Device Controller-Monitor (DCM) Documentation

MECHTRON 3K04 Assignment 1
Group 4
Andrew De Rango, Ali Hussin, Ethan Otteson,
Marco Tan, Rafael Toameh
Friday, October 25, 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 on to the pacemaker microcontroller.

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

Off - 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.

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 Assignment 1 Device Controller-Monitor 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 Assignment 1. 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, 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.

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	Any	Invalid	Any	Any	Any	Error: Password is

special characters						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 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 Telemetry Status

The DCM should indicate telemetry status with a "Connected" or "Disconnected" label. When telemetry is connected, the display should show an animated, beating heart to confirm live connection. Conversely, when telemetry is disconnected, the heart image should remain static. Users should see the current status displayed on the dashboard so they can monitor whether or not the telemetry connection is active. Also, if telemetry is lost, the reason for the disconnectivity should be displayed.

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

Telemetry Status	Condition Triggering Status Change	Heart Animation	Displayed Status
Active	Telemetry session started	Animated (beating)	Connected
Inactive	Telemetry session terminated	Static (paused)	Disconnected

4.5.2 Parameter Requirements

The DCM should offer four operational modes: AOO, VOO, AAI, and VVI each with its corresponding set of adjustable parameters. Additionally, users should be able to turn the mode to 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. 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 available for modification, and their usage in different pacing modes.

Parameter	Types	Summary
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.
Lower rate limit	N/A	Establishes the minimum heart rate allowed before the pacemaker initiates a corrective pulse.

The requirements for applying new parameters are summarized below.

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

Condition	Amplitude	Pulse Width	Refractory Period	Lower Rate Limit	Result
User applies new parameters	Valid	Valid	Valid	Valid	Database successfully updated and parameters applied
Atrium/ventricle amplitude out of range	Invalid	Any	Any	Any	Error: Amplitude out of valid range
Atrium/ventricle pulse width out of range	Any	Invalid	Any	Any	Error: Pulse width out of valid range
Atrial/ventricular refractory period out of range	Any	Any	Invalid	Any	Error: refractory period out of valid range
Lower rate limit out of range	Any	Any	Any	Invalid	Error: Lower rate limit out of valid range
User terminates telemetry	Any	Any	Any	Any	Mode switched to OFF

4.6 Expected Changes

Because of the limited scope of this assignment, 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 Changes to the User Interface

As it currently stands, the user interface of the DCM primarily handles user input for programmable parameters for each included mode, saving and loading of previously programmed parameters, and login and registration functionality. However, in future iterations of the DCM, new requirements would necessitate the following changes:

- The DCM must include any new pacemaker modes added. This would require extending the current UI for more modes and their associated parameters.
- The DCM must display real-time pacemaker information, such as a live electrogram. How this data is represented can depend, though some of the anticipated UI changes would be:

    - The implementation of graphing functionality to display real-time values such as cardiac muscle potential, output voltage of the pacemaker, and voltages sensed by the pacemaker, among other continuous values.
    - Displays for values that are averaged within some window of time, such as heart rate.
- The DCM must be able to display the different failure modes when telemetry is terminated, either through user termination, physical disconnection, or too much noise in the signal. Currently, the UI only displays a placeholder for the connection indicator. This placeholder will have to be modified to accommodate this change.
- The DCM should allow each user to interface with multiple pacemakers, saving a profile for each pacemaker under the user. As it currently stands, the DCM is designed with one pacemaker tied to each user. Some of the UI changes anticipated would be:
    - Interface to switch between pacemaker profiles.
    - An interface for creating new pacemaker profiles.
- The DCM must be localized for specific regions and languages. This would necessitate that the UI support multiple languages with different localizations.

4.6.2 Changes to the Backend

Changes to the UI would also necessitate changes to the main application process. For every functionality change, changes would have to be made such that the main process contains the modules to handle said behaviour. Below are some of these possible changes:

- The DCM must include any new pacemaker modes added.
    - This would require an overhaul of the main process logic to accommodate the new modes.
- The DCM must display real-time pacemaker information. This necessitates the following changes:
    - The main process must have a module to read data from the pacemaker. Most likely, the method of reading data would be via serial input and output. This cannot be worked on separately from the pacemaker as both the pacemaker and the DCM must communicate with the same communication protocol.
- The DCM must be able to display the different failure modes when telemetry is terminated.
    - This also requires a serial input and output module. However, separate modules that work in conjunction with the serial module would have to monitor specific parameters, as well as perform noise filtering and regulation, hardware identification, and manual termination.
- The DCM should allow each user to interface with multiple pacemakers, saving a profile for each pacemaker under the user.
    - This necessitates an overhaul of user registration and data-saving modules, as the schema for a user would have to be modified to accommodate multiple pacemakers with separate profiles.
    - Because of the increased complexity in the possible state changes the application goes through, the current method of storing data in the renderer

process and the main process would have to be overhauled to use a more robust solution, including an overhaul of the APIs used to retrieve profile data, and a better method of storing data on the client side.

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 login

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.

