SYSC4907: Final Report

Graphical Control Interface for a Robotic Arm

Group #81

Supervisor: Dr. Lynn Marshall

Dhriti Aravind (101141942)
Elisha Catherasoo (101148507)
Sarah Chow (101143033)
Evan Smedley (101148695)
Jeremy Trendoff (101160306)

April 10th, 2024

Abstract

Remote, understaffed, and underserved communities struggle to have adequate access to basic healthcare. The shortage of primary care physicians creates a challenge for individuals to access the necessary preventative care resources to detect symptoms of illness. To solve this problem, our group proposed a robotic arm remotely controlled by a doctor that serves patients in remote areas. The team created a website for doctors to remotely control the movements of the robotic arm placed next to a patient. This will allow doctors to perform virtual general physical health check-ups using common medical tools, such as a flashlight, a mallet, and a tongue depressor. A key to the success of this project is the user-friendly website interface, providing doctors with an easy and efficient way to control the robotic arm remotely. This website was created with the frontend framework, React, due to the wide assortment of plugins and open-source libraries available online. The website only allows certified doctors with a valid CPSO number to access the robotic arm. As for the website's backend, it was created using the Spring Boot Java Framework. Hosted in Google Cloud Run, the backend app server acts as a pipeline between the website and the robotic arm. The doctor controls the robot through the buttons on the website. The instructions are sent to the backend server and forwarded to the arm to be executed. Finally, the robotic arm is a Yahboom robotic arm for Raspberry Pi 4B with a camera. This robotic arm has six degrees of freedom which provides more points of manipulability and a higher dexterity for irregularly shaped objects such as a tongue depressor. Using the library provided by the manufacturer and OpenCV, all necessary actions for the doctor can be implemented.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

Remote, understaffed, and underserved communities struggle to have adequate access to basic healthcare. In Ontario, access to healthcare has been a challenge. Ontario's health minister, Sylvia Jones, stated in her plan for connected and convenient care that, "...the status quo is not working. Too many people are waiting too long to get an appointment or surgery, having to travel too far to get care, and spending too much time trying to navigate our healthcare system." [30]. According to the Ontario Medical Association, "at least 1 million Ontarians do not have regular access to primary care" and it is most prevalent in northern and rural areas [31]. The shortage of primary care physicians creates a challenge for individuals to access the necessary preventative care resources to detect symptoms of illness. To solve this problem, our group proposes a robotic arm controlled by a doctor that serves patients in remote areas.

1.2 Problem Motivation and Current State of the Art Solutions

The field of remotely controlled doctor-patient interaction is an emerging field and the current applications are built for specialized diagnoses [26]. For example, there exists a robot that primarily focuses on diagnosing breast cancer [26]. General-purpose robots have been developed in the telemedicine[1] industry, including a robot called the Tactile Robotic Telemedicine system [26]. Some of the differences in the Tactile Robotic Telemedicine system in comparison to the robot described in the project include a robot that utilizes two identical robotic arms with seven degrees of freedom: one on the doctor's site and another on the patient's site. The two robotic arms can interact through LAN, Wi-Fi, or LTE networks. The doctor uses a joystick to interact with the robot on their side, which will then relay the movements to the robotic arm on the patient's side. Additionally, the robotic arm on the doctor's site can receive haptic force feedback to provide more information for the diagnosis. The doctor can view a 3D rendering of the robotic arm on the patient's site as it moves. Our solution aims to bring a more mobile approach allowing doctors to connect from anywhere, not just from a dedicated workstation.

[1] HyperLinks navigate to the Glossary in Appendix A

1.3 Proposed Solution

This project aims to create a website for doctors to remotely control the movements of a robotic arm placed next to a patient. This will allow doctors to perform virtual general physical health check-ups using common medical tools, such as a flashlight, a mallet, and a tongue depressor. This project provides a tool for remote and understaffed communities to receive general physical check-ups. The doctor will be able to control the robotic arm through a website interface and view the perspective of the arm through a Raspberry Pi camera. The system will send a Zoom meeting link to the patient in which they are expected to join the call from their own device and angle the device's camera in a third-person perspective. The doctor must log in to the system using their College of Physicians and Surgeons of Ontario (CPSO) number to access the website. The doctor will have two camera views: one from the mounted camera on the Raspberry Pi and another view of the Zoom meeting with the patient's device. The doctor will be able to take notes during the check-up session. At the end of the appointment, the system will send a text file of the notes to the doctor's email for them to review and send to the patient.

1.4 Project Overview

This project will be completed through a series of milestones and objectives. Throughout the project, our team will complete the objectives through these three main actions:

1. Obtaining a robotic arm that utilizes Raspberry Pi 4B and configuring it.
2. Creating a backend service to facilitate the communication from a web interface to the robotic arm. This service shall be hosted on an Amazon Web Services (AWS) cloud server.
3. Creating a frontend interface (website) for the doctor to control the robotic arm. This will be achieved using React.

1.5 Report Overview

This report provides an update on the final status of the project “Graphical Interface for a Robotic Arm”. This project aims to create a website for doctors to remotely control the movements of a robotic arm placed next to a patient. The report describes the project’s intended purpose, design, implementation, milestones achieved, and conclusions and reflections on the project and team experience. In section 2, we will describe the engineering project as a whole, information about our team, the project management strategy, and some project decisions and considerations for the design of our solution. In sections 3 and 4, we will describe our list of requirements and the resulting designs for this solution. This shall act as a template for the final project. Sections 5, 6, and 7 describe the design and progress of our frontend, backend, and robotic arm subsystems and how they will be connected. Section 8 breaks down our updated timeline of tasks to further display our progress and plan for the future. Finally, sections 9 and 10 describe our conclusions and reflections on this project and our experience building our solution.

2 The Engineering Project

2.1 Health and Safety

All members of the team are stringently following the manual provided by project administrators that outlines the minimum standards and practices for the safe and healthy operation of a laboratory [44]. The following of this manual ensures that all work meets the requirements of the Occupational Health and Safety Act of Ontario (OHSA).

All team members have reviewed Section 4: Responsibilities of Laboratory Workers (from the manual), and certify that we understand our responsibilities in the lab context. While the project room is used, all team members followed the General Health and Safety Principles and Basic Safety Procedures outlined in sections 5 and 6 of the manual.

Since a large portion of the project has been completed in a non-laboratory setting through the writing of software, in a variety of other environments (home, library, etc.), team members have not encountered such hazards. However, a common issue for software project teams is stress, which can have a huge impact on the completion of the project [8]. For all team members to help minimize stress and the chance of burnout, a set of stress health and safety protocols have been implemented. This protocol consists of the following: the team takes part in establishing clear and achievable goals, the team sets realistic timelines, the team communicates with each other regularly, team members take breaks when feeling stressed, and the team prioritizes the most important tasks.

2.2 Engineering Professionalism

During this project. all team members strived to demonstrate professionalism in accordance with what was learned during ECOR 4995 Professional Practice [9]. Some of the ways in which professionalism is demonstrated, as extracted from the Professional Engineering Practice Guideline, are listed below [34]:

1. As none of the project team members are professional engineers at this time, engineering titles (such as ‘Engineer’) outlined in section 7.3 Use of Titles of the referenced guideline, will not be used.
2. The Engineer’s Duty to Report design processes and procedures that are unsafe will be followed by team members in all aspects of the project design and implementation.
3. Interactions among team members and between team members and stakeholders will be managed professionally.
4. Reports will be written not only as per the project guidelines but also in accordance with section 10.5 Report Writing of the referenced guideline (except for the part about the use of the professional engineer’s seal because evidently we are not professional engineers yet).
5. Meeting notes and all project documents will be retained for referencing later on.
6. The Code of Ethics as outlined in section 77 of the Professional Engineers Act will be followed by all team members at all times [32].
7. All members of the team will strive to not engage in professional misconduct as per section 27 of the Professional Engineers Act [32].

2.3 Justification of Suitability for Degree Program

The project team is composed of the five following members. Note that course descriptions can be found under their respective titles in Appendix C.

Dhriti Aravind

Dhriti Aravind is a fourth-year student in computer systems engineering. In her program, she learned the fundamentals of programming (ECOR 1051, SYSC 2004, SYSC 3303, SYSC 4805), the applications of embedded programming, the basics of electronics, and how to analyze systems and simulate them. She has developed her interest in hardware and software through her internships at Ciena, Huawei, Microsoft, and Tesla where she worked in various areas of silicon development and tool support. Through her internship experiences, she familiarized herself with Python, C, Verilog, and System Verilog and participated in Agile scrums and a fast-paced environment. In addition, during her free time, continues to pursue her passion in robotics through various personal projects to automate mundane tasks; while undertaking these projects, she has refined her skills in 3D modeling and programming microcontrollers. In this project, Dhriti focused on programming of the robotic arm, designing the flashlight, and integrating the robotic arm with the website interface.

Elisha Catherasoo

Elisha Catherasoo is a fourth-year student in software engineering. She has gained experience throughout her education at university (ECOR 1051, SYSC 2004, SYSC 3110, SYSC 3310, and SYSC 4806), personal projects, and her internships at Ericsson and Amazon. At university, she learned the basics of programming, such as OOP, software architecture, and real-time systems. Learning real-time systems has taught her how to use a Raspberry Pi and basic coding projects. This experience will help when programming the Raspberry Pi to control the movement of the robotic arm and developing the presets for the robotic arm picking up specific objects (mallet, tongue depressor, and flashlight). When making the database for the authenticated doctors, she can use what she learned in SYSC 3110 and SYSC 4806 to be able to persist the authenticated doctor data. Her internship at Ericsson gave her experience with working collaboratively with her team, which includes people of varying strengths and weaknesses, and being able to know who is best to do what part to better the given project(s). Her experience with Amazon gave her exposure to AWS, which will help when deploying the robotic arm web application to be accessed anywhere. Elisha's experience in university and her internships has given her the valuable skills needed when working in a team, in the basics of programming, and in the technologies needed to be able to successfully complete the project.

Sarah Chow

Sarah Chow is a fourth-year student in software engineering. She has taken several programming courses throughout her undergraduate degree pertaining to software development including software development (SYSC 3110), software architecture (SYSC 4120), and object-oriented programming (SYSC 2004). Throughout her university degree, she has had several co-op placements, including Warner Bros. Discovery and the Royal Bank of Canada. At Warner Bros. Discovery, Sarah worked as an iOS and tvOS developer and implemented frontend features for the company. At the Royal Bank of Canada, Sarah was a frontend and backend developer over the eight-month co-op. She became knowledgeable in web development with Angular and Spring Boot and also participated in an Agile team with daily stand-ups, bi-weekly sprints, and monthly retrospectives using JIRA and Azure DevOps. In 2022, Sarah attended the hackathon, Hack the North, in which she created the frontend using React and open-source libraries to build a website. Sarah has an interest in developing her frontend skills.

Evan Smedley

Evan Smedley is a fourth-year student in software engineering. During his degree, he has gained knowledge of object-oriented software development (SYSC2004, SYSC3110, SYSC4806), software architecture (SYSC3120, SYSC4120), and the development of real-time systems (SYSC3310, SYSC3303). Over approximately the past two years, Evan worked full-time as a Software Engineer Intern at Motorola Solutions Inc. During his time at Motorola, Evan gained experience working with all levels of the software stack. In the infrastructure area of the software stack, Evan learned a lot about continuous integration, highly available systems, and cloud deployment of services with AWS and Microsoft Azure. This will be useful when deploying both the frontend and backend of our web application. In the backend of the stack, Evan gained experience with the use of Spring Boot to interact with message brokers, persist data and pipeline requests in a reactive manner. This will help with the writing of the backend of our web application. During the internship, Evan experienced development in an Agile Kanban and Agile Scrum environment. This experience will come in handy with working as a team to follow a software development cycle.

Jeremy Trendoff

Over the past eight years, Jeremy Trendoff has studied and practiced the fundamentals of software engineering and development. Over the last four years as a software engineering student at Carleton University, Jeremy has taken classes in embedded and real-time systems (SYSC3310, SYSC3303), object-oriented programming (SYSC3110, SYSC4806), software architecture and documentation (SYSC3120, SYSC4120). This knowledge will relate directly to both the development of the interface, as well as the software portion of the arm. On top of this, Jeremy enrolled in a year-long co-op term at SOTI Inc., a leader in Enterprise Mobility Management and the Internet of Things. There, Jeremy worked as a fullstack developer, helping to solve frontend, backend, middleware, architecture, and research problems. At SOTI, Jeremy also learned the basics of professional practice for software development. Jeremy gained experience with tools like Git, JIRA, and the Agile development lifecycle, but, most importantly, learned how to develop software as a team. This knowledge is sure to prove essential during this, and any, professional project.

The Team

As outlined above, each member possesses valuable skills and experience. Collectively, the team's overall skills encompass the requirements to develop this robotic arm project. To tackle the software portion, our four software engineering majors have experience in user interface design and programming, software architecture principles, and object-oriented programming, and have studied embedded and real-time systems programming to competently handle programming on the hardware side. Our computer systems engineering major has not only learned the fundamentals of programming and software design but has built extensive knowledge of the basics of electronics and real-time systems. Combining our strengths, our team has the knowledge and passion to complete this project.

2.4 Project Management

2.4.1 Team Structure

Table 1: Project group sub-teams and their contributors.

Team	**Primary Contributor (Specialist)**	**Second Contributor (Floater)**
Robotic Arm	Dhriti	Elisha
Frontend	Sarah	Jeremy
Backend	Evan	Jeremy, Elisha

Our team decided to structure ourselves using a non-hierarchical organization. This means we operate as one cohesive, democratic group. Instead of having team leaders, we are all equally responsible for driving the project forward. The team operates through three main subteams: the frontend team, backend team, and the robotic arm team. These teams are cross-functional, but the main role divisions have been outlined. While everyone on each team is expected to contribute to the architecture and development, we have structured ourselves into specialist and floater roles. Specialists (primary contributors) mainly focus on their team's work throughout the project. Floaters (secondary contributors) split their time according to the current demands of the project. On the frontend team, Sarah Chow is our specialist and Jeremy Trendoff acts as a frontend floater. On the backend team, Evan Smedley is our specialist. Jeremy Trendoff and Elisha Catherasoo are backend floaters. Finally, on the robotic arm team, Dhriti Aravind is our

specialist and Elisha Catherasoo acts as the arm floater. This structure may be subject to change but at the moment, it suits our team's strengths.

2.4.2 Software Development Lifecycle Methodology

In keeping with some team members' experience during co-op terms, we will be using Agile methodologies for this project. We will be following the 12 Agile principles outlined in the Agile manifesto [1]. The Agile methodology we felt would best suit our needs is Kanban [7]. Developed by Toyota in the 1940s to optimize their engineering process, Kanban has become one of the most popular frameworks used to implement both Agile and DevOps software development. Kanban uses a highly visual Kanban board which provides full transparency of work. On the board, Kanban cards will move from the basic states 'To Do', 'In Progress', and 'Done'. Once a work item has been completed, the next work item in the backlog will take that space on the board. So, as long as the backlog is up to date, there is no need for fixed-length sprints. This will also suit our non-hierarchical structure as all team members are treated with equal privilege with this scheme. Kanban is also great for teams of all sizes. While other Agile methodologies have more structure, Kanban allows for more flexibility across the board. This will allow our team to quickly solve problems and determine the best plans of action when we encounter errors.

2.4.3 Changing from Azure DevOps to GitHub Issues

The team has now decided to move away from using Azure DevOps, and to start using GitHub Issues as our project is using GitHub for the version control. GitHub Issues enables users to track and manage tasks, bug fixes, and enhancements to the project. For every pull request, an Issue explaining what the feature being added should do, what bugs are being fixed and what is being tested, is attached. This allows for other team members checking the code to see exactly what the changes are supposed to do and check if the tests reach 100% code coverage. This allows for the pull requests to be easily and efficiently checked as there will not be any confusion on the need for the pull request. The team was exposed to using GitHub Issues in the class SYSC 4806, where the team started using Issues in their labs, and it was found that it worked seamlessly with their version control, and that it was convenient and efficient to have everything in one place.

2.4.4 Version Control

Git and GitHub will be used for version control for this project. GitHub is an online user interface to the tool, Git. This will allow proper reviewing of code using pull requests and be a useful tool for collaboration. There will be one repository with a folder for each component of the project.

It was decided that the team would use GitHub because:

- GitHub is the industry standard, as demonstrated by the large number of reputable software companies that use GitHub [18]
- Although there are alternatives to GitHub, such as Azure Git, it is less widely used and therefore has less readily available documentation to support development

2.4.5 Mitigation of Project Risks

The project had several risks associated with it. These risks are outlined in Table 2 below.

Table 2: Project risks and mitigation.

Risk	**Mitigation Technique**
Dependency on a third party application, Zoom, for patient-doctor conferencing. Doctors' appointments involve the transferring of personally identifiable information, which is confidential.	To mitigate this risk a review of Zoom's privacy standards must be performed to determine if it is an acceptable solution for this system. If the conclusion is that Zoom is not secure enough, there are many other options for view conferencing software that is created specifically for this sort of application with security as a high priority.
Robotic Arm Malfunctions/or Patient feels uncomfortable	Patient will press hard stop button
Robotic arm is unable to pick up tool	Patient can put the tool in the claw of the robotic arm
Milestones are missed and the project is not able to be finished	Project features and scope can be reduced. The bot will then be focused on the mallet reflex tests. The mallet will then be the only available tool in this interaction of the project.

High latency when connecting to the Raspberry Pi leading to a greater than 3 second delay when maneuvering	Attempt to speed up infrastructure by connecting to the Raspberry Pi through a different method.
The cost of cloud services has been calculated, but sometimes there are some hidden costs for network traffic or storage that are not accounted for.	The team will use Google Cloud free tier in hopes that the $416 of free credit given is enough to cover unexpected costs. The team will monitor the cost of the application and gather metrics in order to be able to predict future costs.

2.5 Individual Contributions

This section carefully itemizes the individual contributions of each team member. Project contributions identify which components of work were done by each individual and report contributions list the author of each major section of this report.

2.5.1 Project Contributions

The following table shows the project contributions of each team member.

Table 3: Team member contributions to the project.

Member	**Project Contributions**
Jeremy Trendoff	- Aided in development of backend API. - Aided in design and problem solving related to rebounding force from mallet impact. - Worked on establishing a connection from the frontend to the backend. - Implemented security procedures for authentication and authorization. - Developed object distance detection for the robotic arm camera. - Implemented Zoom into the frontend site. - Research and set up TLS and HTTPS security (unable to implement due to budget constraints).

Elisha Catherasoo	- Found data for force required for the patella reaction test. - Aided in design and problem solving related to rebounding force from mallet impact. - Connected the picamera from the robot to the web app - Added live mouth detection using OpenCV
Evan Smedley	- Researched and selected backend technologies. - Defined the Java Spring Boot backend API - Wrote the HTTP client and HTTP server, and defined the API for robotic arm endpoints - Implemented robotic arm registration mechanism - Researched and created a proof of concept for app deployment in AWS (abandoned due to cost that did not fit with project budget). - Implemented continuous integration process with GitHub Actions - Used Google Cloud Run and Google Artifact Registry to deploy containerized services (web server, backend server) in the cloud
Sarah Chow	- Created the frontend server using React. - Created the routing to the landing page, login page, and control page. - Created and implemented all frontend components and layout for the landing page, login page, and control page.
Dhriti Aravind	- Assembled the robot. - Created a script for Inverse Kinematics and Forward Kinematics using python - Wrote and tested directional functions in Python for robot movement. - Connected the robot to the internet. - Programmed the robot on the same network over IP. - Designed and attached a flashlight to the robotic arm

2.5.2 Report Contributions

The project consisted of three main reports, the proposal, progress report and final report. The contributions of each team member to each of the reports is outlined in the following sections.

2.5.2.1 Proposal Contributions

Table 4 shows the project proposal contributions of each team member.

Table 4: Team member contributions to the project proposal.

Member	**Proposal Contributions**
Jeremy Trendoff	1.1, 1.2, 1.3, 2.3, 2.4, 2.5, 3, 4.1.2, 4.1.3, 4.3, Glossary, Appendix A, Appendix B
Elisha Catherasoo	1.1, 1.2, 2.3, 2.4, 2.5, 3, 4.1.2, 5, References, Glossary, Appendix A
Evan Smedley	2.1, 2.2, 2.3, 2.5, 4.1.2, 4.1.3, 4.1.4, 4.1.5, 4.2, References
Sarah Chow	1.1, 1.2, 1.3, 1.4, 2.3, 2.4.1, 2.5, 3, 4.1.1, 4.3, Appendix A
Dhriti Aravind	1.1, 1.2, 1.3, 2.3, 2.5, 3, 4.1.6, 4.3, 5, 6, 7, Appendix A

2.5.2.2 Progress Report Contributions

Table 5 shows the project progress report chapter contributions of each team member.

Table 5: Team member contributions to the project progress report.

Member	**Progress Report Contributions**
Jeremy Trendoff	Abstract, 1, 2, 3, 4, 5, 11, 12, References, Appendix A
Elisha Catherasoo	1, 2, 3, 8, 11, 12, Appendix A
Evan Smedley	2, 4, 5, 8, 9, 10, 11, 12
Sarah Chow	1.1 - 1.4, 5, 6, 7, 8, 11, 12, References
Dhriti Aravind	1, 2, 6, 7, 10, 11, 12

2.5.2.3 Final Report Contributions

Table 6 shows the project progress report chapter contributions of each team member.

Table 6: Team member contributions to the project final report.

Member	**Final Report Contributions**
Jeremy Trendoff	Abstract, 1, 2, 3, 4, 5, 9, 10, References, Appendix A
Elisha Catherasoo	1, 2, 3, 7, 8, 9, 10, Appendix A
Evan Smedley	2, 4, 5, 8, 9, 10, References
Sarah Chow	1.1 - 1.4, 5, 6, 7, 8 9, 10, References
Dhriti Aravind	1, 2, 6, 7, 9, 10

3 Requirements

A list of requirements was made before starting the implementation and development of the project. Some aspects that were looked at when creating the requirements are: how the robot needs to move, how doctors and patients interact in the web app, what is shown in the web app, and the speed between the web app controls and the robotic movements.

3.1 Functional Requirements

The following are the functional requirements for the project:

Robot Movement and Connection

- The robot shall be able to move in 3 planes of motion (x, y, and z).
- The robot shall be connected to Wi-Fi and receive an IP address.
- The robot shall interface with the web interface using HTTP protocols.
- The robot shall be able to pick up items less than 500 grams.
- The robot shall be able to pick up items less than 20 cm in the smallest dimension.

Doctor-Patient Interaction

- The doctor and patient shall communicate over a video call throughout the session.
- The robot shall be ready for motion when the patient's physical start button is pressed.
- The robot shall stop when the patient's physical stop button is pressed.
- The robot shall stop when the doctor's virtual stop button is pressed.
- The doctor shall start the robot once the patient's physical start button has been pressed.

Web Interface Interaction

- The doctor shall log in through the login portal using their CPSO number and their password.
- The login portal shall redirect the doctor to the robot interface web page.
- The doctor shall be able to see the robot's camera view on the web interface.
- The doctor shall be able to control the robot through a set of directional and depth control buttons on the web interface.
- The doctor shall be able to write and download notes from the notes section of the web interface.
- The doctor shall be emailed the notes from the notes section of the web interface after the patient session has concluded.

- The doctor shall be able to select between different examination tools through preprogrammed action buttons on the web interface.
- The doctor shall be able to control the flashlight using the flashlight preprogrammed action buttons on the web interface.

3.2 Non-Functional Requirements

The following are the non-functional requirements for the project:

Usability

- Zoom shall be used to facilitate doctor-patient video calls.
- The website should be compatible with major browsers, ie. Chrome, Firefox, and Safari.

Security

- The login portal shall be supported using CPSO's authentication service to validate doctor credentials.
- The login portal shall authenticate the doctor based on CPSO number and password.

Compliance

- The system shall handle and store any personal information of its end users in accordance with the Personal Information Protection and Electronic Documents Act 2000.

Performance

- The latency from the buttons on the doctor interface to the robotic arm movements shall be under 20 milliseconds.
- The delay on the video streamed from the camera on the robotic arm shall be under 20 milliseconds.

Maintainability

- Code should be well documented and be easily understood by new developers.
- Project source code should be managed in a version control system (ie. GitHub).

Robustness

- In the event of a system or session failure, the system shall gracefully terminate all sessions.
- In the event of a camera failure, the system shall deactivate the robotic arm immediately.

4 Project Design

From the identified requirements, the team decided that our project solution would revolve around three main subsystems, a frontend website responsible for providing a user interface for system interaction, a backend server responsible for security and API implementation, and a robotic arm responsible for facilitating the checkup.

4.1 Identifying Use Cases

The team decided that the first step in designing this project solution was to identify the use cases of the doctor and patient to evaluate the necessary actions our solution would need to satisfy. This also helped to identify the components that need to be developed inside each of our three sub-systems: the frontend, backend, and robotic arm. This diagram also depicts the desired workflow between the doctor and the patient. The doctor first logs into the website, which is validated by the backend security. Then, the doctor starts a Zoom call with the patient. Then, once the patient has joined the call and has gone through any setup processes, the doctor can freely execute instructions on the robotic arm. On the other side, the patient joins the Zoom call and plugs in the robotic arm to start the session. This provides patient consent for the checkup. During the checkup, the patient has access to an emergency stop button in the event an unwanted action were to occur.

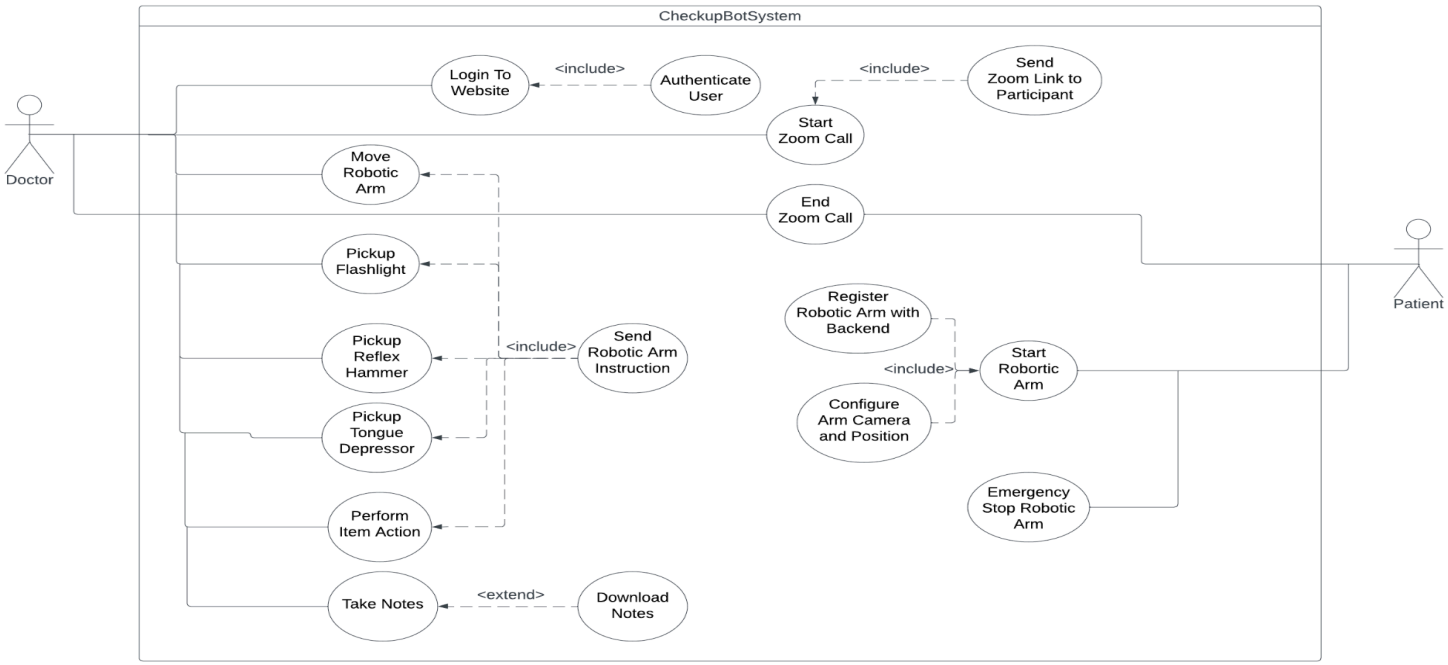


Figure 1: Use Case Diagram for the Doctor and Patient.

4.2 Project Deployment

The next step in the design process was to develop a conceptual understanding of how our system would be deployed in the real world. This diagram provided insights into the necessary deployment services required to deploy in the cloud. The team decided that a cloud service would support our design optimally as it allows for the ability to easily scale, vertically and horizontally, and would reduce the latency of requests.

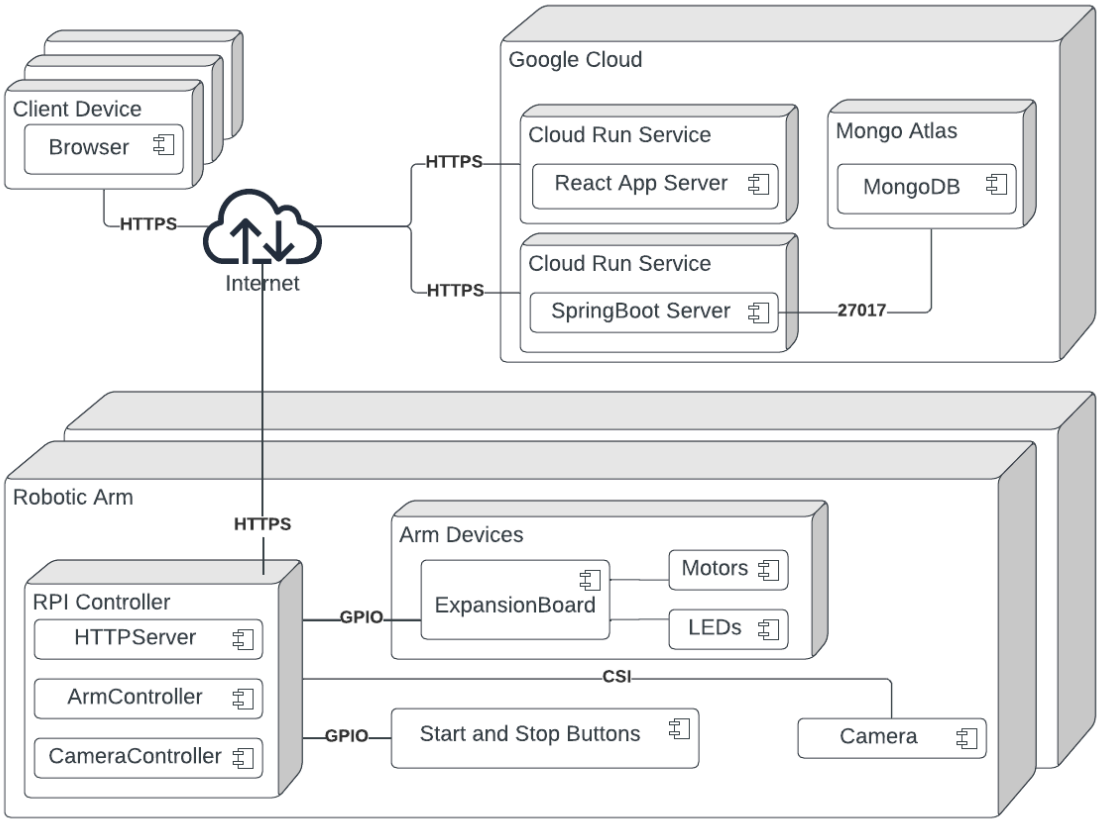


Figure 2: *A deployment diagram visualizing the connection of our different components in the real world.*

5 Frontend

5.1 Trade-Off Analysis

React was chosen as the frontend framework because it offers numerous plugins and open-source libraries and utilizes virtual document object model (DOM) objects. A virtual DOM provides the ability to reload a single object on a page instead of reloading all of the objects on the page when there is a visual change [35]. This feature requires less performance from the cloud server and will decrease the overall costs for the project [35].

The tools utilized for this project include react-router-dom and Material UI. Our website requires communication with the backend server to receive information to render a page. The react-router-dom [37] package allows for client-side routing which implies that the frontend does not need to contact the backend to route to a new page. This provides for faster web page loading time. Material UI [25] is an open-source React library that provides ready-for-use components for a website. All of the components from this library are customizable to suit specific user needs. In our project, Material UI [25] was used for most of the components on the web page, including the text inputs for the login functionality, the drop-down menu to select an endpoint, and the text box to take notes.

5.2 Developments

It was noted the size ratio of the Figma designs was incorrect and this impacted the exported images from the designs. This introduces the challenge where the ratio of the images and placements of the components on screen need to be repositioned and rescaled. This issue was fixed and the Figma designs have the proper aspect ratio.

The navigation bar in the Figma designs was seen overlapping the background of the landing page. During the development of the landing page, the navigation bar appeared to be pushing down the background image. The implementation of the navigation bar was done by creating a header component and then placing it on the home page. The result of this implementation is shown in Figure 3.

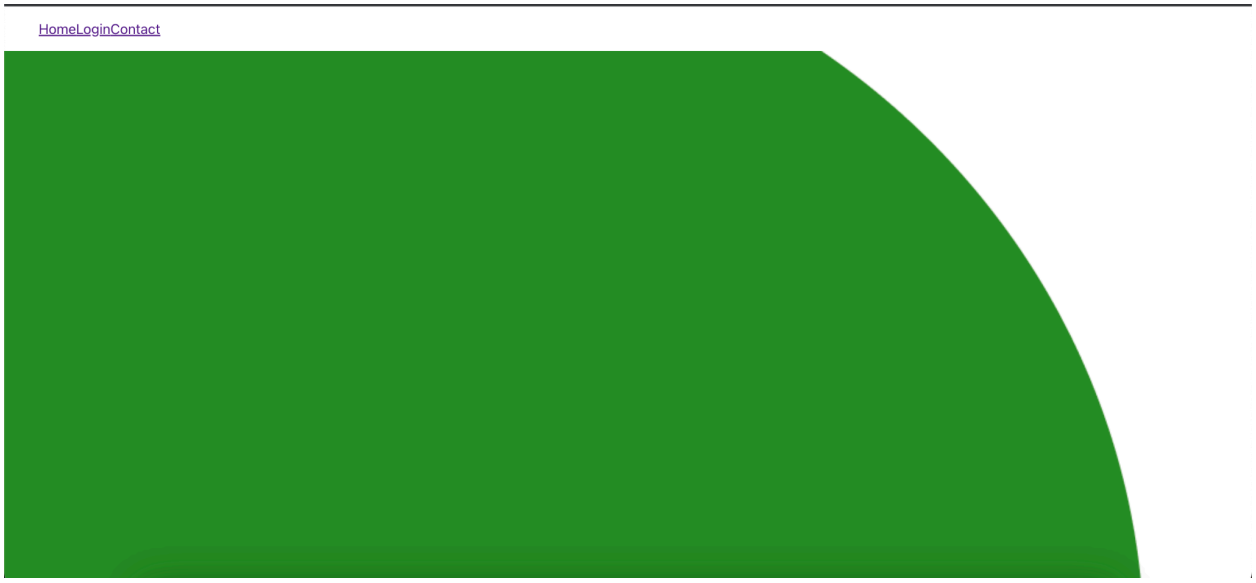


Figure 3: Navigation bar pushed down the background image

Upon further research, there was an alternate method of creating a navigation bar by utilizing a component called App Bar in the Material UI library. Once implemented, the App Bar provided a clearer user experience and allowed for an added profile capability as seen in the top right of Figure 4. The default color of the component is blue, but upon further investigation, it was not possible to change the background color of the header. It was also noted that this color scheme was difficult to read and this impacted accessibility for users.

Figure 4: App bar implemented from material-ui library

The current implementation of the landing page disregards third party components and places the hyperlinks directly on the home page. This method allows the links to be overlapped on top of the home page. As shown in Figure 5, the header text is black, as opposed to the white color in the Figma design. This is due to the varying background colors amongst the pages and black is a visible color on all the pages. The initial Figma designs are visible in Appendix B.

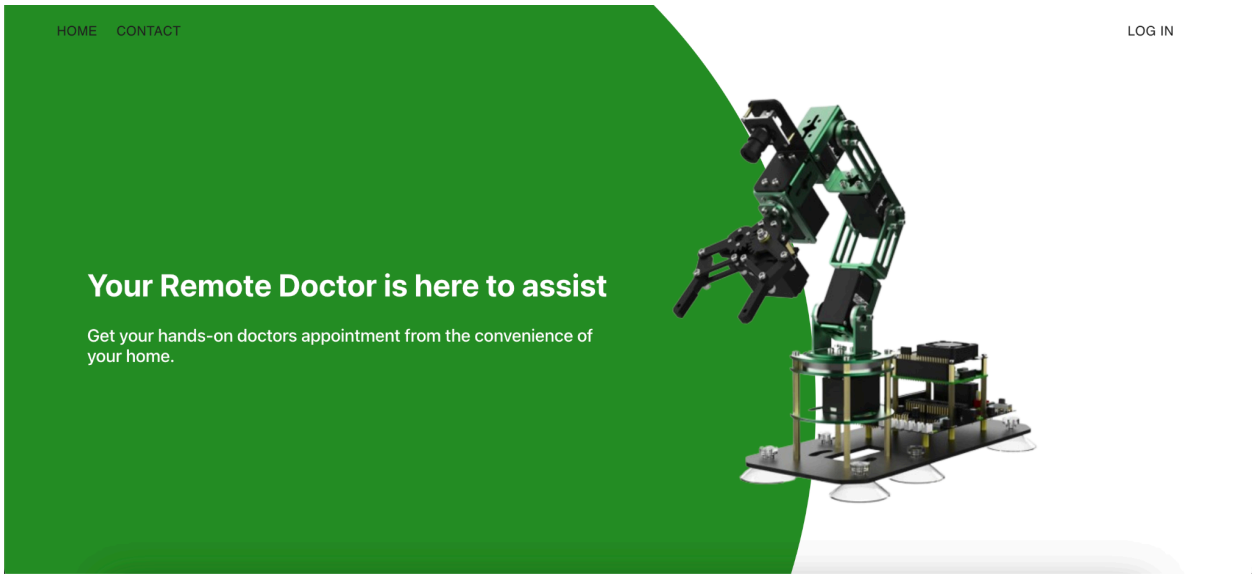


Figure 5: Landing page of the website.

The login page was created with shapes and a form field, as seen in Figure 6. The doctor is prompted for their CPSO number and corresponding password, which they can choose to display or hide by clicking on the eye symbol in the password input.

