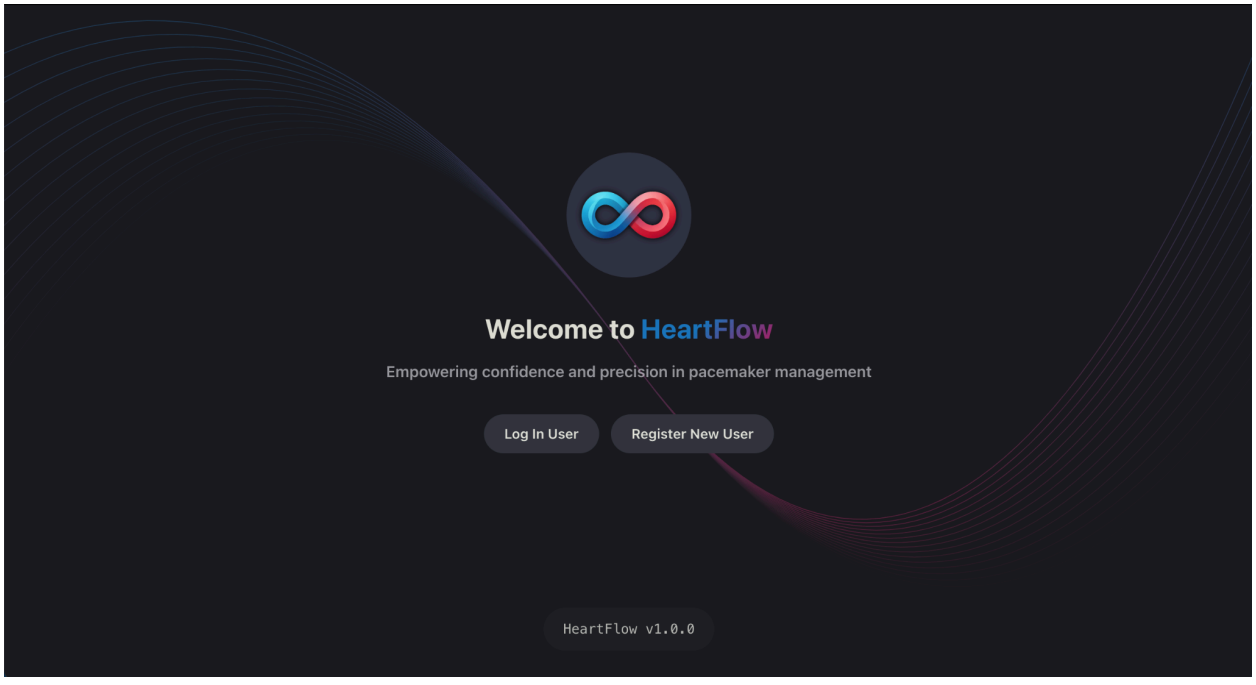

Figure 5.3.a: Screen capture of HeartFlow's home page.

5.4 Registration Page

The User Registration page in the HeartFlow desktop application is designed with a focus on user security and a smooth onboarding experience. The page allows new users to create an account by providing essential information. It follows a simple, intuitive layout to guide users through the registration process efficiently.

The page contains four input fields. Errors in any of these fields are specified by Toast messages. The specific purposes of each input field are as follows:

- **Username**: This field requires users to create a unique identifier for their account. To maintain a minimum standard for username length, it must be at least 3 characters long. The system will check for uniqueness to prevent duplicate usernames, ensuring each user has a distinct account.
- **Password**: For security purposes, the password is typed using hidden characters (displayed as dots) to protect sensitive information during entry. The password must meet a minimum length of 8 characters and must include at least one special character, ensuring that users create strong passwords for account security.
- **Confirm Password**: This field asks users to re-enter their password to ensure that there are no typographical errors during registration. The system will verify that the entries in the password and confirm password fields match before proceeding.
- **Pacemaker Serial Number**: As HeartFlow is designed to interface with the user's pacemaker, the system requires the serial number of the pacemaker. This field ensures that the user's pacemaker is correctly identified. While there are no specific format constraints for this field, it cannot be left empty (null).

In the supplement, there are two buttons:

- **Register**: Once all fields are filled out correctly, this button allows users to submit their information and be added to the local database. Upon activation of this button, the input fields are validated according to their respective validation criteria as indicated above. For privacy and security, the password is encrypted before it is stored, ensuring that user credentials are protected from unauthorized access.
- **Back**: This button provides users the option to return to the home page, in case they decide not to proceed with registration or need to review other options.

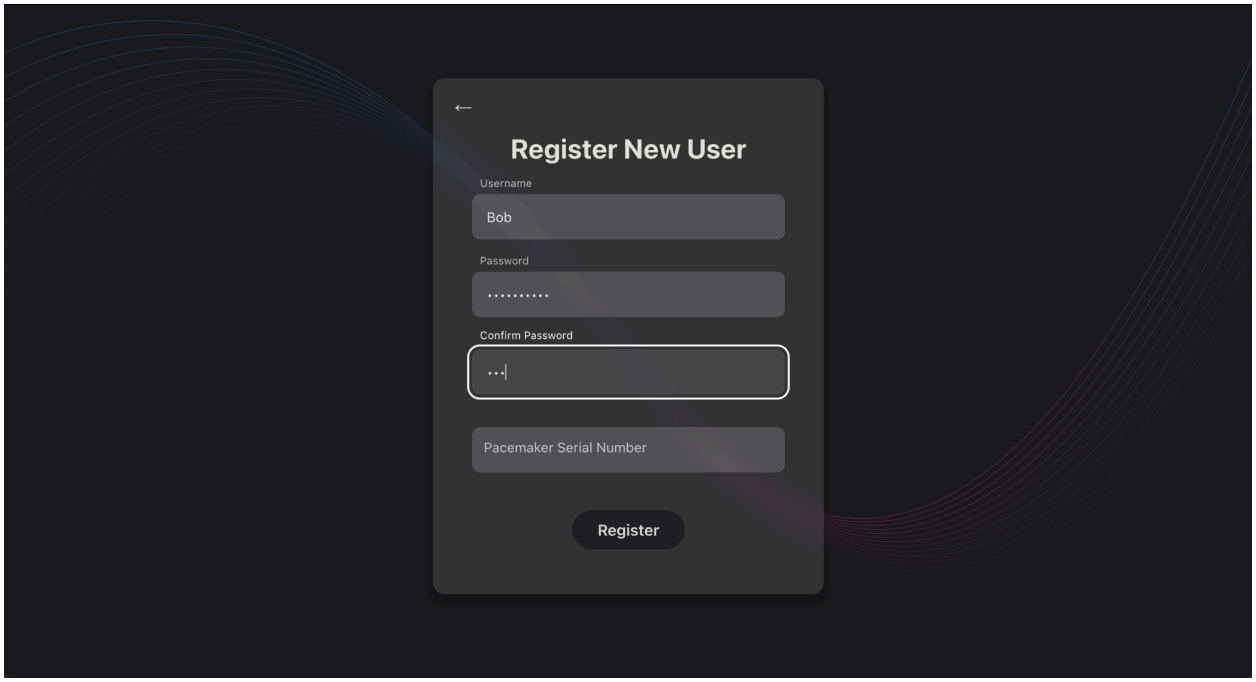

Figure 5.4.a: HeartFlow registration page graphical design.

5.5 Login Page

The user login page in HeartFlow provides a secure and streamlined interface for users who have already registered an account to access their pacemaker controls. This page focuses on simplicity and security, allowing users to safely log in to be directed to the dashboard under their account. In the log-in screen, a centred widget exists with two input fields:

- **Username**: Users are required to enter the unique username they created during the registration process. This field ensures the system can correctly identify the user and pull their stored information from the local database.
- **Password**: For enhanced security, the password is entered in a hidden format, with each character represented by dots. This prevents onlookers from viewing the password as it is typed. The system will match the password against the one stored for the provided username to ensure the credentials are correct.

Beneath the input fields are two buttons:

- **Log In**: This button completes the login process barring any field validation errors. It first checks whether the entered username exists in the system, and then verifies that the associated password matches the one on file. If both are correct, the user is granted access to the application's main features.
- **Back**: This button allows users to return to the home page without attempting to log in, in case they decide not to proceed or need to register a new account.

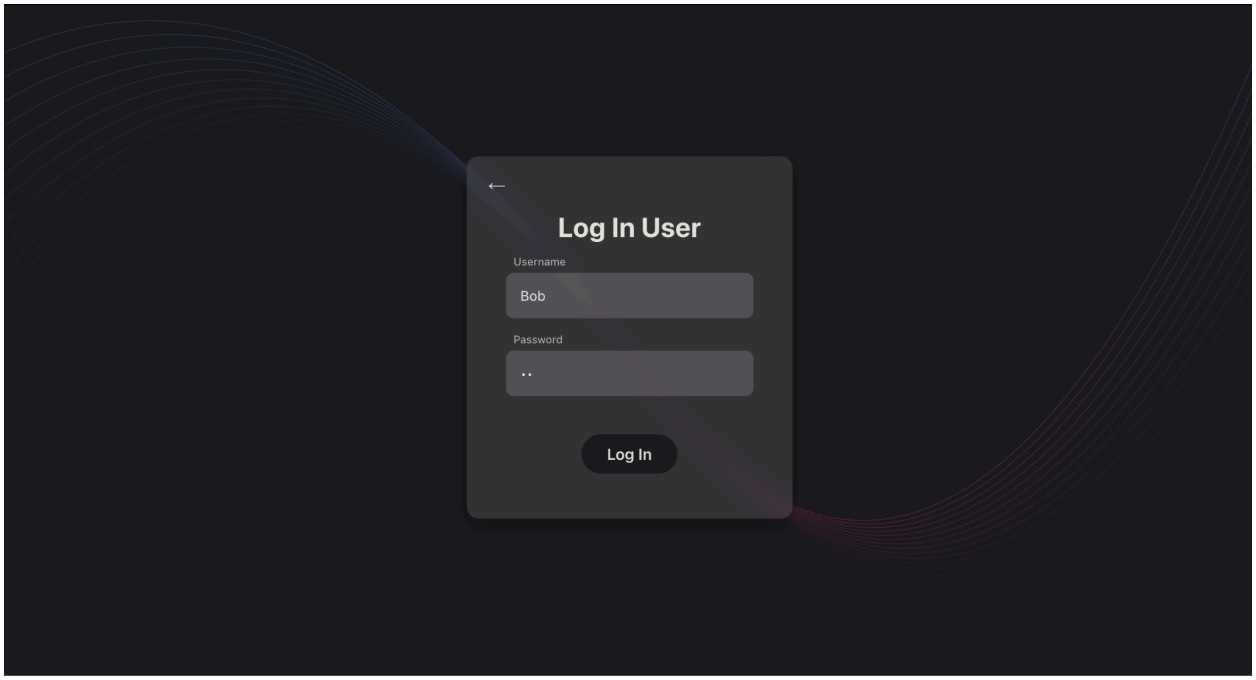

Figure 5.5.a: HeartFlow login page graphical design.

5.6 DCM Dashboard

The DCM dashboard serves as the central hub of the HeartFlow application, providing users with comprehensive access to pacemaker control features. The design emphasizes clarity and functionality, ensuring that users can easily monitor and adjust their pacemaker settings while maintaining a focus on usability and real-time interaction.

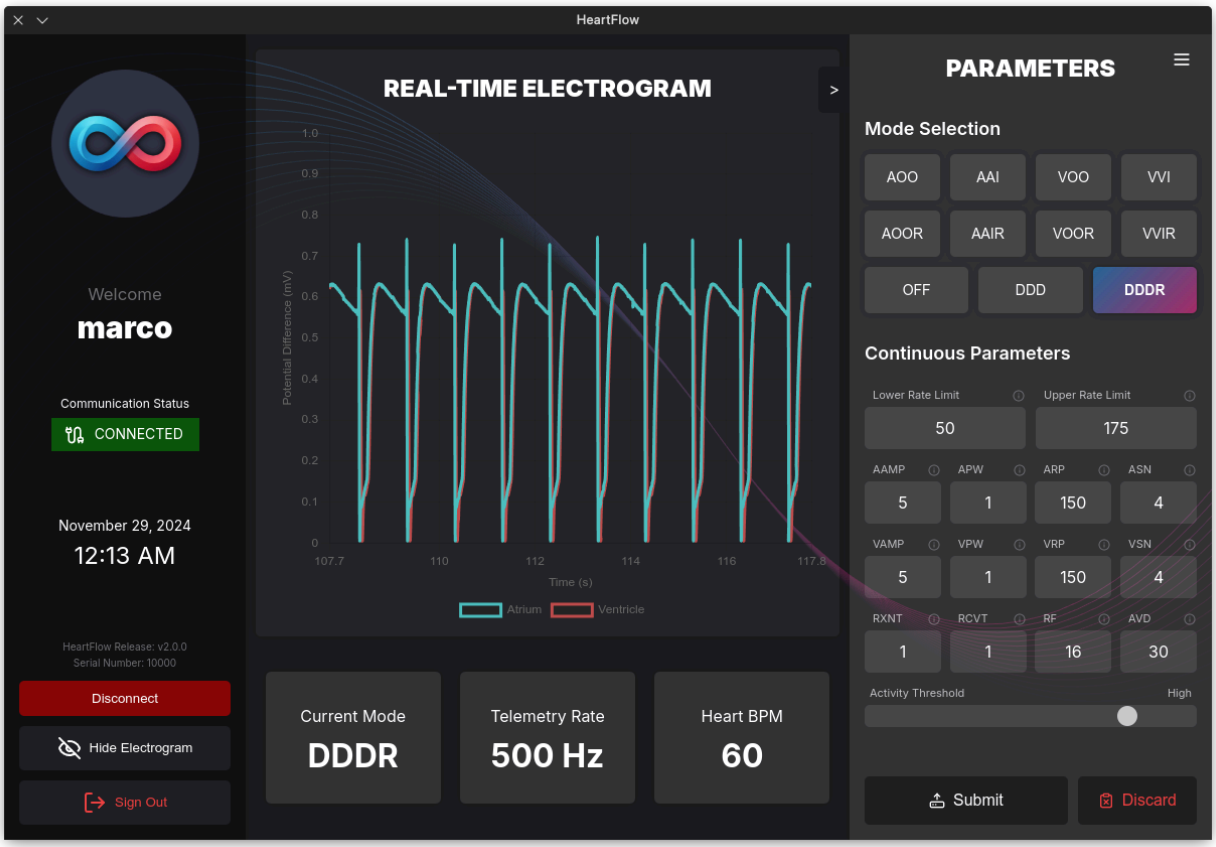

Figure 5.6.a: HeartFlow dashboard graphical design with mode DDDR selected and active pacing.

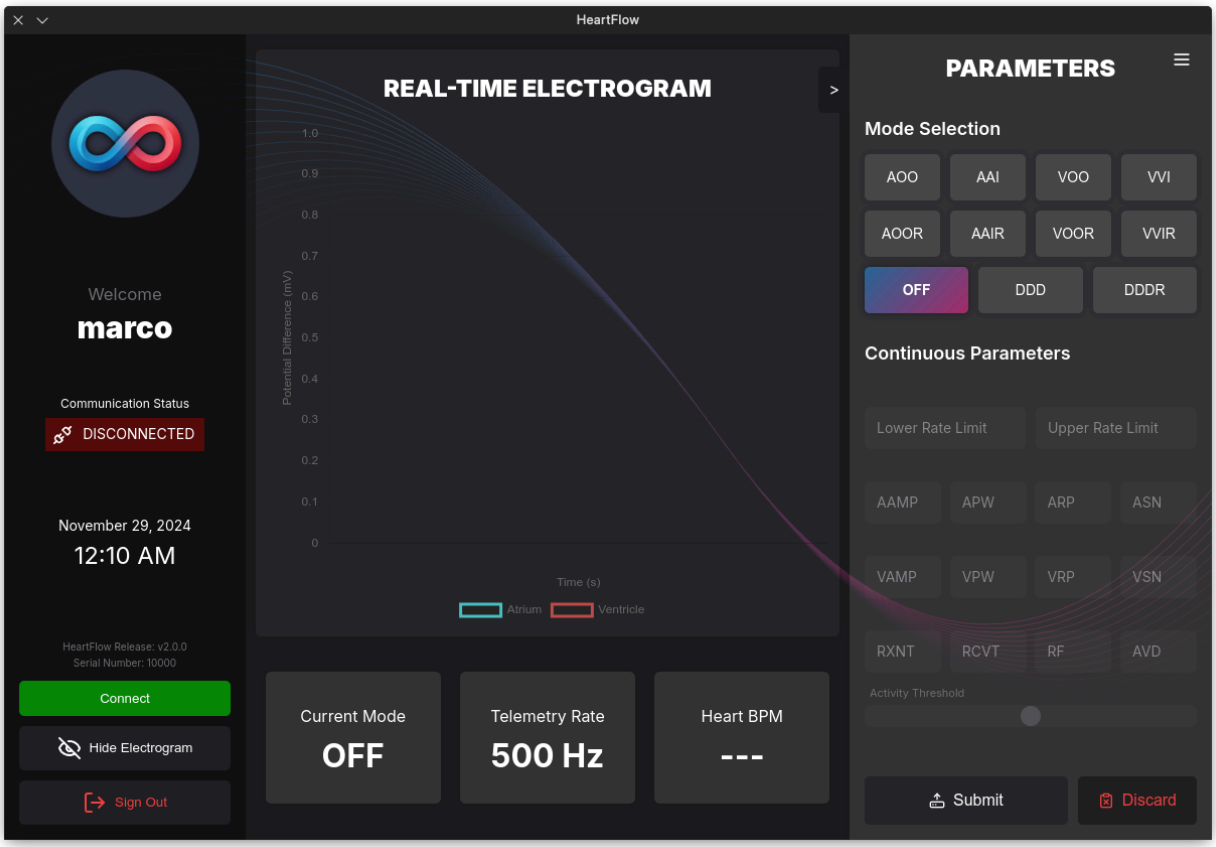

Figure 5.6.b: HeartFlow dashboard graphic design before the user manually initiates handshake.