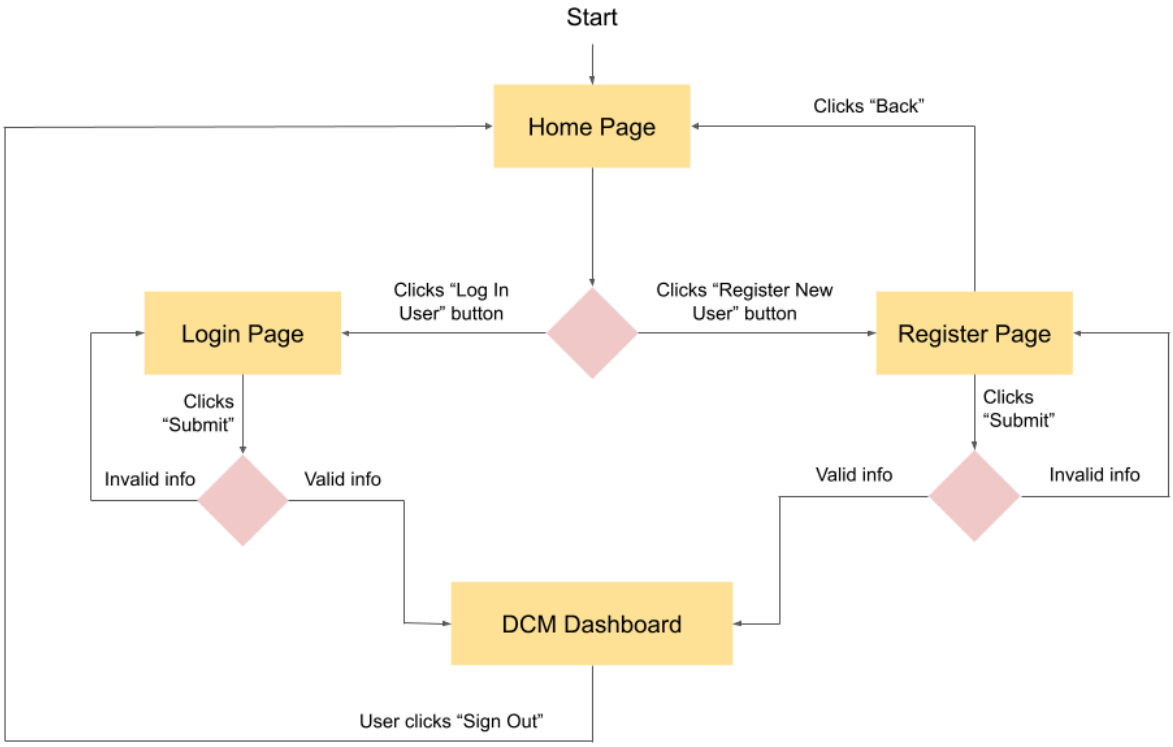

Figure 5.2.a: Page flow diagram for the HeartFlow application.

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 v1.0.0") 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.

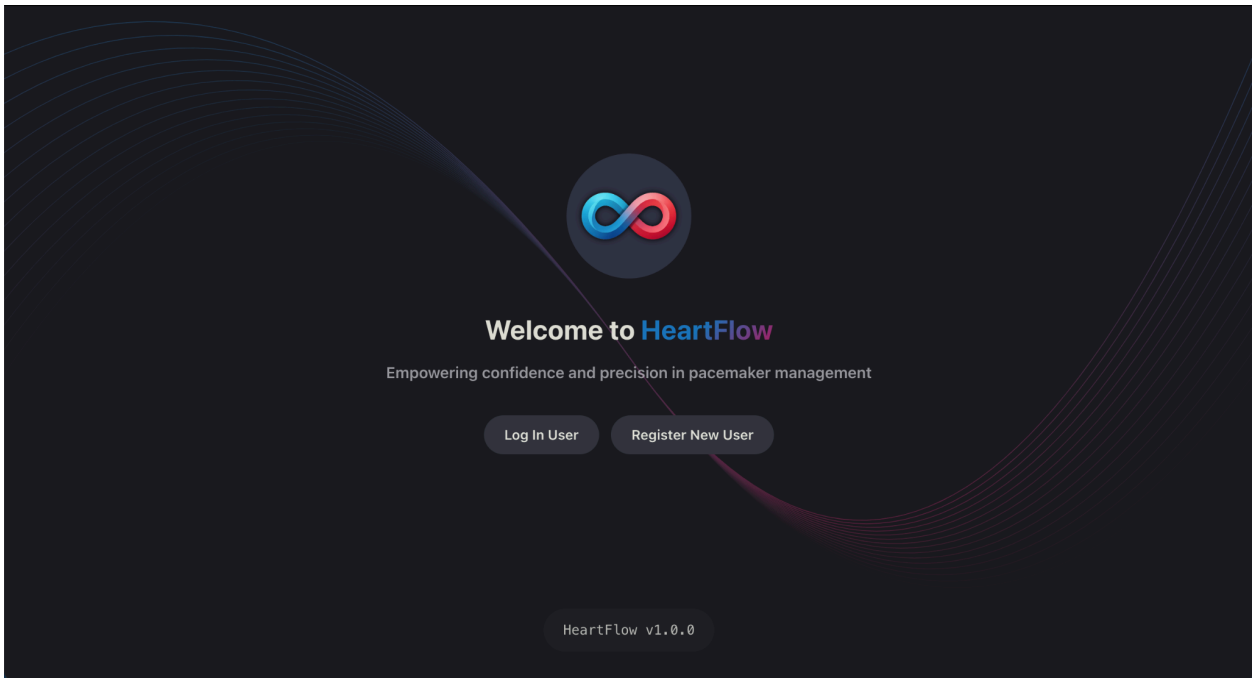

Figure 5.3.a: Screen capture of HeartFlow's home page.

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.

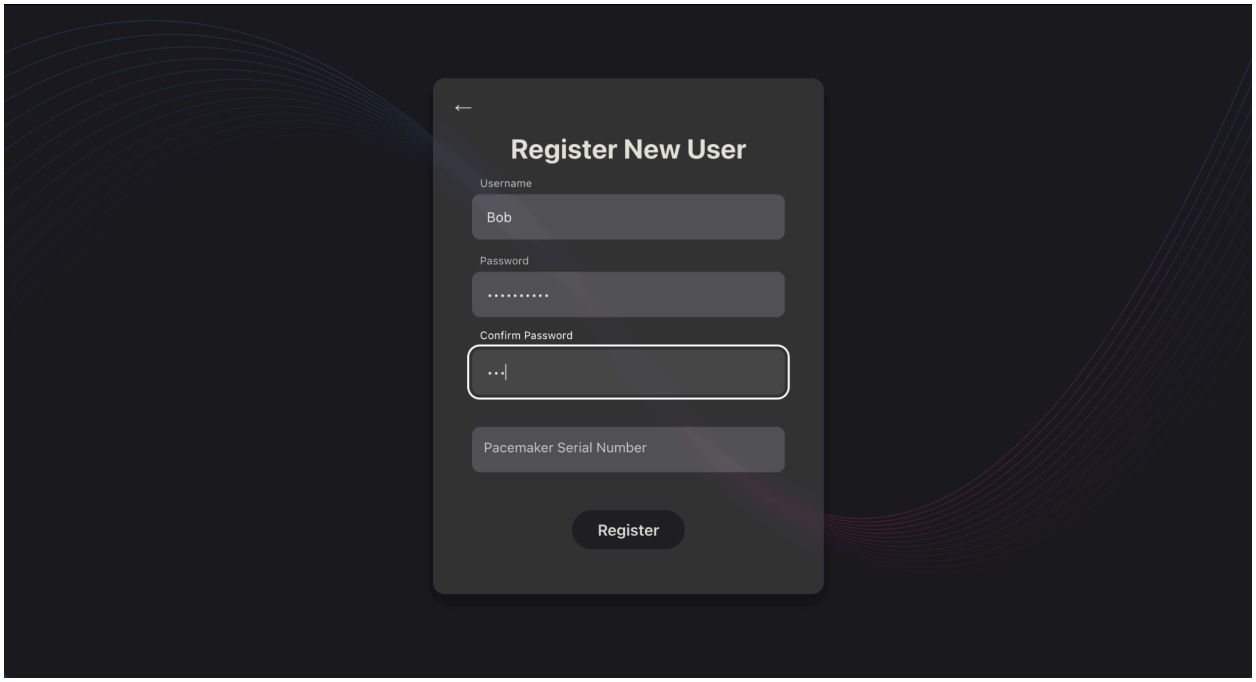

Figure 5.4.a: HeartView registration page graphical design.

