**Isaiah Johnson**

**April 14, 2019**

**Final Architectural Plan**

**QT Ultrasound LLC**

**Design Plan Summary**

QT Ultrasound, LLC is a medical technology company that has created a breast scanner that uses quantitative transmission to produce 3D breast images to aid in the detection of cancer. QT Ultrasound currently utilizes a user interface to perform tasks on the breast scanner, but the current interface is outdated and in need of an overhaul. This project aims to create a user interface that can interact directly with the scanner via a web server that is established locally on the scanner. This will allow the operator of the machine to manipulate the scanner from a multitude of devices. The ability to create an interface via javascript also allows for a more modern, user-friendly experience to those controlling the scanner.

A close up of a map

Description automatically generated**Overview of Architecture**

The architecture of this system is built around wrapping legacy software into usable Javascript notation by utilizing N-API, node.js, and C++. The current system uses C++ and the Qt library to display an interface to control the scanner. This project will take the existing C++ code used to control the scanner, and through the use of Node.js C++ Addons, will wrap the required classes into Javascript usable code, in order to allow commands to be called from the user interface.

**Current System Architecture**

The current system works as follows:

* The user interface is run through a terminal that is connected to the scanner.
* When a user clicks on a command, such as “fill tank”, a command is handled by app
* The message that is sent to app is determined within the ExamManager class if it is in reference to an exam, or to AcquisitionControl if it has to do with scanner manipulation.
* After app determines the message to be sent, it will send the message to the DA node.
* The DA node will then send a response message back to app, containing the information relevant to the message that was sent (ex. If getExam were called, da would send back a message containing exam data).
* Once the response is retrieved, the interface will do whatever manipulation is needed to display information regarding the command that was sent.

**Wireframes**

A screenshot of a cell phone

Description automatically generatedThe goal is a dashboard style interface that allows the operator to have access to as many controls as possible in the least amount of navigation. The user should not have to click to separate tabs in order to access data, as this takes longer than simply pressing a button and knowing where the data is located without have to first scan the page.

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

**UML Diagram**

Given that the user interface is written in react, a standard UML diagram cannot be used. I created a UML diagram based on the components in a hierarchical structure so the team may understand the flow of components within the application. This diagram is located in the project portfolio repo.

URL: <https://github.com/ijohnson11/CapstoneProjectPortfolio/blob/master/doc/HeirarchicalStructure.png>

**How Frontend and Backend Communicate**

The React application will connect to a broker that is centralized in the embedded system. All of the nodes will be interconnected and will contain their own broker for communicating with one another. All of the brokers will send and receive messages from the central broker, which will act as the brain of the backend. The centralized broker will handle all the data exchanges, and the frontend will subscribe to topics that the broker established in order to update components within the React application.