5.6.1 Graphical Layout

The dashboard features a structured layout with a left sidebar and a right sidebar, complemented by a central display area. The left sidebar prominently features the company logo, along with the logged-in user's name, enabling users to quickly identify their session. Below the username, the current telemetry status is displayed, indicating whether the system is connected or disconnected. The sidebar also shows the current date and time, along with the HeartFlow release and the pacemaker's serial number, rendered in a minor text styling for easy reference. For user convenience, a 'Sign Out' button is available to return to the home page, and a 'Hide Electrogram' button is positioned above to hide the electrogram and show the live heart.

The right sidebar, titled "Parameters," houses mode selection widgets that allow users to switch between different operational modes including OFF. Each mode is equipped with corresponding adjustable fields for continuous parameters, which are intelligently disabled or enabled based on the selected mode. At the bottom of this sidebar, users can find the 'Submit' and 'Discard' buttons for managing their changes. An information button located at the top right opens a panel displaying descriptions and units for all options, enhancing user understanding of

each parameter. At the top right, the user can toggle to view the reports they have the option of downloading.

In the center of the dashboard is the electrogram, which can be toggled for a visual representation of a heart with an animation for the heating heart from the pacemaker. Both of these options, the heart animation and the electrogram, dynamically reflect the current telemetry status. Additionally, three widgets are situated prominently in this central area, indicating the current mode, the telemetry sampling frequency, and the computed BPM of the heart from the telemetry data.

5.6.2 Communication Status and Control

Communication functionality is a critical aspect of the DCM dashboard, allowing real-time bidirectional data transmission between the pacemaker and the application. The design includes straightforward controls to start and stop telemetry by initiating and terminating the connection between the hardware and software, ensuring that users can engage or disengage with their device with minimal friction. The “Connect” button on the left sidebar of the dashboard permits the user to initiate the 0x01 serial handshake process which establishes a connection with the pacemaker. The left sidebar has a label that displays the current communication status, which can fulfil one of three states; “Connected”, “Disconnected”, and “Reconnecting”.

The “Connected” state indicates that add serial communication is operating as expected by the UART serial protocol and that the pacemaker is actively and accurately responding to all poll requests. The “Disconnected” state is active by default upon logging in to provide the user with the decision of when to initiate telemetry and communication. It also becomes active when the user clicks the “Disconnect” button while the connection is active, or when the DCM does not receive a poll response after 1000 consecutive poll requests; a time equivalent to 10 seconds. “Reconnecting” is defined as this intermediate state in which the user does not manually terminate the connection but the pacemaker has failed to respond to the most recent poll request. More justification for this design decision is provided in this document in the Serial Integration section.

5.6.3 Real-Time Electrogram

The real-time electrogram is a core feature of the DCM, providing users with a continuous graphical representation of the heart's electrical activity regardless of the pacemaker mode or parameters. Positioned prominently in the central panel of the dashboard, this component is designed to offer instant feedback on the atrial and ventricular voltage readings read and received from the pacemaker. The screen capture above demonstrates the design of the real-time electrogram.

The graph displays voltage on the y-axis and time on the x-axis, updating dynamically to reflect the most recent telemetry data transmitted via UART serial communication. The electrogram currently shows a sampling frequency of 500 Hz corresponding to a datapoint every 2

milliseconds and shows 5,000 points for both the atrium and ventricle at once across its width. This decision was justified by ensuring a sufficient resolution to render a smooth and sensible waveform while maintaining a smooth running process of the program. Each heartbeat is visually represented through waveform patterns corresponding to atrial and ventricular activity, enabling users to observe rhythm trends, pacing spikes, and overall heart function.

Besides visualizing the heart's electrical activity, the real-time electrogram supports diagnostic and troubleshooting workflows. Anomalies such as irregular pacing patterns, signal artifacts, or unexpected voltage levels can be identified and addressed promptly. To enhance usability, the electrogram is designed with a clean, responsive interface that adjusts to various screen sizes and resolutions without compromising detail or readability.

The electrogram leverages the system's real-time data handling capabilities, processing incoming telemetry at high speed to maintain minimal latency. This ensures that users can rely on the display for up-to-date insights into pacemaker performance and cardiac activity, making the real-time electrogram an indispensable tool for both clinical and personal device management.

5.6.4 Changing Modes and Programmable Parameters

The DCM dashboard provides a straightforward interface for changing modes and writing parameter values for the pacemaker. The mode selection widgets allow users to toggle between various modes offered: AOO, AAI, VOO, VVI, AOOR, AAIR, VOOR, VVIR, DDD, DDDR, and OFF, depending on the desired operational settings. The design decision to present these options as distinct widgets enables users to quickly switch between modes while maintaining focus on the parameter adjustments relevant to their current settings. Besides mode selection, the dashboard includes programmable parameters that users can adjust. These parameters are crucial for tailoring the pacemaker's operation to meet individual patient needs. However, the fields for these parameters are enabled or disabled based on the current mode to prevent conflicting settings and for the user's simplicity by eliminating the inclusion of parameters that are meaningless for some modes.

5.6.4.1 Programmable Parameters

The DCM dashboard offers a range of programmable parameters that enable users to fine-tune the operation of the pacemaker to meet specific patient needs. These parameters are critical for optimizing cardiac function and ensuring effective pacing. The following programmable parameters are available:

- **Lower Rate Limit (LRL)**: This parameter establishes the minimum heart rate at which the pacemaker will deliver impulses. By setting an appropriate lower rate limit, users can prevent bradycardia and ensure the heart keeps a proper rhythm.
- **Upper Rate Limit (URL)**: The Upper Rate Limit defines the maximum heart rate at which the pacemaker will deliver impulses. This parameter ensures that the pacemaker

does not overstimulate the heart, preventing conditions such as tachycardia while maintaining appropriate cardiac function during periods of increased activity.

- **Atrium Amplitude (AAMP)**: This parameter controls the strength of the electrical impulses delivered to the atrium. Users can adjust the amplitude to ensure effective stimulation of the atrial myocardium, helping maintain a consistent heart rhythm.
- **Atrium Pulse Width (APW)**: This parameter determines the duration of the electrical pulse delivered to the atrium. By adjusting the pulse width, users can optimize the timing of the atrial stimulation to improve cardiac efficiency and responsiveness.
- **Atrial Refractory Period (ARP)**: The atrial refractory period defines the time during which the atrium is unresponsive to further stimulation following a pacing pulse. Adjusting this parameter helps prevent rapid pacing and allows for effective heart rhythm management.
- **Atrial Sensitivity (ASN)**: Atrial Sensitivity determines the threshold at which the pacemaker detects electrical activity in the atrium. Adjusting this parameter allows the device to appropriately sense natural atrial activity, ensuring proper pacing synchronization and avoiding unnecessary impulses.
- **Ventricle Amplitude (VAMP)**: Similar to the atrium amplitude, this parameter regulates the electrical impulses delivered to the ventricle. Adjusting the ventricle amplitude is essential for achieving optimal ventricular contraction and ensuring sufficient blood flow.
- **Ventricle Pulse Width (VPW)**: This parameter sets the pulse duration for the ventricular stimulation. Correctly configuring the ventricle pulse width is crucial for ensuring that the electrical impulses are effective in triggering proper ventricular contractions.
- **Ventricular Refractory Period (VRP)**: Similar to the atrial refractory period, this parameter specifies the time frame during which the ventricle cannot be stimulated after receiving a pulse. Properly configuring the ventricular refractory period is essential for avoiding arrhythmias and ensuring effective ventricular filling.
- **Ventricular Sensitivity (VSN)**: Ventricular Sensitivity specifies the detection threshold for electrical signals in the ventricle. Proper configuration of this parameter ensures the pacemaker can accurately sense ventricular activity, enabling effective pacing and preventing inappropriate or missed stimuli.
- **Reaction Time (RXNT)**: Reaction Time defines how quickly the pacemaker adjusts to changes in heart rate demand, such as during exercise or stress. A shorter reaction time ensures faster adaptation to increased activity, while a longer reaction time prevents abrupt pacing changes that could affect comfort or stability.
- **Recovery Time (RCVT)**: Recovery Time sets the duration required for the pacemaker to return to a resting rate after periods of increased pacing due to physical activity or stress. This parameter ensures a smooth transition, avoiding sudden drops in pacing that might cause discomfort or dizziness.
- **Rate Factor (RF)**: The Rate Factor is a multiplier that determines how the pacemaker adjusts the pacing rate in response to detected activity levels. A higher rate factor increases physical activity sensitivity, leading to more significant changes in heart rate. A lower rate factor dampens these adjustments for more gradual pacing changes.
- **Atrioventricular Delay (AVD)**: The Atrioventricular Delay specifies the time interval between atrial and ventricular pacing. Adjusting this parameter ensures proper

coordination between atrial and ventricular contractions, optimizing blood flow and overall cardiac efficiency.

- **Activity Threshold (ACT)**: Activity Threshold defines the level of physical activity required to trigger a rate response from the pacemaker. A lower threshold makes the pacemaker more responsive to subtle movements, while a higher threshold requires more vigorous activity to adjust the pacing rate. This parameter helps tailor pacing to the user's lifestyle and activity patterns.

Each of these parameters is adjustable through the DCM dashboard, allowing users to tailor the pacemaker's operation according to clinical requirements and patient responses.

5.6.4.2 Parameter Validation

To maintain system integrity, the DCM dashboard includes robust parameter validation mechanisms. Only valid float values are accepted for parameter input, ensuring accurate data representation. Only numerical characters and decimals are permitted to be entered in the input fields. Furthermore, specific ranges are defined for each parameter to guide users in making appropriate selections. Fields that are not applicable based on the selected mode are disabled, preventing erroneous input. Programmable settings and their validations are detailed below.

Table 5.6.4.2.a: Summary of programmable parameters and their valid ranges.

Programmable Parameter	Units	Min	Max	Modes
Lower Rate Limit (LRL)	bpm	30	175	All except OFF
Upper Rate Limit (URL)	bpm	50	175	All except OFF
Atrium Amplitude (AAMP)	V	0.5	5	AOO, AAI, AOOR, AAIR, DDD, DDDR
Ventricle Amplitude (VAMP)	V	0.5	5	VOO, VVI, VOOR, VVIR, DDD, DDDR
Atrium Pulse Width (APW)	ms	1	30	AOO, AAI, AOOR, AAIR, DDD, DDDR
Ventricle Pulse Width (VPW)	ms	1	30	VOO, VVI, VOOR, VVIR, DDD, DDDR
Atrial Refractory Period (ARP)	ms	150	500	AOO, AAI, AOOR, AAIR, DDD, DDDR
Ventricular Refractory Period (VRP)	ms	150	500	VOO, VVI, VOOR, VVIR, DDD, DDDR
Atrial Sensitivity (ASN)	V	0	5	AAI, AAIR, DDD, DDDR
Ventricular Sensitivity (VSN)	V	0	5	VVI, VVIR, DDD, DDDR
Reaction Time (RXNT)	s	1	50	AOOR, AAIR, VOOR, VVIR, DDDR
Recovery Time (RCVT)	s	1	240	AOOR, AAIR, VOOR, VVIR, DDDR
Rate Factor (RF)	None	1	16	AOOR, AAIR, VOOR, VVIR, DDDR
Atrioventricular Delay (AVD)	ms	30	300	DDD, DDDR
Activity Threshold (ACT)	None	Very Low	Very High	AOOR, AAIR, VOOR, VVIR, DDDR

It is worth noting that the activity threshold does not take numerical values as inputs on the DCM. Instead, it is a slider, with the following options: Very Low, Low, Low Medium, Medium, Medium High, High, and Very High. Internally, these are transcribed and stored as integers and subsequently transmitted to the pacemaker. Further, there are two additional restrictions that are not tabulated above.

The first is that the URL must be greater than or equivalent to LRL. This is essential to ensure logical and physiological consistency in the pacemaker's operation. This condition prevents conflicting settings where the pacemaker would attempt to maintain a minimum heart rate that exceeds or matches the maximum permissible pacing rate. Such a conflict could cause erratic pacing behaviour, including failure to meet the lower rate demand or improper pacing suppression, which could lead to patient discomfort or compromised cardiac function. By enforcing this rule, the pacemaker can maintain a clear and coherent range of operation, ensuring that the heart rate remains within a safe and manageable spectrum under all conditions.

The second is that the sum of the atrial refractory period and the atrial pulse width must be less than the quotient of 60000 and the upper rate limit. This is crucial for maintaining proper pacing timing and ensuring the safety and efficiency of the pacemaker's operation. This condition ensures that the sum of the refractory period and pulse width does not exceed the time available for one cardiac cycle at the maximum pacing rate (determined by the URL). If this sum were too large, it could cause pacing events to overlap or occur too frequently, which may cause arrhythmias or ineffective heart contractions. By enforcing this constraint, the pacemaker ensures sufficient time for the heart to complete each beat cycle, including recovery and natural rhythm resumption while remaining responsive to the programmed upper rate limit. This safeguard helps optimize both cardiac efficiency and patient safety.

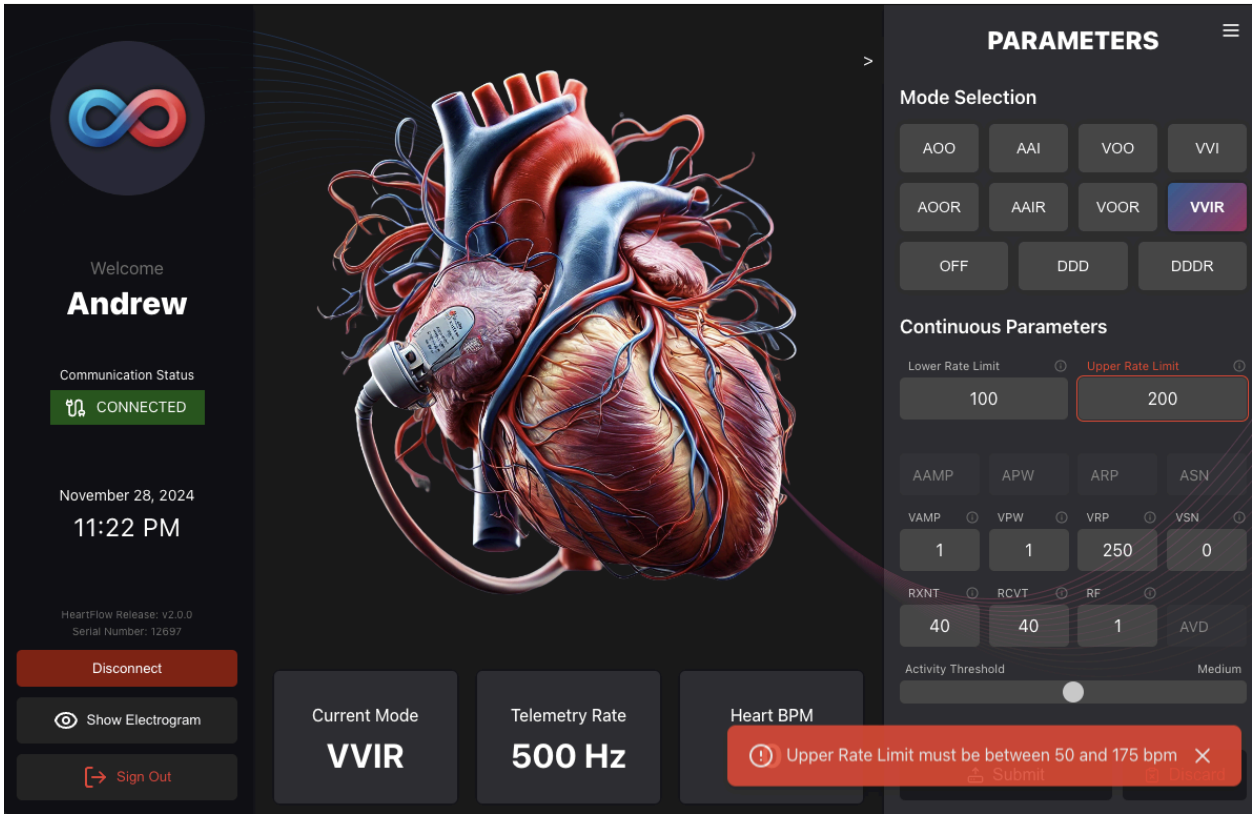

Figure 5.6.4.2.a: Example of the styling when an erroneous value is entered when attempting to change the URL to a value above 175 bpm, which is defined as the maximum URL. The valid range is specified to support users in correcting their inputs.

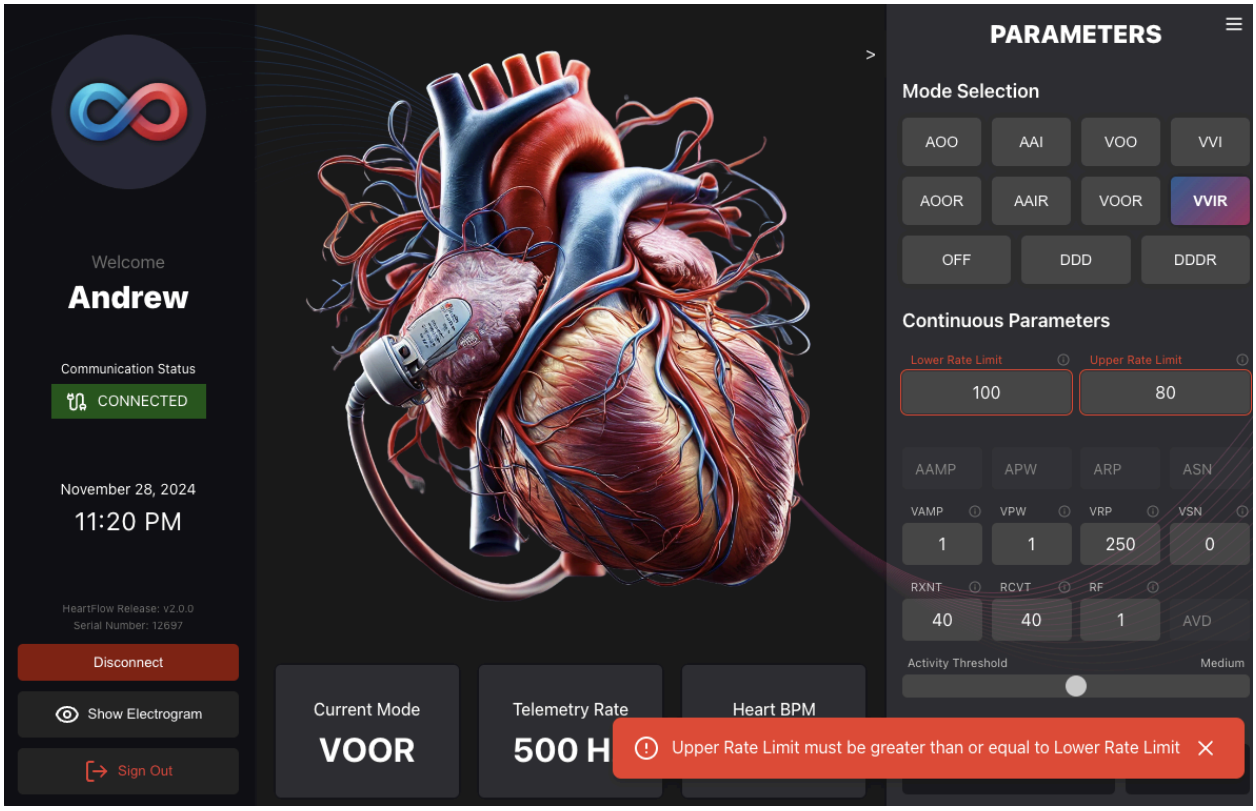

Figure 5.6.4.2.b: Example of the styling when an erroneous set of parameters is entered when attempting to change the URL to be less than the LRL, which is illegal. The condition broken is displayed for simpler user troubleshooting.