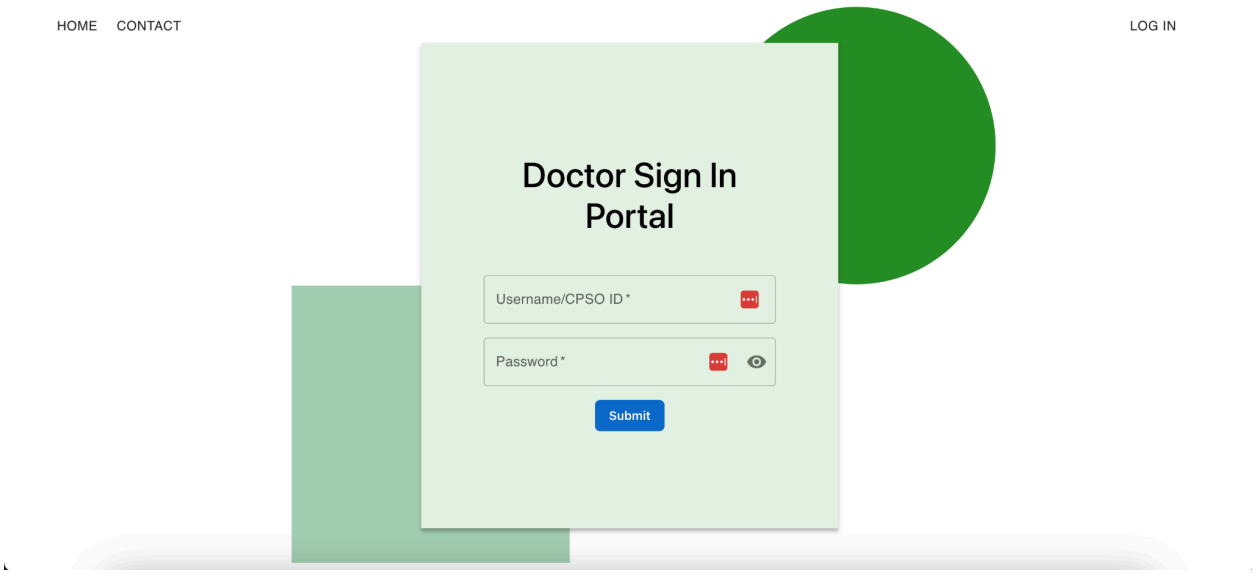


Figure 6: Login page of the website.

An alert banner appears for three seconds with a message conveying an incorrect username or password for an unsuccessful login attempt, as seen in Figure 7.

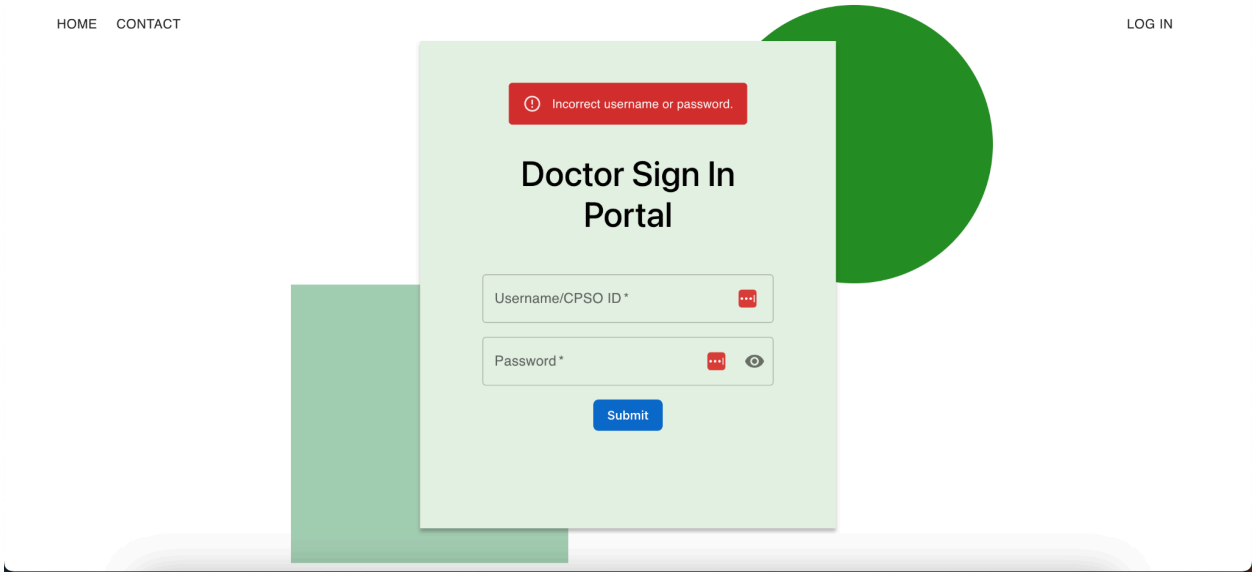


Figure 7: Unsuccessful login attempt.

Once the doctor has successfully logged in, they are redirected to the main control page. The doctor selects an endpoint from the dropdown list and the video feed begins when the connection has been established. The first design of the control page is seen in Figure 8. The doctor controls the desired motor

by selecting one of the buttons labeled one through six, as the robotic arm has six motors. Each motor has a direction (left/right, up/down, clockwise/counter-clockwise, open/close) displayed on the slider. The doctor controls the behavior of the motor by adjusting the slider. Figure 8 displays the first version of the control page, which has the control panel on the right hand side and the endpoint drop-down directly beneath the camera feed. The interface provides a note taking functionality in the bottom right hand corner. The doctor takes notes throughout the checkup and then downloads the notes file to their computer.

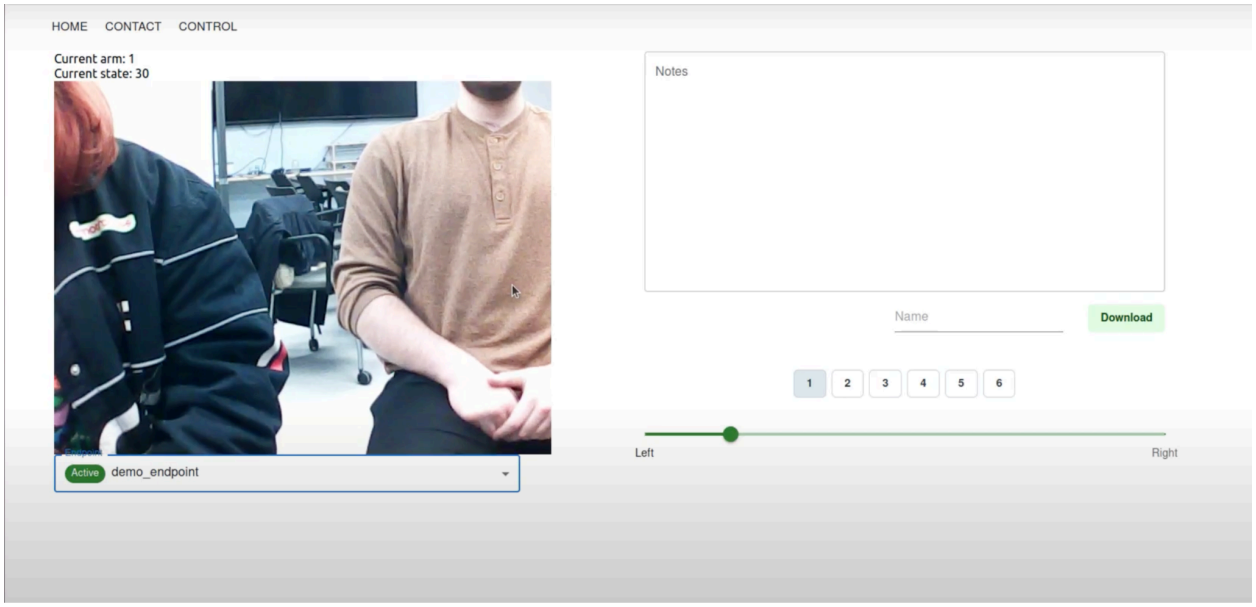


Figure 8: First version of the control page.

The control page was redesigned to include a secondary camera feed from the Zoom meeting. The new design is seen in Figure 9. In this design, the control panel is moved directly beneath the arm camera feed and the notes section is shortened and moved beneath the secondary camera feed. Additionally, the endpoint drop-down is shortened and an endpoint latency is implemented.

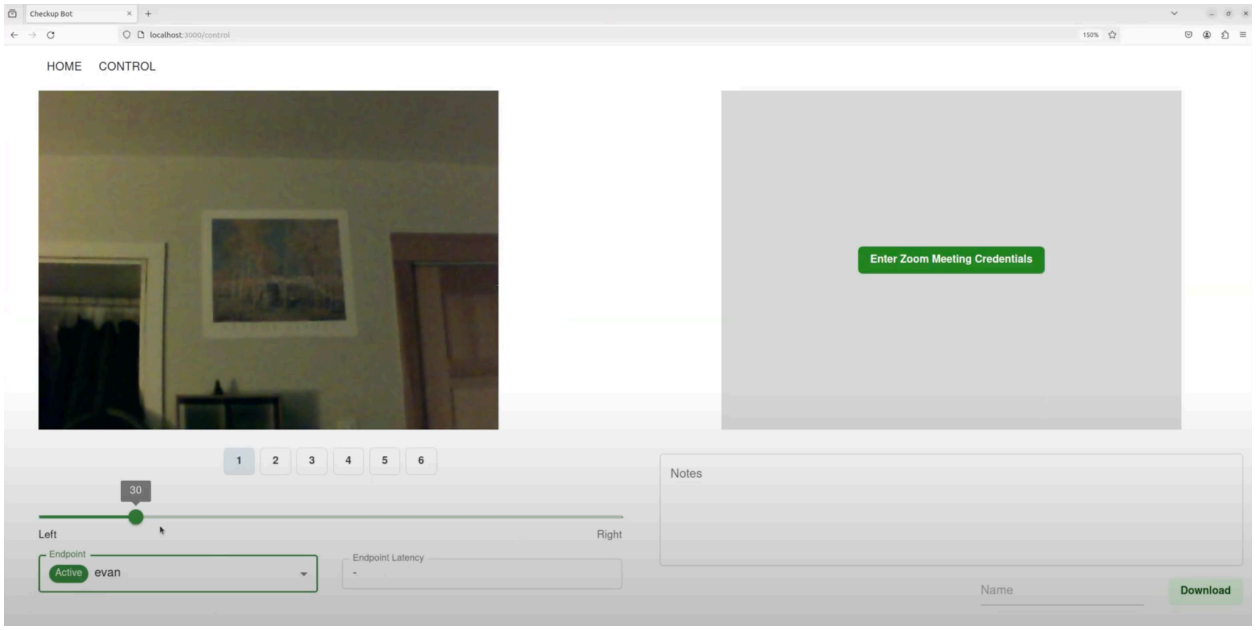


Figure 9: Current implementation of the control page.

5.3 Testing

As the frontend was developed and was changing, most of the testing was done manually by clicking buttons and viewing the web page console. If there were errors, the web page console would output the error code and provide more details about the issue. Unit testing files were created to test individual functions and in the future, it is suggested to add integration testing for increased robustness.

5.4 Accomplishments

The frontend was built using React and TypeScript. The website consists of three pages: the landing page, the login page, and the control page. Users cannot access the control page without providing correct login credentials to the website. The control page consists of two video feeds: one from the Raspberry Pi and another from a Zoom meeting. It contains six buttons representing each motor in the arm and a slider to control the movement of the chosen arm. The doctor can take notes throughout the meeting and can download the file. Additionally, the endpoint latency is displayed on the control page.

5.5 Resolved Issues

The initial Figma designs in Appendix B presented six buttons to move the robotic arm on the control page (left, right, up, down, depth in, depth out). This design limited the total movement of the arm since the arm is composed of six individual motors. This was resolved by implementing the six buttons and the slider for each button. The Figma designs for the new control panel were not created due to time constraints and the website was redesigned as it was being developed.

The adjustment for varying screen sizes posed another challenge as larger screens presented a lot of white space along the bottom of the main control page. The objects on the pages are placed relative to each other, not relative to the screen. This was corrected by placing the objects relative to the screen size using the root element unit [38]. The rem unit is relative to the root element of the page, which is the HTML root DOM element.

6 Backend

The backend assembles many technologies in order to accomplish its primary goals: user login, endpoint management, and cloud deployment. This section provides some background knowledge on each of the involved concepts and technologies, including why they were chosen. We then identify how the use of each technology developed over the course of the project. The final accomplishments of the project and the primary challenges that we overcame are then outlined in the last two subsections.

6.1 Trade-Off Analysis

The background section provides the basic information needed to understand the role of each technology we used in our system and why it was chosen over its counterparts.

6.1.1 Spring Boot

Spring Boot is a framework for Java applications. It was used to implement the backend API and database connection for this project. Spring Boot uses the principle of convention over configuration to make the process of API creation simpler and has libraries to help with the security of an application and with connecting to a database.

Maven is an important tool that is used to import all the dependencies required to use Spring Boot effectively. Maven has a lifecycle that consists of many phases, such as *test* and *package* [5]. These phases produce an executable JAR file that runs our application. The JAR file can be used in a deployment to run the application server that hosts the API.

Spring Boot was selected to be used for the backend of this project because:

- Four team members (Sarah, Elisha, Jeremy, and Evan) have knowledge of Spring Boot from SYSC4806.
- Spring Boot provides many useful dependencies for database interaction, simple creation of a REST API, and reactive processing (more efficient than a regular event-driven model).
- NodeJS, Django, Ruby on Rails, and Go are all alternatives to Spring Boot that provide similar functionality, however, team members do not have any experience with these alternatives at this point so Spring Boot is the clear choice

Spring Boot offers two different architectures to handle requests when developing an API, namely Spring Web MVC and Spring WebFlux [39]. The main difference is that Spring Web MVC uses a single thread for each inbound request (blocking for a request can impact other requests) while Spring WebFlux is reactive (leverages project reactor's reactive stack) and non-blocking, so no thread is blocked while waiting for an action [40]. Of the two, Spring WebFlux is more modern and more suited to handle multiple requests. Spring Boot provides the advantage of offering Web MVC or WebFlux to handle large request loads.

6.1.2 MongoDB and MongoDB Atlas

When selecting a database, the two primary options are SQL and NoSQL. SQL databases are better suited for fast access to large amounts of data with complex queries while NoSQL databases are easier to use for simple applications with smaller amounts of data and simpler queries.

The team decided to use a NoSQL database for the following reasons:

- Interaction with a NoSQL database is simpler than interaction with a relational database.
- The database is only needed to store user credentials and endpoint information. Neither of these request types need to be particularly performant and they each involve a small amount of data. So in comparing a NoSQL database to a relational database, the simplicity of NoSQL is found to outweigh the reduced performance of a NoSQL database.
- NoSQL databases tend to store data as JSON documents, which is a simpler and more flexible format than SQL databases.

As for NoSQL databases, there are several options such as Couchbase, Amazon DynamoDB, and MongoDB. Of these, MongoDB was selected as the NoSQL database to be used for the following reasons:

- MongoDB has better integration with Spring Web MVC and Spring WebFlux [41] than Couchbase.
- Team members Evan and Sarah have experience using MongoDB.
- MongoDB provides an easy and free option called MongoDB Atlas, which is a cloud-hosted (Google Cloud) database that is very easy to navigate using a well-made web interface.
- Amazon DynamoDB is not cloud-agnostic (only offered by Amazon) and we ended up not using Amazon as our cloud provider, so this was not a viable option after all.

As a result, MongoDB was selected as the NoSQL database to be used.

6.1.3 Cloud Deployment

The cloud deployment process was centered around the deployment of containers with the components of our app. This process involves the continuous integration pipelining tool, Docker, for containerization, container storage, and integration with a cloud provider.

6.1.3.1 GitHub Actions and Continuous Integration

Continuous integration is defined as integrating code from multiple contributors to a single project [6]. In our project, there are five contributors to a single repository. It is important to have a continuous integration pipeline to ensure a working and symbiotic code collaboration process. Our team decided that GitHub Actions [19] will be used for continuous integration. GitHub Actions is a tool to automate building, testing, and deploying a GitHub repository [21]. GitHub Actions runs infrastructure as code for free for all public projects, which was leveraged to avoid unnecessary costs [22]. GitHub Actions allowed automatic packaging, building of a Docker image [14], and deployment of the backend and frontend.

It was decided that the team would use GitHub Actions because:

- GitHub Actions integrates very well with GitHub.
- GitHub Actions provides useful templates for the deployment of an application to AWS [20].
- GitHub Actions is very easy to use, which allows a continuous integration system to be established quickly and requires little configuration.
- GitHub Actions is free.

6.1.3.2 Docker and Containerization

Containerization refers to packaging a software artifact with its dependencies and a lightweight operating system, to create a container image. The point of a container image is to package an artifact with its environment to limit the number of things that could go wrong once the app is deployed. Docker is a well known containerization tool that has been used for this project.

It was decided the team would use Docker for the backend service because:

- It simplifies the deployment of the service to the cloud.
- It ensures that the environment in which our backend service is running is uniform, which reduces the amount of troubleshooting needed since each member may have a different development environment.

A backend artifact is placed inside a Docker image [14] using Docker Build, a tool for assembling Docker images, using a ‘Dockerfile’ containing configuration for the building of the container [12]. Docker Compose, a tool for defining and running multi-container Docker applications, is used to streamline and automate the process in the local development environment but is not necessary in the cloud environment [13]. With Docker Compose, a YAML file configures the application's services and the networks [13]. Each service in the project is packaged with its environment into an image based on configuration from its ‘Dockerfile’, and is then ready to be deployed [13].

Docker Hub provides many base images containing an initial environment for the building of containers. These generally consist of a lightweight Linux distribution [2][11]. Some base images have specific tools (Python, Java, C++, Node, etc.) pre-installed to simplify the installation.

6.1.3.3 Cloud Providers

Using cloud compute provided by a cloud provider simplifies the deployment of an application for a reasonable cost. Deploying performant and highly available apps was very complex and offloading that responsibility to a cloud provider made the process quicker. Using a cloud provider was crucial in our situation as it has a global presence and allowed for deployment in different regions to reduce latency.

Two of the most common cloud providers are Amazon Web Services and Google Cloud [42][46]. Both provided similar types of services. The most important service that was needed for this project is cloud compute. Since artifacts were containerized, the simplest method to deploy was utilizing a managed container service such as AWS Fargate or Google Cloud Run.

Containerized deployment was chosen for this project’s services because:

- It eliminated the complexity of managing the environment in which services were deployed, if a container works on a developer machine then it also worked when deployed.
- Installation of dependencies was simplified as they reside within the container.
- Managed container services provided by Google Cloud Run and Amazon Web Services were easy to use and provided log monitoring features.

AWS [4] and Google Cloud were shortlisted as cloud providers due to their extensive free tiers, which would reduce cloud costs while developing a proof of concept.

6.1.3.4 Nginx as a Reverse Proxy

A reverse proxy server is an intermediary server containing the configuration to forward requests from the clients to the desired destination [28]. A reverse proxy acts as a load balancer for systems with more than one instance of a server, is used to perform SSL encryption to reduce the load of the servers, and provides additional security [28]. Cloud providers such as GCloud and Amazon Web Services provide a reverse proxy functionality hidden behind a web interface. A reverse proxy was initially required in the first iteration of cloud deployment, as redirection of requests was needed. However, in the second iteration of cloud deployment there was only one container per service rather than a group, so there was no need to route requests from the single entry point to each service.

6.1.4 Spring Security

As mentioned in the non-functional requirements, security was an important design consideration for this project. The system must ensure that only authenticated users access the robotic arm. If an attacker gains access, a patient's health may be at risk. To implement authorization and authentication Spring Security was used.

Spring Security was selected for this project because:

- Spring Security is a powerful and customizable authentication framework.
- Spring Security is the standard for Spring-Based applications.

The team used JSON Web Tokens signed with a shared secret for the login security method. JSON Web Tokens are compact, self-contained messages that allow for secure data transfer and authentication. As JWTs are signed, it is easy to verify where the token originated. Therefore, these tokens can be trusted.

In addition, JWTs were selected for login because:

- JWTs promote enhanced scalability.
- JWT authentication is stateless.

6.1.5 Flask

Flask is a web framework, similar to Spring Boot, but for Python instead of Java. Flask is commonly referred to as a microframework, meaning it is simple, extensible, and follows convention over configuration [36]. Flask is installed as a Python module and can be imported and used to define an API

[36]. Flask uses the Web Server Gateway Interface (WGSI) which specifies a common interface between clients and servers in web applications, to provide many of its conventions[36].

Flask was used to define the simple robotic arm endpoint API. It was selected because:

- Writing a Python web server from scratch was not an efficient approach and a framework was needed.
- Flask is the most commonly used web framework for Python [42].
- Control of the robotic arm was already written in Python, and for easy integration, a Python framework was selected.
- Other Python web framework options with more complexity such as Django were considered but Flask did not have as steep of a learning curve and, although simple, provided enough functionality.
- FastAPI is a new and popular Python web framework that is more high-performance than Flask but not documented as well so Flask was the preferred choice [42].

6.1.6 Zoom Meeting SDK

The Zoom Meeting SDK [47] is a resource that allows developers to integrate Zoom's video conferencing into their applications. Zoom has proven to be a popular and reliable video conferencing platform and was chosen to be the primary form of communication between the doctor and the patient. This SDK provided the web page with the ability to access Zoom video conferencing without needing to leave the website.

6.2 Developments

Progress on the backend API, use of MongoDB, cloud deployment, security, and the robotic arm endpoint are detailed in this section.

6.2.1 Spring Boot Backend API

Developments with the Spring Boot application are related to passing the robotic arm control messages from the frontend to the endpoint, database connectivity, and user login. The API for the Spring Boot application was documented using OpenAPI and Swagger UI.

Initially, control messages from the frontend were sent to the Spring Boot backend and forwarded to the endpoint. This provided increased security as the endpoint only communicated with the Spring Boot backend and not the client or robotic arm. The latency added by sending the messages through the backend proved to be too high and as a result, the requests were no longer routed through the backend. Requests were sent directly from the frontend to the selected endpoint. This did not require any modifications in the Spring Boot backend but rendered a section of the API unused. This change made the system suddenly less secure. Although not included, HTTPS must be implemented between the frontend and endpoints to secure the system.

The two possible database connectivity options corresponded to the two types of Spring Boot web applications: Spring Web MVC and Spring Boot WebFlux. The Java Persistence API (JPA) dependency was used with Spring Web MVC and Spring Data Reactive Repositories was used with Spring WebFlux. Both database connectivity options had integration with the selected MongoDB database so both were viable options. Initially, Spring Boot WebFlux was decided as it would be faster than Spring Web MVC. When the robotic arm control messages were no longer routed through the Spring Boot backend, its performance requirements diminished greatly and the simple Spring Web MVC was sufficient.

In regards to the user login, Spring Security was the clear choice as it is well integrated with Spring Boot and provides a stateless method of login management. There are two versions of Spring Security, one for WebFlux and one for Web MVC. The version for Web MVC was chosen as it corresponded with the rest of our backend implementation.

In terms of the backend API, several request types are defined based on the responsibilities of the backend. Figure 10 shows the currently defined API. There are 3 categories of requests, split by the controller as can be seen in Figure 10: the authentication related requests, robotic arm registration requests, and robotic arm instruction requests, which are shown in more detail in tables 7, 8, and 9 respectively. Note that the arm movement-controller HTTP endpoints are implemented but they are no longer used. They serve as a fallback mechanism that is used by the frontend if it cannot successfully send requests directly to the robotic arm.

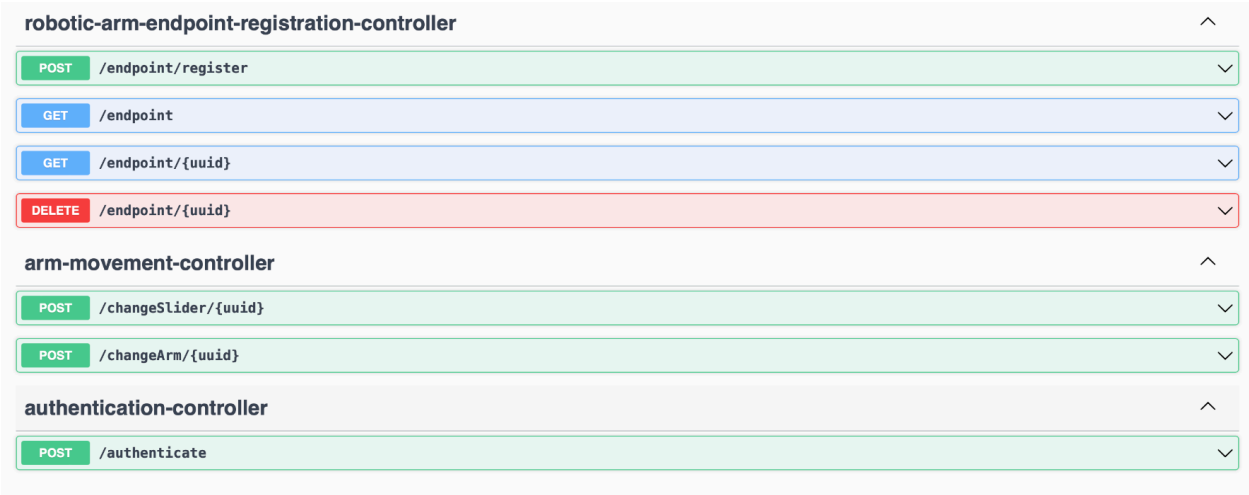


Figure 10: Spring Boot backend API.

Table 7: Authentication-related HTTP requests.

Request Name	Type	Description	Backend logic that occurs as a result
login	POST	"/login" endpoint accepts a post request with user credentials.	Using Spring Security, hash the password and compare it to the password stored in the database. Upon successful authentication, return a session ID to the user so they can remain authenticated for a limited period.
logout	POST	"/logout" endpoint accepts a post request with user credentials.	Terminate the user session and revoke access to directional requests.

Table 8: Robotic arm registration-related HTTP requests.

Request Name	Type	Description	Backend logic that occurs as a result
selectRoboticArm	POST	"/selectRoboticArm" endpoint accepts a POST request allowing the current active user to select which	Set the robotic arm configuration identified in the POST request as the currently active robotic arm the user is using.

		robotic arm they want to use.	
registerRoboticArm	POST	"/registerRoboticArm" endpoint accepts a post request with a new robotic arm configuration.	Store robotic arm configuration (connection details) in the database. Configuration is associated with the id of the currently active user.

Table 9: Robotic arm instruction related HTTP requests.

Request Name	Type	Description	Backend logic that occurs as a result
changeArmRequest	POST	"/changeArm/{uuid}" endpoint accepts a post request to select the current motor that is to be moved.	Send an HTTP request containing the same body to the desired robotic arm endpoint (based on uuid).
changeSliderRequest	POST	"/changeSlider/{uuid}" endpoint accepts a post request to instruct the robotic arm to change the angle of the currently selected motor to that specified in the body of the request.	Send an HTTP request containing the same body to the desired robotic arm endpoint (based on uuid).

6.2.2 MongoDB Database

Initially, the idea was to manually deploy an instance of MongoDB alongside the other services in our application. This proved to be quite difficult when using the first iteration of cloud deployment as the networking was complex. This method of deployment only had one instance of MongoDB running, which was not redundant nor highly available. As a simpler alternative, it was discovered that MongoDB offered a free tier of its 'Atlas' service, which provided a three instance MongoDB cluster that was redundant and highly available. This free tier was used as it had sufficient storage and performance, and suited our use case.

Regarding the organization of databases in the cloud, two databases were provisioned in Atlas: ‘checkup-bot-dev’ and ‘checkup-bot-stage’. The ‘dev’ database was to be used by developers, while the cloud deployed version of the application used ‘stage’.

MongoDB credentials and the name of the database to use were stored as a GitHub secret, and injected as an environment variable into the Spring Boot Docker container by the continuous integration pipeline. Spring Boot was then configured to import and use these credentials. This was a secure method, however, containers couldn’t be stored in a public registry. More information on this container registry development can be found in section 6.2.3.2.

MongoDB was used to store two kinds of entities which were named: DoctorDB and RoboticArmEndpoint. These database entries (or as MongoDB calls them, documents) are stored in a JSON format and can be seen in Figures 11 and 12 respectively.


```
{
        "_id":"65b825b5baa39fc098f0bcc9",
        "cpsoNumber":"123456789",
        "password":"$2a$12$FUdmw5OuxGs/H2eA3ZC2oeNFd60j6qaM9q7IEEMqdjlnkdUhHlJkK"
}
```


Figure 11: DoctorDB document.


```
{
        "_id":"57cef681-a408-4ee1-b495-4cfe7d8a6542",
        "name":"evan",
        "ip":"192.168.2.205",
        "port":{"$numberInt":"8005"},
        "active":false,
        "_class":"com.checkupbot.checkupbotbackend.documents.RoboticArmEndpoint"
}
```


Figure 12: RoboticArmEndpoint document.

6.2.3 Cloud Deployment

Cloud deployment involved the continuous integration pipeline, containerization of components, use of a reverse proxy, and selection of a cloud service to use for deployment. The cloud deployment strategy underwent a massive overhaul due to issues encountered halfway through the project. As a result, each subheading in this section describes iteration one and iteration two, referring to the system before and after the overhaul respectively.

6.2.3.1 Continuous Integration Pipeline

Iteration 1

GitHub actions were used successfully as the continuous integration tool for iteration one. Figure 13 shows a sequence diagram of what happens during continuous integration.

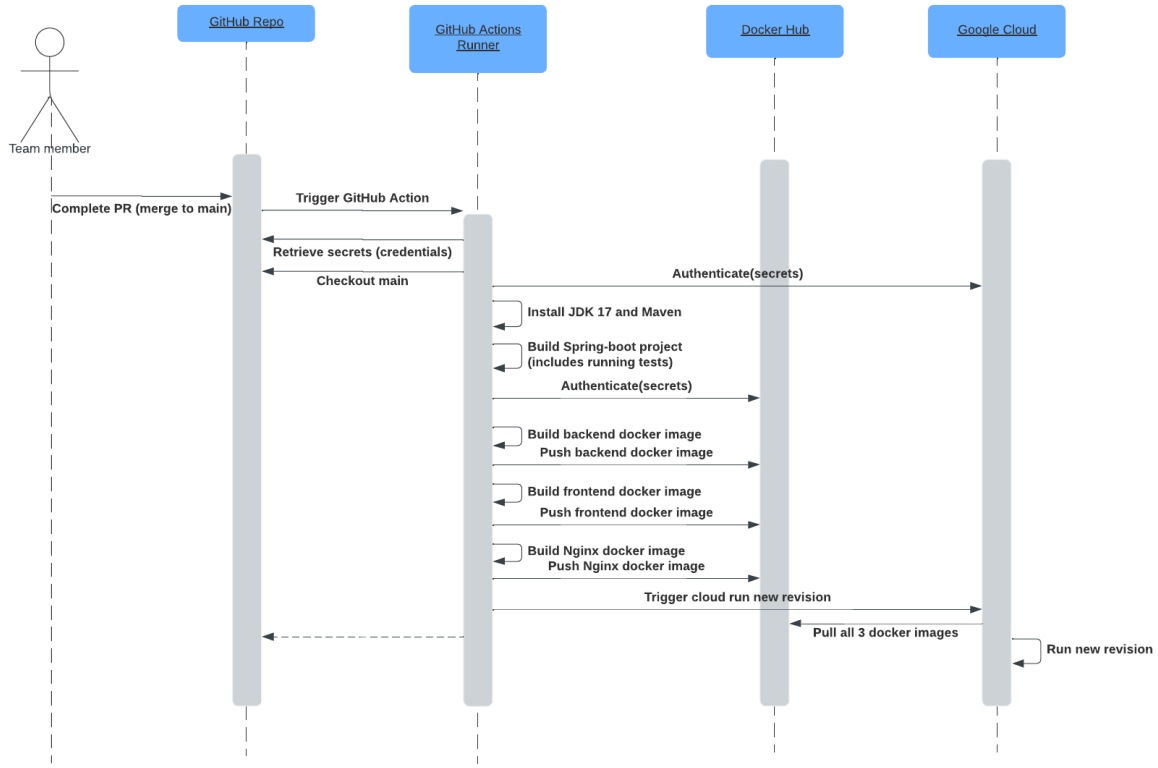


Figure 13: Iteration 1 continuous integration flow.

Iteration 2

In iteration two, the GitHub action was split so that instead of having one GitHub action that deployed everything, there were two GitHub actions that deployed the frontend and backend individually. Each action was modified to inject addresses, for networking between services, and credentials, for accessing the database into the Docker containers that were being built. The new GitHub action flows for the frontend and backend can be seen in figures 14 and 15 respectively.

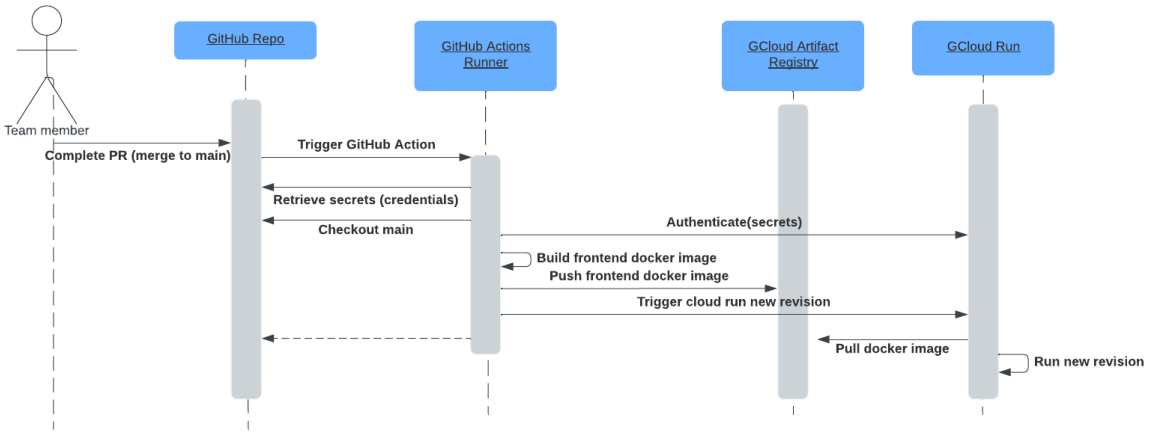


Figure 14: Iteration 2 frontend continuous integration flow.

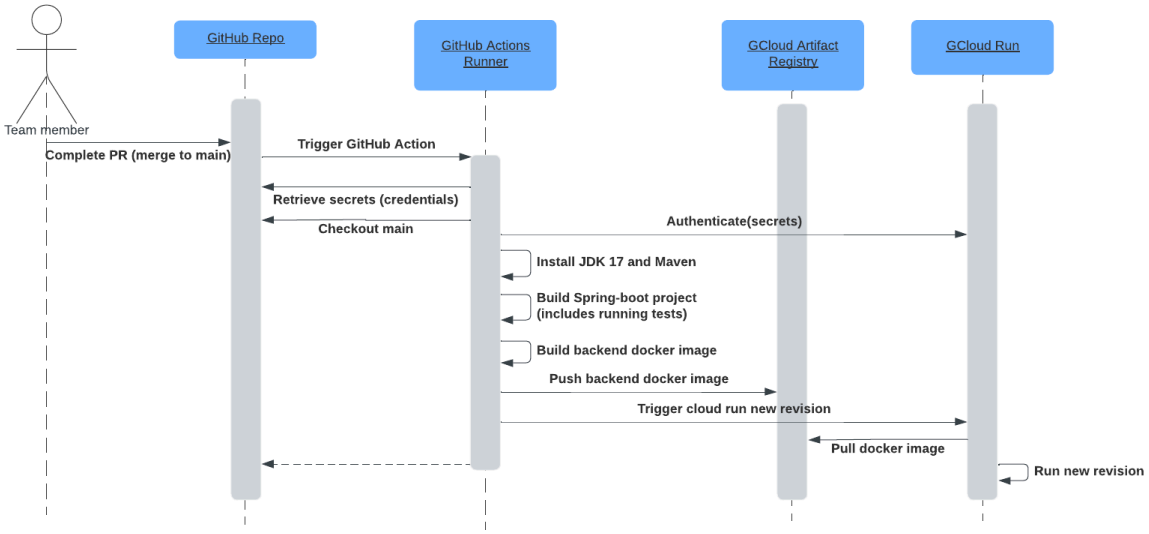


Figure 15: Iteration 2 backend continuous integration flow.

6.2.3.2 Component Containerization

The use of containerization in the deployment process has changed over the course of the project and can be classified into two iterations. The first and second iterations correspond to the first and second iterations of cloud deployment.

Iteration 1

Containerization was implemented successfully for the frontend, backend and reverse proxy. This was made to work in a developer environment using Docker Compose, and in the cloud environment using GitHub Actions. Docker was used to package each of the three components (frontend, backend and reverse proxy). While a lightweight Linux base image was still used (as initially planned), official Docker base images with dependencies already installed on them exist for Java, React and Nginx, so these images were leveraged to simplify the creation of Docker images as these dependencies do not need to be installed on the images. The components name, base images used, and what is required to build each Docker image is described in Table 10 below.

Table 10: Docker Container Components

Component name	Base image	Additional steps to build image
Backend	‘openjdk:17-jdk-alpine’ [15]	● Copy the app JAR file to the image ● Create the app entrypoint (‘java -jar <jarname>.jar’)
frontend	‘node:20-alpine’ [17]	● Copy the app directory to the image ● Install npm packages ● Create the app entry point (‘npm start’)
Nginx	‘nginx:alpine’ [16]	● Create configuration file to specify where to forward requests ● Configure SSL so that HTTPS can be used

To facilitate container version control and prepare containers for deployment, it is common practice to push containers to a container registry. Docker has a container registry (Docker Hub), and so do many cloud providers (Amazon Elastic Container Registry, Cloud Artifact Registry). The selected container registry for this iteration was Docker Hub.

It was decided that the team will use Docker Hub because:

- Docker Hub provides unlimited free public container repositories and one free private container repository, without requiring a credit card. Other cloud repositories only support private container repositories, require a credit card and charge for the amount of storage used.
- Docker Hub is well integrated with GitHub actions, with many reliable marketplace actions that can be used to build and push a Docker image to Docker Hub.
- Docker Hub is cloud agnostic, meaning that if this app was hosted with a different cloud provider, Docker Hub would not have to change.

Several difficulties were encountered with secure insertion of database credentials into the backend container. These difficulties required a complete change in container registry which is addressed by iteration 2.

Iteration 2

In iteration two of containerization, it was discovered that Docker Hub could not be used as the container registry. Docker Hub only offers public container storage for free, meaning that anyone can download your container and that any credentials included in a public container would be exposed. Although Docker Hub offers paid private container storage, this cannot be used with Google Cloud Run as there is no method for authentication. As a result it was decided what a different container registry must be used.

When selecting a container registry initially, Google Cloud Artifact Registry had not been chosen because it had a limited free tier that didn't seem to have enough storage for this use case. In the end, the private registry that it provided proved to be more important than the amount of storage. In order to ensure storage amount was not exceeded, strict limits for the number of container versions would be stored were imposed and container size was optimized. Changing to Google Cloud Artifact Registry ensured that the database credentials were secure and turned out to be quite simple to use.

6.2.3.3 Reverse Proxy Use

Iteration 1

Nginx [27] is a high-performance, open source software for web serving, reverse proxying, caching, load balancing, and media streaming [29]. Since the project application has both a frontend and a backend,

Nginx was used as a reverse proxy to direct requests to either the application server or the web server [28]. An Nginx Docker container was used to ease the setup of the reverse proxy in both a developer environment and the cloud deployment. The following trade off analysis was performed when selecting Nginx as a reverse proxy:

It was decided that the team would use Nginx for the reverse proxy because:

- Of the two most common reverse proxies (Nginx and Apache), Nginx is the more lightweight of the two, consuming fewer resources due to its asynchronous, event-driven architecture
- Nginx is newer than Apache
- Nginx configuration is simpler and more readable than Apache
- Team member Evan has used Nginx before

Iteration 2

Nginx was used successfully in the first iteration of cloud deployment but in the second iteration it is no longer necessary. Deployment of the frontend and backend as separate Google Cloud Run services eliminates the need for networking within the container group, which was the main purpose of using Nginx.

