Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

FAIRVASC Query Application

Sarah Elizabeth Stafford-Langan

August 8, 2021

A Final Year Project submitted in partial fulfilment
of the requirements for the degree of
B.A.Mod. Computer Science

0.1 Declaration

I, Sarah Elizabeth Stafford-Langan, hereby declare that this Final Year Project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

Signed: ____________________ Date: ____________________

0.2 Abstract

ANCA-Associated Vasculitis (AAV) is an auto-immune disease that causes inflammation in small blood vessels, often leading to organ damage; most commonly kidney disease and lung disease. The cause of AAV and the factors involved in the occurrence of flare-ups are mostly unknown, although it's thought that genetic and environmental factors play a role.

The disease affects a little more than 100,000 people in Europe, resulting in very few cases in any one country. The FAIRVASC Project was started by the European Vasculitis Society. Its primary goal is to make national registries containing medical information about AAV patients consistent across Europe. By granting researchers access to a greater amount of patient data, they will be able to learn more about the disease, its onset, and possible treatments and cures.

This project involves the creation of a web application that will act as the front-end for researchers accessing information from FAIRVASC registries.

0.3 Acknowledgements

Firstly, I would like to thank my project supervisor, Professor Declan O'Sullivan, for his continued support, guidance, and feedback throughout this project.

I am grateful to the FAIRVASC project team for their insight into the disease ANCA-Associated Vasculitis, and their feedback on the project.

I would also like to thank my parents, whose continued support and motivation helped me get through this project.

Contents

List of Figures

List of Tables

List Of Code Snippets

1 Introduction

1.1 Motivation

FAIRVASC [1] is a European collaborative project, funded by the European Union, to build an international database of cases of ANCA-associated Vasculitis (hereafter AAV). ANCA are autoantibodies that attack white blood cells. The disease causes inflammation in the capillaries and can lead to many organ-related complications, such as kidney disease, diabetes, and lung hemorrhaging.

AAV affects 150 people per million in Europe [2], which is approximately 110,000 people. The risk of developing AAV increases with age, with its maximum prevalence in male and female patients over 75 years old [3].

A database of all European cases will improve researchers' ability to investigate the potential genetic or environmental factors in the disease's development and flare-ups and patients' prognoses, and will allow doctors all over Europe to use a wider set of previous cases to determine the best course of treatment for their patients.

FAIRVASC hopes to demonstrate how to solve "the challenge of fragmented and heterogeneous registries" [4], a process which they hope will be then undertaken by other organisations to create international registries of other rare diseases.

The current challenges of the project include discovering countries where AAV-related registries or databases currently exist, normalising the data across platforms and languages, and following legal, ethical, and regulatory constraints on accessing and using the data. All European registries chosen must follow FAIR Guidelines (Findable, Accessible, Interoperable and Reusing digital assets) [5].

A standardised schema is being created by the FAIRVASC team using the Resource Description Framework (hereafter RDF), a triple-store non-relational database. This schema will be applied to existing registries in participating countries to "normalise and access registry data across borders."

At present, seven registries are partnered in FAIRVASC, led by the EUVAS Registries Group

[6]. The registries are from Ireland, the UK, Poland, Sweden, France and Czechia, as well as a joint registry of cases from Germany, Austria, and Switzerland. The second part of the project includes expanding the number of registries included in the FAIRVASC system of databases.

1.2 Project Objectives

My brief for this project was to allow users with limited technical knowledge to pose complex queries to one or more databases and represent the results visually.

O1 Conduct a review of data visualisation options.

O2 Design and create a web application front-end that forms a query based on user input.

O3 Implement a web server back-end that poses the user's query to the relevant database or databases.

O4 Use data visualisation to convey the query results to the user effectively.

1.3 Personal Goals for the Project

At the beginning of this project, I set a number of personal goals for myself.

P1 I wanted to learn more about non-relational databases, as I didn't have much experience with them. This project caught my eye because I would be working with RDF, Resource Description Framework, a non-relational graph database.

P2 I wanted to learn more about data visualisation techniques so the data visualisation aspect of this project appealed to me.

P3 I discovered that I would be reporting to a larger team, showing them my work and receiving feedback. I wanted to improve my ability to report to a team and to receive and implement feedback.

P4 This is the largest project that I have worked on to date, and the first that has needed project management controls to be implemented. I wanted to learn how to walk the line between under- and over-planning for this project.

P5 I wanted to learn how to recover from major setbacks during the implementation phase of the project, should any arise

P6 I wanted to learn how to write a formal report documenting the successes and setbacks of the project.

1.4 Report Outline

The following section outlines the background research that went into the project.

Chapter 2 Background discusses the background of the project, such as discussing the disorder, ANCA Associated Vasculitis, in more depth and exploring the FAIRVASC team and their existing frameworks.

Chapter 3 Design outlines the client's functional and non-functional requirements, as well as preliminary research and planning phases of the project. A paper prototype that was developed is also shown and discussed.

Chapter 4 Implementation summarises the technology chosen to implement this project and explores how the the front- and back-ends were implemented as well as any issues that arose during implementation.

Chapter 5 Evaluation discusses the user evaluation that was planned, the success of meeting the client's functional and non-functional requirements, and the future work that is required before the deployment of this application.

Chapter 6 Conclusion assesses the project in terms of meeting the project's objectives and the personal goals of the author.

2 Background

This chapter discusses the background research that was done surrounding this project. First, a glossary outlines the medical terminology used within this document. Then, the disorder ANCA-Associated Vasculitis is explained in detail and finally the FAIRVASC team and the existing technological infrastructure are discussed.

2.1 Medical Glossary

ANCA —Anti-Neutrophil Cytoplasm Antibodies. Autoantibodies that attack a type a white blood cell called "neutrophil" [7].

AAV —ANCA-associated Vasculitis. A collection of diseases marked by the presence of ANCAs and inflammation of capillaries. This inflammation can lead to damage to the kidneys, as well as to other organs, such as the heart, the lungs, or the brain [3].

Capillary —The smallest blood vessel in the human body. Capillaries supply oxygen to and remove carbon dioxide from the insides of organs and perform gas exchange within the lungs [8].

Dialysis —A procedure to remove waste products from the blood when a patient's kidneys no longer function. Dialysis is most commonly done by extracting the blood from the body and filtering it using a machine. This procedure lasts for four hours and is usually performed three times per week [9].

Eosinophil —A white blood cell whose presence often indicates a parasitic infection, allergic reaction, or cancer [10].

ESKD —End-Stage Kidney Disease. The final state of Chronic Kidney Disease, where kidneys can no longer remove waste from the bloodstream. A patient can only survive using dialysis or a kidney transplant [11].

Granulomatosis —A chronic disorder where multiple granulomas, inflamed nodules of newly formed capillaries, form in the body [12].

Peripheral Neuropathy —Numbness, tingling, weakness, and pain in the extremities [3].

Polyangiitis —An inflammation disorder ("-itis") affecting many ("poly-") small blood vessels ("angi-") [13].

Necrosis —The death of living tissue [13].

RKD —Rare Kidney Disease Registry. A database containing the medical information of AAV patients in Ireland [14].

Serotype —A group or subset within a species of microorganism which shares distinctive surface structures. A single microorganism species can have many serotypes [15].

White Blood Cell —Part of the body's immune system. A cell found in the blood that fights infections and diseases [16]

2.2 ANCA-Associated Vasculitis

ANCA-Associated Vasculitis (AAV) is a collection of autoimmune disorders where ANCA cells attack white blood cells in the body. ANCA cells can cause inflammation or necrosis of small blood vessels — "capillaries, venules, arterioles and small arteries" [Yates et al. 2017 [17]].

Three separate disorders are grouped together and studied as ANCA-Associated Vasculitis — Microscopic Polyangiitis (MPA), Granulomatosis with Polyangiitis (GPA, formerly Wegener's Granulomatosis), and Eosinophilic granulomatosis with polyangiitis (EGPA, formerly Churg-Strauss Syndrome). This grouping is a result of the association each disorder has with ANCA cells and "similarities in the clinical and pathological features among these diseases", according to the FAIRVASC website [3].

GPA is a necrotising inflammation disorder usually involving "the upper and lower respiratory tracts and kidneys" [3]. Many patients also experience peripheral neuropathy.

EGPA also usually affects the respiratory tract, but occurs when the patient's eosinophil count is high [17]. 100% of EGPA patients have asthma [3] and many suffer from kidney damage, gastrointestinal distress, and cardiac failure.

MPA involves severe kidney disease [3] and the necrosis of smaller blood vessels, often in the lungs, without the presence of granulomatosis [17].

The two main ANCA serotypes associated with AAV are myeloperoxidase (MPO) and proteinase-3 (PR3) cells [3]. One, both, or neither of these serotypes can be present in patients. GPA patients are often PR3-positive and MPA patients are most frequently MPO-positive, though PR3 cells are often also present. Less than half of EGPA patients are MPO-positive.

The onset of AAV is triggered by unknown causes, though genetics and environmental factors, such as infection, drugs, or silica dust, may play a role. Patients with AAV have a poor quality of life without medication. However, existing medication can have serious side effects, such as infertility or diabetes. Because one of the organs commonly affected by AAV is the kidneys, many patients are eventually diagnosed with end-stage kidney disease and cannot survive without dialysis or a kidney transplant. AAV is incurable and, while medication can reduce the risk of death from the disorder itself, many patients die from co-morbidities or organ failure.

There are approximately 110,000 people in Europe with AAV [2] and its presentation is most common in the age range of 75 and up, and is slightly more common in men than women.

2.3 FAIRVASC Project

As discussed in *section 1.1 Motivation*, access to an increased number of AAV patients' medical information will give researchers better opportunity to investigate the potential genetic or environmental factors that cause the disease to develop or progress. This is the primary goal of the FAIRVASC research project. The project involves "leading scientists, clinicians, and patient organisations" [1].

There are three teams within the FAIRVASC project.

The Harmonisation Implementation Team, HIT, is comprised of AAV experts from each collaborating registry. HIT members discuss clinical terms used within the registries to create a number of "harmonised clinical terms" [4] which the Work Package 3 (WP3) team integrates into the FAIRVASC ontology.

The FAIRVASC Implementation Team, FIT, members are computer scientists who use the FAIRVASC ontology to uplift the data from collaborating registries into "linked data format RDF" [4].

The Query Implementation Team, QIT, designs the queries for the FAIRVASC project. The queries are federated, which means that they access multiple registries. The data returned by the queries must be aggregated and anonymised in order to protect patient privacy.

The queries designed by QIT, explored in *subsection 3.1.4 FAIRVASC SPARQL Queries*, are implemented in this web application.

The clients for this project were the FAIRVASC and Query Implementation Teams, as the teams work together closely.

2.4 Chapter Summary

This background information on AAV and on the FAIRVASC team was paramount in determining how to approach the design of this project, which is discussed in the following chapter.

3 Design

This chapter reviews the design approach taken during this project. The FAIRVASC and Query Implementation teams were the clients for this application and their requirements and the project's scope were discussed with them and are outlined below. Next, this chapter summarises the research done before beginning to program the application; the ethics of the application are considered, the existing FAIRVASC technologies are further explored, and the advantages and disadvantages of possible technologies are discussed. Finally, the paper prototype that was designed and the feedback from the FAIRVASC team are discussed.

3.1 Client Requirements

Early in the planning phase of this project, the requirements for this project were discussed with the FAIRVASC team. Six requirements were agreed upon which have been split into functional and non-functional requirements and explored below.

3.1.1 Functional Requirements

R1 Enable users to review and compare patient data.

R2 Pose SPARQL queries to RDF databases and parse the results.

R3 Allow users to construct SPARQL queries.

The primary purpose of the application is to allow users to review and compare information about AAV patients. The end user will be able to use this application to explore the correlation between different combinations of diagnoses and outcomes.

Since the medical data are stored in RDF databases, the application's back-end needed to be able to send queries to multiple databases in the RDF query language, SPARQL, and parse the responses.

Users need to be able to construct queries that they want to pose to the databases, so that they can compare only relevant patient data.

3.1.2 Non-Functional Requirements

R4 Users should be able to pose queries without needing a prior knowledge of databases or query languages.

R5 Users should not have to wait a long time for results.

R6 Results should be communicated visually.

Users with no experience working with SPARQL or RDF needed to be able to pose queries to the databases. Queries had to be broken down so that users could easily construct complex queries. The FAIRVASC team created a list of queries to be implemented in the final design.

The response time of the RDF databases is beyond the control of this application, so the application needed to send queries and process and display the results quickly in order to minimise the time spent waiting for results.

Lastly, the application should be simple to understand and easy to use and should display the results in a human-readable way, using data visualisation and results tables. The visualisation of results allows users to more effectively compare and contrast the results from the databases.

3.1.3 Out of scope

One element of the application determined to be out of scope for this project was user authorisation and authentication. User authentication is essential to protect the medical data of the patients in the databases. A back-end technology that would support user authentication was chosen so that this support can be added in the future. This is discussed further in *section 5.4 Future Work*.

3.1.4 FAIRVASC SPARQL Queries

The FAIRVASC QIT team, in collaboration with FIT, created a list of queries that they wanted to be implemented in the application.

Users should be able to view the number of patients in the registry who have a certain outcome: either an ESKD diagnosis or death. The user should be able to group their results by ANCA specificity, by sex, by main diagnosis or by any combination of these data elements. The user also needed to be able to view the total number of patients within each contributing database.