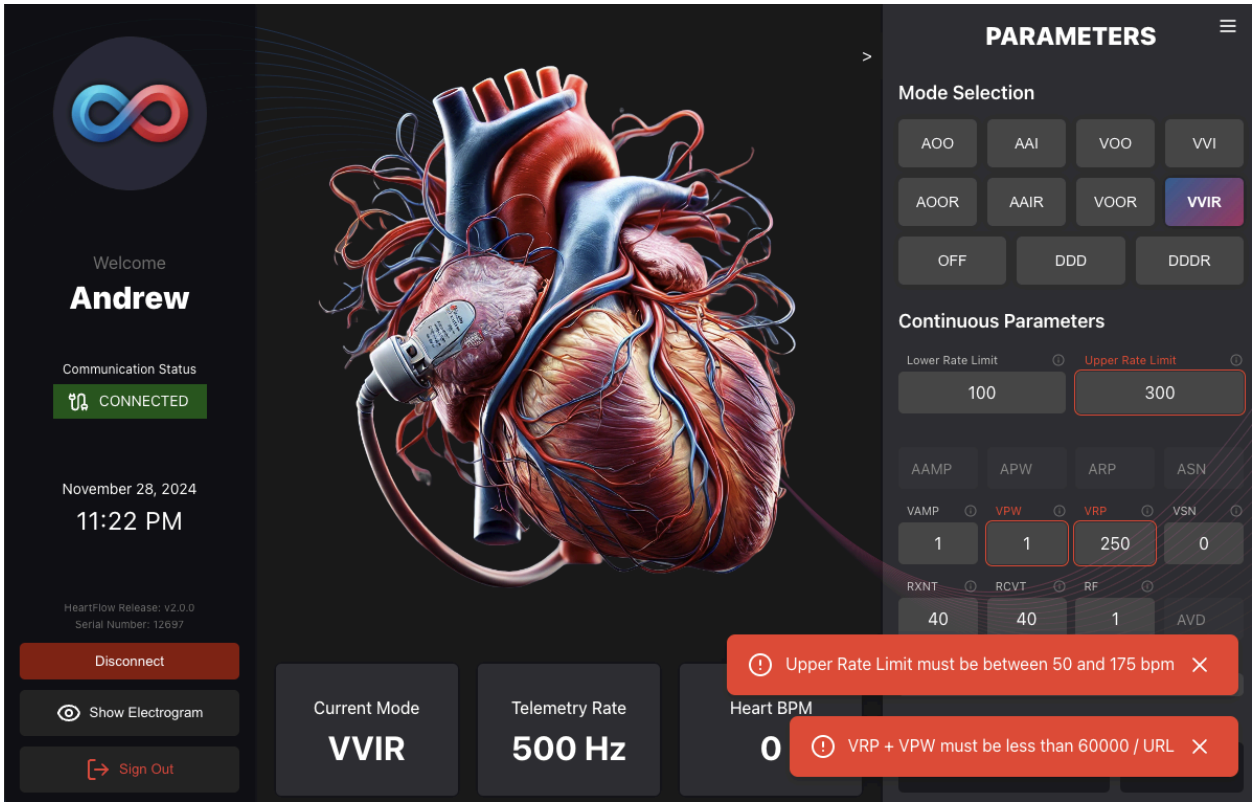

Figure 5.6.4.2.c: Example of the styling when an erroneous set of parameters are entered when attempting to change the URL to be greater than 60000 * (VRP + VPW), which is illegal. The condition broken is displayed for simpler user troubleshooting. It also shows that a URL as 300 bpm is invalid because its maximum legal value is 175 bpm.

5.6.4.3 Storing Data

When users are satisfied with the adjustments made to their pacemaker parameters, the "Submit" button facilitates the secure storage of these updates in the system's local database. This design decision is crucial for ensuring that the pacemaker operates according to the latest configurations set by the user, thus providing them with a sense of control and ownership over their therapy settings.

The application's front-end interacts with the database to retain both short-term and long-term memory of the user's configurations. This means that when users log into their account, the mode is automatically set to the one that was previously active during their last session. This approach promotes continuity in user experience, minimizing the need for repetitive adjustments and allowing users to quickly resume their preferred settings.

Every time the Submit button is clicked, the application checks the validity of the input values. Once confirmed as valid, the database is updated with the user's changes. This ensures that the pacemaker operates under the most recent and accurate settings, improving user confidence in the device's functionality. Furthermore, the database is also updated upon telemetry termination, ensuring that any last-minute changes are not lost.

Storing database interactions related to the previously active mode upon user submission rather than only during logout enhances the reliability of the system. This design consideration safeguards the configuration settings, allowing them to be saved even if the program is closed unexpectedly before the user logs out. This functionality is particularly important in healthcare contexts, where abrupt program exits can occur.

Moreover, when changing modes, the system is designed to display the default parameters that were last employed when that mode was active. This ensures continuity in user experience, allowing users to easily transition between modes without needing to re-enter familiar values.

On initial login, the application defaults the mode to OFF, with all parameter fields intentionally set to 0, which is an invalid entry for each field. This strategic choice is made to prompt users to modify the parameters to their preferred values before applying them to the HeartFlow. By implementing this design, the application effectively encourages user engagement with the parameter settings, ensuring that users take an active role in configuring their devices according to their specific needs.

5.6.4.4 Discarding Changes

If users decide not to proceed with their modifications before submission, the 'Discard' button provides a straightforward way to cancel any changes. Regardless of the selected mode, telemetry status, or entered programmable parameters, the discard button will return the three mentioned states to their previously saved states as defined in the file users.json. This feature helps users revert to the last saved settings without complications, ensuring a user-friendly experience within the DCM dashboard.

5.6.5 Reports

The Reports feature in the DCM allows users to generate, view, and download comprehensive data summaries related to pacemaker operation and user interactions. This section describes the design, functionality, and implementation of the Reports feature, focusing on its role in providing insights into device usage, telemetry, and parameter adjustments. The intended users for these reports are the clinicians using the application, patients who may be the recipients of the reports, and technicians attempting to troubleshoot the firmware, software, or serial communication system.

All reports are downloaded directly to the user’s downloads folder as a CSV file when they click on the associated report in the right sidebar under the “Reports” tab. All reports that are downloaded contain the user’s username, followed by the name of the report in the snake case. This improves system security so clinicians are less likely to mistakenly send their patients the wrong report.

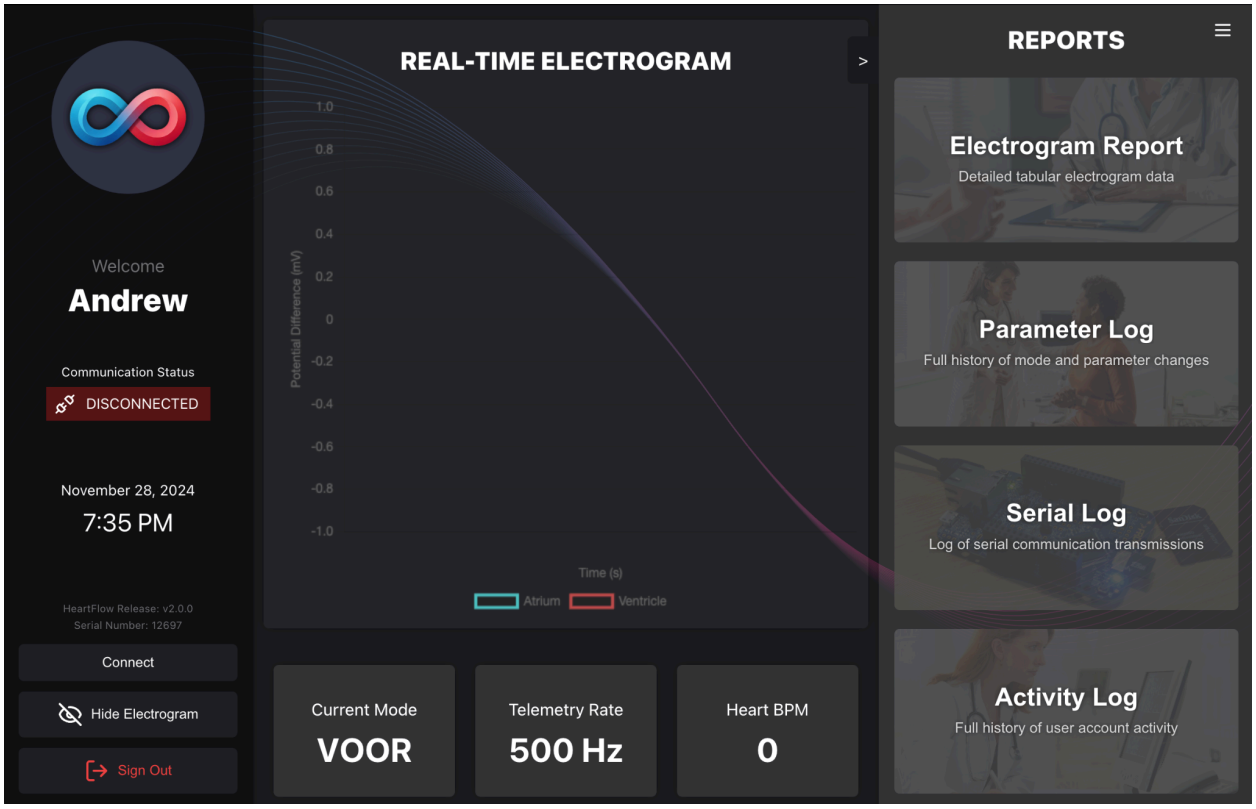

Figure 5.6.5.a: HeartFlow dashboard with the Reports menu active, showing the reports.

5.6.5.1 Serial Log Report

The Serial Log Report captures a detailed record of all UART serial communication between the DCM and the pacemaker hardware during the active session. It includes timestamps for each communication event, the data transmitted and received including the header bytes and the payload. This report is particularly valuable for debugging, enabling developers or technicians to trace communication issues and verify protocol compliance.

5.6.5.2 Electrogram Report

The Electrogram Report provides a comprehensive tabular record of voltage readings from the atrium and ventricle since login. This report includes timestamps, voltage levels, and metadata indicating session details. It serves as a vital diagnostic tool for clinicians or users seeking to analyze heart activity trends or identify abnormalities in the pacing, beyond what can be observed from the electrogram qualitatively.

5.6.5.3 Activity Log Report

The Activity Log Report documents all user login occurrences, providing a timeline of system access. Per session, the report logs the username and login time. This log aids in auditing system usage, ensuring accountability, and identifying potential unauthorized access. This contains login data since the registration of the user, providing continuity and a comprehensive

view of the user's interaction with the system. By maintaining a full history of login events, the Activity Log Report ensures traceability, enabling administrators or clinicians to review long-term access patterns and identify any irregularities.

5.6.5.4 Parameter Log Report

The Parameter Log Report tracks all modifications made to a user's pacemaking settings since the user's initial registration. It lists the time of the parameters' application to the pacemaker, the programmable parameters that were adjusted and their previous values, and the mode that the settings were changed to. This log is instrumental in tracking historical configurations and verifying compliance with clinical recommendations.

5.6.6 Data Tiles

At the bottom of the dashboard, three data tiles are prominently displayed: Current Mode, Telemetry Rate, and Heart BPM. These widgets provide users with critical, real-time information about the pacemaker's status and operation in a clear and accessible format.

The Current Mode tile displays the pacing mode currently in operation, which includes the OFF state. This ensures users can quickly verify the active configuration without navigating through additional settings, reducing potential confusion during critical adjustments or monitoring sessions.

The Telemetry Rate tile indicates the frequency at which the pacemaker is sending data to the DCM. By default, this is 500 Hz and must be agreed upon by both the DCM and Simulink. By displaying this rate, users can confirm that telemetry is functioning as expected and adjust settings if communication appears irregular. This transparency helps maintain confidence in the system's real-time data flow and operational integrity.

The Heart BPM tile provides the real-time heart rate. Using the telemetry data being acquired serially via UART, the DCM then computes the BPM by determining the number of times both the atrium and ventricle's electrical activity traverse a specified voltage threshold in both directions and dividing by time. This allows users to monitor the immediate effectiveness of the pacing and assess the heart's response to the programmed parameters. Any significant deviations from expected rates can prompt users to review and modify settings as necessary, ensuring optimal performance and patient safety. A warning system has been devised and is summarized below to alert users as early as possible when dangerous readings are being transmitted from the pacemaker.

Table 5.6.6.a: Summary of warning states and their activation conditions. When a warning state is active, the heartrate box pulses in colour as defined below, and a Toast message appears. The warning states are dependent on the current BPM and/or the rate of change of the BPM.

Warning State	Activation Conditions	Styling	
		Box Pulsing Outline	Toast Message

Nominal	BPM >= 35 && BPM ROC >= -5	None	None
Moderate Warning	BPM ROC < -5	Yellow	Heart rate is low.
Critical Warning	BPM < 35 || BPM ROC < -10	Red	Heart rate is critically low.

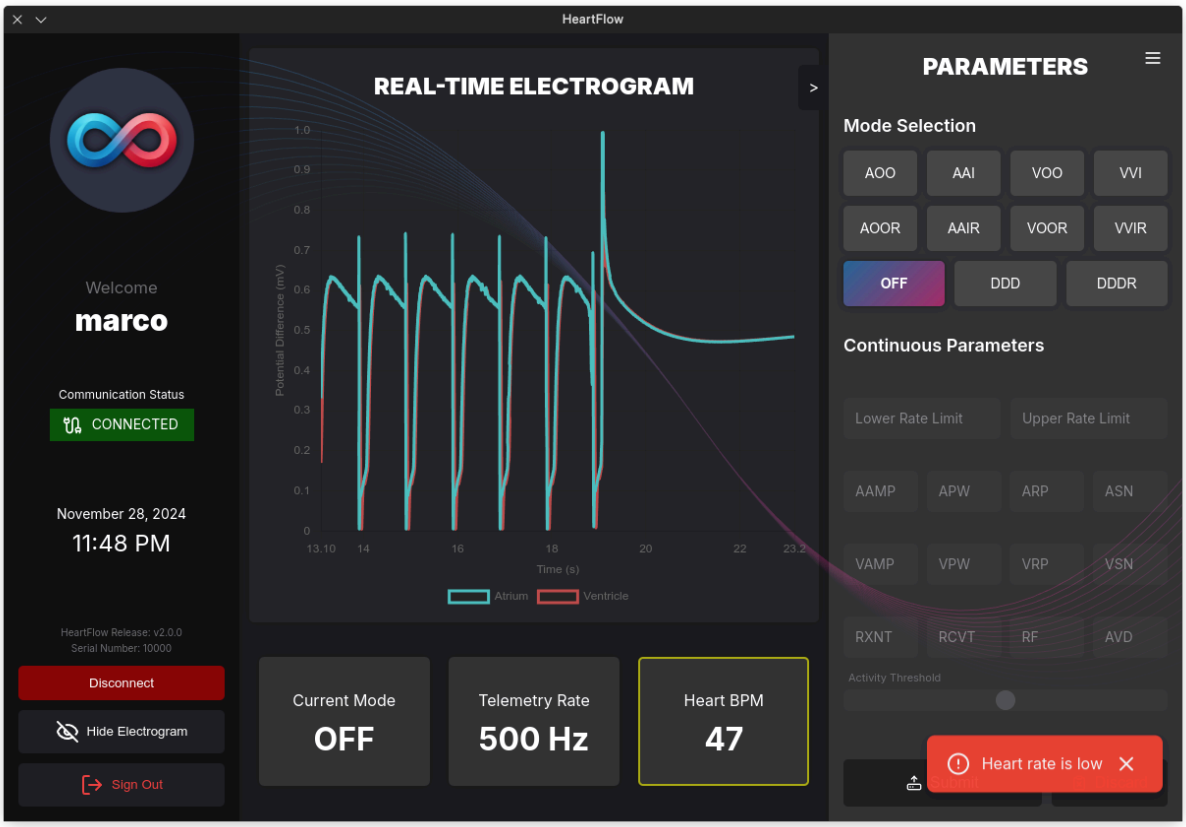

Figure 5.6.6.a: HeartFlow dashboard with a low BPM, with the heart rate box pulsing yellow indicating a moderate warning.