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.

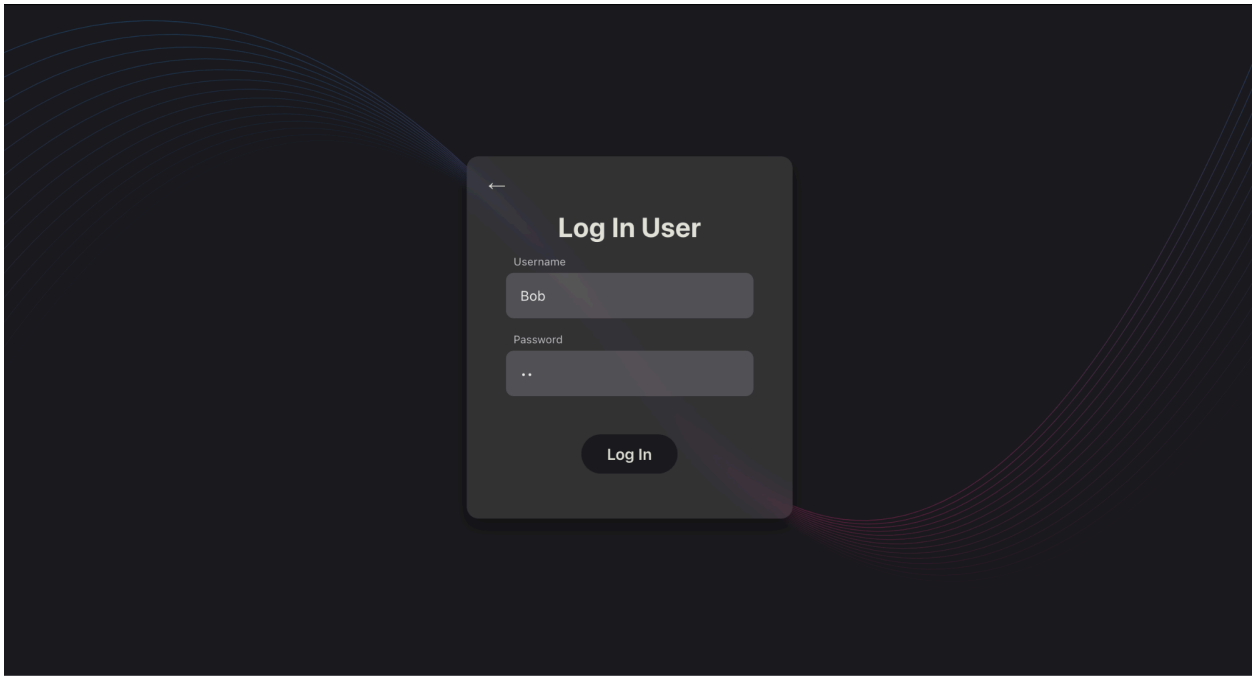

Figure 5.5.a: HeartView login page graphical design.

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.

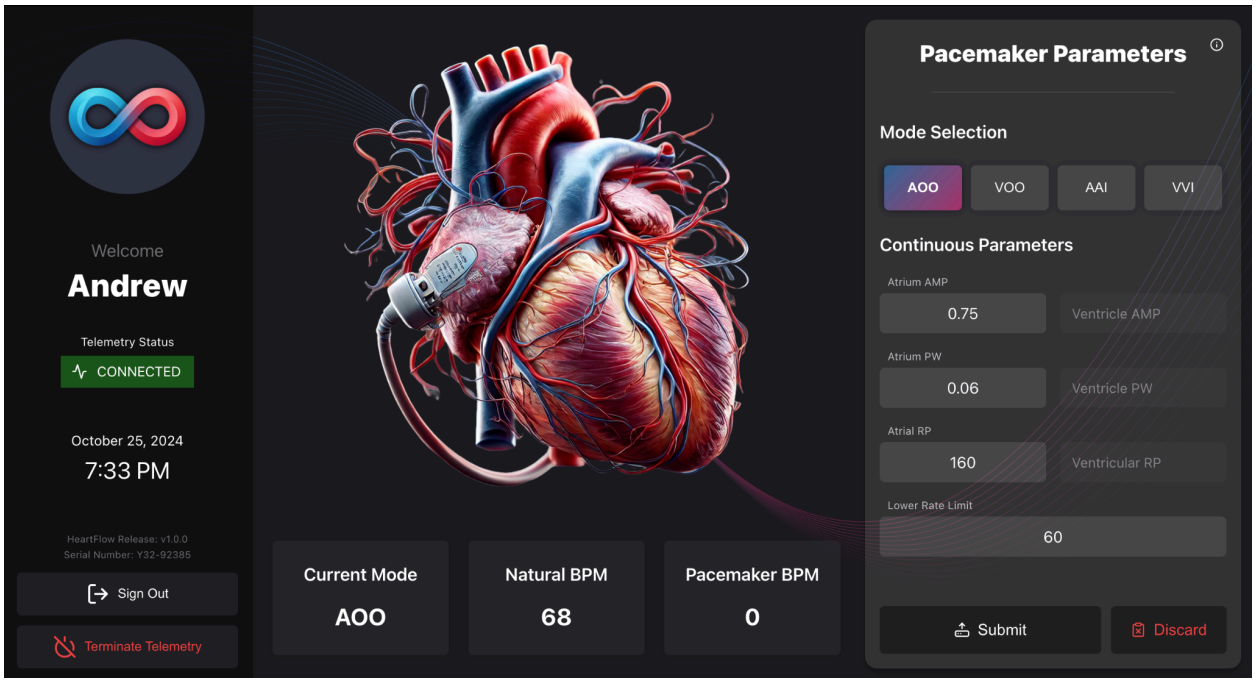

Figure 5.6.a: HeartView dashboard graphical design with mode AOO selected.

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 'Terminate Telemetry' button is positioned below to stop telemetry operations with a single click.

The right sidebar, titled "Pacemaker Parameters," houses four mode selection widgets that allow users to switch between different operational modes: AOO, VOO, AAI, and VVI. 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.

