

A3 Final Report

S23M Health Informatics Project

ROMA Development

Riley Head (n10540717)
Max Goodwin (n10084444)
Mackenzie Smith (n10233644)
Rodo Nguyen (n10603280)

November 6th, 2022

IFB399 IT Capstone Phase 2

Contents

1	Introduction	1
1.1	Project Context, Goals, Scope	1
1.2	Project Outcomes and Success	2
2	Project Setup	3
2.1	Project Management Approach	3
2.2	Client Expectations	3
2.3	Team Collaboration	4
2.4	Communication Plan	5
3	Project Plan and Risk	6
3.1	Project Planning and Progress	6
3.2	Risk Management	9
4	Artefact Description	10
4.1	Functionality	10
4.2	Architecture	13
4.3	Technical Description: JavaScript Application	14
4.4	Technical Description: C# application	19
4.5	Quality	23
5	Individual Chapter: Riley Head	25
5.1	Project Setup	25
5.1.1	Project Management Approach	25
5.1.2	Client Expectations	25
5.1.3	Team Collaboration	26
5.1.4	Communication Plan	26
5.2	Project Plan and Risk (Phase 2)	27
5.2.1	Project Planning and Progress	27
5.2.2	Risk Management	28
5.3	Project Experience	28
6	Individual Chapter: Max Goodwin	30

6.1	Project Setup	30
6.1.1	Project Management Approach	30
6.1.2	Client Expectations	30
6.1.3	Team Collaboration	31
6.1.4	Communication Plan	31
6.2	Project Plan and Risk (Phase 2)	32
6.2.1	Project Planning and Progress	32
6.2.2	Risk Management	32
6.3	Project Experience	33
7	Individual Chapter: Mackenzie Smith	35
7.1	Project Setup	35
7.1.1	Project Management Approach	35
7.1.2	Client Expectations	35
7.1.3	Team Collaboration	36
7.1.4	Communication Plan	36
7.2	Project Plan and Risk (Phase 2)	37
7.2.1	Project Planning and Progress	37
7.2.2	Risk Management	37
7.3	Project Experience	38
8	Individual Chapter: Rodo Nguyen	39
8.1	Project Setup	39
8.1.1	Project Management Approach	39
8.1.2	Client Expectations	40
8.1.3	Team Collaboration	40
8.1.4	Communication Plan	41
8.2	Project Plan and Risk (Phase 2)	41
8.2.1	Project Planning and Progress	41
8.2.2	Risk Management	42
8.3	Project Experience	42
9	Appendix	44

1 Introduction

Our client, S23M has been operating in the Australian and New Zealand region for 20 years offering a variety of services for clients including, but not limited to: enterprise SaaS, innovation and product development and creative collaboration. They have provided these services for a range of industries such as, healthcare, construction and logistics. S23M aims to accelerate innovation and stimulate ethical change in organisations by bringing expertise and creating new insights.

1.1 Project Context, Goals, Scope

In October 2021, S23M co-sponsored a new lab in New Zealand to support the Asia eHealth Information Network (AeHIN) in its mission to accelerate digital health development in the Asia Pacific region. ROMA development has been brought in to assist S23M and the Aotearoa NZ Interoperability Lab (ANZIL) to explore healthcare interoperability based on the HL7 Fast Healthcare Interoperability Resources (FHIR) and International Patient Summary (IPS) open source standards. Through careful consultation with S23M, the ROMA development team has agreed to develop an application to read a variety of test patient data from a server. The data read is chosen according to the IPS, which is split into 4 categories:

- Header,
- Required,
- Recommended,
- Optional.

Figure 1 below shows the type of data contained in the above categories.

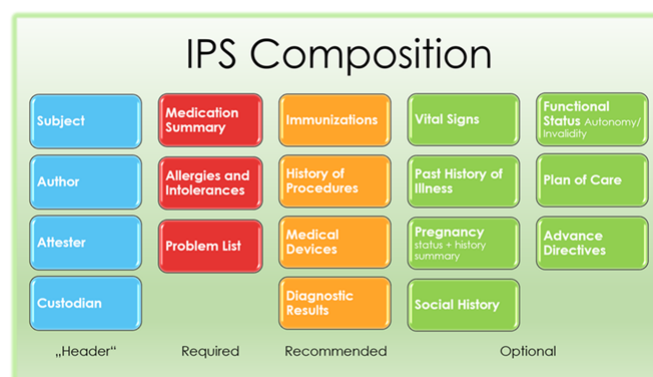


Figure 1: IPS Composition

As part of the ANZIL labs mission, S23M wants this project to educate current and future developers on how to read components of the IPS from a database and process them into a viewable format. ROMA development has created the application in two programming languages; C# and JavaScript. These were chosen at the client's request but also because the team is well-versed in these languages. Additionally, a set of tutorial pages has been written to explain how certain aspects of the code function. It is assumed that the audience has a solid understanding of C# or JavaScript so the tutorials only cover FHIR and IPS specific code. The tutorials are written and hosted on S23M's Notion site who will then use this as required to support the deployment of our application.

1.2 Project Outcomes and Success

After 11 weeks of development the team has been able to produce two applications that both share the same functionality. These applications are able to retrieve patient health data from a test server and display it in the relevant format. The team also delivered detailed written tutorials that explain and show how the code that was developed functions so that future developers can learn, replicate and improve it in the future.

The simple goal of this project was to educate developers as part of ANZIL's mission to accelerate digital health development in the Asia Pacific region. One of the largest components of interconnected health system is the reading and transferring of patient health data between hospitals within a country but also across borders. Users of our web application can clearly see how to search for patients and display health data related to those patients according to the International Patient Summary (IPS). This is an important foundation for establishing a well connected health system.

The tutorials that have been written to complement our code make our application more useful as an educational tool. Users can first browse our tutorial pages to get a high level overview of the application's functionality and because our tutorials reference the code; they can then go to the code and see it as part of the wider application. This follows the format that most other IT educational resources, such as W3 schools use to teach new concepts.

Although writing in two languages limited the amount of functionality that could be included; the applications are seen as being more successful as an educating tool due to the simple fact that it will appeal to more developers because we can't assume that every program will be written in just one language. It can be comfortably said that this project has been a success and will go a long way in supporting the development of the Asia eHealth Information Network and the wider business goals of our client S23M.


2 Project Setup

2.1 Project Management Approach

ROMA Development utilised an Agile style of project management. This was suitable due to the nature of our development project (creating a web application). Features could be easily divided into sprints that had a 2-week length. The 2-week length was chosen as it aligned with our fortnightly meetings where we could show the results of the previous sprint and then discuss the current sprint in the same meeting but also because the tasks could be completed in that time frame.

In semester 1, roles were given to members of the ROMA development team. After a quick discussion at the beginning of semester 2, it was decided that the roles would remain the same apart from splitting the team into two development teams. One for C# and one for JavaScript. See figure 2.

Team Roles & Responsibilities

 Created by Riley Head
Aug 02, 2022 • 1 min read •  Signatures: 3

Name	Role\s
Riley Head	Project Manager, Developer (C#)
Max Goodwin	Developer (JavaScript)
Rodo Nguyen	Developer (JavaScript)
Mackenzie Smith	Developer (C#), Client Relations

Figure 2: Semester 2 Team Roles

It was important that the team used a well tested and effective project management tool so that the Agile process could be managed well. For these reasons the team selected Atlassian Jira as the project management tool. Using this tool allowed us to create tasks and allocate them to sprints and members of the team. Furthermore, as the tool is hosted in the cloud we invited our client so they could get a high level progress overview at any point in time. See appendix item 1 for an example of one of these tasks.

2.2 Client Expectations

Our client S23M having participated in the QUT Capstone program previously had a good idea of what we were able to produce in the given time frame. Over the course of the project we dealt with one person

on the client side. As we were working for the client, their role was to give us as much information as possible as to what they wanted developed. As a team we asked the right questions but our client was able to provide us with examples and other material to help us understand what we were going to build. As this project was health focused our team experienced a steep learning curve as none of us had any experience in this field. The client made sure to provide us with enough material to understand the health specific terminology but also explained what the application should look like in the end.

Knowing what to build was good, but our client went one step further and gave us a reason and a purpose. We were told that this application and supporting material would go a long way in helping less developed countries improve their health service. Knowing this made the project extra special and rewarding to work on.

To manage expectations it was important to show the client our progress on a frequent basis. Our fortnightly meetings went for 1 hour where the first 30 minutes would be for us to demonstrate what had been completed since the previous meeting and the last 30 minutes was for both the client and our team to ask questions of each other. We would conclude these meetings by informing the client of what we were moving onto next and this meant he could send us any resources and give us some initial guidance if required.

Despite the project having 2-week sprints, there was no requirement for us to provide a release at the conclusion of each sprint. The goal was to have everything complete by about week 11. On Friday of week 11 (14 October 2022) we handed over the code and tutorials to client in addition to meeting with them in person and discussing over lunch. The following weeks allowed the client to review our work as well as testing it on their end. It also meant that we had 2 weeks to make any last minute changes at the clients request.

2.3 Team Collaboration

All members of our team had the same goal in mind which was to do well in this unit. Therefore, our work culture was very good. Completing this project was not a case of delivering an ok product but going that extra step to deliver a great product. There was no need for a social contract as we had proved to ourselves that we could well together during semester 1. As a team we would try to meet weekly however, there were some weeks that did not require us to meet. The purpose of these meetings was to:

- Make sure everyone knew what they were working on,
- Address and roadblocks the team hit and ensure the project is on track,
- and discuss anything the client may have brought up in meetings.

Our decision making process depended on the decision that needed to be made. Each development team had the capacity to make decisions pertaining to their specific project. Decisions that were likely to have an affect on both applications were made as a team and there was never a case where the project manager made a decision without the rest of the teams input.

As mentioned previously Jira was used as our project management tool and Confluence for housing any documents. These two pieces of software are industry recognised and have been developed for the

purpose of making project management easier and more efficient. Jira was set up to compliment the Agile project management methodology. We had a backlog where we created tasks that would end up being organised into our sprints. When activated, each sprint would have a board with 4 buckets that tasks could be placed in. These can be seen in appendix item 2.

Confluence was used mainly to create and store documents associated with the tutor meetings. You can see in appendix item 3 for an example of a Confluence document for one of the tutor meetings. The document clearly outlines what the team is discussing and who is discussing it. This helped everyone get prepared for the meeting so it ran smoothly and we could get through more in the allocated time. These documents are provided in the folder labelled 'project' that accompanies this report.

2.4 Communication Plan

A comprehensive communication plan for this project was made to better manage communication amongst the stakeholders: Development team, Project manager, Product owner and QUT tutor. Formally defining the communication methods allowed us to work efficiently and choose the right method of communication in any given scenario. This plan can be seen below in figure 3

Notes	
Development team	ROMA's development team
Project manager	ROMA's project manager
Product owner	S23M supervisor

Communication Plan						
Method	Who	How	When	What	Why	Responsible
Project status board	Development team, Project manager, Product owner	Jira Board	Throughout project duration	List of tickets that are in "to do", "in progress", "in review" and "done" categories. Ticket's status is updated by development team to reflect the project progress.	Monitor backlog and project's progress	Development team, Project manager
Client meetings	Development team, Project manager, Product owner	Discord	Fortnightly, even weeks of the semester	Update Project owner of the project's progress and the next sprint's plan.	Update project's progress, ensure it's on track	Development team, Project manager
Team meetings	Development team, Project manager	Discord	Fortnightly, even weeks of the semester	Discuss progress and obstacles on the technical side, prepare for tutor meetings.	Share information in ROMA team, offer support	Development team
Announcements, Instant concerns	Development team, Project manager, Product owner	Email, Jira, Discord	Throughout project duration	(Email) General project discussion, (Jira, Discord) quick discussion on emerging issues, broadcast project news.	Resolve small issues/Update information quickly	Project manager, Product owner, Development team
Project Documentation	Development team, Project manager	Shared OneDrive, Confluence	Throughout project duration	(OneDrive) Store important documents, (Confluence) Fortnightly report	Ensure important documents are stored and shared properly	Development team
Optional meetings	Development team, Project manager, Product owner	Discord	Fortnightly, odd weeks of the semester	Same purposes as Client and Team's meetings, the time slot is reserved for any cancelled meeting or urgent discussion	Reserve timeslot for unexpected occasion	Project manager, Product owner, Development team
Tutor meetings	Development team, Project manager, QUT tutors	Zoom	Fortnightly, even weeks of the semester	Each member of ROMA team takes turn to report their work, roadblocks, upcoming plan. After that, tutor can provide feedback and further guidance	Provide an overview of the project, ensure major roadblocks are resolved	QUT tutor

Figure 3: Communication Plan

3 Project Plan and Risk

3.1 Project Planning and Progress

In Phase 1, ROMA focused on 3 objectives: 1) Understanding the client's needs, 2) Identifying suitable deliverables and 3) Implementing a demo product.

With a greater understanding of the topic, the aim of Phase 2 was to accelerate and optimize artefact development. In the first week of semester 2 we met with our liaison at S23M to make a plan for our team. During this initial meeting he informed us that the scope of the project had changed quite a bit from where we left it at the end of Phase 1. The project was initially a set of individual tutorials covering development using the FHIR standard. The updated artefact was two independent web applications in different programming languages with accompanying tutorials. These applications needed to showcase accessing and displaying FHIR standard data (patient info, medication etc. see figure 1) from an appropriate test server. These applications would be the source material for creating tutorials to help developers new to the standard.

As the clients expectations had changed, the sprint schedule and stages developed in phase 1 were now redundant. Due to the nature of the project and the decisions of our client, there was no requirement to deliver specific releases, but rather proof of notable progress. Our team still decided to create a release plan (see figure 4) to plan our work and keep each other accountable.