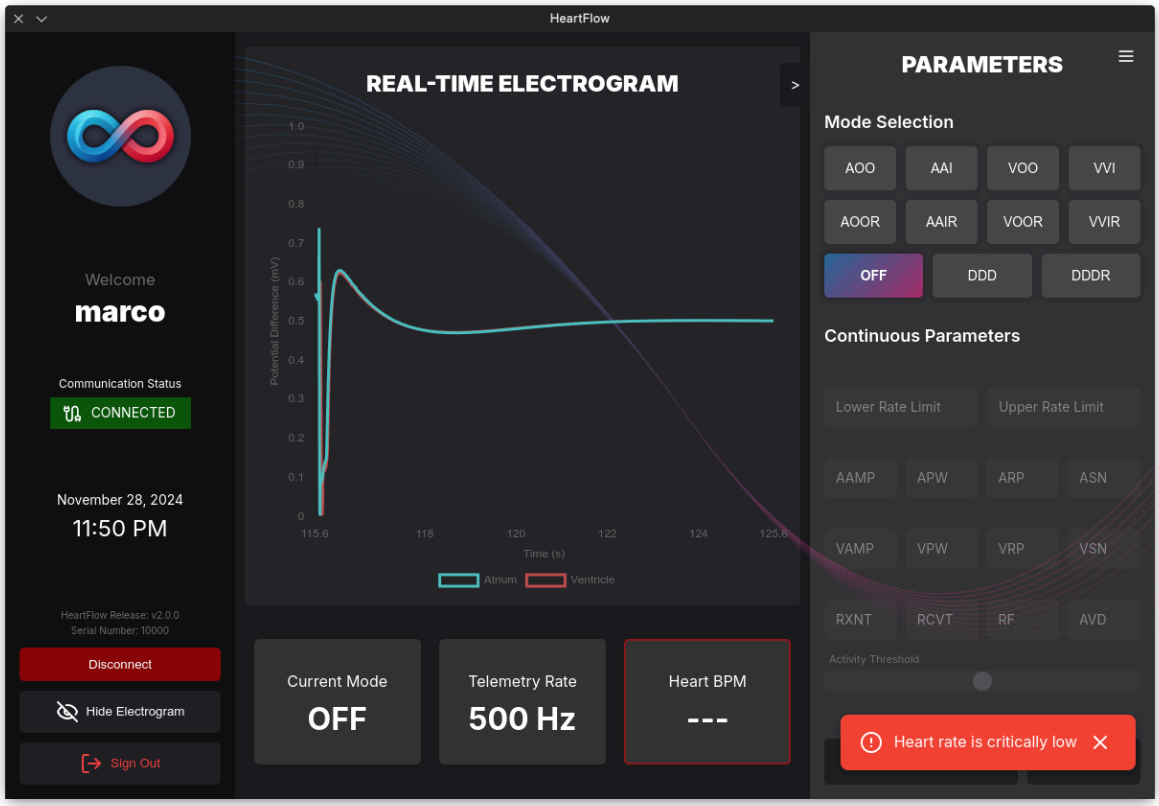

Figure 5.6.6.b: HeartFlow dashboard with a critically low BPM, with the heart rate box pulsing red indicating a critical warning.

Together, these data tiles enhance the user experience by presenting essential metrics in a concise and visually organized manner, enabling informed decision-making and streamlined device management.

5.7 DCM Back-end

The DCM backend consists of all the functions that handle the primary logic of the application, such as verifying user login credentials, registering users, and reading and saving user data, and serial integration. Due to the sensitive nature of some of the information handled by the application, most of the data is kept on the back-end until explicitly requested for by the frontend. The back-end, otherwise known as the main process, is completely isolated from the front-end, or the renderer process.

Inter-process communication (IPC) is used to “expose” the functions in the main process to the renderer process by using “channels”—instead of letting the renderer process call the functions directly, the renderer sends a signal to the main process using an appropriate channel. The main process receives the signals, processes them, runs the corresponding function, and returns the response through the same channel. This hardens the application as critical

processes cannot be directly accessed by the front end. It improves hardware hiding by isolating hardware-specific processes in the main process.

The following class diagram highlights the interactions between major components of the DCM. Note that descriptions of which can be found in 6 Implements.

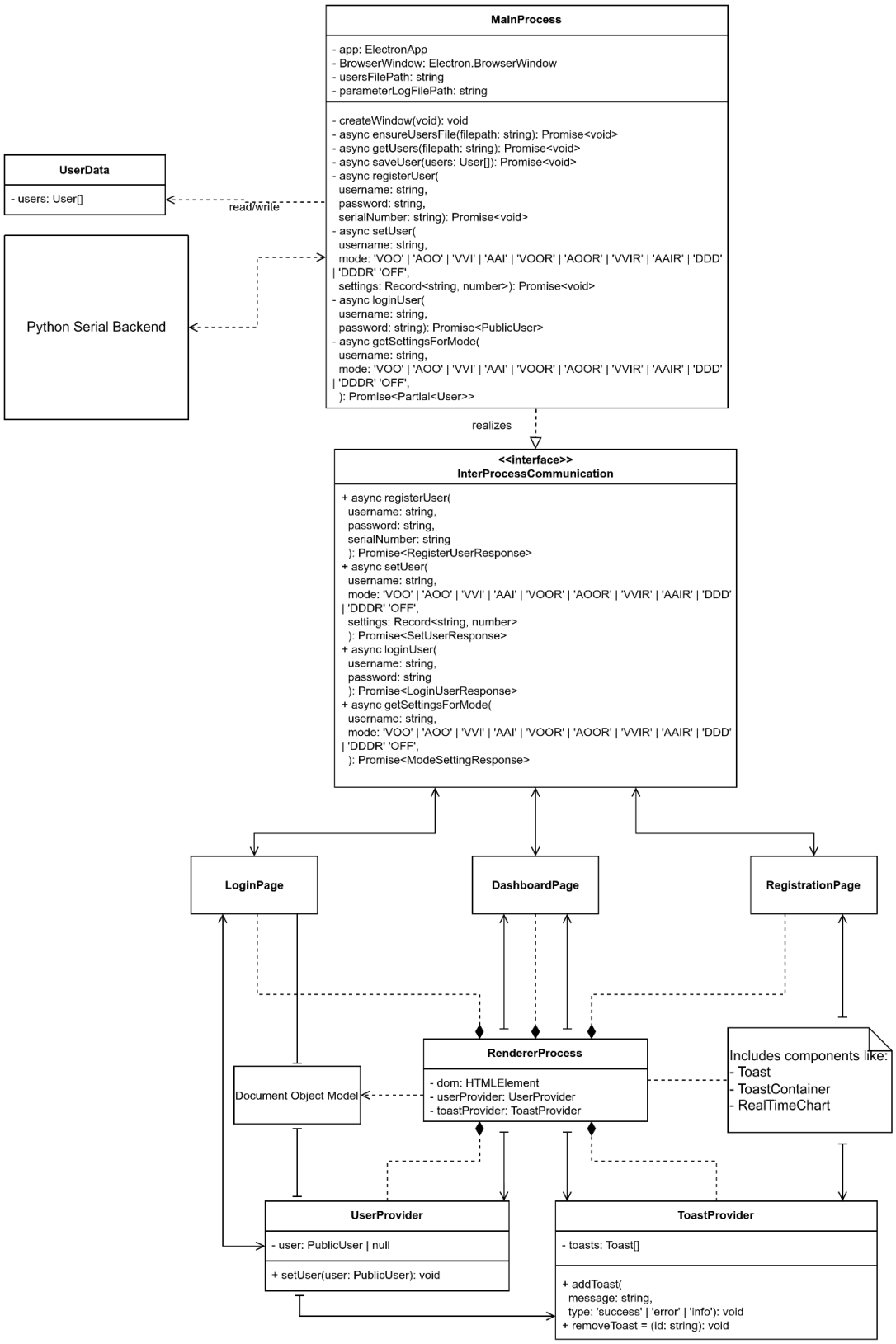

Figure 5.7.a: Class diagram for the DCM back-end.

5.7.1 Serial Integration

This section pertains to the integration of the serial protocol and the behaviour of the DCM. For more specifics on the protocol itself, view the Serial Protocol Documentation.

The design of the serial integration within the DCM is fundamental to the system's ability to communicate effectively with the pacemaker hardware. This integration ensures that real-time data, such as heart activity and pacemaker settings, can be seamlessly transmitted in a bidirectional fashion between the DCM and the firmware, enabling a responsive and interactive user experience.

The DCM's backend is designed to handle serial communication via UART, using a custom protocol that was developed specifically for this system. For more specifics on the protocol itself, view the Serial Protocol Documentation. This communication is vital for the operation of features like real-time electrogram visualization, telemetry data retrieval, and device mode control. One of the backend's primary responsibilities is to manage the flow of data between the DCM and the pacemaker in a way that ensures reliability, consistency, and minimal latency. This is accomplished by maintaining an active connection that continuously monitors for incoming data, processes it, and reflects the changes in the application's interface.

From a design perspective, serial integration is structured to prioritize responsiveness and data integrity. When the pacemaker transmits telemetry data, the DCM is designed to immediately process and display the updated information on the electrogram, providing users with real-time feedback. In parallel, user commands such as mode changes or parameter adjustments are sent via the serial connection to the pacemaker, ensuring that user inputs are promptly acted upon. Additionally, non-user-centric processes such as 0x01 handshakes and 0x02 poll requests are continuously conducted throughout the pacemaker's connection.

To optimize system performance and ensure a smooth user experience, the backend is designed to handle serial communication asynchronously, allowing the DCM to process incoming and outgoing data without interrupting other tasks. This approach reduces the likelihood of communication bottlenecks, ensuring that the system remains responsive even when large amounts of data are being transferred.

The decision to design the serial integration with these principles in mind was driven by the need for reliable, real-time communication in a medical context, where latency or errors could significantly affect the quality of care. By focusing on minimizing delays, ensuring accurate data capture, and providing users with a clear interface that reflects the current state of the pacemaker, the DCM's serial integration enables efficient device control and monitoring.

5.7.1.1 Handshake (0x01)

The 0x01 handshake message is sent only by the DCM and occurs when the user clicks the “Connect” button on the dashboard. The decision to initiate the 0x01 handshake message only upon user request, specifically when the "Connect" button is clicked on the dashboard, was made to provide users with greater control over the pacemaker's connection. This design ensures that the DCM does not automatically begin communication with the hardware upon user login, reducing the risk of unintended device interactions or interruptions. By requiring explicit user action to establish the connection, the system allows for a more deliberate and controlled setup, especially in scenarios where users may want to review settings or prepare the device before initiating communication. This approach also minimizes the load on system resources, as the serial communication is only activated when needed, preventing unnecessary data exchanges and optimizing the overall performance of the DCM. Additionally, this user-driven approach aligns with safety protocols, giving users the opportunity to ensure the pacemaker is in the correct mode or state before establishing a connection, which is crucial in clinical environments where accuracy and timing are critical.

5.7.1.2 Polling (0x02)

The 0x02 polling process can be further divided into how they are handled as poll requests and poll responses.

5.7.1.2.1 Poll Request

The poll request is an asynchronous process that continuously runs only after a connection is established via the completion of a handshake. The DCM sends a poll request to ports in which a successful handshake was completed every 10 milliseconds. The continuous poll requesting process will be arrested only when the DCM does not receive a poll response from the “handshaken” device’s ID for 1000 consecutive poll requests, a time equivalent to 10 seconds.

Poll requesting too often would overflow the pacemaker’s buffer and slow down the DCM operations due to the continuous background operations, and poll requesting too sparsely temporally would carry the issue of there being a lag between device disconnection and the DCM detecting the disconnection. We found a 10-millisecond polling rate to strike the balance between these two extremes. Practically, the communication would be done wirelessly using a wand to communicate with the pacemaker, which motivated the 10-second grace period of no communication. This effectively differentiates between the clinician momentarily moving the wand out of the vicinity of the pacemaker and the clinician putting the wand away for storage.

5.7.1.2.2 Poll Response

The DCM is continuously and asynchronously listening for incoming serial communications from ports that have accepted a handshake in the active session. A conditional statement is executed upon retrieval of a packet, which checks if the incoming message was an 0x03 poll response. If it was, the DCM resets the 10 second timer mentioned in the previous section to signify that the device is still actively connected.

5.7.1.3 Parameters and Mode Change Request (0x03)

The parameters and mode change request message type is only dispatched upon the user clicking “Submit” on the right sidebar. The button must be active for the message to be dispatched. The serial message will be dispatched even if the previously submitted parameters are the same. This decision was justified by the safety value added; this feature could be useful in the unlikely event that the pacemaker’s firmware and the DCM believe the pacemaker to currently exist in a different mode or with differing parameters.

Upon receipt of a 0x03 message, the DCM checks whether it matches its previously transmitted 0x03 message. If it does, it will initiate the transmission of a 0x04 message. Otherwise, it will initiate the transmission of the previously submitted 0x03 message.

5.7.1.4 Parameters and Mode Change Confirmation (0x04)

As previously mentioned, the DCM is asynchronously and continuously listening for serial data through handshake-accepted ports. Upon submitting a 0x03 message, the DCM will expect a 0x03 in response. If these two 0x03 messages match, this 0x04 message is constructed and transmitted.

5.7.1.5 Electrogram Data (0x05)

Upon acknowledging the acceptance of the handshake, the DCM will construct and transmit a 0x05 message to indefinitely subscribe to future electrogram data. While continuously listening, the DCM expects, retrieves, and maps the electrogram data directly onto the electrogram and gets incorporated into the electrogram report metadata.

5.8 Toast Components

Toast notifications are employed throughout the HeartFlow application to provide users with immediate, non-intrusive feedback on actions such as parameter changes, errors, or successful operations. These notifications are designed to enhance user experience by offering clear and concise messages typically regarding backend processes without disrupting workflow.

Toasts appear briefly in the bottom right corner of the screen and automatically dismiss after a short duration. They are categorized based on the type of feedback: Success, error, or information, ensuring users can quickly distinguish the context of the notification. Design decisions for toast components focus on maintaining minimalism and clarity, using contrasting colours and icons to convey their respective messages. For instance, green is used for success notifications, red for errors, and blue for informational messages. Each toast includes a short message and, where applicable, further details on corrective action or confirmation of success. Examples of their application within the DCM include upon registration or login success, parameter discarding, and upon submission of invalid login or registration information. The table below highlights all Toast messages in the program.

6 Implementation

The Implementation section provides an overview of the code structure and how various functionalities of the HeartFlow application have been implemented. This section breaks down the front-end implementation into specific components, outlining the logic and organization of the code for key features such as user registration, login processes, and the dashboard interface.

The DCM, which was developed using React running on Electron, consists of two primary processes–the main process and the renderer process. The main process is responsible for operating system and hardware level functionality, such as reading and writing to files, encrypting and verifying user passwords, creation of operating system windows, and communication with the pacemaker. The renderer process is responsible for handling all UI-related functionality, such as UI animations, handling inputs and events for fields and buttons, and rendering data. This setup is required to decouple hardware-related and operating system-level functionality from the interface, allowing users to use the DCM without understanding the implementation of certain features. Further, it allows flexible modification of the UI.

6.1 Front-end Implementation

The front-end of the HeartFlow application is built using React, a popular JavaScript library for creating user interfaces. It employs a component-based architecture that enhances reusability and maintainability. State management is handled with React's **`useState`** and **`useEffect`** hooks, allowing for a more functional approach to component lifecycle and state changes. The application utilizes React Router for navigation, enabling seamless transitions between different pages, such as the registration and login screens. The design incorporates a responsive layout, ensuring compatibility across various devices.

Each component follows a structured approach that includes form validation and error handling. For instance, when a user attempts to register or log in, the application checks for specific criteria, such as username length and password complexity. Feedback is provided through Toast notifications to inform users about the success or failure of their actions, enhancing the overall user experience.

6.1.1 Registration Page Implementation

The **`Register.tsx`** component is responsible for handling user registration. It consists of a form with input fields for username, password, password confirmation, and pacemaker serial number. State management for each input is handled using **`useState`**, allowing for dynamic updates as users type.

The form validation logic is implemented in the **`handleSubmit`** function, which is triggered upon form submission. This function checks the validity of the input data against

predefined criteria, such as ensuring that the username is at least three characters long and that the password contains at least one special character. If validation fails, appropriate error messages are displayed using the **`addToast`** method from the **`ToastContext`**.

The **`handleSubmit`** function also interacts with the backend through the **`window.api.registerUser`** method, which sends the registration data for processing. If the registration is successful, the user is navigated to the login page. The form includes floating label inputs, which enhance the user experience by providing clear labels for each field while keeping the interface clean.

6.1.2 Login Page Implementation

The **`Login.tsx`** component manages user authentication. Similar to the registration page, it features input fields for the username and password, with state management handled via **`useState`**.

The **`handleSubmit`** function processes login attempts by validating the user's credentials against the backend using the **`window.api.loginUser`** method. On a successful login, user data is stored in the global context via the **`setUser`** method, allowing for state persistence across the application.

To maintain a responsive user experience, the login form includes validation and feedback mechanisms similar to those in the registration component. Users receive toast notifications based on the success or failure of their login attempts. The implementation also utilizes a floating label design for input fields, ensuring that users can easily identify the required information while interacting with the login form.

6.1.3 Dashboard Implementation

The DCM Dashboard is the primary interface for managing pacemaker settings in HeartFlow. The component is designed for real-time telemetry status, mode selection, and parameter configuration. The dashboard is further subdivided into three key areas:

- The **main content** is the content in the center of the screen which includes the electrogram, the currently used mode, the telemetry rate, and the current BPM.
- The **left divider content** is the content in the leftmost section that contains the HeartFlow logo. Key information and interactivity here are:
    - The current user.
    - The current connection status.
    - The time and date.
    - A button to connect/disconnect from the pacemaker.
    - A button to hide the electrogram.
    - A button to sign out.
- The **right divider content** contains most of the functionality of the DCM.

- The default view of the right divider is mode and parameter selection. It contains all the applicable modes, areas to edit the parameter values, a button to submit changes to the pacemaker, and a button to discard changes and load the most recently submitted mode.
- Each parameter has a small information icon by the name which can be hovered over to view a tooltip, detailing the units of the values and their ranges.
- The hamburger button at the top right corner drops down a menu, allowing the user to switch to the Reports view of the right divider.
    - The Reports view allows the user to select reports they would like to download to their computer. Currently available reports include:
        - A parameter log is a full history of all modes and parameter changes to the user's pacemaker.
        - An activity log which is the full history of all user account activity.