6.2.3.4 Cloud Provider

Iteration 1

Initially AWS EC2 was going to be used to deploy the application. AWS EC2 would provide a VM to deploy the application [3]. However, after further research, the cloud service AWS Fargate was a more modern and low difficulty option. AWS Fargate is a managed container running service that allows the deployment of containers, pulled from a container registry. This simplifies the deployment process as otherwise a VM image would have to be created with everything necessary installed on it, making it very challenging to configure what was installed on the VM as a part of a continuous integration pipeline. Instead, during continuous integration, a docker image can be built with all necessary dependencies installed on it, the image can be stored in a container registry and then the container can be pulled to the managed container service.

After an extensive proof of concept with AWS Fargate, it was determined that this service could not be used within the bounds of AWS free tier and that it would cost too much to use. In an effort to still use this modern deployment method, research was performed to look into use of the Google Cloud managed

container offering, Cloud Run. The Google Cloud free tier is structured differently, allowing for a certain amount of requests, memory usage and CPU usage with a Cloud Run service for free each month [23]. Additionally, after account creation, a $416 of free credit ($300 USD) is applied to your account, which would cover any additional costs incurred during deployment of this project until the end of April [24].

Google Cloud Run allows creation of multiple containers in one service, with one ingress container (the reverse proxy) and several sidecar containers that can be networked to work with the reverse proxy. This fits this project's use case perfectly and can be configured with continuous integration to mimic the local dev environment exactly. Figure 16 shows an example of how the app would be deployed.

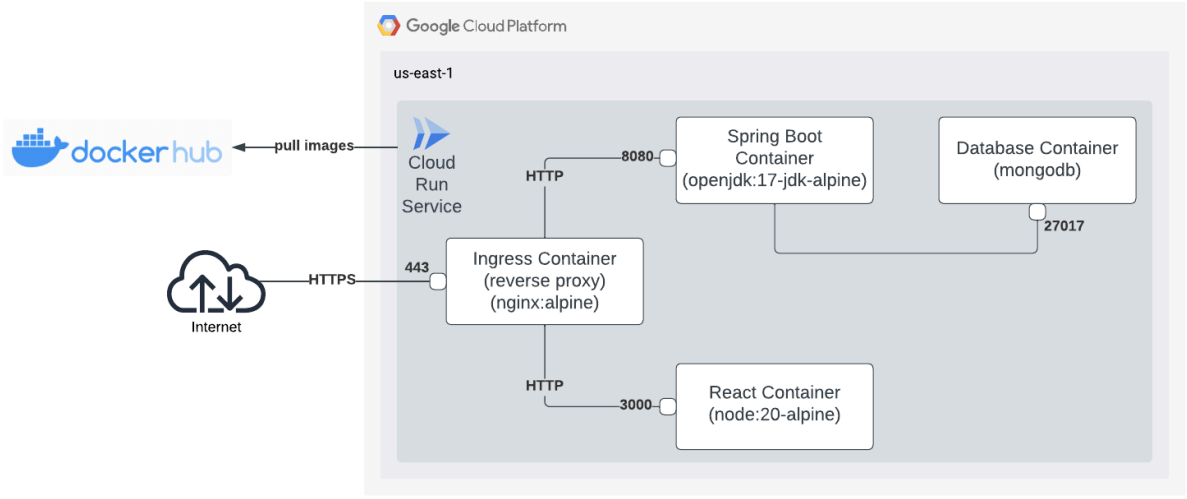


Figure 16: Example Docker Deployment Diagram

The reasoning for the selection of Google Cloud Run as the deployment platform for the Checkup Bot app is given above, but to summarize, Google Cloud Run was chosen because:

- Google Cloud offers a Free Tier that is superior to that of AWS in the short term, allowing Google Cloud Run to be used for free for the duration of this project.
- Google Cloud Run allows containerized deployment of multiple containers with networking between them in a simple way, which suits this project's use case very well.
- Google Cloud Run has easy-to-understand methods for deployment of new containers using continuous integration.

Iteration 2

In iteration two the cloud provider did not change, and nor did the service used (still Cloud Run), however the way in which Cloud Run was used changed. Instead of deploying all containers (frontend and backend) in one container group and networking within the container group, the frontend and backend were deployed separately. This is beneficial as they can each individually be scaled based on the load they are required to handle and they can each be redeployed independently. Figure 17 shows the architecture of the second iteration of cloud deployment.

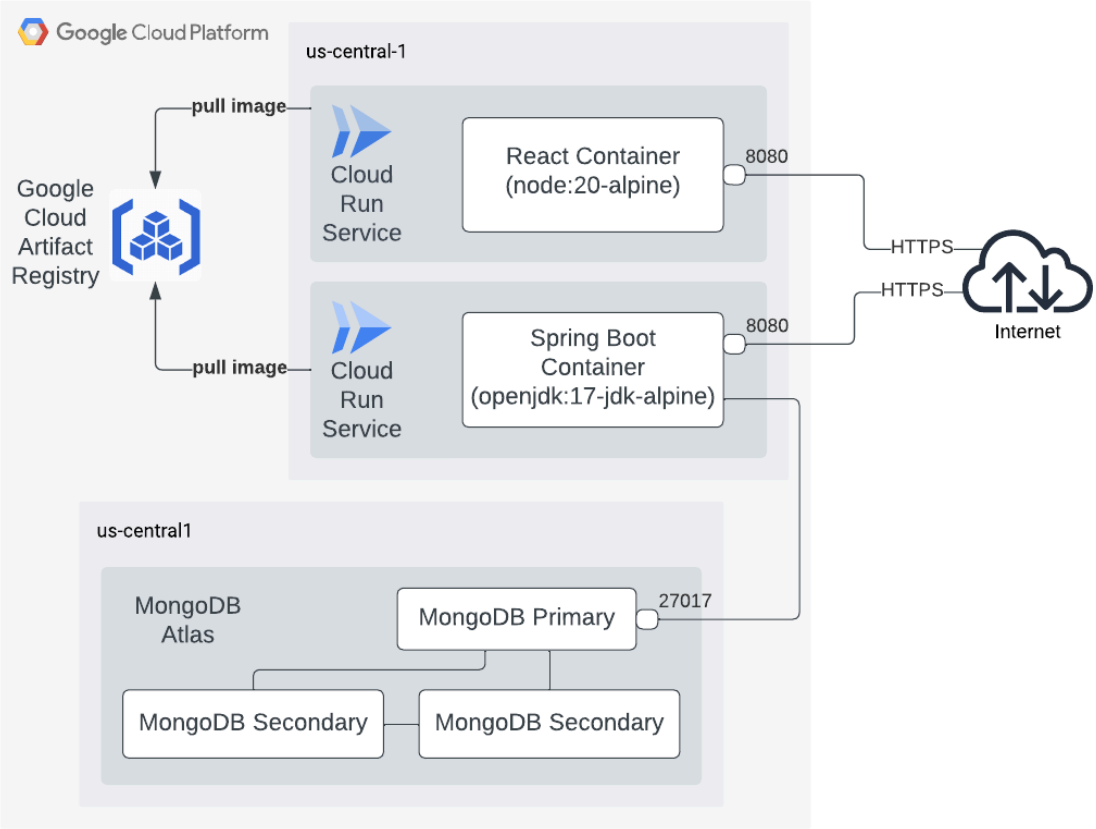


Figure 17: Iteration two of cloud deployment with frontend and backend separated, new MongoDB deployment method and new artifact registry.

6.2.4 Security

The Spring Security implementation in our backend system has been built around four main components. First, HTTP Requests will be validated by an authentication filter. The filter has the responsibility of checking if the request contains a JSON Web Token, or if the request has been directed to a whitelisted endpoint. The only whitelisted endpoints are those for login and robotic arm registration as they will need to be provided with new JSON Web Tokens. A request that passes through the authentication filter will then be validated by the JWT Service. This service is responsible for checking the validity of a request token, extracting data from a token, and creating a new token when necessary. When a user makes a login request, the User Details Service will check our database for matching credentials. Finally, our authentication controller is responsible for providing the response data.

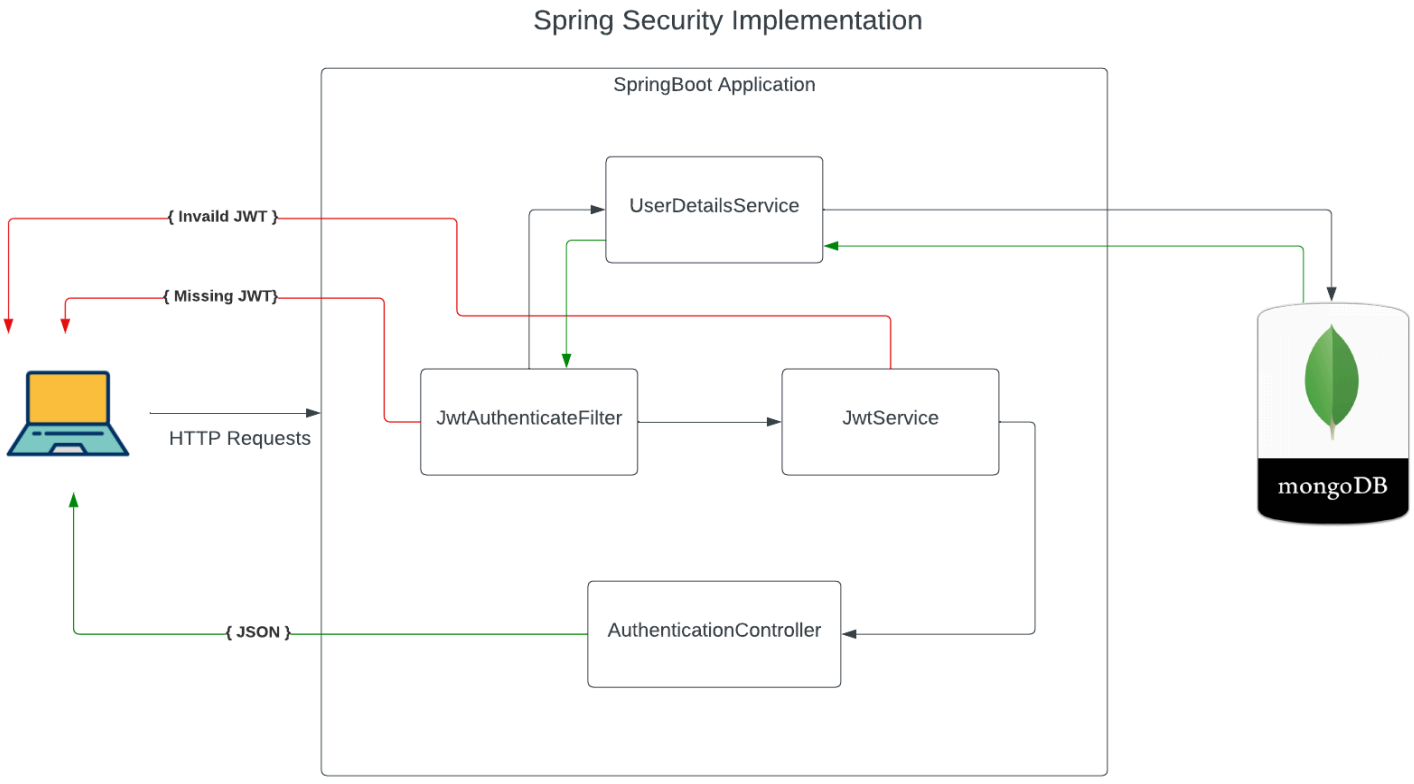


Figure 18: Spring Security Implementation

6.2.5 Robotic Arm Endpoint API

The robotic arm endpoint API is written in Flask (see 6.1.5 for more information about Flask) and exposes three HTTP POST endpoints that are used for monitoring endpoint activity and sending movement instructions for the robotic arm.

Activity monitoring is carried out through the liveness probe endpoint. The backend will periodically send liveness probes to the endpoint and the endpoint must authenticate itself by returning a description of itself including its port, address and uuid. The backend will then authenticate the endpoint and show it as active.

Arm control is carried out through the /changeArm and /changeSlider methods, which contain a motor number and angle respectively. These instructions are interpreted by the endpoint and used to actually move the arm.

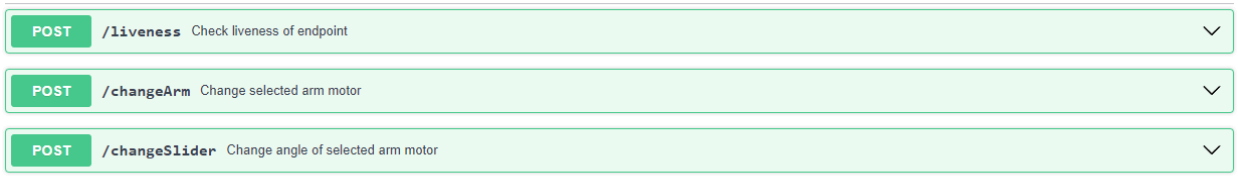


Figure 19: Robotic arm endpoint API visualized with OpenAPI and SwaggerUI.

6.2.6 Zoom Video Conferencing Integration

To facilitate the integration of Zoom video conferencing directly into the website, the Zoom Meeting SDK, described in section 6.1.6, was used. This integration was done in three parts. First, a private Zoom for Developers app was created on the Zoom App Marketplace to retrieve Meeting SDK credentials to authorize our custom Zoom app to join Zoom calls. Then, since the Meeting SDK uses a JWT for authorization, a authorization server was created to generate, validate, and renew this JWT. This server was also deployed in the cloud allowing for Zoom authentication at all times. Finally, the meeting sdk was used on the frontend to add the Zoom meeting window. To join a zoom call, the frontend needed the authentication server endpoint path and the Meeting SDK credentials obtained from the custom app. This data was stored securely in the MongoDB database and APIs were created for the secure passing of this information to the frontend upon request.

6.3 Testing

Testing of the Spring Boot backend was performed entirely through integration tests. The functionality the backend provides is very trivial and primarily based around database operations, so unit tests were not needed. Integration tests were written for each HTTP endpoint using a mock database to ensure that requests and responses were properly formatted. Additionally, the connection to MongoDB was also tested using an integration test.

All of the Spring Boot tests are automatically run during the continuous integration process and, following good practice, a pull request is only allowed if all tests pass.

Testing of the robotic arm endpoint has been limited and was mostly done manually. The endpoint is only responsible for unpacking requests and calling arm methods, making testing trivial and also difficult unless the whole system is set up with the robotic arm. Developing a method for testing is a good next step and would require more extensive thought.

6.4 Accomplishments

The following is a list of what has been accomplished with regards to the backend over the course of the project.

- The Spring Boot API for the backend service was written and tested.
- Continuous integration pipelines for deployment of the backend and frontend services were created. They included the running of tests and injection of secrets needed for the configuration of each service. They are both triggered by PR completion.
- A MongoDB Atlas cluster was provisioned in the Google Cloud and successfully configured for cloud and local developer use.
- The backend did what it was supposed to do consistently and its deployment did the same!

6.5 Resolved Issues

Table 11 contains a short description of all of the issues faced and solutions found for the backend. Many of these issues were covered in more detail in the developments section and are summarized here.

Table 11: Issues and solutions encountered with respect to the backend.

Issue	**Solution(s)**
How can the system be deployed in the cloud while staying within a strict budget of $0?	● Selected Google Cloud as a cloud provider due to its extensive free tier ($300 of free credits). ● Set hosted services (frontend and backend) to scale down to zero instances when not in use (reduce costs to zero). ● Imposed strict storage limits for all images stored in Google Cloud Artifact Registry, so that older versions were automatically deleted to avoid exceeding the free tier.
When within the same container group, how can the frontend and backend communicate?	● Strategic container naming and Docker network configuration allowed addresses frontend and backend to communicate with one another to be hard coded.
When there is only one entry point for a container group, how can requests be separated based on whether they are intended for the frontend or backend?	● Nginx was used as a reverse proxy to redirect requests based on their path.
The backend service needs to be able to access MongoDB, how can it be provided with MongoDB credentials in a secure manner?	● No credential hardcoding. ● Credentials are stored as a GitHub secret so that they do not show up in any GitHub actions run logs (which may be public). ● Credentials are injected into the backend Docker image through a GitHub action by specifying them as arguments and setting them in the backend Dockerfile. ● Google Cloud Artifact Registry was used to store containers privately.
The latency is too high. How can it be decreased?	● In order to monitor latency, a metric was added to the frontend to monitor it easily. ● The number of hops a request must make to get from the frontend to the endpoint was reduced from two to one by having it go directly to the endpoint from the frontend, reducing

	transmission and processing delay by half.
How can the frontend and backend be scaled individually?	● Components were deployed as individual services. ● Separate GitHub actions for the deployment of each were used.
The robotic arm position shown in the frontend does not reflect its actual position (no feedback). This can sometimes result in erratic movement. How can this feedback be performed?	● Motor angle is monitored on the robotic arm. ● Motor angle is returned to the frontend each time a different motor is selected.
How can HTTPS be implemented between the robotic arm and the other components?	● A certificate must be purchased. ● This was not implemented
There needs to be separation between which users can control which arms. How can this be implemented?	● By including a list of robotic arm endpoints that a user is allowed to access in said user's DoctorDB record. ● This was not implemented.

7 Robotic Arm

7.1 Trade-Off Analysis

The project uses a Yahboom robotic arm for Raspberry Pi 4B with a camera, purchased from Amazon [45]. This robotic arm was controlled by a Raspberry Pi using Python.

It was decided that the team would use the Yahboom robotic arm [45] for Raspberry Pi 4B as it was the only robot which met the desired specifications:

- Had 6 degrees of freedom in the robotic arm, which provides more points of manipulability and a higher dexterity for irregularly shaped objects such as tongue depressor.
- Had a camera mounted on the robot.
- Used a Raspberry Pi 4B which was already accessible via the department, so we did not need to buy extra parts.
- Is small enough to be non-intimidating to the end user.
- Could pick up a load of up to five hundred grams, which is more than enough for our purposes.
- Supported Wi-Fi connectivity out of the box.
- Costed under 500 dollars.

7.2 Robotic Arm Components

For this project, the robotic arm that fits the project requirements (degrees of freedom and cost) the best was the Yahboom Robotic Arm for Raspberry Pi 4B AI Smart Robotic Arm with Camera. This robot kit included six servo motors and a camera mounted on the fifth linkage. The cost of the robot kit was $430 without tax and shipping. LEDs mounted on the robotic arm serving as a flashlight costs 50 cents. The Raspberry Pi 4B was provided by the school at no additional cost.

Table 12: List of hardware components.

Component	**Voltage**	**Current Limit**	**Communication Protocol(s)**	**Description**
Raspberry Pi 4B	3.3 or 5V output	1-2A output	GPIO/ I2C/ CSI/ UART	Handles the data from the camera and sends instructions to motors. Communicates directionally to and from the web application.
Yahboom Robotic Arm for Raspberry Pi 4B	100 - 240 V (50/60 Hz) AC to 12V 5A			The robotic arm consists of 6 servo motors and 1 Raspberry Pi Camera Module V2 and a full-function expansion board which interfaces the Raspberry Pi, power supply and arm.
Digital Servo Motor (x4) - Torque 15KG Angle 1- 180 degrees	Not Specified	Not Specified	GPIO	To control and read each degree of freedom for the bottom 4 motors.
Digital Servo Motor (x1) - Torque 6KG Angle 1- 180 degrees	Not Specified	Not Specified	GPIO	The motor to control the “open and close” function of the gripper for the robotic arm.
Digital Servo Motor (x1) - Torque 15KG Angle 1- 270 degrees	Not Specified	Not Specified	GPIO	The motor rotates the gripper from 1-270 degrees.
Raspberry Pi Camera Module V2	3.3V	2A	CSI	Camera streams the video back to the camera in 1080p. It is attached to the 5th linkage

				on the Robotic Arm and is included in the kit.
(x3) 5V LED [43]	5V	2A	GPIO	A series of Light Emitting Diodes (LED) attached to the 5th linkage on the Robotic Arm.

7.3 Developments

The three main objects the robot needed to pick up, as outlined in the project proposal, is the mallet, the popsicle stick, and the flashlight. The flashlight needed to be turned on prior to usage. Initially, to implement this task, either the user is expected to turn it on, or a holder would be made to stabilize the flashlight when the robot pushes the button. In order to simplify the mechanism and reduce dependency on the patient, 3 [43] LEDs on a PCB mounted on the fifth link attached with a clamp served as an integrated flashlight. The LEDs were controlled through the expansion boards GPIO pins. When the command to turn on the flashlight was sent by the doctor, all the LEDs were turned on. All LEDs emitted the RGB color (255, 255, 255), otherwise known as white.

In order for the picamera stream from the robot to be seen on the web app, an HTTP streaming server was set up on the robot. The server continuously looks for each new frame from the camera stream, which is then converted into JPEG format and sent to the HTTP streaming server. The stream also has mouth detection functionality which was completed by using OpenCV [33]. Each frame is first converted into grayscale, then OpenCV takes a Haar cascade file, which includes the training model for detecting mouths, to then create a box around detected mouths in the camera stream.

7.4 Testing

The Robotic Arm was mainly tested manually. This includes testing how far the robotic arm should move depending on the angles of each motor. The robotic arm has motors set to specific angles with a set delay, then the testing requests the current position of the robotic arm and validates the position by comparing the sent angle values and the current position and checking if they are equal. To test if the robot is still executing commands, an LED on the expansion board was set to rainbow colours. The robot arm's mouth

detection was also tested by running the robot arm's camera and visually seeing a bounding box around any mouth's in the camera.

7.5 Accomplishments

The Robotic Arm was assembled by using the instructions provided, connected to the WiFi network and able to pick up objects from a flat surface. The Raspberry Pi had been configured to use a static IP address so the backend interface had an easier time addressing the robot. The library provided by the manufacturer, Yahboom, was used to implement functions to direct the robot in 3 axis actions by the robot. It used a combination of those actions and OpenCV [33] to find the contours of the objects to pick them up. The picamera stream is sent to the web app through HTTP streaming and OpenCV [33] is used for mouth detection.

7.6 Resolved Issues

When the doctor was using tele-operation to manipulate the robot, they maneuvered each of the joints using the keyboard. Each joint had its own frame of reference with which it moved in. Therefore, the robot was prone to collisions with the table it rested on and the Raspberry Pi mounted on it. In order to minimize damage to the Robotic Arm, restrictions for movement were placed around 2 planes on the robot, namely the z plane for the table and a x-y plane over the robot where any joint could not cross. Each joint position was calculated using forward kinematics to see if a joint would go below the 2 defined planes. If it did, the robot would cease moving and record its position. The robot would resume its movements if the next position was not below the defined planes. In addition, motor 6 which activated the claw, was no longer closing due to the voltage not inverting within the motor. Therefore, the claw was open no matter the signal from the command. To temporarily resolve the issue, the wire connecting to motor 6 was disconnected and an elastic band was wrapped around the end effector. The patient is to place the tool between the jaws of the claw and secure it with an elastic band.

8 Timeline

The timeline of the project outlines the milestones, tasks, and approaches for the stages required for this project. The stages include problem and solution, identification, development, and testing. A Gantt chart detailing the expected start dates of each milestone, the expected duration of each milestone, and the percent done so far for each milestone. As requirements have become better understood, the timeline from the progress report has been revised, with some tasks changed and some completely new milestones added.

8.1 Problem and Solution Identification Milestone

Milestone 1: Submit draft project proposal [October 13th] (Completed)

Task 1.1 Research technologies of interest to use in the project.

Task 1.2 Find a problem to solve and propose a solution.

Task 1.3 Create project objectives.

- Who are the stakeholders (end-users, target audience).

8.2 Research Milestone

Milestone 2: Submit project proposal [October 30th] (Completed)

Task 2.1 Create milestones and tasks for the project.

Task 2.2 Create project functional requirements. This is done by:

- Defining the project scope and identifying stakeholders.
- Define what the project should be able to do at the end.

Task 2.3 Create project non-functional requirements.

- Define stakeholder expectations of the project.
    - Usability, accessibility, reliability, availability, etc.

Task 2.4 Create sequence diagrams.

Task 2.5 Create a pros and cons list for technologies that can be used in the project.

- Consider the technologies that can be used and explain why a specific technology was used over another in a concise manner.

Task 2.6 Research possible risks in the project and ways to mitigate.

8.3 Testing Milestone

Note, Testing (Section 8.3) and Development (Section 8.4) are done concurrently.

Milestone 3: Develop a test plan [December 30th] (Partially Completed)

Task 3.1 Develop an integration test plan.

- Identify components that need testing.
- Identify key scenarios that need to be tested.

Task 3.2 Develop a unit test plan.

- Identify components that need testing.
- Identify key scenarios that need to be tested.

Milestone 4: Develop a test suite on December [March 31st] (Partially Completed)

Task 4.1 Develop integration tests and select testing criteria.

- Integration testing will begin when at least one feature has been developed.
- For every new feature, if the API of the service has been changed, a new integration test will be added.

Task 4.2 Develop unit tests and select testing criteria.

- Unit testing will begin when at least one feature has been developed.
- For every new feature, a new unit test will be added for each unit of code.

8.4 Development Milestone

Milestone 5: Robotic Arm is acquired [October 18th] (Completed)

Task 5.1 Connect and test the robotic arm with board to see if all components are functional according to the specifications.

Task 5.2 Write and test code to control each motor using only the Raspberry Pi and the robotic arm's associated components.

Task 5.3 Wire the hard stop and start buttons. Test if the robotic arm powers on and off accordingly.

Milestone 6: Deploy servers and establish continuous integration [November 30th] (Completed)

Task 6.1 Deploy web server in the cloud.

Task 6.2 Deploy application server in the cloud.

Task 6.3 Establish continuous integration pipeline.

Milestone 7: Create all static user interface pages [December 20th] (Completed)

Task 7.1 Web application has a landing page that directs to a login page.

Task 7.2 Web application has a login page.

Task 7.3 Web application has a view for controlling the robot with the necessary buttons and screens.

- The camera stream, doctor's notes, buttons to move the arm, buttons to pick up items (mallet, tongue depressor, and flashlight).
- Note: the buttons moving the robotic arm should not be functional at this point, just shown on the web application. This will be added later.

Milestone 8: Acquire a mallet, tongue depressor, and flashlight [December 21st] (Partially Completed)

Task 8.1 Create using Fusion360, build or find a tray that fits the dimensions of each medical tool.

Task 8.2 Establish/create a mount for the tray next to the robot so it is in a consistent spot for the robotic arm.

Task 8.3 Write a pre-set routine for each medical device.

Milestone 9: Add persistence to the backend server for login functionality [January 7th] (Completed)

Task 9.1 Select relational databases to be used.

Task 9.2 Configure the relational database to connect to the backend.

Task 9.3 Persist doctor data in a database, such as username and CPSO number, to be able to distinguish between doctor's that have been authenticated and anyone else trying to access the robotic arm controls.

Task 9.4 Users can successfully login through the website.

Milestone 10: Establish a connection between the frontend, backend, and robotic arm [January 7th] (Completed)

Task 10.1 frontend button presses can properly send http requests to the backend.

Task 10.2 Http requests received in the backend trigger communication with the robotic arm endpoint.

Milestone 11: Web application buttons can move the robotic arm [January 12st] (Completed)

Task 11.1 Develop functionality for the web application buttons to move the robotic arm up/down, left/right and change the depth.

Task 11.2 Develop buttons to move the robotic arm in preset routines to pick up a mallet, tongue depressor and flashlight.

Milestone 12: Establish camera live streaming from the robotic arm to Twitch [February 29th] (Completed)

Task 12.1 Integrate the Twitch application into the web application.

Task 12.2 Configure the Twitch stream on the Raspberry Pi.

8.5 Project Deliverables Milestone

Milestone 13: Submit project progress report [December 8th] (Completed)

Task 13.1 Complete draft of progress report and request feedback from the supervisor on November 17th.

Task 13.2 Make adjustments based on the feedback.

Milestone 14: Oral presentation [January 25th-30th] (Completed)

Task 14.1 Submit the oral presentation form.

Task 14.2 Create a script to present a working product.

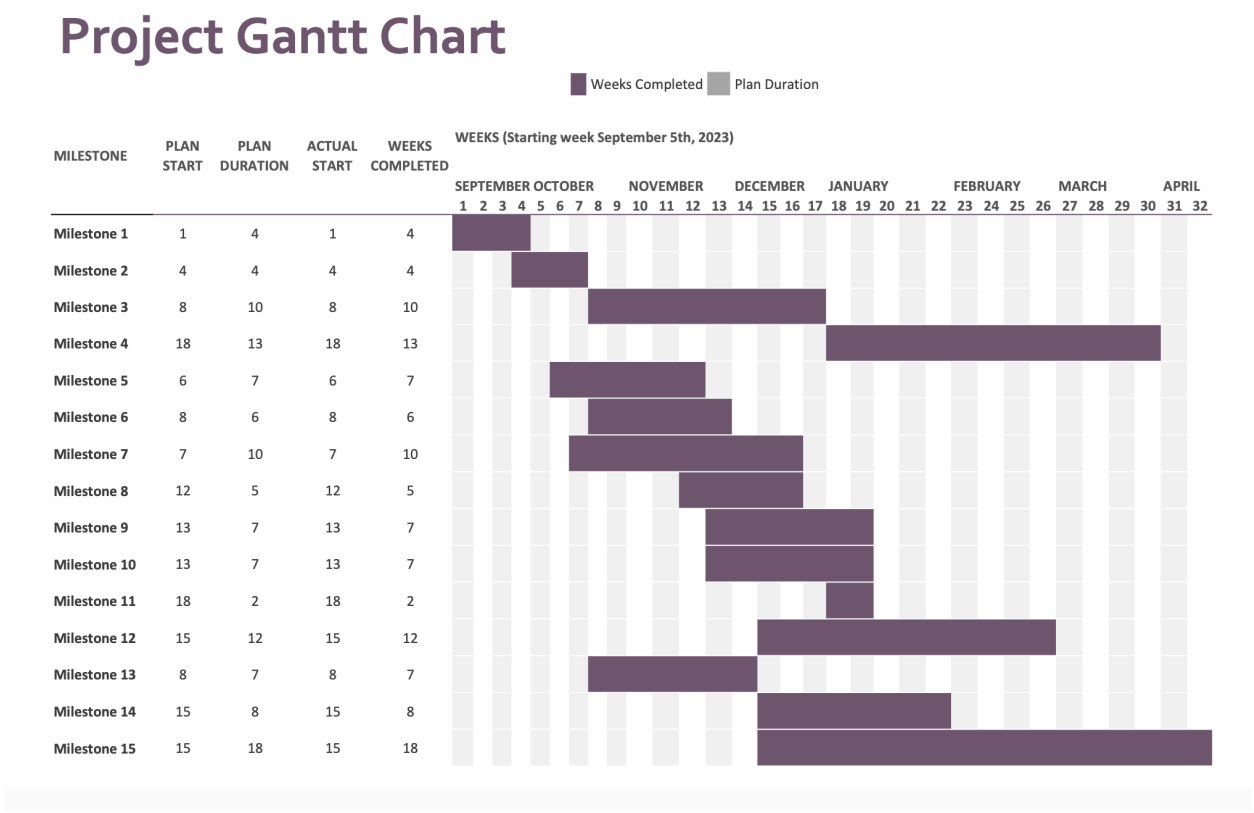
- Show the robotic arm moving according to the buttons pressed on the web application.

Milestone 15: Submit final project report [April 10th] (Completed)

Task 15.1 Complete draft 1 of progress report and request feedback from the supervisor on February 16th.

Task 15.2 Complete draft 2 of progress report by making adjustments based on the feedback from draft 1 and request feedback from the supervisor on March 14th.

Task 15.3 Make final adjustments and based on the feedback from draft 2.



MILESTONE	PLAN START	PLAN DURATION	ACTUAL START	WEEKS COMPLETED
Milestone 1	1	4	1	4
Milestone 2	4	4	4	4
Milestone 3	8	10	8	10
Milestone 4	18	13	18	13
Milestone 5	6	7	6	7
Milestone 6	8	6	8	6
Milestone 7	7	10	7	10
Milestone 8	12	5	12	5
Milestone 9	13	7	13	7
Milestone 10	13	7	13	7
Milestone 11	18	2	18	2
Milestone 12	15	12	15	12
Milestone 13	8	7	8	7
Milestone 14	15	8	15	8
Milestone 15	15	18	15	18

Figure 20: Gantt chart for the robotic arm project.

9 Conclusions

Changes have been made to how we implement functionalities in this project. This is due to discovering new and better ways of solving challenges.

In the process of creating the frontend, it was found that React and Typescript were the best frameworks and languages to use to develop this web page. Additionally, MaterialUI library provided a cohesive theme that brought all of the components on the page together.

While researching the use of Spring Boot Webflux with a database for the backend, it was found that the reactive features cannot be effectively leveraged unless Spring Data Reactive Repositories is used. JPA cannot be used reactively. Spring Data Reactive Repositories has integration with different databases, making some databases more desirable than others for use with Reactive Spring Boot.

While attempting to stay within the free tier when using cloud services for this project, use of the AWS Elastic Container Service was found to be unusable with backing instances that do not have GPU. None of the free tier EC2 instance types have any GPU so ECS cannot be used for free. The cost of running the service in AWS for the duration of the project was calculated to exceed the project budget, therefore AWS ECS was not used for this project.

Google Cloud was found to provide a better free tier service, but only for the first 3 months. After the first 3 months, the service runs off of the free credits from the trial period until it runs out. This free tier is more suitable for this project as it only needs to be functional until the end of April. In summation, attempting to use a cloud free tier puts severe limitations on how the application can be deployed.

During the configuration and setup of the robotic arm, it has been found that OpenCV would need to be a critical part of the robot to operate with efficacy and precision. It gives the robot the ability to perceive objects and contours. Adding the computer vision component to this project allows for the pre-programmed buttons to dynamically adapt to its surroundings in contrast to the original idea of “hard coding” actions. Therefore, even when objects are misplaced or placed outside of a predefined position in relation to the robot, it can still apprehend the appropriate tool. In addition, guiding the doctor to the approximate area of the ulnar nerve will improve the experience for the doctor and speed up the process of the examination. Another change is relying minimally on the user for setup which removes a potential hurdle in the user accessibility and issues with setup. Thus, we have chosen to attach the flashlight on the

robot arm itself to guarantee the flashlight is turned on, illuminates hard to see areas, and has a long enough battery life for the task.

The identified next steps for improvements on the frontend include:

- Keyboard controls for the arm
    - Using functions 1-6 for the buttons
- Logout functionality
- Implement the use of a console as an alternative method to control the robotic arm

The identified steps for the backend portion of the project include the following:

- Switch backend to Spring Boot Webflux to use a non blocking streams API rather than the Java Servlet API
- Evaluate cloud deployment mechanism to minimize cost
- Keep base Docker images up to date
- Continue to optimize latency

The next step for the robot arm is to design a solution so the robot can test a patient's reflexes. The arm team has identified a list of tasks to complete this design. They are as follows:

- Find a dataset or pretrained model to find the ulnar nerve. If such a data set is not available, create a dataset by recording a video and separating them into frames, then draw bounding boxes around the target area. Use OpenCV [33] to translate those bounding boxes on the joint (to assist the doctor hit the correct point).
- Use OpenCV [33] to find the correct distance away from the joint to hit the ulnar nerve properly.
- Understand the effects of the rebounding force generated from the mallet colliding with the ulnar nerve and how it can affect the system. This includes displacement when contact occurs with any of the objects causing a shift in grip positioning.
- Add a pressure sensor to the end effector of the robot to receive feedback on how much pressure is being applied.
- Add haptic feedback through a game controller for the doctor to sense how much pressure they are applying at a given position.
- Replace sixth servo motor to allow the claw to open and close.
- Unlike the tongue depressor (popsicle stick), the mallet’s center of gravity is not at the midpoint of the object. Objects with a lopsided center of gravity will slip out of the robot’s claw if the tool is grasped too far from the object center point, and will require more force to keep it stable.

Therefore we needed to implement computer vision software to identify the mallet's position and its center of gravity. The density of the mallet's head and the popsicle sticks are similar. Thus, OpenCV [33] will be used to identify the centroid of the shape from the camera's capture of the object.

- Use OpenCV [33] to find the center of gravity of the mallet, flashlight, and popsicle stick

10 Reflections

Throughout the implementation of this project, when attempting to design a solution for a large portion of the project, the team initially tried to tackle the problem simultaneously. This usually results in a lack of adequate time allocation as the team does not fully understand the scope and the potential challenges. When approaching problems, the team should have prioritized deconstructing the problem into smaller components. Then, by solving each component separately, the task becomes less daunting and more achievable through collective effort. This would have resulted in a more productive work environment and would have prevented learning unnecessary new technologies or project related topics.

In order to work most efficiently, aspects of the project should have been broken down into items with consideration towards which items can be completed in parallel. When items could not have been done in parallel, some team members were prevented from continuing until another team member completed their item, thereby wasting valuable time. Parallel items and design should have been considered for the duration of the project and at each meeting. The team should have ensured that all team members had a work item to complete that is not dependent on another component of the project.

In order to have continuous progress, the creation of an action plan for each team member with attainable goals would have been key to progress being consistently being made. This would have been a useful goal to have in place for the project.

Retrospectively, ensuring the budget allocated was used cautiously would have allowed more funds for software tools. For this project, the overwhelming majority of the budget was allocated towards the robotic arm, leaving a measly budget left for cloud hosting. The team thought that free tier cloud hosting would be sufficient, but, as we quickly found out, we required more computing power than what was provided in free tiers. If there were more in the budget for hosting, this would not have been an issue, but instead the team was forced to find a more creative deployment strategy.

References

[1] "Agile Manifesto - 12 Principles," Agile Manifesto, [Online]. Available: https://Agilemanifesto.org/principles.html. Accessed on: October 24, 2023.

[2] "About - Alpine Linux," Alpine Linux, [Online]. Available: https://www.alpinelinux.org/about/#:~:text=Alpine%20Linux%20is%20built%20around,of%20packages%20from%20the%20repository. Accessed on: October 24, 2023.

[3] "Amazon EC2 - Elastic Compute Cloud," Amazon Web Services, [Online]. Available: https://aws.amazon.com/ec2/?did=ft_card&trk=ft_card. Accessed on: October 24, 2023.

[4] "AWS Free Tier - Amazon Web Services (AWS)," Amazon Web Services, [Online]. Available: https://aws.amazon.com/free/. Accessed on: October 24, 2023.

[5] "Apache Maven - Welcome to Apache Maven," Apache Maven, [Online]. Available: https://maven.apache.org/index.html. Accessed on: October 24, 2023.

[6] "Continuous Delivery and Continuous Integration," Atlassian, [Online]. Available: https://www.atlassian.com/continuous-delivery/continuous-integration. Accessed on: October 24, 2023.

[7] "Kanban - Atlassian," Atlassian, [Online]. Available: https://www.atlassian.com/Agile/Kanban#:~:text=In%20Japanese%2C%20Kanban%20literally%20translates,in%20a%20highly%20visual%20manner. Accessed on: October 24, 2023.

[8] “Stress Health And Safety In Software Project Management,” Business Managment, [Online]. Available: https://business.joellemena.com/project-managment/stress-health-and-safety-in-software-project-management/. Accessed December 8, 2023.

[9] Carleton University, "ECOR 4995 - Undergraduate Academic Support," [Online]. Available: https://carleton.ca/engineering-design/current-students/undergrad-academic-support/ecor-495/. Accessed on: October 24, 2023.

[10] “Carleton University Calendars 2023-24 edition,” Systems and Computer Engineering (SYSC) < Carleton University, https://calendar.carleton.ca/undergrad/courses/SYSC/. Accessed on October 25, 2023.