One query supplied by the QIT team is seen in *Snippet 3.1*. This query determines if the patient has an ESKD diagnosis and returns the number of patients in the registry who are

```
SELECT (COUNT(?patient) as ?count) ?anca_spec ?diagnosis ?sex ?has_eskd
WHERE
{
   ?patient a fvc:Patient ;
         fvc:hasANCA ?anca ;
         fvc:hasDiagnosis ?dia ;
         fvc:gender ?sex ;
         fvc:hasOutcomes ?outc .
   ?anca fvc:ancaSpec ?anca_spec .
   ?dia fvc:mainDiagnosis ?diagnosis .
   BIND ( EXISTS { ?outc fvc:dateOfESKD ?date } AS ?eskd_bool )
   BIND ( IF ( ?eskd_bool, "true", "false" ) AS ?has_eskd )
   FILTER (
      ?anca_spec = "MPO positive" &&
      ?diagnosis = "Microscopic polyangiitis (MPA)" &&
      ?sex = "Female"
   )
} group by ?anca_spec ?diagnosis ?sex ?has_eskd
```

Snippet 3.1: ***One of the SPARQL queries supplied by QIT, returning the number of patients in the registry who have ESKD, are Female and MPO-positive, and have an MPA diagnosis***

Female and MPO-positive and have an ESKD and MPA diagnosis. SPARQL queries are discussed in greater detail in *subsection 3.2.3 SPARQL Protocol and RDF Query Language* and *subsection 3.2.4 Querying RKD*.

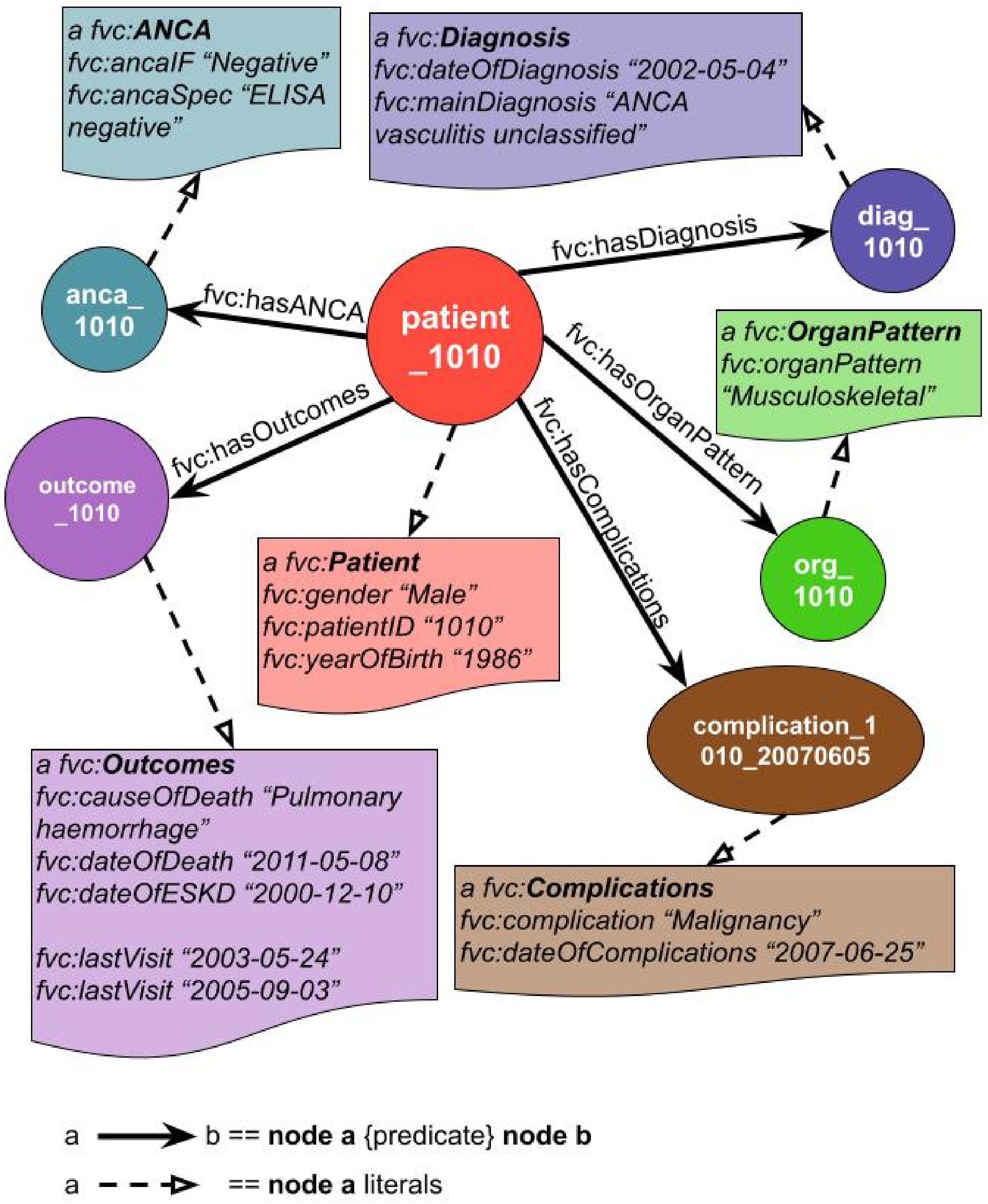

a fvc:ANCA
fvc:ancaIF "Negative"
fvc:ancaSpec "ELISA negative"
a fvc:Diagnosis
fvc:dateOfDiagnosis "2002-05-04"
fvc:mainDiagnosis "ANCA vasculitis unclassified"
diag_ 1010
fvc:hasDiagnosis
anca_ 1010
fvc:hasANCA
patient _1010
a fvc:OrganPattern
fvc:organPattern "Musculoskeletal"
fvc:hasOrganPattern
fvc:hasOutcomes
outcome _1010
fvc:hasComplications
org_ 1010
a fvc:Patient
fvc:gender "Male"
fvc:patientID "1010"
fvc:yearOfBirth "1986"
complication_1 010_20070605
a fvc:Outcomes
fvc:causeOfDeath "Pulmonary haemorrhage"
fvc:dateOfDeath "2011-05-08"
fvc:dateOfESKD "2000-12-10"
fvc:lastVisit "2003-05-24"
fvc:lastVisit "2005-09-03"
a fvc:Complications
fvc:complication "Malignancy"
fvc:dateOfComplications "2007-06-25"
a b == node a {predicate} node b
a == node a literals

Figure 3.1: ***Visual representation of synthetic Patient 1010 from Hoth Registry***

3.2 Planning and Research

3.2.1 Ethical Considerations

The protection of medical data was paramount during this project. During development, synthetic data were used both for programming and for demonstrating the application to the FAIRVASC team. Synthetic data were also used during the project demonstration.

In order to comply with patient confidentiality guidelines, only queries that aggregate patient data could be posed to the databases. This means that no user should be able to access data pertaining to a single patient that makes them identifiable.

Before deployment, user authorisation and authentication would be needed to protect the patients' data from being viewed by anyone who is not an approved member of the research team or an approved medical staff member.

3.2.2 Resource Description Framework

Resource Description Framework (RDF) is the database used by the FAIRVASC team.

RDF is a non-relational graph database that stores data in "triples" of a subject, a predicate, and an object [18]. The predicate describes the directional relationship of the subject to the object. The subject of a triple must be a node, or "URI" (Uniform Resource Identifier), but the object can be a node or a "literal", such as a string, an integer or a date.

An example of an RDF graph that has been visualised can be seen in *Figure 3.1*. The central node representing the patient is connected to five other nodes and the predicate for each triple is shown on the directional arrow. One of these triples is denoted
$patient\_1010\ fvc : hasANCA\ anca\_1010$

Each node has a "type", which is denoted $patient\_1010\ a\ fvc : Patient$

Each node has a number of "literal" properties. The literals and the predicates that relate each to the node are listed in a box that matches the colour of the node. One of these triples is $patient\_1010\ fvc : yearOfBirth\ "1986"^{integer}$

3.2.3 SPARQL Protocol and RDF Query Language

SPARQL Protocol and RDF Query Langauge (SPARQL) is the standardised query language used to query or modify RDF databases [19]. SPARQL allows the user to efficiently pattern-match within the database, using variables to represent the subject, object, or predicate of the triples.

Snippet 3.2 shows a simple SPARQL query. A prefix is defined to allow the use of the shorthand to represent a URL. The URL denoted by *fvc* directs to where the FAIRVASC standardised ontology is found. Variables in SPARQL are denoted by ?*foo*. These variables are used to pattern-match in the database. The *SELECT* keyword determines which variables are returned. The *COUNT* keyword is used to aggregate the data, which is important for anonymising patient data.

The triples in the *WHERE* clause use variables to represent subjects or objects, which could be nodes or literals. Predicates and types (such as *causeOfDeath* and *Patient*) are defined

```
PREFIX fvc: <http://ontologies.adaptcentre.ie/fairvasc#>
SELECT ?cause (COUNT(?patient) as ?patient_count)
WHERE {
   ?patient a fvc:Patient ;
          fvc:hasOutcomes  ?outc .
   ?outc fvc:dateOfDeath ?date.
   OPTIONAL {?outc  fvc:causeOfDeath ?cause .}
} GROUP BY ?cause ORDER BY ?patient_count
```

Snippet 3.2: ***A SPARQL query searching for the number of patients who have died, grouped by their cause of death, if listed***

by the FAIRVASC ontology, so the prefix *fvc* is used. The *OPTIONAL* clause specifies a cause of death where one exists, but also allows patients without a cause of death listed to be returned. *GROUP BY* allows for aggregation of results and *ORDER BY* enables the sorting of results.

3.2.4 Querying RKD

A synthesised copy of RKD, the Rare Kidney Disease registry that contains the medical information of AAV patients in Ireland, was used to explore the data. RKD has already been standardised by FAIRVASC. This allowed for familiarisation with the SPARQL query language and with the format of the RKD graph, and therefore the FAIRVASC RDF graphs as a whole.

A single patient was examined to view the format of the patients within the database. A simplified visualisation of a patient from the synthetic registry "Hoth" is included in *Figure 3.1*. This visual representation of a patient shows the complexity of querying the FAIRVASC registries, as relevant information about a patient is spread across different nodes. This graph is further explained in *subsection 3.2.2 Resource Description Framework*.

Multiple aggregate queries were posed to the RKD database in order to explore the shape of the data in the database. One of the queries run can be seen in *Snippet 3.2*, explained in *subsection 3.2.3 SPARQL Protocol and RDF Query Language*, which counts the number of patients who have died by each listed cause of death, if the cause of death is listed, and the number of patients who have no cause of death listed. The results of posing this query to the synthetic RKD registry can be seen in *Table 3.1*. This table shows three important facts. Firstly, that many doctors do not enter a cause of death for their AAV patients. Secondly, that there is a wide array of causes of death of AAV patients. Finally, it highlights how a small number of patients in a registry would lead to insufficient data to research the disorder and would lead to patients in the registry being identifiable by their cause of death, which would breach patient confidentiality guidelines.

cause	**patient_count**
[none listed]	48
Pneumonia	10
Sepsis	4
AAV-myocardial infarction	1
ANCA-associated cardiac arrest	1
Acute heart failure	1
Chest infection	1
Lung cancer	1
Pulmonary haemorrhage	1
Stroke	1
Vasculitis cardiomyopathy	1
Withdrawal from dialysis	1

Table 3.1: ***Some of the causes of death of patients in the synthetic RKD registry***

3.2.5 Application's Basic Components

The web application developed during this project has three primary components.

The front-end gathers the query parameters from the user and displays the query results using data visualisation.

The back-end hosts the front-end. It receives user queries from the front-end and sends them to the third-party database. It parses and formats the results and sends them to the front-end.

The third-party component consists of the RDF database which is hosted remotely. The back-end must be able to send and receive queries from multiple RDF databases.

3.2.6 Back-End Research

The back-end was needed to host the front-end web pages and to act as a middleman between the front-end and the third-party RDF registries. Three frameworks were considered for the back-end of the application.

Express Server

Advantages

+ Express is a JavaScript framework, which means the back-end would be written in the same language as the front end.
+ Express offers simultaneous connections, which is beneficial when interacting with third-party resources.

+ Express allows the client to easily track visitors and their actions, enabling the client to find malicious actors.

Disadvantages

– Since Express is an asynchronous JavaScript framework, it uses callbacks to simulate multi-threading within Node.JS's single threaded framework [20]. Callbacks vastly reduce code readability.

– Express does not offer user authentication or authorisation, and there is no simple way to integrate a database into the server.

– Express servers consist of a single Javascript file, which means they do not offer the complexity that is supported by other frameworks.

Django

Advantages

+ Django is a Python framework which means that Python's many third-party libraries can be used, such as SPARQLWrapper for posing queries and receiving results from an RDF database.

+ Django has a rigid project structure. This is an advantage in projects where another developer or team of developers will be expanding upon the application [21]

+ Django offers built-in database functionality which allows for the creation and storage of user accounts.

+ Django supports a REST-ful API for user authorisation and authentication which, while out of scope for this project, will be necessary for this application in future.

Disadvantages

– Django is a heavier-weight framework than both Express and Flask, which means that it requires more memory to run.

Flask

Advantages

+ Flask is also a Python framework, so it can be used with SPARQLWrapper and other third-party Python libraries.

+ Flask's less structured project framework means that fewer files are needed for the application to function than for a Django project.

+ Flask's applications are fast and lightweight.

Disadvantages

– Flask lacks built-in user authentication and local database functionality.