In the center of the dashboard, a visual representation of a heart with an animated pacemaker beating is displayed, dynamically reflecting the current telemetry status. This animation only activates when telemetry is connected and operational, providing a clear visual cue. Additionally, three widgets are situated prominently in this central area, indicating the current mode (OFF, AOO, VOO, AAI, VVI), the natural BPM, and the pacemaker BPM.

5.6.2 Starting and Stopping Telemetry

Telemetry functionality is a critical aspect of the DCM dashboard, allowing real-time data transmission between the pacemaker and the application. The design includes straightforward controls to start and stop telemetry, ensuring that users can engage or disengage with their device with minimal friction. When telemetry is initiated by selecting a mode and inputting valid parameters, real-time data from the pacemaker is captured and displayed, allowing users to monitor performance and make necessary adjustments. This capability is essential for users to manage their pacemakers effectively, especially in situations where immediate data access is required.

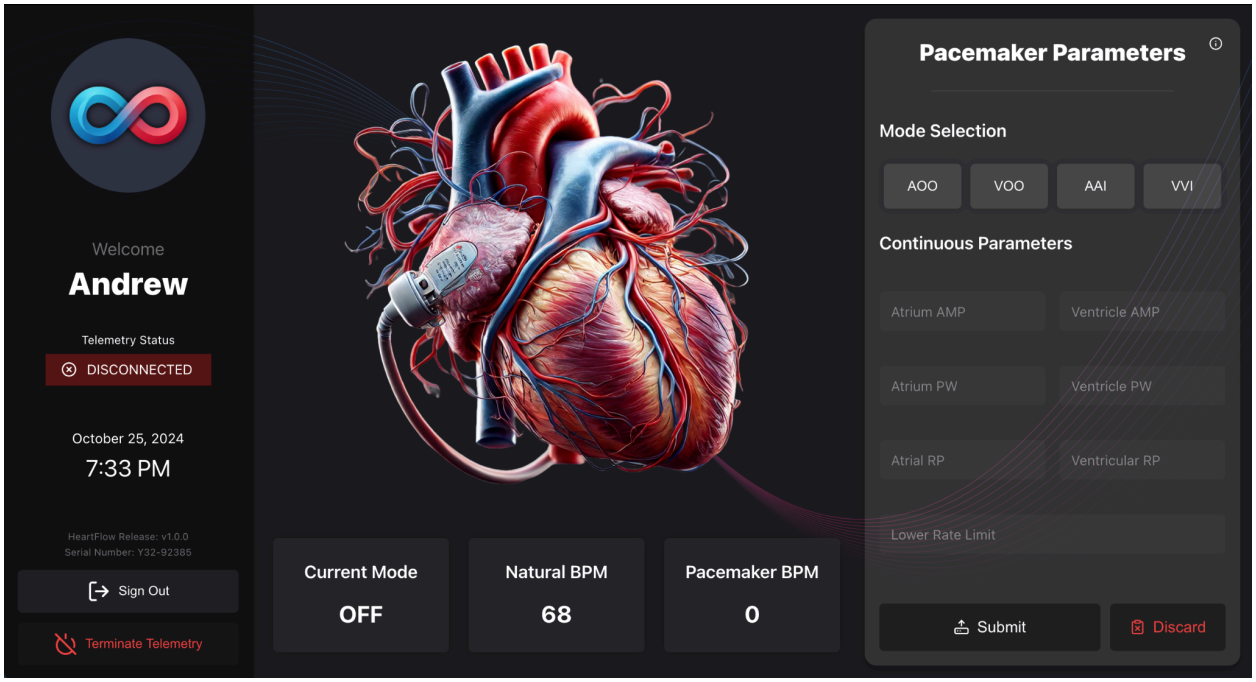

Figure 5.6.2.a: Graphical interface of the dashboard when telemetry is terminated. Notice the disabled input fields and termination button.

5.6.3 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 such as AOO, VOO, AAI, and VVI, 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.

5.6.3.1 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:

- **Atrium Amplitude (Atrium AMP)**: 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.
- **Ventricle Amplitude (Ventricle AMP)**: 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.

- **Atrium Pulse Width (Atrium PW)**: 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.
- **Ventricle Pulse Width (Ventricle PW)**: 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.
- **Atrial Refractory Period (Atrial RP)**: 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.
- **Ventricular Refractory Period (Ventricular RP)**: 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.
- **Lower Rate Limit**: 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.

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.

5.6.3.2 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 5.6.3.2.a: Summary of programmable parameters and their valid ranges.

Programmable Parameter	Units	Range Min	Range Max	Modes
Atrium Amplitude (AMP)	mV	0.5	5	AOO, AAI
Ventricle Amplitude (AMP)	mV	0.5	5	VOO, VVI
Atrium Pulse Width (PW)	ms	0.05	1.90	AOO, AAI
Ventricle Pulse Width (PW)	ms	0.05	1.90	VOO, VVI
Atrial Refractory Period (RP)	ms	150	500	AOO, AAI
Ventricular Refractory Period (RP)	ms	150	500	VOO, VVI