[11] "Docker Official Images - Alpine," Docker Hub, [Online]. Available: https://hub.docker.com/_/alpine. Accessed on: October 24, 2023.

[12] "Docker Documentation - Build Images," Docker, [Online]. Available: https://docs.docker.com/build/. Accessed on: October 24, 2023.

[13] "Docker Documentation - Compose," Docker, [Online]. Available: https://docs.docker.com/compose/. Accessed on: October 24, 2023.

[14] “Docker images,” Docker Documentation, https://docs.docker.com/engine/reference/commandline/images/. Accessed on October 25, 2023.

[15] “OpenJDK,” Docker Hub, [Online]. Available: https://hub.docker.com/_/openjdk. Accessed December 8, 2023.

[16] “NgInx,” Docker, [Online]. Available: https://hub.docker.com/_/nginx. Accessed December 8, 2023.

[17] “Node,” Docker, [Online]. Available: https://hub.docker.com/_/node. Accessed December 8, 2023.

[18] "Companies using Linked Data in 2023," GitHub, [Online]. Available: https://github.com/d2s/companies/blob/master/src/index.md. Accessed on: October 24, 2023.

[19] "GitHub Actions Documentation," GitHub, [Online]. Available: https://docs.github.com/en/actions. Accessed on: October 24, 2023.

[20] "GitHub Actions Starter Workflows - AWS Deployment Workflow," GitHub, [Online]. Available: https://github.com/actions/starter-workflows/blob/main/deployments/aws.yml. Accessed on: October 24, 2023.

[21] "GitHub Actions Documentation - Understanding GitHub Actions," GitHub, [Online]. Available: https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions. Accessed on: October 24, 2023.

[22] "GitHub Actions Documentation - Usage Limits, Billing, and Administration," GitHub, [Online]. Available: https://docs.github.com/en/actions/learn-github-actions/usage-limits-billing-and-administration. Accessed on: October 24, 2023.

[23] “Cloud Run pricing,” Google Cloud, [Online]. Available: https://cloud.google.com/run/pricing. Accessed December 8, 2023.

[24] “Free cloud features and trial offer,” Google Cloud, [Online]. Available: https://cloud.google.com/free/docs/free-cloud-features. Accessed December 8, 2023.

[25] “Material UI - Overview,” Material UI, [Online]. Available: https://mui.com/material-ui/getting-started/. Accessed December 8, 2023.

[26] A. Naceri et al., “Tactile robotic telemedicine for safe remote diagnostics in times of Corona: System design, feasibility and usability study,” IEEE robotics and automation letters, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9454265/. Accessed on: October 25, 2023.

[27] "NGINX Documentation - Web Server," NGINX, [Online]. Available: https://docs.nginx.com/nginx/admin-guide/web-server/. Accessed on: October 24, 2023.

[28] “Nginx reverse proxy,” NgInx Docs, [Online]. Available: https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/. Accessed on: December 8, 2023.

[29] “What is Nginx?,” NgInx, [Online]. Available: https://www.nginx.com/resources/glossary/nginx/. Accessed on: December 8, 2023.

[30] "Your Health Plan: Connected and Convenient Care," Ontario.ca, [Online]. Available: https://www.ontario.ca/page/your-health-plan-connected-and-convenient-care. Accessed on: October 24, 2023.

[31] Fact sheet: Ontario’s doctor shortage - oma, https://www.oma.org/uploadedfiles/oma/media/public/hcp-factsheet-doctor-shortage.pdf. Accessed on: October 25, 2023.

[32] "Ontario Regulation 900/94 - PROFESSIONAL ENGINEERS ACT," Ontario.ca, [Online]. Available: https://www.ontario.ca/laws/regulation/900941. Accessed on: October 24, 2023.

[33] OpenCV documentation index, https://docs.opencv.org/. Accessed on: December 8, 2023.

[34] "Professional Engineers Ontario (PEO) Guideline," Professional Engineers Ontario, [Online]. Available: https://www.peo.on.ca/sites/default/files/2020-12/PEPGuideline_Nov2020.pdf. Accessed on: October 24, 2023.

[35] N. Raval, "React vs Angular: Which JS Framework to Pick for frontend Development?," Radixweb. Available: https://radixweb.com/blog/react-vs-angular. Accessed on: October 24, 2023.

[36] "What is Flask Python," What is Flask Python - Python Tutorial, https://pythonbasics.org/what-is-flask-python / (accessed Apr. 10, 2024).

[37] "Feature overview," React Router, [Online]. Available: https://reactrouter.com/en/main/start/overview. Accessed on: December 8, 2023.

[38] "REM in CSS: Understanding and using REM units," SitePoint, https://www.sitepoint.com/understanding-and-using-rem-units-in-css/ (accessed Apr. 10, 2024).

[39] "Reactive", Spring, [Online]. Available: https://spring.io/reactive. Accessed on: December 8, 2023.

[40] "Spring Framework - Reactive Web Framework," Spring Framework Documentation, [Online]. Available: https://docs.spring.io/spring-framework/reference/web/webflux/new-framework.html. Accessed on: October 24, 2023.

[41] "Spring Framework - Reactive Web Applications," Spring Framework Documentation, [Online]. Available: https://docs.spring.io/Spring Boot/docs/current/reference/html/web.html#web.reactive. Accessed on: December 8, 2023.

[42] "Stack Overflow Developer Survey 2023 - Most Popular Technologies," Stack Overflow, [Online]. Available: https://survey.stackoverflow.co/2023/#technology-most-popular-technologies. Accessed on: October 24, 2023.

[43] “Intelligent control LED integrated light source,” Worldsemi, [Online]. Available: https://cdn-shop.adafruit.com/datasheets/WS2812B.pdf. Accessed on: December 8, 2023.

[44] R. Van Loon et al., Laboratory Health and Safety Manual, https://sce-soft-web-ist.sce.carleton.ca/wp-content/uploads/2023/07/health-and-safety.pdf.

[45] "DOFBOT-Pi - Yahboom Robotics," Yahboom, [Online]. Available: https://category.yahboom.net/collections/rp-robotics/products/dofbot-pi. Accessed on: October 24, 2023.

[46] M. Zhang, “Top 10 cloud service providers globally in 2023,” Dgtl Infra, https://dgtlinfra.com/top-cloud-service-providers/#:~:text=Amazon%20Web%20Services20(AWS)%2C%20the%20cloud%20computing%20service%20of,compute%2C%20storag%2C%20and%20database. Accessed on October 25, 2023.

[47] “Zoom meeting SDK,” zoomvideocommunications, https://developers.zoom.us/docs/meeting-sdk/ (accessed Apr. 10, 2024).

Appendix A: Glossary

Angular: Typescript-based, free, and open-source single-page web application framework based upon the Model-View-Controller design pattern.

Application Programming Interface (API): a set of rules and protocols that allows different software applications to communicate with each other.

Degrees of Freedom: the number of independent variables that define the possible motions of a mechanical system.

Docker: a set of platform as a service products that use operating system level virtualization to deliver software in packages called containers.

Document Object Model (DOM): a programming interface that provides a structural representation of HTML and XML documents, allowing scripts to dynamically access, modify, and update the content, structure, and style of web pages

Haar cascade:

LAN: acronym for Local Area Network. Refers to a collection of connections connected in one physical location.

LED: acronym for Light Emitting Diode. Refers to a semiconductor device that emits lights when an electric current passes through it

LTE: acronym for Long-Term Evolution. Refers to a standard for wireless broadband communication.

Maven: a build automation tool predominantly used for Java projects. It simplifies the building and management of Java-based applications.

OpenCV: stands for Open Source Computer Vision. It is an open-source computer vision and machine learning software library mainly to be used for real-time computer vision.

Persistence of Objects: the ability to save the state of an object and retrieve it at a later time.

Raspberry Pi: a single-board computer designed for various computing tasks, encompassing programming and hardware prototyping.

React: open-source JavaScript library, created by Meta, allowing for the efficient creation of interaction user interfaces.

REST API: acronym for Representational State Transfer Application Programming Interface. Refers to a standardized set of rules for facilitating communication between software applications over the internet.

Spring Boot: open-source Java-based framework that simplifies the development of production-grade, stand-alone, and web-based applications.

Telemedicine: describes an area of medicine in which medical care is not provided directly but over a more or less greater distance.

Telemonitoring: refers to using different types of technology to monitor patients at a distance.

Twitch: a video live streaming service that focuses on video game live streaming, including broadcasts of esports competitions, in addition to offering music broadcasts, creative content, and "in real life" streams.

Wi-Fi: a wireless networking technology that uses radio waves to provide high-speed internet access.

Zoom: Zoom is a communications platform that allows users to connect with video, audio, phone, and chat.

Appendix B: Frontend Designs

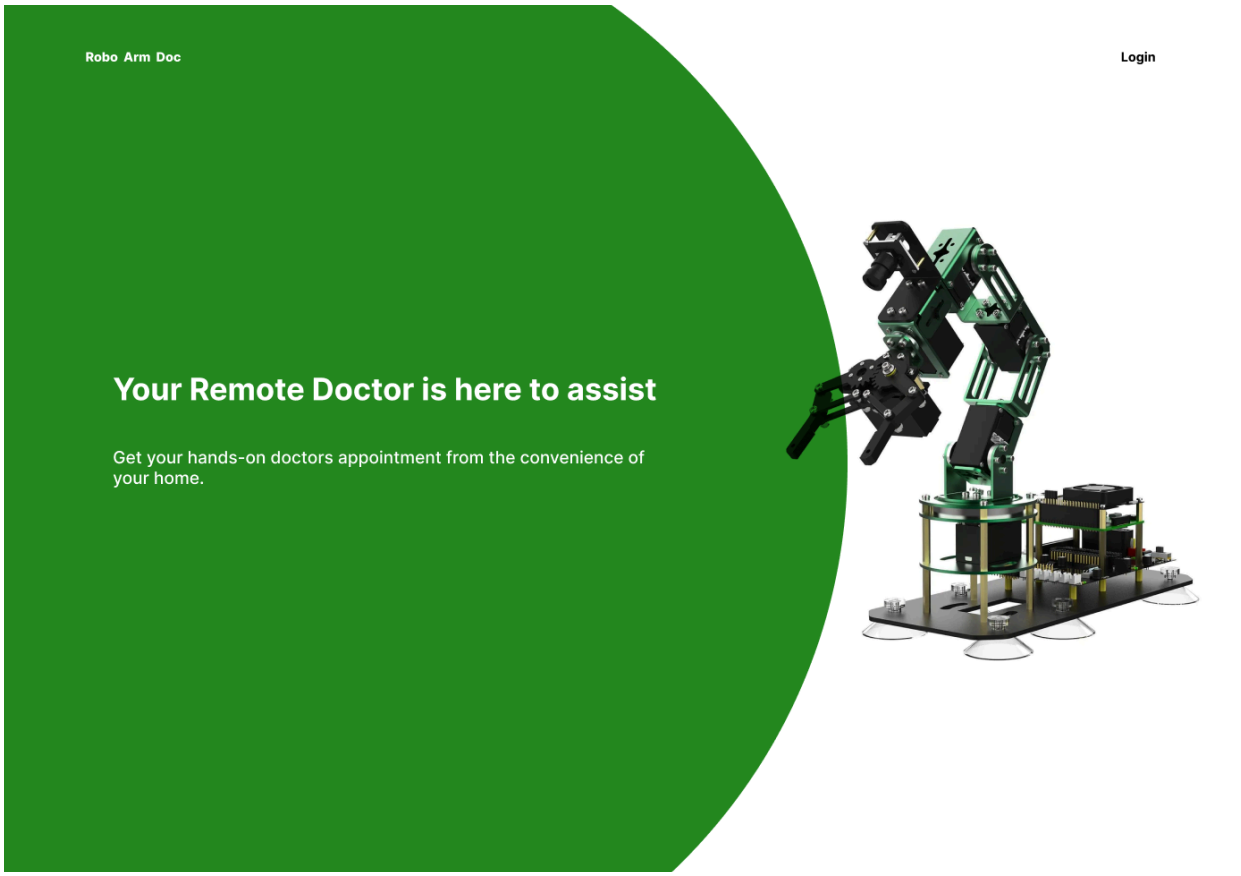


Figure 21: The Figma mockup for the website landing page.

Figure 22: The Figma mockup for the website sign-in portal.

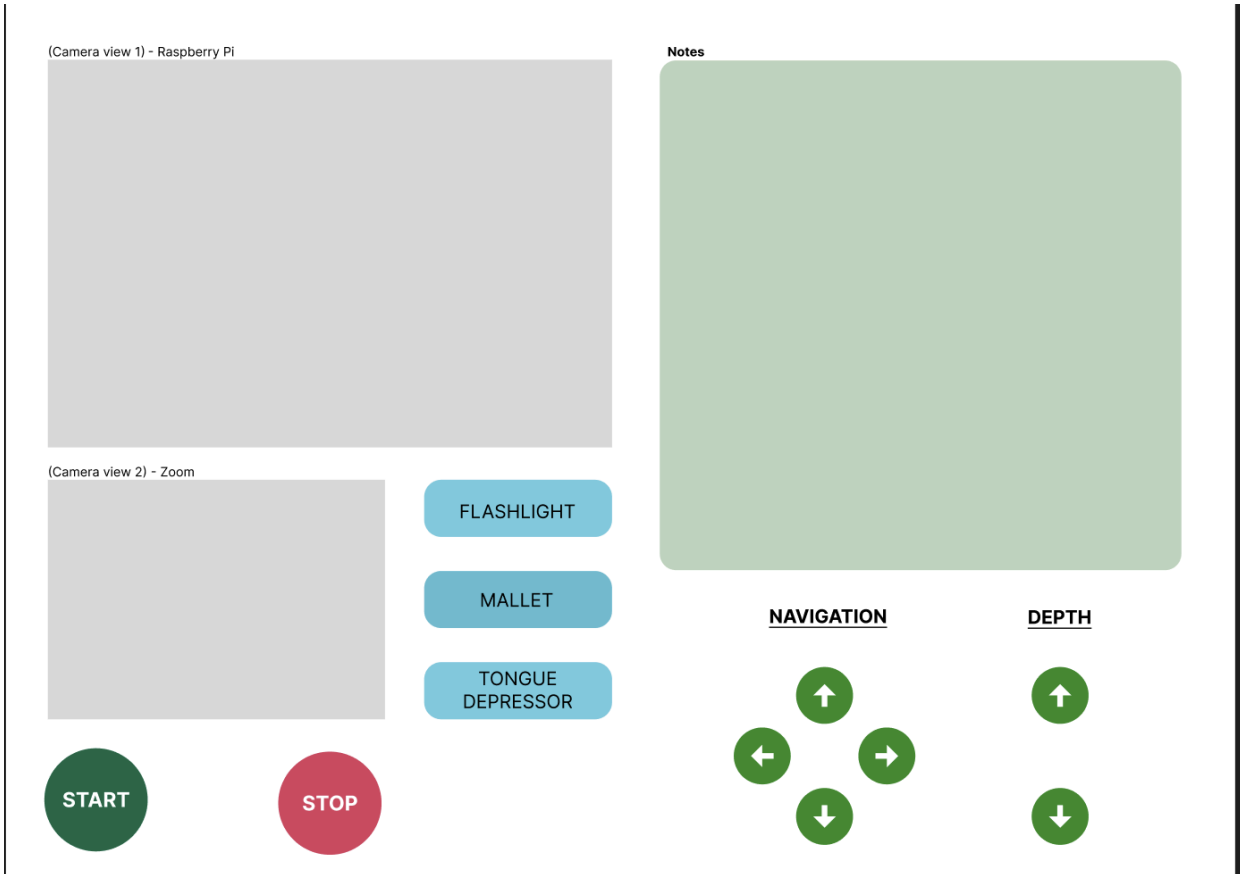


Figure 23: Doctor view of healthcare web application

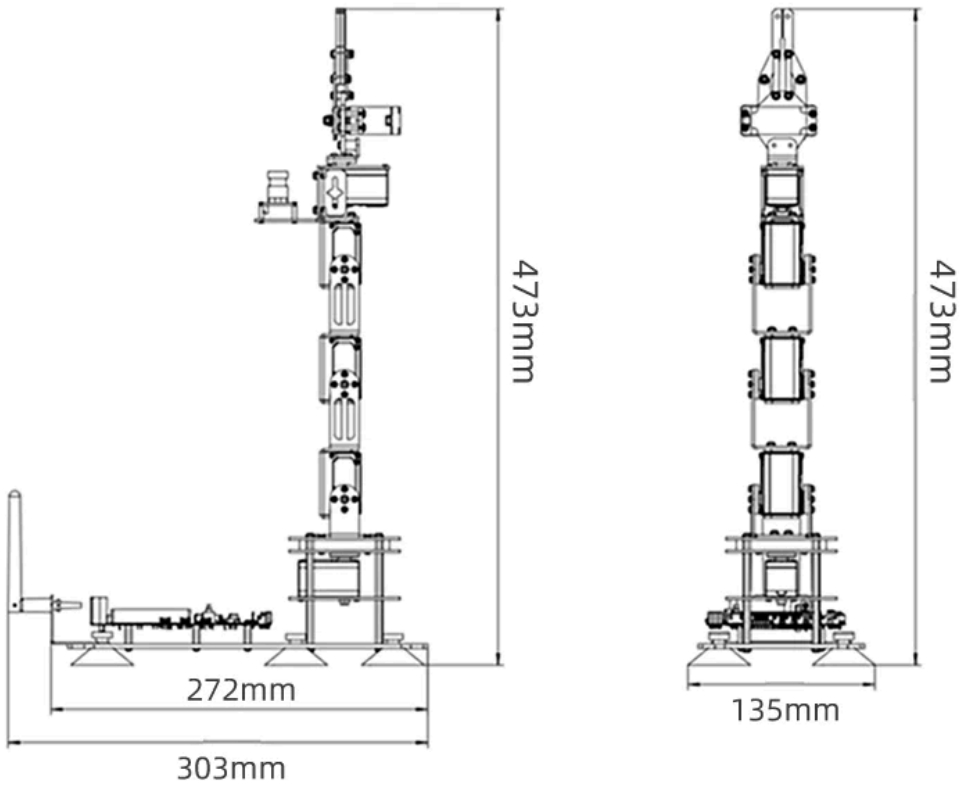


Figure 24: Robotic arm schematic (orthographic) [45]

Appendix C: Course Descriptions

Table 13: Course descriptions taken directly from the course outlines [10]

Courses	Course Description
ECOR 1051: Fundamentals of Engineering I	Software development as an engineering discipline, using a modern programming language. Tracing and visualization of program execution. Testing and debugging. Data management: digital representation of numbers; numerical algorithms; storing data in files; container data types: sequences, sets, maps.
SYSC 2004: Object-Oriented Software Development	Designing and implementing small-scale programs as communities of collaborating objects, using a dynamically-typed or statically-typed programming language. Fundamental concepts: classes, objects, encapsulation, information hiding, inheritance, polymorphism. Iterative, incremental development and test-driven development.
SYSC 3110: Software Development Project	Development of expertise in designing, implementing and testing maintainable, reusable software through team projects. Applying modern programming languages, design patterns, frameworks, UML and modern development processes (detection of olfactible source code defects, refactoring, iterative and incremental development, version control techniques) to medium-scale projects.
SYSC 3120: Software Requirements Engineering	Current techniques, notations, methods, processes and tools used in Requirements Engineering. Requirements elicitation, negotiation, modeling requirements, management, validation. Skills needed for Requirements Engineering and the many disciplines on which it draws. Requirements analysis: domain modeling, modeling object interactions; UML modeling. Introduction to software development processes.

SYSC 3303: Real-Time Concurrent Systems	Principles and practice of a systems engineering approach to the development of software for real-time, concurrent, distributed systems. Designing to achieve concurrency, performance, and robustness, using visual notations. Converting designs into programs. Introduction to hard real-time systems. Team project.
SYSC 3310: Introduction to Real-Time Systems	Principles of event-driven systems. Microcontroller organization. Development of embedded applications. Programming external interfaces, programmable timer. Input/output methods: polling, interrupts. Real-time issues: concurrency, mutual exclusion, buffering. Introduction to concurrent processes.
SYSC 4120: Software Architecture and Design	Introduction and importance of software architectures and software system design in software engineering. Current techniques, modeling notations, methods, processes and tools used in software architecture and system design. Software architectures, architectural patterns, design patterns, software qualities, software reuse.
SYSC 4805: Computer Systems Design Lab	Developing professional-level expertise in selected, important areas of the field by applying, honing, integrating, and extending previously acquired knowledge in team projects in the laboratory. Lecture periods are devoted to new knowledge required for the selected areas, to project-related issues, and to student presentations.
SYSC 4806: Software Engineering Lab	Applying the full spectrum of engineering and programming knowledge acquired in the program through team projects in the laboratory. Practice in doing presentations and reviews. Lectures will discuss software engineering issues as they relate to the projects, from a mature point of view.

Appendix D: Project Proposal

SYSC4907: Project Progress Report

Graphical Control Interface for a Robotic Arm

Group #110

Supervisor: Dr. Lynn Marshall

Dhriti Aravind (101141942)
Elisha Catherasoo (101148507)
Sarah Chow (101143033)
Evan Smedley (101148695)
Jeremy Trendoff (101160306)

December 8th, 2023

Abstract

Remote, understaffed, and underserved communities struggle to have adequate access to basic healthcare. The shortage of primary care physicians creates a challenge for individuals to access the necessary preventative care resources to detect symptoms of sickness. To solve this problem, our group proposes a robotic arm remotely controlled by a doctor that can serve patients in remote areas. The team shall create a website for doctors to remotely control the movements of the robotic arm placed next to a patient. This will allow doctors to perform virtual general physical health check-ups using common medical tools, such as a flashlight, a mallet, and a tongue depressor. A key to the success of this project is the user-friendly website interface, providing doctors with an easy and efficient way to control the robotic arm remotely. This website was created with the frontend framework, React, due to the wide assortment of plugins and open-source libraries available online. The website only allows certified doctors, with a valid CPSO number, access to use the robotic arm. As for the website's backend, it was created using the Spring-Boot Java Framework. Hosted in Google Cloud Run, the backend app server acts as a pipeline between the website and robotic arm. The doctor controls the robot through the buttons on the website. The instructions are sent to the backend server and forwarded to the arm to be executed. Finally, the robotic arm is a Yahboom robotic arm for Raspberry Pi 4B with a camera. This robotic arm has six degrees of freedom, which provides more points of manipulability and a higher dexterity for irregularly shaped objects such as a tongue depressor. Using the library provided by the manufacturer and OpenCV, all necessary actions for the doctor can be implemented.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

Remote, understaffed, and underserved communities struggle to have adequate access to basic healthcare. In Ontario, access to healthcare has been a challenge. Ontario's health minister, Sylvia Jones, stated in her plan for connected and convenient care that, "...the status quo is not working. Too many people are waiting too long to get an appointment or surgery, having to travel too far to get care, and spending too much time trying to navigate our healthcare system." [35]. According to the Ontario Medical Association, "at least 1 million Ontarians do not have regular access to primary care" and it is most prevalent in northern and rural areas [36]. The shortage of primary care physicians creates a challenge for individuals to access the necessary preventative care resources to detect symptoms of sickness. To solve this problem, our group proposes a robotic arm controlled by a doctor that can serve patients in remote areas.

1.2 Problem Motivation and Current State of the Art Solutions

The field of remotely controlled doctor-patient interaction is an emerging field and the current applications are built for specialized diagnoses [30]. For example, there exists a robot that primarily focuses on diagnosing breast cancer [30]. General-purpose robots have been developed in the telemedicine industry as well, including a robot called the Tactile Robotic Telemedicine system [30]. Some of the differences in the Tactile Robotic Telemedicine system in comparison to the robot described in the project include a robot that utilizes two identical robotic arms with seven degrees of freedom: one on the doctor's site and another on the patient's site. The two robotic arms can interact through LAN, Wi-Fi, or LTE networks. The doctor uses a joystick to interact with the robot on their side, which will then relay the movements to the robotic arm on the patient's side. Additionally, the robotic arm on the doctor's site can receive haptic force feedback to provide more information for the diagnosis. The doctor can view a 3D rendering of the robotic arm on the patient's site as it moves. Our solution aims to bring a more mobile approach allowing doctors the ability to connect from anywhere, not just from a dedicated workstation.

1.3 Proposed Solution

This project aims to create a website for doctors to remotely control the movements of a robotic arm placed next to a patient. This will allow doctors to perform virtual general physical health check-ups using common medical tools, such as a flashlight, a mallet, and a tongue depressor. This project provides a tool for remote and/or understaffed communities to receive general physical check-ups. The doctor will be able to control the robotic arm through a website interface and view the perspective of the arm through a Raspberry Pi camera. The system will send a Zoom meeting link to the patient in which they are expected to join the call from their own device and angle the device's camera in a third-person perspective. The doctor must log in to the system using their College of Physicians and Surgeons of Ontario (CPSO) number to access the website. The doctor will have two camera views: one from the mounted camera on the Raspberry Pi and another view of the Zoom meeting with the patient's device. The doctor will be able to take notes during the check-up session. At the end of the appointment, the system will send a text file of the notes to the doctor's email for them to review and send to the patient.

1.4 Project Overview

This project will be completed through a series of milestones and objectives. Throughout the project, our team will complete the objectives through these three main actions:

1. Obtaining a robotic arm that utilizes Raspberry Pi 4B and configuring it.
2. Creating a backend service to facilitate the communication from a web interface to the robotic arm. This service shall be hosted on an Amazon Web Services (AWS) cloud server.
3. Creating a frontend interface (website) for the doctor to control the robotic arm. This will be done using React.

1.5 Report Overview

This report provides an update on the current status of the project "Graphical Interface for a Robotic Arm". This project aims to create a website for doctors to remotely control the movements of a robotic arm placed next to a patient. The report includes technology methods used, tasks in progress, milestones achieved, and challenges faced. In section 2, we will describe our team and project management strategy to display our strategy to design this project solution. In section 3, we will describe our list of

requirements for this project. This shall act as a template for what our successful, feature complete, project will look like. Sections 5, 6, and 7 describe the design and progress of our frontend, backend, and robotic arm subsystems and how they will be connected together. Section 8 breaks down our updated timeline of tasks to further display our progress and plan for the future.

2 The Engineering Project

2.1 Health and Safety

All members of the project team are stringently following the manual provided by project administrators that outlines the minimum standards and practices for the safe and healthy operation of a laboratory [48]. The following of this manual ensures that all work meets the requirements of the Occupational Health and Safety Act of Ontario (OHSA).

All team members have reviewed Section 4: Responsibilities of Laboratory Workers (from the manual), and certify that we understand what our responsibilities are in the lab context. While the project room is used, all team members followed the General Health and Safety Principles and Basic Safety Procedures outlined in sections 5 and 6 of the manual.

Since a large portion of the project has been completed in a non-laboratory setting through the writing of software, in a variety of other environments (home, library, etc.), team members have not encountered such hazards. However, a common issue for software project teams is stress, which can have a huge impact on the completion of the project [10]. In order for all team members to help minimize stress and the chance of burnout, a set of stress health and safety protocols have been implemented. This protocol consists of the following: the team takes part in establishing clear and achievable goals, the team sets realistic timelines, the team communicates with each other regularly, team member's take breaks when feeling stressed, and the team prioritizes the most important tasks.

2.2 Engineering Professionalism

During this project all team members strive to demonstrate professionalism in accordance with what was learned during ECOR 4995 Professional Practice [11]. Some of the ways in which professionalism is demonstrated, as extracted from the Professional Engineering Practice Guideline, are listed below [39]:

1. As none of the project team members are professional engineers at this time, engineering titles (such as 'Engineer') outlined in section 7.3 Use of Titles of the referenced guideline, will not be used.

2. The Engineer's Duty to Report design processes and procedures that are unsafe will be followed by team members in all aspects of the project design and implementation.
3. Interactions among team members and between team members and stakeholders will be managed professionally.
4. Reports will be written not only as per the project guidelines but also in accordance with section 10.5 Report Writing of the referenced guideline (except for the part about the use of the professional engineer's seal because evidently we are not professional engineers yet).
5. Meeting notes and all project documents will be retained for referencing later on.
6. The Code of Ethics as outlined in section 77 of the Professional Engineers Act will be followed by all team members at all times [37].
7. All members of the team will strive to not engage in professional misconduct as per section 27 of the Professional Engineers Act [37].

2.3 Justification of Suitability for Degree Program

The project team is composed of the five following members. Note that course descriptions can be found under their respective titles in Appendix B.

Dhriti Aravind

Dhriti Aravind is a fourth-year student in computer systems engineering. In her program, she learned the fundamentals of programming (ECOR 1051, SYSC 2004, SYSC 3303, SYSC 4805), the applications of embedded programming, the basics of electronics, and how to analyze systems and simulate them. She has developed her interest in hardware and software through her internships at Ciena, Huawei, Microsoft, and Tesla where she worked in various areas of silicon development and tool support. Through her internship experiences, she familiarized herself with Python, C, Verilog, and System Verilog and participated in Agile scrums and a fast-paced environment. In addition, during her free time, continues to pursue her passion in robotics through various personal projects to automate mundane tasks; while undertaking these projects, she has refined her skills in 3D modeling and programming microcontrollers. In this project, Dhriti focuses on the design and programming of the robotic arm, designing the tray containing the medical equipment, and integrating the robotic arm with the website interface.

Elisha Catherasoo

Elisha Catherasoo is a fourth-year student in software engineering. She has gained experience throughout her education at university (ECOR 1051, SYSC 2004, SYSC 3110, SYSC 3310, and SYSC 4806),

personal projects, and her internships at Ericsson and Amazon. At university, she learned the basics of programming, such as OOP, software architecture, and real-time systems. Learning real-time systems has taught her how to use a Raspberry Pi and basic coding projects. This experience will help when programming the Raspberry Pi to control the movement of the robotic arm and developing the presets for the robotic arm picking up specific objects (mallet, tongue depressor, and flashlight). When making the database for the authenticated doctors, she can use what she learned in SYSC 3110 and SYSC 4806 to be able to persist the authenticated doctor data. Her internship at Ericsson gave her experience with working collaboratively with her team, which includes people of varying strengths and weaknesses, and being able to know who is best to do what part to better the given project(s). Her experience with Amazon gave her exposure to AWS, which will help when deploying the robotic arm web application to be accessed anywhere. Elisha's experience in university and her internships has given her the valuable skills needed when working in a team, in the basics of programming, and in the technologies needed to be able to successfully complete the project.

Sarah Chow

Sarah Chow is a fourth-year student in software engineering. She has taken several programming courses throughout her undergraduate degree pertaining to software development including software development (SYSC 3110), software architecture (SYSC 4120), and object-oriented programming (SYSC 2004). Throughout her university degree, she has had several co-op placements, including Warner Bros. Discovery and the Royal Bank of Canada. At Warner Bros. Discovery, Sarah worked as an iOS and tvOS developer and implemented frontend features for the company. At the Royal Bank of Canada, Sarah was a frontend and backend developer over the eight-month co-op. She became knowledgeable in web development with Angular and Spring Boot and also participated in an Agile team with daily stand-ups, bi-weekly sprints, and monthly retrospectives using JIRA and Azure DevOps. In 2022, Sarah attended the hackathon, Hack the North, in which she created the frontend using React and open-source libraries to build a website. Sarah has an interest in developing her frontend skills.

Evan Smedley

Evan Smedley is a fourth-year student in software engineering. During his degree, he has gained knowledge of object-oriented software development (SYSC2004, SYSC3110, SYSC4806), software architecture (SYSC3120, SYSC4120), and the development of real-time systems (SYSC3310, SYSC3303). Over approximately the past two years, Evan worked full-time as a Software Engineer Intern at Motorola Solutions Inc. During his time at Motorola, Evan gained experience working with all levels of the software stack. In the infrastructure area of the software stack, Evan learned a lot about

continuous integration, highly available systems, and cloud deployment of services with AWS and Microsoft Azure. This will be useful when deploying both the frontend and backend of our web application. In the backend of the stack, Evan gained experience with the use of Spring-Boot to interact with message brokers, persist data and pipeline requests in a reactive manner. This will help with the writing of the backend of our web application. During the internship, Evan experienced development in an Agile Kanban and Agile Scrum environment. This experience will come in handy with working as a team to follow a software development cycle.

Jeremy Trendoff

Over the past eight years, Jeremy Trendoff has studied and practiced the fundamentals of software engineering and development. Over the last four years as a software engineering student at Carleton University, Jeremy has taken classes in embedded and real-time systems (SYSC3310, SYSC3303), object-oriented programming (SYSC3110, SYSC4806), software architecture and documentation (SYSC3120, SYSC4120). This knowledge will relate directly to both the development of the interface, as well as the software portion of the arm. On top of this, Jeremy enrolled in a year-long co-op term at SOTI Inc., a leader in Enterprise Mobility Management and the Internet of Things. There, Jeremy worked as a fullstack developer, helping to solve frontend, backend, middleware, architecture, and research problems. At SOTI, Jeremy also learned the basics of professional practice for software development. Jeremy gained experience with tools like Git, JIRA, and the Agile development lifecycle, but, most importantly, learned how to develop software as a team. This knowledge is sure to prove essential during this, and any, professional project.

The Team

As outlined above, each member possesses valuable skills and experience. Collectively, the team's overall skills encompass the requirements to develop this robotic arm project. To tackle the software portion, our four software engineering majors have experience in user interface design and programming, software architecture principles, object-oriented programming, and have studied embedded and real-time systems programming to competently handle programming on the hardware side of things. Our computer systems engineering major has not only learned the fundamentals of programming and software design but has built extensive knowledge of the basics of electronics and real-time systems. Combining our strengths, our team has the knowledge and passion to complete this project.

2.4 Project Management

2.4.1 Team Structure

Table 1: Project group sub-teams and their contributors.

Team	**Primary Contributor (Specialist)**	**Second Contributor (Floater)**
Robotic Arm	Dhriti	Elisha
Frontend	Sarah	Jeremy
Backend	Evan	Jeremy, Elisha

Our team decided to structure ourselves using a non-hierarchical organization. This means we operate as one cohesive, democratic group. Instead of having team leaders, we are all equally responsible for driving the project forward. The team operates through three main subteams: the frontend team, backend team, and the robotic arm team. These teams are cross-functional, but the main role divisions have been outlined. While everyone on each team is expected to contribute to the architecture and development, we have structured ourselves into specialist and floater roles. Specialists (primary contributors) mainly focus on their team's work throughout the project. Floaters (secondary contributors) split their time according to the current demands of the project. On the frontend team, Sarah Chow is our specialist and Jeremy Trendoff acts as a frontend floater. On the backend team, Evan Smedley is our specialist. Jeremy Trendoff and Elisha Catherasoo are backend floaters. Finally, on the robotic arm team, Dhriti Aravind is our specialist and Elisha Catherasoo acts as the arm floater. This structure may be subject to change but at the moment, it suits our team's strengths.

2.4.2 Software Development Lifecycle Methodology

In keeping with some team members' experience during co-op terms, we will be using Agile methodologies for this project. We will be following the 12 Agile principles outlined in the Agile manifesto [1]. The Agile methodology we felt would best suit our needs is Kanban [9]. Developed by Toyota in the 1940s to optimize their engineering process, Kanban has become one of the most popular frameworks used to implement both Agile and DevOps software development. Kanban uses a highly

visual Kanban board which provides full transparency of work. On the board, Kanban cards will move from the basic states ‘To Do’, ‘In Progress’, and ‘Done’. Once a work item has been completed, the next work item in the backlog will take that space on the board. So, as long as the backlog is up to date, there is no need for fixed-length sprints. This will also suit our non-hierarchical structure as all team members are treated with equal privilege with this scheme. Kanban is also great for teams of all sizes. While other Agile methodologies have more structure, Kanban allows for more flexibility across the board. This will allow our team to quickly solve problems and determine the best plans of action when we get stuck.

2.4.3 Changing from Azure DevOps to GitHub Issues

The team has now decided to move away from using Azure DevOps, and to start using GitHub Issues as our project is using GitHub for the version control. GitHub Issues enables users to track and manage tasks, bug fixes, and enhancements to the project. For every pull request, an Issue explaining what the feature being added should do, what bugs are being fixed and what is being tested, is attached. This allows for other team members checking the code to see exactly what the changes are supposed to do and check if the tests reach 100% code coverage. This allows for the pull requests to be easily and efficiently checked as there will not be any confusion on the need for the pull request. The team was exposed to using GitHub Issues in the class SYSC 4806, where the team started using Issues in their labs, and it was found that it worked seamlessly with their version control, and that it was convenient and efficient to have everything in one place.

2.4.4 Version Control

Git and GitHub will be used for version control for this project. GitHub is an online user interface to the tool, Git. This will allow proper reviewing of code using pull requests and be a useful tool for collaboration. There will be one repository with a folder for each component of the project.

It was decided that the team would use GitHub because:

- GitHub is the industry standard, as demonstrated by the large number of reputable software companies that use GitHub [22]
- Although there are alternatives to GitHub, such as Azure Git, it is less widely used and therefore has less readily available documentation to support development

2.5 Individual Contributions

This section carefully itemizes the individual contributions of each team member. Project contributions identify which components of work were done by each individual and report contributions list the author of each major section of this report.

2.5.1 Project Contributions

The following table shows the project contributions of each team member.

Table 2: Team member contributions to the project.

Member	**Project Contributions**
Jeremy Trendoff	- Aided in development of backend API. - Aided in design and problem solving related to rebounding force from mallet impact. - Currently working on establishing a connection from the frontend to the backend.
Elisha Catherasoo	- Found data for force required for the patella reaction test. - Aided in design and problem solving related to rebounding force from mallet impact.
Evan Smedley	- Researched and selected backend technologies. - Created the backend project skeleton. - Dockerized the backend, frontend and reverse proxy in order to create a developer environment. - Defined a basic backend API for connectivity testing. - Researched and created a proof of concept for app deployment in AWS (abandoned due to cost that did not fit with project budget). - Configured the reverse proxy to redirect traffic to the desired server. - Created a GitHub action for building and pushing containers containing app components to Docker hub.

	- Created a GitHub action for continuous deployment of the containerized frontend, backend and reverse proxy in Google Cloud Run.
Sarah Chow	- Created the frontend server using React. - Created the routing to the landing page, contact us page, and login page. - Completed the landing page. - Created the base shapes for the login page.
Dhriti Aravind	- Assembled the robot. - Wrote and tested directional functions in Python for robot movement. - Connected the robot to the internet. - Programmed the robot on the same network over IP.

2.5.2 Project Proposal Contributions

The following table shows the project proposal contributions of each team member.

Table 3: Team member contributions to the project proposal.

Member	**Proposal Contributions**
Jeremy Trendoff	Jeremy has contributed to the following sections:1.1, 1.2, 1.3, 2.3, 2.4, 2.5, 3, 4.1.2, 4.1.3, 4.3, Glossary, Appendix A, Appendix B
Elisha Catherasoo	Elisha has contributed to the following sections: 1.1, 1.2, 2.3, 2.4, 2.5, 3, 4.1.2, 5, References, Glossary, Appendix A
Evan Smedley	Evan has contributed to the following sections: 2.1, 2.2, 2.3, 2.5, 4.1.2, 4.1.3, 4.1.4, 4.1.5, 4.2, References
Sarah Chow	Sarah has contributed to the following sections: 1.1, 1.2, 1.3, 1.4, 2.3, 2.4.1, 2.5, 3, 4.1.1, 4.3, Appendix A
Dhriti Aravind	Dhriti has contributed to the following sections: 1.1, 1.2, 1.3, 2.3, 2.5, 3, 4.1.6, 4.3, 5, 6, 7, Appendix A

2.5.3 Project Progress Report Contributions

Table 4 shows the project progress report chapter contributions of each team member.