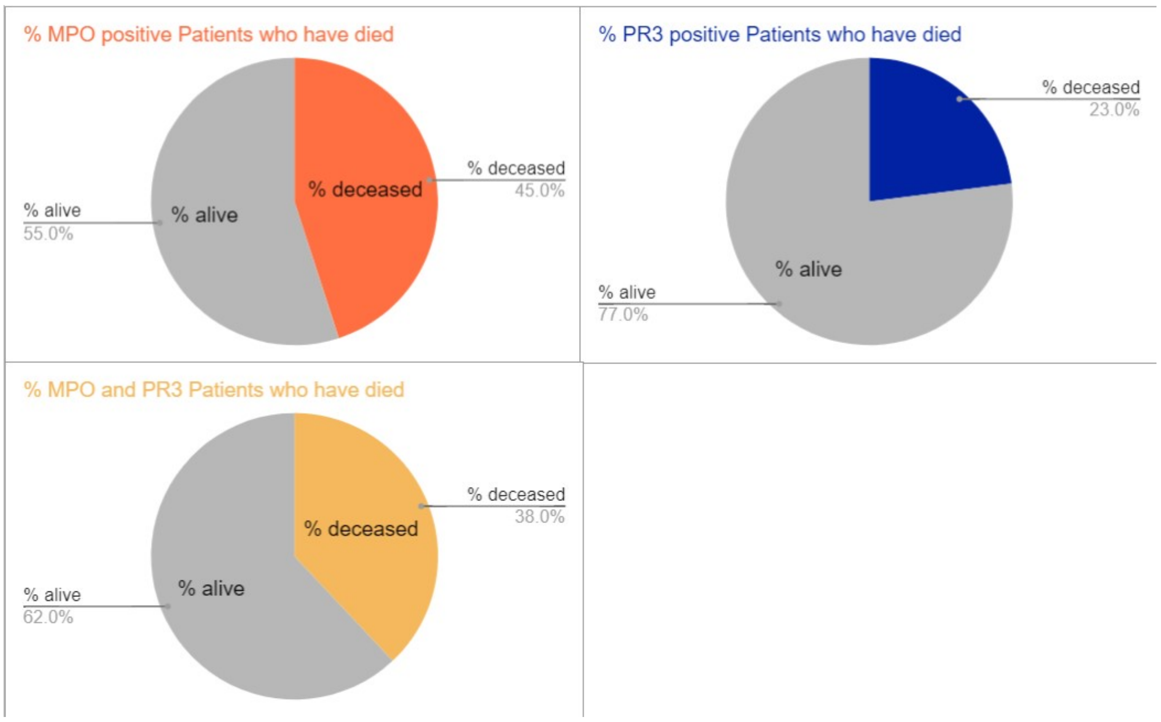


Figure 3.2: ***Example series of pie charts to demonstrate a visualisation technique that could be used in the web application***

3.2.7 Data Visualisation Research

The nature of the data was discussed with the FAIRVASC team. There are four data elements to be considered —patient's outcome, patient's sex, main diagnosis, and ANCA serotypes. Data could be pulled from multiple registries.

Each of these elements is either binary, such as sex, or has a limited selection, such as the four diagnosis options.

The primary purpose of the visualisation element of the system is to allow users to easily compare and contrast different subsets of the data based on any combination of selections.

Three main approaches to visualisation were considered.

Since the results are expressed as percentages of a larger subset, one option considered was a series of parallel pie charts, such as in *Figure 3.2*. Pie charts are effective at communicating a value as a percentage of a whole. However, since the system allows users to select any combination of data elements, the number of pie charts would be variable and could be too numerous to allow for easy comparisons between figures.

Since the objective was to compare and contrast, another option was a scatter plot. However, since the data elements are discrete, a scatter plot proved an ineffective way of communicating the results of the queries outlined in *subsection 3.1.4 FAIRVASC SPARQL Queries*. *Figure 3.3* shows a sample plot where two numeric series are displayed for a single discrete data element.

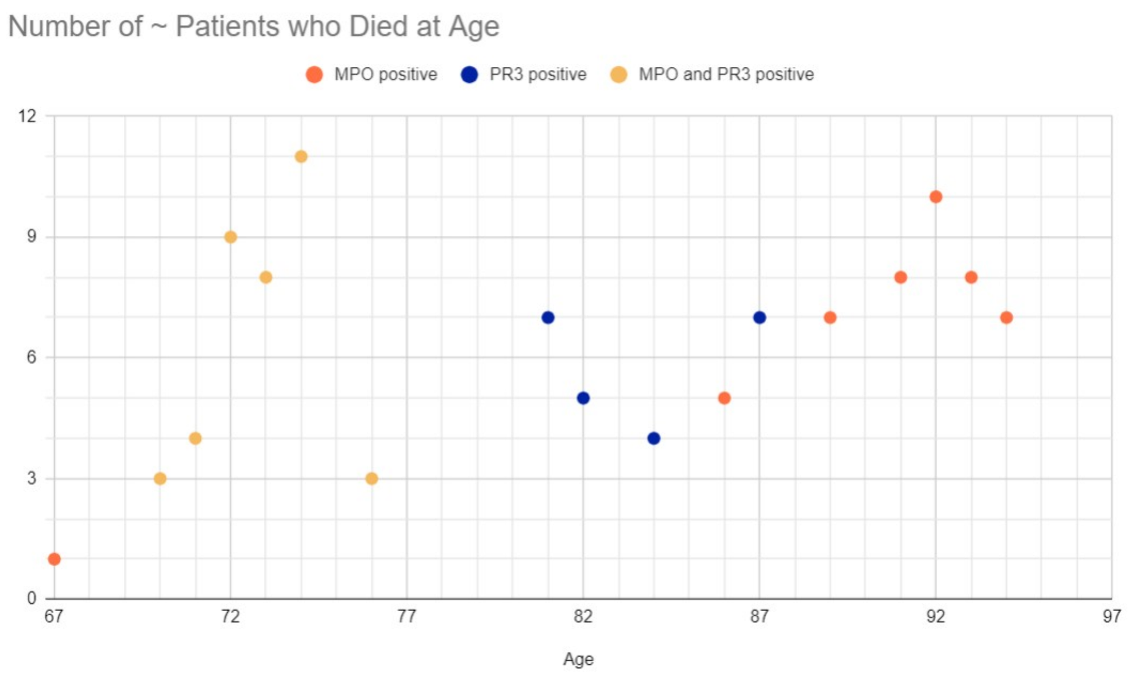


Figure 3.3: ***Example scatter chart to demonstrate a visualisation technique that could be used in the web application***

The final option was using a bar chart. Bar charts allowed for variable a number of data points and provided clear comparison between bars, such as the sample plot in *Figure 3.4*.

Additional functionality could be added to a bar chart to enable more complex comparisons. The elements on the bar chart could be transposed to enable the user to compare the data points using a different visualisation. When three data elements were selected, two parallel bar charts could be used to split the data along a binary element, patient sex by default.

A consistent colour palette was chosen to avoid confusion when comparing and contrasting figures. The palette does not change between queries or between uses of the application, to improve clarity. The colours were chosen so that a distinct and identifiable colour corresponded to each parameter in each data element, as shown in *Figure 3.5*. Although only three registries were used during development and currently the FAIRVASC project has seven collaborating registries, ten colours were chosen to represent the registries. This would allow the project to grow to include three more registries before the colour system would

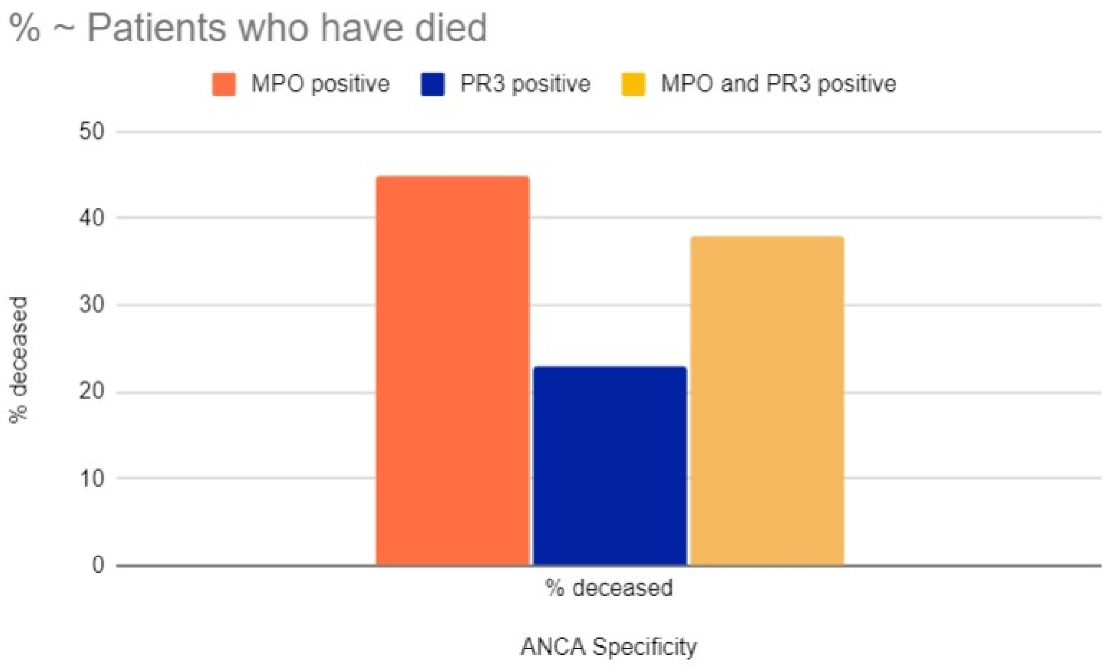
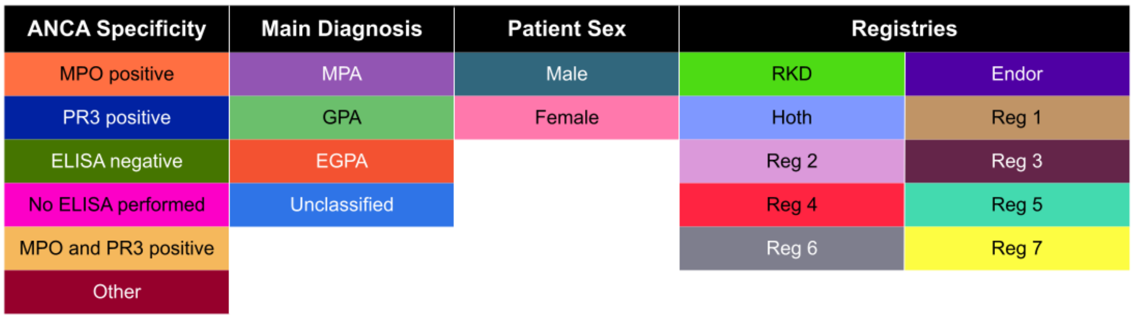


Figure 3.4: ***Example bar chart to demonstrate a visualisation technique that could be used in the web application***

need to be adapted.



ANCA Specificity	Main Diagnosis	Patient Sex	Registries	
MPO positive	MPA	Male	RKD	Endor
PR3 positive	GPA	Female	Hoth	Reg 1
ELISA negative	EGPA		Reg 2	Reg 3
No ELISA performed	Unclassified		Reg 4	Reg 5
MPO and PR3 positive			Reg 6	Reg 7
Other				

Figure 3.5: ***A representation of the colours chosen for all query parameters and ten possible registries***

3.2.8 User Input Research

The goal of the user input is to reduce the technological knowledge needed to pose queries to the RDF databases. Three options were considered for user input.

Text Input

Text input would consist of an empty text box, as seen in *Figure 3.6*

This would give the user a lot of freedom with the queries they could pose. However, the user would need to have a lot of knowledge of SPARQL and the framework used in the FAIRVASC registries. The risk of human error is high.

The freedom allowed to the user could result in bad actors violating patients' medical privacy by posing queries for information about a specific user.

Query Builder

A Query Builder would consist of multiple single- or multi-line text boxes where the user could fill in their parameters, as seen in *Figure 3.7*.

This gives the user less freedom than an empty text box, but still requires a lot of knowledge of RDF and SPARQL. There is still a high risk of human error.

Input Widgets

Input Widgets would consist of checkboxes, radio buttons and dropdown menus to allow the user to build a query, as seen in *Figure 3.8*.

This option requires the least knowledge of RDF and SPARQL and is compliant with medical privacy since only approved queries can be posed.

Unfortunately, this restriction in freedom for the user makes it more difficult to expand the application to allow for more queries to be posed in the future.

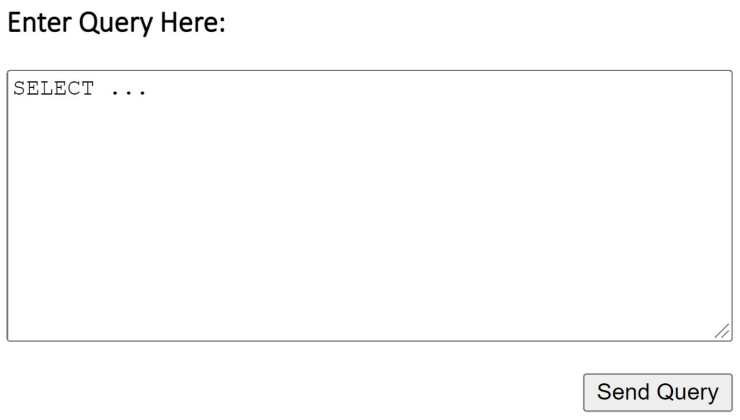


Figure 3.6: ***Example user input with a simple text box for query entry***

3.2.9 Front-End Research

The front-end of this application needed to be able to accommodate user input and the visualisation of the results. Three JavaScript libraries were considered for the front-end of the application.