Lower Rate Limit	bpm	30	175	AOO, VOO, AAI, VVI

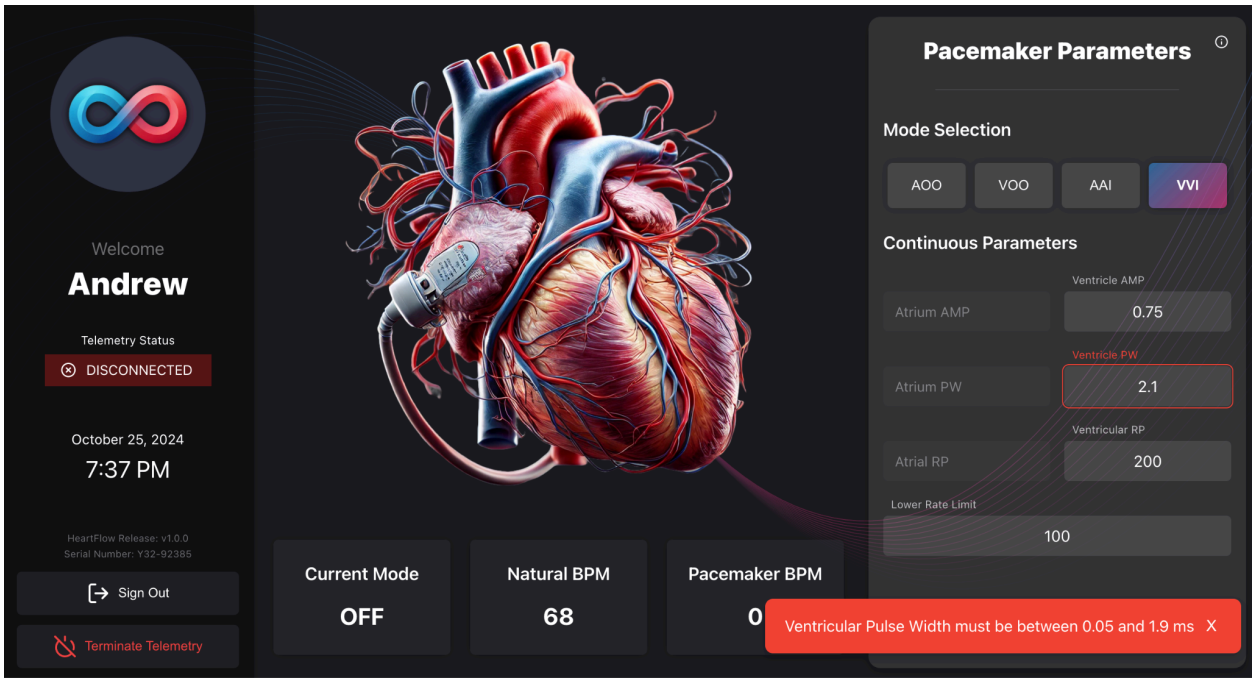

Figure 5.6.3.2.a: Example of the styling when an erroneous value is entered when attempting to begin telemetry using the VVI mode. The valid range is specified to support users in correcting their inputs.

5.6.3.2 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.

4.6.3.3 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.

5.7 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. 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 it, 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 (descriptions of which can be found in 6 Implementation):

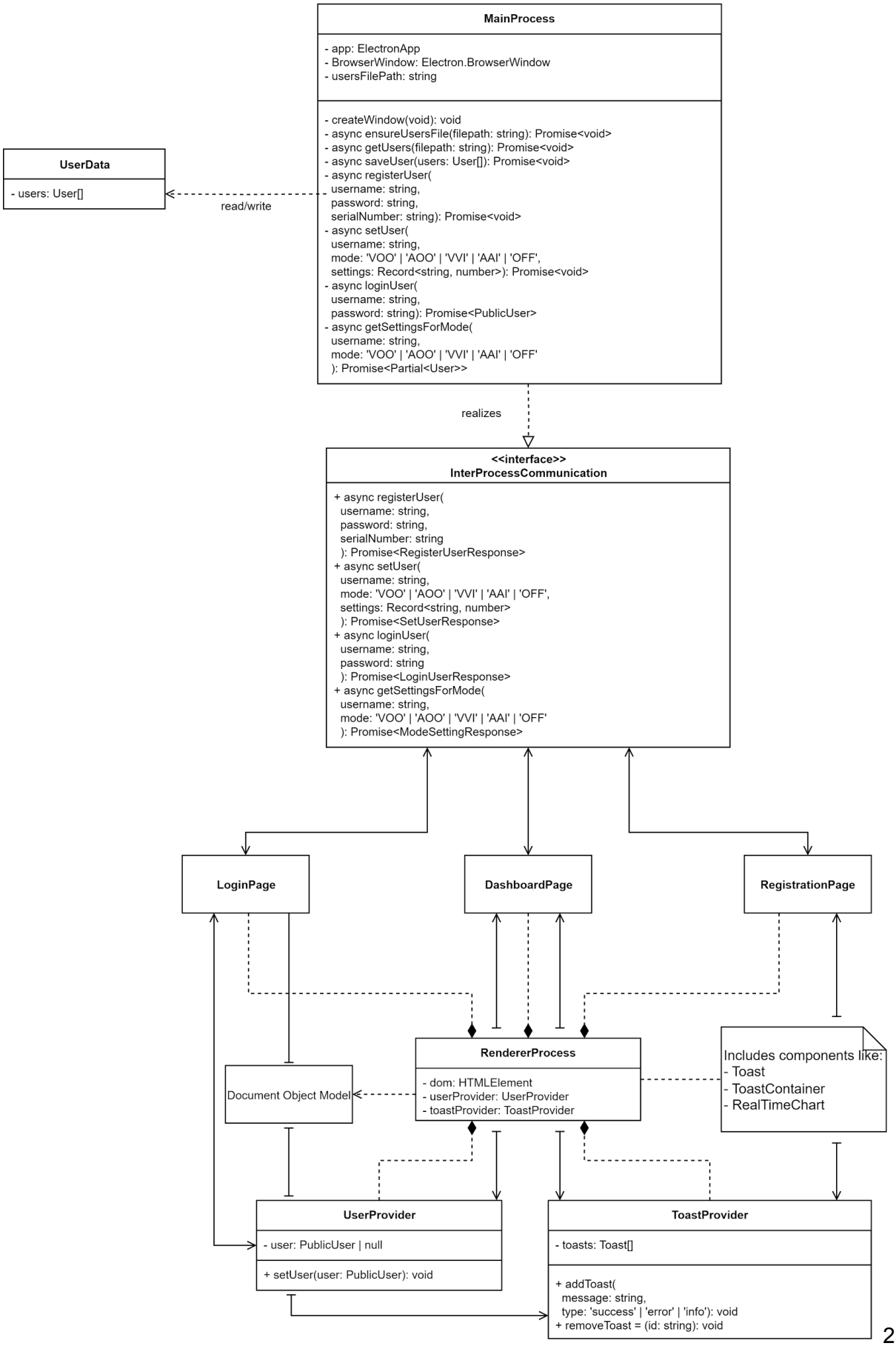
MainProcess
- app: ElectronApp
- BrowserWindow: Electron.BrowserWindow
- usersFilePath: string
- createWindow(void): void
- async ensureUsersFile(filepath: string): Promise<void>
- async getUsers(filepath: string): Promise<void>
- async saveUser(users: User[]): Promise<void>
- async registerUser(
username: string,
password: string,
serialNumber: string): Promise<void>
- async setUser(
username: string,
mode: 'VOO' | 'AOO' | 'VVI' | 'AAI' | 'OFF',
settings: Record<string, number>): Promise<void>
- async loginUser(
username: string,
password: string): Promise<PublicUser>
- async getSettingsForMode(
username: string,
mode: 'VOO' | 'AOO' | 'VVI' | 'AAI' | 'OFF'
): Promise<Partial<User>>
UserData
- users: User[]
read/write
realizes
<<interface>>
InterProcessCommunication
+ async registerUser(
username: string,
password: string,
serialNumber: string
): Promise<RegisterUserResponse>
+ async setUser(
username: string,
mode: 'VOO' | 'AOO' | 'VVI' | 'AAI' | 'OFF',
settings: Record<string, number>
): Promise<SetUserResponse>
+ async loginUser(
username: string,
password: string
): Promise<LoginUserResponse>
+ async getSettingsForMode(
username: string,
mode: 'VOO' | 'AOO' | 'VVI' | 'AAI' | 'OFF'
): Promise<ModeSettingResponse>
LoginPage
DashboardPage
RegistrationPage
RendererProcess
- dom: HTMLElement
- userProvider: UserProvider
- toastProvider: ToastProvider
Document Object Model
Includes components like:
- Toast
- ToastContainer
- RealTimeChart
UserProvider
- user: PublicUser | null
+ setUser(user: PublicUser): void
ToastProvider
- toasts: Toast[]
+ addToast(
message: string,
type: 'success' | 'error' | 'info'): void
+ removeToast = (id: string): void

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

5.8 Miscellaneous Components

5.8.1 Toast-Related 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.

Table 5.8.1.a: Summary of all Toast messages.