Table 4: Team member contributions to the project progress report.

Member	**Progress Report Contributions**
Jeremy Trendoff	Abstract, 1, 2, 3, 4, 5, 11, 12, References, Appendix A
Elisha Catherasoo	1, 2, 3, 8, 11, 12, Appendix A
Evan Smedley	2, 4, 5, 8, 9, 10, 11, 12
Sarah Chow	1.1 - 1.4, 5, 6, 7, 8, 11, 12, References
Dhriti Aravind	1, 2, 6, 7, 10, 11, 12

3 Requirements

3.1 Functional Requirements

The following are the functional requirements for the project:

Robot Movement and Connection

- The robot shall be able to move in 3 planes of motion (x, y, and z).
- The robot shall be connected to Wi-Fi and receive an IP address.
- The robot shall interface with the web interface using HTTP protocols.
- The robot shall be able to pick up items less than 500 grams.
- The robot shall be able to pick up items less than 20 cm in the smallest dimension.

Doctor-Patient Interaction

- The doctor and patient shall communicate over a video call throughout the session.
- The robot shall be ready for motion when the patient's physical start button is pressed.
- The robot shall stop when the patient's physical stop button is pressed.
- The robot shall stop when the doctor's virtual stop button is pressed.
- The doctor shall start the robot once the patient's physical start button has been pressed.

Web Interface Interaction

- The doctor shall log in through the login portal using their CPSO number and their password.
- The login portal shall redirect the doctor to the robot interface webpage.
- The doctor shall be able to see the robot's camera view on the web interface.
- The doctor shall be able to control the robot through a set of directional and depth control buttons on the web interface.
- The doctor shall be able to write and download notes from the notes section of the web interface.
- The doctor shall be emailed the notes from the notes section of the web interface after the patient session has concluded.
- The doctor shall be able to select between different examination tools through preprogrammed action buttons on the web interface.
- The doctor shall be able to control the flashlight using the flashlight preprogrammed action buttons on the web interface.

3.2 Non-Functional Requirements

The following are the non-functional requirements for the project:

Usability

- Twitch shall be used to stream the robot's camera view to the web interface.
- Zoom shall be used to facilitate doctor-patient video calls.
- The website should be compatible with major browsers, ie. Chrome, Firefox, and Safari.

Security

- The login portal shall be supported using CPSO's authentication service to validate doctor credentials.
- The login portal shall authenticate the doctor based on CPSO number and password.

Compliance

- The system shall handle and store any personal information of its end users in accordance with the Personal Information Protection and Electronic Documents Act 2000.

Performance

- The latency from the buttons on the doctor interface to the robotic arm movements shall be under 5 seconds.
- The delay on the video streamed from the camera on the robotic arm shall be under 5 seconds.

Maintainability

- Code should be well documented and be easily understood by new developers.
- Project source code should be managed in a version control system (ie. GitHub).

Robustness

- In the event of a system or session failure, the system shall gracefully terminate all sessions.
- In the event of a camera failure, the system shall deactivate the robotic arm immediately.

4 Implementation Diagrams

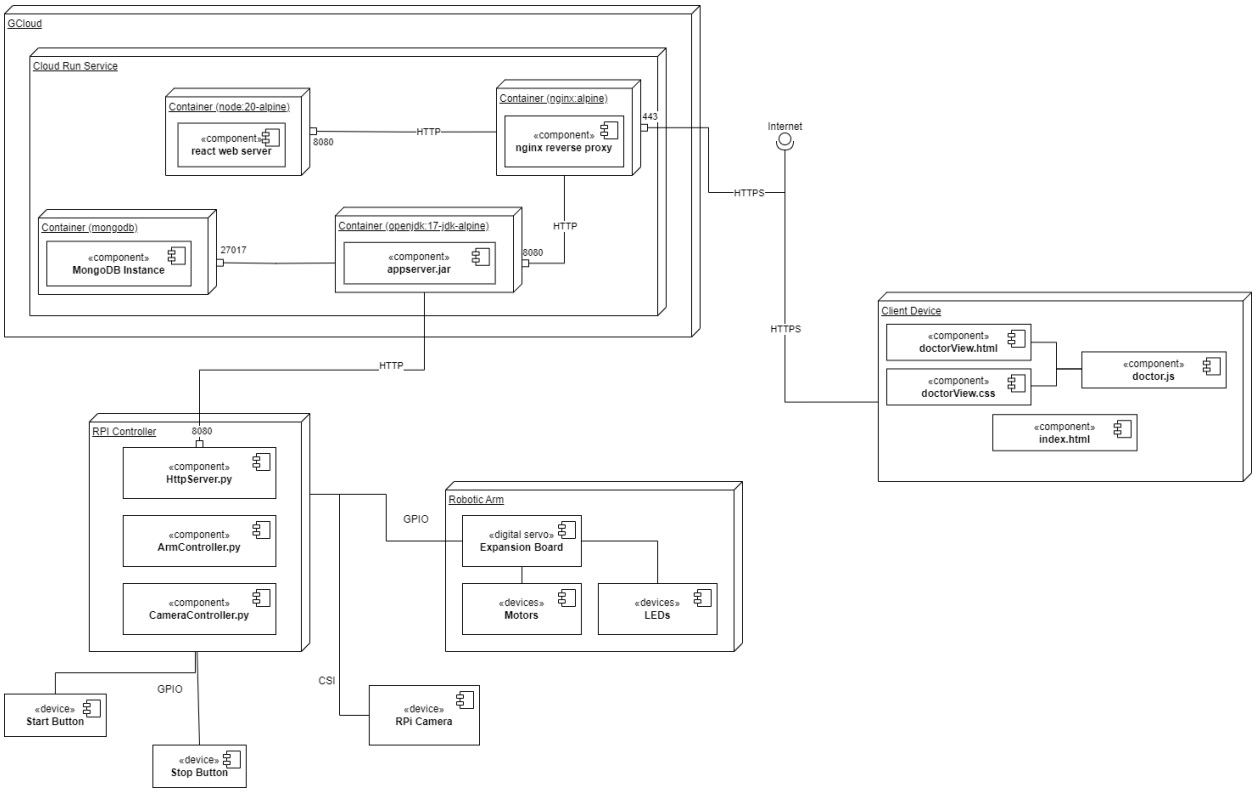


Figure 1: A deployment diagram visualizing the connection of our different components in the real world.

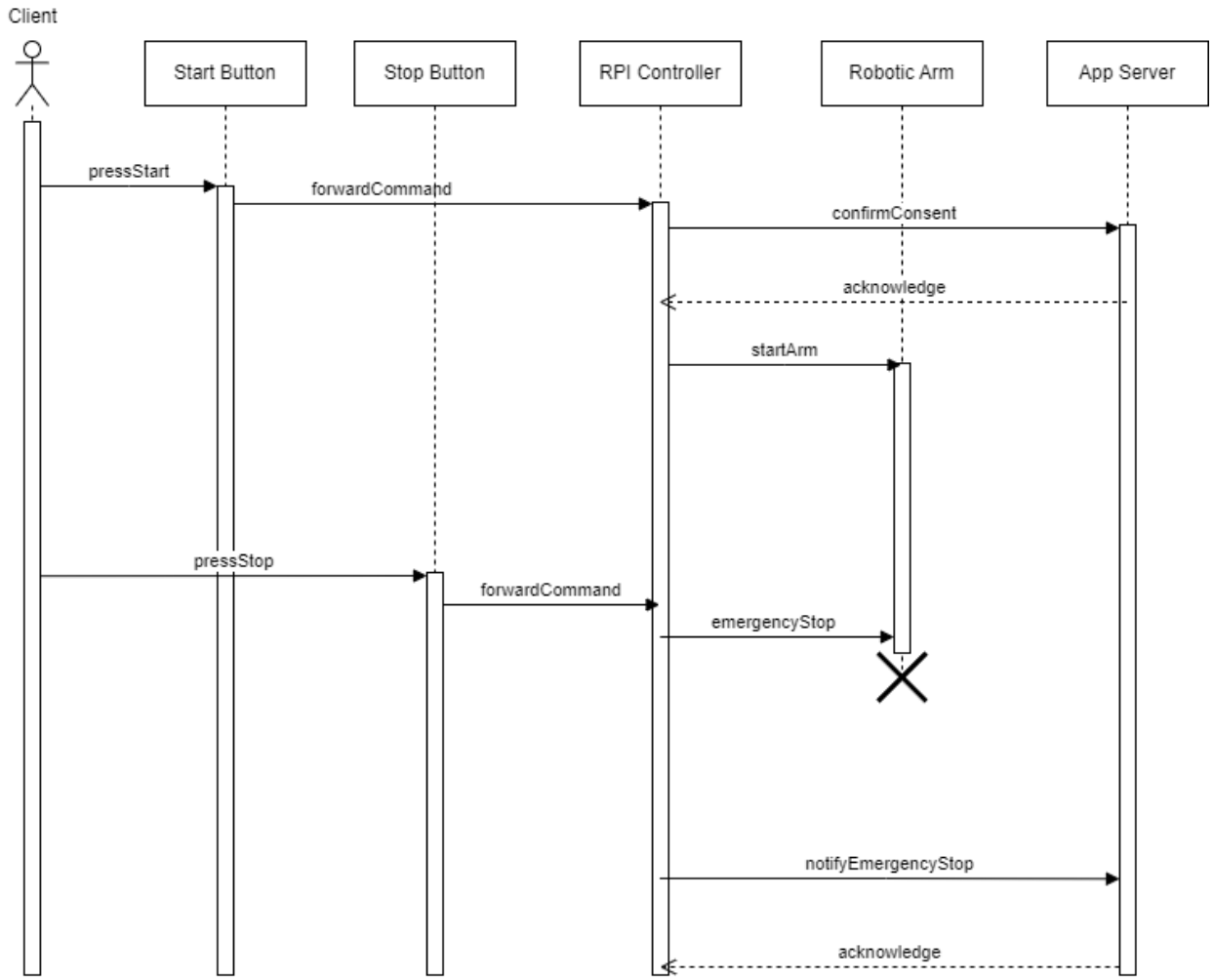


Figure 2: A sequence diagram visualizing the use cases of the patient (client).

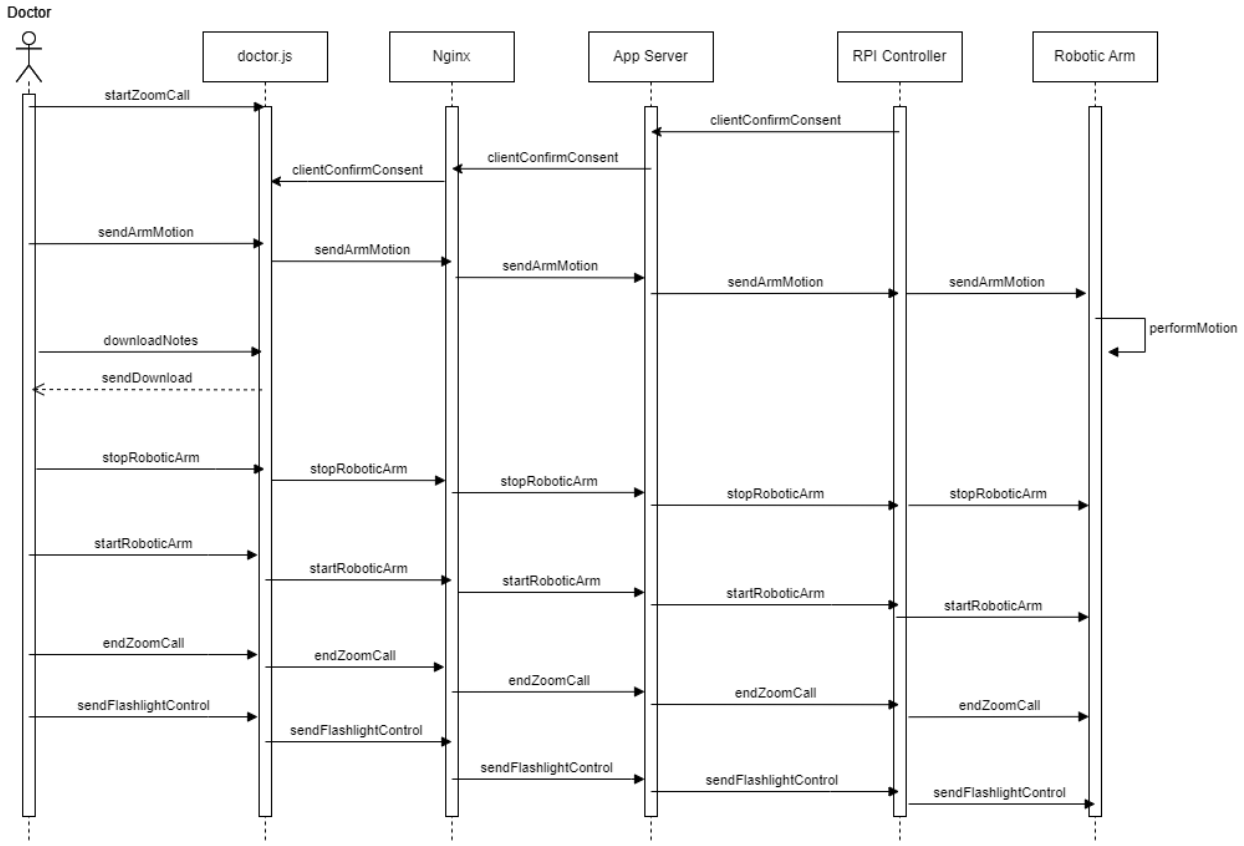


Figure 3: A sequence diagram visualizing the use cases of the doctor.

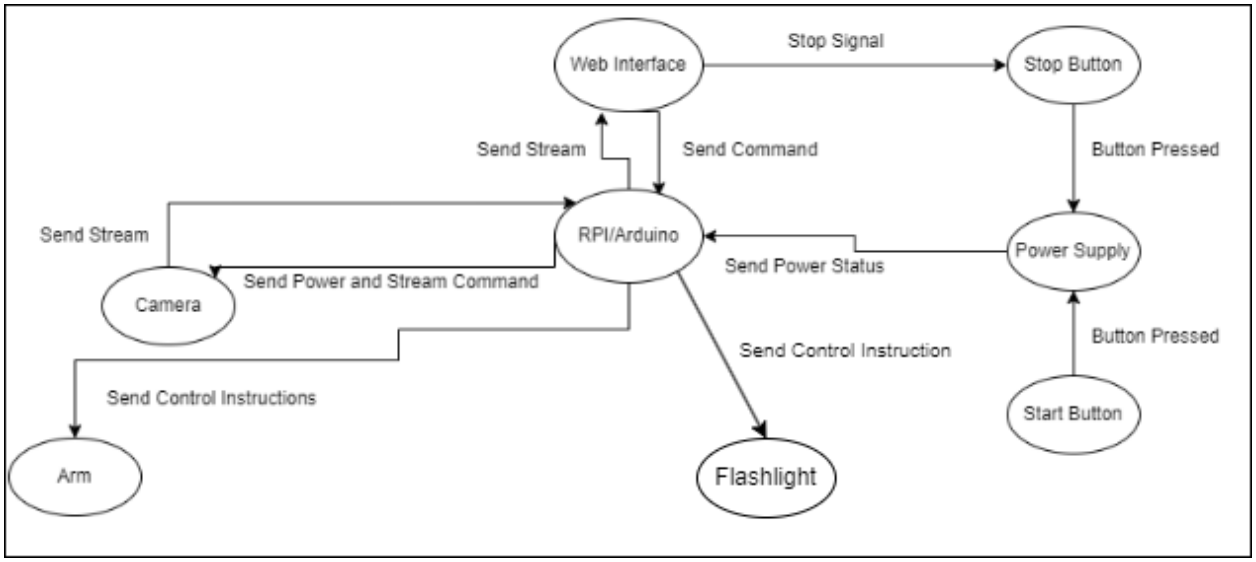


Figure 4: A high-level communication diagram of the robotic arm system interfacing with the web interface.

5 Backend

The main jobs allocated to the backend are facilitating doctor login from the frontend and sending instructions from the frontend to the robotic arm endpoint. This will be accomplished by building the backend API, which is a set of methods to facilitate this communication.

5.1 Background

5.1.1 Spring Boot Application

The backend service for this project is written in Java and is using the Spring Boot framework. Spring Boot is a Java application framework, which is used to implement the backend service for this project. Maven is used as a build tool to package our application and to import the dependencies that allow the use of the Spring Boot framework. Maven has a lifecycle which consists of many phases, such as *test* and *package* which produces an executable JAR file that runs our application. Running the JAR file will boot up an application server that listens for http requests from the frontend.

Spring Boot was selected to be used for the backend of this project because:

- Four team members (Sarah, Elisha, Jeremy and Evan) have knowledge of Spring Boot from SYSC4806.
- Spring Boot provides many useful dependencies for database interaction, simple creation of a REST API and reactive processing (more efficient than a regular event-driven model).
- NodeJS, Django, Ruby on Rails and Go are all alternatives to Spring Boot, however team members do not have any experience with these alternatives at this point so Spring Boot is the clear choice.

Spring Boot Webflux Framework is a modern reactive web framework [45]. Reactive processing is ideal for low-latency, high-throughput workloads, which is required for this project to support a large number of HTTP requests as a user controls the robotic arm [43]. If there are multiple robotic arms and users controlling them through the same app server, the added efficiency of the reactive approach would reduce cost by increasing the number of users that a single app server with fixed resources can serve. Multiple users' support is outside of the scope of this project due to its complexity. In considering future implementations of the project, obtaining more than one robotic arm would be an important requirement.

5.1.2 MongoDB Database

A database is required in order to provide a way for doctors to login. In a full production version of this application, the system would have access to CPSO's internal database of doctors for authentication. However, since the team does not have access, doctor identification data will be stored in a database that is managed by the team.

Initially the team had decided to use a NoSQL database for the following reasons:

- Interaction with a NoSQL database is simpler than interaction with a relational database.
- The database is only needed in this project for storage of user account information and login credentials which is not a super frequent operation and involves a small amount of data. So in comparing a NoSQL database to a relational database, the simplicity of NoSQL is found to outweigh the slight reduced performance of a NoSQL database.

MongoDB was selected as the NoSQL database for the following reasons:

- Reactive Spring Boot has better integration with MongoDB than with relational databases.
- Team members Evan and Sarah have experience using it.
- The project was to be deployed in AWS cloud which provides Amazon DynamoDB, a fully managed NoSQL database offering, as part of the AWS free tier.

5.1.3 Reverse Proxy

The need for a reverse proxy was not outlined in the project proposal as it is specific to the deployment method. A reverse proxy server is an intermediary server that contains configuration for forwarding requests from clients to desired destinations [33]. A reverse proxy acts as a load balancer for systems with more than one instance of a server, is used to perform SSL encryption to take load off of the servers they are positioned in front of, and provide additional security [33]. Cloud providers such as GCloud and Amazon Web Services provide reverse proxy functionality that is entirely hidden behind a web interface. The need for a reverse proxy has become clearer as the project progressed and details on the purpose and use of this reverse proxy is outlined in sections 5.2.3 and 5.3.3.

5.1.4 Containerization

Containerization refers to the packaging of software with its dependencies, and a lightweight operating system to create a portable environment to run the software. Docker is a containerization tool that will be used for this project. The reasoning for the selection of docker can be seen below.

It was decided the team would use Docker for the backend service because:

- It simplifies the deployment of the service to the cloud.
- It ensures that the environment in which our backend service is running is uniform, which reduces the amount of troubleshooting needed since each member of the team may have a different development environment.

A backend artifact will be placed inside a Docker image [16] using Docker Build, a tool for assembling docker images, generally using a 'Dockerfile' containing configuration for the building of the container [14]. Docker Compose, a tool for defining and running multi-container Docker applications, will be used to streamline and automate the process [15]. With Docker Compose, a YAML file is used to configure the application's services and the networks [15]. All the services can be packaged into images, started and connected, based on the aforementioned configuration. [15].

A lightweight Linux base image such as Ubuntu Server LTS (22.04) [17] or Alpine Linux [13] was chosen to reduce the size and cost of cloud hosting.

It was decided that the team would use a lightweight Linux base image because:

- Lightweight Linux base images boot up faster, speeding up deployment time.
- Lightweight Linux base images use less storage, lowering the cost of operating a container registry to store versions.
- Although functionality can sometimes be limited by an image being more lightweight, this project does not require any functionality that cannot be provided by a lightweight Linux image.
- Alpine Linux was identified as an option as it is small, simple, and secure [2].
- Ubuntu Server was identified as another option as Ubuntu is the most widely used Linux distribution among professional developers [46].

The way in which the application is packaged using docker has become more elaborate. These changes are outlined in section 5.2.5.

5.1.5 Cloud Deployment

Instead of hosting backend services on personal computers, a cloud service will be leveraged which provides access to a remote server. Using cloud services managed by third-party vendors ensures the web interface is always available and provides reliable performance. For this project, the initial plan was to use AWS. The cloud provider presence in multiple regions can be used to dramatically reduce the latency a user experiences through strategic selection of regions for deployment [6]. Due to cost restrictions of this project, the cloud deployment will be limited to a cloud free tier. Initially, Amazon EC2, an AWS service that allows the provision of a VM, was going to be used. Application components would all be deployed on one VM (or EC2 instance).

It was initially decided that the team would use AWS because:

- It was believed that the AWS free tier would provide everything needed for a proof of concept for this project [5].
- AWS is the leading cloud provider used by professional developers [51] .

The cloud provider used in this project is the portion of the project that has changed the most since the project proposal. Details on the developments and accomplishments in this aspect of the backend have been described in sections 5.2.4 and 5.3.4 respectively.

5.1.6 Continuous Integration

Continuous integration is defined as integrating code from multiple contributors to a single project [8]. In our project, there are five contributors to a single repository. It is important to have a continuous integration pipeline to ensure a working and symbiotic code collaboration process. Our team has decided that GitHub Actions [23] will be used for continuous integration. GitHub Actions is a tool to automate the building, testing, and deployment of a GitHub repository [25]. GitHub Actions is available on GitHub and runs infrastructure as code for free for all public projects, which will be leveraged to avoid unnecessary costs [26]. GitHub Actions will allow automatic packaging, building of a Docker image [16], and deployment of our backend and frontend.

It was decided that the team would use GitHub Actions because:

- GitHub Actions integrates very well with GitHub.
- GitHub Actions provides useful templates for the deployment of an application to AWS [24].

- GitHub Actions is very easy to use, which will make it so that a continuous integration system can be established quickly and will require little configuration.

5.2 Developments

5.2.1 Spring Boot Application

Developments that have occurred with respect to the Spring Boot application are related to database connectivity and user login.

Database connectivity was originally supposed to be implemented using Java Persistence API (JPA), a commonly used Spring Boot feature that allows the use of repositories with methods that define queries rather than embedded SQL. Upon further research into the use of JPA with Spring Boot Webflux (A Spring Boot dependency that enables use of reactive programming, see section 5.1.1 for more information), it was discovered that in order to interact with a database in a reactive manner, a different dependency should be used. This new dependency is called Spring Data Reactive Repositories. Fortunately Spring Data Repositories is well integrated with the already chosen database system, MongoDB, so MongoDB will still be used.

In regards to the user login, Spring Security is the clear choice as it provides a login mechanism and tools for session management.

In terms of the backend API, several request types have been defined based on the responsibilities of the backend. There are 3 categories of requests: authentication related requests, robotic arm registration requests, and robotic arm instruction requests, which are shown in tables 5, 6 and 7 respectively.

Table 5: Authentication related HTTP requests.

Request Name	Type	Description	Backend logic that occurs as a result
login	POST	“/login” endpoint accepts a post request with user credentials.	Using Spring Security, hash the password and compare it to the password stored in the database. Upon successful authentication, return a session id to the user so they can remain authenticated for

			a limited period of time.
logout	POST	“/logout” endpoint accepts a post request with user credentials.	Terminate user session and revoke access to directional requests.

Table 6: Robotic arm registration related HTTP requests.

Request Name	Type	Description	Backend logic that occurs as a result
selectRoboticArm	POST	“/selectRoboticArm” endpoint accepts a POST request allowing the current active user to select which robotic arm they want to use.	Set the robotic arm configuration identified in the POST request as the currently active robotic arm the user is using.
registerRoboticArm	POST	“/registerRoboticArm” endpoint accepts a post request with a new robotic arm configuration.	Store robotic arm configuration (connection details) in the database. Configuration is associated with the id of the currently active user.

Table 7: Robotic arm instruction related HTTP requests.

Request Name	Type	Description	Backend logic that occurs as a result
positiveX	POST	“/positivex” endpoint accepts a post request to instruct the robotic arm to move in the positive x direction.	Reformat and forward the request to the robotic arm that the user is configured to use.
negativeX	POST	“/negativex” endpoint accepts a post request to instruct the robotic arm to move in the negative x direction.	Reformat and forward the request to the robotic arm that the user is configured to use.
positiveY	POST	“/positivey” endpoint accepts a post request to instruct the	Reformat and forward the request to the robotic arm that the user is

		robotic arm to move in the positive y direction.	configured to use.
negativeY	POST	“/negativey” endpoint accepts a post request to instruct the robotic arm to move in the negative y direction.	Reformat and forward the request to the robotic arm that the user is configured to use.
positiveZ	POST	“/positivez” endpoint accepts a post request to instruct the robotic arm to move in the positive z direction.	Reformat and forward the request to the robotic arm that the user is configured to use.
negativeZ	POST	“/negativez” endpoint accepts a post request to instruct the robotic arm to move in the negative z direction.	Reformat and forward the request to the robotic arm that the user is configured to use.
tightenGrip	POST	“/tightenGrip” endpoint accepts a post request to instruct the robotic arm to tighten its grip.	Reformat and forward the request to the robotic arm that the user is configured to use.
loosenGrip	POST	“/loosenGrip” endpoint accepts a post request to instruct the robotic arm to loosen its grip.	Reformat and forward the request to the robotic arm that the user is configured to use.
getArmPosition	GET	“/getArmPosition” endpoint accepts a get request to retrieve robotic arm position.	Include the coordinates of the current position of the robotic arm in the response to this request.

5.2.2 MongoDB Database

The selected database is still a MongoDB relational database, this has not changed. However since the cloud provider has changed, the selected method of deployment for this database has been impacted.

The new method of deployment will not be utilizing a cloud managed database offering, but instead involves placing our own MongoDB container in the cloud. The container will be built with pre-configured credentials, allowing our backend to authenticate with it.

Use of the new deployment method makes the deployment more cloud agnostic, as it does not depend on a cloud provider's specific NoSQL offering, the container can be deployed using any cloud provider. Deployment of our own database will also allow the cloud environment to mimic the developer environment exactly, reducing the chance of failure during deployment due to environmental differences.

5.2.3 Reverse Proxy

Nginx is a high-performance, open source software for web serving, reverse proxying, caching, load balancing, and media streaming [34]. Since the project application has both a frontend and a backend, Nginx will be used as a reverse proxy to direct requests to either the application server or the web server [33]. Nginx can also be used to provide HTTPS support [32]. An Nginx docker container is used to ease the setup of the reverse proxy in both a developer environment and the cloud deployment. The following trade off analysis was performed when selecting Nginx as a reverse proxy:

It was decided that the team would use Nginx for the reverse proxy because:

- Of the two most common reverse proxies (Nginx and Apache), Nginx is the more lightweight of the two, consuming fewer resources due to its asynchronous, event-driver architecture
- Nginx is newer than Apache
- Nginx configuration is simpler and more readable than Apache
- Team member Evan has used Nginx before

5.2.4 Containerization

The use of containerization for the backend of the project has remained the same, however, now it is also used for the frontend, reverse proxy, and database. Docker is used to package each of these four components for use in both a local developer environment and in the cloud. Additionally, while a lightweight Linux base image is still used, official docker base images with dependencies already installed on them exist for Java, React, Nginx, and Mongodb, so these images will be leveraged to simplify the creation of docker images as these dependencies do not need to be installed on the images.

The four components, the base image used for each component and what is required to build the desired docker image is all described in table 8 below.

Table 8 : Docker Container Components

Component name	Base image	Additional steps to build image
Backend	‘openjdk:17-jdk-alpine’ [18]	• Copy the app JAR file to the image • Create the app entrypoint (‘java -jar <jarname>.jar’)
Frontend	‘node:20-alpine’ [21]	• Copy the app directory to the image • Install npm packages • Create the app entry point (‘npm start’)
Nginx	‘nginx:alpine’ [20]	• Create configuration file to specify where to forward requests • Configure SSL so that HTTPS can be used
MongoDB	‘mongodb’ [19]	• Configure mongo login credentials • Create a docker volume to ensure that data persists if the container is restarted

To facilitate container version control and prepare containers for deployment, it is common practice to push containers to a container registry. Docker has a container registry (Docker Hub), and so do many cloud providers (Amazon Elastic Container Registry, Cloud Artifact Registry). The selected container registry is Docker Hub.

It was decided that the team will use Docker Hub because:

- Docker Hub provides unlimited free public container repositories and one free private container repository, without requiring a credit card. Other cloud repositories only support private container repositories, require a credit card and charge for the amount of storage used.
- Docker Hub is well integrated with GitHub actions, with many reliable marketplace actions that can be used to build and push a docker image to Docker Hub.
- Docker Hub is cloud agnostic, meaning that if this app was all of a sudden to be hosted with a different cloud provider, the use of Docker Hub would not have to change.

5.2.5 Cloud Deployment

The cloud provider used has completely changed for this project. This is a result of further research and a failed proof of concept.

Initially AWS EC2 was going to be used to deploy the application. AWS EC2 would provide a VM to deploy the application. However, after further research, the cloud service AWS Fargate was a more modern and low difficulty option. AWS Fargate is a managed container running service that allows the deployment of containers, pulled from a container registry. This simplifies the deployment process as otherwise a VM image would have to be created with everything necessary installed on it, making it very challenging to configure what was installed on the VM as a part of a continuous integration pipeline. Instead, during continuous integration, a docker image can be built with all necessary dependencies installed on it, the image can be stored in a container registry and then the container can be pulled to the managed container service.

After an extensive proof of concept with AWS Fargate, it was determined that this service could not be used within the bounds of AWS free tier and that it would cost too much to use. In an effort to still use this modern deployment method, research was performed to look into use of the Google Cloud managed container offering, Cloud Run. The Google Cloud free tier is structured differently, allowing for a certain amount of requests, memory usage and CPU usage with a Cloud Run service for free each month [27]. Additionally, after account creation, a $416 of free credit ($300 USD) is applied to your account, which would cover any additional costs incurred during deployment of this project until the end of April [28].

Google Cloud Run allows creation of multiple containers in one service, with one ingress container (the reverse proxy) and several sidecar containers that can be networked to work with the reverse proxy. This fits this project's use case perfectly and can be configured with continuous integration to mimic the local dev environment exactly. Figure 5 shows an example of how the app would be deployed.

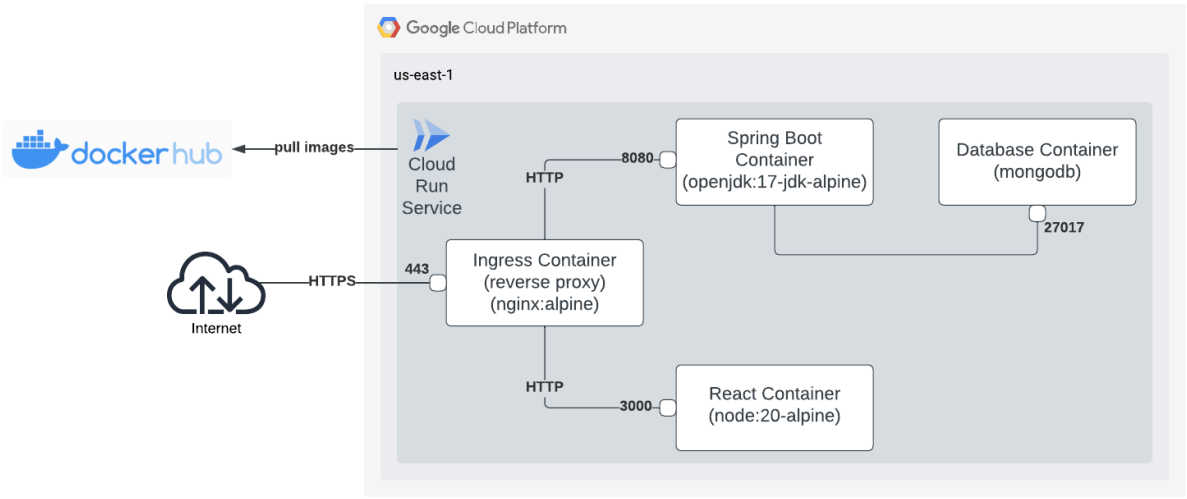


Figure 5: Example Docker Deployment Diagram

The reasoning for the selection of Google Cloud Run as the deployment platform for the Checkup Bot app is given above, but to summarize, Google Cloud Run was chosen because:

- Google Cloud offers a Free Tier that is superior to that of AWS in the short term, allowing Google Cloud Run to be used for free for the duration of this project.
- Google Cloud Run allows containerized deployment of multiple containers with networking between them in a simple way, which suits this project's use case very well.
- Google Cloud Run has easy-to-understand methods for deployment of new containers using continuous integration.

5.2.6 Continuous Integration

GitHub actions have been used successfully and will continue to be used as the continuous integration tool for this project, no developments have occurred with respect to how this is used in the project.

5.3 Accomplishments

The following are the accomplishments made with regards to the backend so far:

- The Spring-boot project skeleton with required dependencies was created.
- A reverse proxy was identified as a needed component.
- The reverse proxy was researched to select Nginx as the solution, and Nginx configuration was written to set up forwarding of requests to the frontend and backend services.

- Dockerfile and docker-compose.yml files were created to facilitate the running of the frontend, backend and reverse proxy services in a local environment.
- A continuous integration pipeline was used to build the Spring-boot project and publish a docker image of it to docker hub, from which it can be pulled during deployment.
- A lot of research into running an application on AWS using only free tier services was done, it was established that the original plan of using EC2 is better than the secondary plan of using ECS on EC2, as ECS on EC2 is not possible on AWS free tier EC2 instances.
- A proof of concept for deployment of the app on google cloud was created.
- A continuous integration pipeline to deploy the app in google cloud was created.
- A database was selected and research was performed to learn how to connect it with the Spring Boot backend.

5.4 Next Steps

Some next steps for the backend portion of the project includes the following:

- Add SSL support to the reverse proxy, to allow for HTTPS.
- Setup Spring Security to add user login functionality.
- Add a MongoDB image to the developer environment docker compose file and add the MongoDB container to the continuous integration pipeline for deployment.
- Create a MongoDB volume in Google Cloud Run to persist data even when the container is redeployed.
- Finalize and Implement login, robotic arm registration and robotic arm control methods in the Spring Boot backend.
- Establish the connection between the robotic arm and the backend through the control methods.

6 Frontend

6.1 Background and Terminology

React was chosen as the frontend framework because it offers numerous plugins and open-source libraries and utilizes virtual document object model (DOM) objects; the ability to reload a single DOM object instead of reloading the entire structure of the page when there is a visual change made to the website [41]. This feature requires less performance from the cloud server and will decrease the overall costs for the project [41].

The tools utilized thus far include react-router-dom and Material UI. Websites are usually required to communicate with the backend server to receive the necessary information to render a new page, including the CSS and Javascript assets and the HTML file. The react-router-dom [42] package allows for client-side routing which implies that the frontend does not need to contact the backend to route to a new page. This provides for faster web page loading time. Material UI [29] is an open-source React library that provides ready-for-use components for a website. All of the components from this library are customizable to suit specific user needs. In our project, Material UI [29] was used in the experimentation of implementing a navigation bar to the website. The use of these two tools allowed the skeleton of the app to be created.

Figma designs were created as a guideline for the frontend user experience. The website is based off of the Figma designs and this progress report focuses on the creation of the landing page as shown in figure 6.

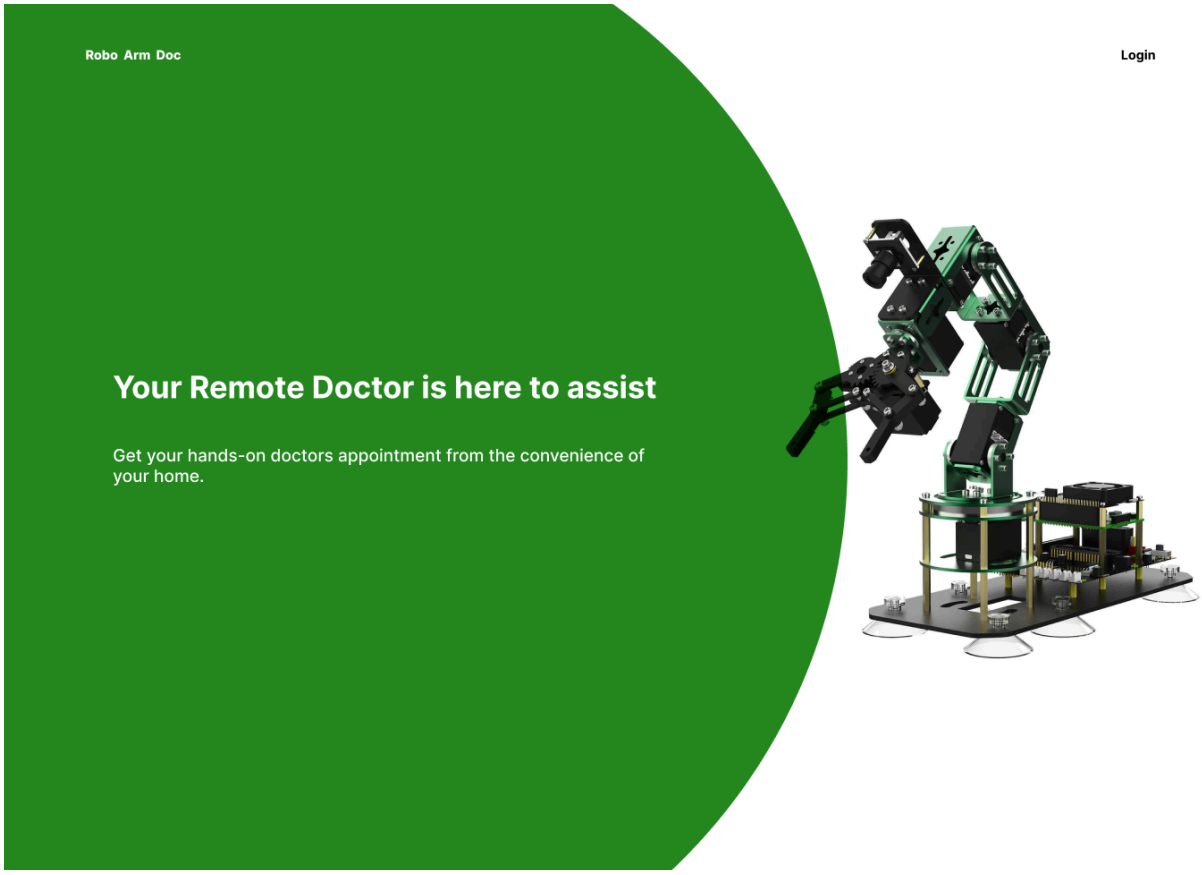


Figure 6: Figma design of the landing page

6.2 Developments

It was noted the size ratio of the Figma designs was incorrect and this impacted the exported images from the designs. This introduces the challenge where the ratio of the images and placements of the components on screen need to be repositioned and rescaled. This issue was fixed and the Figma designs currently have the proper aspect ratio.

