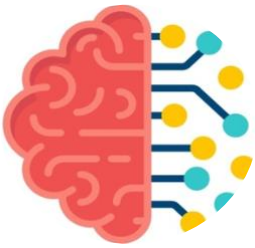



# AI Research Graph

CSC3003S Capstone Project

**Client:** Prof. Deshen Moodley

Jane Imrie, IMRJAN001

Nichola Pennington, PNNNIC003

Yuval Ardenbaum, ARDYUV001

# Abstract

This project focused on the creation of a knowledge graph that can be queried (with various filters, or combinations of filters by an end-user) to yield insights of varying depth into the Artificial Intelligence research ecosystem in South Africa.

To this end, we have implemented a minimalist web application that enables users to interface with a graph database easily and intuitively, by abstracting complex functionality away from the user interface. Several categories of entities were identified as having importance to understanding the AI research environment, including researchers, research institutions, and other bodies of research, and specialisations within the field. These entities have been brought into effect as nodes within the knowledge graph, with their relationships shown as edges/vertices.

This has been accomplished through the use of several frameworks. The creation of an underlying graph database was achieved by using the Stardog graph database management software. The Python-based Flask framework was utilized for UI design, while the front- and back-end were integrated with the use of Python in an object-oriented style. This knowledge graph ultimately enables users to visualize data, including specifically queried data and relationships, in order to analyze trends in one of the fastest evolving fields of study being pursued currently.

# Knowledge Graph of Artificial Intelligence in South Africa

Table of Contents

# 1. Introduction

In this project, we have been tasked with developing an Artificial Intelligence Research graphing interface. The project aims to enable users to query the Microsoft Academic Graph (MAG) database, as well as the National Research Foundation (NRF) dataset, and return a visualized summary of all AI research correlating to the user-inputted query. The target audience of the web interface is those who are interested in AI research within South Africa, such as South African researchers wanting insight into their peers' work, students looking to further their studies and the stakeholders of the project.

The stakeholders of the AI Research Graph project are the development team, Project Supervisor and client, Associate Professor Deshen Moodley, and facilitator and tutor Willie Macharia. The stakeholders of the project helped guide the development team in terms of how the back-end functionality should operate and how the front-end should capture user input, among other things.

When solving the problem at hand we made use of agile development methodologies, following more of an incremental or cyclic development and design process. Agile development techniques worked best for our team, given the skill sets of team members, the learning curve that needed to take place for software being used, and the time constraints of the project. The team also followed test-driven development practices to ensure the system was functioning as expected as we developed. This prevented us from producing a flawed final product close to the deadline that performed in unexpected or unpredictable ways.

We started the project by performing analysis and requirements gathering to gain better insight into the project and better understand the expectations of our team and the complete system. Thereafter, we deeply engaged in online research on what kind of software, languages and frameworks would be the most appropriate fit for this project. Initially, we considered Java for back-end development, and React.js framework for front-end. However, after struggling to find a framework that would allow us to access the MAG dataset in the backend, we changed over to Python and Flask to develop our front- and back-end, making use of Stardog graph database software and the SPARQL query language.

As a team, we then developed horizontal, evolutionary prototypes to demo to our stakeholders as a proof of concept for approval. Since it was early on in the project, the prototypes were developed using React Framework in the front-end and GraphDB for the backend. Despite the framework needing to be changed, our stakeholders accepted our proposed design and we migrated it to Python and flask frameworks, which were used to develop the final product.

In the ever-more-interconnected research environment, with greater numbers of sub-disciplines being discovered and spawned each year, as well as cross-disciplinary convergence, capturing a snapshot of the current research environment is essential to gaining a broader perspective on a subject in its entirety. Several cross-disciplinary or novel AI-centric research areas have unfolded in the past five years alone, and this volume of research can easily appear chaotic and disparate to the naked eye. Hence (perhaps ironically so), by making use of an emergent field of study within AI, namely knowledge graphs and an additional field, ontology, we are able interact with, filter and visualize trends and relationships amongst and between relevant entities within the field of AI in South Africa.

In acknowledging the utility of knowledge graphs, and the need for "...the general public to develop intuitions about the complex field of AI"[1], as well as understanding the broadly-encompassing implications of development in this field for society, commerce and governance, we have embarked upon a process of developing a simple iteration of such a knowledge graph. This has been undertaken with a view toward implementing object-oriented software development principles and best practices, as well as refining our development methodologies iteratively.