Condition	Page	Type	Message (Result)
User successfully registers a new account, less than 10 exist in the system, and credentials are valid	Registration	Success	User registered successfully
User attempts to register a new account but their username is less than 3 characters	Registration	Error	Username must be at least 3 characters long
User attempts to register a new account but their username is valid and their password is less than 8 characters	Registration	Error	Password must be at least 8 characters long
User attempts to register a new account but their username is valid, their password is 8+ characters, and their password does not contain a special character	Registration	Error	Password must contain at least one special character
User attempts to register a new account but their username and password are valid, but the “Confirm Password” field is not equal to the “Password” field	Registration	Error	Passwords do not match
User attempts to register a new account but their username and password are valid, they properly confirm their password, but the pacemaker serial number is null.	Registration	Error	Serial number cannot be null

User attempts to register a new account with a username that already exists	Registration	Error	User already exists
User attempts to register a new account by entering valid credentials but 10+ accounts already exist in the system	Registration	Error	Maximum number of users reached
User logs in to an existing account with the correct credentials	Login	Success	User logged in successfully
User logs into an unknown account	Login	Error	User not found
User attempts to log in to an existing account with the incorrect password	Login	Error	Incorrect password
An unexpected API response is obtained or the API call fails	Login	Error	An unknown error occurred
New atrium amplitude parameter is submitted and is outside of the valid range	Dashboard	Error	Atrium Amplitude must be between 0.5 and 5 mV
New ventricle amplitude parameter is submitted and is outside of the valid range	Dashboard	Error	Ventricle Amplitude must be between 0.5 and 5 mV
New atrium pulse width parameter is submitted and is outside of the valid range	Dashboard	Error	Atrial Pulse Width must be between 0.05 and 1.9 ms
New ventricle pulse width parameter is submitted and is outside of the valid range	Dashboard	Error	Ventricular Pulse Width must be between 0.05 and 1.9 ms
New atrial refractory period parameter is submitted and is outside of the valid range	Dashboard	Error	Atrial Refractory Period must be between 150 and 500 ms
New ventricular refractory period parameter is submitted and is outside of the valid range	Dashboard	Error	Ventricular Refractory Period must be between 150 and 500 ms
New lower rate limit parameter is submitted and is outside of the valid range	Dashboard	Error	Lower Rate Limit must be between 30 and 175 bpm
New parameters submitted and are valid	Dashboard	Success	Settings sent and saved
Parameters configuration discarded	Dashboard	Information	Settings discarded

5.8.2 Real-Time Visualization Chart

Although it was not used in the current iteration of the DCM, a component for real-time data visualization with a chart was created for later use. This component renders a live-updating line chart, receiving data from the main process to display to the user. The real-time visualization chart was meant to display continuous variables such as pacemaker voltage and sensed heart voltage, among other monitored variables.

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 component is the primary interface for managing pacemaker settings in HeartFlow. The component is designed for real-time telemetry status, mode selection, and parameter configuration. It leverages Lucide-react for UI and Context API for user and notification management. Key components include **`LogoutButton`** and **`TerminateButton`** to facilitate interaction.

State management is handled through **`useState`** and **`useEffect`** hooks, tracking various parameters: **`currentTime`** (updates every second), **`communicationStatus`** (telemetry status), **`selectedMode`**, and pacemaker parameters like **`atriumAmp`**, **`ventricleAmp`**, and **`lowerRateLimit`**. Error states support validation for safe configuration.

Key functions manage interactions: **`handleTerminate`** ends telemetry and resets parameters, **`handleModeSelect`** changes pacemaker modes, **`handleInputChange`** validates inputs, **`handleDiscard`** restores previous settings, and **`handleSubmit`** submits updated

settings. Validation uses **`validateInput`** to check values and display error messages via **`addToast`**. Backend API integration enables fetching and submitting settings.

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:

- **`UserProvider:`** This context provider manages and shares the state of the currently logged-in user, including username, pacemaker serial number, and last used pacemaker mode.
- **`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 User Information

`UserProvider` contains one state variable, **`user`**, which is set using the public method **`setUser`**. This context provider is consumed by the login page when it sets the user after successful user authentication and sign-in, and by the dashboard for retrieving user information, and clearing the user state upon a successful sign-out. The relationships between components that use this context provider are highlighted below:

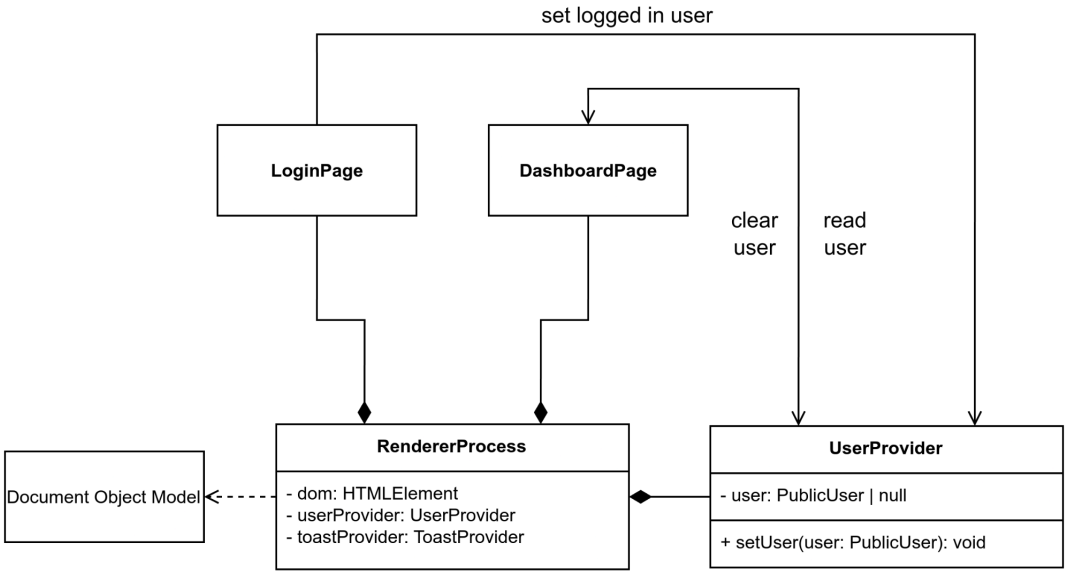

Figure 6.1.4.1.a: Class diagram displaying components and their relationships under the user information context provider.