The navigation bar in the Figma designs is seen overlapping the background of the landing page. During the development of the landing page, the navigation bar appeared to be pushing down the background image. The implementation of the navigation bar was done by creating a header component and then placing it on the home page. The result of this implementation is shown in figure 7.

Figure 7 : Navigation bar pushed down the background image

Upon further research, there was an alternate method of creating a navigation bar through utilizing a component called app bar in the Material UI library. Once implemented, the app bar provided a clearer user experience and allowed for an added profile capability as seen in the top right of figure 8. The default color of the component is blue, but upon further investigation it was not possible to place the app bar component over the home page. It was also noted that this color scheme was difficult to read and this impacts accessibility for users.

Figure 8 : App bar implemented from material-ui library

The current implementation of the landing page disregards third party components and places the hyperlinks directly on the home page. This method allows the links to be overlapped on top of the home page. This is still being developed as it would be beneficial to create a navigation bar component and apply it on all the pages on the website to reduce the amount of repeated code. The current implementation is shown in figure 9. Note there are two navigation bars present: the navigation bar with the white background and the navigation bar below it with the green text.

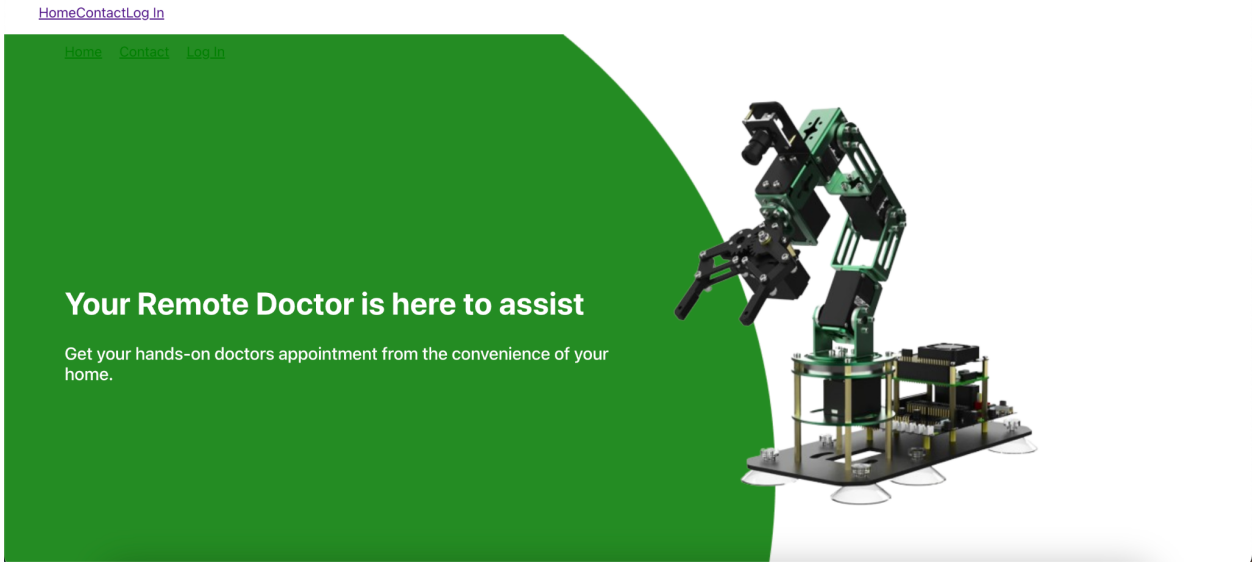


Figure 9 : Current implementation of the landing page

The login page is currently in progress, as shown in figure 10. Since the correction of the Figma aspect ratio, the shapes were exported to the website. The order and size of the shapes need to be adjusted and a form field will be to be implemented.

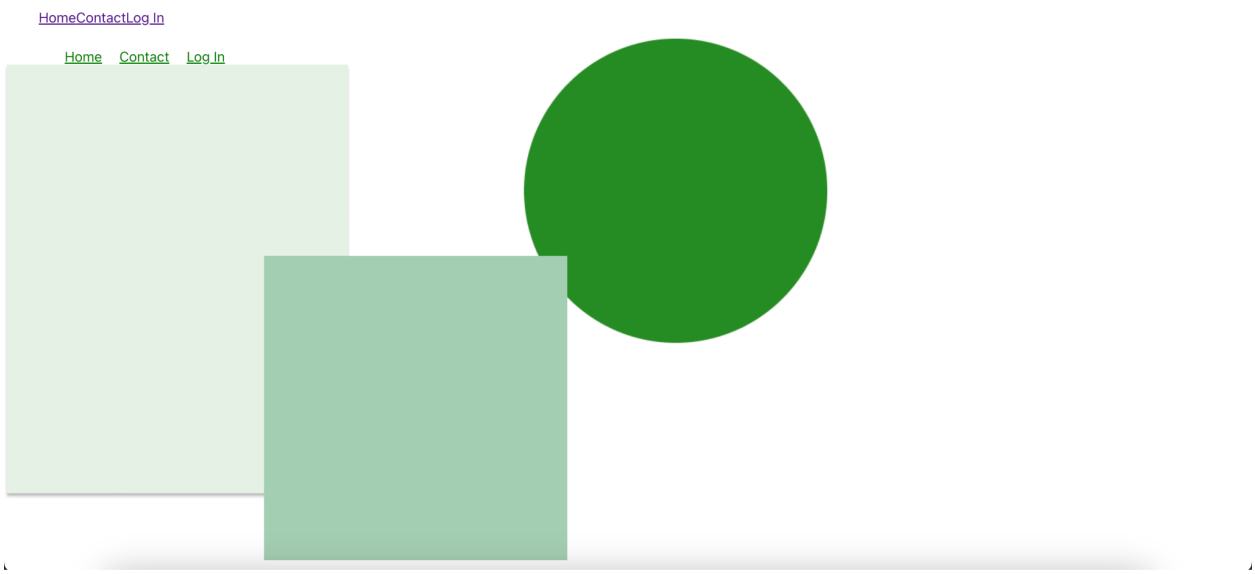


Figure 10: Current implementation of the log in page

6.3 Accomplishments

The website was created with the frontend framework, React. Utilizing the Figma designs, images were imported to the React server and displayed on the landing page. The landing page is completed, with the exception of the navigation bar, and the login page is in progress.

6.4 Next Steps

The next step in the frontend development is to achieve a transparent background for the navigation bar. This will be completed through deeper research into React routers. The login page will be completed, including a form field that will be forwarded to the backend to sign in a user. Lastly, the doctor interface will be built.

7 Robotic Arm

7.1 Background

The project will use a Yahboom robotic arm for Raspberry Pi 4B with a camera, to be purchased from Amazon [50]. This robotic arm will be controlled by the included Raspberry Pi using Python.

It was decided that the team would use the Yahboom robotic arm [49] for Raspberry Pi 4B as it was the only robot which met the desired specifications:

- Has a 6 degree of freedom in the robotic arm, which provides more points of manipulability and a higher dexterity for irregularly shaped objects such as tongue depressor.
- Has a camera mounted on the robot.
- Uses a Raspberry Pi 4B which was already accessible via the department, so we did not need to buy extra parts.
- Is small enough to be non-intimidating to the end user.
- Can pick up a load of up to five hundred grams, which is more than enough for our purposes.
- Supports Wi-Fi connectivity out of the box.
- Costs under 500 dollars.

7.2 Developments

The three main objects the robot needs to pick up, as outlined in the project proposal, is the mallet, the popsicle stick, and the flashlight. The flashlight needs to be turned on prior to usage. Initially, to implement this task, either the user would be expected to turn it on, or a holder would be made to stabilize the flashlight when the robot pushes the button. In order to simplify the mechanism and reduce dependency on the patient, WS2812B [47] LEDs mounted on the fifth link with a 3D printed case attached with a clamp will serve as an integrated flashlight. The LEDs will be controlled through the Raspberry Pi's GPIO pins. When the command to turn on the flashlight is sent by the doctor, all the LEDs will turn on. All LEDs will emit the RGB color (255, 255, 255), otherwise known as white.

Unlike the tongue depressor (popsicle stick), the mallet's center of gravity is not at the midpoint of the object. Objects with a lopsided center of gravity will slip out of the robot's claw if the tool is grasped too far from the object center point, and will require more force to keep it stable. Therefore we need to implement computer vision software to identify the mallet's position and its center of gravity. The density

of the mallet's head and the popsicle sticks are similar. Thus, OpenCV [38] will be used to identify the centroid of the shape from the camera's capture of the object.

7.3 Accomplishments

The Robotic Arm has been assembled by using the instructions provided, connected to the WiFi network and has been able to pick up objects from a flat surface. The Raspberry Pi has been configured to use a static IP address so the backend interface will have an easier time addressing the robot. The library provided by the manufacturer, Yahboom, was used to implement functions to direct the robot in 3 axis actions by the robot. It uses a combination of those actions and OpenCV [38] to find the contours of the objects to pick them up.

7.4 Next Steps

The next step for the robot arm is to design a solution so the robot can test a patient's reflexes. The arm team has identified a list of tasks to complete this design. They are as follows:

- Find a dataset or pretrained model to find the ulnar nerve. If such a data set is not available, create a dataset by recording a video and separating them into frames, then draw bounding boxes around the target area. Use OpenCV [38] to translate those bounding boxes on the joint (to assist the doctor hit the correct point).
- Find the force to swing the mallet at the ulnar nerve.
- Use OpenCV [38] to find the correct distance away from the joint to hit the ulnar nerve properly.
- Understand the effects of the rebounding force generated from the mallet colliding with the ulnar nerve and how it can affect the system. This includes displacement when contact occurs with any of the objects causing a shift in grip positioning.
- Complete the connection between the Raspberry Pi and the backend.

8 Timeline

The timeline of the project outlines the milestones, tasks, and approaches for the stages required for this project. The stages include problem and solution, identification, development, and testing. A Gantt chart detailing the expected start dates of each milestone, the expected duration of each milestone, and the percent done so far for each milestone. As requirements have become better understood, the timeline from the progress report has been revised, with some tasks changed and some completely new milestones added.

8.1 Problem and Solution Identification

Milestone 1: Submit draft project proposal [October 13th]

Task 1.1 Research technologies of interest to use in the project.

Task 1.2 Find a problem to solve and propose a solution.

Task 1.3 Create project objectives.

- Who are the stakeholders (end-users, target audience).

8.2 Research

Milestone 2: Submit project proposal [October 30th]

Task 2.1 Create milestones and tasks for the project.

Task 2.2 Create project functional requirements. This is done by:

- Defining the project scope and identifying stakeholders.
- Define what the project should be able to do at the end.

Task 2.3 Create project non-functional requirements.

- Define stakeholder expectations of the project.
    - Usability, accessibility, reliability, availability, etc.

Task 2.4 Create sequence diagrams.

Task 2.5 Create a pros and cons list for technologies that can be used in the project.

- Consider the technologies that can be used and explain why a specific technology was used over another in a concise manner.

Task 2.6 Research possible risks in the project and ways to mitigate.

8.3 Testing

Note, Testing (Section 7.3) and Development (Section 7.4) are done concurrently.

Milestone 3: Develop a test plan [December 30th]

Task 3.1 Develop an integration test plan.

- Identify components that need testing.
- Identify key scenarios that need to be tested.

Task 3.2 Develop a unit test plan.

- Identify components that need testing.
- Identify key scenarios that need to be tested.

Milestone 4: Develop a test suite on December [March 31st]

Task 4.1 Develop integration tests and select testing criteria.

- Integration testing will begin when at least one feature has been developed.
- For every new feature, if the API of the service has been changed, a new integration test will be added.

Task 4.2 Develop unit tests and select testing criteria.

- Unit testing will begin when at least one feature has been developed.
- For every new feature, a new unit test will be added for each unit of code.

8.4 Development

Milestone 5: Robotic Arm is acquired [October 18th]

Task 5.1 Connect and test the robotic arm with board to see if all components are functional according to the specifications.

Task 5.2 Write and test code to control each motor using only the Raspberry Pi and the robotic arm's associated components.

Task 5.3 Wire the hard stop and start buttons. Test if the robotic arm powers on and off accordingly.

Milestone 6: Deploy servers and establish continuous integration [November 30th]

Task 6.1 Deploy web server in the cloud.

Task 6.2 Deploy application server in the cloud.

Task 6.3 Establish continuous integration pipeline.

Milestone 7: Create all static user interface pages [December 20th]

Task 7.1 Web application has a landing page that directs to a login page.

Task 7.2 Web application has a login page.

Task 7.3 Web application has a view for controlling the robot with the necessary buttons and screens.

- The camera stream, doctor's notes, buttons to move the arm, buttons to pick up items (mallet, tongue depressor, and flashlight).
- Note: the buttons moving the robotic arm should not be functional at this point, just shown on the web application. This will be added later.

Milestone 8: Acquire a mallet, tongue depressor, and flashlight [December 21st]

Task 8.1 Create using Fusion360, build or find a tray that fits the dimensions of each medical tool.

Task 8.2 Establish/create a mount for the tray next to the robot so it is in a consistent spot for the robotic arm.

Task 8.3 Write a pre-set routine for each medical device.

Milestone 9: Add persistence to the backend server for login functionality [January 7th]

Task 9.1 Select relational databases to be used.

Task 9.2 Configure the relational database to connect to the backend.

Task 9.3 Persist doctor data in a database, such as username and CPSO number, to be able to distinguish between doctor's that have been authenticated and anyone else trying to access the robotic arm controls.

Task 9.4 Users can successfully login through the website.

Milestone 10: Establish a connection between the frontend, backend, and robotic arm [January 7th]

Task 10.1 Frontend button presses can properly send http requests to the backend.

Task 10.2 Http requests received in the backend trigger communication with the robotic arm endpoint.

Milestone 11: Web application buttons can move the robotic arm [January 12st]

Task 11.1 Develop functionality for the web application buttons to move the robotic arm up/down, left/right and change the depth.

Task 11.2 Develop buttons to move the robotic arm in preset routines to pick up a mallet, tongue depressor and flashlight.

Milestone 12: Establish camera live streaming from the robotic arm to Twitch [February 29th]

Task 12.1 Integrate the Twitch application into the web application.

Task 12.2 Configure the Twitch stream on the Raspberry Pi.

8.5 Project Deliverables

Milestone 13: Submit project progress report [December 8th]

Task 13.1 Complete draft of progress report and request feedback from the supervisor on November 17th.

Task 13.2 Make adjustments based on the feedback.

Milestone 14: Oral presentation [January 25th-30th]

Task 14.1 Submit the oral presentation form.

Task 14.2 Create a script to present a working product.

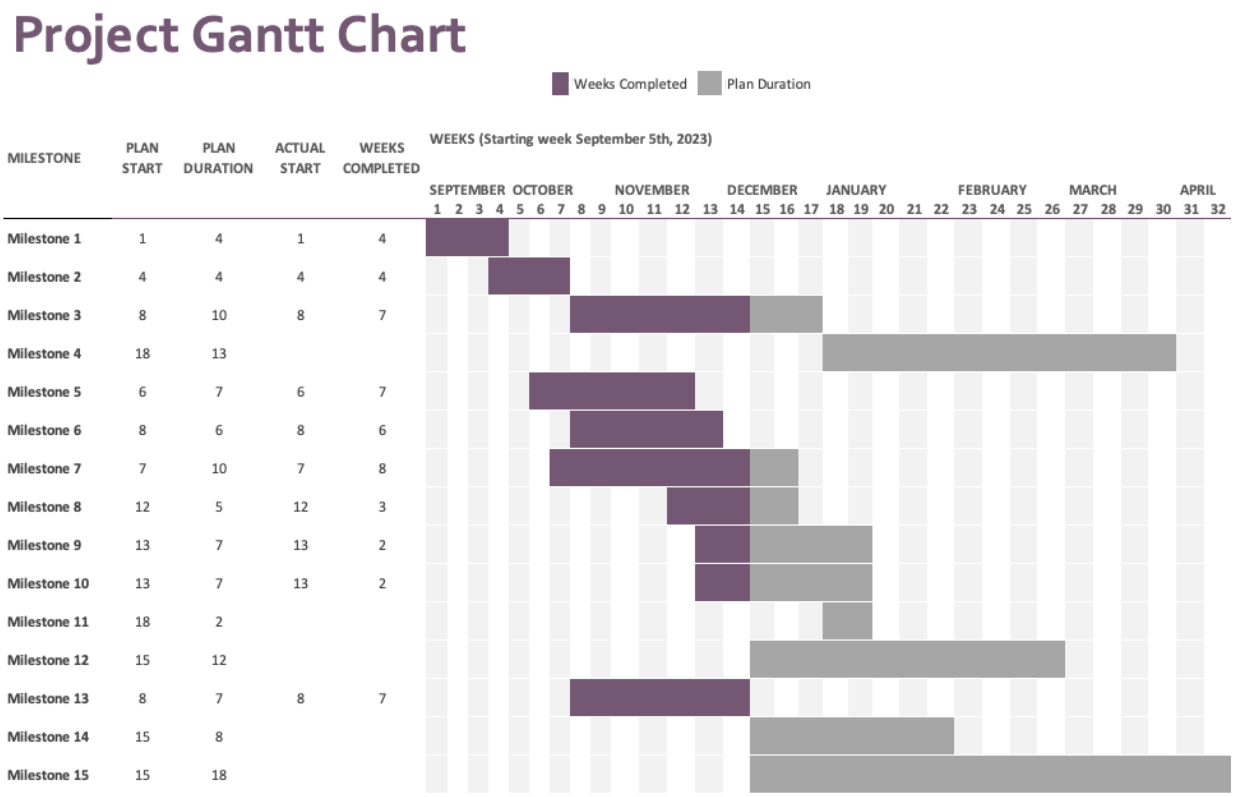
- Show the robotic arm moving according to the buttons pressed on the web application.

Milestone 15: Submit final project report [April 10th]

Task 15.1 Complete draft 1 of progress report and request feedback from the supervisor on February 16th.

Task 15.2 Complete draft 2 of progress report by making adjustments based on the feedback from draft 1 and request feedback from the supervisor on March 14th.

Task 15.3 Make final adjustments and based on the feedback from draft 2.



MILESTONE	PLAN START	PLAN DURATION	ACTUAL START	WEEKS COMPLETED
Milestone 1	1	4	1	4
Milestone 2	4	4	4	4
Milestone 3	8	10	8	7
Milestone 4	18	13		
Milestone 5	6	7	6	7
Milestone 6	8	6	8	6
Milestone 7	7	10	7	8
Milestone 8	12	5	12	3
Milestone 9	13	7	13	2
Milestone 10	13	7	13	2
Milestone 11	18	2		
Milestone 12	15	12		
Milestone 13	8	7	8	7
Milestone 14	15	8		
Milestone 15	15	18		

Figure 11: Gantt chart for the robotic arm project.

9 Mitigation of Risks

Table 9: Project risks and mitigation.

Risk	**Mitigation Technique**
Dependency on a third party application, Zoom, for patient-doctor conferencing. Doctors' appointments involve the transferring of personally identifiable information, which is confidential.	To mitigate this risk a review of Zoom's privacy standards must be performed to determine if it is an acceptable solution for this system. If the conclusion is that Zoom is not secure enough, there are many other options for view conferencing software that is created specifically for this sort of application with security as a high priority.
Robotic Arm Malfunctions/or Patient feels uncomfortable	Patient will press hard stop button
Robotic arm is unable to pick up tool	Patient can put the tool in the claw of the robotic arm
Milestones are missed and the project is not able to be finished	Project features and scope can be reduced. The bot will then be focused on the mallet reflex tests. The mallet will then be the only available tool in this interaction of the project.
High latency when connecting to the Raspberry Pi leading to a greater than 3 second delay when maneuvering	Attempt to speed up infrastructure by connecting to the Raspberry Pi through a different method.
The cost of cloud services has been calculated, but sometimes there are some hidden costs for network traffic or storage that are not accounted for.	The team will use Google Cloud free tier in hopes that the $416 of free credit given is enough to cover unexpected costs. The team will monitor the cost of the application and gather metrics in order to be able to predict future costs.

10 Required Components and Services

For this project, the robotic arm that fits the project requirements (degrees of freedom and cost) the most is the Yahboom Robotic Arm for Raspberry Pi 4B AI Smart Robotic Arm with Camera. This robot kit will include six servo motors and a camera mounted on the fifth linkage. The cost of this robot kit is $430 without tax and shipping. LEDs mounted on the robotic arm serving as a flashlight costs $10. The Raspberry Pi 4B is provided by the school at no additional cost.

Table 10: List of hardware components.

Component	**Voltage**	**Current Limit**	**Communication Protocol(s)**	**Description**
Raspberry Pi 4B	3.3 or 5V output	1-2A output	GPIO/ I2C/ CSI/ UART	Handles the data from the camera and sends instructions to motors. Communicates directionally to and from the web application.
Yahboom Robotic Arm for Raspberry Pi 4B	100 - 240 V (50/60 Hz) AC to 12V 5A			The robotic arm consists of 6 servo motors and 1 Raspberry Pi Camera Module V2 and a full-function expansion board which interfaces the Raspberry Pi, power supply and arm.
Digital Servo Motor (x4) - Torque 15KG Angle 1- 180 degrees	Not Specified	Not Specified	GPIO	To control and read each degree of freedom for the bottom 4 motors.
Digital Servo Motor (x4) - Torque 6KG Angle 1- 180 degrees	Not Specified	Not Specified	GPIO	The motor to control the “open and close” function of the gripper for the robotic arm.

Digital Servo Motor (x1) - Torque 15KG Angle 1- 270 degrees	Not Specified	Not Specified	GPIO	The motor rotates the gripper from 1-270 degrees.
Raspberry Pi Camera Module V2	3.3V	2A	CSI	Camera streams the video back to the camera in 1080p. It is attached to the 5th linkage on the Robotic Arm and is included in the kit.
WS2812B [47]	5V	2A	GPIO	A series of RGB Light Emitting Diodes (LED) attached to the 5th linkage on the Robotic Arm.

11 Conclusions

While implementing this project, there have been a few changes to the course of action when new information on what would be a better way of implementing comes up.

While researching the use of Spring Boot Webflux with a database for the backend, it was found that the reactive features cannot be effectively leveraged unless Spring Data Reactive Repositories is used. JPA cannot be used reactively. Spring Data Reactive Repositories has integration with different databases, making some databases more desirable than others for use with Reactive Spring Boot.

While attempting to stay within the free tier when using cloud services for this project, use of the AWS Elastic Container Service was found to be unusable with backing instances that do not have GPU, none of the free tier EC2 instance types have any GPU so ECS cannot be used for free. The cost of running the service in AWS for the rest of the duration of the project was calculated to be greater than the remaining project budget, therefore AWS ECS cannot be used for this project.

Google Cloud was found to provide a free tier that is more substantial, but only for the first 3 months. After the first 3 months, a certain number of free credit from the free trial is available until it runs out. This free tier is more suitable for this project as it only needs to be functional until the end of April. As an overall conclusion, attempting to use a cloud free tier puts severe limitations on how the application can be deployed.

During the configuration and setup of the robotic arm, it has been found that OpenCV would need to be a critical part of the robot to operate with efficacy and precision. It gives the robot the ability to perceive objects and contours. Adding the computer vision component to this project allows for the pre-programmed buttons to dynamically adapt to its surroundings in contrast to the original idea of “hard coding” actions. Therefore, even when objects are misplaced or placed outside of a predefined position in relation to the robot, it can still apprehend the appropriate tool. In addition, guiding the doctor to the approximate area of the ulnar nerve will improve the experience for the doctor and speed up the process of the examination. Another change is relying minimally on the user for setup which removes a potential hurdle in the user accessibility and issues with setup. Thus, we have chosen to attach the flashlight on the robot arm itself to guarantee the flashlight is turned on, illuminates hard to see areas, and has a long enough battery life for the task.

The frontend development is on track. No major adjustments or changes have been made with the exception of importing the Material UI library and the React router library.

12 Reflections

Throughout the implementation of this project, when attempting to design a solution for a large portion of the project, the team initially tries to tackle the problem simultaneously. This usually results in a waste of time as the team does not fully understand the whole picture and the possible causes of the issues. When approaching problems, the team will prioritize spending time breaking down the problem into smaller components. Then, by solving each component separately, the team will be able to piece together a complete solution. This will result in less time wasted learning new technologies or project related topics.

In order to work most productively, aspects of the project should be broken down into items with consideration towards which items can be completed in parallel. When items cannot be done in parallel, some team members can become blocked until another team member completes their item, which wastes valuable time. Parallel item design should be considered for the remainder of the project and at each planning meeting. The team should ensure that all team members have a work item to complete that is not blocked by another team member.

In order to attempt to have continuous progress, the creation of an attainable goal for the number of items completed by each team member each week is helpful. Using a goal of one item per person per week would help to ensure that progress is consistently being made. This would be a useful goal to have in place for the remainder of the project.

A consideration future groups should have is to treat the allocated budget sparingly. For this project, the overwhelming majority of the budget was allocated towards the robotic arm, leaving little-to-no budget left for cloud hosting or a “rainy-day” fund. The team thought that free tier cloud hosting would be sufficient, but, as we quickly found out, we would require more computing power than what was provided in free tiers. If there were more in the budget for hosting, this would not have been an issue, but instead the team was forced to find a more creative deployment strategy.

References

[1] "Agile Manifesto - 12 Principles," Agile Manifesto, [Online]. Available: https://Agilemanifesto.org/principles.html. Accessed on: October 24, 2023.

[2] "About - Alpine Linux," Alpine Linux, [Online]. Available: https://www.alpinelinux.org/about/#:~:text=Alpine%20Linux%20is%20built%20around,of%20packages%20from%20the%20repository. Accessed on: October 24, 2023.

[3] "Amazon EC2 - Elastic Compute Cloud," Amazon Web Services, [Online]. Available: https://aws.amazon.com/ec2/?did=ft_card&trk=ft_card. Accessed on: October 24, 2023.

[4] "Amazon S3 - Simple Storage Service," Amazon Web Services, [Online]. Available: https://aws.amazon.com/s3/. Accessed on: October 24, 2023.

[5] "AWS Free Tier - Amazon Web Services (AWS)," Amazon Web Services, [Online]. Available: https://aws.amazon.com/free/. Accessed on: October 24, 2023.

[6] "Reduce Latency for End Users with Multi-Region APIs with CloudFront," Amazon Web Services (AWS), [Online]. Available: https://aws.amazon.com/blogs/networking-and-content-delivery/reduce-latency-for-end-users-with-multi-region-apis-with-cloudfront/. Accessed on: October 24, 2023.

[7] "Apache Maven - Welcome to Apache Maven," Apache Maven, [Online]. Available: https://maven.apache.org/index.html. Accessed on: October 24, 2023.

[8] "Continuous Delivery and Continuous Integration," Atlassian, [Online]. Available: https://www.atlassian.com/continuous-delivery/continuous-integration. Accessed on: October 24, 2023.

[9] "Kanban - Atlassian," Atlassian, [Online]. Available: https://www.atlassian.com/Agile/Kanban#:~:text=In%20Japanese%2C%20Kanban%20literally%20translates,in%20a%20highly%20visual%20manner. Accessed on: October 24, 2023.

[10] “Stress Health And Safety In Software Project Management,” Business Managment, [Online]. Available:

https://business.joellemena.com/project-managment/stress-health-and-safety-in-software-project-management/. Accessed December 8, 2023.

[11] Carleton University, "ECOR 4995 - Undergraduate Academic Support," [Online]. Available: https://carleton.ca/engineering-design/current-students/undergrad-academic-support/ecor-495/. Accessed on: October 24, 2023.

[12] “Carleton University Calendars 2023-24 edition,” Systems and Computer Engineering (SYSC) < Carleton University, https://calendar.carleton.ca/undergrad/courses/SYSC/. Accessed on October 25, 2023.

[13] "Docker Official Images - Alpine," Docker Hub, [Online]. Available: https://hub.docker.com/_/alpine. Accessed on: October 24, 2023.

[14] "Docker Documentation - Build Images," Docker, [Online]. Available: https://docs.docker.com/build/. Accessed on: October 24, 2023.

[15] "Docker Documentation - Compose," Docker, [Online]. Available: https://docs.docker.com/compose/. Accessed on: October 24, 2023.

[16] “Docker images,” Docker Documentation, https://docs.docker.com/engine/reference/commandline/images/. Accessed on October 25, 2023.

[17] "Docker Official Images - Ubuntu," Docker Hub, [Online]. Available: https://hub.docker.com/_/ubuntu. Accessed on: October 24, 2023.

[18] “OpenJDK,” Docker Hub, [Online]. Available: https://hub.docker.com/_/openjdk. Accessed December 8, 2023.

[19] “Mongo,” Docker, [Online]. Available: https://hub.docker.com/_/mongo. Accessed December 8, 2023.

[20] “NgInx,” Docker, [Online]. Available: https://hub.docker.com/_/nginx. Accessed December 8, 2023.

[21] “Node,” Docker, [Online]. Available: https://hub.docker.com/_/node. Accessed December 8, 2023.

[22] "Companies using Linked Data in 2023," GitHub, [Online]. Available: https://github.com/d2s/companies/blob/master/src/index.md. Accessed on: October 24, 2023.

[23] "GitHub Actions Documentation," GitHub, [Online]. Available: https://docs.github.com/en/actions. Accessed on: October 24, 2023.

[24] "GitHub Actions Starter Workflows - AWS Deployment Workflow," GitHub, [Online]. Available: https://github.com/actions/starter-workflows/blob/main/deployments/aws.yml. Accessed on: October 24, 2023.

[25] "GitHub Actions Documentation - Understanding GitHub Actions," GitHub, [Online]. Available: https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions. Accessed on: October 24, 2023.

[26] "GitHub Actions Documentation - Usage Limits, Billing, and Administration," GitHub, [Online]. Available: https://docs.github.com/en/actions/learn-github-actions/usage-limits-billing-and-administration. Accessed on: October 24, 2023.

[27] "Cloud Run pricing," Google Cloud, [Online]. Available: https://cloud.google.com/run/pricing. Accessed December 8, 2023.

[28] "Free cloud features and trial offer," Google Cloud, [Online]. Available: https://cloud.google.com/free/docs/free-cloud-features. Accessed December 8, 2023.

[29] "Material UI - Overview," Material UI, [Online]. Available: https://mui.com/material-ui/getting-started/. Accessed December 8, 2023.

[30] A. Naceri et al., "Tactile robotic telemedicine for safe remote diagnostics in times of Corona: System design, feasibility and usability study," IEEE robotics and automation letters, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9454265/. Accessed on: October 25, 2023.

[31] "NGINX Documentation - Web Server," NGINX, [Online]. Available: https://docs.nginx.com/nginx/admin-guide/web-server/. Accessed on: October 24, 2023.

[32] "Configuring HTTPS servers," NgInx, [Online]. Available: https://nginx.org/en/docs/http/configuring_https_servers.html. Accessed on: December 8, 2023.

[33] “Nginx reverse proxy,” NgInx Docs, [Online]. Available: https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/. Accessed on: December 8, 2023.

[34] “What is Nginx?,” NgInx, [Online]. Available: https://www.nginx.com/resources/glossary/nginx/. Accessed on: December 8, 2023.

[35] "Your Health Plan: Connected and Convenient Care," Ontario.ca, [Online]. Available: https://www.ontario.ca/page/your-health-plan-connected-and-convenient-care. Accessed on: October 24, 2023.

[36] Fact sheet: Ontario’s doctor shortage - oma, https://www.oma.org/uploadedfiles/oma/media/public/hcp-factsheet-doctor-shortage.pdf. Accessed on: October 25, 2023.

[37] "Ontario Regulation 900/94 - PROFESSIONAL ENGINEERS ACT," Ontario.ca, [Online]. Available: https://www.ontario.ca/laws/regulation/900941. Accessed on: October 24, 2023.

[38] OpenCV documentation index, https://docs.opencv.org/. Accessed on: December 8, 2023.

[39] "Professional Engineers Ontario (PEO) Guideline," Professional Engineers Ontario, [Online]. Available: https://www.peo.on.ca/sites/default/files/2020-12/PEPGuideline_Nov2020.pdf. Accessed on: October 24, 2023.

[40] J. Patadiya, “React vs react native - key difference, features, advantages,” Radixweb, Available: https://radixweb.com/blog/react-vs-react-native. Accessed October 24, 2023.

[41] N. Raval, "React vs Angular: Which JS Framework to Pick for Front-end Development?," Radixweb. Available: https://radixweb.com/blog/react-vs-angular. Accessed on: October 24, 2023.

[42] “Feature overview,” React Router, [Online]. Available: https://reactrouter.com/en/main/start/overview. Accessed on: December 8, 2023.

[43] “Reactive”, Spring, [Online]. Available: https://spring.io/reactive. Accessed on: December 8, 2023.

[44] "Spring Framework - Reactive Web Framework," Spring Framework Documentation, [Online]. Available: https://docs.spring.io/spring-framework/reference/web/webflux/new-framework.html. Accessed on: October 24, 2023.

[45] "Spring Framework - Reactive Web Applications," Spring Framework Documentation, [Online]. Available: https://docs.spring.io/spring-boot/docs/current/reference/html/web.html#web.reactive. Accessed on: December 8, 2023.

[46] "Stack Overflow Developer Survey 2023 - Most Popular Technologies," Stack Overflow, [Online]. Available: https://survey.stackoverflow.co/2023/#technology-most-popular-technologies. Accessed on: October 24, 2023.

[47] "Intelligent control LED integrated light source," Worldsemi, [Online]. Available: https://cdn-shop.adafruit.com/datasheets/WS2812B.pdf. Accessed on: December 8, 2023.

[48] R. Van Loon et al., Laboratory Health and Safety Manual, https://sce-soft-web-ist.sce.carleton.ca/wp-content/uploads/2023/07/health-and-safety.pdf.

[49] "DOFBOT-Pi - Yahboom Robotics," Yahboom, [Online]. Available: https://category.yahboom.net/collections/rp-robotics/products/dofbot-pi. Accessed on: October 24, 2023.

[50] Yahboom robotic arm for Raspberry Pi 4b AI smart robotic ... - amazon.ca, https://www.amazon.ca/Yahboom-Robotic-Raspberry-%EF%BC%88DOFBOT-Without/dp/B08T8XG2J6. Accessed on Oct. 25, 2023.

[51] M. Zhang, "Top 10 cloud service providers globally in 2023," Dgtl Infra, https://dgtlinfra.com/top-cloud-service-providers/#:~:text=Amazon%20Web%20Services20(AWS)%2C%20the%20cloud%20computing%20service%20of,compute%2C%20storag%2C%20and%20database. Accessed on October 25, 2023.

Appendix A

Angular: Typescript-based, free, and open-source single-page web application framework based upon the Model-View-Controller design pattern.

Application Programming Interface (API): a set of rules and protocols that allows different software applications to communicate with each other.

Degrees of Freedom: the number of independent variables that define the possible motions of a mechanical system.

Docker: a set of platform as a service products that use operating system level virtualization to deliver software in packages called containers.

Document Object Model (DOM): a programming interface that provides a structural representation of HTML and XML documents, allowing scripts to dynamically access, modify, and update the content, structure, and style of web pages

LAN: acronym for Local Area Network. Refers to a collection of connections connected in one physical location.

LED: acronym for Light Emitting Diode. Refers to a semiconductor device that emits lights when an electric current passes through it

LTE: acronym for Long-Term Evolution. Refers to a standard for wireless broadband communication.

Maven: a build automation tool predominantly used for Java projects. It simplifies the building and management of Java-based applications.

OpenCV: stands for Open Source Computer Vision. It is an open-source computer vision and machine learning software library mainly to be used for real-time computer vision.

Persistence of Objects: the ability to save the state of an object and retrieve it at a later time.

Raspberry Pi: a single-board computer designed for various computing tasks, encompassing programming and hardware prototyping.

React: open-source JavaScript library, created by Meta, allowing for the efficient creation of interaction user interfaces.

REST API: acronym for Representational State Transfer Application Programming Interface. Refers to a standardized set of rules for facilitating communication between software applications over the internet.

Spring-Boot: open-source Java-based framework that simplifies the development of production-grade, stand-alone, and web-based applications.

Telemedicine: describes an area of medicine in which medical care is not provided directly but over a more or less greater distance.

Telemonitoring: refers to using different types of technology to monitor patients at a distance.

Twitch: a video live streaming service that focuses on video game live streaming, including broadcasts of esports competitions, in addition to offering music broadcasts, creative content, and "in real life" streams.

Wi-Fi: a wireless networking technology that uses radio waves to provide high-speed internet access.

Zoom: Zoom is a communications platform that allows users to connect with video, audio, phone, and chat.

Appendix B

Mock Frontend Design

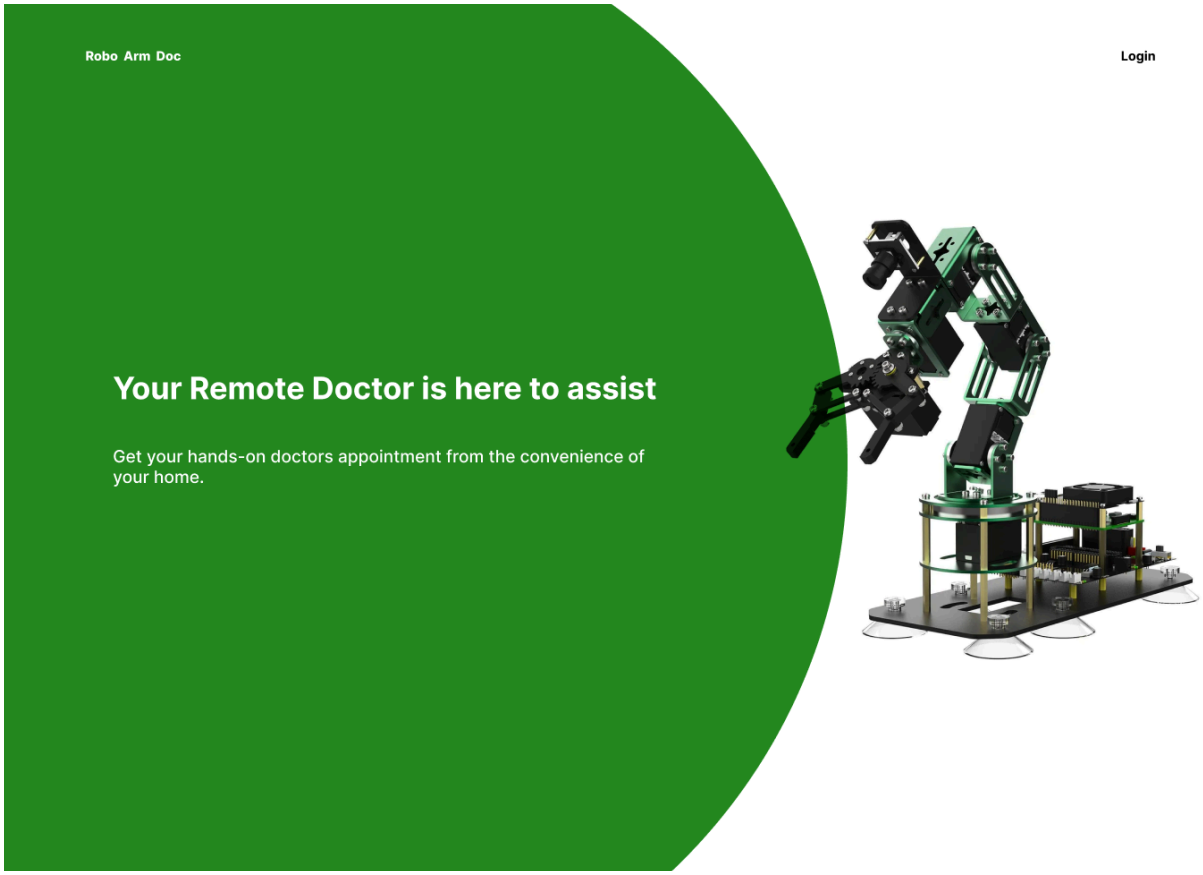


Figure 12: The Figma mockup for the website landing page.