Release	Title	Description	Length
1	Project Planning and Choosing Program Structure	<ul style="list-style-type: none"> Communicate with the client, ensuring the plan created in phase 1 is still suitable, making changes where necessary For both <i>C#</i> and <i>JavaScript</i>, research and set up a development environment with appropriate frameworks and version control 	1 Sprint
2	Search and Required Components	<ul style="list-style-type: none"> Create a webpage to implement and showcase the search functionality. Displaying all relevant patient info Display required components attached to patients (Medication Summary, Allergies and Intolerances, Problem List) 	1 Sprint
3	Recommended Components and Tutorials	<ul style="list-style-type: none"> Display recommended components (Immunizations, Procedures, Medical Device use, Diagnostic results) With more experience with the frameworks and models, create accompanying tutorial pages for all code relating to the FHIR standard. 	1 Sprint
4	Optional Components, Polishing and Finalizing	<ul style="list-style-type: none"> Continuing/Finalizing tutorial development Polishing code and adding comments to aid user experience when using the accompanying tutorials Where applicable, displaying optional components 	1 Sprint
5	Further requests from S23m	<ul style="list-style-type: none"> Due to the nature of the project being likely to change, this extra sprint is allocated to allow for agile development 	1 Sprint

Figure 4: Phase 2 Release Plan

A sprint plan/timeline was created during the first sprint of phase 2, which outlined the tasks of each release and estimated the amount of work required for each. This plan showcases the hopes for both teams, and aims for an early handover time to safeguard last minute changes/challenges. The planned sprint timeline for phase 2 can be seen below in figure 5.

Release	Sprint 1		Sprint 2			Sprint 3		Sprint 4		Sprint 5	
	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11
Project Planning and Choosing Program Structure											
Search and Required Components											
Recommended Components and Tutorials											
Optional Components, Polishing and Finalizing											
Further requests from S23m											

Legend	
C# Team	
JavaScript Team	

Figure 5: Phase 2 Planned Sprint Timeline

When it came to implementing the plan, our team was largely able to follow the sprint structure and timing. As the client wanted deliverables in both C# and JavaScript, the decision was made during the first sprint to create a team for each language. Half of our team would work on the C# and the other half on the JavaScript. This came with some challenges, as although the FHIR standard is consistent between languages, the frameworks, packages, and systems that are used are different. Figure 6 showcases the progression of each team. Due to JavaScript not having a pre-existing framework to help with FHIR standard development, sprint 2 - the longest expected sprint - took a little longer for that team. The C# team used the pre-existing *Firely Hl7.Fhir SDK* to assist in development, which came with its own challenges, but ultimately proved to be very helpful.

There were no major changes/adaptations made to sprint plan throughout Phase 2. We worked well as a team, were transparent with our progress, and asked for help when we needed it.

Release	Sprint 1		Sprint 2			Sprint 3		Sprint 4		Sprint 5	
	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11
Project Planning and Choosing Program Structure											
Search and Required Components											
Recommended Components and Tutorials											
Optional Components, Polishing and Finalizing											
Further requests from S23m											

Legend	
C# Team	
JavaScript Team	

Figure 6: Phase 2 Actual Sprint Timeline

3.2 Risk Management

The risk register in phase 1 was brought into phase 2 with little changes being made. The most notable was that unit testing could not be used to ensure data was handled correctly. This was due to the fact that the test data is sometimes not entered correctly in the server. After informing the client of this we were allowed to ignore this. Risks were identified based on their impact on the ability to either deliver the project on time or deliver the agreed scope of the project. The mitigation strategies were written as a result of our analysis and included a response to the risk. If the risk occurred then the team could use this a quick reference guide (see figure 7).

ID	Risk	Likelihood of the risk occurring	Impact if the risk occurs	Mitigating Action
1	Medical data is not handled correctly or compromised.	Low	High	HL7 has a published list of verified public testing servers. These servers are generally password protected and the team trusts that they are maintained and audited regularly.
2	Malicious attack on testing servers as they are public in nature.	Low	High	Testing servers contain an audit log of all requests, no incidents of an attack have been recorded. In the case an attack occurs the team will cease using the compromised server and notify users.
3	Incorrect data translation when handling test data.	Medium	Low	Code will be reviewed at the conclusion of each sprint to ensure functionality is working as described. The reviewer will have experience in the medical informatics field and should be able to identify data that is incorrectly handled
4	Testing servers may be shut down or unavailable for a period of time.	Medium	Medium	In the event a testing server is no longer available during the project the tutorial code will be re-written to allow a connection to another server. If a server is unavailable due to maintenance or other issues this will be made available to the user. There are several testing servers available so there is a low chance of not having one to use.

Figure 7: Phase 2 Risk Register

During the project the team experienced intermittent server outages (Risk ID 4). As we had prepared for this risk by having backup servers the team was able to quickly switch to another server and continue developing. The impacts experienced lasted at most 1 day and so the impact to development was negligible as the team could work on other components (such as the tutorials) whilst waiting for the server to be fixed.

COVID-19 risks were not an issue as all of our meetings and correspondence throughout the project were in an online format. With the change in artefact in week 1 there was a risk of not meeting the clients needs as we did not initially have a plan for the new project design. This was mitigated through the creation of sprint and release plans, as we as revisions when needed to keep our goals realistic and unachievable. During the 11 weeks of development there were no other risks identified.

4 Artefact Description

As noted in the introduction, our team has developed an application in two different languages; C# and JavaScript. It is important to note that the scope of work remained the same for each so therefore this description covers both applications. Application screenshots within this section are taken from the C# application.

As stated in section 1.1.1, the overall goal of the artefact was to create an educational resource coupled with a web-application to help other developers understand how to retrieve, read, and use FHIR IPS data.

4.1 Functionality

The functional design of the web-application was planned with the purpose of being useful for a healthcare professional. The first step towards creating a useful application was to determine how to enable users to quickly find patient information, as displaying patient data was the focus of the application. To find an answer to this problem, we came up with a user story to view the issue from the user's perspective:

- As a healthcare professional working in a time-sensitive environment, I want to be able to find a patient's information quickly, so that I can quickly get access to the information I need.

From this user story, we decided on developing a search bar feature which would enable users to quickly search for a patient. From there, users could access a patients information by clicking on a search result. Figure 8 below shows the patient search page of the C# application.

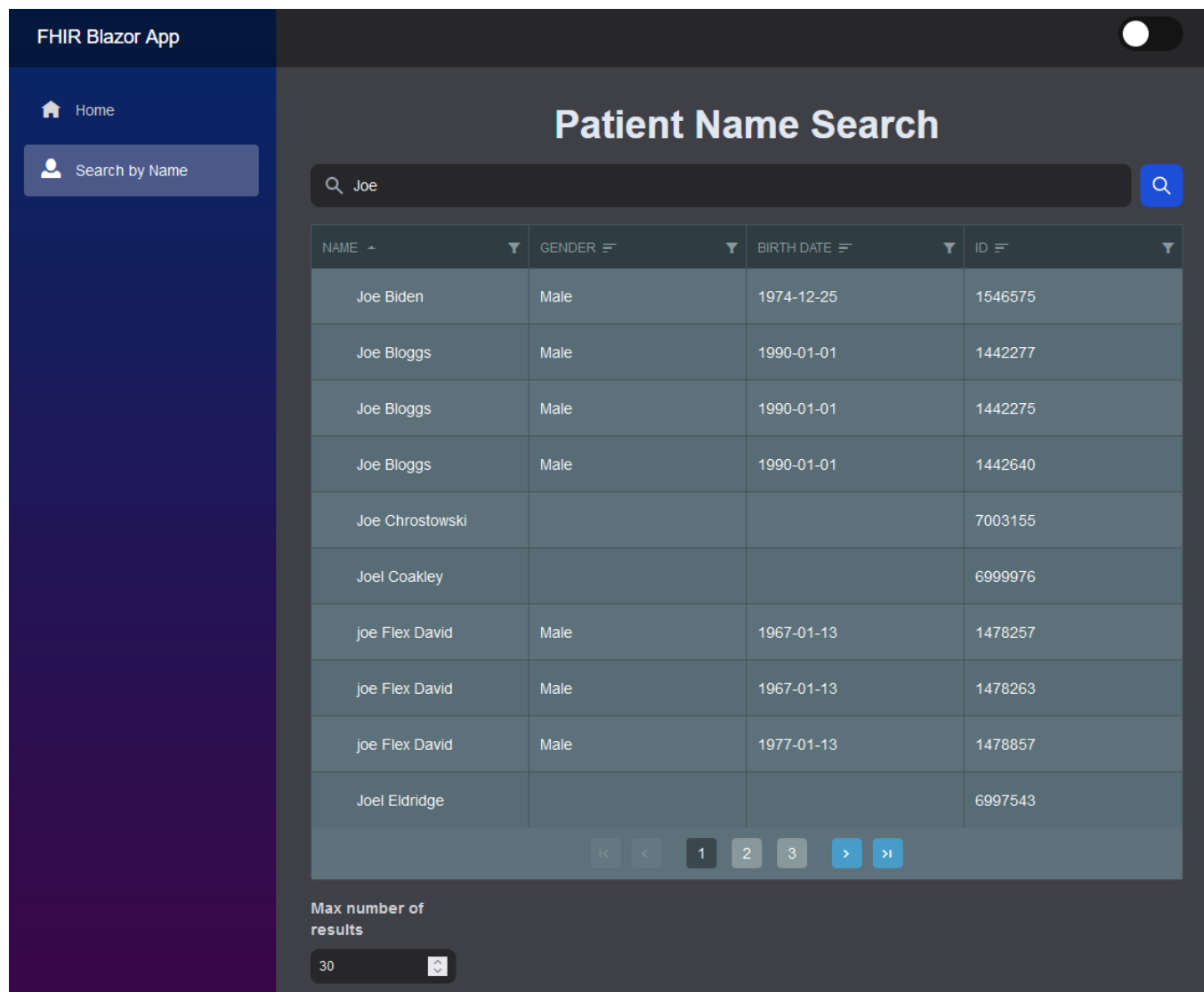


Figure 8: C# Patient search

Following from implementing the search by feature, we focused on adding patient details. The HL7 FHIR standard contains far too many data definitions and resource types relating to patient information for us to cover in 2 semesters of work. So our team along with our industry partner S23M had to prioritise resource types to cover. This was done by choosing resource types that: would be most widely used by others working with FHIR data, and would fit well with other resource types. Our decisions on what resource types to cover were decided on in each sprint planning, and from here we created user stories to flesh out our ideas for the functionality of components. The overall functional goal of each resource type was to create a component that would display information in an easily readable way, and would fit with other components. For example, we first developed the component to display basic patient information. We used the following user story to guide this development:

- As a healthcare professional working in a time-sensitive environment, I want to be able to view a patient's basic information at a glance, so that I can quickly understand the demographic of patient

I am working with.

This user story prompted us to develop the basic information components. These components displayed essential information as the first thing the user would see when opening a patient's resource page, as can be seen in figure 11 below.

The figure shows a user interface for patient information. It features two columns at the top: 'Info' and 'Personal'. The 'Info' column includes a profile icon, the name 'Betsy Smith-Johnson', and the gender 'Female'. The 'Personal' column includes the birthdate '1950-11-01' and the mobile phone number '210-222-1111'. Below these columns is a horizontal bar with the text 'All available patient details for id: BSJ-C0522-Incoherent' and a 'Patient' button. At the bottom, there is a tabbed interface with tabs for 'General', 'Address', 'Emergency Contact', 'Known Allergies', 'Conditions', and 'Immunizations'. The 'General' tab is selected, showing a section for 'Other Names' with the family name 'Smith-Johnson' and the given name 'Betsy'.

Figure 9: C# Patient basic information

All other FHIR data components followed the same design approach: decide on a resource type to display, write a user story, and develop a solution to that user story. As we added more data types, we needed a better way of displaying different components to be intuitive for the end user. We created the following user story:

- As a healthcare professional working in a time-sensitive environment, I want to be able to selectively view patient information, so that I am not distracted by information I don't need.

We decided creating tabs to display different types of information would provide a great way for users to get a focused, clutter-free view of the information they need. These tabs can be seen in figure 11.

4.2 Architecture

The architecture of the web-application was designed to be simple and easy for developers to follow. Given our artefact serves as an educational resource for developers, it was important to make the web-application as accessible as possible. The outline of the architecture for the two application was the same:

- Client-side dynamic web-application
- Can run locally on Windows, MacOS or Linux
- Connects to FHIR test servers to retrieve data via a HTTP request library

We along with S23M chose to make two separate applications using 2 different languages, so that we could appeal to a larger number of developers. Based on the above architectural requirements and the guidance of S23M, we chose to use:

- JavaScript with React
- C# with Blazor

Figure 10 below is the architectural diagram showing how these applications work.

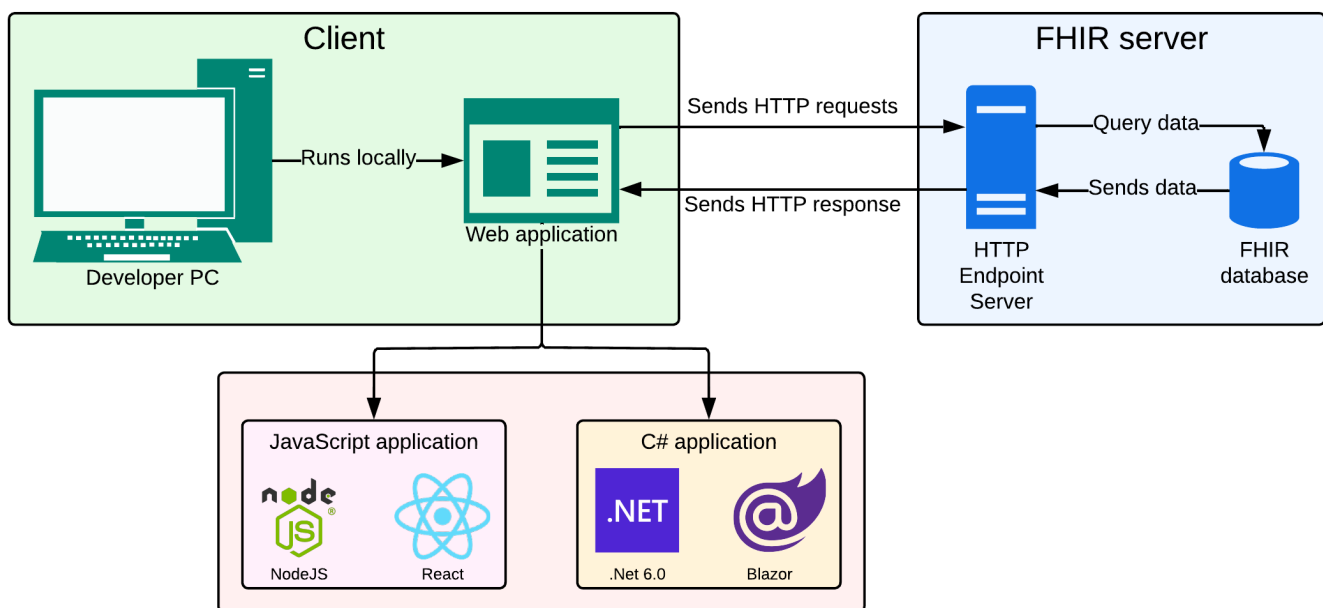


Figure 10: Architectural diagram

For the JavaScript application, the run-time environment is NodeJS, and React is used to create and display components using data retrieved from the FHIR server. For the C# application, the run-time environment is .NET 6.0, and Blazor is used to create and display components using data retrieved from the FHIR server.