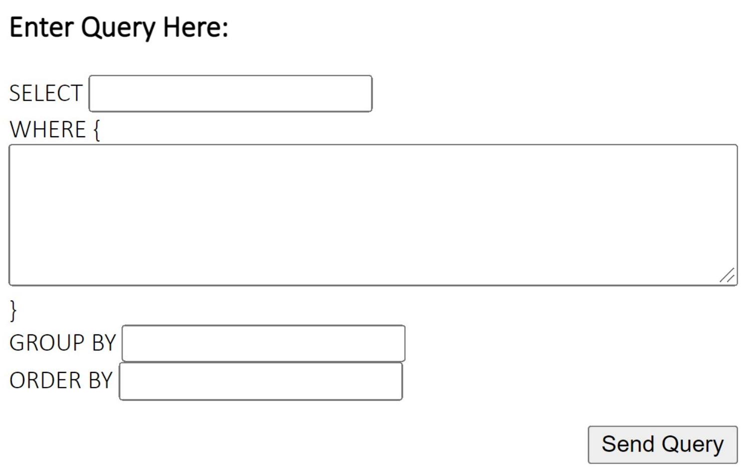


Figure 3.7: ***Example user input using multiple text entry fields to construct a query***

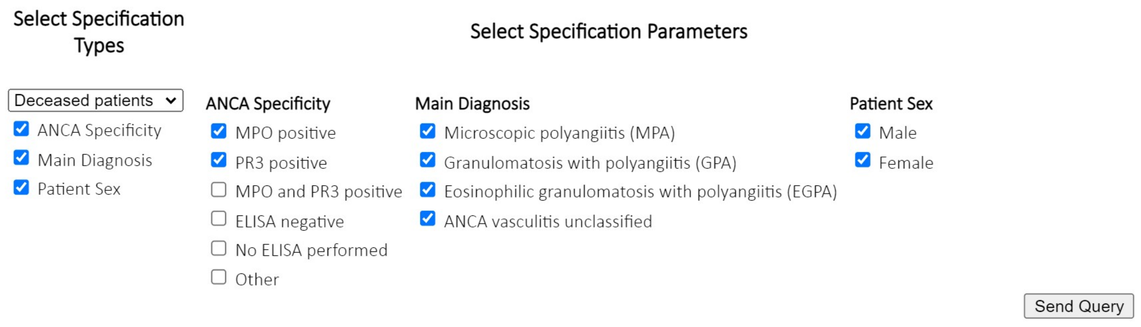

Select Specification Types
Select Specification Parameters
Deceased patients
ANCA Specificity
Main Diagnosis
Patient Sex
ANCA Specificity
MPO positive
PR3 positive
MPO and PR3 positive
ELISA negative
No ELISA performed
Other
Main Diagnosis
Microscopic polyangiitis (MPA)
Granulomatosis with polyangiitis (GPA)
Eosinophilic granulomatosis with polyangiitis (EGPA)
ANCA vasculitis unclassified
Patient Sex
Male
Female
Send Query

Figure 3.8: ***Example user input using check-boxes and a drop-down menu to construct a query***

React

Advantages

+ React is currently the most commonly used JavaScript front-end framework, so there is a lot of third-party support for React applications.

+ React applications are written in JSX, instead of JavaScript, which adds additional readability when working directly with HTML.

Disadvantages

– Components within React applications can only communicate using a third-party library, such as Flux, which adds additional complexity.

– React does not have a built-in development workflow. There are many similar third-party options to choose from, but a number of them would have to be considered to find the best fit for this application.

Angular 2+

Advantages

+ Angular 2+ has a more rigid workflow than React and depends on fewer third-party libraries to function.
+ Angular 2+ applications are written in TypeScript, which "ensures type safety" [22]. This also increases code readability.

Disadvantages

– Each Angular component consists of three distinct files; a logic file, a layout file and a stylesheet. This reduces readability of the code.

– Angular's rigid framework can be restrictive.

Vue 3

Vue is a more recent front-end library, which borrows elements from both Angular and React

Advantages

+ Vue's reusable components are written in a single structured file that contains the logic, layout and style of the component.
+ Vue has a number of logic tags that can be used directly within the HTML, such as conditions (*v-if*)
+ Communication methods between individual Vue components are built into the Vue library without relying on a third-party application.
+ Vue uses JavaScript, instead of a "superset of JavaScript like TypeScript ... or JSX" [22], which reduces the learning curve when working with Vue.

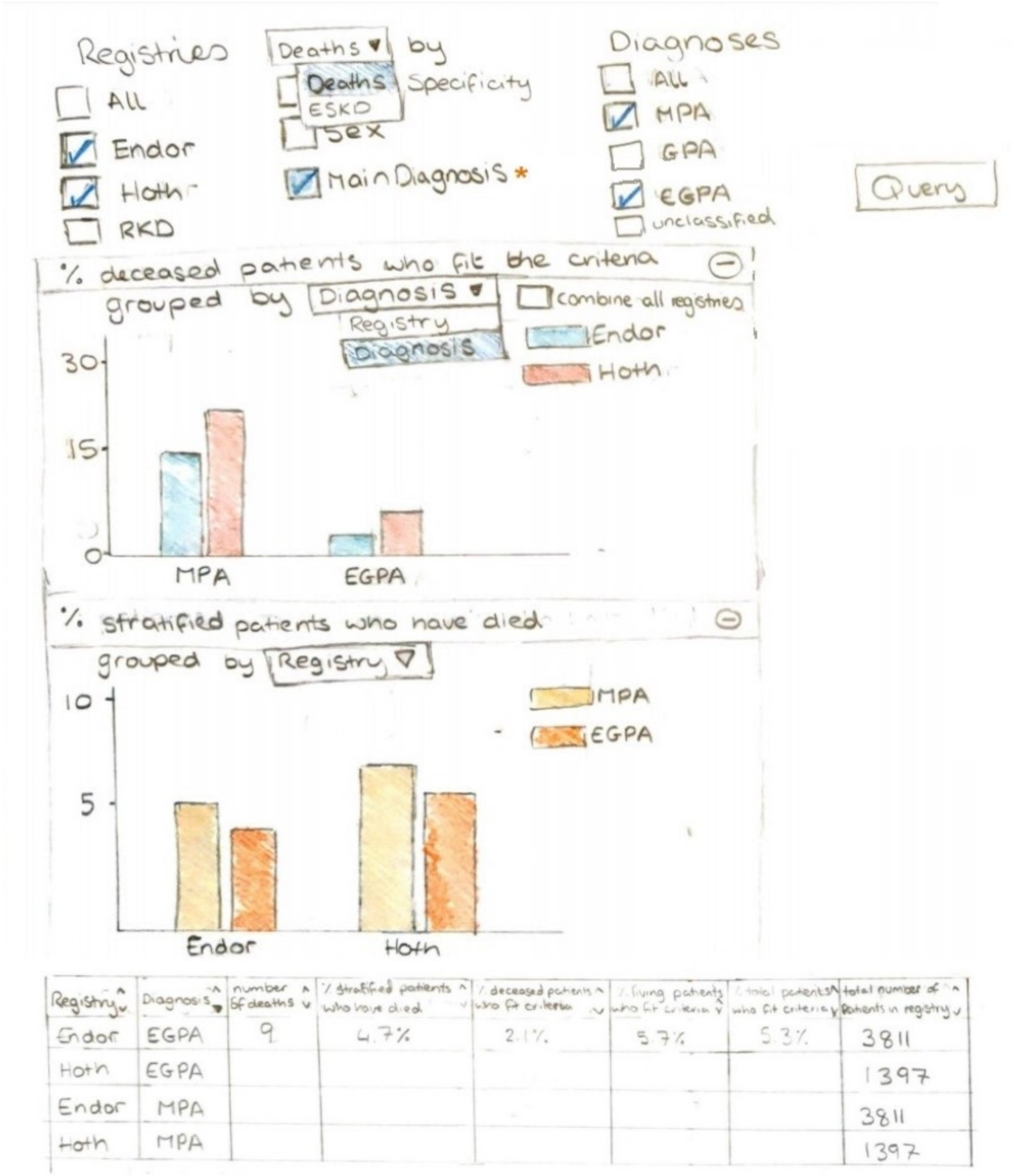
Disadvantages

– Vue is the least used of the three frameworks, so it is less supported by third-party libraries.

3.3 Paper Prototype

3.3.1 Purpose of the Paper Prototype

A paper prototype was created to confirm a shared understanding of the functional and non-functional requirements between the client and the developer. The paper prototype explores expected user interaction and displays chosen data visualisation techniques.



Registry	Diagnosis	number of deaths	% stratified patients who have died	% deceased patients who fit criteria	% living patients who fit criteria	% total patients who fit criteria	total number of patients in registry
Endor	EGPA	9	4.7%	2.1%	5.7%	5.3%	3811
Hoth	EGPA						1397
Endor	MPA						3811
Hoth	MPA						1397

Figure 3.9: ***Paper prototype of the FAIRVASC Web Application user interface designed to show to the FAIRVASC QIT and FIT teams***

3.3.2 Discussion of the Paper Prototype

The paper prototype, which can be seen in *Figure 3.9*, has three main components which are elaborated on below.

User Input

From the discussion in *subsection 3.2.8 User Input Research*, it was decided that the users' queries would be gathered using input widgets.

In the first column, the user can choose from any number of registries. In the paper prototype, the names of the registries containing synthetic data were used.

The second column provides the user with a dropdown menu to choose the outcome: either death or ESKD. Then, they can choose any number of the following three data elements: ANCA Specificity (or serotype), Main Diagnosis, and Patient Sex.

In the prototype, only the Main Diagnosis has been selected. This means that the third column displays the diagnosis parameters for the user to further filter their results.

When the user has completed their query selection, they can click the Query button, which sends their query to the back-end, and then wait for the results.

Data Visualisation: Charts

After the research discussed in *subsection 3.2.7 Data Visualisation Research*, the visualisation approach that was chosen was bar charts.

Two bar charts are visible in the prototype in *Figure 3.9*. The first shows the percentage of all deceased patients in the selected registries who have a diagnosis of MPA or EGPA. This shows the prevalence of these diagnoses and this outcome within the registries. The second shows the percentage of all MPA or EGPA patients in the registries who are deceased. This shows the survival rate of MPA and EGPA patients.

The bar charts can be transposed, allowing the user to group the data by diagnosis or by registry. Grouping by diagnosis shows more clearly the discrepancies between outcomes of patients in different registries. Grouping by registry highlights the discrepancies between the outcomes of patients with different diagnoses.

Data Visualisation: Table

All the data returned from the registry are shown in a table that provides the user with a more complete view of the information. The table can be sorted by any of the columns to highlight different aspects of the data. A table is a less intuitive way of visualising data for comparison, but it provides a more in-depth view for deeper analysis.

3.3.3 Feedback

The paper prototype was shown to the members of the FAIRVASC team to determine if this was an appropriate user interface for the final application. The feedback was positive for all of the elements of the interface.

One of the members of the team suggested that in the case where all three data elements have been chosen, the data could be displayed using two adjacent bar charts, split by Patient Sex. This feedback was considered and enhanced upon. It was decided that the user should be able to choose along which of the elements the split occurred, provided exactly two parameters were chosen for that element. Since a maximum of two parameters can be chosen for Patient Sex, there will always be at least one option by which the data can be split.

3.4 Chapter Summary

This chapter explored into the design approach taken during this project. The research into technologies and data visualisation techniques, as well as the FAIRVASC team's feedback about the paper prototype, was used during the implementation of the web application, as seen in the following chapter.

4 Implementation

This chapter delves further into the technologies chosen to develop the application, outlining both the front- and back-ends. Next, the implementation of the front-end and of the back-end server is illustrated. Finally, the complications that occurred during implementation are discussed.

4.1 Technology Used

As discussed in *subsection 3.2.5 Application's Basic Components*, there are three primary components of the web application: the front-end, the back-end, and the third-party RDF databases.

4.1.1 Front-End Technologies

JavaScript is an open-source programming language that is the most commonly used language for complex web applications. JavaScript is object-oriented and is allows users to interact with web pages [23].

Axios is a "promise-based HTTP client" [24] which allows browser clients to send HTTP Requests. Using an XMLHttpRequest, an application can request and receive information from a URL without refreshing the entire web page [25].

Vue is an open-source JavaScript framework that provides additional complexity for web applications, using reusable components. Vue provides comprehensive and intuitive handling for JavaScript events [26] and supports conditional rendering.

HTML, HyperText Markup Language, is the standard language used to create web pages. A HTML page consists of elements with tags and labels that communicate to the browser how the element should be displayed [27]. **CSS**, Cascading Style Sheets, give the browser further information about the style or layout of HTML elements [28].

Chart.JS is an open-source JavaScript library that can be used to create many different charts for data visualisation [29]. Chart.JS is easy to implement and creates clean interactive charts.

Tabulator is a JavaScript library that can create tables from JSON data [30]. Tabulator tables are interactive and customisable using CSS.

4.1.2 Back-End Technologies

Python is a fast high-level open-source language. Python has a comprehensive standard library and thousands of third-party libraries have been registered to further expand its functionality [31].

SPARQLWrapper is a third-party Python library that sends a SPARQL query to a specified RDF endpoint and returns the result in the requested format [32].

Django is a Python framework used to create web applications [33]. Django projects have a rigid structure which allows for ease of readability and comprehension, which is important for a project such as this, since it will be further developed by a team. Django enables the developer to easily create a database of user accounts and an authentication system. This is vital for an application that uses and displays sensitive medical data.

4.1.3 Database Technologies

Resource Description Framework (RDF) is a non-relational graph database that stores data in "triples" of a subject, a predicate, and an object [18]. **SPARQL Protocol and RDF Query Langauge (SPARQL)** is the standardised query language used to query or modify RDF databases [19]. RDF and SPARQL are explained in more detail in *subsection 3.2.2 Resource Description Framework* and *subsection 3.2.3 SPARQL Protocol and RDF Query Language*.

Apache is an open-source web server software [34]. **Apache Jena Fuseki** is an Apache server that can be used to host RDF databases [35]. It was used during the development of this application to host synthetic data locally to allow testing of the application.

4.2 Implementing the Front-End

As mentioned in *subsection 3.2.5 Application's Basic Components*, the front-end is a webpage where the user inputs their query parameters and where the query results are displayed.

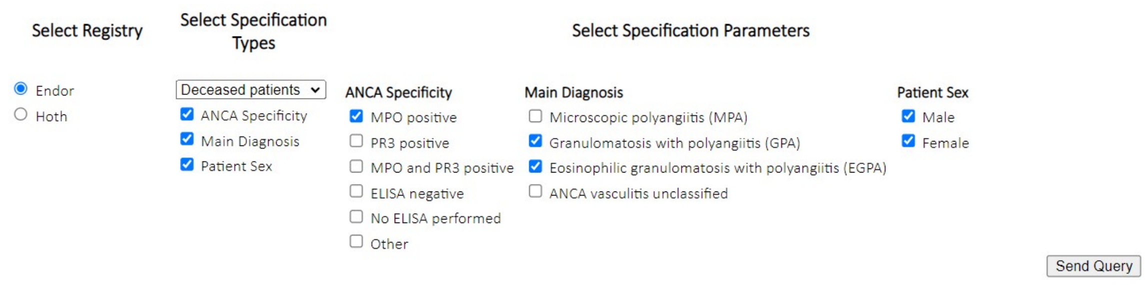
4.2.1 Front-End Overview

The front-end consists of a single web-page contained within a single HTML file. The Vue application's and components' logic, layout, and style are found here. The main Vue application processes the user input and interacts with the back-end. Two Vue components were written to display the results.

