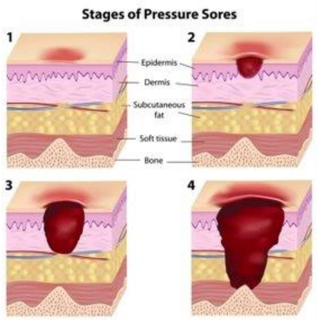
Care-Mate: Pressure Distribution Mapping System

Team 17: Benjamin Allen, Clay Griscom, Hugo Serrano, Kira Threlfall, and David Whelan
Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR
Sponsors: Jennifer Steinaur, PTA, Natha Jowers, PT, UAMS Outpatient Therapy Clinic, Fayetteville, AR. Mike Fohner, Josh Fohner.

Computer Science & Computer Engineering

Introduction

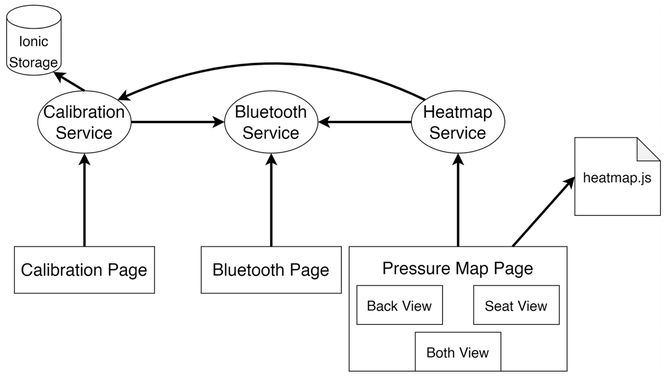


Patients that are bound to wheelchairs suffer from restricted blood flow. This is caused by pressure between boney areas and the wheelchair itself. This restricted blood flow leads to the formation of pressure ulcers. These pressure ulcers can burrow down into the flesh, decrease the patient's quality of life, and increase risk of infection.

Currently the only way to address these pressure ulcers is through preventative measures. The patient must be constantly adjusted, and the chair kept clean. However, the caretaker has no feedback on where the high-pressure areas are occurring. In this project we create an application that will interface with a pressure mapping system.

Backend Organization

The **Backend** is the portion of the application code responsible for management of data and application logic. It is split into **services**, which are classes that have a single logical concern.



CalibrationService stores calibration information and calibrates input data. Calibration data is saved between sessions, so it is unnecessary to re-calibrate after each app launch.

HeatmapService formats data for use by the heatmap.js library.

BluetoothService manages connection to the pressure sensor and acquisition of data. The BluetoothService class is a contract implemented by two subclasses. LocalBluetoothService creates random data for testing during development, allowing for faster iteration. HC06BluetoothService connects to the pressure pad Bluetooth transmitter and decodes the serial data it sends. Both services expose data in the same manner, so dependent services are unconcerned with which one is used.

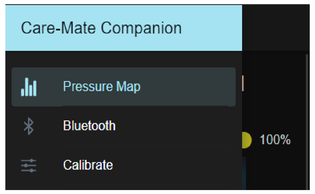
We also developed an event system to allow data to be passed from BluetoothService to its dependent services without constant polling. These events can also trigger a refresh in the views.

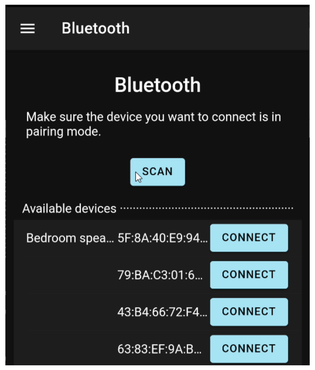
Frontend

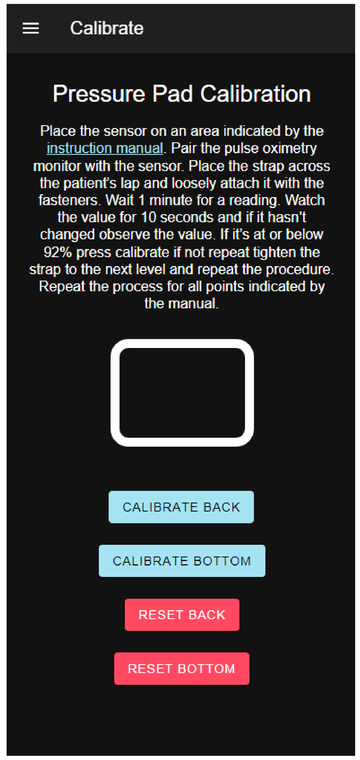
Our visuals needed to be neat and simple so that important information wasn't lost. A light-blue color with a dark theme was chosen for visual appeal and eye strain reduction