Figure 13: The Figma mockup for the website sign-in portal.

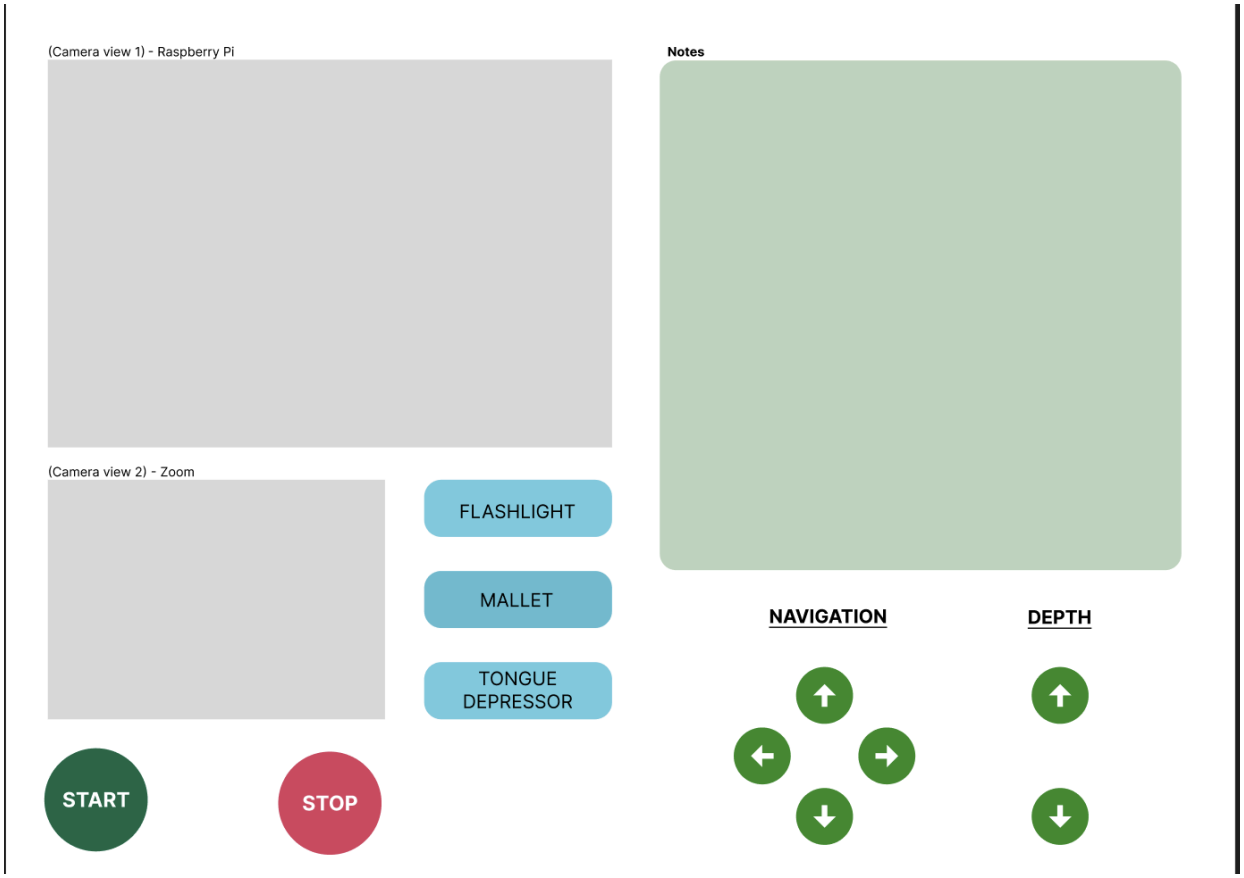


Figure 14: Doctor view of healthcare web application

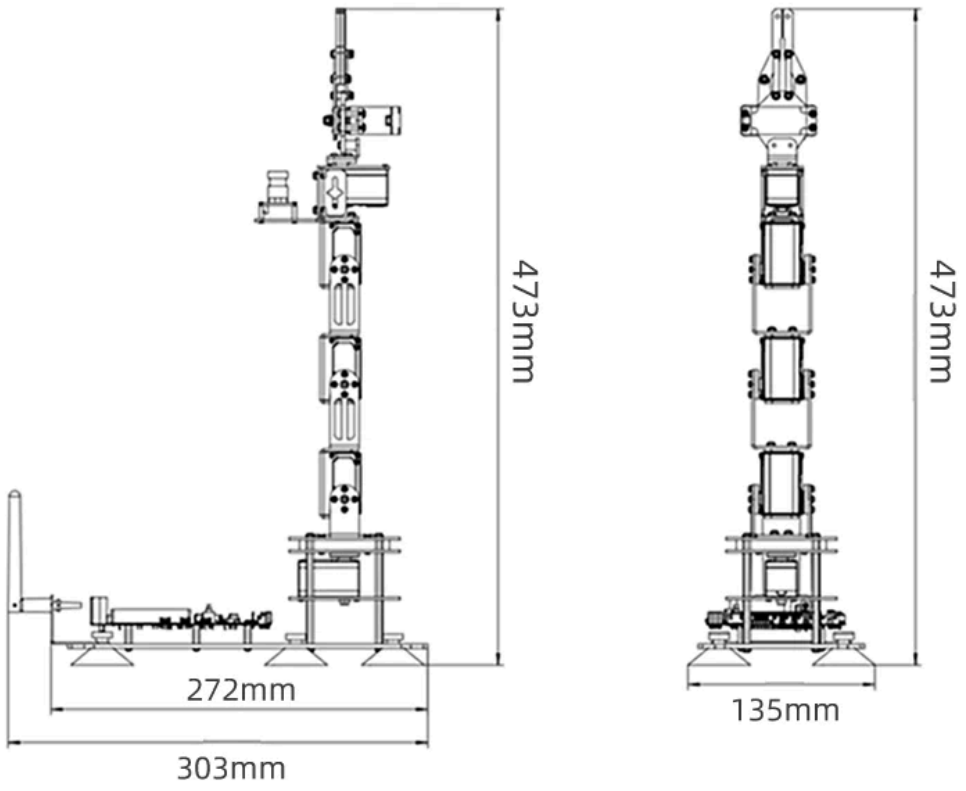


Figure 15: Robotic arm schematic (orthographic) [49]

Appendix C

Course Descriptions

Table 11: Course descriptions taken directly from the course outlines [12]

Courses	Course Description
ECOR 1051: Fundamentals of Engineering I	Software development as an engineering discipline, using a modern programming language. Tracing and visualization of program execution. Testing and debugging. Data management: digital representation of numbers; numerical algorithms; storing data in files; container data types: sequences, sets, maps.
SYSC 2004: Object-Oriented Software Development	Designing and implementing small-scale programs as communities of collaborating objects, using a dynamically-typed or statically-typed programming language. Fundamental concepts: classes, objects, encapsulation, information hiding, inheritance, polymorphism. Iterative, incremental development and test-driven development.
SYSC 3110: Software Development Project	Development of expertise in designing, implementing and testing maintainable, reusable software through team projects. Applying modern programming languages, design patterns, frameworks, UML and modern development processes (detection of olfactible source code defects, refactoring, iterative and incremental development, version control techniques) to medium-scale projects.
SYSC 3120: Software Requirements Engineering	Current techniques, notations, methods, processes and tools used in Requirements Engineering. Requirements elicitation, negotiation, modeling requirements, management, validation. Skills needed for Requirements Engineering and the many

	disciplines on which it draws. Requirements analysis: domain modeling, modeling object interactions; UML modeling. Introduction to software development processes.
SYSC 3303: Real-Time Concurrent Systems	Principles and practice of a systems engineering approach to the development of software for real-time, concurrent, distributed systems. Designing to achieve concurrency, performance, and robustness, using visual notations. Converting designs into programs. Introduction to hard real-time systems. Team project.
SYSC 3310: Introduction to Real-Time Systems	Principles of event-driven systems. Microcontroller organization. Development of embedded applications. Programming external interfaces, programmable timer. Input/output methods: polling, interrupts. Real-time issues: concurrency, mutual exclusion, buffering. Introduction to concurrent processes.
SYSC 4120: Software Architecture and Design	Introduction and importance of software architectures and software system design in software engineering. Current techniques, modeling notations, methods, processes and tools used in software architecture and system design. Software architectures, architectural patterns, design patterns, software qualities, software reuse.
SYSC 4805: Computer Systems Design Lab	Developing professional-level expertise in selected, important areas of the field by applying, honing, integrating, and extending previously acquired knowledge in team projects in the laboratory. Lecture periods are devoted to new knowledge required for the selected areas, to project-related issues, and to student presentations.
SYSC 4806: Software Engineering Lab	Applying the full spectrum of engineering and programming knowledge acquired in the program through team projects in the laboratory. Practice in doing presentations and reviews. Lectures will discuss software engineering issues as they relate to the projects, from a mature point of view.

Appendix D: Project Proposal

SYSC4907: Project Proposal

Graphical Control Interface for a Robotic Arm

Group #110

Supervisor: Dr. Lynn Marshall

Dhriti Aravind (101141942)
Elisha Catherasoo (101148507)
Sarah Chow (101143033)
Evan Smedley (101148695)
Jeremy Trendoff (101160306)

October 25th, 2023

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Project Objective

This project aims to create a website for doctors to remotely control the movements of a robotic arm placed next to a patient. This will allow doctors to perform virtual general physical health check-ups using common medical tools, such as a flashlight, a mallet, and a tongue depressor. This project provides a tool for remote and/or understaffed communities to receive general physical check-ups. The doctor will be able to control the robotic arm through a website interface and view the perspective of the arm through a Raspberry Pi camera. The system will send a Zoom meeting link to the patient in which they are expected to join the call from their own device and angle the device's camera in a third-person perspective. The doctor must log in to the system using their College of Physicians and Surgeons of Ontario (CPSO) number to access the website. The doctor will have two camera views: one from the mounted camera on the Raspberry Pi and another view of the Zoom meeting with the patient's device. The doctor will be able to take notes during the check-up session. At the end of the appointment, the system will send a text file of the notes to the doctor's email for them to review and send to the patient.

1.2 Background

Remote, understaffed, and underserved communities struggle to have adequate access to basic healthcare. In Ontario, access to healthcare has been a challenge. Ontario's health minister, Sylvia Jones, has stated in her plan for connected and convenient care that, "...the status quo is not working. Too many people are waiting too long to get an appointment or surgery, having to travel too far to get care, and spending too much time trying to navigate our healthcare system." [1]. According to the Ontario Medical Association, "at least 1 million Ontarians do not have regular access to primary care" and it is most prevalent in northern and rural areas [2]. The shortage of primary care physicians creates a challenge for individuals to access the necessary preventative care resources to detect symptoms of sickness. To solve this problem, our group proposes a robotic arm controlled by a doctor that can serve patients in remote areas.

1.3 Current State of the Art Solutions

The field of remotely controlled doctor-patient interaction is an emerging field and the current applications are built for specialized diagnoses [3]. For example, there exists a robot that primarily focuses on diagnosing breast cancer [3]. General purpose robots have been developed in the telemedicine industry as well, including a robot called the Tactile Robotic Telemedicine system [3]. Some of the differences in the Tactile Robotic Telemedicine system in comparison to the robot described in the project includes a robot that utilizes two identical robotic arms with seven degrees of freedom: one on the doctor's site and another on the patient's site. The two robotic arms can interact through LAN, Wi-Fi, or LTE networks. The doctor uses a joystick to interact with the robot on their side, which will then relay the movements to the robotic arm on the patient's side. Additionally, the robotic arm on the doctor's site can receive haptic force feedback to provide more information for the diagnosis. The doctor can view a 3D rendering of the robotic arm on the patient's site as it moves.

1.4 Project Overview

This project will be completed through a series of milestones and objectives. Throughout the project, our team will complete the objectives through these three main actions:

1. Building a robotic arm that utilizes Raspberry Pi 4B
2. Creating a backend service to facilitate the communication from a web interface to the robotic arm. This service shall be hosted on an Amazon Web Services (AWS) cloud server.
3. Creating a frontend interface (website) for the doctor to control the robotic arm. This will be done using React.

2 The Engineering Project

2.1 Health and Safety

During this project all members of the project team will stringently follow the manual provided by project administrators that outlines the minimum standards and practices for the safe and healthy operation of a laboratory [4]. The following of this manual will ensure that all work meets the requirements of the Occupational Health and Safety Act of Ontario (OHSA).

All team members have reviewed Section 4: Responsibilities of Laboratory Workers (from the manual), and certify that we understand what our responsibilities are in the lab context. For any time when the project room is used, all team members will be sure to follow the General Health and Safety Principles and Basic Safety Procedures outlined in sections 5 and 6 of the manual.
Since a large portion of the project will be completed in a non-laboratory setting through the writing of software, in a variety of other environments (home, library, etc.), team members will likely not encounter such hazards.

2.2 Engineering Professionalism

During this project all team members will strive to demonstrate professionalism in accordance with what was learned during ECOR 4995 Professional Practice [5]. Some of the ways in which professionalism will be demonstrated, as extracted from the Professional Engineering Practice Guideline, are listed below [6]:

1. As none of the project team members are professional engineers at this time, engineering titles (such as ‘Engineer’) outlined in section 7.3 Use of Titles of the referenced guideline
2. The Engineer’s Duty to Report design processes and procedures that are unsafe will be followed by team members in all aspects of the project design and implementation
3. Interactions among team members and between team members and stakeholders will be managed professionally
4. Reports will be written not only as per the project guidelines but also in accordance with section 10.5 Report Writing of the referenced guideline (except for the part about the use of the professional engineer’s seal because evidently we are not professional engineers yet)
5. Meeting notes and all project documents will be retained for referencing later on

6. The Code of Ethics as outlined in section 77 of the Professional Engineers Act will be followed by all team members at all times [7]
7. All members of the team will strive to not engage in professional misconduct as per section 27 of the Professional Engineers Act [7].

2.3 Justification of Suitability for Degree Program

The project team is composed of the five following members. Note that course descriptions can be found under their respective titles in Appendix B.

Dhriti Aravind

Dhriti Aravind is a fourth-year student in computer systems engineering. In her program, she learned the fundamentals of programming (ECOR 1051, SYSC 2004, SYSC 3303, SYSC 4805), the applications of embedded programming, the basics of electronics, and how to analyze systems and simulate them. She has developed her interest in hardware and software through her internships at Ciena, Huawei, Microsoft, and Tesla where she worked in various areas of silicon development and tool support. Through her internship experiences, she familiarized herself with Python, C, Verilog, and System Verilog and participated in Agile scrums and a fast-paced environment. In addition, during her free time, continues to pursue her passion in robotics through various personal projects to automate mundane tasks; while undertaking these projects, she has refined her skills in 3D modeling and programming microcontrollers. In this project, Dhriti will focus on the design and programming of the robotic arm, designing the tray containing the medical equipment, and integrating the robotic arm with the website interface.

Elisha Catherasoo

Elisha Catherasoo is a fourth-year student in software engineering. She has gained experience throughout her education at university (ECOR 1051, SYSC 2004, SYSC 3110, SYSC 3310, and SYSC 4806), personal projects, and her internships at Ericsson and Amazon. At university, she learned the basics of programming, such as OOP, software architecture, and real-time systems. Learning real-time systems has taught her how to use a Raspberry Pi and basic coding projects. This experience will help when programming the Raspberry Pi to control the movement of the robotic arm and developing the presets for the robotic arm picking up specific objects (mallet, tongue depressor, and flashlight). When making the database for the authenticated doctors, she can use what she learned in SYSC 3110 and SYSC 4806 to be able to persist the authenticated doctor data. Her internship at Ericsson gave her experience

with working collaboratively with her team, which includes people of varying strengths and weaknesses, and being able to know who is best to do what part to better the given project(s). Her experience with Amazon gave her exposure to AWS, which will help when deploying the robotic arm web application to be accessed anywhere. Elisha's experience in university and her internships has given her the valuable skills needed when working in a team, in the basics of programming, and in the technologies needed to be able to successfully complete the project.

Sarah Chow

Sarah Chow is a fourth-year student in software engineering. She has taken several programming courses throughout her undergraduate degree pertaining to software development including software development (SYSC 3110), software architecture (SYSC 4120), and object-oriented programming (SYSC 2004). Throughout her university degree, she has had several co-op placements, including Warner Bros. Discovery and the Royal Bank of Canada. At Warner Bros. Discovery, Sarah worked as an iOS and tvOS developer and implemented frontend features for the company. At the Royal Bank of Canada, Sarah was a frontend and backend developer over the eight-month co-op. She became knowledgeable in web development with Angular and Spring Boot and also participated in an Agile team with daily stand-ups, bi-weekly sprints, and monthly retrospectives using JIRA and Azure DevOps. In 2022, Sarah attended the hackathon, Hack the North, in which she created the frontend using React and open-source libraries to build a website. Sarah has an interest in developing her frontend skills.

Evan Smedley

Evan Smedley is a fourth-year student in software engineering. During his degree, he has gained knowledge of object-oriented software development (SYSC2004, SYSC3110, SYSC4806), software architecture (SYSC3120, SYSC4120), and the development of real-time systems (SYSC3310, SYSC3303). Over approximately the past two years, Evan worked full-time as a Software Engineer Intern at Motorola Solutions Inc. During his time at Motorola, Evan gained experience applying his software knowledge and learning much more about all layers of the software stack. In the infrastructure area of the software stack, Evan learned a lot about continuous integration, highly available systems, and cloud deployment of services with AWS and Microsoft Azure which will be useful when deploying the backend of our web application. In the backend of the stack, Evan gained experience with the sending of media over various protocols and developing and packaging React services all while considering latency requirements, which will help with the writing of the backend of our web application. During the internship, Evan experienced Agile Kanban and Agile Scrum with a variety of spring lengths and metrics to monitor productivity, which he hopes to bring to this project to promote a satisfying and transparent

experience for all stakeholders. Some tools and technologies will be used in the project that were chosen partly because Evan is experienced with them. These include Azure DevOps, containerization with Docker, AWS as a cloud hosting platform, a SpringBoot backend, and GitHub Actions.

Jeremy Trendoff

Over the past eight years, Jeremy Trendoff has studied and practiced the fundamentals of software engineering and development. Over the last four years as a software engineering student at Carleton University, Jeremy has taken classes in embedded and real-time systems (SYSC3310, SYSC3303), object-oriented programming (SYSC3110, SYSC4806), software architecture and documentation (SYSC3120, SYSC4120). This knowledge will relate directly to both the development of the interface, as well as the software portion of the arm. On top of this, Jeremy enrolled in a year-long co-op term at SOTI Inc., a leader in Enterprise Mobility Management and the Internet of Things. There, Jeremy worked as a fullstack developer, helping to solve frontend, backend, middleware, architecture, and research problems. At SOTI, Jeremy also learned the basics of professional practice for software development. Jeremy gained experience with tools like Git, JIRA, and the Agile development lifecycle, but, most importantly, learned how to develop software as a team. This knowledge is sure to prove essential during this, and any, professional project.

The Team

As outlined above, each member possesses valuable skills and experience. Collectively, the team's overall skills encompass the requirements to develop this robotic arm project. To tackle the software portion, our four software engineering majors have experience in user interface design and programming, software architecture principles, object-oriented programming, and have studied embedded and real-time systems programming to competently handle programming on the hardware side of things. Our computer systems engineering major has not only learned the fundamentals of programming and software design but has built extensive knowledge of the basics of electronics and real-time systems. Combining our strengths, our team has the knowledge and passion to complete this project.

2.4 Project Management

2.4.1 Team Structure

Team	**Primary Contributor (Specialist)**	**Second Contributor (Floater)**
Robotic Arm	Dhriti	Elisha
Frontend	Sarah	Jeremy
Backend	Evan	Jeremy, Elisha

Table 1: Project group sub-teams and their contributors.

Our team has decided to structure ourselves using a non-hierarchical organization. This means we will operate as one cohesive, democratic group. Instead of having team leaders, we will all be equally responsible for driving the project forward. The team will operate through three main subteams: The frontend team, backend team, and the robotic arm team. These teams will be cross-functional, but the main role divisions have been outlined. While everyone on each team will be expected to contribute to the architecture and development, we have structured ourselves into specialist and floater roles. Specialists (primary contributors) will mainly focus on their team's work throughout the project. Floaters (secondary contributors) will split their time according to the current demands of the project. On the frontend team, Sarah Chow will be our specialist and Jeremy Trendoff will act as a frontend floater. On the backend team, Evan Smedley will be our specialist. Jeremy Trendoff and Elisha Catherasoo will be backend floaters. Finally, on the robotic arm team, Dhriti Aravind will be our specialist and Catherasoo will act as the arm floater. This structure may be subject to change but at the moment, it suits our team's strengths.

2.4.2 Software Development Lifecycle Methodology

In keeping with some team members' experience during co-op terms, we will be using Agile methodologies for this project. We will be following the 12 Agile principles outlined in the Agile manifesto [8]. The Agile methodology we felt would best suit our needs is Kanban [9]. Developed by Toyota in the 1940s to optimize their engineering process, Kanban has become one of the most popular frameworks used to implement both Agile and DevOps software development. Kanban uses a highly

visual Kanban board which provides full transparency of work. On the board, Kanban cards will move from the basic states ‘To Do’, ‘In Progress’, and ‘Done’. Once a work item has been completed, the next work item in the backlog will take that space on the board. So, as long as the backlog is up to date, there is no need for fixed-length sprints. This will also suit our non-hierarchical structure as all team members are treated with equal privilege with this scheme. Kanban is also great for teams of all sizes. While other Agile methodologies have more structure, Kanban allows for more flexibility across the board. This will allow our team to quickly solve problems and determine the best plans of action when we get stuck.

2.5 Individual Contributions

This section carefully itemizes the individual contributions of each team member. Project contributions identify which components of work were done by each individual and report contributions list the author of each major section of this report.

2.5.1 Project Contributions

The following table shows the project contributions of each team member.

Member	Project Contributions
Jeremy Trendoff	As well as contributing to this proposal, Jeremy has contributed to the software architecture, web design, Figma mockups, project management strategy, and team structure, and has actively participated in all team discussions.
Elisha Catherasoo	Alongside contributing to the proposal, Elisha has contributed to the software architecture, web application design, and has actively participated in all team discussions.
Evan Smedley	Evan has contributed to this proposal and all team discussions about the architecture, scope, and requirements for this project. Evan researched the tech stack of the project to select tools with the right combination of functionality and simplicity to accomplish our objectives. This research is responsible for our chosen backend solution. Additionally, Evan set up the project backlog in Azure DevOps, along with permissions for the project and the Kanban board to be used to track everyone's progress.
Sarah Chow	Sarah has contributed to all team discussions, the project proposal, and frontend web design by creating Figma mock-ups. Additionally, Sarah's research into frontend frameworks is directly responsible for our chosen frontend solution.
Dhriti Aravind	Dhriti has contributed to this proposal, all team discussions, Figma mockups, and research on the robot arm. She has submitted a form to request the purchasing of the chosen arm and has gained access to the mechanical workshop. In addition, she has acquired, built, and tested the robotic arm.

Table 2: Team member contributions to the project.

2.5.2 Project Proposal Contributions

The following table shows the project proposal contributions of each team member.

Member	Proposal Contributions
Jeremy Trendoff	Jeremy has contributed to the following sections:1.1, 1.2, 1.3, 2.3, 2.4, 2.5, 3, 4.1.2, 4.1.3, 4.3, Glossary, Appendix A, Appendix B
Elisha Catherasoo	Elisha has contributed to the following sections: 1.1, 1.2, 2.3, 2.4, 2.5, 3, 4.1.2, 5, References, Glossary, Appendix A
Evan Smedley	Evan has contributed to the following sections: 2.1, 2.2, 2.3, 2.5, 4.1.2, 4.1.3, 4.1.4, 4.1.5, 4.2, References
Sarah Chow	Sarah has contributed to the following sections: 1.1, 1.2, 1.3, 1.4, 2.3, 2.4.1, 2.5, 3, 4.1.1, 4.3, Appendix A
Dhriti Aravind	Dhriti has contributed to the following sections: 1.1, 1.2, 1.3, 2.3, 2.5, 3, 4.1.6, 4.3, 5, 6, 7, Appendix A

Table 3: Team member contributions to the project proposal.

3 Requirements

3.1 Functional Requirements

The following are the functional requirements for the project:

Robot Movement and Connection

- The robot shall be able to move in 3 planes of motion (x, y, and z).
- The robot shall be connected to Wi-Fi and receive an IP address.
- The robot shall interface with the web interface using HTTP protocols.
- The robot shall be able to pick up items less than 500 grams.
- The robot shall be able to pick up items less than 20 cm in the smallest dimension.

Doctor-Patient Interaction

- The doctor and patient shall communicate over a video call throughout the session.
- The robot shall be ready for motion when the patient's physical start button is pressed.
- The robot shall stop when the patient's physical stop button is pressed.
- The robot shall stop when the doctor's virtual stop button is pressed.
- The doctor shall start the robot once the patient's physical start button has been pressed.

Web Interface Interaction

- The doctor shall log in through the login portal using their CPSO number and their password.
- The login portal shall redirect the doctor to the robot interface webpage.
- The doctor shall be able to see the robot's camera view on the web interface.
- The doctor shall be able to control the robot through a set of directional and depth control buttons on the web interface.
- The doctor shall be able to write and download notes from the notes section of the web interface
- The doctor shall be emailed the notes from the notes section of the web interface after the patient session has concluded.
- The doctor shall be able to select between different examination tools through preprogrammed action buttons on the web interface.

3.2 Non-Functional Requirements

The following are the non-functional requirements for the project:

Usability

- Twitch shall be used to stream the robot's camera view to the web interface.
- Zoom shall be used to facilitate doctor-patient video calls.

Security

- The login portal shall be supported using CPSO's authentication service to validate doctor credentials.
- The login portal shall authenticate the doctor based on CPSO number and password.

Compliance

- The system shall handle and store any personal information of its end users in accordance with the Personal Information Protection and Electronic Documents Act 2000.

Performance and Reliability

- The latency from the buttons on the doctor interface to the robotic arm movements shall be under 5 seconds.
- The delay on the video streamed from the camera on the robotic arm shall be under 5 seconds.

4 Methods

4.1 Technology Stack

This project's technology stack consists of a frontend, a backend, infrastructure tools and robotic arm control.

4.1.1 Frontend

The team debated between three frontend frameworks: React, React Native, and Angular. It was decided that the project would be a website, which excluded React Native from the framework selection as it is used primarily for web application development [10]. React and Angular are frameworks that can be used for website development. React is more flexible as it has many plugins and open-source libraries available and utilizes virtual document object model (DOM) objects; the ability to reload a single DOM object instead of reloading the entire structure of the page when there is a visual change made to the website [11]. This serves for a faster response time and requires less computational power. Angular provides developers with a routing service; a service to handle the website routing, and does not use virtual DOMs, in which the entire page structure is reloaded if there is a visual change to the website [11]. This creates a slower response time and requires more computational power.

The team has decided the React framework will be used for this project. The wide assortment of plugins and open-source libraries allows more flexibility for the website. Although Angular provides a pre-built routing service, it is not necessary as the project will only require routing to three pages. React also uses virtual DOMs which will require less performance from the cloud server than Angular and this will decrease the overall costs for the project. Refer to Appendix A for mock frontend Figma designs.

4.1.2 Backend

The main jobs allocated to the backend are facilitating doctor login from the frontend and sending instructions from the frontend to the robotic arm endpoint. This will be accomplished by building out a set of methods to facilitate this communication called our backend API. To build this backend API, the SpringBoot framework will be used. We shall also leverage Spring WebFlux, a framework that allows for Reactive programming to respond to events from the frontend in an efficient and non-blocking way [12].

It was decided that the team would use the SpringBoot framework because the framework massively simplifies the creation of APIs and persistence of objects, which will speed up this portion of the project.

This application will be packaged using Maven [13], which will also take care of artifact versioning. Maven will be used because it simplifies the process of managing project dependencies, allowing developers to specify the necessary external libraries and automatically download them. Nginx [14] will be used as the web server for this application as it is a widely used web server efficient in handling concurrent connections.

4.1.3 Database

A database is required to provide a way for doctors to log in to the doctor frontend. In a full production version of this application, the web interface would be allowed access to CPSO's internal database of doctors for authentication. However, since we do not have access, we will create a mock database of fake doctor identification data to validate as proof of concept for this login. To do this, the team has decided that Amazon DynamoDB will be used.

It was decided that the team would use Amazon DynamoDB because:

- It is a key-value NoSQL database, which allows the developers to interact with the information
- It is offered as part of the AWS free tier
- It integrates with Java SpringBoot repositories very easily.

4.1.4 Cloud

Instead of hosting backend services on personal computers, a cloud service will be leveraged, which provides access to a remote server. Using cloud services managed by third-party vendors ensures the web interface is always available and provides reliable performance. For this project, we will use AWS. Additionally, because of AWS' presence in multiple regions, a user's latency can be reduced dramatically [15]. Due to cost restrictions of this project, use of the cloud will be limited to the AWS free tier [16] which provides 5GB of Amazon S3 Bucket (for storing artifacts and images) [17] and 750 hours of certain sizes of Amazon Elastic Cloud Compute (EC2) Instances [18]. EC2 is essentially an AWS service that allows you to provision a VM. For testing purposes and to limit potential cost, local machines will be used as much as possible and AWS instances will be turned off when not in use.

It was decided that the team would use AWS because:

- The AWS free tier provides everything needed for a proof of concept for this project [16].
- AWS is the leading cloud provider used by professional developers [19] .

4.1.5 Containerization

Containerization refers to the packaging of software with its dependencies and a lightweight operating system to create a portable environment in which the software can run. This allows for testing in a uniform environment and makes it easier to deploy the software in different situations. Backend artifacts (jar files) will be used to create Docker images [20], using Docker Build [21] and Docker Compose [22] to streamline and automate the process. Docker compose refers to a configuration file that can be used to set up an image while Docker Build builds the image itself.

It was decided that the team would use Docker for the backend service because:

- It simplifies the deployment of the service to the cloud.
- It ensures that the environment in which our backend service is running is uniform, which reduces the amount of troubleshooting needed since each member of the team may have a different development environment.

A lightweight Linux base image such as Ubuntu Server LTS (22.04) [23] or Alpine Linux [24] will be used to reduce the size and cost of cloud hosting.

It was decided that the team would use a lightweight Linux base image because:

- Lightweight Linux base images boot up faster, speeding up deployment time.
- Lightweight Linux base images take up less storage, lowering the cost of operating a container registry to store versions.
- Although functionality can sometimes be limited by an image being more lightweight, this project does not require any functionality that cannot be provided by a lightweight Linux image.
- Alpine Linux was identified as an option as it is small, simple, and secure [25].
- Ubuntu Server was identified as another option as Ubuntu is the most widely used Linux distribution among professional developers [26].

4.1.6 Robotic Arm Control

The project will use a Yahboom robotic arm for Raspberry Pi 4B with a camera, to be purchased from Amazon [27]. This robotic arm will be controlled by the included Raspberry Pi using Python.

It was decided that the team would use the Yahboom robotic arm for Raspberry Pi 4B it was the only robot which [28]:

- Has a 6 degree of freedom robotic arm which provides more points of manipulability and a higher dexterity for irregularly shaped objects such as tongue depressor.
- Has a camera mounted on the robot
- Uses a Raspberry Pi 4B which was already accessible via the department, so we did not need to buy extra parts.
- Small enough to be non-intimidating to the end user.
- Pick up a load of up to five hundred grams, which is more than enough for our purposes.
- Supports Wi-Fi connectivity out of the box.
- Under 500 dollars.

4.2 Tools

4.2.1 GitHub (Version Control)

Git and GitHub will be used for version control for this project. GitHub is an online user interface to the tool, Git. This will allow proper reviewing of code using pull requests and be a useful tool for collaboration. There will be one repository with a folder for each component of the project.

It was decided that the team would use GitHub because:

- GitHub is the industry standard, as demonstrated by the large number of reputable software companies that use GitHub [29]
- Although there are alternatives to GitHub, such as Azure Git, it is less widely used and therefore has less readily available documentation to support development

4.2.2 Azure DevOps

Azure DevOps provides a simple and free platform for the creation of a project backlog, organization of ongoing tasks using boards (in our case a Kanban board) and tracking of progress using metrics. An Azure DevOps Kanban board has been set up for use during this project and an associated backlog has been started. To promote transparency, anyone hoping to get access to this backlog and board will be asked to create a free account and, upon request, will be granted stakeholder permissions.

It was decided that the team would use Azure DevOps because:

- Azure DevOps' backlog system provides more features and flexibility than other solutions such as Trello and GitHub Issues, although those two options could have worked just as well.

4.2.3 Continuous Integration

Continuous integration is defined as integrating code from multiple contributors to a single project [30]. In our project, there will be five contributors to a single repository. It is important to have a continuous integration pipeline for this project to ensure a working and symbiotic code collaboration process. Our team has decided that GitHub Actions [31] will be used for continuous integration. GitHub Actions is a tool to automate the building, testing, and deployment of a GitHub repository [32]. GitHub Actions is a free tool available from GitHub that runs infrastructure as code for all public projects and this will be leveraged to avoid unnecessary costs for our project [33]. GitHub Actions will allow automatic packaging, building of a Docker image [20], and deployment of our backend and frontend.

It was decided that the team would use GitHub Actions because:

- GitHub Actions integrates very well with GitHub.
- GitHub Actions provides useful templates for the deployment of an application to AWS [34].
- GitHub Actions is very easy to use, which will make it so that a continuous integration system can be established quickly and will require little configuration.

4.3 Diagrams

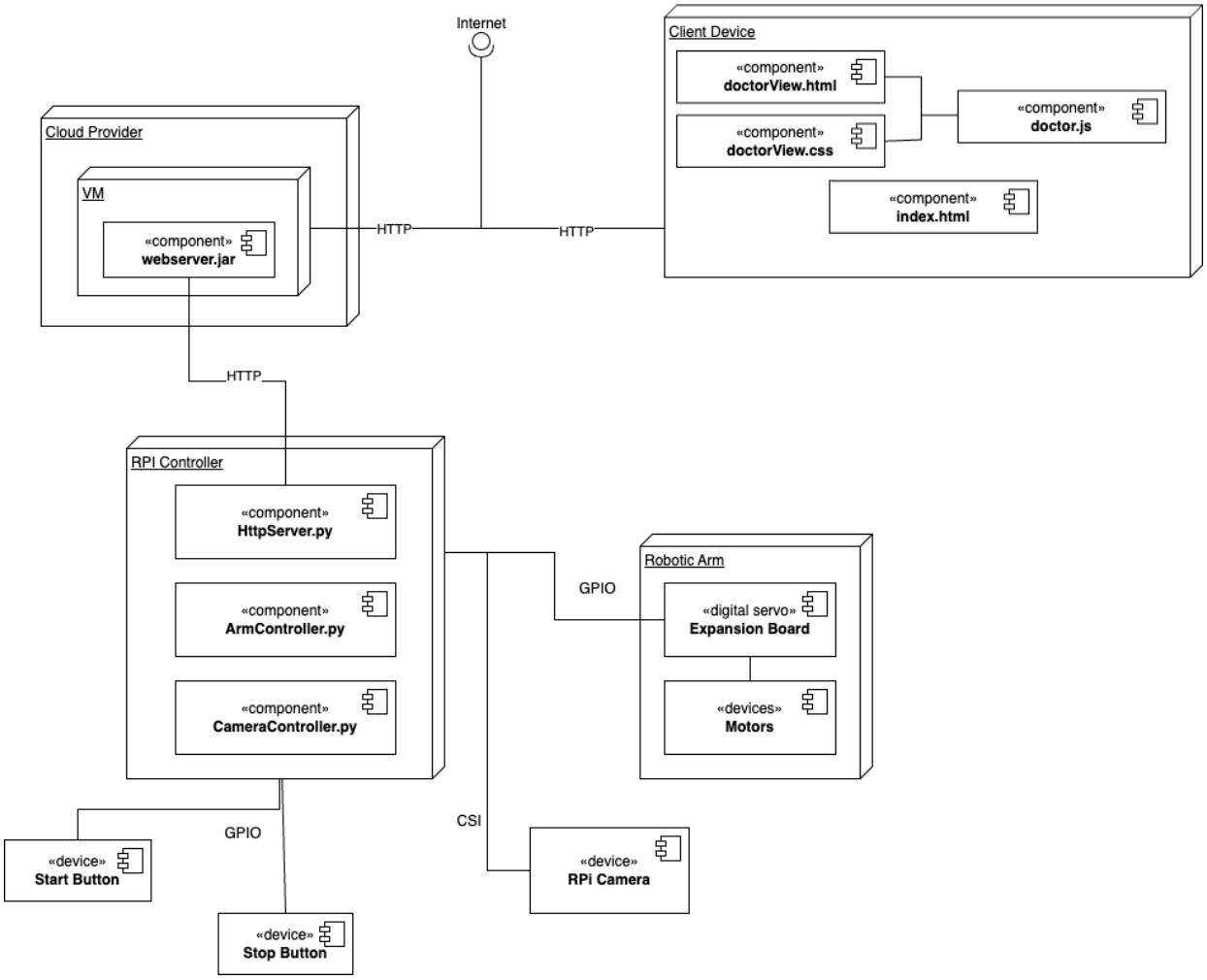


Figure 1: A deployment diagram visualizing the connection of our different components in the real world.