6.1.4 Context Providers

Context providers are React components that enable state management and sharing across different components. They are used to avoid the issue of "prop drilling", wherein props are passed down through multiple levels of components, even if intermediate components do not need the data.

In the context of the DCM, context providers are used to manage global state variables throughout the application. There are two primary context providers used in the DCM:

- **`ToastProvider:`** This context provider manages and shares the state of toast notifications throughout the application. The context provider adds toasts, helps with the rendering of toast components, and initiates the automatic removal of toasts after a set amount of time.

6.1.4.1 Toast Notifications

`ToastProvider` contains one state variable, **`toasts`**. Toasts are added via the **`addToast`** method and can be manually destroyed with **`removeToast`**. The provider will also automatically remove toasts after three seconds. The pages never access the **`toasts`** array directly, as the rendering of toast notifications is delegated to **`ToastContainer`**, which is always rendered as the topmost component in the DOM. The relationship between components that use this context provider is highlighted below:

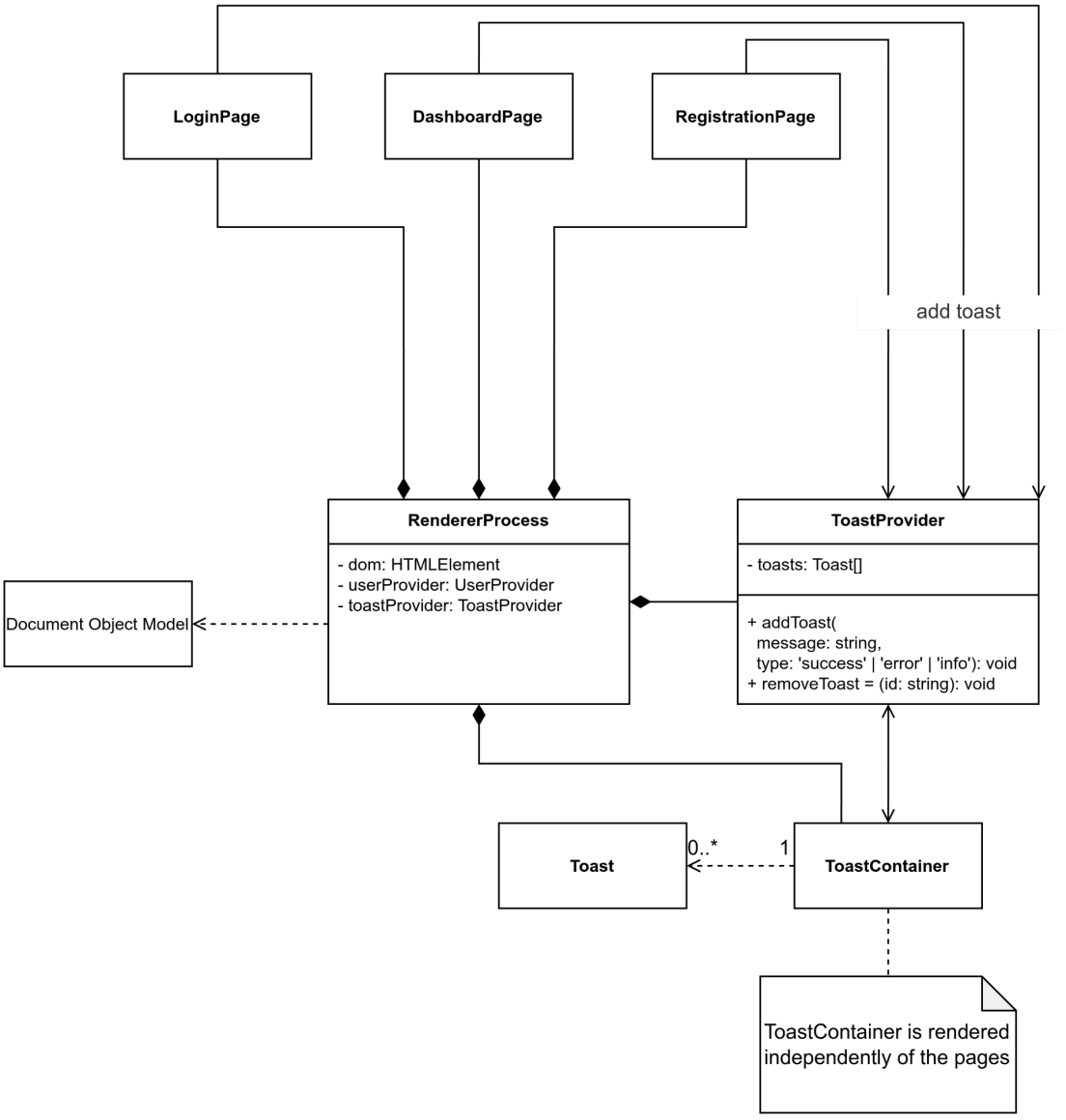

Figure 6.1.4.2.a: Class diagram displaying components and their relationships under the Toast context provider.

6.1.5 Global State Management

Because there is a significant amount of user information shared across various pages and components, the use of a context provider would be inappropriate as it would cause frequent re-renders which could potentially slow down the application, become disorganized when there are significant state updates and access, and increases the amount of coupling in the application. Therefore, a more robust state management solution is required.

A state management library called Zustand was ultimately chosen as the best solution to the state management problem in the DCM. Zustand can be used to create a centralized store

where all values pertaining to the application state can be organized and updated. Zustand is also designed such that state updates are granular, and components that subscribe to a specific state variable only re-render if that variable updates. Finally, using Zustand increases cohesion while decreasing coupling by providing a unified dispatch method where all state updates can be performed. This section describes the Zustand store implementation in the DCM.

6.1.5.1 Global Store Organization and Structure

The global store is separated into three sections: **UserState**, **PacemakerState**, and **TelemetryState**:

- **UserState** is responsible for storing information about the current user logged in. The information in this section of the store is:
  - **username** as a string. This is primarily used to retrieve more sensitive information later.
  - **serialNumber** as a string. This is the pacemaker serial number and is required for operations like establishing the initial handshake.
- **PacemakerState** is responsible for storing recently retrieved parameters or modified parameters. This section is empty by default and is only populated when values are retrieved via the appropriate IPC channel mentioned in 6.2.7 Exposure of Methods with IPC Channels. The information stored in this section of the store is:
  - **lastUsedMode** which is the last submitted mode.
  - **currentMode** which is the current mode in operation.
  - **modes** which is a key-value map, mapping each mode to their appropriate parameters. For example, under **VOO** in **modes**, the parameters stored are:
    - **ventricularAmplitude**
    - **ventricularPulseWidth**
    - **ventricularRefractoryPeriod**
    - **upperRateLimit**
    - **lowerRateLimit**
  - **telemetry** which is an object that contains pacemaker telemetry values. The only value under this object is **heartRate**.
- **TelemetryState** is responsible for storing information regarding connection status and whether or not the pacemaker is returning telemetry. This section includes:
  - **connectionStatus**, the pacemaker connection status.
  - **telemetryStatus**, the status of telemetry transmission from the pacemaker to the DCM.

6.1.5.2 State Update Dispatcher

To prevent unnecessary coupling between the state store and the rest of the DCM, all updates to the DCM state are done via the **dispatch** method. **dispatch** takes in two arguments:

- **type**: this specifies what state variables are to be updated. The only valid values for this parameter are:
  - UPDATE_USER

    - UPDATE_MODE_SETTINGS
    - UPDATE_TELEMETRY
    - UPDATE_LAST_USED_MODE
    - UPDATE_CURRENT_MODE
    - UPDATE_CONNECTION_STATUS
    - UPDATE_TELEMETRY_STATUS
- **payload**: This parameter contains the values that will update the state given a specific **type** of state update. The expected payloads vary depending on what is being updated. The following table maps each type to the expected payload format.

Dispatch Type	Expected Dispatch Payload

UPDATE_USER	`Partial<{`
	`  username: string`
	`  serialNumber: string`
	`}>`

UPDATE_MODE_SETTINGS	`Partial<{`
	`  /* MODE SPECIFIC VALUES */`
	`}>`
	
	`---- PER MODE ----`
	
	`VOO: {`
	`  ventricularAmplitude: number`
	`  ventricularPulseWidth: number`
	`  ventricularRefractoryPeriod: number`
	`  lowerRateLimit: number`
	`  upperRateLimit: number`
	`}`
	
	`AOO: {`
	`  atrialAmplitude: number`
	`  atrialPulseWidth: number`
	`  atrialRefractoryPeriod: number`
	`  lowerRateLimit: number`
	`  upperRateLimit: number`
	`}`
	
	`VVI: {`
	`  ventricularAmplitude: number`
	`  ventricularPulseWidth: number`
	`  ventricularRefractoryPeriod: number`
	`  lowerRateLimit: number`
	`  upperRateLimit: number`
	`  ventricularSensitivity: number`
	`}`

	AAI: {
	  atrialAmplitude: number
	  atrialPulseWidth: number
	  atrialRefractoryPeriod: number
	  lowerRateLimit: number
	  upperRateLimit: number
	  atrialSensitivity: number
	}
	
	DDDR: {
	  atrialAmplitude: number
	  atrialPulseWidth: number
	  atrialRefractoryPeriod: number
	  ventricularAmplitude: number
	  ventricularPulseWidth: number
	  ventricularRefractoryPeriod: number
	  lowerRateLimit: number
	  upperRateLimit: number
	  rateFactor: number
	  reactionTime: number
	  recoveryTime: number
	  avDelay: number
	  atrialSensitivity: number
	  ventricularSensitivity: number
	}
	
	DDD: {
	  atrialAmplitude: number
	  atrialPulseWidth: number
	  atrialRefractoryPeriod: number
	  ventricularAmplitude: number
	  ventricularPulseWidth: number
	  ventricularRefractoryPeriod: number
	  lowerRateLimit: number
	  upperRateLimit: number
	  avDelay: number
	  atrialSensitivity: number
	  ventricularSensitivity: number
	}
	
	AOOR: {
	  atrialAmplitude: number
	  atrialPulseWidth: number
	  atrialRefractoryPeriod: number
	  lowerRateLimit: number
	  upperRateLimit: number
	  rateFactor: number
	  reactionTime: number
	  recoveryTime: number

	}
	
	AAIR: {
	  atrialAmplitude: number
	  atrialPulseWidth: number
	  atrialRefractoryPeriod: number
	  lowerRateLimit: number
	  upperRateLimit: number
	  rateFactor: number
	  reactionTime: number
	  recoveryTime: number
	  atrialSensitivity: number
	}
	
	VOOR: {
	  ventricularAmplitude: number
	  ventricularPulseWidth: number
	  ventricularRefractoryPeriod: number
	  lowerRateLimit: number
	  upperRateLimit: number
	  rateFactor: number
	  reactionTime: number
	  recoveryTime: number
	}
	
	VVIR: {
	  ventricularAmplitude: number
	  ventricularPulseWidth: number
	  ventricularRefractoryPeriod: number
	  lowerRateLimit: number
	  upperRateLimit: number
	  rateFactor: number
	  reactionTime: number
	  recoveryTime: number
	  ventricularSensitivity: number
	}

UPDATE_TELEMETRY	Partial<{ heartRate: number }>

UPDATE_LAST_USED_MODE	lastUsedMode: | 'OFF'
	  | 'AOO'
	  | 'AAI'
	  | 'VOO'
	  | 'VVI'
	  | 'DDDR'
	  | 'DDD'
	  | 'AOOR'

	`| 'AAIR'`
	`| 'VOOR'`
	`| 'VVIR'`

`UPDATE_CURRENT_MODE`	`currentMode: | 'OFF'`
	`| 'AOO'`
	`| 'AAI'`
	`| 'VOO'`
	`| 'VVI'`
	`| 'DDDR'`
	`| 'DDD'`
	`| 'AOOR'`
	`| 'AAIR'`
	`| 'VOOR'`
	`| 'VVIR'`

`UPDATE_CONNECTION_STATUS`	`connectionStatus: 'CONNECTED' | 'DISCONNECTED' | 'CONNECTING' | 'RECONNECTING'`
`UPDATE_TELEMETRY_STATUS`	`telemetryStatus: 'ON' | 'OFF'`

6.1.6 Other Components

This section covers any other components that do not fit in the other sections.

6.1.6.1 Toasts and Related Components

`Toast` components contain the following props that are used by the **`ToastContainer`** during rendering:

- **`message:`** the message to display in the toast notification.
- **`type:`** toast type to render. Options are *`success`*, *`error`*, and *`info`*.
- **`onClose:`** a callback that runs when the toast is removed.
- **`removing:`** an optional Boolean value used to control the toast removal animation.

The **`ToastContainer`** dynamically renders toast notifications based on the state of **`toasts`** in **`ToastProvider`**. When generating the toast notifications, **`ToastContainer`** provides each **`Toast`** component with an anonymous function that removes the toast from **`toasts`** after playing the removal animation.

6.1.6.2 Real-Time Visualization Chart

`RealTimeChart` is the component responsible for rendering line graphs for visualizing continuous variables such as pacemaker output voltage. This component contains the following props:

- **`series1`:** The default series that is rendered in the generated line chart. This must always exist.
    - **`data`:** A list of type `ChartPoint` (each pair of X and Y values).
    - **`title`:** The series' label.
    - **`xWidth`:** The number of points along the x-axis.
    - **`yMin`:** The lower bound of the y-axis.
    - **`yMax`:** The upper bound for the y-axis.
- **`series2`:** An optional series that can be supplied to the graph. Subprops underneath this series are identical to **`series1`**.
- **`width`:** The component width in pixels.
- **`height`:** The component height in pixels.

`RealTimeChart` will create and re-render the line graph automatically whenever any of the series are updated. It is expected that some loop/function that consumes this component continuously updates **`data`** at a set time interval. **`RealTimeChart`** will also render the chart in a separate thread to avoid blocking the main thread with constant re-renders.

6.2 Back-end Implementation

The DCM backend consists of functions in the main process responsible for handling the major functions of the DCM, such as validating user login credentials, registering new users, and retrieving and writing user data. The following class diagram is an overview of the main process:

MainProcess
- app: ElectronApp - BrowserWindow: Electron.BrowserWindow - usersFilePath: string - parameterLogFilePath: string
- createWindow(void): void - async ensureUsersFile(filepath: string): Promise<void> - async getUsers(filepath: string): Promise<void> - async saveUser(users: User[]): Promise<void> - async registerUser(username: string, password: string, serialNumber: string): Promise<void> - async setUser(username: string, mode: 'VOO' | 'AOO' | 'VVI' | 'AAI' | 'VOOR' | 'AOOR' | 'VVIR' | 'AAIR' | 'DDD' | 'DDDR' 'OFF', settings: Record<string, number>): Promise<void> - async loginUser(username: string, password: string): Promise<PublicUser> - async getSettingsForMode(username: string, mode: 'VOO' | 'AOO' | 'VVI' | 'AAI' | 'VOOR' | 'AOOR' | 'VVIR' | 'AAIR' | 'DDD' | 'DDDR' 'OFF',): Promise<Partial<User>>

Figure 6.2.a: Class representation of the MainProcess module.

Although the main process is not what would be considered a “traditional class,” the visibility of the methods of the main process is effectively treated as private, as the only method of calling the functions is through IPC channels. The next few sections describe the functions and their operations.

6.2.1 Electron Boilerplate

Although a lot of the Electron setup code does not vary during the lifetime of the application, functions related to this are essential for the application to start correctly. During initial app creation, the main process sets up the following:

- **`app:`** This is the main application instance for the Electron framework. It is set up automatically by the framework when the application initially starts.
- **`BrowserWindow:`** This is the main instance for the application window. Electron works by using a Chromium browser instance to render a Document Object Model (DOM) window which is used for the graphical user interface of the application. This instance is set up by…
- **`createWindow:`** This is the main function responsible for the creation of the OS window and browser instance once the Electron app instance is ready to start rendering the DOM.

6.2.2 Error Handling

Exceptions thrown by any function in the main process are not directly handled by the main process. Instead, the exceptions are propagated through the IPC channel to the renderer process and are handled there instead. This is possible due to two factors:

- Electron's IPC uses **asynchronous** message passing. When an exception occurs within a main process IPC handler, it is caught and sent as an error response back to the renderer process.
- Electron's architecture **isolates** the main process and renderer process. This isolation prevents unhandled exceptions in the main process from directly crashing the renderer process, allowing the renderer process to handle the error gracefully.

All exceptions in the main process are handled this way.

6.2.3 User File Input/Output

The main process has functions defined to read and write from the main JavaScript Object Notation (JSON) file used to store user information, including usernames, password hashes, and values for individual pacemaker modes. The following functions are treated effectively as private methods and are never exposed via IPC to the renderer process as they are meant to be consumed by main process functions only.

`ensureUsersFile` ensures the JSON file exists. Generally, on the first startup of the program, the function creates the file. Subsequent runs of the application will ensure the file exists before the rest of the application loads. The file is always created in an automatically generated program-specific application data folder henceforth referenced as **userData**. The folder location varies depending on the operating system:

- On **Linux**-based distributions, the userData folder is located at `~/.config/dcm`.
- On **Windows**, the userData folder is located at `C:\Users\<user>\AppData\Roaming\dcm`.
- On **macOS**, the userData folder is located at `~/Library/Application Support/dcm`.

The following expression table highlights the behaviour of this function.

Figure 6.2.3.a: Expression table for user file input and output.

	Result	
Condition	*Successful Result*	*Failed Result*
The file does not exist.	The file is created in the userData folder.	An exception is thrown that is handled later.
The file does exist.	X	An exception is thrown that is handled later.

If **`ensureUsersFile`** cannot verify that the file exists or cannot create the file, the exception thrown will prevent the application from starting.

`getUsers` simply returns the list of users from the file. There is no expression table for this function, as the function is guaranteed to return a list of users due to the check done by **`ensureUsersFile`**.

Finally, **`saveUser`** simply writes data to the file. Like **`getUsers`**, the behaviour of this function is guaranteed due to the check done by **`ensureUsersFile`**.