The first Vue component implements the Chart.JS bar chart code. It is reused twice within the main application. The second component implements the Tabulator results table and is used once.

The file is rendered by the Django framework and hosted on the Django server.

4.2.2 Selecting and Sending the Query


Select Registry
[✓ Endor]
Hoth

Select Specification Types
Deceased patients
[✓ ANCA Specificity]
[✓ Main Diagnosis]
[✓ Patient Sex]

Select Specification Parameters

ANCA Specificity
[✓ MPO positive]
PR3 positive
MPO and PR3 positive
ELISA negative
No ELISA performed
Other

Main Diagnosis
Microscopic polyangiitis (MPA)
[✓ Granulomatosis with polyangiitis (GPA)]
[✓ Eosinophilic granulomatosis with polyangiitis (EGPA)]
ANCA vasculitis unclassified

Patient Sex
[✓ Male]
[✓ Female]

Figure 4.1: ***Application's user input using radio buttons, check-boxes and a drop-down menu***

The user input was modelled after the the paper prototype seen in *Figure 3.9* and discussed in *subsection 3.3.2 Discussion of the Paper Prototype*. A series of radio buttons, check-boxes and drop-down widgets are displayed for the user to build their query, as seen in *Figure 4.1*.

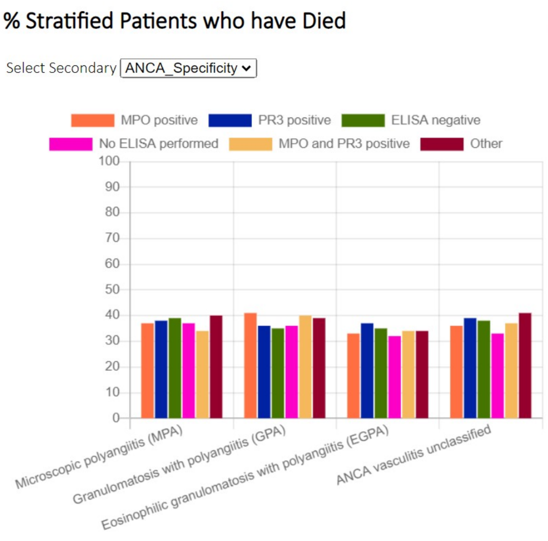
The user can choose a single registry from a radio button list. They can choose the patient outcome of death or ESKD cases. Next, they choose one or more of the 3 data elements, ANCA Specificity, Main Diagnosis and Patient Sex. When an element is selected, a list of check-boxes appears to the right, referencing the parameters of that element.

After constructing their query, the user hits the "query" button, which triggers the front-end to format the query as a HTML Request. This HTML Request is sent to the Django back-end.

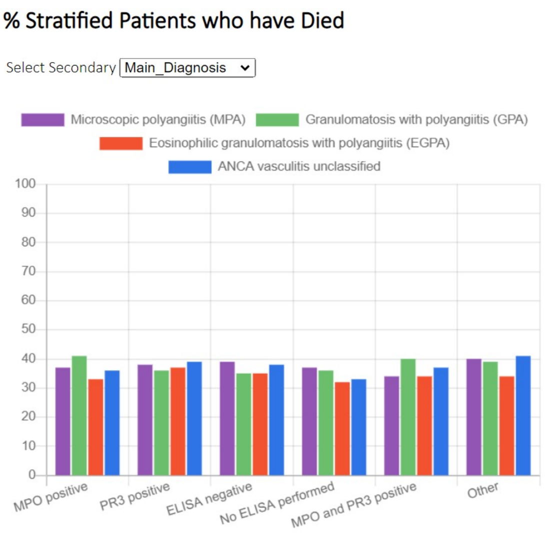
The results are received from the back-end in JSON format. The raw data is formatted to allow for easier processing by the data visualisation components.

4.2.3 Data Visualisation: Charts

Chart.JS was chosen to create the bar charts because of the simple, interactive bar charts that are displayed. The two statistics measured in the paper prototype in *Figure 3.9* were also chosen for the final application.



(a) ***Chart where data is grouped by Main Diagnosis***



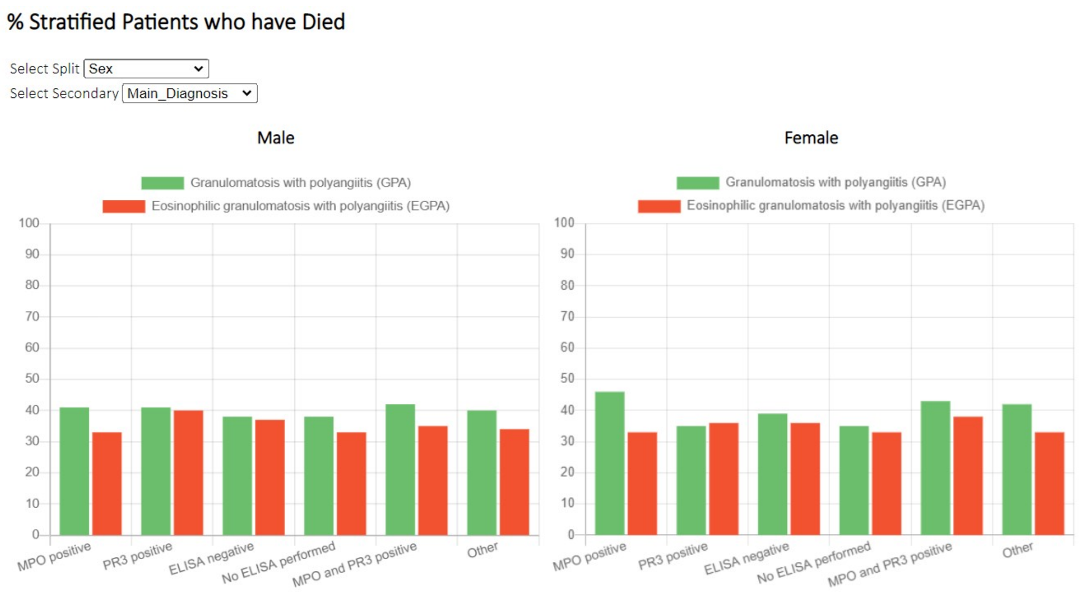
(b) ***Chart where data is grouped by ANCA Specificity***

Figure 4.2: ***Charts showing how the user can transpose the data in the application***

As mentioned in *subsection 3.2.7 Data Visualisation Research*, a unique colour was chosen for each parameter and for each registry. These colours were easily integrated into the Chart.JS bar-chart instantiation.

A drop-down menu above the chart allows the user to transpose the data, just like in the paper prototype. This required a lot of data manipulation for Chart.JS to display the new data, but the application completes the manipulation quickly. The timing of this application is discussed in *section 5.3 Application Response Times*.

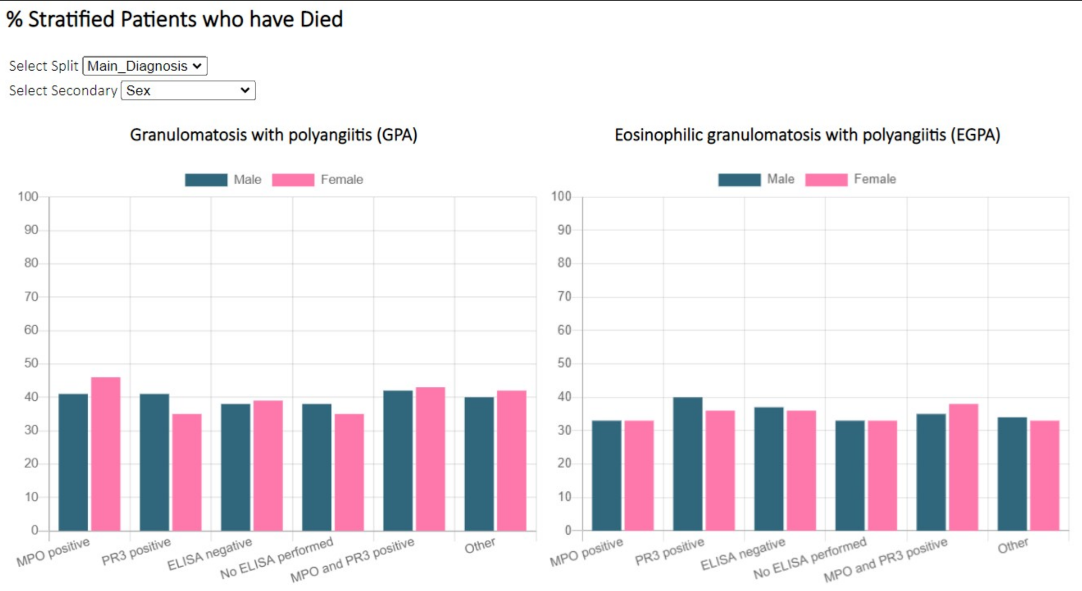
In the case where two charts are displayed, another drop-down menu allows users to switch the element along which the charts are split, as seen in *Figure 4.3*. Any element where exactly two parameters have been selected can be chosen for the split. The first drop-down menu transposes both side-by-side charts simultaneously for user clarity.


% Stratified Patients who have Died

Select Split Sex

Select Secondary Main_Diagnosis

(a) ***Chart where data is split by Patient Sex***


% Stratified Patients who have Died

Select Split Main_Diagnosis

Select Secondary Sex

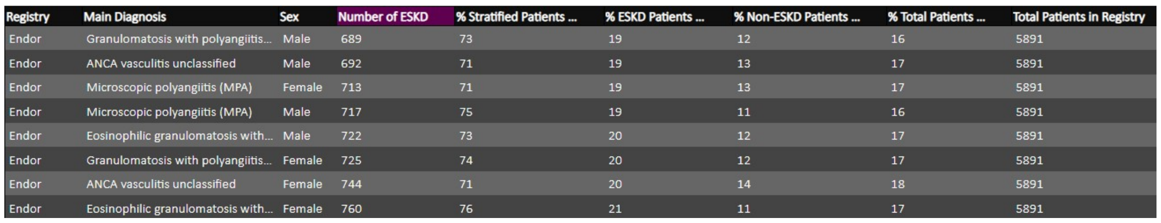
(b) ***Chart where data is grouped by Main Diagnosis***

Figure 4.3: ***Charts showing how the user can change the split of the data in the application***

4.2.4 Data Visualisation: Results Table

The results table shows more information than the bar charts, as seen in *Figure 4.4*. This example shows the patients from synthetic registry Endor stratified by ANCA specificity and patient sex.

Tabulator allows for a lot of manipulation of the table without extra coding by the developer. The results table in the application can be sorted by any of the columns. The columns can be re-organised to allow users to compare different figures more easily.



Registry	Main Diagnosis	Sex	Number of ESKD	% Stratified Patients ...	% ESKD Patients ...	% Non-ESKD Patients ...	% Total Patients ...	Total Patients in Registry
Endor	Granulomatosis with polyangiitis...	Male	689	73	19	12	16	5891
Endor	ANCA vasculitis unclassified	Male	692	71	19	13	17	5891
Endor	Microscopic polyangiitis (MPA)	Female	713	71	19	13	17	5891
Endor	Microscopic polyangiitis (MPA)	Male	717	75	19	11	16	5891
Endor	Eosinophilic granulomatosis with...	Male	722	73	20	12	17	5891
Endor	Granulomatosis with polyangiitis...	Female	725	74	20	12	17	5891
Endor	ANCA vasculitis unclassified	Female	744	71	20	14	18	5891
Endor	Eosinophilic granulomatosis with...	Female	760	76	21	11	17	5891

Figure 4.4: ***Screenshot of the table of results from a query posed to Endor stratifying ESKD patients by main diagnosis and patient sex and sorted by number of ESKD patients in each stratification***

4.3 Implementing the Server

4.3.1 Creating a Django project

Django projects consist of a main project folder with a number of sub-folders. Initially the project starts with one sub-folder which represents the project. Inside this folder is a file where the user can create URL paths to other parts of the application. Other sub-folders are created by the developers and contains apps.

One such app holds the HTML files that are hosted by the Django server. Another app was created to act as a middle-man between the front-end and the third-party RDF registries.

4.3.2 Communication between the back- and front-ends

Python methods that take a HTTPRequest as a parameter are used for communicating with the front-end of an application. These methods are given a unique URL within the Django project so they can be referenced by the front-end.

Axios is a JavaScript library that is used in an application's front-end to send HTTPRequests to a specified URL with parameters. Axios was used to send the query parameters from the front-end to the back end.

A method was created that takes in a HTTPRequest from the front-end, which contains the query specifications supplied by the user. The query is sent to the relevant registries and the response or responses are re-formatted as a Python dictionary, which can easily be transformed into a JSONResponse and returned to the front end.

4.3.3 SPARQL Query Builder

Multiple queries are posed to a registry to gather all of the data needed to correctly format the response to the request from the front-end.