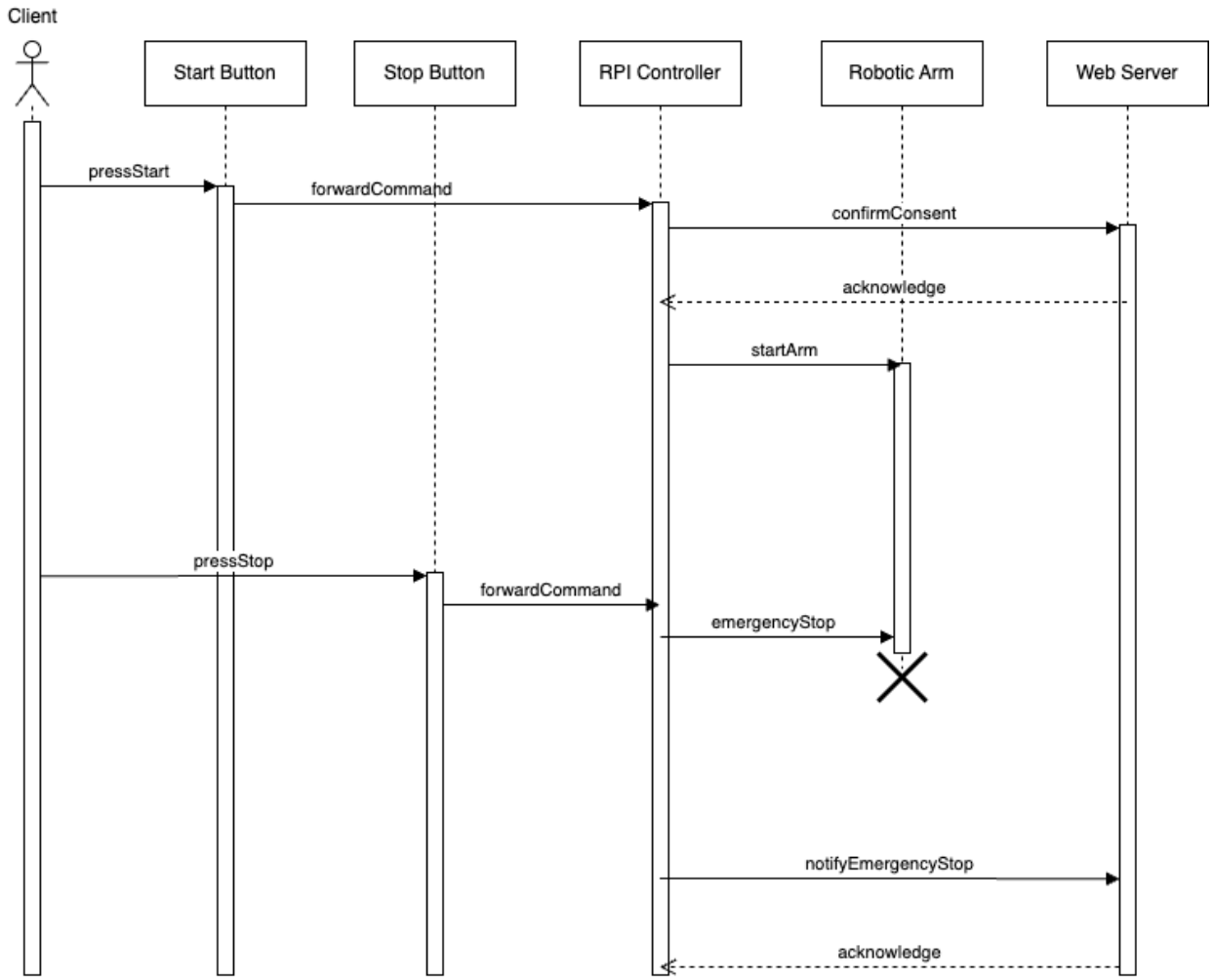


Figure 2: A sequence diagram visualizing the use cases of the patient (client).

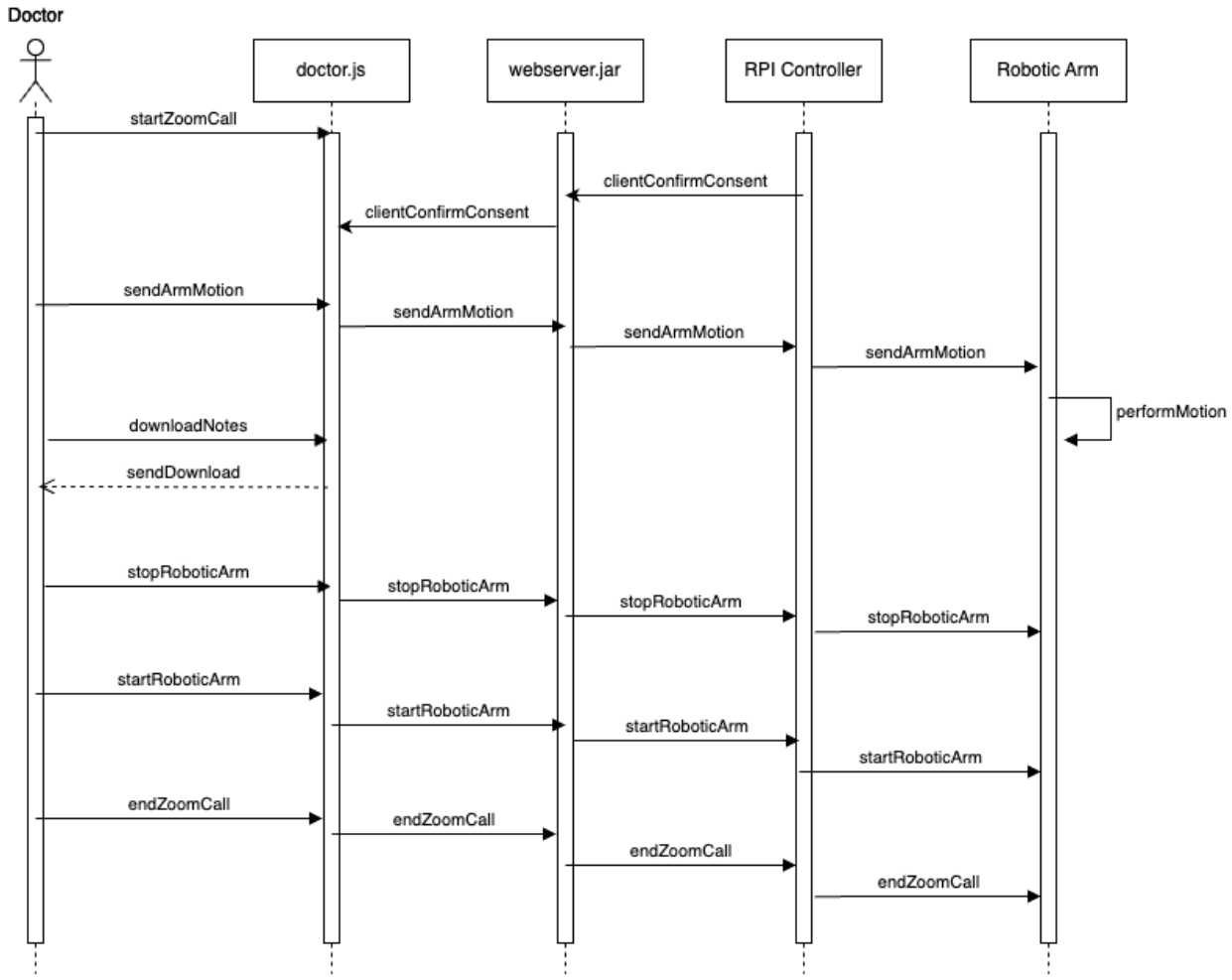


Figure 3: A sequence diagram visualizing the use cases of the doctor.

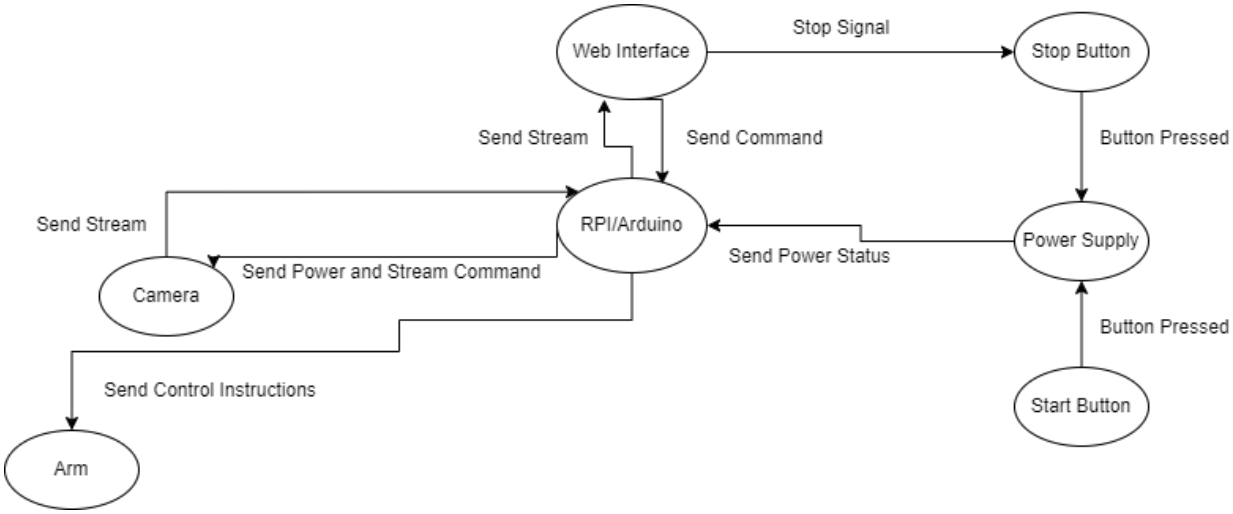


Figure 4: A high-level communication diagram of the robotic arm system interfacing with the web interface.

5 Timeline

The timeline of the project outlines the milestones, tasks, and approaches for the stages required for this project. The stages include problem and solution, identification, development, and testing. A Gantt chart detailing the expected start dates of each milestone, the expected duration of each milestone, and the percent done so far for each milestone.

5.1 Problem and Solution Identification

Milestone 1: Submit draft project proposal

Task 1.1 Research technologies of interest to use in the project

Task 1.2 Find a problem to solve and propose a solution

Task 1.3 Create project objectives

- Who are the stakeholders (end-users, target audience)

5.2 Research

Milestone 2: Submit project proposal

Tasks 2.1 Create milestones and tasks for project

Tasks 2.2 Create project functional requirements. This is done by:

- Defining the project scope and identifying stakeholders
- Define what the project should be able to do at the end

Tasks 2.3 Create project non-functional requirements

- Define stakeholder expectations of the project
    - Usability, accessibility, reliability, availability, etc.

Tasks 2.4 Create sequence diagrams

Tasks 2.5 Create a pros and cons list for technologies that can be used in the project.

- Consider the technologies that can be used and explain why a specific technology was used over another in a concise manner

Tasks 2.6 Research possible risks in the project and ways to mitigate

5.3 Development

Milestone 3: Create all static user interface pages

Task 3.1 Web application has a landing page that directs to a login page

Task 3.2 Web application has a login page for patients and doctors and distinguishes them

Task 3.3 Web application has the necessary button and screens

- The camera stream, doctor's notes, buttons to move the arm, buttons to pick up items (mallet, tongue depressor, and flashlight)
- Note: the buttons moving the robotic arm should not be functional at this point, just shown on the web application. This will be added later.

Milestone 4: Robotic Arm is Acquired

Task 4.1 Connect and test the robotic arm with board to see if all components are functional according to the specifications

Task 4.2 Write and test code to control each motor using only the Raspberry Pi and the robotic arm's associated components

Task 4.3 Wire the hard stop and start buttons. Test if the robotic arm powers on and off accordingly

Milestone 5: Acquire a mallet, tongue depressor, and flashlight

Task 5.1 Create using Fusion360, build or find a tray that fits the dimensions of each medical tool

Task 5.2 Establish/create a mount for the tray next to the robot so it's in a consistent spot for the robotic arm

Task 5.3 Write a preset routine for each medical device

Milestone 6: Establish camera live streaming from the robotic arm to Twitch

Task 6.1 Integrate the Zoom and Twitch application into the web application

Task 6.2 Have a button to send a Zoom link to patient's email

Milestone 7: Web application buttons can move the robotic arm

Task 7.1 Develop functionality for the web application buttons to move the robotic arm up/down, left/right and changing the depth

Task 7.2 Develop buttons to move the robotic arm in preset routines to pick up a mallet, tongue depressor and flashlight

Milestone 8: Distinguish between doctors and others for login

Task 8.1 Persist doctor data in a database, such as username and CPSO number, to be able to distinguish between doctor's that have been authenticated and anyone else trying to access the robotic arm controls

5.4 Testing

Milestone 9: Develop a test plan

Task 9.1 Develop an integration test plan

- Identify components that need testing
- Identify key scenarios that need to be tested

Task 9.2 Develop a unit test plan

- Identify components that need testing
- Identify key scenarios that need to be tested

Milestone 10: Develop a test suite

Task 10.1 Develop integration tests

- Integration testing will begin when at least one feature has been developed
- For every new feature, a new integration test will be added

Task 10.2 Develop unit tests

- Unit testing will begin when at least one feature has been developed
- For every new feature, a new unit test will be added

5.5 Project Deliverables

Milestone 11: Submit project progress report due December 8th

Task 11.1 Complete draft of progress report and request feedback from the supervisor on November 17th

Task 11.2 Make adjustments based on the feedback

Milestone 12: Oral presentation starting January 22nd to

Task 12.1 Submit the oral presentation form

Task 12.2 Create a script to present a working product

- Show the robotic arm moving according to the buttons pressed on the web application

Milestone 13: Submit final project report due April 10th

Task 13.1 Complete draft 1 of progress report and request feedback from the supervisor on February 16th

Task 13.2 Complete draft 2 of progress report by making adjustments based on the feedback from draft 1 and request feedback from the supervisor on March 14th

Task 13.3 Make final adjustments and based on the feedback from draft 2

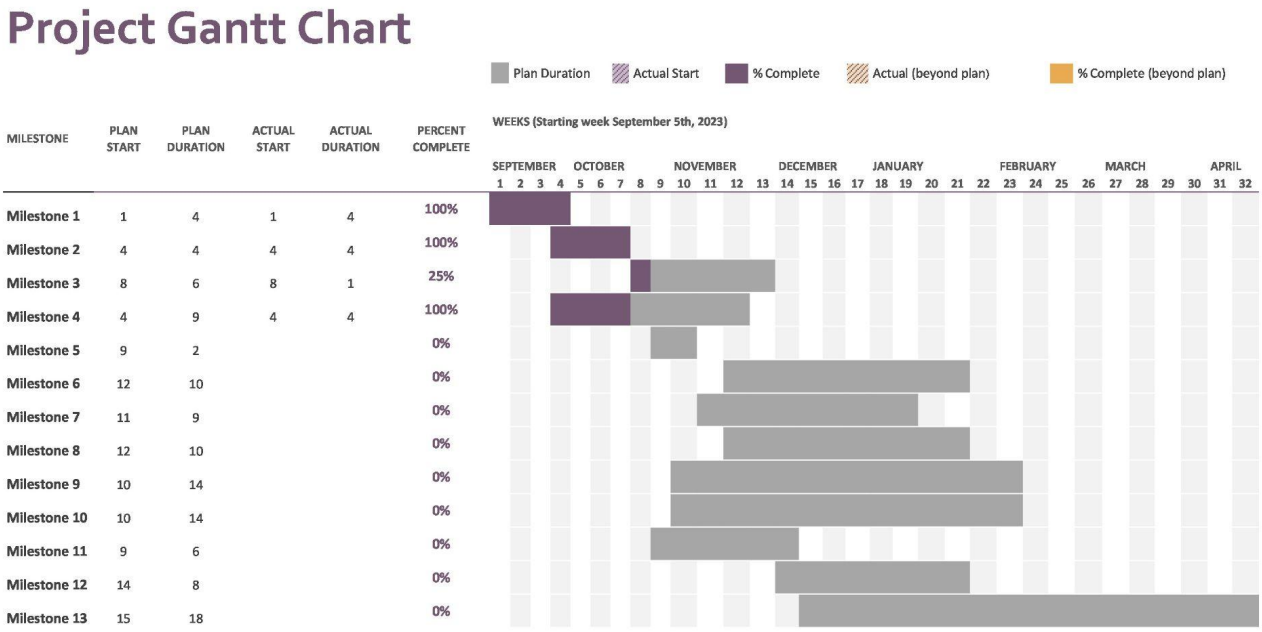


Figure 5: Gantt chart of the robotic arm project

6 Mitigation of Risks

Risk	**Mitigation Technique**
Dependency on a third party application, Zoom, for patient-doctor conferencing. Doctors' appointments involve the transferring of personally identifiable information, which is confidential.	To mitigate this risk a review of Zoom's privacy standards must be performed to determine if it is an acceptable solution for this system. If the conclusion is that Zoom is not secure enough, there are many other options for view conferencing software that is created specifically for this sort of application with security as a high priority.
Robotic Arm Malfunctions/or Patient feels uncomfortable	Patient will press hard stop button
Robotic arm is unable to pick up tool	Patent can put the robotic arm in the claw
Milestones are missed and the project is not able to be finished	Project features and scope can be reduced. The bot will then be focused on the mallet reflex tests. The mallet will then be the only available tool in this interaction of the project.
High latency when connecting to the Raspberry Pi leading to a greater than 3 second delay when maneuvering	Attempt to speed up infrastructure by connecting to the Raspberry Pi through a different method.
AWS free tier services end up being insufficient for the team's needs	The team will limit the use of AWS services and host the backend locally whenever possible.

Table 4: Project risks and mitigation.

7 Required Components and Services

For this project, the robotic arm that fits the project requirements (degrees of freedom and cost) the most is the Yahboom Robotic Arm for Raspberry Pi 4B AI Smart Robotic Arm with Camera. This robot kit will include 6 servo motors and a camera mounted on the 5th linkage. The cost of this robot kit is $430 without tax and shipping. The Raspberry Pi 4B is provided by the school at no additional cost.

Component	**Voltage**	**Current Limit**	**Communication Protocol(s)**	**Description**
Raspberry Pi 4B	3.3 or 5V output	1-2A output	GPIO/ I2C/ CSI/ UART	Handles the data from the camera and sends instructions to motors. Communicates directionally to and from the web application
Yahboom Robotic Arm for Raspberry Pi 4B	100 - 240 V (50/60 Hz) AC to 12V 5A			The robotic arm consists of 6 servo motors and 1 Raspberry Pi Camera Module V2 and a full-function expansion board which interfaces the Raspberry Pi, power supply and arm
Digital Servo Motor (x4) - Torque 15KG Angle 1- 180 degrees	Not Specified	Not Specified	GPIO	To control and read each degree of freedom for the bottom 4 motors.
Digital Servo Motor (x4) - Torque 6KG Angle 1- 180 degrees	Not Specified	Not Specified	GPIO	The motor to control the “open and close” function of the gripper for the robotic arm
Digital Servo Motor (x1) - Torque 15KG Angle 1- 270	Not Specified	Not Specified	GPIO	The motor rotates the gripper from 1-270 degrees.

degrees				
Raspberry Pi Camera Module V2	3.3V	2A	CSI	Camera streams the video back to the camera in 1080p. It is attached to the 5th linkage on the Robotic Arm and is included in the kit.

Table 5: List of hardware components.

References

[1] "Your Health Plan: Connected and Convenient Care," Ontario.ca. [Online]. Available: https://www.ontario.ca/page/your-health-plan-connected-and-convenient-care. Accessed on: October 24, 2023.

[2] Fact sheet: Ontario's doctor shortage - oma, https://www.oma.org/uploadedfiles/oma/media/public/hcp-factsheet-doctor-shortage.pdf. Accessed on: October 25, 2023.

[3] A. Naceri et al., "Tactile robotic telemedicine for safe remote diagnostics in times of Corona: System design, feasibility and usability study," IEEE robotics and automation letters, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9454265/. Accessed on: October 25, 2023.

[4] R. Van Loon et al., Laboratory Health and Safety Manual, https://sce-soft-web-ist.sce.carleton.ca/wp-content/uploads/2023/07/health-and-safety.pdf.

[5] Carleton University, "ECOR 4995 - Undergraduate Academic Support," [Online]. Available: https://carleton.ca/engineering-design/current-students/undergrad-academic-support/ecor-495/. Accessed on: October 24, 2023.

[6] "Professional Engineers Ontario (PEO) Guideline," Professional Engineers Ontario, [Online]. Available: https://www.peo.on.ca/sites/default/files/2020-12/PEPGuideline_Nov2020.pdf. Accessed on: October 24, 2023.

[7] "Ontario Regulation 900/94 - PROFESSIONAL ENGINEERS ACT," Ontario.ca, [Online]. Available: https://www.ontario.ca/laws/regulation/900941. Accessed on: October 24, 2023.

[8] "Agile Manifesto - 12 Principles," Agile Manifesto, [Online]. Available: https://Agilemanifesto.org/principles.html. Accessed on: October 24, 2023.

[9] "Kanban - Atlassian," Atlassian, [Online]. Available: https://www.atlassian.com/Agile/Kanban#:~:text=In%20Japanese%2C%20Kanban%20lite

ally%20translates,in%20a%20highly%20visual%20manner. Accessed on: October 24, 2023.

[10] J. Patadiya, “React vs react native - key difference, features, advantages,” Radixweb, Available: https://radixweb.com/blog/react-vs-react-native. Accessed October 24, 2023.

[11] N. Raval, "React vs Angular: Which JS Framework to Pick for Front-end Development?," Radixweb. Available: https://radixweb.com/blog/react-vs-angular. Accessed on: October 24, 2023.

[12] "Spring Framework - Reactive Web Framework," Spring Framework Documentation, [Online]. Available: https://docs.spring.io/spring-framework/reference/web/webflux/new-framework.html. Accessed on: October 24, 2023.

[13] "Apache Maven - Welcome to Apache Maven," Apache Maven, [Online]. Available: https://maven.apache.org/index.html. Accessed on: October 24, 2023.

[14] "NGINX Documentation - Web Server," NGINX, [Online]. Available: https://docs.nginx.com/nginx/admin-guide/web-server/. Accessed on: October 24, 2023.

[15] "Reduce Latency for End Users with Multi-Region APIs with CloudFront," Amazon Web Services (AWS), [Online]. Available: https://aws.amazon.com/blogs/networking-and-content-delivery/reduce-latency-for-end-users-with-multi-region-apis-with-cloudfront/. Accessed on: October 24, 2023.

[16] "AWS Free Tier - Amazon Web Services (AWS)," Amazon Web Services, [Online]. Available: https://aws.amazon.com/free/. Accessed on: October 24, 2023.

[17] "Amazon S3 - Simple Storage Service," Amazon Web Services, [Online]. Available: https://aws.amazon.com/s3/. Accessed on: October 24, 2023.

[18] "Amazon EC2 - Elastic Compute Cloud," Amazon Web Services, [Online]. Available: https://aws.amazon.com/ec2/?did=ft_card&trk=ft_card. Accessed on: October 24, 2023.

[19] M. Zhang, “Top 10 cloud service providers globally in 2023,” Dgtl Infra, https://dgtlinfra.com/top-cloud-service-providers/#:~:text=Amazon%20Web%20Services20(AWS)%2C%20the%20cloud%20computing%20service%20of,compute%2C%20storag%2C%20and%20database. Accessed on October 25, 2023.

[20] “Docker images,” Docker Documentation, https://docs.docker.com/engine/reference/commandline/images/. Accessed on October 25, 2023.

[21] "Docker Documentation - Build Images," Docker, [Online]. Available: https://docs.docker.com/build/. Accessed on: October 24, 2023.

[22] "Docker Documentation - Compose," Docker, [Online]. Available: https://docs.docker.com/compose/. Accessed on: October 24, 2023.

[23] "Docker Official Images - Ubuntu," Docker Hub, [Online]. Available: https://hub.docker.com/_/ubuntu. Accessed on: October 24, 2023.

[24] "Docker Official Images - Alpine," Docker Hub, [Online]. Available: https://hub.docker.com/_/alpine. Accessed on: October 24, 2023.

[25] "About - Alpine Linux," Alpine Linux, [Online]. Available: https://www.alpinelinux.org/about/#:~:text=Alpine%20Linux%20is%20built%20around,of%20packages%20from%20the%20repository. Accessed on: October 24, 2023.

[26] "Stack Overflow Developer Survey 2023 - Most Popular Technologies," Stack Overflow, [Online]. Available: https://survey.stackoverflow.co/2023/#technology-most-popular-technologies. Accessed on: October 24, 2023.

[27] Yahboom robotic arm for Raspberry Pi 4b AI smart robotic ... - amazon.ca, https://www.amazon.ca/Yahboom-Robotic-Raspberry-%EF%BC%88DOFBOT-Without/dp/B08T8XG2J6. Accessed on Oct. 25, 2023.

[28] "DOFBOT-Pi - Yahboom Robotics," Yahboom, [Online]. Available: https://category.yahboom.net/collections/rp-robotics/products/dofbot-pi. Accessed on: October 24, 2023.

[29] "Companies using Linked Data in 2023," GitHub, [Online]. Available: https://github.com/d2s/companies/blob/master/src/index.md. Accessed on: October 24, 2023.

[30] "Continuous Delivery and Continuous Integration," Atlassian, [Online]. Available: https://www.atlassian.com/continuous-delivery/continuous-integration. Accessed on: October 24, 2023.

[31] "GitHub Actions Documentation," GitHub, [Online]. Available: https://docs.github.com/en/actions. Accessed on: October 24, 2023.

[32] "GitHub Actions Documentation - Understanding GitHub Actions," GitHub, [Online]. Available: https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions. Accessed on: October 24, 2023.

[33] "GitHub Actions Documentation - Usage Limits, Billing, and Administration," GitHub, [Online]. Available:https://docs.github.com/en/actions/learn-github-actions/usage-limits-billing-and-administration. Accessed on: October 24, 2023.

[34] "GitHub Actions Starter Workflows - AWS Deployment Workflow," GitHub, [Online]. Available: https://github.com/actions/starter-workflows/blob/main/deployments/aws.yml. Accessed on: October 24, 2023.

[35] “Carleton University Calendars 2023-24 edition,” Systems and Computer Engineering (SYSC) < Carleton University, https://calendar.carleton.ca/undergrad/courses/SYSC/. Accessed on October 25, 2023.

Glossary

Amazon DynamoDB: a fully managed NoSQL database service provided by AWS.

Amazon S3 Bucket: Simple Storage Service (S3) is a scalable, secure, and highly available cloud storage service provided by AWS designed to store and retrieve any amount of data from anywhere on the web.

Angular: Typescript-based, free, and open-source single-page web application framework based upon the Model-View-Controller design pattern.

Application Programming Interface (API): a set of rules and protocols that allows different software applications to communicate with each other.

Artifact Versioning: the practice of assigning unique identifiers or labels to software artifacts, enabling tracking and control of changes in software development and deployment.

Auscultation: the action of listening to sounds from the heart, lungs, or other organs, typically with a stethoscope, as a part of medical diagnosis.

Degrees of Freedom: the number of independent variables that define the possible motions of a mechanical system.

Docker: a set of platform as a service products that use operating system level virtualization to deliver software in packages called containers.

Document Object Model (DOM): a programming interface that provides a structural representation of HTML and XML documents, allowing scripts to dynamically access, modify, and update the content, structure, and style of web pages

LAN: acronym for Local Area Network. Refers to a collection of connected connected in one physical location.

LTE: acronym for Long-Term Evolution. Refers to a standard for wireless broadband communication.

Maven: a build automation tool predominantly used for Java projects. It simplifies the building and management of Java-based applications.

Nginx: a web server that can also be used as a reverse proxy, load balancer, mail proxy, and HTTP cache.

Persistence of Objects: the ability to save the state of an object and retrieve it at a later time.

Raspberry Pi: a single-board computer designed for various computing tasks, encompassing programming and hardware prototyping.

React: open-source JavaScript library, created by Meta, allowing for the efficient creation of interaction user interfaces.

React Native: an open-source framework that enables the creation of native mobile applications using JavaScript and React.

SpringBoot: open-source Java-based framework that simplifies the development of production-grade, stand-alone, and web-based applications.

Telediagnosis: refers to a remote diagnosis.

Telemedicine: describes an area of medicine in which medical care is not provided directly but over a more or less greater distance.

Telemonitoring: refers to using different types of technology to monitor patients at a distance.

Twitch: a video live streaming service that focuses on video game live streaming, including broadcasts of esports competitions, in addition to offering music broadcasts, creative content, and "in real life" streams.

Wi-Fi: a wireless networking technology that uses radio waves to provide high-speed internet access.

Zoom: Zoom is a communications platform that allows users to connect with video, audio, phone, and chat.

Appendix A

Mock Frontend Design

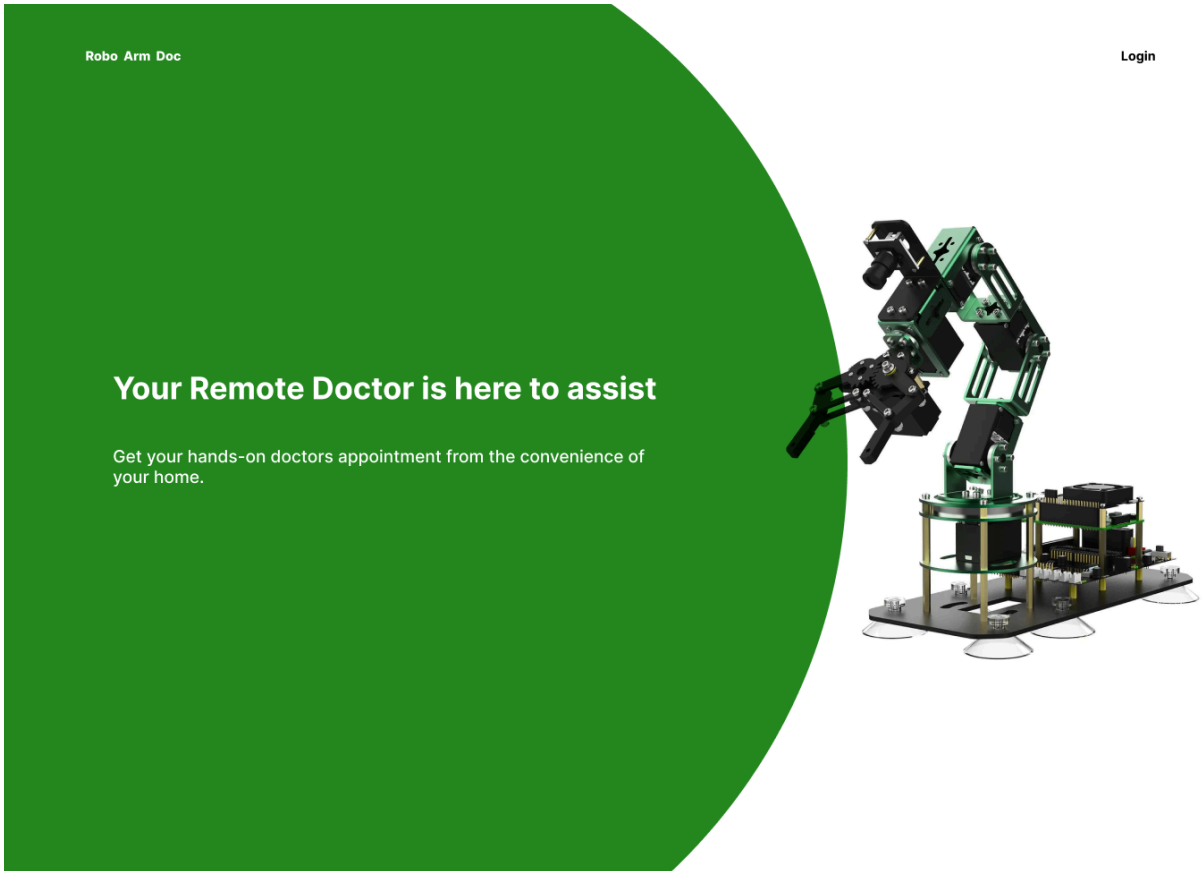


Figure 6: The Figma mockup for the website landing page.

Figure 7: The Figma mockup for the website sign-in portal.

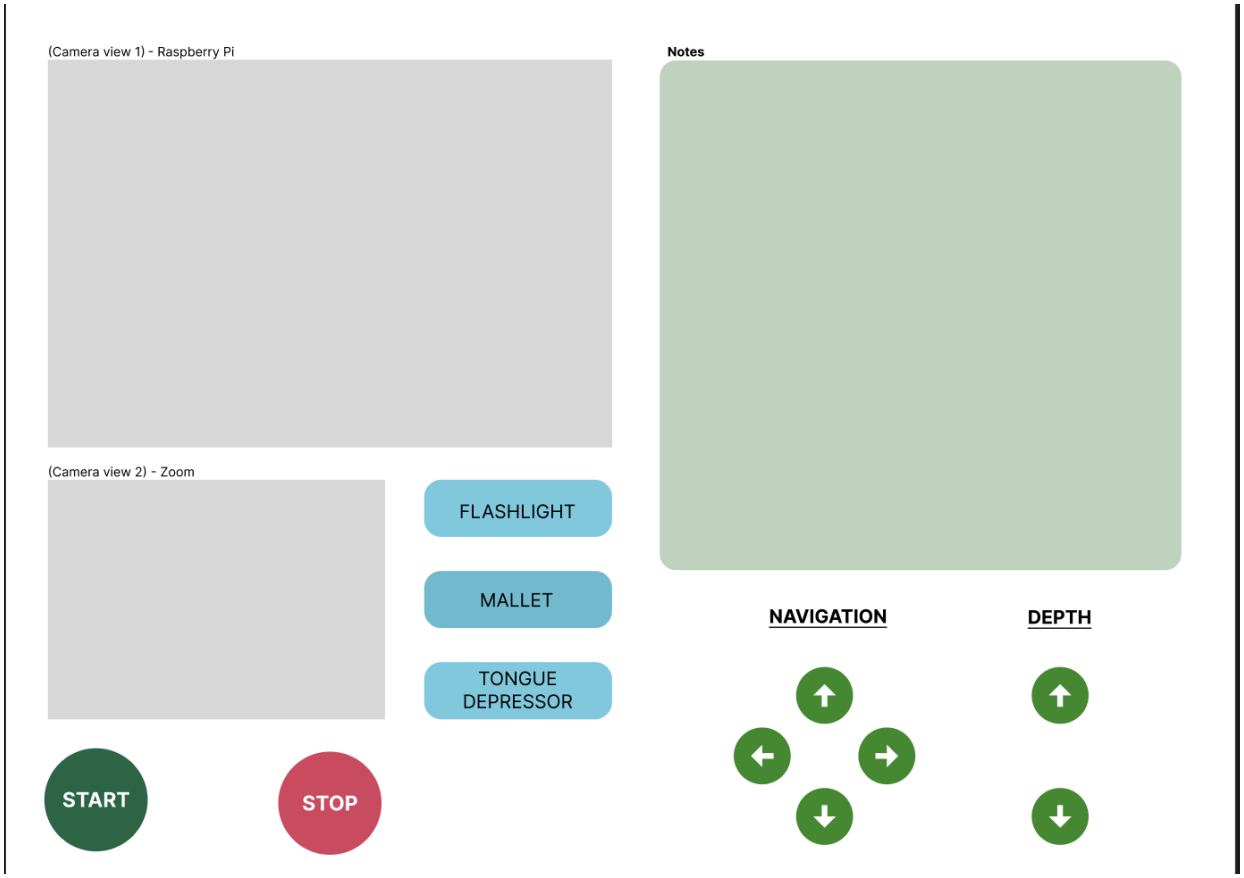


Figure 8: Doctor view of healthcare web application

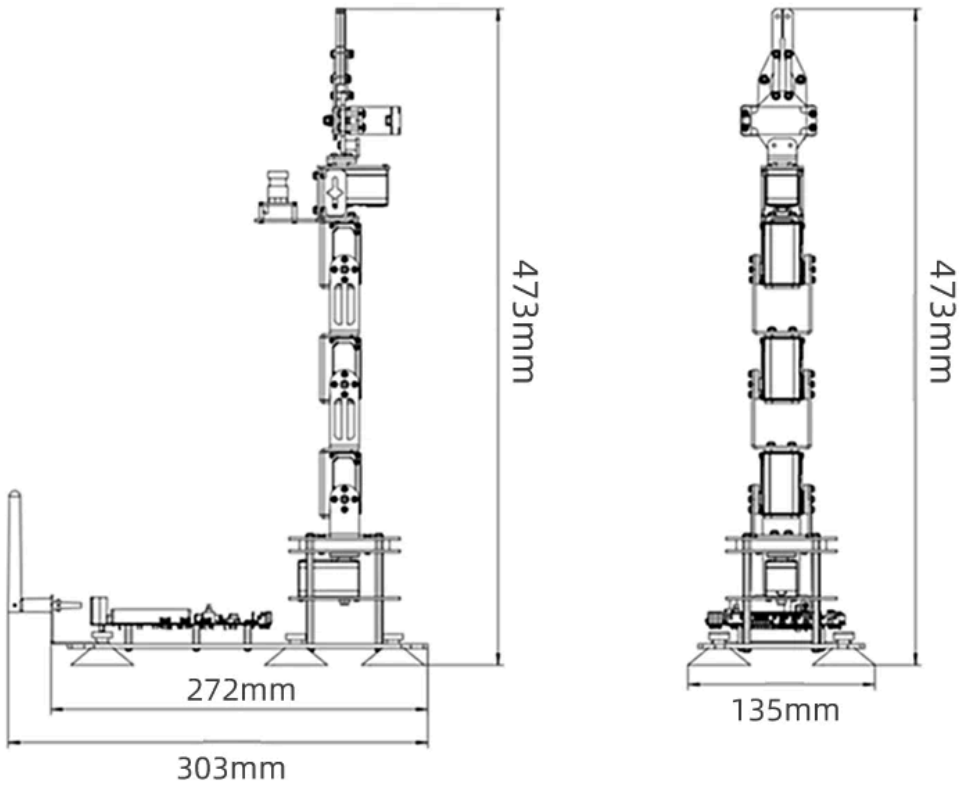


Figure 9: Robotic arm schematic (orthographic) [28]

Appendix B

Course Descriptions

Courses	Course Description
ECOR 1051: Fundamentals of Engineering I	Software development as an engineering discipline, using a modern programming language. Tracing and visualization of program execution. Testing and debugging. Data management: digital representation of numbers; numerical algorithms; storing data in files; container data types: sequences, sets, maps.
SYSC 2004: Object-Oriented Software Development	Designing and implementing small-scale programs as communities of collaborating objects, using a dynamically-typed or statically-typed programming language. Fundamental concepts: classes, objects, encapsulation, information hiding, inheritance, polymorphism. Iterative, incremental development and test-driven development.
SYSC 3110: Software Development Project	Development of expertise in designing, implementing and testing maintainable, reusable software through team projects. Applying modern programming languages, design patterns, frameworks, UML and modern development processes (detection of olfactible source code defects, refactoring, iterative and incremental development, version control techniques) to medium-scale projects.
SYSC 3120: Software Requirements Engineering	Current techniques, notations, methods, processes and tools used in Requirements Engineering. Requirements elicitation, negotiation, modeling requirements, management, validation. Skills needed for Requirements Engineering and the many disciplines on which it draws. Requirements analysis: domain

	modeling, modeling object interactions; UML modeling. Introduction to software development processes.
SYSC 3303: Real-Time Concurrent Systems	Principles and practice of a systems engineering approach to the development of software for real-time, concurrent, distributed systems. Designing to achieve concurrency, performance, and robustness, using visual notations. Converting designs into programs. Introduction to hard real-time systems. Team project.
SYSC 3310: Introduction to Real-Time Systems	Principles of event-driven systems. Microcontroller organization. Development of embedded applications. Programming external interfaces, programmable timer. Input/output methods: polling, interrupts. Real-time issues: concurrency, mutual exclusion, buffering. Introduction to concurrent processes.
SYSC 4120: Software Architecture and Design	Introduction and importance of software architectures and software system design in software engineering. Current techniques, modeling notations, methods, processes and tools used in software architecture and system design. Software architectures, architectural patterns, design patterns, software qualities, software reuse.
SYSC 4805: Computer Systems Design Lab	Developing professional-level expertise in selected, important areas of the field by applying, honing, integrating, and extending previously acquired knowledge in team projects in the laboratory. Lecture periods are devoted to new knowledge required for the selected areas, to project-related issues, and to student presentations.
SYSC 4806: Software Engineering Lab	Applying the full spectrum of engineering and programming knowledge acquired in the program through team projects in the laboratory. Practice in doing presentations and reviews. Lectures will discuss software engineering issues as they relate to the projects, from a mature point of view.

Table 6: Course descriptions taken directly from the course outlines [35]