We did not develop our own FHIR server, given there are already several highly-developed test servers publicly available. Our applications connect to one of these test servers.

4.3 Technical Description: JavaScript Application

The JavaScript application is a React based JavaScript web-app designed to allow users to find and view patient medical information, and teach developers how to interact with FHIR resources.

Application code structure

The JavaScript application is contained within the *FHIR-React* folder within the project's root. The source code of the JavaScript applicaiton is contained within the *FHIR-React/src* folder. The code within *FHIR-React/src* is split into three main categories:

APIs:

Within the *FHIR-React/src/apis* folder. Contains methods used to retrieve data from the FHIR-Server. The URL for the FHIR server is stored in the *baseURL.js* file, and all other API files use this base URL to connect to the FHIR server. This helps ensure all APIs are retrieving information from the same server, and makes it easy to change servers if needed. The API methods typically take URL parameters as method parameters. Using the base URL and these parameters, the URL for the specific API endpoint is created and used to retrieve data. For example, figure 11 shows the *searchPatient* method within *patient.js*.

```

5  // Add targeted endpoint
6  const PATIENT_URL = `${BASE_URL}Patient`;
7
8  /**
9   * Get patient search results from a set of parameter values
10  * Each element in queryTypes should have a matching value in queryValues
11  * Matching types and values should be at the same index
12  * @param {Array<String>} queryTypes
13  * @param {Array<String>} queryValues
14  * @returns response data
15  */
16  export const searchPatient = async (queryTypes, queryValues) => {
17    if (queryTypes.length !== queryValues.length)
18      throw new Error('Invalid query. Unequal number of query types and query values.');
```

```

19
20    // Construct query portion of request URL
21    // by appending each provided parameter key and value
22    const query = queryTypes.reduce(
23      (queryStr, queryType, i) =>
24        `!!queryType ? `${queryStr}${queryType}=${queryValues[i]}&` : queryStr,
25      ''
26    );
27
28    // Request and return response data
29    const fullUrl = `${PATIENT_URL}?${query}_format=json&_count=100`;
30    const response = await axios.get(fullUrl);
31    return response.data;
32  };

```

Figure 11: *searchPatient* method within *FHIR-React/src/apis/patient.js*

The *searchPatient* method on line 16 takes two parameters, both of type *Array<String>*. The *queryTypes* parameter contains the keys for the parameters used within the API url. In this case, an example for a *queryTypes* element could be *family* if the user is searching for a patient by family name. The *queryValues* parameter contains values for the keys within *queryTypes*. An example could be *Johnson*, if the user wanted to search for a patient with the family name "Johnson". The element index within *queryTypes* pairs with the same element index within *queryValues*, so on line 17 a check is performed to make sure there are an equal number of *queryTypes* and *queryValues* elements, otherwise the query is invalid. The *reduce* function used on line 22 converts the elements of *queryTypes* and *queryValues* into a string of the parameter section of the request URL. Another check is done on line 24 to ensure each *queryTypes* element is not an empty string or null element. On line 30 the data is retrieved sending a request to the URL created by combining the base URL with the query section, and on line 31 this data is returned. The *searchPatient* method is the most complicated of the API methods because it can use multiple URL query parameters. Most of the other API methods simply take an ID as their method parameter.

Components:

Within the *FHIR-React/src/components* folder. Contains JSX elements created with React and used to search for and display data retrieved from API methods. Components are added to pages where they are rendered on screen. By separating components from pages, we are able to independently

develop components without affecting the way other components on a page look and behave. One of the more complex components used in the JavaScript application is *PatientExtraInfoCard.js*. This component is used to display information about a patient other than their basic details such as age and birthday, as can be seen in figure 12 below. The *PatientExtraInfoCard* component is the lower card in the screenshot, containing tabs such as "Additional Info", "Allergy intolerances" etc.

ROMA FHIR Patients

Patient's basic information

ID	1059
Name	Maria MN123 Max Mountbaton
Gender	male
Date of birth	2001-10-06
Deceased?	N/A

Additional info

Allergy intolerances

Procedures

Medication statement

Immunization Recommendation

Device Use Statement

Diagnostic Report

Identifiers	Contact
Medical record number: MR1039 Placer Identifier: EPID1031 Provider number: 191114444 Serial Number: 4444 Filler Identifier: 181111111 Driver's license number: 20333333333	home phone: 1311111111 work phone: 1422222222 home email: 13abc@abc.com work email: 14abc@abc.com
Languages	Address
English	home L0111 L0112 L0113 L1112 Seattle WA 98052 US
Marital status	Active?
Married	N/A

Figure 12: Patient page containing *PatientExtraInfoCard* component

This component contains other components, which are each rendered in their own tabs. For example, the "Additional Info" tab contains the component from the *PatientAdditionalInfo.js* file. The *PatientAdditionalInfo.js* component provides a good example of how data is converted into a viewable format, as shown in figure 13 below.

```

7 // Assistive functions to get required data from patient response
8 // Each function follows the same format
9 const getPatientTelecomInfo = (patientInfo) => {
10   // Check if there is any telecom info
11   if (!patientInfo.telecom || patientInfo.telecom.length === 0) return NA_ARRAY;
12
13   // Filter through telecom information for valid entries
14   const validPatientTelecoms = patientInfo.telecom.filter(
15     (telecom) => !!telecom.use && !!telecom.system && !!telecom.value
16   );
17
18   // Format filtered telecom info into data useable by front-end component
19   const patientTelecoms = validPatientTelecoms.map(
20     (telecom) => `${telecom.use} ${telecom.system}: ${telecom.value}`
21   );
22
23   return patientTelecoms.length > 0 ? patientTelecoms : NA_ARRAY;
24 };
25
26 const getPatientIdentifiers = (patientInfo) => {
27   if (!patientInfo.identifier || patientInfo.identifier.length === 0) return NA_ARRAY;
28
29   const validPatientIdentifiers = patientInfo.identifier.filter(
30     (identifier) =>
31       !!identifier.type && !!identifier.type.coding && identifier.type.coding.length > 0
32   );
33
34   const patientIdentifierInfo = validPatientIdentifiers.map(
35     (identifier) => `${identifier.type.coding[0].display}: ${identifier.value}`
36   );
37
38   return patientIdentifierInfo.length > 0 ? patientIdentifierInfo : NA_ARRAY;
39 };

```

Figure 13: Code within *FHIR-React/src/components/PatientAdditionalInfo.js*

The two methods shown on lines 9 and 26 essentially work the same way. They filter information provided through the *patientInfo* parameter and convert it to useful information to be rendered in the component. The *patientInfo* parameter is the JSON response from a FHIR server for the information of a given patient ID. First the functions check if *patientInfo* is empty. If it isn't, the methods move on to filter for the information being retrieved by the function. For the *getPatientTelecomInfo* function on line 9, it is looking for valid communications information within the *patientInfo* record, by filtering the *patientInfo.telecom* array on line 14. Then, on line 19 the method maps this filtered array onto a new array, where each filtered telecom element is converted into an easily viewable format, shown on line 20. The final array is then returned by the function, and it is used within the component. These steps of extracting and formatting data are used within most of the components.

Pages:

Within the *FHIR-React/src/pages* folder. Contains the pages rendered in the browser. Pages are made up of components, as previously discussed. For example, the *PatientInfo.js* renders the */pa-*

tients/{id} page. Figure 14 below shows code used to create the *PatientInfo* page.

```
11  const PatientInfo = () => {
12    const { id } = useParams(); // Get patient's ID from URL
13    const [patientData, setPatientData] = useState({});
14
15    useEffect(() => {
16      getPatient(id).then((response) => {
17        console.log('Patient Response:', response);
18        setPatientData(response);
19      });
20    }, [id]);
21
22    return (
23      <div>
24        <PatientInfoBasicCard patientInfo={patientData} />
25        <PatientExtraInfoCard patientInfo={patientData} id={id} />
26      </div>
27    );
28  };
```

Figure 14: Code used to create *FHIR-React/src/pages/PatientInfo.js* page

PatientInfo.js retrieves the patient ID from the URL parameter on line 12. This ID is then used to retrieve patient data on line 18. The */patients/{id}* page presents as a fairly complex web page, but the *PatientInfo.js* file which renders this page is quite simple. Only two components are used to display data on this page, which are shown on lines 24 and 25. The *PatientExtraInfoCard* component on line 25 does however contain several components within it's component. Overall, separating components and pages allows for the simplicity seen in the *PatientInfo.js* file.

Within *FHIR-React/src* there are a few other file types which are much more simple or commonly seen in web-applications than the three main types discussed above.

- *assets*: contains images used within the application.
- *mock-data*: contains data used for testing purposes.
- *stylesheets*: contains CSS files used to style the application.
- *testing*: contains unit test files (we did not use these however).

Everything else within *FHIR-React/src* are either configuration files such as *.gitignore* or *.prettierrc*, or installation and application launching files such as *package.json*.

4.4 Technical Description: C# application

The C# application is a Blazor web-app designed to allow users to find and view example patient medical information, and teach developers how to interact with FHIR resources.

Application code structure

The C# application is contained within the *FHIR-Blazor* folder within the projects root. The code is built on the Blazor Server App template provided by Visual Studio 2022. The application makes great use of the *Firely* NuGet packages: *HI7.Fhir.ElementModel*, *HI7.Fhir.R4*, *HI7.Fhir.Support*, and *HI7.Fhir.Serialization*. The project also uses *Radzen* components to help with displaying data and formatting. The *FHIR-Blazor* folder has many sub folders, but the two notable ones are *CustomComponents* and *Pages*. The *Pages* folder was created by the template and houses the files that are shown as pages in the web application. The *Index.razor* and *NameSearch.razor* files are of interest to the project as they reference the Custom Components created. The file structure can be seen in figure 15.

The *CustomComponents* folder houses the large majority of the code created throughout the project. These components all make calls to the Fhir test server and retrieve specific information relevant to the *Patient ID* inputted. Within the *CustomComponents* folder there are 4 razor files, these are the starting points for all other components within subfolders. The *PatientNameSearch.razor* file contains all the code relevant to searching for and viewing simple patient data.

The *NameSearching()* function can be seen below in figure 16, where the Fhir server is defined on line 106, the search parameters on 109, and the search query itself within a try block at line 118. The search returns a *Bundle* - a datatype defined by the *Firely* packages - which is then converted into a list of patients. Most of the other components in the project follow a very similar format of receiving a bundle from the server, then converting it to the relevant datatype to be displayed on the page. Once the patient name results have been received they are displayed in a *Radzen Table* where the user can click on a patient to open a quick info modal. This modal is formatted in the *PatientQuickInfo.razor* component. From there the user can choose to view in depth patient info, or go back to the search.

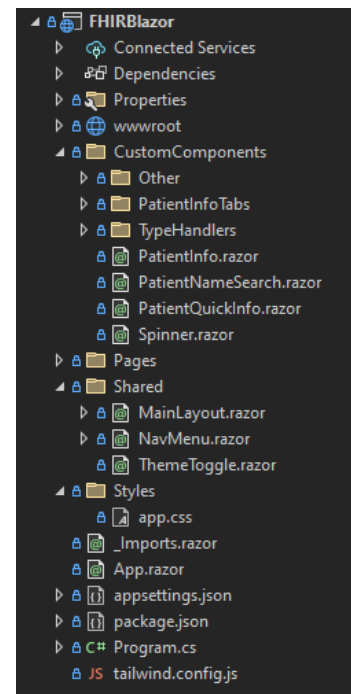


Figure 15: C# Project File Structure

```
101 void NameSearching()
102 {
103     Console.WriteLine("Searching");
104
105     //connecting to the server
106     var client = new FhirClient(NameSearch.serverURL);
107
108     //Defining search paramaters, more information on these can be found here in the
109     var q = new SearchParams()
110         .Where("given=" + inputPatientName)
111         .OrderBy("birthdate", SortOrder.Descending)
112         .SummaryOnly().Include("Patient:organization")
113         .LimitTo(maxNumOfEntries);
114
115     //HAPI returns a Bundle when searching (i assume other servers do the same)
116     try
117     {
118         Bundle results = client.Search<Patient>(q);
119         patients = new Patient[results.Entry.Count];
120
121         //for every search result
122         for (int i = 0; i < results.Entry.Count; i++)
123         {
124             //add patient to our list
125             patients[i] = (Patient)results.Entry[i].Resource;
126         }
127         Console.WriteLine("Results Found");
128     }
```

Figure 16: Code used to search for patients in FHIR-Blazor/CustomComponents/PatientNameSeach.razor

Once a user wants to view in depth patient information, they are taken to the page which is handled by the *PatientInfo.razor* component. An example of this page can be seen in the Artefact Description in figure 11. This page makes great use of *Radzen* Components, specifically the *RadzenCards*, and *RadzenTabs* components to display information and keep the page looking clean. The code for the address card and the lower set of tabs can be seen below in figures 17 and 18 respectively.

```

<RadzenTabsItem Text="Address">
  @if (patient.Address.Count > 0)
  {
    <ErrorBoundary>
      <ChildContent>
        <RadzenCard class="m-3">
          <h3 class="h5"><b>@patient.Address.First().Use Address</b></h3>

          @if (patient.Address.First().Country != null)
          {
            <div class="row">
              <div class="col-md-4">
                <div>Country</div>
                <b>@(patient.Address.First().Country)</b>
                <div class="mt-3">State</div>
                <b>@(patient.Address.First().State)</b>
              </div>
              <div class="col-md-4">
                <div>City</div>
                <b>@(patient.Address.First().City)</b>
                <div class="mt-3">Line</div>
                <b>@(patient.Address.First().Line.First()) </b>
              </div>
              <div class="col-md-4">
                <div>PostCode</div>
                <b>@(patient.Address.First().PostalCode)</b>
              </div>
            </div>
          }
          else
          {
            <b>@patient.Address.First().Text</b>
          }
        </RadzenCard>
      </ChildContent>
      <ErrorContent>
        <RadzenCard class="m-3">
          <h3 class="h5"><b>There was an error</b></h3>
        </RadzenCard>
      </ErrorContent>
    </ErrorBoundary>
  }
  else
  {
    <RadzenCard class="m-3">
      <h3 class="h5"><b>No Address information</b></h3>
    </RadzenCard>
  }
</RadzenTabsItem>