6.2.4 User Registration and Login Verification

`registerUser` handles the registration of new users in the application. The function takes a username, password and pacemaker serial number. It verifies if the user already exists or if the application has reached the maximum number of allowed users, then proceeds to either reject the request or create the user. The following tabular expression table highlights the behaviour of this function.

Figure 6.2.4.a: Tabular expression table for user registration.

	Result	
Condition	*Successful Result*	*Failed Result*
The user already exists.	X	An exception is thrown that is handled later.
The application has ten registered users.	X	An exception is thrown that is handled later.

The user does not exist, and the application has not reached user capacity.	User is created and stored in the JSON.	An exception is thrown that is handled later.

The function hashes the passwords and then stores them since hashes are designed as one-way operations—the password cannot be reversed or engineered from a hash, so long as the password is sufficiently strong enough.

`loginUser` handles user password verification during login. The function receives a username and password, verifies that the user exists, hashes the cleartext password and verifies that the hash is equal to the stored password hash, then returns user information on a successful login, or rejects the request. The following tabular expression table highlights the behaviour of this function.

Figure 6.2.4.b: Tabular expression table for user login verification.

	Result	
Condition	*Successful Result*	*Failed Result*
The user does not exist.	X	An exception is thrown that is handled later.
User exists but the password hash is incorrect.	X	An exception is thrown that is handled later.
The user exists and the password hash is correct.	A user data object, PublicUser, is returned.	An exception is thrown that is handled later.

6.2.5 User Data Input/Output

`setUser` handles changes to the settings for a specific pacemaker mode for a given user. The function takes in the username, the mode, and an associative array representing the settings for that given mode. The function then verifies that the user exists, updates the settings for the given mode for that user, updates the last used mode to the provided mode, and writes all changes to disk. The shape of the associative array is guaranteed as the correct parameters are ensured in the renderer process before the object is sent through the appropriate IPC channel. This function runs whenever new setting information is sent to the pacemaker. The following tabular expression table highlights the behaviour of this function.

Figure 6.2.5.a: Tabular expression table for setting the user.

	Result	
Condition	*Successful Result*	*Failed Result*
The user does not exist.	X	An exception is thrown that is handled later.
The user exists and a correct mode is supplied.	The updated values are written to disk for the given user.	An exception is thrown that is handled later.

Generally, the user is guaranteed to exist as the login page only proceeds to the dashboard if valid user data can be retrieved and stored on the front-end.

`getSettingsForMode` retrieves the settings for a specified mode for a given user. The function takes in the username, and the mode, and returns a partial User object containing information for the given mode if the user exists. The following tabular expression table highlights the behaviour of this function.

Figure 6.2.5.b: Tabular expression table for getting settings for a specified mode.

	Result	
Condition	*Successful Result*	*Failed Result*
The user does not exist.	X	An exception is thrown that is handled later.
The user exists and a correct mode is supplied.	Return a partial User object containing the mode settings.	An exception is thrown that is handled later.

Like **`setUser`**, the user is generally guaranteed to exist as the login page only proceeds to the dashboard if valid user data can be retrieved and stored on the front-end.

6.2.6 Login and Parameter History Logging

The main process also has functions to set up logging of parameter changes and user logins. Similarly to the functions set up for user file I/O, these methods are never directly exposed to the renderer process and instead are consumed in the main process through IPC channel handlers.

`ensureDirectoryExists` and **`ensureParameterHistoryFile`** work similarly to **`ensureUsersFile`** to verify that the JSON file for storing all parameter changes and login times exists. This file will henceforth be called **parameterHistory**. If the file does not exist, it is created, typically on start-up along with the userData JSON file. No tabular expression is provided as the behaviour of both functions is nearly identical to **`ensureUsersFile`**.

`addUserToHistory` is used within **`registerUser`** to add a new entry in the log file for a newly registered user. The behaviour of this function is guaranteed to not fail as parameterHistory is guaranteed to exist.

`logUserLogin` is used within **`loginUser`** to add a new login entry to parameterHistory. Likewise, this function is also guaranteed to not fail as parameterHistory is guaranteed to exist. This function is guaranteed to not be invoked if the login fails.

`logUserParameterChange` is used within **`setUser`** to add a new parameter change entry to parameterHistory. This function will not be invoked unless the parameter change is successful and is guaranteed to always succeed as parameterHistory is guaranteed to exist.

6.2.7 Exposure of Methods with IPC Channels

This section details the implementation of inter-process communication channels to facilitate interaction between the main process and the renderer process of the application with regard to user management and value retrieval. IPC enables these two processes, which are isolated and operate in separate environments, to exchange data and invoke functionality across boundaries.

The DCM leverages Electron's built-in IPC module, which provides a robust and efficient mechanism for communication. The module offers two primary methods for communication:

- **`ipcMain:`** This module resides in the main process and handles incoming messages from the renderer process. It listens for specific "channels" (which are identified by unique strings) and executes corresponding functions when messages are received on those channels. Responses can be returned through the same channels. Additionally, **`ipcMain`** may send messages to **`ipcRenderer`**.

- **`ipcRenderer:`** Similar to **`ipcMain`**, this module resides in the renderer process and sends messages to the module in the main process. **`ipcRenderer`** can also be configured to listen for messages from **`ipcMain`** and execute changes to the UI relative to the message.

Currently, the DCM is set up to receive messages on **`ipcMain`** and return appropriate responses based on the execution of corresponding functions. **`ipcRenderer`** is set up to send messages to appropriate channels to invoke functions in the main process. The following functions are exposed via the following channels:

Figure 6.2.7.a: Main process functions, channel IDs, and expected inputs and outputs.

Main Process Function	**Channel ID & Expected Input**	**Expected Output**	
		Success	*Failure*

`registerUser`	ID: `register-user`
	
	Expected input:
	`username: string`
	`password: string`
	`serialNumber: string`
		`success: boolean = true`
		
		No message is returned.
			`success: boolean = false`
			
			`message: string = <error>`

`setUser`	ID: `set-user`
	
	Expected input:
	`username: string`
	`mode: string`
	`settings: Record<string, number>`
		`success: boolean = true`
		
		No message is returned.
			`success: boolean = false`
			
			`message: string = <error>`

`loginUser`	ID: `login-user`
	
	Expected input:
	`username: string`
	`password: string`
		`success: boolean = true`
		
		`user: PublicUser = <user>`
			`success: boolean = false`
			
			`message: string = <error>`

`getSettingsForMode`	ID: `get-settings-for-mode`
	
	Expected input:
	`username: string`
	`mode: string`
		`success: boolean = true`
		
		`settings: Record<string, number>`
			`success: boolean = false`
			
			`message: string = <error>`

Channels also exist for retrieving login and parameter setting histories. These channels are not tied to specific main process functions and thus are separated into the following table:

Figure 6.2.7.b: Main process functions, channel IDs, and expected inputs and outputs.

Channel ID & Expected Input	**Function**	**Expected Output**	
		Success	*Failure*

ID:
`download-parameter-log`
 	
Expected input:
`username: string`
	To generate a CSV containing the user's parameter setting history. The CSV is written to the user's default downloads folder.	`success: boolean = true`
		`directory: string`
		
		The directory is where the file was downloaded.
			`success: boolean = false`
			
			`message: string = <error>`

ID:
`download-login-history`
 	
Expected input:
`username: string`
	To generate a CSV containing the user's login history. The CSV is written to the user's default downloads folder.	`success: boolean = true`
		`directory: string`
		
		The directory is where the file was downloaded.
			`success: boolean = false`
			
			`message: string = <error>`

6.2.8 Serial Communication

This section details the implementation of serial communication to facilitate interaction between the DCM and the pacemaker. While a separate document provides a comprehensive description of the serial protocol itself, this section will focus on the architecture and data flow within the DCM that allows communication with the pacemaker firmware.

The DCM leverages a multi-layered approach utilizing Python for serial communication, WebSockets for communication between Python and the main process in the Electron application, and IPC within the DCM to manage data transfer between the renderer and the main process.

6.2.8.1 Python Serial Communication Backend and WebSocket Server

The Python serial communication backend can be decomposed into two primary components:

- The **`PacemakerMPSerial`** class, which uses PySerial to implement routines for establishing and managing the serial UART connection to the pacemaker, to provide methods for sending commands, to establish handshakes, to organize processes to continually poll the device to ensure connectivity, and to robustly handle errors, including automatic reconnection attempts and error propagation to the WebSocket server.
- The primary process, which establishes a local WebSocket server, enables bi-directional communication with the DCM's main process. The server is responsible for receiving and parsing commands from the Electron GUI and forwarding them to the PySerial module for execution, and the transmission of data received from the pacemaker back to the DCM for processing and display.

6.2.8.1.1 Serial Communication with PySerial on Python Backend Process

To enable direct and safe serial communication with the pacemaker, **`PacemakerMPSerial`** implements a series of internal parallel processes and exposed methods per the protocol defined in serial protocol documentation. The following table outlines the processes and exposed methods, as well as primary functions within this subunit of the Python serial communication backend:

Figure 6.2.8.1.1.a: Primary Python serial communication functions with expected inputs and outputs.

Process/Method	**Function**	**Expected Input and Output**	
		Input	*Output*
`consume_egram_data`	To return any electrogram data received by the serial reading process, `read_process`	None.	A key-value map where: ● The key is the current UNIX epoch timestamp. ● Two lists of atrial and ventricular sensing data, containing twenty values each, spaced at 2 milliseconds apart.
`search_and_connect`	To perform searching of valid serial port devices and to attempt to establish connections to these devices via the handshake routine defined in the serial protocol.	● The mode as either “`init`” or “`reconnect`” ● An optional seconds timeout value that defaults to ten seconds.	A formatted serial method message containing: ● The status of the command, of either `SUCCESS` or `ERROR`. ● An optional message string.
`send_parameters`	To handle the parameter sending routine defined in the serial protocol including the initial sending of parameters, verification of parameters, and final confirmation.	● A named tuple of all available pacemaker parameters as defined in the firmware documentation. ● An optional retry count that defaults to five tries.	A formatted serial method message containing: ● The status of the command, of either `SUCCESS` or `ERROR`. ● An optional message string.
`toggle_egram`	Toggle whether the pacemaker returns electrogram data.	● An optional value which dictates if toggling behaviour	A formatted serial method message containing:

		is based on an internal state, defaulting to `True`. ● An optional value which dictates an explicit state to set the electrogram to, defaulting to `False`. This value is only used if the previous value is set to `False`.	● The status of the command, of either `SUCCESS` or `ERROR`. ● An optional message string.
`close`	To close a serial connection.	None.	None.
`read_process`	To continuously read messages returned by the pacemaker and fulfill requests created by the rest of the submodule.	● A value dictating whether the process is running or not. ● A key-value buffer containing requests waiting to be fulfilled. ● A key-value buffer specifically for electrogram data returned by the pacemaker.	Returns messages that fulfill a given request via the buffer passed in during process spawning.
`poll_process`	To continuously send polling commands to the pacemaker to verify if the connection exists.	● A value dictating whether the process is running or not. ● A global value to write to, dictating whether the submodule is connected to a serial port. ● A global value to write to, dictating whether the submodule should try to connect to a serial port. ● Optional timeout value defaults to 0.5 seconds.	Does not directly return a value, but sets the provided global values which dictate the state of the submodule.

These processes work in tandem with the WebSocket server to determine the state of the serial connection and to return values received from the pacemaker. The exposed methods are consumed by the WebSocket server by mapping socket messages to the appropriate method.

6.2.8.1.2 WebSocket Server in Python Backend Process

The following table lists the valid WebSocket message types, the PySerial command that it is mapped to, the expected input in the message payload, and the expected output returned to the DCM via the WebSocket. This table also includes electrogram data that is automatically returned to the DCM when the pacemaker is configured for data streaming.

Figure 6.2.8.1.2.a: Mappings of WebSocket message types to appropriate methods in the Python serial backend with expected inputs and outputs.

WebSocket Message Type	**Mapped Command Method**	**Expected Input**	**Expected Output**	
			Success	*Failure*
`initialize`	`search_and_connect("init")`	The pacemaker ID to look for, `pm_id`, of type `uint16`.	A response message of: • `type: initialize` • `status: success`	A response message of: • `type: initialize` • `status: failed` • `message (optional)`
`disconnect`	`close`	None.	A response message of: • `type: disconnect` • `status: success`	A response message of: • `type: disconnect` • `status: failed` • `message (optional)`
`send_parameters`	`send_parameters`	A JSON object containing all available pacemaker parameters as defined in the firmware documentation.	A response message of: • `type: send_parameters` • `status: success`	A response message of: • `type: send_parameters` • `status: failed` • `message (optional)`
`toggle_egram`	`toggle_egram`	The mode to toggle the electrogram data with, is either "`start`", "`stop`" or "`internal`".	A response message of: • `type: toggle_egram` • `status: success`	A response message of: • `type: toggle_egram` • `status: failed` • `message (optional)`
`error`	`None.`	Any WebSocket message request containing an invalid command type.	None.	A response message of: • `type: error` • "`Unknown message type`"
`egram_data`	`read_process consume_egram_d`	None. Values are returned based on	A response message of:	None.

	`ata`	the state of the electrogram toggle.	• `type: egram_data` • `data` encoded as a JSON object, with the same data directly returned by `consume_egram_data`	

6.2.8.2 WebSocket Client on Electron Main Process

The Electron main process establishes a WebSocket client to the Python backend process using the **`ws`** library. This connection allows for real-time, bidirectional communication between the DCM front end and the pacemaker. Upon initialization of the DCM, the main process spawns a new process for the Python webserver and then attempts to establish the WebSocket connection to it. Once the connection is established, hooks are set up to connect the appropriate IPC channels to each WebSocket message type as stated in the previous section. Listeners are also created to connect responses back to the appropriate IPC emitters on the renderer side. The next section covers the IPC channels established that expose the WebSocket to the front end in a controlled manner.

6.2.8.3 Exposure of Serial Communication WebSocket via IPC Channels

Each of the previously defined WebSocket mappings to appropriate Python functions is exposed safely in the renderer process via IPC channels. Although these channels are similar in function to the channels used for file input and output, the use cases are distinct enough to necessitate separation into their overall submodule of channels. These IPC channels are mapped to high-level APIs in the renderer process. The following sets of tables highlight each method, the IPC channels they are mapped to (and subsequently the WebSocket message type), the expected inputs of each function, and tabular expressions for applicable methods.

Figure 6.2.8.3.a: High-level API methods available to the renderer process for interaction with the WebSocket connection between the main process and the Python backend.

High-level API Method	Function	Expected Input
`serialInitialize`	Establish a serial connection with the pacemaker.	A serial number `pm_id` of type `uint16`.
`serialDisconnect`	Disconnect from an existing serial connection.	None.
`serialSendParameters`	To send mode parameters to a pacemaker, set the mode, and set the parameters using	The pacemaker parameters. Note that not all parameters are used for each mode, but

	the routine outlined in the serial protocol documentation.	all parameters must be assigned a value. ● `mode` of type `string`. Must be a valid mode of the pacemaker. ● `lowerRateLimit` in BPM as a `uint8`, bounded by [30, 175]. ● `upperRateLimit` in BPM as a `uint8`, bounded by [50, 175]. Must be larger than `lowerRateLimit`. ● `atrialRefractoryPeriod` in ms as a `uint16`, bounded by [150, 500]. ● `ventricularRefractoryPeriod` in ms as a `uint16`, bounded by [150, 500]. ● `atrialPulseWidth` in ms as a `uint8`, bounded by [1, 30]. ● `ventricularPulseWidth` in ms as a `uint8`, bounded by [1, 30]. ● `atrialAmplitude` in V as a `float`, bounded by [0.5, 5]. ● `ventricularAmplitude` in V as a `float`, bounded by [0.5, 5]. ● `atrialSensitivity` in V as a `float`, bounded by [0, 5]. ● `ventricularSensitivity` in V as a `float`, bounded by [0, 5]. ● `avDelay` in ms as a `uint16`, bounded by [30, 300]. ● `rateFactor` (unitless) as a `uint8`, bounded by [1, 16]. ● `activityThreshold` (unitless) as a `uint8`, bounded by [1, 7]. ● `reactionTime` in seconds as a `uint8`, bounded by [1, 50].

		● `recoveryTime` in seconds as a `uint8`, bounded by [1, 240].
`serialToggleEgram`	To toggle whether or not the pacemaker sends back electrogram telemetry data to the pacemaker.	`mode` as a string specifying the toggle operation mode. Valid options are `start`, `stop` and `internal`.
`onSerialConnectionMessage`	To register callback functions that are invoked when the main process receives a WebSocket response for initialization, disconnection, and reconnection events.	A function that must take in a message with the following format: ● `type` of `connection`. ● `connectionType` of `initialize`, `disconnect`, or `reconnect`. ● `status` of `reconnecting`, `success`, or `failed`. ● `message` as a string. It is an optional value and may not be assigned.
`onSerialActionMessage`	To register callback functions that are invoked when the main process receives a WebSocket response for mode and parameter setting or electrogram telemetry toggling.	A function that must take in a message with the following format: ● `type` of `action`. ● `action` of either `send_parameters` or `toggle_egram`. ● `status` of `success` or `failed`. ● `message` as a string. It is an optional value and may not be assigned.
`onSerialDataMessage`	To register callback functions that are invoked when the main process receives a WebSocket response for electrogram telemetry data.	A function that must take in a message with the following format: ● `type` of `data`. ● `dataType` of `egram`. ● `data` that is a stringified JSON object containing electrogram data in the format outlined in section 6.2.7.1.1.
`onSerialErrorMessage`	To register callback functions	A function that must take in a

	that are invoked when the main process receives a WebSocket response for an unknown error.	message with the following format: ● `type` of `error`. ● `error` of type `string`.
`removeSerialConnectionMessageListener`	To clear any and all callbacks registered using `onSerialConnectionMessage`.	None.
`removeSerialActionMessageListener`	To clear any and all callbacks registered using `onSerialActionMessage`.	None.
`removeSerialDataMessageListener`	To clear any and all callbacks registered using `onSerialDataMessage`.	None.
`removeSerialErrorMessageListener`	To clear any and all callbacks registered using `onSerialErrorMessage`.	None.