Instead of creating a string for each possible query that the user could pose, a Python class

```
prefixes = ("PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
   ↪ PREFIX fvc: <http://ontologies.adaptcentre.ie/fairvasc#> ")
select_base = ("SELECT (COUNT(?p) as ?Patient_Count) ")
where_base_open = ("WHERE { ?p a fvc:Patient. ")
where_close_bracket = ("}")
group_by = (" group by ")
where_deceased = (" ?p fvc:hasOutcomes ?o. ?o fvc:dateOfDeath ?dd. ")
where_eskd = (" ?p fvc:hasOutcomes ?o. ?o fvc:dateOfESKD ?de. ")
strat_dict = {
    "sex" : {
        "where" : (" ?p fvc:gender ?Sex. "),
        "var" : (" ?Sex "),
    },
    "ancaSpec" : {
        "where" : ( "?p fvc:hasANCA ?anca. ?anca fvc:ancaSpec
           ↪ ?ANCA_Specificity. " ),
        "var" : (" ?ANCA_Specificity "),
    },
    "diagnosis" : {
        "where" : ( "?p fvc:hasDiagnosis ?diag. ?diag fvc:mainDiagnosis
           ↪ ?Main_Diagnosis. " ),
        "var" : (" ?Main_Diagnosis "),
    },
}
```

Snippet 4.1: ***Python code containing the building block strings used to create a SPARQL query***

was constructed that concatenated shorter building block strings to create the user's query using the parameters passed from the front-end. The building block strings are found in *Snippet 4.1*. Each possible line that could be in the query is listed. The data elements that the user can stratify by are in a dictionary. The class constructor is found in *Snippet 4.2*. The user's query specifications are taken into the constructor as parameters and the correct SPARQL string is created. The rest of the QueryBuilder class can be found in *Appendix A1 QueryBuilder.py*

This constructed query was sent to the relevant registry using the SPARQLWrapper library. The response from the registries was in the form of comma-separated values (CSV). The query builder class parses the CSV and formats the information as a Python dictionary.

The results from multiple queries to the same registry are merged so the information is complete. Results from different registries are concatenated in the dictionary.

```
class QueryBuilder():
    def __init__(self, total_counts=True, outcome_deceased=False,
        ↪ by_sex=False, by_ancaSpec=False, by_diagnosis=False):
        self.logger = logging.getLogger('console')
        self.total_counts = total_counts
        self.outcome_deceased = outcome_deceased
        self.by_sex = by_sex
        self.by_ancaSpec = by_ancaSpec
        self.by_diagnosis = by_diagnosis
        self.select_string = select_base
        self.where_string = where_base_open
        self.group_string = where_close_bracket
        self.result_dict = {}
        self._get_empty_results()
        if not total_counts:
            # if stratified, strat by deceased xor by eskd
            if outcome_deceased: self.where_string = self.where_string +
                ↪ where_deceased
            else : self.where_string = self.where_string + where_eskd
        if by_sex or by_ancaSpec or by_diagnosis:
            self.group_string = self.group_string + group_by
            if by_sex: self._add_strat("sex")
            if by_ancaSpec: self._add_strat("ancaSpec")
            if by_diagnosis: self._add_strat("diagnosis")
        self.query_string = prefixes + self.select_string +
            ↪ self.where_string + self.group_string

    def _add_strat(self, strat_by):
        strat_var = strat_dict[strat_by]["var"]
        self.select_string = self.select_string + strat_var
        self.where_string = self.where_string +
            ↪ strat_dict[strat_by]["where"]
        self.group_string = self.group_string + strat_var
```

Snippet 4.2: ***Python class constructor for the QueryBuilder class***

4.4 Implementation Issues

4.4.1 Issues with the Server

Initially, the back-end was written using Express. This was because of its ease of integration with Vue and because the simplicity of the server being contained within one file. However, during the implementation of increasingly complex queries, it became evident that the Express server was too simple for this project.

Further research into back-end framework options was done and Django was chosen for its increased complexity and for its built-in support for user authentication, as discussed in *subsection 3.2.6 Back-End Research*.

The changes to the back-end caused a setback during the project, where little progress was made on the front-end.

4.4.2 Issues with the Visualisation

Vue was chosen for the front-end because it combines positives from Angular and React. However, since it is less commonly used than both of these frameworks, many JavaScript visualisation libraries do not integrate with Vue 3. A lot of research had to be done into possible libraries that could be used within a Vue application. Chart.JS and Tabulator were finally selected, as they could both be implemented within Vue.

4.5 Chapter Summary

This chapter outlined the technology used in the FAIRVASC Query Application, the implementation of the elements of the application, and issues that arose during development. The evaluation of the final application is discussed in the following chapter.

5 Evaluation

This chapter outlines the planned user evaluation and discusses the level of success at which the client's requirements were met. The future work the author proposes is needed before the deployment of this application is discussed.

5.1 User Evaluation

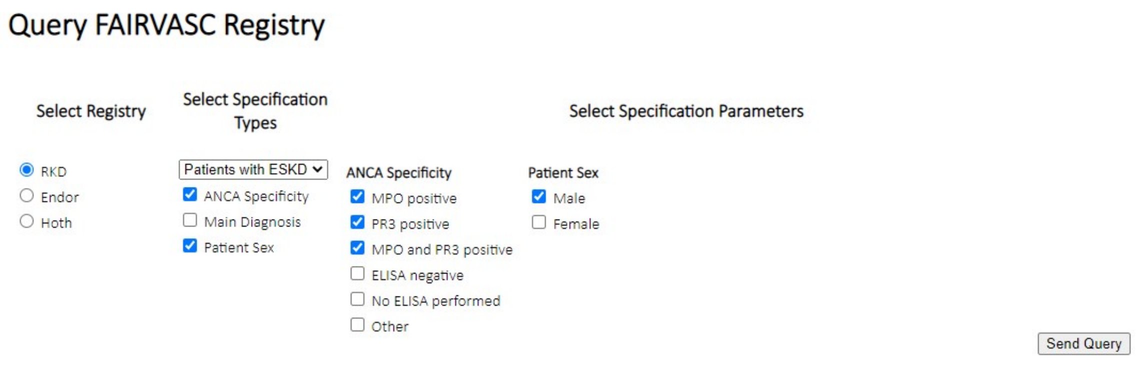
A series of user evaluation interviews were planned with members of the FAIRVASC team. Unfortunately, those interviews could not be conducted before the end of the project due to time constraints and scheduling issues.

A Post-Study System Usability Questionnaire (PSSUQ) was planned. PSSUQ standard questions are found in *Appendix A2 PSSUQ Questionnaire*, which are designed to assess a system's usability and usefulness.

This questionnaire would be posed to users after completing an user evaluation interview with the developer. The evaluation interview consists of the interviewer giving the user tasks to complete using the system.

Two tasks with several sub-tasks were designed for the user evaluation interviews.

1. Male ESKD patients from RKD
    (a) Pose a query to the RKD registry to find ESKD patients that are male and MPO-positive, PR3-positive, or MPO- and PR3-positive. This query can be seen in *Figure 5.1*.
    (b) Compare the percentage of patients who are MPO-positive with ESKD with those that are MPO- and PR3-positive with ESKD. *Figure 5.2* shows the bar chart that conveys this comparison.
    (c) Determine how many patients are in the RKD registry. This information is found in the results table, a screenshot of which is found in *Figure 5.3*.



Query FAIRVASC Registry

Select Registry	Select Specification Types	Select Specification Parameters	
● RKD	Patients with ESKD	ANCA Specificity	Patient Sex
○ Endor	✓ ANCA Specificity	✓ MPO positive	✓ Male
○ Hoth	☐ Main Diagnosis	✓ PR3 positive	☐ Female
	✓ Patient Sex	✓ MPO and PR3 positive	
		☐ ELISA negative	
		☐ No ELISA performed	
		☐ Other	

Send Query

*Figure 5.1: **User Evaluation Task 1(a): Query to RKD for male ESKD patients who are MPO-positive, PR3-positive, or MPO- and PR3-positive***

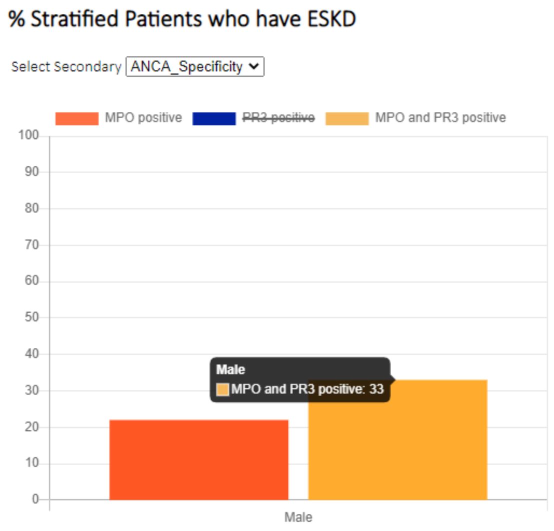
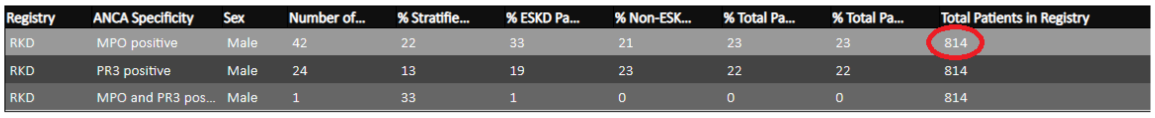


*Figure 5.2: **User Evaluation Task 1(b): A bar chart displaying the comparison between the percentage of male patients who are either MPO-positive or MPO- and PR3-positive who have ESKD***



Registry	ANCA Specificity	Sex	Number of...	% Stratifie...	% ESKD Pa...	% Non-ESK...	% Total Pa...	% Total Pa...	Total Patients in Registry
RKD	MPO positive	Male	42	22	33	21	23	23	814
RKD	PR3 positive	Male	24	13	19	23	22	22	814
RKD	MPO and PR3 pos...	Male	1	33	1	0	0	0	814

*Figure 5.3: **User Evaluation Task 1(c): Results Table showing that the RKD registry has 814 patients in it***

2. Deceased Patients from Endor

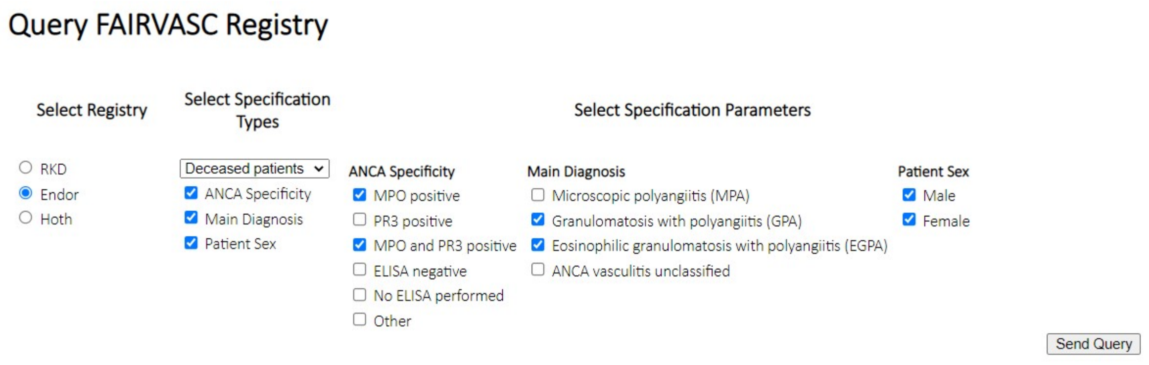
   (a) Pose a query to the Endor registry to find deceased patients that were male or female, MPO-positive or MPO- and PR3-positive, and had a GPA or EGPA diagnosis. This query is seen in *Figure 5.4*.

   (b) For deceased patients who had EGPA and were MPO- and PR3-positive, compare the percentage of those who are male and female, using the drop-down menus to change the orientation of the charts. The chart where this information is found can be seen in *Figure 5.5*.

(c) Find the set of specifications that has the lowest total number of patient deaths. *Figure 5.6* shows this specification.

During the interview, the user should perform these tasks to the best of their ability, and then complete the questionnaire to formally submit their feedback of the application.

This feedback can be used to make improvements to the application. After implementing any necessary changes, further user evaluation interviews could be performed with a different group of potential users.



Query FAIRVASC Registry

Select Registry
- ○ RKD
- ● Endor
- ○ Hoth

Select Specification Types
- Deceased patients
- ☑ ANCA Specificity
- ☑ Main Diagnosis
- ☑ Patient Sex

Select Specification Parameters

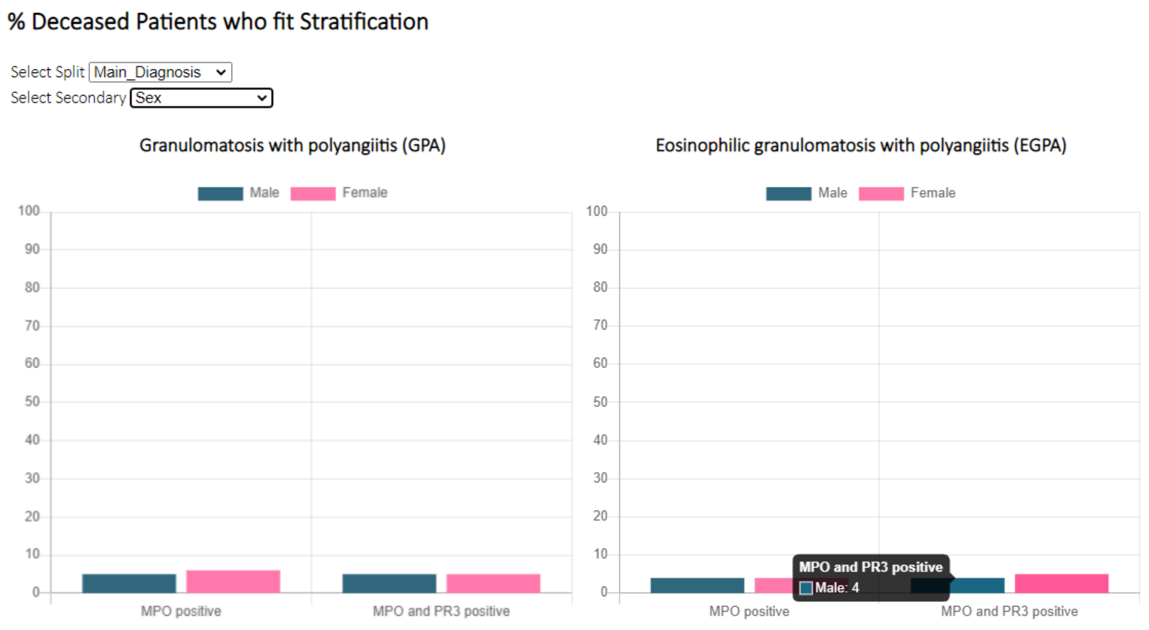
ANCA Specificity
- ☑ MPO positive
- ☐ PR3 positive
- ☑ MPO and PR3 positive
- ☐ ELISA negative
- ☐ No ELISA performed
- ☐ Other