```

Figure 17: Code used to display address info in PatientInfo.razor lines 117-167


```

253 <RadzenTabs TabPosition="@tabPosition" style=" margin: 20px auto;" RenderMode="TabRenderMode.Server">
254   <Tabs>
255     @*Medications tab, see MedStatements.razor*@
256     <RadzenTabsItem Text="Medication Statements">
257       <MedStatements PatientID=@PatientID />
258     </RadzenTabsItem>
259
260     @*Diagnostic Report tab, see DiagnosticReports.razor*@
261     <RadzenTabsItem Text="Diagnostic Reports">
262       <DiagnosticReports PatientID=@PatientID />
263     </RadzenTabsItem>
264
265     @*Procedure History tab, see Procedures.razor*@
266     <RadzenTabsItem Text="Procedure History">
267       <Procedures PatientID=@PatientID />
268     </RadzenTabsItem>
269
270     @*DevicesUse tab, see DeviceStatements.razor*@
271     <RadzenTabsItem Text="Device Statements">
272       <DeviceStatements PatientID=@PatientID />
273     </RadzenTabsItem>
274   </Tabs>
275 </RadzenTabs>

```

Figure 18: Code used to create tabs in *PatientInfo.razor* lines 253-275

Within *CustomComponents*, the sub folder *PatientInfoTabs* holds all the components that are being displayed in the tabs on the *PatientInfo* page. The reference to these components can be seen above in figure 18. As stated earlier, these components follow a similar format for data retrieval as the name search outline in figure 16. They all use the *RadzenDataGrid* component to create a table to display all the retrieved data. The code used to display the retrieved information in a table can be seen below in figure 19.

```

<RadzenDataGrid Data="@results" TItem="Immunization" PageSize="10" AllowPaging="true"
    AllowFiltering="true" AllowColumnResize="true" AllowSorting="true"
    PagerHorizontalAlign="HorizontalAlign.Center">
  <Columns>
    <RadzenDataGridColumn TItem="Immunization" Property="Vaccine" Title="Vaccine" Filterable="false" Sortable="false">
      <Template Context="immunization">
        <CodeableConceptHandler Data=immunization.VaccineCode />
      </Template>
    </RadzenDataGridColumn>

    <RadzenDataGridColumn TItem="Immunization" Property="Site" Title="Site" Filterable="false" Sortable="false">
      <Template Context="immunization">
        <CodeableConceptHandler Data=immunization.Site />
      </Template>
    </RadzenDataGridColumn>

    <RadzenDataGridColumn TItem="Immunization" Property="Dosage" Title="Dosage" Filterable="false" Sortable="false">
      <Template Context="immunization">
        @if (immunization.DoseQuantity != null)
        {
          <a>@immunization.DoseQuantity.Value @immunization.DoseQuantity.Code</a>
        }
      </Template>
    </RadzenDataGridColumn>

    <RadzenDataGridColumn TItem="Immunization" Property="Occur" Title="Date" Filterable="false" Sortable="false">
      <Template Context="immunization">
        <DateHandler Data=immunization.Occurrence />
      </Template>
    </RadzenDataGridColumn>

    <RadzenDataGridColumn TItem="Immunization" Property="Expiration" Title="Expiration" Filterable="false" Sortable="false">
      <Template Context="immunization">
        <a>@immunization.ExpirationDate</a>
      </Template>
    </RadzenDataGridColumn>

    <RadzenDataGridColumn TItem="Immunization" Property="Id" Title="ID">
      <Template Context="immunization"><a href="@($"{NameSearch.serverURL}/Immunization/{immunization.Id}")">@immunization.Id</a></Template>
    </RadzenDataGridColumn>
  </Columns>
</RadzenDataGrid>

```

Figure 19: Code used to create Immunizations tab in Immunizations.razor lines 18-61

There are other sub folders and files in the project. The *TypeHandlers* folder holds two *.razor* files that handle displaying Fhir datatypes due to their variable nature. For example some dates within results can be stored as strings, periods of time, ages, or ranges, so the *DateHandler.razor* component was created to handle those variable data types. The use of this component can be seen in figure 19 in the creation of the 4th table column. The *Other* folder houses extra components like *ViewReport.razor* that is used to create a modal to view more details on Diagnostic Reports. The *Devices.razor* file is not used in the final product, but was a draft tab.

4.5 Quality

During semester 1 of IT Capstone we discussed testing with the client to figure out what we should be doing to ensure a high quality artefact. The client recommended we did not need to perform unit testing on our code, given the extra time this would have added to our development. We, along with the client decided it would be better to spend that time adding more features and creating a more polished application, as opposed to unit testing, as we believe this would overall increase the quality and usefulness of the application. We still needed to ensure our code worked as expected, and to do this we manually tested our changes as we developed. Doing this we were able to catch bugs as they occurred, and fix them quickly.

The JavaScript application was created as a single-page application. We chose to design the application this way to improve the quality of the user experience. Given the navigation bar at the top of the page is a component that is used on all pages of the application, creating this application as a single-page application made the load times faster and the transition between pages smoother. This is because the application does not have to reload the navigation bar for every page. It only loads new information on each page, creating a higher-quality application. Utilising a single-page application also allows the page to display information while other information loads. There are several different loading points within the application, such as on the `/patients/{id}` page, where each tab within the lower card shown in figure 12 loads different data from the FHIR server. Using a single-page application means the page can display what is already loaded while still waiting for data from other tabs to load. During loading, the page can also display a loading symbol when needed.

The C# application was also created as a single-page application for the same reasons. This was one of the reasons we chose to use Blazor as a web development framework as opposed to others such as ASP.Net. Blazor is currently the only framework providing development of single-page applications within the .Net environment.

The speed of the applications was important to us and the client S23M. By following best practices and focusing on developing with simplicity, we were able to achieve very high performance results using Chrome's Lighthouse web-page analyser. For the JavaScript application, we received performance and best practice ratings of 100/100 for the homepage and `/patients` page. The `/patients/{id}` page received 92 for performance, and 100 for best practices. These results can be seen in section 9: Appendix in figures 24, 25, and 26.

The performance of the C# application was slightly lower, but still good. The homepage received ratings of 72 for performance and 100 for best practices. The `/searchbyname` page received ratings of 70 for performance and 100 for best practices. The `/patientInfo/{id}` page received ratings of 67 for performance and 100 for best practices. These results can be seen in section 9: Appendix in figures 27, 28, and 29.

The load times when searching for patients and loading patient info on the applications can be slow, but this is due to the FHIR server we are connected to, not the applications themselves. We tested a range of publicly available FHIR servers for robustness, speed, and quality of data, and found the currently used server: <https://hapi.fhir.org> to be the best, most well rounded server.

The developer tutorials were tested for quality by having our client S23M review the material, as well as some fellow students who are developers. The feedback was mostly positive, other than some small changes. For example, S23M suggested we shouldn't explain the code in the tutorial text, but instead use comments within the code, then add those code snippets to the tutorials. This way it is easier for a developer to follow along and understand what is happening in the code. By taking on feedback and making changes, we were able to ensure our tutorials were high quality.

5 Individual Chapter: Riley Head

5.1 Project Setup

5.1.1 Project Management Approach

In semester 1 I was designated the project manager and it was agreed that I should continue the role in semester 2. We agreed in semester 1 to use Atlassian Jira as our project management tool and Confluence as our document repository. As the project manager I setup these tools by assigning accounts and sending an invite to our industry supervisor so they could access the tools as well. On a weekly basis I assisted in creating and updating tasks in Jira as well as maintaining the Jira configuration and improving any workflows where necessary.

Team Roles & Responsibilities

 Created by Riley Head
Aug 02, 2022 • 1 min read •  Signatures: 3

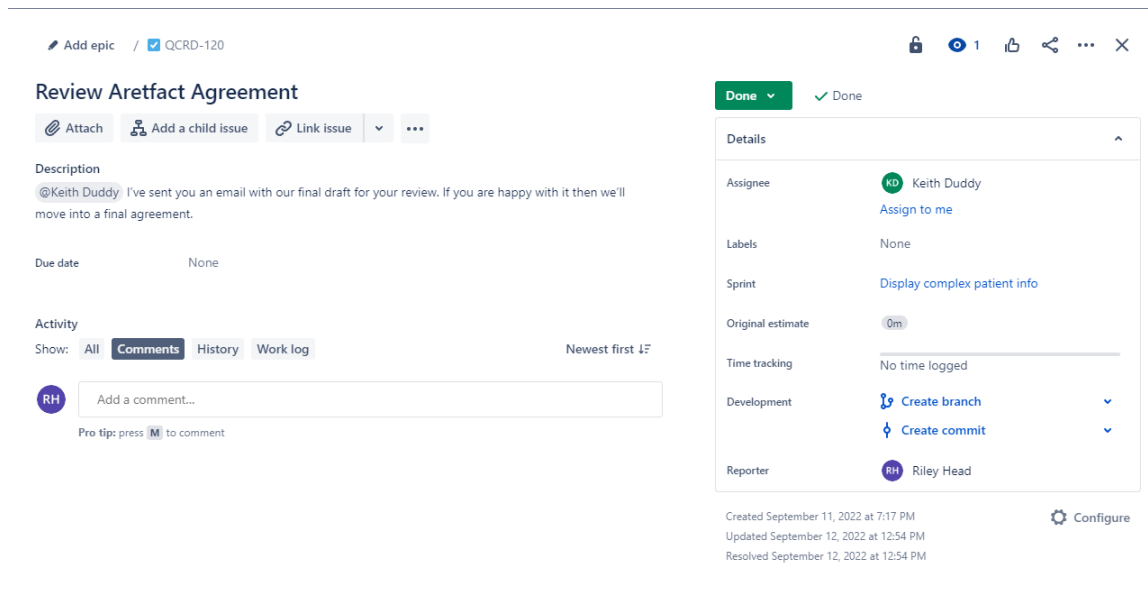
Name	Role\s
Riley Head	Project Manager, Developer (C#)
Max Goodwin	Developer (JavaScript)
Rodo Nguyen	Developer (JavaScript)
Mackenzie Smith	Developer (C#), Client Relations

5.1.2 Client Expectations

As a team, we agreed to fortnightly meetings with our client S23M. To prepare for these meetings I made sure the rest of the team was aware of what we were going to be showing the client in our meetings so that they could be prepared which made the client confident that we would deliver what was agreed. I lead these meetings from our end and answered questions from the client but also gave the rest of the team the opportunity to do.

When creating the artefact agreement I made sure to consult with the client whilst writing the agreement. After creating the draft version, I sent it to the client to get their thoughts and comments. Having collected the clients comments I then made the relevant changes and sent through the final version for his approval.

Jira was a useful tool in managing the clients expectations as I could assign tickets to him which would also send an email notifying him of tasks to complete. The tickets would inform the client of any reviews of code or documentation be done. You can see an example below.



5.1.3 Team Collaboration

By defining team roles and responsibilities we were able to collaborate much easier. As our project involved programming in two different languages I decided as project manager that the team should be split into two sub-teams. A C# and JavaScript development team. This segregation was also reflected in Jira where tasks were created for each of the teams (see the example to the right).

After the first few weeks of development the team got into a good rhythm and so there was not much management required from my end. I would occasionally hold team meetings just to see where we were at and to also discuss anything important that the client had told us. As we were developing in two programming languages it was important to ensure that the product functioned in a similar manner across both applications. I made sure in our team meetings that each application had the same functionality and that we were not deviating from what was agreed on.



Figure 20: Task Segregation

5.1.4 Communication Plan

Communication in the team was handled through a discord server. I used this platform for general conversations or to ask questions of the team. The server had a few channels to organise communications. For example, when I had issues with programming or bugs I would post to the development channel.

In our fortnightly meetings with the client I started by giving a general overview of the work that the team would be showcasing. Following this I would then direct members of the team to share their screen and showcase our work to the client. If the client had questions I would try to answer them but I made sure to let the rest of the team engage in conversation as well.

I took a chill approach as project manager as we had proven to ourselves that we can trusted to get the job done with little supervision. I would occasionally ask and remind but always trusted that the work would be done. In our team meetings I would make sure to give direction so that we were all aware of where we we are heading. This helped ensure that the two application would be very similar in functionality.

I realised quickly that it was important to be prepared for the fortnightly tutor meetings so before the meetings I prepared a document outlining our points to discuss and who would speak to these points. This meant that the team could open up the document before the meetings so they knew and could be prepared for what was going to be shown. This helped us in coming across and professional and well prepared to our tutor.

5.2 Project Plan and Risk (Phase 2)

5.2.1 Project Planning and Progress

Being project manager, my main job was to create and allocate tasks efficiently. In our project management software Jira, I created basic tasks and sub-tasks and then assigned them to the appropriate members of the team. The example below shows an example task for the C# development that includes sub-tasks as well.

The screenshot displays a Jira issue page for 'C# Development (See Sub-Tasks)'. The issue is in the 'In review' status and is assigned to Mackenzie Smith (MS). It has no labels and is part of the 'Final Capstone Deliverables' sprint. The original estimate is 0m. The time tracking section shows 'No time logged' and 'Include child issues' is checked. The development section has links for 'Create branch' and 'Create commit'. The reporter is Riley Head (RH). The issue was created on July 27, 2022, at 5:46 PM and was updated 1 hour ago.

Child issues:

Issue ID	Task Name	Assignee	Status
QCRD-106	Display Immunizations	RH	DONE
QCRD-107	Display History of Procedures	RH	DONE
QCRD-108	Display Medical Devices	MS	DONE
QCRD-109	Diagnostic Results	MS	DONE

Activity:

Show: **All** | Comments | History | Work log

Activity feed (Newest first):

- RH: Add a comment...

Pro tip: press **M** to comment

As seen in the above screenshot the tickets contain information such as the due date and who is assigned. Tickets are also assigned a status (To Do, In Progress, In Review and Done) this is useful for getting a high level overview of the project at any given time. As the client had access to Jira they could also see this at any time.