Figure 6.2.8.3.b: Tabular expression table for invoking `serialInitialize`.

	Result	
Condition	*Successful Result*	*Failed Result*
No serial number is provided.	X	The error message must be handled via a callback registered using `onSerialConnectionMessage`. Message returned is in the format: ● `type` of `connection`. ● `connectionType` of `initialize`. ● `status` of `failed`. ● `message` of ”No pacemaker ID provided”.
The pacemaker is already connected.	X	The error message must be handled via a callback registered using `onSerialConnectionMessage`. Message returned is in the format: ● `type` of `connection`. ● `connectionType` of `initialize`. ● `status` of `failed`. ● `message` of ”Already

		connected”.
The Python serial backend failed to establish a handshake.	X	The error message must be handled via a callback registered using `onSerialConnectionMessage`. Message returned is in the format: ● `type` of `connection`. ● `connectionType` of `initialize`. ● `status` of `failed`.
The Python serial backend successfully establishes a handshake.	The success message must be handled via a callback registered using `onSerialConnectionMessage`. Message returned is in the format: ● `type` of `connection`. ● `connectionType` of `initialize`. ● `status` of `success`.	X

Figure 6.2.8.3.b: Tabular expression table for invoking `serialDisconnect`.

	Result	
Condition	*Successful Result*	*Failed Result*
The Python serial backend fails to reconnect to the pacemaker.	X	The error message must be handled via a callback registered using `onSerialConnectionMessage`. Message returned is in the format: ● `type` of `connection`. ● `connectionType` of `disconnect`. ● `status` of `failed`. ● `message` of ”Already disconnected”.
The pacemaker was already disconnected.	X	The error message must be handled via a callback registered using `onSerialConnectionMessage`. Message returned is in the format: ● `type` of `connection`. ● `connectionType` of `disconnect`. ● `status` of `failed`. ● `message` of ”Not connected”.

The Python serial backend successfully disconnects from the pacemaker.	The success message must be handled via a callback registered using `onSerialConnectionMessage`. Message returned is in the format: ● `type` of `connection`. ● `connectionType` of `disconnect`. ● `status` of `success`.	X

Figure 6.2.8.3.c: Tabular expression table for invoking `serialSendParameters`.

	Result	
Condition	*Successful Result*	*Failed Result*
The pacemaker is not connected.	X	The error message must be handled via a callback registered using `onSerialActionMessage`. Message returned is in the format: ● `type` of `action`. ● `action` of `send_parameters`. ● `status` of `failed`. ● `message` of ”Not connected”.
No parameters are provided.	X	The error message must be handled via a callback registered using `onSerialActionMessage`. Message returned is in the format: ● `type` of `action`. ● `action` of `send_parameters`. ● `status` of `failed`. ● `message` of ”No parameters provided”.
The Python serial backend fails to set the mode and parameters.	X	The error message must be handled via a callback registered using `onSerialActionMessage`. Message returned is in the format: ● `type` of `action`. ● `action` of `send_parameters`. ● `status` of `failed`. ● `message` of ”Failed to send parameters”.

The Python serial backend successfully sets the mode and parameters.	The success message must be handled via a callback registered using `onSerialActionMessage`. Message returned is in the format: ● `type` of `action`. ● `action` of `send_parameters`. ● `status` of `success`.	X

Figure 6.2.8.3.d: Tabular expression table for invoking `serialToggleEgram`.

	Result	
Condition	*Successful Result*	*Failed Result*
The pacemaker is not connected.	X	The error message must be handled via a callback registered using `onSerialActionMessage`. Message returned is in the format: ● `type` of `action`. ● `action` of `toggle_egram`. ● `status` of `failed`. ● `message` of ”Not connected”.
An invalid toggling mode was supplied.	X	The error message must be handled via a callback registered using `onSerialActionMessage`. Message returned is in the format: ● `type` of `action`. ● `action` of `toggle_egram`. ● `status` of `failed`. ● `message` of ”Invalid egram mode”.
The Python serial backend fails to toggle electrogram telemetry.	X	The error message must be handled via a callback registered using `onSerialActionMessage`. Message returned is in the format: ● `type` of `action`. ● `action` of `toggle_egram`. ● `status` of `failed`.
The Python serial backend successfully toggles	The success message must be handled via a callback registered using `onSerialActionMessage`.	X

electrogram telemetry.	Message returned is in the format: ● `type` of `action`. ● `action` of `toggle_egram`. ● `status` of `success`.	

7 Validation and Verification

The Validation and Verification (V&V) process for the HeartFlow desktop application is essential in ensuring that the product meets user requirements and performs its intended functions effectively. This section outlines the methodologies and results of the V&V process conducted for HeartFlow, emphasizing both the validation and verification phases.

7.1 Validation

This section validates the selected requirements by demonstrating how they effectively address the core problem and contribute to a comprehensive solution. The primary goal of this section is to provide sufficient evidence and reasoning to establish a rationale behind each requirement.

7.1.1 Problem Restatement

The primary problem the DCM is meant to address is the need for a reliable, user-friendly, secure interface for healthcare professionals to monitor and manage pacemaker parameters. Specifically, the DCM is designed to interface with the pacemaker firmware in a bidirectional relationship, supplying integral data regarding parameter and mode changes and receiving telemetry data. By providing a streamlined and intuitive interface with robust security measures and seamless integration with the new firmware, the DCM aims to improve the efficiency and safety of pacemaker management.

7.1.2 Validation Strategy

The following factors will be used to establish the validity of the requirements:

- Each requirement will be mapped to the specific sections or clauses within the Boston Scientific specification for pacemaker control software, demonstrating that the requirements are grounded by established industry standards and best practices.
- Each requirement will have a clear and concise rationale, explaining why it is essential for addressing the problem and achieving problem objectives.
- Each requirement will have a set of user-centred design considerations, showing how they contribute to a positive user experience for healthcare professionals.
- Each requirement will have a set of risk mitigation strategies, addressing the potential risks associated with pacemaker controller software.

7.1.3 Validation of Requirements

- **Requirement:** The home page should be the default page opened upon launching of the application. The home page should permit the user to either log in to their existing account or register a new account. The home page should convey the logo branding and a short slogan, with the software release version.
    - **Sections:**
        - 3.2.2 DCM User Interface - Point 1. The user interface shall be capable of utilizing and managing windows for the display of text and graphics.
    - **Rationale:** Essential as an entry point to the main application window, which is required for a user-friendly interface.
    - **User-Centered Design Considerations:** Provides a clear call to action to log in or register a new account to continue using the DCM.
    - **Risk Mitigation:** Ensures that confidential user data can not be accessed immediately upon login. The user must first log in to access potentially sensitive data.
- **Requirement:** Users should be able to register with a unique username that is no less than 3 characters long. Passwords must meet minimum security standards, including at least 8 characters with one special character, and must be confirmed correctly in the "Confirm Password" field. The "Pacemaker Serial Number" is required and cannot be left blank. If all fields meet these conditions, the system should register the user and store the data in its local database. Otherwise, an error should indicate which input field needs correction, such as a username that is too short, a weak password, or a mismatched confirmation password.
    - **Sections:** N/A
    - **Rationale:** Registration of unique accounts is essential to store separate pacemaker settings for each individual pacemaker. This ensures accountability in medical environments since each set of settings are tied to a healthcare provider’s account.
    - **User-Centered Design Considerations:** Unique accounts are also essential for user-centred design by allowing users to personalize their settings and preferences, improving their workflow efficiency.
    - **Risk Mitigation:** Finally, having unique accounts prevents pacemaker settings from being mixed around, preventing potentially fatal accidents. The security of the unique accounts is also ensured by using sufficiently strong passwords with an industry-standard hashing algorithm, argon2.
- **Requirement:** The DCM should allow registered users to log in using their previously created credentials. A valid username and password must be entered to access the system. If the username exists and the correct password is provided, the user should access the dashboard. If the password is incorrect, an error message should prompt the user to try again. If the username does not exist, the system should display an error notifying the user that the account is not found. The login process ensures that only authorized users can access pacemaker settings.
    - **Sections:** N/A

- **Rationale:** Verification of login credentials is essential to prevent any unauthorized changes to pacemaker settings since unauthorized users would not be able to log in. Further, it is an industry best practice to secure applications handling sensitive data with logins. Finally, implementing logins paves the way for implementing more advanced security measures like two-factor authentication, session timeouts, and activity logging.
- **User-Centered Design Considerations:** Logins allow users to save their preferences on separate accounts, allowing for a tailored experience. Adding a login also gives users a sense of ownership and control over their data. Finally, the login signifies that the system is safe and secure, increasing user confidence and trust.
- **Risk Mitigation:** Logins act as the first line of defence against unauthorized access to highly sensitive pacemaker setting information. They force accountability since settings and their changes can be tied back to a specific account. Finally, logins meet industry standard regulations such as HIPAA by enforcing access control and user authentication.

- **Requirement:** The DCM dashboard should serve as the main interface within the HeartFlow software, providing users with access to essential functions and device information. At a minimum, the dashboard should display all programmable pacemaker parameters, enabling users to view and adjust the settings of the connected device directly from the interface. When a pacemaker is connected, the device's serial number should be prominently displayed, ensuring users can verify the correct device connection. Additionally, the dashboard should show the current software release version, date, and time, offering clear context for the software environment and ensuring users know the precise system status.
  - **Sections:**
    - 3.2.2 DCM User Interface - Point 3. The user interface shall be capable of displaying all programmable parameters for review and modification.
    - 3.2.2 DCM User Interface - Point 4. The user interface shall be capable of visually indicating when the DCM and the device are communicating.
    - 3.2.3 DCM Utility Functions - Point 1. The About function displays the following: Application model number, Application software revision number currently in use, DCM serial number, Institution name
  - **Rationale:** A comprehensive dashboard provides a centralized location for users to access critical pacemaker information and perform key tasks like changing programmable parameters. The dashboard also prominently displays important information such as real-time information and serial number to minimize accidents and ensure optimal patient care.
  - **User-Centered Design Considerations:** Having a centralized dashboard organizes information in a clear and logical format, minimizing cognitive load and facilitating quick comprehension of important information. Using a central dashboard also means it can provide quick error prevention.
  - **Risk Mitigation:** The central dashboard provides clear and accurate information about the connected pacemaker and parameters, reducing the risk of errors that

could compromise patient safety. The central dashboard also ensures device integrity to prevent unintended modifications to the wrong device.

- **Requirement:** The dashboard should also reflect user information, identifying the currently signed-in user and, where applicable, facilitating easy and secure multi-user access, with a capacity for storing login credentials for up to ten users locally. This feature allows multiple users to retain access settings specific to their profiles. A visual notification should inform the user if a different pacemaker device is detected than the one previously connected. In the event of telemetry disconnection, the dashboard should indicate both the connectivity status and specific causes of connection loss, such as high noise levels or a failed connection, allowing users to troubleshoot quickly.
    - **Sections:**
        - 3.2.2 DCM User Interface - Point 5. The user interface shall be capable of visually indicating when telemetry is lost due to the device being out of range.
        - 3.2.2 DCM User Interface - Point 6. The user interface shall be capable of visually indicating when telemetry is lost due to noise.
        - 3.2.2 DCM User Interface - Point 7. The user interface shall be capable of visually indicating when a different PACEMAKER device is approached than was previously interrogated.
    - **Rationale:** Displaying the currently logged-in user reinforces security and accountability, ensuring that actions are attributed to the correct individual. Clear and informative error messages about telemetry disconnections facilitate rapid troubleshooting and minimize downtime, ensuring continuous patient monitoring.
    - **User-Centered Design Considerations:** The interface for managing user accounts (adding, removing, modifying) should be easy to understand and use, even for those less familiar with technology. Device connection and telemetry disconnection notifications should be clear, prominent, and informative, without being overly disruptive.
    - **Risk Mitigation:** Multi-user support with secure login credentials helps prevent unauthorized access to patient data and pacemaker settings. Notifying users of device changes minimizes the risk of unintended modifications to the wrong pacemaker, enhancing patient safety. Storing user credentials locally allows for quick access and minimizes the risk of data loss due to network connectivity issues.
- **Requirement:** The dashboard should support essential user actions, such as adjusting and saving pacemaker parameters and safely logging out of their account as needed. To enhance user control, the system should permit users to terminate the telemetry connection with the pacemaker at any time during use. For data continuity, user-specific configurations should save automatically after each session, so programmable settings are retained when users return.
    - **Sections:** N/A
    - **Rationale:** Supporting essential actions like parameter adjustment and logout directly within the dashboard streamlines the user experience and reduces unnecessary navigation. Automatically saving user-specific configurations

ensures that settings are preserved across sessions, preventing data loss and improving efficiency.
    - **User-Centered Design Considerations:** Controls for adjusting parameters, saving changes, and logging out should be clearly labelled and easily accessible. Implement confirmation dialogues for actions that could have significant consequences, such as logging out or terminating the telemetry connection, to prevent accidental actions.
    - **Risk Mitigation:** Automatic saving of user configurations mitigates the risk of losing important settings due to unexpected interruptions or system errors. A secure logout mechanism helps protect patient data and device settings from unauthorized access.
- **Requirement**: The DCM must establish a robust and efficient mechanism for serial communication to facilitate seamless interaction between the desktop application and the pacemaker hardware. The integration must support a custom UART protocol, as defined in the Serial Protocol Documentation. The system must ensure bidirectional communication for telemetry data retrieval, real-time electrogram updates, mode and parameter adjustments, and other critical interactions.
    - **Sections**:
        - 3.2.6 PG-Telemetry - Point 2. The DCM shall use some medium, such as RF or ultrasound, that is safe and legal to use, for maintaining consistent telemetry with an implanted medical device
    - **Rationale**: This requirement ensures reliable and efficient communication between the desktop application and pacemaker hardware, which is essential for the system to perform its intended functions, including device management, telemetry visualization, and parameter control. Bidirectional communication is critical for real-time data exchange, enabling clinicians to monitor and adjust pacemaker settings promptly. The use of a custom UART protocol ensures the communication mechanism is tailored to the system's specific needs, addressing unique operational and safety requirements.
    - **User-Centered Design Considerations**: The serial communication system prioritizes responsiveness and accuracy to support real-time electrogram visualization and seamless telemetry updates. This ensures clinicians can make informed decisions quickly, improving their workflow and enhancing patient care. By supporting parameter adjustments and mode changes through the serial interface, the design ensures intuitive and efficient device control, aligning with the needs of healthcare professionals.
    - **Risk Mitigation**: The use of a custom protocol ensures precise control over data formats, reducing the risk of misinterpretation or corruption during transmission promotes data integrity. Asynchronous communication mechanisms are implemented to minimize delays, ensuring real-time responsiveness. For additional security, only authenticated sessions can initiate or maintain communication, preventing unauthorized access to pacemaker data or controls. Polling mechanisms detect device disconnection promptly, ensuring the system provides feedback to the user in case of interrupted communication.

- **Requirement**: The real-time electrogram must provide a dynamic and continuous graphical representation of the atrial and ventricular voltage data received from the pacemaker via UART serial communication. The graph should plot voltage on the y-axis and time on the x-axis, ensuring a clear and comprehensible display of cardiac electrical activity. It should dynamically update to reflect the most recent telemetry data, maintaining minimal latency to deliver near-instantaneous feedback.
    - **Sections**:
        - 4.7 Real-Time Electrograms - Point 2: Real-time internal electrograms shall be made available from a surface electrogram. The real-time electrogram transmission shall be re-initiated if the telemetry link was broken during the transmission of electrograms and then reestablished.
        - 4.7.1 Electrogram Viewing - Point 1: The user shall have the option of viewing the electrogram on the screen.
    - **Rationale**: The real-time electrogram is a crucial feature that allows healthcare professionals to monitor cardiac electrical activity, ensuring accurate and timely feedback on the heart's performance. A dynamic, continuously updating graph ensures that clinicians can detect changes in heart rhythm or voltage patterns, which is vital for diagnosing irregularities or adjusting pacemaker settings in real-time. By representing voltage on the y-axis and time on the x-axis, the electrogram provides a clear and easy-to-interpret visualization of the heart's electrical signals, supporting effective decision-making during patient care.
    - **User-Centered Design Considerations**: The electrogram display must be intuitive and easy to interpret, allowing clinicians to quickly assess the heart's electrical activity at a glance. The real-time updates and dynamic nature of the graph ensure clinicians can follow trends in voltage data, aiding in identifying any abnormalities or trends in the patient's heart activity. This feature prioritizes minimal latency, enabling nearly instantaneous feedback, which is essential for making swift adjustments to pacemaker settings during clinical procedures.
    - **Risk Mitigation**: To prevent misinterpretation of cardiac data, the system will ensure the electrogram graph is rendered with high precision, maintaining the integrity of voltage readings, ultimately supporting the accuracy of the data shown. To promote latency minimization, the graph will be designed to update dynamically with minimal delay by employing efficient data handling and processing algorithms, ensuring near-instantaneous feedback of telemetry data. For extra security, the system will incorporate secure telemetry transmission via UART, ensuring that no unauthorized access or tampering with electrogram data occurs, preserving patient confidentiality.
-

7.2 Verification

The verification process for HeartFlow was designed to ensure that each functional and non-functional requirement of HeartFlow is correctly implemented, and accurately performs as intended. Each component is verified against predefined test case requirements and expected

outcomes to confirm robust functionality, accuracy, and reliability. For example, one component of the testing process involved simulating all possible input permutations and React state configurations, and observing the resulting behaviours to ensure the interface responds according to specifications. Verification was conducted with attention to cross-platform compatibility, as HeartFlow was developed on Mac and Linux but tested on Mac, Windows, and Linux systems to guarantee smooth performance across all supported operating systems. For the tests on serial-associated components, the pacemaker was utilized to receive the UART data and its firmware was modified to use its LEDs to signify the data it receives. The following sections are broken down by each set of tests that were performed.

7.2.1 Page Navigation Tests

The page navigation tests were conducted to verify the application's navigation functionality across various user interfaces. Each test case was designed to gauge the app's response when users interacted with navigation elements, ensuring that they were directed to the expected pages based on their actions.

Figure 7.2.1.a: Tabularly expressed test cases for page navigation.