Main Diagnosis
- ☐ Microscopic polyangiitis (MPA)
- ☑ Granulomatosis with polyangiitis (GPA)
- ☑ Eosinophilic granulomatosis with polyangiitis (EGPA)
- ☐ ANCA vasculitis unclassified

Patient Sex
- ☑ Male
- ☑ Female

Send Query

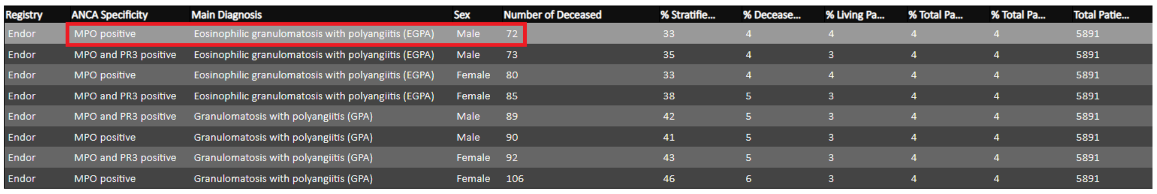
Figure 5.4: ***User Evaluation Task 2(a): Query to Endor for deceased male or female patients who have GPA or EGPA and are MPO-positive or MPO- and PR3-positive***



% Deceased Patients who fit Stratification

Select Split: Main_Diagnosis
Select Secondary: Sex

Figure 5.5: ***User Evaluation Task 2(b): A pair of bar charts split by patient diagnosis. The rightmost grouping of bars shows deceased male and female patients with EGPA who were MPO- and PR3-positive***



Registry	ANCA Specificity	Main Diagnosis	Sex	Number of Deceased	% Stratifie...	% Decease...	% Living Pa...	% Total Pa...	% Total Pa...	Total Patie...
Endor	MPO positive	Eosinophilic granulomatosis with polyangiitis (EGPA)	Male	72	33	4	4	4	4	5891
Endor	MPO and PR3 positive	Eosinophilic granulomatosis with polyangiitis (EGPA)	Male	73	35	4	3	4	4	5891
Endor	MPO positive	Eosinophilic granulomatosis with polyangiitis (EGPA)	Female	80	33	4	4	4	4	5891
Endor	MPO and PR3 positive	Eosinophilic granulomatosis with polyangiitis (EGPA)	Female	85	38	5	3	4	4	5891
Endor	MPO and PR3 positive	Granulomatosis with polyangiitis (GPA)	Male	89	42	5	3	4	4	5891
Endor	MPO positive	Granulomatosis with polyangiitis (GPA)	Male	90	41	5	3	4	4	5891
Endor	MPO and PR3 positive	Granulomatosis with polyangiitis (GPA)	Female	92	43	5	3	4	4	5891
Endor	MPO positive	Granulomatosis with polyangiitis (GPA)	Female	106	46	6	3	4	4	5891

Figure 5.6: ***User Evaluation Task 2(c): Results Table sorted by the number of deceased patients in ascending order, showing the set of specifications with the lowest number of deaths***

5.2 Meeting the Clients' Requirements

Six requirements were determined by the client and the developer, as outlined in *section 3.1 Client Requirements*

R1 required that users could review and compare patient data. This requirement was met by displaying the data in bar charts and a table. The bar charts can be transposed and customised by the user, as seen in *Figure 5.2*, and follow a consistent colour scheme. The rows of the table can be sorted by any of the columns and the columns can be reordered for ease of comparison within rows.

R2 required that the application pose SPARQL queries to RDF databases and parse the results. This requirement was met by developing the QueryBuilder class, discussed in *subsection 4.3.3 SPARQL Query Builder*. This python class uses building block strings to create a complex SPARQL query and poses that query the relevant RDF database. The class contains a method to parse the CSV response and the results are returned to the front-end.

R3 required that the users of the application could construct SPARQL queries. This requirement was met using the input widgets on the user interface, as shown in *Figure 5.4*. The user can select the registry they wish to query and the query parameters for the data elements they select.

R4 required that users could pose queries without needing prior knowledge of databases or query languages. This requirement was met using the input widgets previously outlined. The method of gathering user information using check-boxes and a drop-down menu allows users to have no knowledge of how the queries are constructed and posed or how the databases are structured.

R5 required that users should not have to wait a long time for results. The success of meeting this requirement cannot be evaluated, since the databases used during development were hosted locally and therefore an accurate review of the time taken to query a remote database could not be performed. However, the time needed to display the visualisations after receiving the results from the back-end were recorded and are discussed in *section 5.3*

Query Elements	**Number of Parameters**	**Run 1 (ms)**	**Run 2 (ms)**	**Run 3 (ms)**	**Run 4 (ms)**	**Average Time (ms)**
ANCA Spec	1	55	47	50	51	51
ANCA Spec	all (6 * 1)	184	134	141	136	149
ANCA Spec & Main Diagnosis	all (6 * 4)	585	544	565	542	559
ANCA Spec & Sex	all (6 * 2)	266	295	255	265	270
ANCA Spec, Main Diagnosis& Sex	all (6 * 4 * 2)	1219	1111	1216	1163	1177

Table 5.1: ***Table showing application response times for displaying the query results***

Query Elements	**Number of Parameters**	**Run 1 (ms)**	**Run 2 (ms)**	**Run 3 (ms)**	**Run 4 (ms)**	**Average Time (ms)**
ANCA Spec	all (6 * 1)	25	24	22	25	24
ANCA Spec & Main Diagnosis	all (6 * 4)	30	37	27	27	30

Table 5.2: ***Table showing application response times for transposing a single chart***

Application Response Times.

R6 required that results were communicated visually. This requirement was met by implementing bar charts. As outlined above, bar charts that can be transposed and customised by the user are displayed. This method of visualisation was chosen for its ease of interpretation of complex data.

5.3 Application Response Times

As mentioned in *section 5.2 Meeting the Clients' Requirements*, users should not have to wait a long time for query results. The time to perform actions on the front end was recorded.

The first aspect of the application that was tested was the time taken to display the initial visualisations after the results were returned. The results are examined in *Table 5.1*. The response time was recorded for multiple different queries to determine the time taken to display simple or more complex results. The time ranges from approximately 50 milliseconds to approximately 1.2 seconds. This response is fast enough to meet the requirement **R5** set by the client.

Since the charts in the application can be changed by the user, the times taken for the application to perform these actions were also recorded. First, two queries were posed that would return a single chart and the time taken to "transpose" those charts was recorded. Both of these charts were transposed by the application in under 40 milliseconds, as shown in *Table 5.2*. Next, two queries were posed that would return a set of double charts. Both of these sets were transposed and the timing was recorded in *Table 5.3*, showing that the application does not take longer to complete transposing two charts simultaneously than transposing a single chart. Finally, a query was posed that allowed the user to change the split of the charts and the response times were recorded in *Table 5.4*, showing that performing this action takes an average of 37 milliseconds.

These response times were considered quick enough to satisfy requirement **R5**.

Query Parameters	Number of Parameters	Run 1 (ms)	Run 2 (ms)	Run 3 (ms)	Run 4 (ms)	Average Time (ms)
ANCA Spec, Main Diagnosis & Sex	all (6 * 4 * 2)	38	33	30	29	33
ANCA Spec, Main Diagnosis & Sex	2 each (2 * 2 * 2)	32	33	27	31	31

Table 5.3: ***Table showing application response times for transposing a pair of charts***

Change Chart Split	Number of Parameters	Run 1 (ms)	Run 2 (ms)	Run 3 (ms)	Run 4 (ms)	Average Time (ms)
ANCA Spec, Main Diagnosis & Sex	2 each (2 * 2 * 2)	39	43	34	32	37

Table 5.4: ***Table showing application response times for changing the data element splitting a pair of charts***

5.4 Future Work

While the requirements determined to be within the scope of this project have been met, the application is not yet ready for deployment.

1 Formal user evaluation interviews should be conducted according the outline in *section 5.1 User Evaluation*. Feedback gathered from the questionnaire shown in *Appendix A2 PSSUQ Questionnaire* should be assessed and any proposed changes should be reviewed and implemented.

2 Support should be added for querying multiple registries simultaneously. This support was not added due to the limitations of the bar chart visualisation which would not support data with four elements. However, by communicating the results solely using the table, this limitation could be avoided until a solution is reached.

3 While authorisation and authentication of users was determined out of scope for this project, it will need to be implemented before the deployment of the web application.

- Django allows the developer to easily create and access a database which can store user log-in information [36]. One possible database that could be used is an SQLite database which is lightweight and fast [37] and is easily integrated into a Django project.
- A new webpage will need to be added to the application where the user can log into the system. This should be the landing page for the web application. Upon successfully logging in, the application should redirect the user to the existing webpage where they can pose queries to the registries.
- Django's database integration system allows unique permissions to be granted to

different users of the application. Certain users could be given access to only a subset of the registries, which would increase patient privacy.

4 The FAIRVASC team may propose additional queries that the user should be able to pose to the registries. In this case, handling for these queries should be added to the existing web page or another page should be created to and from which the user can navigate.

5.5 Chapter Summary

The planned user evaluation, the discussion of the client's requirements, and future work needed for deployment were discussed within this chapter. The following chapter discusses the project objectives, the author's personal goals for the project and the author's final thoughts, which concludes the paper.

6 Conclusion

The conclusion of this paper examines the success in meeting both the project objectives and the author's personal goals. Finally, the author's final thoughts about this project are discussed.

6.1 Meeting the Project Objectives

Four objectives were outlined in *section 1.2 Project Objectives*

O1 was to conduct a review of data visualisation options. This objective was met during the Planning phase of the project and is discussed in *subsection 3.2.7 Data Visualisation Research*. The nature of the data was examined and four data elements each containing discrete data were considered within the project. Three visualisation approaches were considered and a bar chart was chosen as the best way to represent the complex data.

O2 was to create a web application front-end that forms a query based on user input. This objective was met by creating a user interface that contains input widgets, such as check-boxes and drop-down menus. The user's input is processed and sent to the back-end as raw data where a python class constructs a SPARQL query.

O3 was to implement a web server back-end that poses the user's query to the relevant database or databases. This objective was only partially met. User queries can currently be posed to only a single database at a time. The back-end of the application can pose queries to a database and can parse the results.

O4 was to use data visualisation to convey the query results to the user effectively. This objective was met using bar charts and a table. Bar charts allow the user to quickly and easily compare the results of their query. A more in-depth review of the data can be done using the table, which is a less intuitive form of visualisation, but can communicate more detailed information.

6.2 Meeting my Personal Goals

Six personal goals were set out in *section 1.3 Personal Goals for the Project*.

P1 was to learn more about non-relational databases. During the research I conducted for this project, I worked with multiple RDF databases. I created a small, simple RDF database to learn how a graph database is formatted and populated. Through posing queries to DBPedia [38], an RDF adaptation of Wikipedia, using query software YASGUI [39], I gained insight into the complexity that can be contained within a graph database and I learnt how to pose complex SPARQL queries that could traverse multiple nodes of a graph. Finally, as discussed in *subsection 3.2.4 Querying RKD*, I posed queries to a local copy of RKD, which had been populated with synthetic data. This allowed me to explore the intricacies of the graph with which I would be working, and examine meta-data about the patients within the registry.

P2 was to learn about data visualisation techniques. A large aspect of this project was visualising data. I gained access to the notes used in a visualisation module for Computer Science Masters students. I studied the advantages and disadvantages of different visualisation techniques, the most relevant of which are outlined in *subsection 3.2.7 Data Visualisation Research*. I also focused on the importance of colour in visualisation, which led to my decision to create a palette of visually distinctive colours, each of which would correspond to a different parameter which could be shown on the bar chart.

P3 was to improve my ability to report to a team and receive and implement feedback. The FAIRVASC team met on a weekly basis to share their progress with the other members of the team. On three occasions, I presented my project to the team; once using the paper prototype and twice using the web application. Each time, I discussed the aspects of the user interface with the team and noted their feedback, which influenced how I proceeded with development.

P4 was to implement project management without over- or under-planning. At the beginning of this project, I created an overly-ambitious plan with many minute details. As the project progressed, I learnt which aspects of my plan were too demanding or were too detailed, and relaxed the rigid structure. The new plan that I created allowed for more freedom with project development, so that I could work around my other college obligations, and meant that I could more easily adapt the plan when I encountered an issue during implementation.

P5 was to learn how to recover from setbacks during implementation. Two such setbacks occurred, as discussed in *section 4.4 Implementation Issues*. Rewriting the server using Django after first implementing it in Express was a major setback. Django is written in Python rather than JavaScript like Express, so the entire back-end had to be rewritten. The project timeline was adapted to account for this setback.

P6 was to learn how to write a formal project report. During the planning phase of the project, I read three of the reports by previous students supplied by the School of Computer Science and Statistics. I created a report outline which I shared with my project supervisor. During the planning and implementation phases of the project, I kept notes detailing what aspects of the project I had done and where I was encountering issues. These notes proved vital during the creation of this report.

6.3 Final Thoughts

To conclude, this project's success can be determined in the completion of the project objectives and author's goals. The development of this web application will be handed to the FAIRVASC project teams where it will be deployed and used to help researchers learn more about ANCA-Associated Vasculitis.

Bibliography

[1] FAIRVASC. *About Us*. URL: https://fairvasc.eu/about-us/. last accessed 3 August 2021.

[2] See Me Hear Me. *What is ANCA-Associated Vasculitis*. URL: https://www.myancavasculitis.com/what-is-aav/. last accessed 8 August 2021.