5.2.2 Risk Management

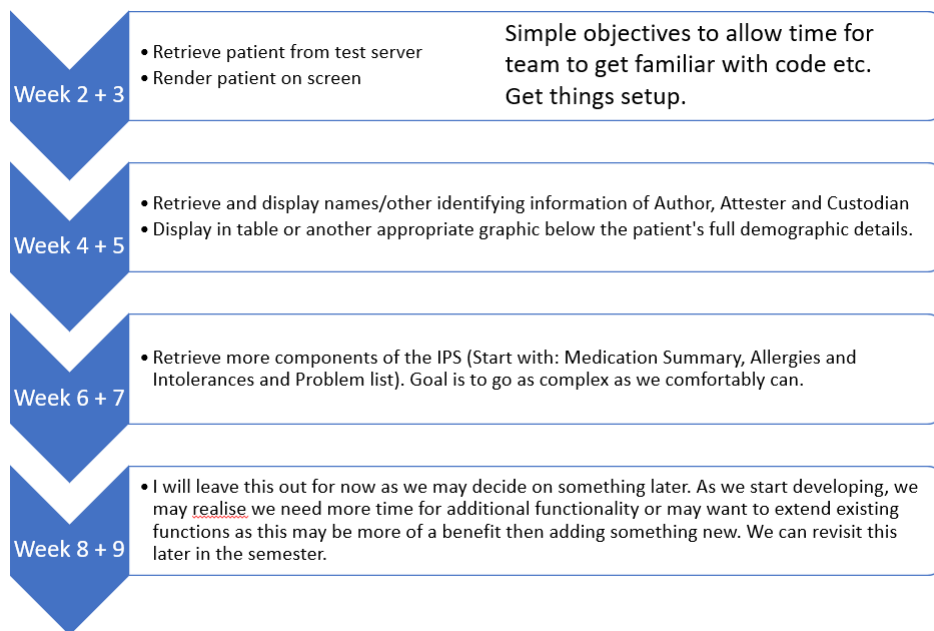
As the project manager, there were a few risk I was concerned about. The main one was scope creep. When we came back from the mid-year break we found out that the project had changed direction and we had to rethink a lot of what we had initially planned. We had to redo a lot of the planning and figure out what we were making. Luckily this was done quickly and I knew in the back of my head that we should not over-promise. When it came to writing the artefact agreement I also took this view and so we listed a few features as optional giving us the flexibility to deliver a good product without the pressure of promising too much. Risks related to my role as a project manager include:

5.3 Project Experience

As mentioned previously, after our first meeting with the client in semester 2 we found that the scope had changed quite a bit from what we understood in semester 1. Our understanding in semester 1 was that we were going to build a few small applications and some written material to accompany them. Now we had to build a full web application in two different languages that could support read-only access to a variety of patient data. With this new information I had to quickly redo all the planning we had done in semester 1. This included

- Developing a proposal of the new sprints and tasks for the client
- Updating sprints and tasks to reflect the new deliverable
- Talking with the team to understand what we can and can't do

The hard part was that the actions above had to be completed in less than a week since it was eating into our development time. The most important thing was to determine what we should be building and discussing the features that are a must-have and features that would be good to have but not required. After these discussions I came up with a proposed sprint schedule (see below)



The client was happy with this schedule and understood that things were likely to change as we progressed with development. I learnt from this situation that no matter how good you plan, things can always change. The important point is that you should always be ready for things to change and so your tools should be setup so they can be changed quickly and easily. Additionally, I felt that maybe I should have made sure that at the end of semester 1 everyone was on the same page. Maybe this was due to a miscommunication or because the client and our team were interpreting the same thing differently. It's important to make sure that everyone understands what has been proposed and has a clear picture of what it should look like.

Over the course of the project our client contact was going through some personal medical complications that meant at times he could not review work or look at some material. As project manager this tested my skill because I couldn't rely on us always getting feedback for things that we had built. I consider feedback over the course of the project very useful as we don't want to get to the end and realise there is lots wrong. Whenever I was met with these delays I made sure to always be communicating with the client about what we were doing and going to be doing. I paid special attention to being as descriptive as possible in both my written and verbal communications so that the client felt comfortable with what we had done. The result of these actions is that the client trusted us to keep going despite not being able to review some material at certain points during the project. This meant there were no delays or times where the team was being underutilised. I learnt from this that it's very important to be accommodating to clients because the most important thing to get from them is their trust and confidence in me and the rest of the team to deliver a good quality of work.

6 Individual Chapter: Max Goodwin

6.1 Project Setup

6.1.1 Project Management Approach

I was assigned to the roles of developer and solution architect in IT Capstone 1. Myself and Rodo Nguyen decided to work on the JavaScript application together. During semester 1, I mostly worked on planning the technical architecture of the JavaScript application along with Rodo. First, we decided on what type of web application we wanted to develop. We decided on creating a single-page application for the reasons discussed in section 4.5. Following from this decision, we needed to choose a JavaScript framework. We chose React because we both had experience using React and React is one of the most popular web development frameworks, so it would allow us to reach a wider audience for our tutorials. I also setup GitHub repositories for our code, and created rules within the repository such as pull-requests requiring review before being merged into the main branch.

During IT Capstone 2, Rodo and I developed the JavaScript application. Riley Head was our project manager. As project manager, he would work with S23M and the rest of the group to decided on overall development goals for each sprint. For example, in the first sprint one of the development goals was to create a search feature for searching patients by name. Rodo and I then broke this goal down into smaller tasks, such as creating the API to query the FHIR server, and creating the front-end search bar component. From here we would each choose tasks to work on, and help each other when needed. We also reviewed each other's code through pull-requests on GitHub. I also continued to handle the overall management of the GitHub repositories. Initially we setup two separate repositories, one for each application. Later we decided it would be best to combine these into a single repository. I handled this migration, creating a single repository by merging the previous two repositories while maintaining all commit history.

6.1.2 Client Expectations

At the end of every fortnightly sprint our team met with Keith from S23M to discuss progress and goals for the next sprint. I would often showcase new features in the JavaScript application, and then as a group we would discuss next steps. To manage client expectations, I would give estimates in these meetings for how long new features would take, and make sure we were not promising too much. I would also reach out to S23M and Riley as the project manager when we ran into any issues that could slow development. I would do the same when our development was faster than expected, and could take on more work than expected for a sprint. By keeping constant communication with the S23M I was always able to meet or exceed expectations for each sprint.

We used Jira as our project management software. I updated tickets with information about progress whenever progress was made, and would link to the commits of my progress. This allowed my teammates and S23M to easily view and understand progress by checking on my tickets in Jira. This meant S23M

could always understand the state of my development, and what specific tickets I was working on.

6.1.3 Team Collaboration

By creating defined roles for each team member, we were able to easily delegate tasks and understand who was responsible for what. Of course, there were many times when we had to collaborate on tasks to achieve the best results. Whenever I needed help with something, my first inclination was to message the team on our Discord server. By doing this, I knew everyone would see my messages and be able to respond in their own time. It was faster and easier than sending emails, and easy to keep track of conversations. Beyond this, I would also post comments on Jira tickets so that questions I had could easily be related to a particular task. This was also useful for keeping history of task progress, and there were times when I ran into a problem which was similar to something I had dealt with previously, so having this history made it easier for me to solve the problem again. If I needed more urgent help or someone else was online and available to help, I would join the Discord group chat and go through the problem by sharing my screen.

I tried to lend a helping hand whenever my teammates had an issue. I installed Discord on my phone so that I could always message if someone needed help. I kept updated with progress of git commits and Jira ticket updates by enabling notifications for GitHub and Jira. There were times when Rodo needed help with a task he was working on. For example, he needed help fixing a bug where the JavaScript application wasn't rendering anything. I was online at the time when he messaged our Discord chat about this, so we both joined the Discord voice chat and he showed me the issue. We talked through and tried possible solutions. We initially weren't able to solve the problem, so I pulled Rodo's git development branch onto my computer, and worked on finding a solution locally. Within another 10 minutes, I figured out it was an issue with how one of the components was being called within a page, and we solved the problem together.

6.1.4 Communication Plan

As previously mentioned, we as a team met with S23M fortnightly to discuss progress. Outside of this we would also have meetings separate from the client. Usually these would take place directly after our meetings with S23M, and after our fortnightly tutor meetings. This worked well for all of us as we were usually all free at these times, and meeting directly after talking with the client and tutor helped us quickly determine next steps and check in with everyone's progress. During these meetings, I made an effort to make sure everyone was on the same page with development, and help out anyone where possible. I also wanted to build the personal relationships between our team members. During our meetings, I always tried to get some casual conversation going. We would talk about hobbies, other subjects, work, and through this we grew more comfortable with one another and built friendships.

We also used Discord as a method for ongoing communication. Whenever any of us had questions or progress to talk about we would send a message in the Discord chat, and then everyone was able to respond when they had the chance. We also had a casual chat for talking about anything going on in

our lives, or sharing memes, which helped us build a friendly team culture. I was responsible for setting up and maintaining the Discord chat. I setup different chats for different purposes, and pinned important messages.

6.2 Project Plan and Risk (Phase 2)

6.2.1 Project Planning and Progress

As mentioned in section 6.1.1, our project manager Riley was primarily responsible for setting the development goals of each sprint. From there, Rodo and I broke down goals into smaller tasks, and then we would each assign tasks to ourselves. We would usually each take on half of the JavaScript tasks for each sprint. We tried to make each task roughly the same amount of work, but sometimes some tasks would be more or less effort than others. If this was the case, Rodo and I would let each other know if something is taking longer than expected. If needed we would move tasks between us in order to meet expectations for the sprint and share the workload equally.

To report on progress, I posted comments on Jira tickets explaining updates. Usually these comments would coincide with a git commit, and I would link to the git commit in the comment. This way, my entire team and S23M was able to track my progress whenever they wanted to. This helped me keep accountable and on-track to meet or exceed expectations for each sprint. I also showcased updates to S23M during our fortnightly meetings. As these meetings took place toward the end of each sprint, these were great opportunities to get feedback and make changes before the sprint ended. These updates on Jira and during our meetings provided an easy way for our project manager Riley Head to keep track of my progress.

6.2.2 Risk Management

Risk ID 3 and 4 in 7 were the biggest risks I faced during this project. Risk ID 3 was mostly mitigated by testing different FHIR servers to find one that had a broad and detailed data set. The server we decided was best overall was <https://hapi.fhir.org>. This server does however have some malformed patient data. This isn't a huge risk for the JavaScript application because there is no defined data type for FHIR resources. Instead JavaScript converts the JSON response from the FHIR server into an object of key-value pairs. If the data is malformed, the application will simply not display data that is malformed. No errors are thrown, so the application does not break.

For risk ID 4, Rodo and I decided to add a line containing a backup server to the *FHIR-React/src/apis/baseUrl.js* file. If the primary server goes down, we can easily change the server to the backup. In retrospect, we could have automated this by testing server connection within the application. We also used this mitigation strategy for risk ID 2.

Another risk I was concerned about throughout the entire project was my ability to maintain progress and meet expectations. I have a highly sporadic job in terms of the hours I work and when I work, so I don't always know when I will have the time to work on university assignments. My main method of mitigating

this risk was to do as much work as possible early in the sprints. This meant whenever I had spare time at the start of a sprint, I would focus my time on developing the JavaScript application to decrease my chances of not being able to meet expectations by the end of the sprint.

6.3 Project Experience

The first major challenge I experienced in IT Capstone 2 was a change in requirements of our artefact. When we got back from the semester break and had our first meeting with Keith from S23M, he informed us that he wanted to make some changes to our artefact. In semester 1, the plan was to create several small applications with accompanying tutorials, with each application covering a different aspect of FHIR data. In semester 2, this planned changed to creating a single application in two separate languages that would cover many FHIR data topics. As I was a technical architect in semester 1, I had to rethink the technical design of our artefact to align with these changes. In semester 1 I had planned that each small application would only need a few files and we did not have to worry much about separation of responsibilities due to the small size. We also wouldn't need to be too concerned with how we would maintain high quality code, because as developers working on small application, Rodo and I would each be viewing and using each other's code constantly, and this would act as a passive form of code review. To address the separation of responsibilities issue with a larger application, I decided we should have a very clear structure for our application files. This is where the file structure discussed in section 4.3 was manifested, and it proved to be very helpful in creating a robust larger application. To address the code quality concerns that come with a larger application, I decided we should only commit to the main branch on the git repository through pull-requests, and each pull-request would need to be reviewed and approved before being merged. With a larger application, there were some files Rodo developed which I never needed to view for the features I was working on, so we needed a formal code-review process to check each-others code. This proved essential for us to maintain high quality code.

Another very unfortunate issue that took place was to do with Keith from S23M and his personal health struggles. Throughout the project Keith faced issues that prevented him from interacting with screens for more than an hour or two per day. The severity of his health challenges ebbed and flowed throughout the two semesters. Usually this was not a huge problem, because he was still able to talk with us over voice chat, where we could tell him our progress and he would be able to answer questions and guide us. Keith was very supportive and always did his best to support us. But there were times where we needed to look to others for help in some areas. For example, toward the end of semester 2 when we were writing our code tutorials, Keith was having a particular bad time with his ability to use devices. This meant he wasn't able to read and review our tutorials for about a week. During this time, I decided to reach out to a friend who is a software developer. I asked them to review the tutorial material. They gave me some great feedback, and I was able to make changes that improved the quality of the JavaScript tutorial. This was an important lesson for me. It taught me that things will not always go to plan, and sometimes there are issues which are out of everyone's control that you need to deal with in creative ways. It is important to keep on track with progress as best as possible, as there will inevitably be issues you cannot predict.

Another issue I ran into was to do with merging code. After the first two sprints, the C# project

was ahead of the JavaScript project in terms of front-end design. The C# application had a better looking and easier to use design, and I brought up this issue with Keith and the rest of my team in a meeting. We decided that in the next sprint I would focus on redesigning the front-end. This took up most of the time I had available for that sprint, as it was a fairly large task. At that point we had already developed several features and so the front-end redesign would need to keep these features working as they were, while also making them look good and be easily usable together. I completed the redesign toward the end of the sprint, made a pull-request for my changes and after approval, merged them into the main branch. The problem was, the development Rodo had been doing during that sprint was based off the old design, and there were many merge conflicts that prevented him from merging his pull-request into the main branch. We wanted to make sure we kept on track with our sprint progress, so we needed to work together to make Rodo's code compliant with the new front-end design to merge his pull-request and meet expectations for the sprint. Rodo and I joined the Discord voice chat together and I went through my code and explained all the changes I made. We then went through his code, and together figured out how to integrate his code with my changes. We spent over an hour on voice chat making changes to Rodo's code, but eventually we solved all the merge conflicts and merged his pull-request. The front-end was now much improved and we finished all the tasks for that sprint. This is a good example where developers who are working on different things can create conflicts within the code-base. Often developers will not interact with one-another while writing code because they are working on separate tasks. But separate tasks can interact with each other, like they did in this example. This showed me the importance of considering how changes to code might affect other developer's work. If I had planned out my front-end redesign better initially, I could have communicated these changes with Rodo before he started development and he would have been able to develop during that sprint with these changes in mind.