Test Case	Initial Page	Navigation Action	Expected Page	Actual Outcome	Pass/ Fail
PN1	Home	Click “Register New User”	Registration	As expected	Pass
PN2	Home	Click “Log In User”	Login	As expected	Pass
PN3	Registration	Click “Back”	Home	As expected	Pass
PN4	Registration	Click “Register User”, credentials valid	Home	As expected	Pass
PN5	Registration	Click “Register User”, credentials invalid	Registration	As expected	Pass
PN6	Login	Click “Back”	Home	As expected	Pass
PN7	Login	Click “Log In User”, credentials valid	Dashboard	As expected	Pass
PN8	Login	Click “Log In User”, credentials invalid	Login	As expected	Pass
PN9	Dashboard	Click “Sign Out”	Home	As expected	Pass
PN10	Any	Click any button not listed above	Initial page	As expected	Pass

7.2.2 User Registration Tests

The user registration tests were designed to evaluate the functionality and robustness of the user registration process within the application. Each test case aimed to assess how the system handles various input scenarios during the registration process, ensuring that appropriate feedback is provided to users based on their input.

Figure 7.2.2.a: Tabularly expressed test cases for user registration.

Test Case	Conditions/Inputs	Expected Outcome	Actual Outcome	Pass/Fail
UR1	Valid username, valid password, matching confirm password, valid serial number.	Successful registration; account found in users.json and home page is loaded	As expected	Pass
UR2	Username < 3 characters	Error: Short username	As expected	Pass
UR3	Username not unique; already exists in system database	Error: User exists	As expected	Pass
UR4	Valid username, password < 8 chars	Error: Weak password	As expected	Pass
UR5	Valid username, password has 8+ chars but no special char	Error: Weak password	As expected	Pass
UR6	Valid username, non-matching passwords	Error: Password mismatch	As expected	Pass
UR7	Valid username, valid password, null serial number	Error: Null serial number	As expected	Pass

7.2.3 User Login Tests

The user login tests were designed to evaluate the effectiveness and reliability of the login functionality within the application. Each test case aimed to assess how the system responds to various login attempts based on the validity of the username and password provided by the user.

Figure 7.2.3.a: Tabularly expressed test cases for user login.

Test Case	Username	Password	Expected Outcome	Actual Outcome	Pass/Fail
UL1	Exists	Correct	Successful login; dashboard is loaded with information found in users.json	As expected	Pass
UL2	Exists	Incorrect	Error: Incorrect password	As expected	Pass
UL3	Does not exist	Any	Error: User not found	As expected	Pass

7.2.4 Telemetry Status and Heartbeat Tests

The telemetry status and heartbeat tests were conducted to verify the accuracy of the telemetry functionality and the corresponding heart animation based on user actions and system states. Each test case focused on different scenarios related to the telemetry status and the heartbeat animation to ensure proper operation.

Figure 7.2.4.a: Tabularly expressed test cases for telemetry status and heartbeat tests.

Test Case	Conditions	Expected Heart Animation	Expected Telemetry Status	Actual Outcome	Pass/Fail
T1	Telemetry is terminated using the Disconnect button	None	Disconnected	As expected	Pass
T2	Telemetry is continued or begun using	Beating	Connected	As expected	Pass

	the Connect button				
T3	User logs in after having an active telemetry session before previously signing out	None	Disconnected	As expected	Pass
T4	User logs in after being disconnected while previously signing out	None	Disconnected	As expected	Pass
T5	User registers and logs in for the first time	None	Disconnected	As expected	Pass

7.2.5 Mode Selection Widget State Tests

The Mode Selection Widget State Tests were conducted to evaluate the functionality and behaviour of the mode selection feature in the application. Each test case focused on a specific action and assessed the expected states of the various mode indicators to ensure proper functionality during user interactions. The test inputs were the user's selection of pacemaker mode, and the test outputs were the system's declaration of CSS states for each of the mode labels. In short, the selected mode should correspond to the only active label where all other labels are in normal state.

Figure 7.2.5.a: Tabularly expressed test cases for model selection widget states.
N denotes normal, and A denotes active.

Test Case	Input	Expected System Output States											Results	
	Action (Select)	OFF	AOO	AAI	AOOR	AAIR	VOO	VVI	VOOR	VVIR	DDD	DDDR	Actual Outcome	Pass/Fail
MS1	Disconnect	N	N	N	N	N	N	N	N	N	N	N	As expected	Pass
MS2	OFF	A	N	N	N	N	N	N	N	N	N	N	As expected	Pass
MS3	AOO	N	A	N	N	N	N	N	N	N	N	N	As expected	Pass
MS4	AAI	N	N	A	N	N	N	N	N	N	N	N	As expected	Pass
MS5	AOOR	N	N	N	A	N	N	N	N	N	N	N	As expected	Pass
MS6	AAIR	N	N	N	N	A	N	N	N	N	N	N	As expected	Pass
MS7	VOO	N	N	N	N	N	A	N	N	N	N	N	As expected	Pass
MS8	VVI	N	N	N	N	N	N	A	N	N	N	N	As expected	Pass
MS9	VOOR	N	N	N	N	N	N	N	A	N	N	N	As expected	Pass
MS10	VVIR	N	N	N	N	N	N	N	N	A	N	N	As expected	Pass
MS11	DDD	N	N	N	N	N	N	N	N	N	A	N	As expected	Pass
MS12	DDDR	N	N	N	N	N	N	N	N	N	N	A	As expected	Pass

7.2.6 Parameter Input Field State Tests

The parameter input field state tests were designed to verify the behaviour of the input fields for pacemaker parameters based on the selected mode. Each test case assessed the expected state of the amplitude, pulse width, refractory period, and lower rate limit fields to ensure they conform to the specifications associated with the various operational modes.

Figure 7.2.6.a: Tabularly expressed test cases for parameter input field states.

Disc. denotes Disconnected, D denotes disabled, and A denotes active.

Test Case	Input	Expected System Output States															Result	
	Mode	LRL	URL	AAMP	APW	ARP	ASN	VAMP	VPW	VRP	VSN	RXNT	RCVT	RF	AVD	ACT	Actual Outcome	Pass/ Fail
PS1	Disc.	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	As expected	Pass
PS2	OFF	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	As expected	Pass
PS3	AOO	A	A	A	A	A	D	D	D	D	D	D	D	D	D	D	As expected	Pass
PS4	AAI	A	A	A	A	A	A	D	D	D	D	D	D	D	D	D	As expected	Pass
PS5	AOOR	A	A	A	A	A	D	D	D	D	D	A	A	A	D	A	As expected	Pass
PS6	AAIR	A	A	A	A	A	A	D	D	D	D	A	A	A	D	A	As expected	Pass
PS7	VOO	A	A	D	D	D	D	A	A	A	D	D	D	D	D	D	As expected	Pass
PS8	VVI	A	A	D	D	D	D	A	A	A	A	D	D	D	D	D	As expected	Pass
PS9	VOOR	A	A	D	D	D	D	A	A	A	D	A	A	A	D	A	As expected	Pass
PS10	VVIR	A	A	D	D	D	D	A	A	A	A	A	A	A	D	A	As expected	Pass
PS11	DDD	A	A	A	A	A	A	A	A	A	A	D	D	D	A	D	As expected	Pass
PS12	DDDR	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	As expected	Pass

7.2.7 Parameter Validation Tests

The parameter validation tests were conducted to ensure that the application correctly enforces the valid range of input values for the programmable parameters of the pacemaker. Each test case aimed to verify that inputs outside the defined bounds cause appropriate error messages, while valid inputs are accepted and processed correctly. Note that some of the tests were impossible to conduct because the DCM does not permit the entering of the “-” symbol as negative values are never permitted. For the sensitivities, the lowest possible value to enter is 0, which is a valid value for ASN and VSN and thus a lower bound test can not be conducted. This is still expected behaviour however. Additionally, it is not possible to have a bounds error on the ACT because it is a slider, so it was not included in this section’s testing.

Figure 7.2.7.a: Tabularly expressed test cases for parameter verification.

Test Case	Parameter	Value	Test Type	Expected Outcome	Actual Outcome	Pass/Fail
PV1	LRL	20	Min bound	Bounds error	As expected	Pass
PV2	LRL	100	Valid	Submission	As expected	Pass
PV3	LRL	200	Max bound	Bounds error	As expected	Pass
PV4	URL	45	Min bound	Bounds error	As expected	Pass
PV5	URL	100	Valid	Submission	As expected	Pass
PV6	URL	180	Max bound	Bounds error	As expected	Pass

PV7	AAMP	0.4	Min bound	Bounds error	As expected	Pass
PV8	AAMP	3.1	Valid	Submission	As expected	Pass
PV9	AAMP	6	Max bound	Bounds error	As expected	Pass
PV10	APW	0	Min bound	Bounds error	As expected	Pass
PV11	APW	15	Valid	Submission	As expected	Pass
PV12	APW	40	Max bound	Bounds error	As expected	Pass
PV13	ARP	100	Min bound	Bounds error	As expected	Pass
PV14	ARP	200	Valid	Submission	As expected	Pass
PV15	ARP	600	Max bound	Bounds error	As expected	Pass
PV16	ASN	N/A	Min bound	Impossible	As expected	Pass
PV17	ASN	3	Valid	Submission	As expected	Pass
PV18	ASN	6	Max bound	Bounds error	As expected	Pass
PV19	VAMP	0.4	Min bound	Bounds error	As expected	Pass
PV20	VAMP	3.1	Valid	Submission	As expected	Pass
PV21	VAMP	6	Max bound	Bounds error	As expected	Pass
PS22	VPW	0	Min bound	Bounds error	As expected	Pass
PS23	VPW	15	Valid	Submission	As expected	Pass
PS24	VPW	40	Max bound	Bounds error	As expected	Pass
PS25	VRP	100	Min bound	Bounds error	As expected	Pass
PS26	VRP	200	Valid	Submission	As expected	Pass
PS27	VRP	600	Max bound	Bounds error	As expected	Pass
PS28	VSN	N/A	Min bound	Impossible	As expected	Pass
PS29	VSN	3	Valid	Submission	As expected	Pass
PS30	VSN	6	Max bound	Bounds error	As expected	Pass
PS31	RXNT	0	Min bound	Bounds error	As expected	Pass
PS32	RXNT	20	Valid	Submission	As expected	Pass
PS33	RXNT	60	Max bound	Bounds error	As expected	Pass
PS34	RCVT	0	Min bound	Bounds error	As expected	Pass
PS35	RCVT	20	Valid	Submission	As expected	Pass
PS36	RCVT	200	Max bound	Bounds error	As expected	Pass
PS37	RF	0	Min bound	Bounds error	As expected	Pass

PS38	RF	10	Valid	Submission	As expected	Pass
PS39	RF	20	Max bound	Bounds error	As expected	Pass
PS40	AVD	10	Min bound	Bounds error	As expected	Pass
PS41	AVD	100	Valid	Submission	As expected	Pass
PS42	AVD	1000	Max bound	Bounds error	As expected	Pass

7.2.8 DCM Serial Integration Testing

For each of the listed test cases, distinct firmware programs were created which emulates the pacemaker's serial functionality. Thus, the expected outcomes as seen in the pacemaker's firmware, which appear in outbound message tests, are not universal and were subjectively selected during the testing process. This was done for two principal reasons. Firstly, we wanted to be able to have flexibility in the pacemaker's output to signify if the correct serial data was received. This was done entirely using different LED colours which were designed to illustrate the UART header being read using the custom protocol, and sometimes the payload. The target LED would be lit up if and only if the entire packet header and payload were received as expected, ensuring a full verification for the entire data sequence. Further, this was done for ease of debugging and to adhere to the purpose of these tests. As the purpose of this test was to test the functionality of the DCM, we wanted to isolate any possible errors caused by the internal logic of the pacemaker so that it could be confirmed that each function was expected individually, and together as an integrated system. This is important if changes were to be made to either the pacemaker firmware or the DCM software so that new bugs may be isolated to which system is causing the bug.

Figure 7.2.8.a: Tabularly expressed test cases for parameter verification.

Test Case	MSG Test	MSG Direction	Condition	Expected Outcome	Actual Outcome	Pass/Fail
SI1	0x01	Outbound	User clicks Connect button	Pacemaker lights red	As expected	Pass
SI2	0x01	Inbound	User clicks Connect button	Connected status appears	As expected	Pass
SI3	0x02	Outbound	User clicked Connect button and connect status appeared 10 ms prior	Pacemaker lights green	As expected	Pass
SI4	0x02	Inbound	User clicked Connect button and connect status appeared > 10 ms prior	Connected status persists while user idling	As expected	Pass
SI5	0x03	Outbound	User clicks Submit to send parameters and mode	Pacemaker lights white	As expected	Pass
SI6	0x03	Inbound	User clicks Submit to send parameters and mode	Toast message appears verifying successful parameter acceptance to the pacemaker firmware	As expected	Pass

SI7	0x04	Outbound	User clicks Submit to send parameters and mode and DCM receives identical 0x03 packet as was previously transmitted	Pacemaker lights blue	As expected	Pass
SI8	0x05	Outbound	User clicks Connect button	Pacemaker lights red	As expected	Pass
SI9	0x05	Outbound	User clicks Disconnect button	Pacemaker lights blue	As expected	Pass
SI10	0x05	Outbound	User clicks Sign Out button	Pacemaker lights green	As expected	Pass
SI11	0x05	Inbound	User clicks Connect button	Electrogram data mapped onto the electrogram chart as seen in HeartView	As expected	Pass
SI12	0x05	Inbound	User clicks Disconnect button	Electrogram data appending is halted and becomes static	As expected	Pass

7.2.9 Electrogram Testing

The electrogram lies at the core of the data visualization component of HeartFlow. To test the electrogram, an emulated pacemaker firmware was developed on the FRDM-K64F board for the purpose of testing. The rationale for developing an emulated pacemaker firmware rather than incorporating the true firmware is outlined in Serial Testing. Several diverse cases of electrogram data test cases were developed to be tested on the DCM's live electrogram to analyze its behaviour and ensure that its functionality is as expected.

Figure 7.2.9.a: Tabularly expressed test cases for electrogram rendering verification.
Let A represent the stream of atrium voltage readings and V represent the stream of ventricle voltage readings.

Test Case	Input Emulated Atrial Electrogram Data	Input Emulated Ventricular Electrogram Data	Expected Outcome	Actual Outcome	Pass/ Fail
EG1	A = [0, 0, …, 0]	B = [0, 0, …, 0]	Two flat series at y=0	As expected	Pass
EG2	A = [10, 10, …, 10]	B = [0, 0, …, 0]	Red series at y=10 and one blue series at y=0	As expected	Pass
EG3	A = [0, 0, …, 0]	B = [10, 10, …, 10]	Red series at y=0 and blue series at y=10	As expected	Pass
EG4	A = [0, 0.01, 0.02, …, 10]	B = [10, 9.99, 9.98, …, 0]	Red series as y=x and blue series as y=10-x	As expected	Pass
EG5	A = [0, 0, …, 0, 5, 0, …, 0]	B = [0, 0, …, 0, -5, 0, …, 0]	Two flat series other than one point where red impulses to 5 and blue impulses to -5	As expected	Pass
EG6	A is a sine wave	B is a cosine wave	Two sinusoidal waves, pi/2 radians out of phase	As expected	Pass
EG7	A is a rectangular pulse	B is a triangular pulse	A red rectangular pulse and a blue triangular	As expected	Pass

			pulse		
EG8	A is a rectangular pulse train with tau=5	A is a triangular pulse train with tau=5	A red rectangular pulse train and a blue triangular pulse train, each with a period of 5 seconds.	As expected	Pass

7.2.10 Reports Testing

The reports component of HeartFlow formulate an integral part of its long-term data accessibility features. Below is a set of tabular test cases for the reports feature, designed to verify the functionality, integrity, and usability of each report in the system. These test cases cover a variety of scenarios that ensure reports are generated correctly, downloaded appropriately, and contain accurate data.

The purpose of these tests is to validate that each report generated by the DCM system functions as expected in terms of both data accuracy and usability. The Serial Log Report, Electrogram Report, Activity Log Report, and Parameter Log Report each serve distinct purposes but share the common requirement of accurate data capture, formatting, and download behavior. Verifying these tests ensures that the application produces reliable, secure, and correctly formatted CSV files, which clinicians, patients, and technicians can use for diagnosis, system auditing, and parameter management.

Figure 7.2.10.a: Tabularly expressed test cases for reports verification.

Test Case	Report Test	System Input	Expected Output	Actual Outcome	Pass /Fail
RP1	Serial Log Report	User clicks on the "Serial Log Report" button in the Reports menu	CSV file is generated with user's username, timestamps, and all UART serial communication details	As expected	Pass
RP2	Electrogram Report	User clicks on the "Electrogram Report" button in the Reports menu	CSV file with voltage readings for atrium and ventricle, including timestamps and session details	As expected	Pass
RP3	Activity Log Report	User clicks on the "Activity Log Report" button in the Reports menu	CSV file is generated containing usernames and login times since user registration	As expected	Pass
RP4	Parameter Log Report	User clicks on the "Parameter Log Report" button in the Reports menu	CSV file with time-stamped changes of parameters, including previous and new values	As expected	Pass
RP5	Serial Log Report	User requests the report when no serial data has been transmitted	An empty CSV file is generated with the header but no data entries	As expected	Pass
RP6	Electrogram Report	User clicks "Electrogram Report" while no electrogram data exists	An empty CSV file is generated with the header but no data entries	As expected	Pass
RP7	Parameter Log Report	User requests the report when no parameter	CSV file with headers, but no rows for parameter changes	As expected	Pass

		changes have occurred			

7.2.11 Compatibility Testing

HeartFlow's development and testing took place across Mac, Windows, and Linux environments to ensure a smooth user experience across various operating systems. Specifically, development occurred on Mac and Linux operating systems, and the built executables were verified on Mac, Windows, and Linux. All modules were tested for proper rendering and responsive interaction on each platform, validating that interface elements were displayed correctly, navigation was smooth, and all database interactions were reliable. Each identified issue was resolved, and tests were repeated to confirm full compatibility across systems.