[1] https://aiindex.stanford.edu/wp-content/uploads/2021/03/2021-AI-Index-Report_Master.pdf

# 2. Requirements Captured

As a result of our implementation of an Agile development methodology, we approached requirements gathering with our client as a mutual process of clarifying the desired functionality and understanding relevant constraints. Furthermore, elicitation of requirements has been iterative, evolving as we developed the system, communicated further with the client and uncovered new tools or obstacles that directed the development process. Outlined below are the various stages of requirements evolution.

## 2.1 Initial Requirements

In our initial meeting with the client for the project, we were able to formulate a set of requirements that were composed of non-negotiable functional use-cases, non-functional needs, and some recommendations from the client for our approach. Our understanding of these initial requirements was further informed by the short brief provided by the client. These requirements included:

### Initial Functional Requirements

The final application, as outlined by the client, was envisioned to satisfy the following requirements:

1. The ability to submit plain-English queries to a graph database through a web application interface, with the query results visually displayed to the user through the interface as a graph, with nodes and edges. This would form the basis of a simple knowledge graph of the South African AI research community.
2. Several possible query types, and combinations thereof, should be made possible for the end user, as described below.
3. A given researcher should serve as a node in the database, and other columns in that researcher's row (entity nodes associated with the researcher) should be their affiliated institution, NRF rating and their topics, subtopics or specializations of research.
4. The full suite of data associated with each entity should be query-able by the end user through the web interface.
5. Multi-attribute or integrated queries should be possible insofar as they are valid/sensible, such that a user can query the number of papers published by a researcher, or the number of researchers affiliated with a given institution. The query results should then be visually displayed.
6. The Microsoft Academic Graph and NRF datasets should be available to the user for some query types.
7. The relevant datasets should be implemented as a local graph database instance through some form of graph database management software.
8. The graph database should be capable of integrating with existing databases of researchers in addition to MAG, including AMiner, a large-scale AI knowledge graph built by researchers in China.

The graph database was stipulated to be a means of organizing and accessing data about AI research entities and their relationships. Namely, entities were determined to be researchers, research papers, conference or workshop proceedings, research institutions, research topics within AI, and research subtopics. Data about each instance of every entity type is included in the database.

### Initial Non-Functional Requirements

1. The graph database itself should be abstracted away from the web application interface, in order to allow end users to interact with the services provided without any expertise on the underlying software frameworks or database query languages.

2. With respect to performance, the service is required to process queries and return results timeously, as well as utilize the most current data available from the most relevant sources.

3. To this end, it should be capable of updating at any stage to utilize the most recent iterations of the NRF data, as well as data sourced from the Microsoft Academic Graph and AMiner.

4. The application's design is also required to adhere to best practices of secure software design, as well as reliability.

5. In this regard, it is required to have security measures implemented that prevent bad actors from directly accessing, and potentially altering, the graph database itself. This is crucial to avoid providing users with inaccurate or tampered data that might skew their perceptions of South African AI research as a whole.

## 2.2 Evolving Requirements

As our initial efforts unfolded and we became more acquainted with the frameworks suggested by our client, several insights were gained into the realistic achievement of the requirements initially stipulated. Our team members grappled with the question of which web-development stack to use, as well as the best graph database management system for the task. Given our limited experience in both disciplines (front-end development and use of graph databases), we explored multiple options, before attempting to use GraphDB as suggested, alongside React.js, a javascript-based web-development framework recognized for its reasonable learning curve and low barrier to entry.

After investing time and effort into the use of these frameworks in our first prototyping 'sprint', we determined that GraphDB documentation assumed significant prior knowledge that we didn't possess, and that the number of available resources online were limited, particularly for integration with a web application. As such, we researched and made the switch to a graph database management system called Stardog, with slightly more available resources, and native Python integration. It was more commonly used in full stack graph database applications that we could identify in our research, and so it appeared to be more plausible to troubleshoot or address any issues that might arise. Furthermore, to gain access to continued tutor support for our web-development efforts, we made a decision to transition to Flask, a Python-based web application framework, in which our tutor had the most experience and was most equipped to assist us. This seemed to be a natural decision given that Stardog was well integrated with the Python language.