Overall I thoroughly enjoyed both semesters of IT Capstone. Everyone in my team was professional, friendly, and easy to work with. The client S23M was very supportive and provided excellent guidance throughout both semesters. Our tutor Ross Schamburg did a great job at making sure we knew what was needed to do well in this subject, both academically and for the sake of our relationships with one-another and our client. I learnt a great deal about how the medical system works with data, which was very interesting and gave me motivation for this project because I was able to see how important the work we were doing could be.

7 Individual Chapter: Mackenzie Smith

7.1 Project Setup

7.1.1 Project Management Approach

In semester 1 I was allocated the roles of Developer and Client Relations as my IT major is Computer Science and i have a background in leadership training/communication. It was agreed that i would continue in these two roles as we started Phase 2 of the project in semester 2. As Riley was handling most of the project management, my role as a developer was to ensure i was on top of updates in the project management software, and keeping him in the loop of my progress. This would look like me being allocated a task in the Jira board, asking any clarifying questions if i needed too and specifying technical details, aiming to complete the work within the allocated time, and notifying Riley/the team of the tasks completion. As I was doing fair amount of the C# development, i also informed the team of general progress to ensure both development teams were equally far along. I enjoyed Riley's approach to project management and trusted the team to be honest and productive throughout the project.

7.1.2 Client Expectations

Our team along with our liaison at S23M agreed to fortnightly meetings with an optional weekly meeting if we had any questions/essential communication. During these meetings it was often my responsibility to showcase progress on the C# application, highlighting points of interest and asking questions to ensure what was being developed was what the client wanted. Often the client would be happy with our progress, offering some small feedback here and there as he has a lot more experience in the field than we do. Before these meetings I would try and ensure the rest of the team was in the loop of my progress too, so that we all had a good understanding of our whole product before entering the meetings.

As I had the Client Relations role, it was my responsibility to communicate with our liaison outside of meetings on behalf of our team. This made it simple for S23M to know who to contact in order to ensure the whole team received communication and allowed me to pass information along in a format that is easy for our client to understand. As Project manager, Riley would also sometimes contact the liaison, but I would be involved to keep me in the loop. Our client also had my personal contact number for when we met up in person, as he didn't use Discord - our main form of communication - on his phone.

7.1.3 Team Collaboration

Throughout the whole project our team upheld a strong and healthy work culture. We were productive on our own, and collaborated well when it benefited the team. We all fulfilled our roles well, and were supportive and understanding with each other. As I was a part of the C# development team, I did not work/collaborate with the JavaScript team very much, but we still worked together to create resources like a set of helpful patient ID's that benefited both teams. I - just as every member of our team - was involved in the creation of the team agreement, and didn't have any issues with other members.

As the other member of the C# development team was also the Project Manager, I had more time to focus on making progress on the programming side of things. This resulted in me completing more of the allocated programming tasks. He and I talked about this and were completely fine and on the same page as he had been making progress on the project in other areas, such as the tutorial pages and preparing for tutor meetings. This collaboration worked very well as we both became very familiar with our specific areas, where we were able to optimize the process due to our experience. Where we needed assistance we helped each other out but mainly we played to our strengths and the project was better for it.

7.1.4 Communication Plan

Communication within the team occurred mainly through discord. The discord server had different channels for different topics. For example I would post updates on my progress and things I found helpful in the #development channel. This channel proved to be very helpful for our team as it was very easy to scroll through it to check on each others progress. Whenever talking about the project with the team, I ensured to use professional communication to keep productivity high. We had other channels of communication in the discord that were used, which allowed us to be more relational and build a healthy culture of balance within the team. I would also keep the team in the loop if I was unable to meet a deadline, and ask for assistance.

As I've stated previously, the team also used Jira to communicate. This platform was used to manage our project, but i also communicated in it by adding comments on specific tasks asking questions, and updating the status of the task with my progress. As a rule i would inform the team on both Jira and Discord of any updates from my development.

As I was sending communication to our client i wanted to ensure my correspondence was clear, concise and respectful. I would use consistent subject headings in emails, sincere greetings, provide clear technical updates on our progress without information dumping, and encourage an open line of communication if the client wanted more information or to provide feedback for our team.

Throughout the project, I prepared for our meetings by having the latest version of the C# project ready and running on my device, ready to showcase to the client or tutor. I would keep track of what progress was showcased in the past to ensure I was concise and clear on new progress updates.

7.2 Project Plan and Risk (Phase 2)

7.2.1 Project Planning and Progress

Throughout the project I would frequently check Jira to ensure the work I was doing was the most urgent and correct. As stated previously, I would report on my progress by changing the status of the ticket/task, adding any relevant information as a comment. Jira was essential throughout our project as it kept the most important information present, meaning i wouldn't have to search to find the next task.

As well as updating the team in Jira, I also frequently showed the team screenshots of my progress in the #development discord channel. In regards to project planning, I was involved in the creation of scope and estimation of task length due to my experience in C#. I also helped with the creation of the sprint/release plans. We used a software called Notion to create the tutorial pages, and I am extremely familiar with it, so creating a plan with the client around that was simplified.

7.2.2 Risk Management

The main risks that affected me were Risk ID 3 and 4, outlined in figure 7. As we did not control the test server we were using, we had to rely on it giving us correctly formatted data, and it not going down or losing connection. The correctly formatted data was especially important for the C# project as the data-type structures are defined by the *Firely* packages, and can't be adapted. The server does have some incorrectly formatted data on it, which would crash the program. Error handling was implemented to ensure the program doesn't crash when receiving information from the test server, and a pop up was created to show the user what error occurred. This error would show what part of the data received from the server was incorrect, so it could be avoided in the future. The server going down risk was managed by implementing one 'global' variable that stores the server URL. This would allow the whole program to continue to function by just changing one variable in the back end. As we did experience some outages throughout development, this became very helpful and worked as intended. I informed the other members of the team of my solution to assist in their development. There were no other risks that presented in my experience of the project and i was involved in the creation of the Risk Register.

7.3 Project Experience

As mention in the group section of our report, our first meeting with the client in semester 2 informed us of a rather large change in scope of the project. This was a little alarming at first, as I was prepared and researched for the project we planned in phase 1. But as i understood more about the new project design, i felt I had more clarity and a better plan than before. The old phase 1 plan was to make tutorials on varying topics, none of which had yet been decided by the client, which presented a lot of uncertainty. The new phase 2 plan created in week 1 gave myself and the team a lot more clarity on the deliverables and also allowed us to more accurately estimate how long tasks would take. This newer project design better fit the needs of the client and tested my adaptation skills.

Throughout the project, our liaison at S23M was experiencing some personal medical complications which often resulted in him being unable to review our work. This meant that when I would showcase progress or update him, there would often be minimal feedback, which made it hard to know I was on the right track. This challenge was overcome by doing more research in field, and more critically reviewing our own work. As a developer It's often challenging to understand the users experience, as we have a much different understanding than them. This resulted in myself asking for feedback from the rest of the team on my deliverables, and presenting them to our client in the format which best suited him. For example high levels of light negatively affected him, so I - along with other members of the team - created a dark mode to assist his experience. This taught me that i can do my own due diligence to ensure the quality of my work.

My experience of the team throughout the project was great. As a group of university students all doing different subjects, our timetables didn't always match up, so we would be doing work at different times. This could've presented an issue, but because we had built rapport and trust from the previous semester, it was easy to know that team members were being honest. Our group very rarely missed a deadline and was very communicative when it was of benefit to the team. This showed me that trusting teams are productive teams, even if at a first glance it might not seem that way.

At times during development the two different projects were rather different in design. This meant that one development team had to change their design to fit the other, to ensure a smooth user experience. As I was doing a lot of the C# development, I often found myself on a roll working for a few hours and making good progress. I would sometimes not check on the progress of the JavaScript team, which meant that sometimes we had both done work that clashed with one another. There were situations where we both compromised and had to re work our design slightly. In future I would make a clearer plan for the visual design of the project at the start, to ensure no work is wasted.

I very much enjoyed working on this project with this team, and I have learnt a lot about working in my industry.

8 Individual Chapter: Rodo Nguyen

8.1 Project Setup

8.1.1 Project Management Approach

Max and I chose to develop the app using JavaScript. Despite ROMA development developing the app in 2 languages, we made sure to keep 2 teams in the loop and, more specifically, aimed to have as many similar features between the 2 apps as possible. To achieve this, we laid out the main components to implement, which was quite straightforward as they were already defined plainly in the IPS composition (Figure 1).

As mentioned in the above sections, Atlassian Jira was used as the main project management and communication tool regarding our tasks, features and overall progress. In addition to the requirements Riley - the project manager - has gathered from the client and defined in Jira, I as a developer would often be able to expand these tasks, express them explicitly in technical language and then identify and create detailed sub-tasks based on this expression.

For example, from an original abstract requirement "Display Diagnostic Report of the IPS", I created sub-tasks/sub-user-stories that developers can easily relate to and implement:

- Code a function that gets Diagnostic Report (DR) data of a Patient
- Code a convertEntry function that handles the DR data
- Code a component that DR data in a table

Since the industry supervisor only provided us with an overall vision of the final product, ROMA team has great flexibility on the specific features as long as they add value. In some cases, as I had worked with the features for a good amount of time, new ideas for additional features arose. I would bring these ideas to the team in our fortnightly meeting to discuss their priority and the value they will bring to the client or the final product. The client was there to give the final approval as well.

For instance, each Diagnostic Report (or each row) has a number of References and Results associated so it did not look good if I tried to display all of this information in a cell. Note that these kinds of discoveries only happened after we have spent some time closely examining the data so it was hard to plan everything at the beginning. So I extended it with a modal to display this information. In the end, the additional tasks were:

- A "View result" button on the right-most column
- Modal containing References and Results appears when a user clicks the "View result" button

It is important that I updated my progress with other team members and the S23M supervisor. This was done mostly in Jira so that all related stakeholders can receive instant update notifications via Jira App or Jira's email system. In addition, for any major changes or problems, I also sent a message and tagged related ROMA developers in the chat channel to make sure I had their full attention and the issue could be resolved in the fastest possible manner.

When a component/task was finished, I would also usually attach screenshots, a short description,

and any shortcomings (if applicable) in the comment section (see an example in Figure 21). The ticket was marked as "In Progress" when I was working on the ticket to avoid work collision and marked as "In Review" when they were being reviewed by other team members and/or industry supervisors. After that, if no issues arise, I would mark it as "Done".

8.1.2 Client Expectations

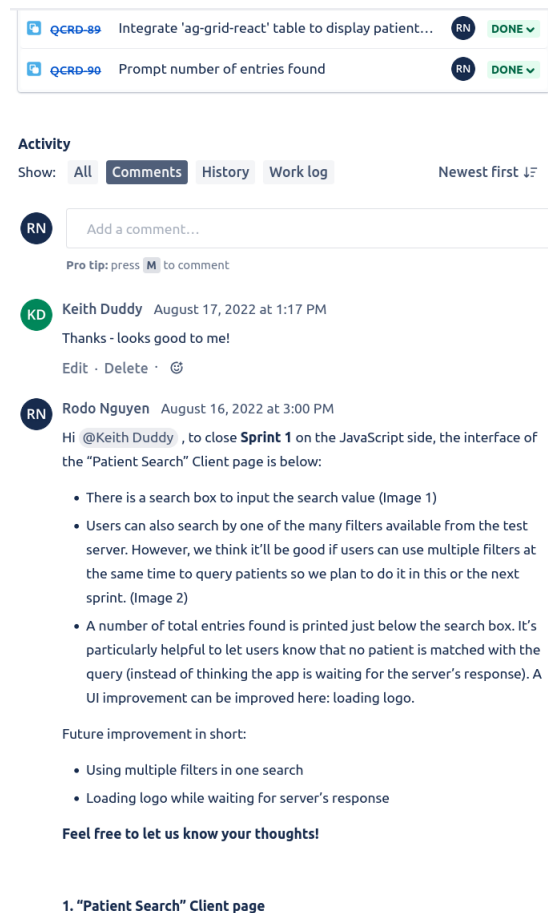
We met with the client fortnightly to discuss our upcoming plan and at the same time help form their expectations. With that said, ROMA and I tried our best to communicate efficiently and effectively so that the client had a good understanding of the project's progress and what I planned to finish in the next 2 weeks. I would list the features/functionalities I was implementing and describe or present a sketch if necessary. These features/functionalities would also be recorded as tickets on our Jira's Agile board. Since our industry supervisor is a non-technical executive, I focused on presenting the visual side of the product - the web application's User Interface itself - since it explained what we had been doing more vividly.

It's no less important that I was responsible to share any delays or difficulties in delivering the tasks. This would make sure the stakeholders had realistic expectations and our team could take into account these possible delays in future planning.

8.1.3 Team Collaboration

I worked closely with Max to deliver the JavaScript App. So, for our collaboration strategy, I discussed with him in the first-week meeting what our collaboration plan was going to look like. Basically, it was about the approximate total available hours that we were going to put into this project each fortnight, what days we were likely to do it, and any uncertainties that may prevent us from finishing it, such as assignments from other units, jobs, and other personal commitments. With this information, I could anticipate how much work we were able to do in a fortnight and lay out some backup plans in case of problem arises.

With Jira as the project management tool, I would create additional tasks if necessary, discuss them



1. "Patient Search" Client page



Figure 21: A discussion in Jira

with Max/ROMA team fortnightly if they were prioritised features for the app, and split them reasonably between Max and me (considering one's available time and specialty).

8.1.4 Communication Plan

Atlassian Jira platform was chosen to be our main communication tool as well as the backlog of all the tasks and features that happened in the development process. In Jira's task board, after the fortnightly meetings, I would assign the tasks I was going to do to my name. This helped to provide clear accountability of who does what and avoid task conflict when 2 people doing the same task. By default, the industry supervisor, Keith had access to view all of the tickets, the progress and the assignee but I could also tag him in the comment section to raise important information or ask clarifying questions. Emailing was the second option and I would use it if Keith did not respond for more than 2 business days.