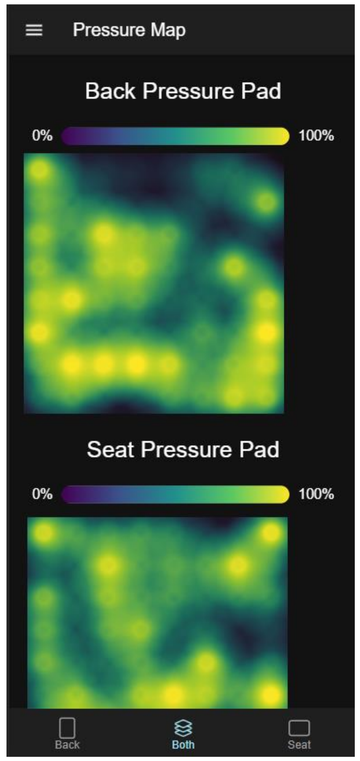
Navigation of the application is handled by the side menu. A list of pages will appear, highlighting the current page, that will allow you to switch from one page to another.

The **Bluetooth** page initially displays a button to scan for Bluetooth devices. Once found a list of Bluetooth devices appears below that will allow the device to connect.









The **Pressure Map** page displays the live visual pressure maps in a color-blind friendly format, using the "viridis" color gradient. A tabs section located at the bottom of the page allows for quick map changes between bottom, back, or both pressure maps.

The **Calibration** page displays a set of instructions that will help the user calibrate the pressure pad. The blue buttons allow the user to calibrate a pad, while the red buttons allow the user to reset a pad.

Testing

Software – To test the software a set of mock data was created that mimicked the sensor data that would be sent to the app. This data was a series of 64 8-bit number sets divided by one 8-bit number that would act as a signal bit telling the app that the whole data set has been sent and a new one is starting. To test the Bluetooth software components before the hardware was complete an Arduino fitted with an HC-06 Bluetooth module was set to send mock data to the app. This tested the Bluetooth connection and data collection inside the app with the specific Bluetooth module used on the hardware.

Hardware – The hardware was tested throughout the process to test each of the hardware components functionality. A set of eight LEDs were put on the board to act as testing points. These LEDs were set between the microcontroller and the multiplexer control pins. This allowed us to always know which sensor was currently being read into the micro-controller so we could test each individual sensor. We began by testing the Bluetooth module by sending signals from the microcontroller to be broadcast to a phone with a Bluetooth terminal. We tested the analog to digital converter by reading in the data of one sensor and broadcasting it to the Bluetooth terminal on a phone.

Integration Testing - Once the hardware was functioning correctly, we began testing the hardware and software components together. We began sending live sensor data from the microcontroller over Bluetooth to the application while pressing different sensors to test the end-to-end flow of data.

Challenges

The pressure pad system and application needed to be built in parallel, making integration testing difficult. To accommodate for this, we created test data in the expected format we would receive from the pressure array and left three weeks for integration testing. The main difficulties we faced in testing were mostly expected and related to data formatting: we found that the characters between 0-33 were unusable and adjusted for it. We also a delay in data transmission which caused the heatmap to become blank before redrawing. To remedy this, we ensured that the application had completely received the data before redrawing.

Future Work

While our original design goals have been met, there are still many features that could expand the impact of this project. Possible additions could be:

- Alert users when pressure exceeds a set value
- Record pressure history and use ML analysis to detect trends over time
- Customizability, such as multiple font sizes and colors
- A calibration wizard with images to better display calibration instructions