[3] FAIRVASC. *About the Disease*. URL: https://fairvasc.eu/about-the-disease-aav/. last accessed 8 August 2021.

[4] FAIRVASC. *About Us*. URL: https://fairvasc.eu/the-project/. last accessed 3 August 2021.

[5] GO FAIR. *FAIR Principles*. URL: https://www.go-fair.org/fair-principles/. last accessed 3 August 2021.

[6] FAIRVASC. *About Us*. URL: https://fairvasc.eu/registries/. last accessed 3 August 2021.

[7] Vasculitis UK. *What is ANCA?* URL: https://www.vasculitis.org.uk/about-vasculitis/what-is-anca. last accessed 21 June 2021.

[8] Merriam-Webster Medical Dictionary. *Capillary*. URL: https://www.merriam-webster.com/dictionary/capillary#medicalDictionary. last accessed 3 August 2021.

[9] NHS. *Dialysis*. URL: https://www.nhs.uk/conditions/dialysis/. last accessed 8 August 2021.

[10] Mayo Clinic. *Symptoms: Eosinophilia*. URL: www.mayoclinic.org/symptoms/eosinophilia/basics/definition/sym-20050752. last accessed 21 June 2021.

[11] Davita Kidney Care. *ESKD - What is End Stage Kidney Disease*. URL: https://www.davita.com/education/kidney-disease/stages/what-is-end-stage-renal-disease. last accessed 8 August 2021.

[12] Merriam-Webster Medical Dictionary. *Granulomatosis*. URL: https://www.merriam-webster.com/medical/granulomatosis. last accessed 21 June 2021.

[13] Vasculitis Foundation. *What does the word Polyangiitis mean?* URL: https://www.vasculitisfoundation.org/mcm_faq/what-does-the-word-polyangiitis-mean-related-to-the-name-change/. last accessed 21 June 2021.

[14] Trinity College Dublin School of Medicine. *The Rare Kidney Disease Registry and Biobank*. URL: https://www.tcd.ie/medicine/thkc/research/rare.php. last accessed 8 August 2021.

[15] Centers for Disease Control and Protection. *Serotypes and the Importance of Serotyping Salmonella*. URL: https://www.cdc.gov/salmonella/reportspubs/salmonella-atlas/serotyping-importance.html. last accessed 21 June 2021.

[16] National Cancer Institute. *Definition of White Blood Cell*. URL: https://www.cancer.gov/publications/dictionaries/cancer-terms/def/white-blood-cell. last accessed 8 August 2021.

[17] M Yates and R Watts. "ANCA-associated vasculitis". In: *Clinical Medicine, Journal of the Royal College of Physicians of London* 17 (1 2017), pp. 60–64.

[18] W3C. *RDF Primer*. URL: https://www.w3.org/TR/rdf-primer/. last accessed 3 August 2021.

[19] Dataversity. *Introduction to: SPARQL*. URL: https://www.dataversity.net/introduction-to-sparql/. last accessed 3 August 2021.

[20] Apiko. *Express.js Mobile App Development: Pros and Cons for Developers*. URL: https://apiko.com/blog/express-mobile-app-development/. last accessed 5 July 2021.

[21] Reseller Club. *Flask vs Django: How to Choose the Right Web Framework for Your Web App*. URL: https://blog.resellerclub.com/flask-vs-django-how-to-choose-the-right-web-framework-for-your-web-app/. last accessed 5 July 2021.

[22] RubyGarage. *The Best JS Frameworks for Front End*. URL: https://rubygarage.org/blog/best-javascript-frameworks-for-front-end. last accessed 5 July 2021.

[23] MDN. *Introduction: What is JavaScript?* URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction#what_is_javascript. last accessed 3 August 2021.

[24] Axios Docs. *Getting Started*. URL: https://axios-http.com/docs/intro. last accessed 3 August 2021.

[25] MDN. *XMLHttpRequest*. URL: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest. last accessed 3 August 2021.

[26] JScrambler. *Introduction to Vue*. URL: https://blog.jscrambler.com/introduction-to-vue. last accessed 3 August 2021.

[27] W3 Schools. *Introduction to HTML*. URL: https://www.w3schools.com/html/html_intro.asp. last accessed 3 August 2021.

[28] W3 Schools. *CSS Introduction*. URL: https://www.w3schools.com/css/css_intro.asp. last accessed 3 August 2021.

[29] Stanley Ulili. *The Beginner's Guide to Chart.JS*. URL: https://www.stanleyulili.com/javascript/beginner-guide-to-chartjs/. last accessed 3 August 2021.

[30] Tabulator. *Tabulator*. URL: http://tabulator.info/. last accessed 3 August 2021.

[31] Python. *About Python*. URL: https://www.python.org/about/. last accessed 29 June 2021.

[32] Python Package Index. *SPARQLWrapper*. URL: https://pypi.org/project/SPARQLWrapper/. last accessed 14 March 2021.

[33] Django. *The Web Framework for Perfectionists with Deadlines*. URL: https://www.djangoproject.com/. last accessed 3 August 2021.

[34] The Apache HTTP Server Project. *Welcome!* URL: https://httpd.apache.org/. last accessed 3 August 2021.

[35] Apache Jena Fuseki. *Apache Jena*. URL: https://jena.apache.org/documentation/fuseki2/. last accessed 3 August 2021.

[36] Django Documentation. *User authentication in Django*. URL: https://docs.djangoproject.com/en/3.2/topics/auth/. last accessed 13 July 2021.

[37] SQLite. *About SQLite*. URL: https://www.sqlite.org/about.html. last accessed 13 July 2021.

[38] DBPedia. *About DBPedia*. URL: https://www.dbpedia.org/about/. last accessed 8 August 2021.

[39] Triply. *Yasgui*. URL: https://yasgui.triply.cc/. last accessed 8 August 2021.

A1 QueryBuilder.py

The QueryBuilder.py file can be found at: https://github.com/sastaffo/FAIRVASC-query-app/blob/master/fairvasc_backend/sparql_query/QueryBuilder.py

```
import pandas as pd
from io import StringIO
from SPARQLWrapper import SPARQLWrapper as sparql
from SPARQLWrapper import JSON as sparql_JSON
from SPARQLWrapper import CSV as sparql_CSV
import json

from sparql_query.constants import FVC_PARAMS

import logging

prefixes = ("PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
   ↪ PREFIX fvc: <http://ontologies.adaptcentre.ie/fairvasc#> ")
select_base = ("SELECT (COUNT(?p) as ?Patient_Count) ")
where_base_open = ("WHERE { ?p a fvc:Patient. ")
where_close_bracket = ("}")
group_by = (" group by ")
where_deceased = (" ?p fvc:hasOutcomes ?o. ?o fvc:dateOfDeath ?dd. ")
where_eskd = (" ?p fvc:hasOutcomes ?o. ?o fvc:dateOfESKD ?de. ")
strat_dict = {
    "sex" : {
        "where" : (" ?p fvc:gender ?Sex. "),
        "var" : (" ?Sex "),
    },
    "ancaSpec" : {
        "where" : ( "?p fvc:hasANCA ?anca. ?anca fvc:ancaSpec
           ↪ ?ANCA_Specificity. " ),
        "var" : (" ?ANCA_Specificity "),
    },
    "diagnosis" : {
        "where" : ( "?p fvc:hasDiagnosis ?diag. ?diag fvc:mainDiagnosis
            ↪ ?Main_Diagnosis. " ),
        "var" : (" ?Main_Diagnosis "),
    },
}



class QueryBuilder():
    def __init__(self, total_counts=True, outcome_deceased=False,
        ↪ by_sex=False, by_ancaSpec=False, by_diagnosis=False):
        self.logger = logging.getLogger('console')
        self.total_counts = total_counts
        self.outcome_deceased = outcome_deceased
        self.by_sex = by_sex
        self.by_ancaSpec = by_ancaSpec
        self.by_diagnosis = by_diagnosis
        self.select_string = select_base
        self.where_string = where_base_open
        self.group_string = where_close_bracket
        self.result_dict = {}
        self._get_empty_results()
        if not total_counts:
            # if stratified, strat by deceased xor by eskd
            if outcome_deceased: self.where_string = self.where_string +
                ↪ where_deceased
            else : self.where_string = self.where_string + where_eskd
        if by_sex or by_ancaSpec or by_diagnosis:
            self.group_string = self.group_string + group_by
            if by_sex: self._add_strat("sex")
            if by_ancaSpec: self._add_strat("ancaSpec")
            if by_diagnosis: self._add_strat("diagnosis")
        self.query_string = prefixes + self.select_string +
            ↪ self.where_string + self.group_string
    # end init

    def __str__(self):
        return self.query_string

    def _add_strat(self, strat_by):
        strat_var = strat_dict[strat_by]["var"]
        self.select_string = self.select_string + strat_var
        self.where_string = self.where_string +
            ↪ strat_dict[strat_by]["where"]
        self.group_string = self.group_string + strat_var
    # end _add_strat

    def send_query(self, endpoint):
        sparql_query = sparql(endpoint)
        sparql_query.setReturnFormat(sparql_CSV)
        sparql_query.setQuery(self.query_string)
        result_str = (sparql_query.query().convert()).decode("utf-8")
        result_dict_raw = self._parse_csv_string(result_str)
        self._add_results_to_empty(result_dict_raw)
        return self.result_dict
    # end send_query

    def _parse_csv_string(self, result_str):
        result_buffer = StringIO(initial_value=result_str, newline="\r\n")
        df = pd.read_csv(result_buffer)
        result_dict_raw = df.to_dict(orient="index")
        return result_dict_raw

    def _add_results_to_empty(self, result_dict_raw):
        for r_raw in result_dict_raw:
            row_raw = result_dict_raw[r_raw]
            for r in self.result_dict:
                row = self.result_dict[r]
                if (not self.by_ancaSpec) or (self.by_ancaSpec and
                    ↪ (row['ANCA_Specificity'] ==
                    ↪ row_raw['ANCA_Specificity'])):
                    if (not self.by_diagnosis) or (self.by_diagnosis and
                        ↪ (row['Main_Diagnosis'] ==
                        ↪ row_raw['Main_Diagnosis'])):
                        if (not self.by_sex) or (self.by_sex and
                            ↪ (row['Sex'] == row_raw['Sex'])):
```

```
                        row['Patient_Count'] = row_raw['Patient_Count']
            self.result_dict[r] = row
    return

def _get_empty_results(self):
    anca_list = []
    diag_list = []
    sex_list = []
    if self.by_ancaSpec:
        for a in FVC_PARAMS['ancaSpec']:
            anca_list.append({ "Patient_Count" : 0,
                ↪ "ANCA_Specificity" : a })
    else: anca_list.append({ "Patient_Count" : 0 })

    if self.by_diagnosis:
        for d in FVC_PARAMS['diagnosis']:
            for a_row in anca_list:
                d_row = a_row.copy()
                d_row['Main_Diagnosis'] = d
                diag_list.append(d_row)
    else: diag_list = anca_list

    if self.by_sex:
        for s in FVC_PARAMS['sex']:
            for d_row in diag_list:
                s_row = d_row.copy()
                s_row['Sex'] = s
                sex_list.append(s_row)
    else: sex_list = diag_list

    for index in range(0, len(sex_list)):
        self.result_dict[index] = sex_list[index]
    return
```

A2 PSSUQ Questionnaire

1. **Overall, I am satisfied with how easy it is to use this application**
   Strongly Agree ☐ ☐ ☐ ☐ ☐ Strongly Disagree ☐ N/A

2. **It was simple to use this application**
   Strongly Agree ☐ ☐ ☐ ☐ ☐ Strongly Disagree ☐ N/A

3. **I was able to complete the tasks and scenarios quickly using this application.**
   Strongly Agree ☐ ☐ ☐ ☐ ☐ Strongly Disagree ☐ N/A

4. **I felt comfortable using this application.**
   Strongly Agree ☐ ☐ ☐ ☐ ☐ Strongly Disagree ☐ N/A

5. **It was easy to learn to use this application.**
   Strongly Agree ☐ ☐ ☐ ☐ ☐ Strongly Disagree ☐ N/A

6. **I believe I could become more productive using this system.**
   Strongly Agree ☐ ☐ ☐ ☐ ☐ Strongly Disagree ☐ N/A

7. **The application gave error messages that clearly told me how to fix problems.**
   Strongly Agree ☐ ☐ ☐ ☐ ☐ Strongly Disagree ☐ N/A

8. **Whenever I made a mistake using the application, I could recover easily and quickly.**
   Strongly Agree ☐ ☐ ☐ ☐ ☐ Strongly Disagree ☐ N/A

9. **The information (such as online help, on-screen messages, and other**

documentation) provided with this system was clear.

Strongly Agree ☐ ☐ ☐ ☐ ☐ Strongly Disagree ☐ N/A

10. It was easy to find the information I needed.

    Strongly Agree ☐ ☐ ☐ ☐ ☐ Strongly Disagree ☐ N/A

11. The information was effective in helping me complete the tasks and scenarios.

    Strongly Agree ☐ ☐ ☐ ☐ ☐ Strongly Disagree ☐ N/A

12. The organisation of information on of the application was clear.

    Strongly Agree ☐ ☐ ☐ ☐ ☐ Strongly Disagree ☐ N/A

13. The interface of this system was pleasant.

    Strongly Agree ☐ ☐ ☐ ☐ ☐ Strongly Disagree ☐ N/A

14. I liked using the interface of this application.

    Strongly Agree ☐ ☐ ☐ ☐ ☐ Strongly Disagree ☐ N/A

15. This system has all of the functions and capabilities I expect it to have.

    Strongly Agree ☐ ☐ ☐ ☐ ☐ Strongly Disagree ☐ N/A

16. Overall I am satisfied with this system.

    Strongly Agree ☐ ☐ ☐ ☐ ☐ Strongly Disagree ☐ N/A

If you answered "Somewhat Disagree", "Disagree", or "Strongly Disagree" to any of the above statements, can you elaborate on your reasoning?

Do you have any other feedback regarding the Registry Query Application?