Jira's task board also has a section that allows me to add the progress status with the ticket, whether it's 'to do', 'in progress', 'in review' or 'done'. This allowed Riley, the project manager, or Keith to view the team's progress as a whole easily.

In addition to Jira, I usually communicated with ROMA team through Discord (using either video call or text) for technical discussion, progress, meeting preparation and miscellaneous things like meeting absence notification.

Moreover, we also utilised fortnightly meetings to show the client our progress by presenting / demonstrating the application visually and in real-time, which was more efficient in helping the client capture what we had been doing. ROMA team also had different fortnightly meetings where I summarise the tasks/features I had done, ask for assistance about some technical challenges or offer help if others were struggling with something. Overall, I found online meetings beneficial since we could exchange information, resolve certain types of problems faster, and interact in a way that can't be replaced by text messages.

8.2 Project Plan and Risk (Phase 2)

8.2.1 Project Planning and Progress

In ROMA scope, main features/tasks were first collated in our fortnightly meeting. This was where I discussed with the team, added the must-have features in the most abstract form, and then decided which ones were crucial and would actually be implemented. Despite having 2 separate development teams, we still did this in the general meetings in order to unify the main features between the 2 apps and allowed potential users to switch between the apps without being baffled by the difference.

Within the JavaScript team, I also participated in expanding the 'parent' tasks by adding specific application features and components. Then, I would have a quick chat (via video call/text messages) to understand each person's technical strengths and total available time in that sprint so as to assign tasks reasonably for each one.

A catch-up at mid-sprint was often organised for the team to briefly report on the progress. The purpose was the same as the fortnightly meetings: getting updates from teammates, making changes to the task if necessary, and opening space to request support or offer one.

8.2.2 Risk Management

Regarding the technical risks, there are 2 'likely' risks: the test server being made unavailable and different data formats breaking the application. Firstly, unavailable server happened around 3 times throughout semester 2 and it caused some difficulties to test the functionalities of the app. To be more specific, I could only code the app but not be sure how it runs with the real data from a live test server. Therefore, to deal with this issue, I would code the parts where I could without needing the server and get back to the parts which require the server later. Another solution was that the team searched for another working test server and make it the backup server, which proved to work well for us. Secondly, I resolved errors caused by different data formats by adding 'error handler' functions which were like a pre-processing step before parsing the data to the main components. However, this would not guarantee to prevent future errors from happening so I informed the ROMA team as well as Keith about this issue. In fact, we followed the documentation by the FHIR website so this is a problem of the test server not having the correct data format. Keith also confirmed this and said wrong-format data entries could be safely ignored.

There was also a risk that was related largely to the human factor: someone being behind schedule. Since we were all students studying and/or working, having time constraints with other commitments was almost inevitable. So I had to consider thoroughly when was the best time to set a meeting, how many tasks were appropriate for me and my teammates, etc. Fortunately, ROMA had a very supportive culture and provided regular check-ups so no serious incident regarding late delivery happened during this project.

Importantly, all the potential risks identified by me or anyone else in the team were brought up in our meetings as agreed in the beginning in order to keep others informed and prevent them from happening in the other app. Particularly, 'different data formats breaking the application' had happened several times during the development stage and as a result, had been brought up in our meetings or Discord channels to warn others.

8.3 Project Experience

It's inevitable that a problem would arise when we were delivering the product and a thoughtful, professional approach must be done on such an occasion. There was one time, a teammate was behind the schedule and a feature was delayed. However, from my observation, he was a good developer and had always done his part on time and with high quality. So, only 1 day after that task's due date, I decided to check up on him before making any false assumptions. It turned out that he was busy with other assignments and his part-time job too, which was a valid reason for falling behind in this project. It was pretty early to set up any formal team meeting to discuss this issue so I only messaged him on Discord informally.

After we had talked about the 'why', he also added that he would be available on Monday and finish the feature. Otherwise, I would take over the task to prevent further delay in the project. Fortunately, he did keep his promise to finish it on Monday and no further intervention was required. I think having proactively approached my teammate who seemed to be having trouble and discussed the solution together in a timely and suitable manner helped resolve the problem smoothly.

The second experience is that our test server sometimes turned unavailable temporarily during phase 2. As a result, it impacted our workflow while I was writing the code to interact with the server. I immediately notified the team on Discord about what happened and it turned out that other teammates also experienced the server shutdown. A temporary solution, which of course I let others know, was that we could work on parts where we don't need the server and get back to them when the server becomes available again. However, this was not a sustainable solution for a likely incident like this. Through a quick discussion and finding an alternative together, one person found out that we can use a different but similar test server listed on the FHIR test server webpage which could be utilised in the same way as the main one if it goes down. Therefore, in the event of the main server being shut down, I could easily replace the server address and keep on developing the app. In this situation, I had been able to identify early issues in the development process even the smallest bit and communicate early with the team, which pushed the team to have faster action and find the best solution for the issue.

The last experience was about the lack of code management at the beginning of our development phase. According to the standard, we would have full access to S23M's GitHub repository and the procedure to contribute to the project was creating a development branch, making changes to it, and then merging the development branch with the main branch. This procedure helps protect the main functional branch and allows easy revert should a change from one of the developers causes an error. However, I was not aware of this and thought that only the technical leader can have full access and the remaining developers had to fork the S23M's repository to their own accounts and contribute from their versions but in truth, everyone was having full access to S23M's repository except me. This was actually what I did and it continued to happen until sprint 2 when Max was helping me fix a conflict error from merging my separate repository with S23M's repository. I myself found my procedure a bit lengthy and raised it to Max and to this point, I knew that I don't have full access to S23M's repository and an error had happened at the beginning when the client gave us access. Part of this also fell into ROMA for not formalising the procedure and not communicating it across the development team. As soon as I was aware of the issue, I contacted Keith to get access which was done in 1 business day.

After all, my progress was impacted mildly but the experience could have been better if ROMA team had discussed and formalised the code developing procedure clearly. I did well on my part to have raised the question when noticing something wrong and took action without delay to resolve issues.

Overall, different kinds of experiences happened during this project and I'm glad to have had the chance to go through them and learned valuable lessons. Especially, from my perspective, there needs to be professional, clear and timely communication between all stakeholders and a supportive working environment so that a project can succeed and each individual can have the best experience from the journey.

9 Appendix

The screenshot shows a Jira task view for 'C# Development (See Sub-Tasks)'. The task is in 'In review' status. The description field is empty. The due date is set to 'None'. A progress bar for 'Child issues' is at 100% Done. The child issues list includes:

Issue ID	Issue Name	Status
QCRD-106	Display Immunizations	DONE
QCRD-107	Display History of Procedures	DONE
QCRD-108	Display Medical Devices	DONE
QCRD-109	Diagnostic Results	DONE

The details panel on the right shows the assignee as Mackenzie Smith, labels as 'None', and the sprint as 'Final Capstone Deliverables'. It also shows the original estimate as 0m, no time tracking, and development actions like 'Create branch' and 'Create commit'. The reporter is Riley Head. The task was created on July 27, 2022, at 5:46 PM and updated 1 hour ago.

Figure 22: Appendix Item 1 - A Jira Task

The screenshot shows the Jira Dashboard for the 'QUT Capstone Roma Development' project. The 'All sprints' view is active, showing a Kanban board with columns for 'TO DO 1 ISSUE', 'IN PROGRESS 2 ISSUES', 'IN REVIEW', and 'DONE 8 ISSUES'. The 'TO DO' column contains 'Final Feedback Form' (due 02 NOV) and 'QCRD-129'. The 'IN PROGRESS' column contains 'Video' (due 04 NOV) and 'Report' (due 04 NOV). The 'DONE' column contains several completed items, including 'C# Development (See Sub-Tasks)', 'JavaScript Development (See Sub-Tasks)', 'Draft artefact agreement', 'Document JS App Tutorials', and 'Supervisor Feedback Form'. The dashboard also shows a sidebar with navigation options like 'Roadmap', 'Backlog', 'Board', 'Reports', 'Code', 'Project pages', and 'Project settings'.

Figure 23: Appendix Item 2 - Jira Dashboard

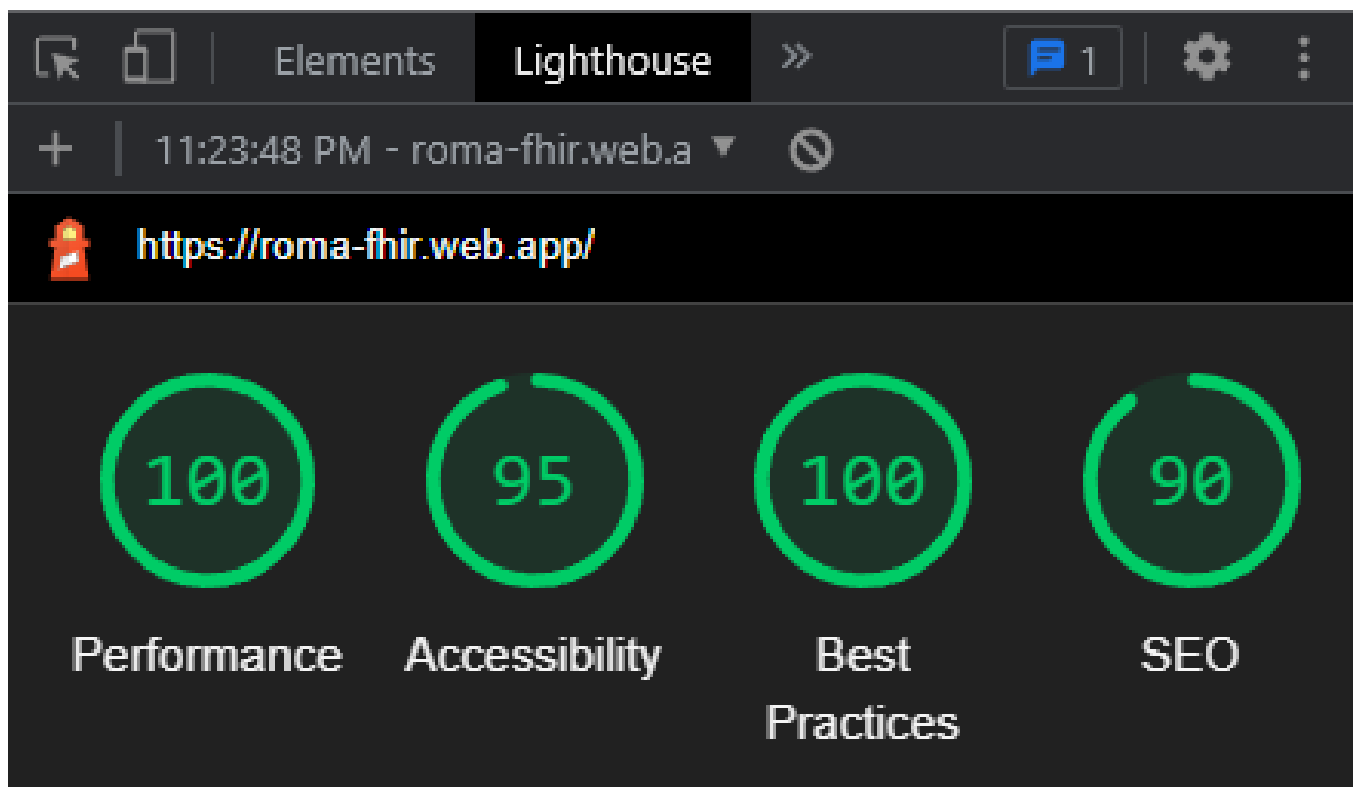


Figure 24: Chrome Lighthouse analysis for JavaScript application homepage

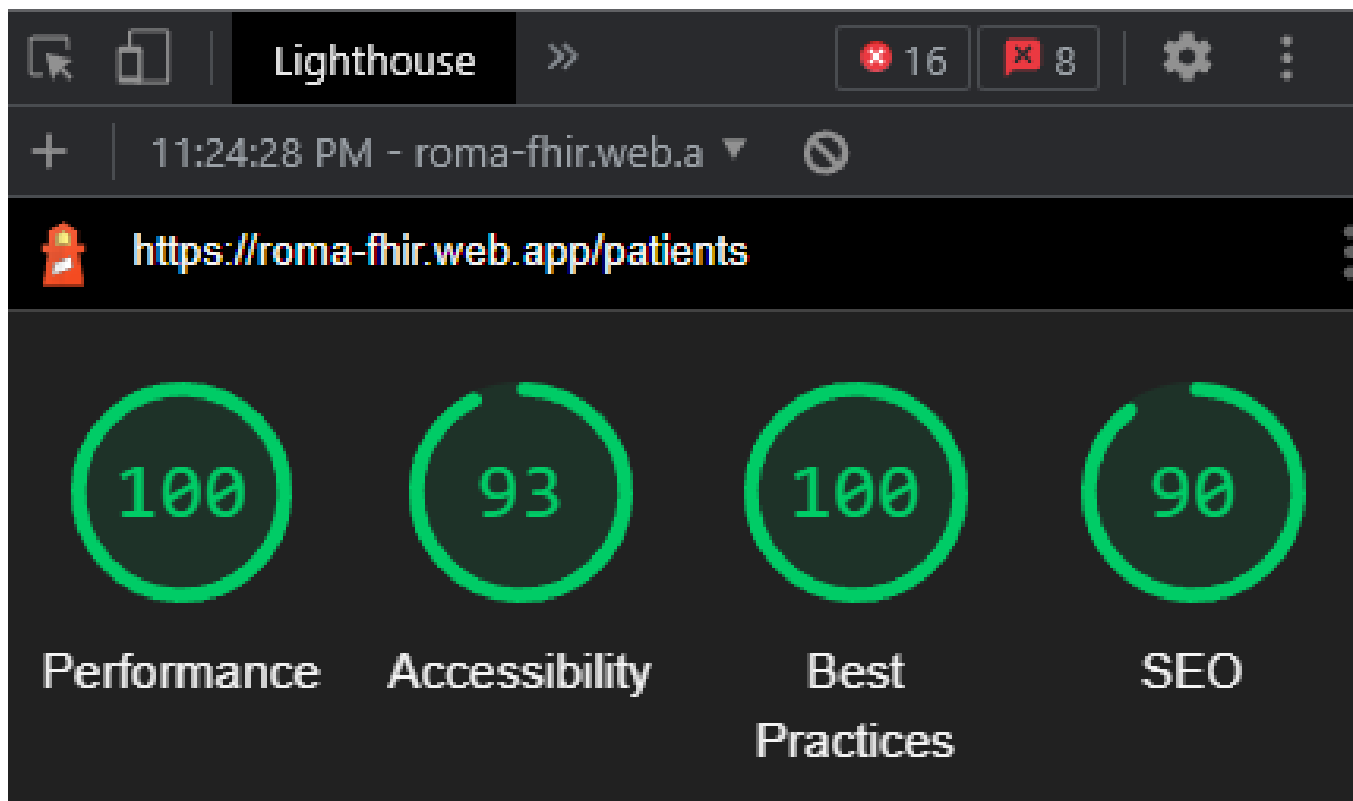


Figure 25: Chrome Lighthouse analysis for JavaScript application `/patients` page