6.1.4.2 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 are highlighted below:

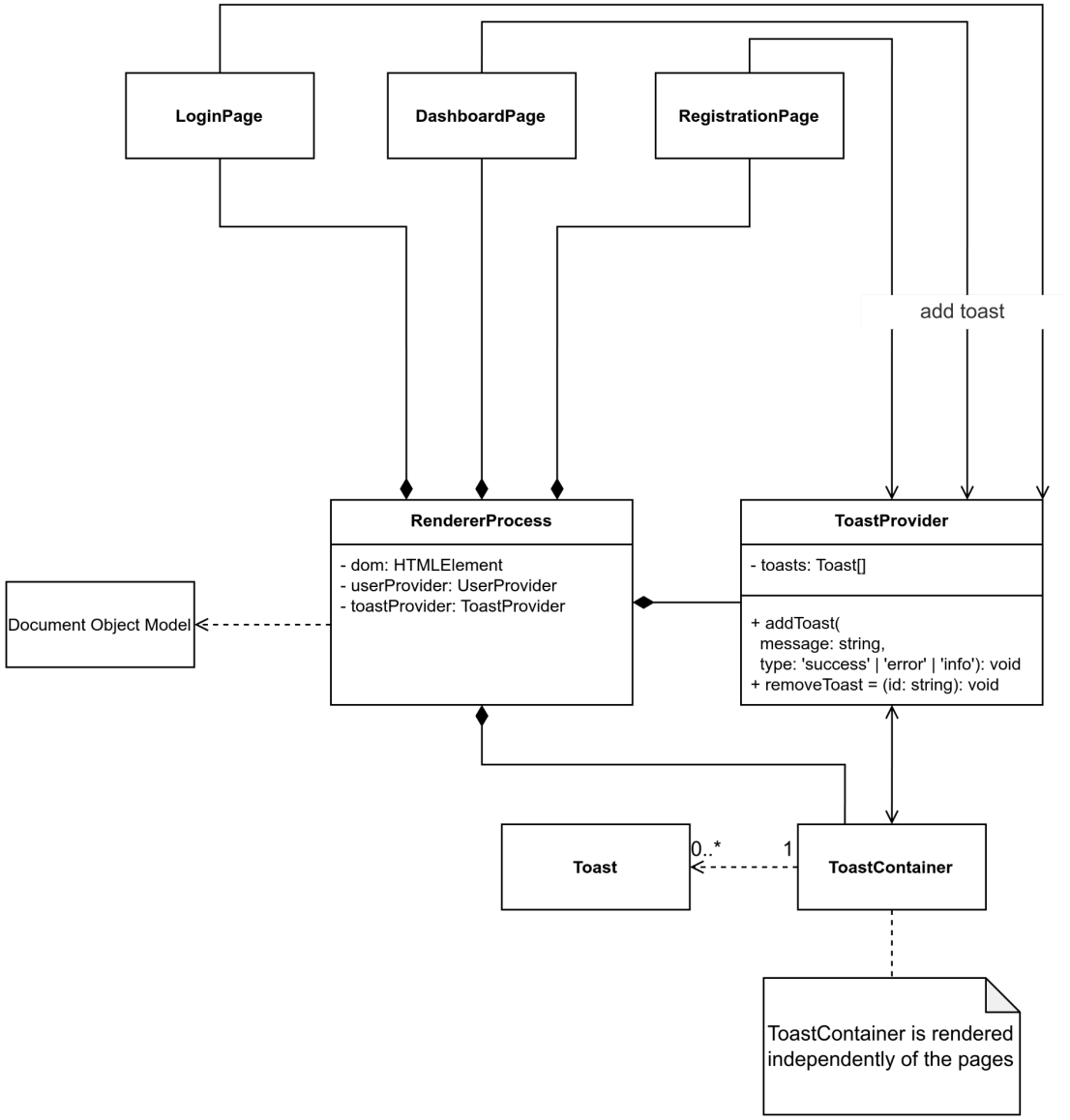

Figure 6.1.4.2.a: Class diagram displaying components and their relationships under the Toast context provider.

6.1.5 Other Components

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

6.1.5.1 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.

6.1.5.2 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:

- **`data:`** data points to render on the chart.
- **`title:`** chart title.

`RealTimeChart` will create and re-render the line graph automatically whenever **`data`** is updated. It is expected that some loop/function that consumes this component continuously updates **`data`** at a set time interval.

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:

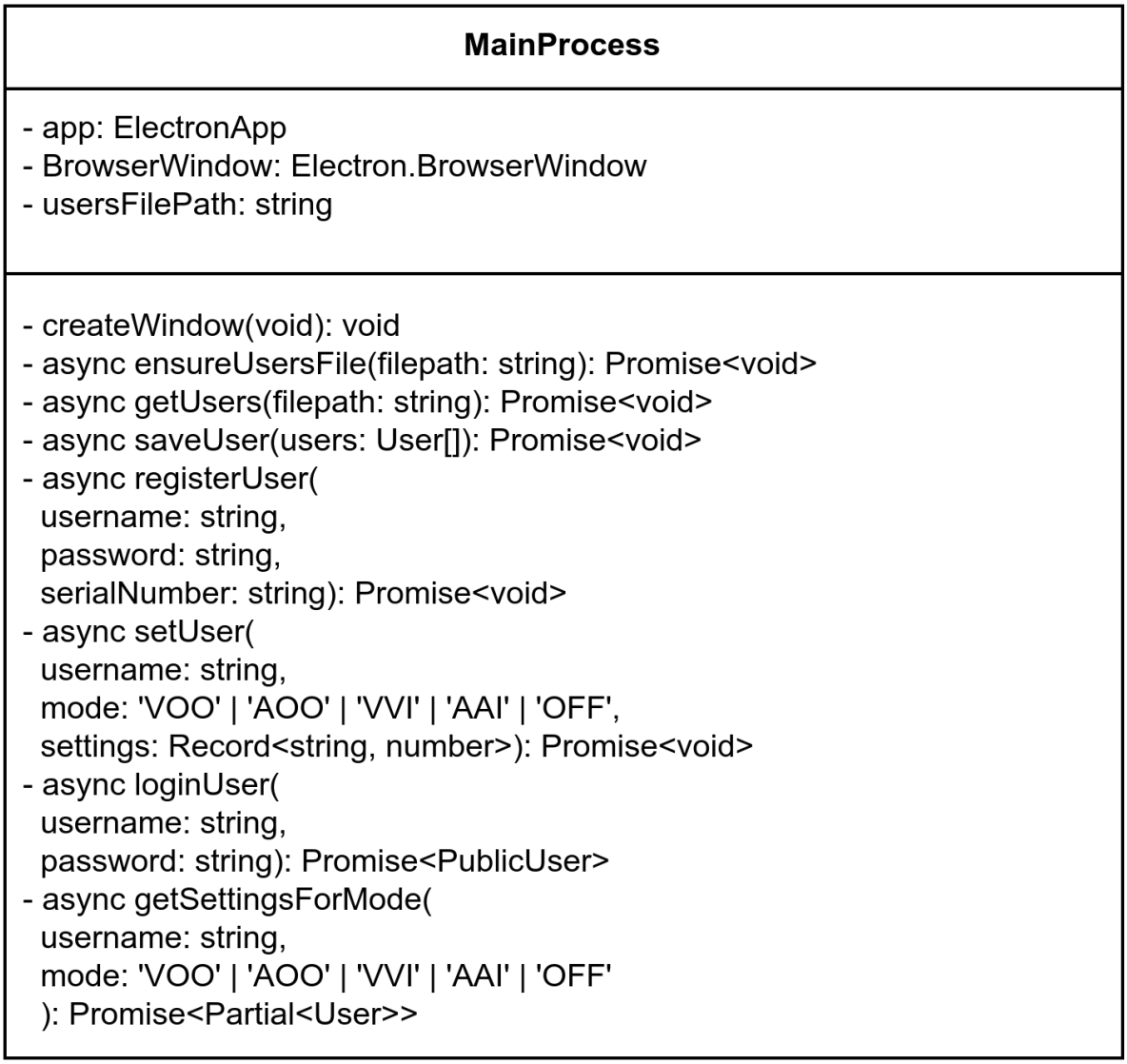

Figure 6.2.a: Class representation of the MainProcess module.

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 are 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 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. 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 6.2.6.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`	`success: boolean = true`	`success: boolean = false`

	Expected input: username: string password: string	user: PublicUser = <user>	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>

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 that was developed alongside the DCM independently. 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:** N/A
- **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.

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. 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 Terminate Telemetry button	None	Disconnected	As expected	Pass
T2	Telemetry is continued or begun using mode submission	Beating	Connected	As expected	Pass
T3	User logs in after having set the mode to OFF before previously signing out	None	Disconnected	As expected	Pass
T4	User logs in after having set the mode to anything other than OFF before previously signing out	Beating	Connected	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 (AOO, VOO, AAI, VVI) to ensure proper functionality during user interactions.

Figure 7.2.5.a: Tabularly expressed test cases for model selection widget states.

Test Case	Action	Expected AOO	Expected VOO	Expected AAI	Expected VVI	Pass/Fail
MS1	Terminate telemetry	Normal	Normal	Normal	Normal	Pass
MS2	Select AOO Mode	Active	Normal	Normal	Normal	Pass
MS3	Select VOO Mode	Normal	Active	Normal	Normal	Pass
MS4	Select AAI Mode	Normal	Normal	Active	Normal	Pass
MS5	Select VVI Mode	Normal	Normal	Normal	Active	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.

Test Case	Mode (State)	Expected A AMP	Expected A PW	Expected A RP	Expected V AMP	Expected V PW	Expected V RP	Expected LRL	Actual Outcome	Pass/ Fail
PS1	OFF	Disabled	Disabled	Disabled	Disabled	Disabled	Disabled	Disabled	As expected	Pass
PS2	AOO	Normal	Normal	Normal	Disabled	Disabled	Disabled	Normal	As expected	Pass
PS3	VOO	Disabled	Disabled	Disabled	Normal	Normal	Normal	Normal	As expected	Pass
PS4	AAI	Normal	Normal	Normal	Disabled	Disabled	Disabled	Normal	As expected	Pass
PS5	VVI	Disabled	Disabled	Disabled	Normal	Normal	Normal	Normal	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.

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

Test Case	Mode	Parameter	Value	Test Type	Expected Outcome	Actual Outcome	Pass/Fail
PV1	AOO/AAI	Atrium AMP	0	Min bound	Bounds error	As expected	Pass
PV2	AOO/AAI	Atrium AMP	1	Valid	Submission	As expected	Pass
PV3	AOO/AAI	Atrium AMP	10	Max bound	Bounds error	As expected	Pass
PV4	VOO/VVI	Ventricle AMP	0	Min bound	Bounds error	As expected	Pass
PV5	VOO/VVI	Ventricle AMP	1	Valid	Submission	As expected	Pass
PV6	VOO/VVI	Ventricle AMP	10	Max bound	Bounds error	As expected	Pass
PV7	AOO/AAI	Atrium PW	0	Min bound	Bounds error	As expected	Pass
PV8	AOO/AAI	Atrium PW	1	Valid	Submission	As expected	Pass
PV9	AOO/AAI	Atrium PW	2	Max bound	Bounds error	As expected	Pass
PV10	VOO/VVI	Ventricle PW	0	Min bound	Bounds error	As expected	Pass
PV11	VOO/VVI	Ventricle PW	1	Valid	Submission	As expected	Pass
PV12	VOO/VVI	Ventricle PW	2	Max bound	Bounds error	As expected	Pass
PV13	AOO/AAI	Atrial RP	100	Min bound	Bounds error	As expected	Pass

PV14	AOO/AAI	Atrial RP	500	Valid	Submission	As expected	Pass
PV15	AOO/AAI	Atrial RP	1000	Max bound	Bounds error	As expected	Pass
PV16	VOO/VVI	Ventricular RP	100	Min bound	Bounds error	As expected	Pass
PV17	VOO/VVI	Ventricular RP	500	Valid	Submission	As expected	Pass
PV18	VOO/VVI	Ventricular RP	1000	Max bound	Bounds error	As expected	Pass
PV19	Not OFF	LRL	0	Min bound	Bounds error	As expected	Pass
PV20	Not OFF	LRL	100	Valid	Submission	As expected	Pass
PV21	Not OFF	LRL	200	Max bound	Bounds error	As expected	Pass

7.2.8 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.