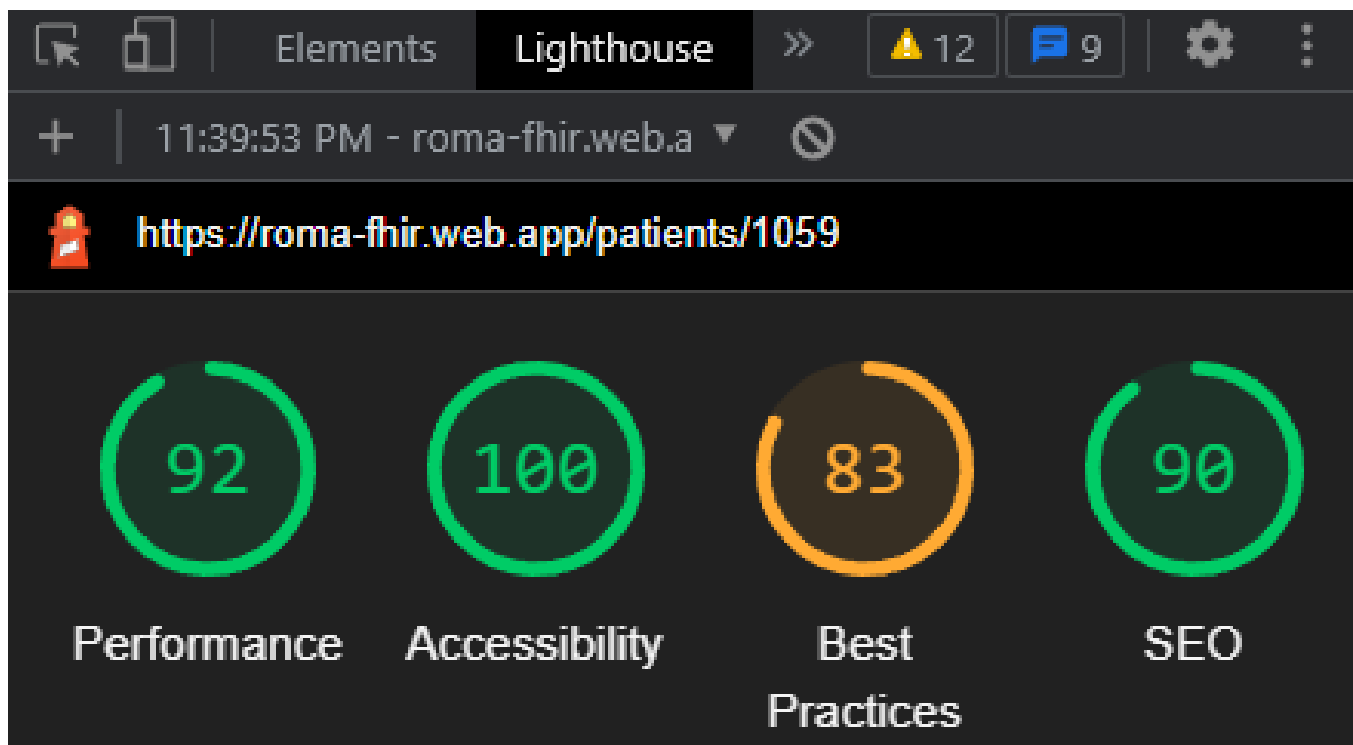


Figure 26: Chrome Lighthouse analysis for JavaScript application /patients/{id} page

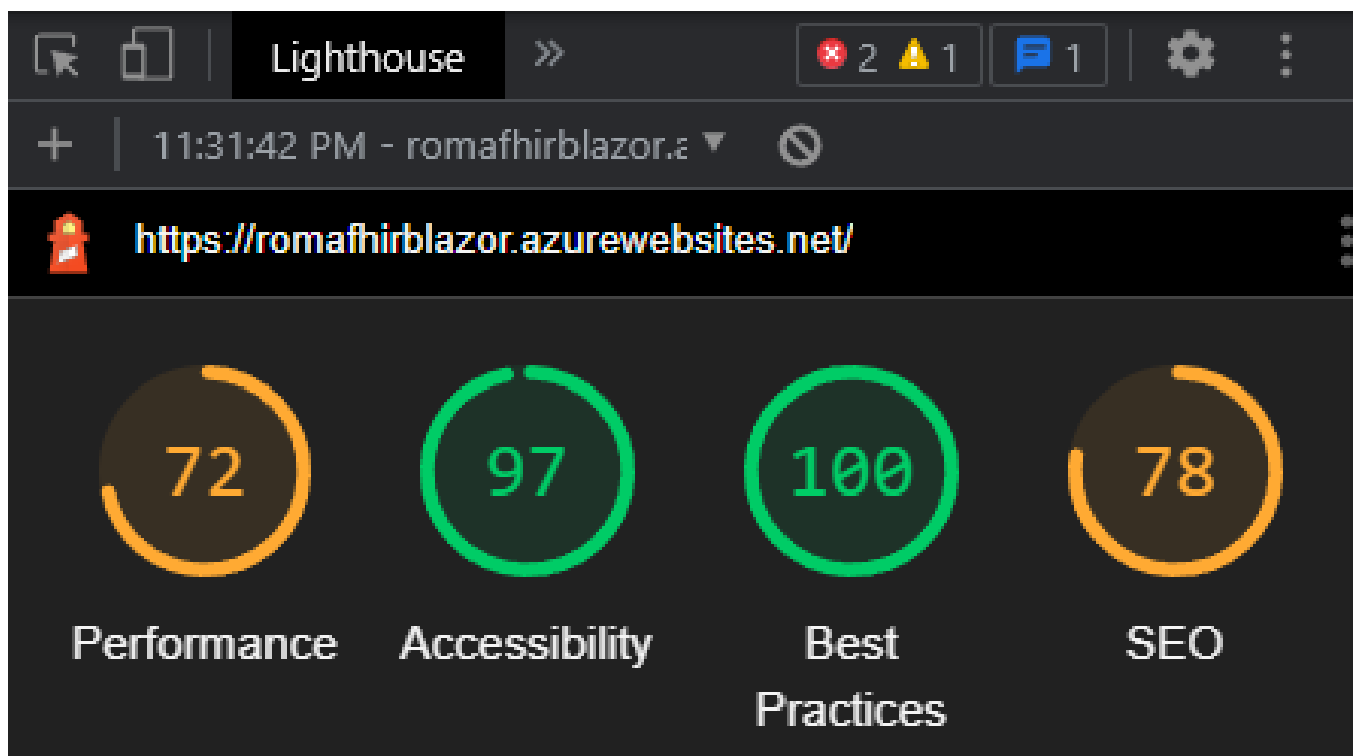


Figure 27: Chrome Lighthouse analysis for C# application homepage

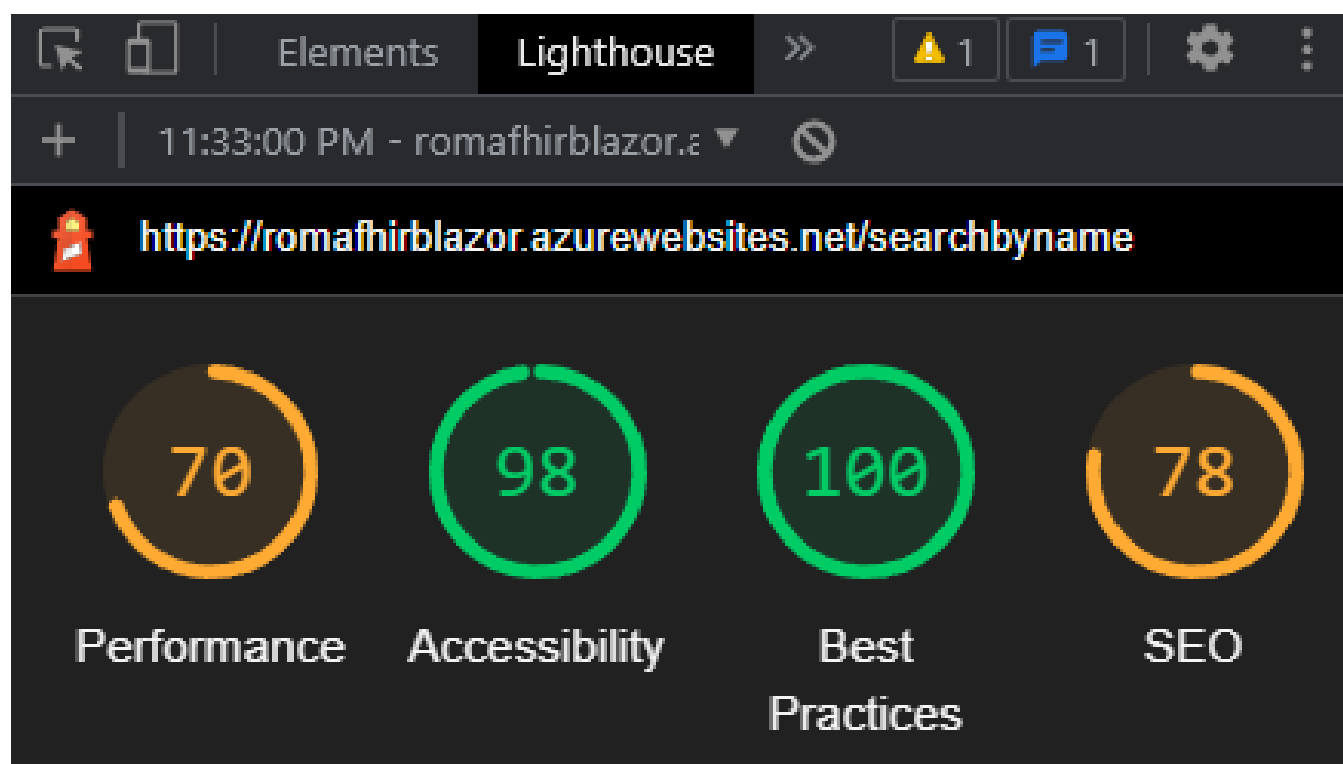


Figure 28: Chrome Lighthouse analysis for C# application /searchbyname page

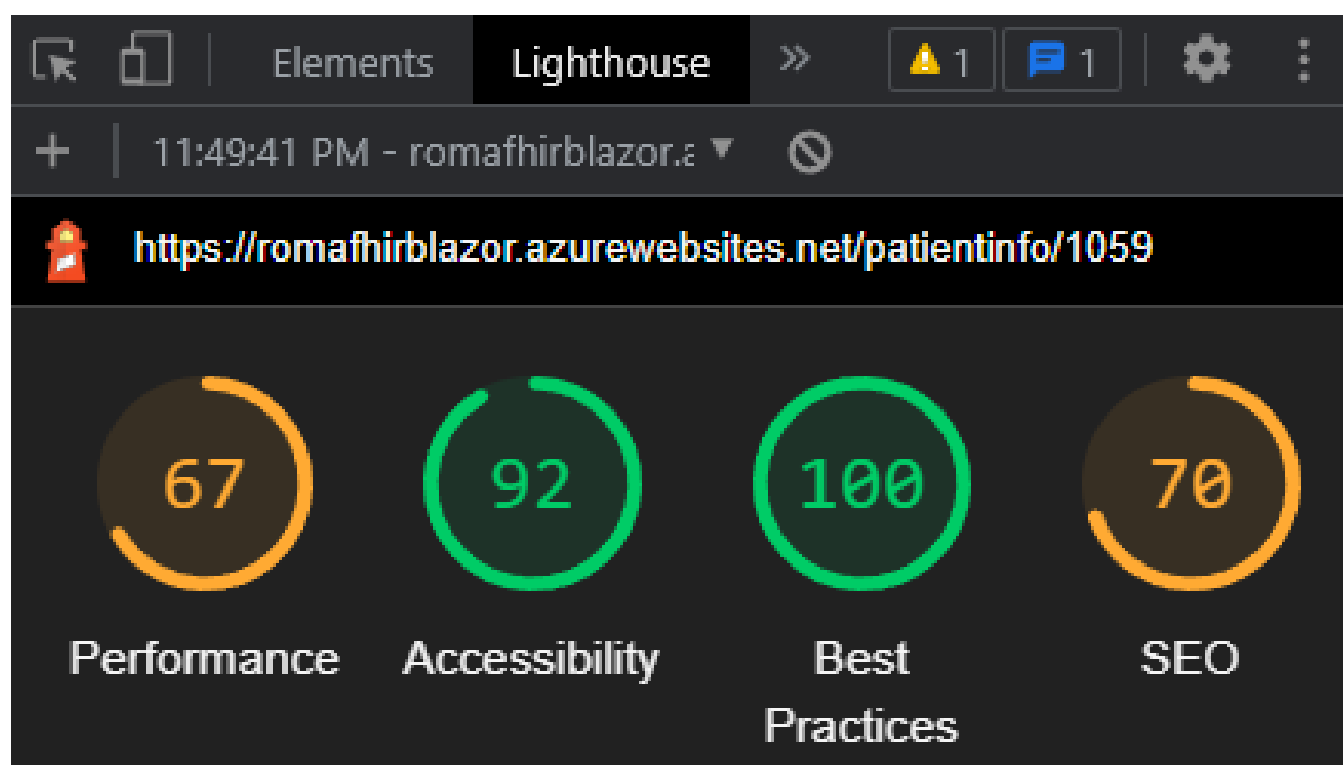


Figure 29: Chrome Lighthouse analysis for JavaScript application /patientInfo/{id} page