In exploring our initial choice of frameworks, researching alternatives, and undergoing the transition phase, we realized that some of the initial requirements requested by the client would not be possible to implement in the given timeframe of this project, or with the resources available to us. After communicating our concerns to the client in our following meeting, we mutually arrived at a more realistic and focused set of requirements that we felt was more attainable. This updated set of requirements is outlined below.

## The Evolution of Functional Requirements

Having convened with our client, we established a new constrained requirements set to encompass lesser functionality and smaller datasets, in order to demonstrate core functionality in the final product, without having a wide but superficial scope.

This final scope included two possible 'integrated' queries with Microsoft Academic Knowledge Graph and NRF researcher data, with limited fields from each being query-able. It was determined that we could enable remote usage of the MAKG SPARQL endpoint without having to locally store all the encompassed RDF data (more than a terabyte) in a graph database. However, given the low-fidelity, small-scale nature of the researcher data from the NRF, a local graph database instance would be constructed, and a schema instance created to store and query information pertaining to this dataset. Further, we would focus on those two datasets, and forego integration with AMiner.

Moreover, there were issues in reconciling the output from MAG and the NRF data. One needs to be able to identify when a researcher in the NRF dataset, e.g. AJ Herbst, is the same AJ Herbst present in the MAG dataset. The names in the NRF schema have been standardized, to be of the format <Two Letter Inititals><Surname>. However, this is not the case with the MAG, where names are unstandardized. An author in MAG, for example, CJ De Beer (from NRF) in MAG could possibly be presented as "Charles J De Beer" or "Charles John DB" or "C.J De beer". This makes combining the results from an integrated query with the NRF data and MAG very difficult, and often resulted in a null result set, because there is no way of testing if the Charles J De Beer from institution X in MAG is the same CJ De Beer from MAG. Hence, we made the decision to provide integrated query functionality without concerning ourselves with the overlap between or integrity of the datasets themselves, as altering the datasets for our purposes would be infeasible, and disingenuous to the end user.

The user should be able to select keywords or search terms from dropdown selections on the UI, which are then converted into SPARQL queries in the backend infrastructure, and outputs displayed in visual and easily understandable formats in the web interface to the user.

# 2.3 Use Cases

1. **Submit a Query**

   **Actor:** Application user

   The user would like to conduct a query to view the research topics within Artificial Intelligence that different South African institutions are engaged in. They open the web-based Artificial Intelligence knowledge graph interface in their browser, and a section of the UI displays various selection criteria to query the database (as dropdown options).

   Alternatively, the user may choose not to submit any queries – in which case an error will be thrown and an error screen will be displayed to the user.

   The user will utilize various checkboxes, drop-down menus and radio buttons in order to formulate their query. Once they have submitted the desired query parameters, the query will be forwarded to the back-end for processing. If their query is valid, then the results will be displayed as a graph with nodes and edges, and descriptions for each node and edge or as table, summarizing the results of the query. If their query is invalid, an error will be thrown and an error screen will be displayed to the user.

2. **Clear Query Filter**

**Actor:** Application user

The user has submitted a query and been presented with visualized results. Now, the user would like to clear all filters put on the graph, so that they can submit a new query and visually display some relationship within the data.

The user will select the 'Home" option on the navigation bar to return to the home screen, to capture a new query. When navigating to the home page, all query filters will be reset to capture a new query. In addition to this, there is also a "reset" button on the home GUI to allow users to clear selections made. Then the system will display the full results graph or table to the user once again once they're clicked the "Find Research" button.

3. **Access Web-based UI**

**Actor:** Application User

The user would like to access the application. As such, they will enter the URL into their browser and be directed to the application. The system will then access the graph database via an API (Chart.js), and visually display the unfiltered data to the user. The system will also display a section of the webpage that enables the user to enter query parameters and filter the data (index.html).

4. **Upload new NRF schema and Data**

**Actors:** App-User, Developer

The NRF data gets updated periodically, as researchers can obtain higher ratings and new researchers are added. As a result, the end-user can make a request to the application developer in order to request that the database be updated to include the new data. The application developer can then obtain the new NRF data from the institution's website and convert it into a *ttl* file using the relevant software. Then, they can ensure the data is in the correct directory and follows the correct naming convention. When the application is next run, the old data will be dropped from the database and the new data will be added.

If there is an issue with the data, and it cannot be added to the database, an exception will be raised and the developer will have to reformat the file until it is correct.

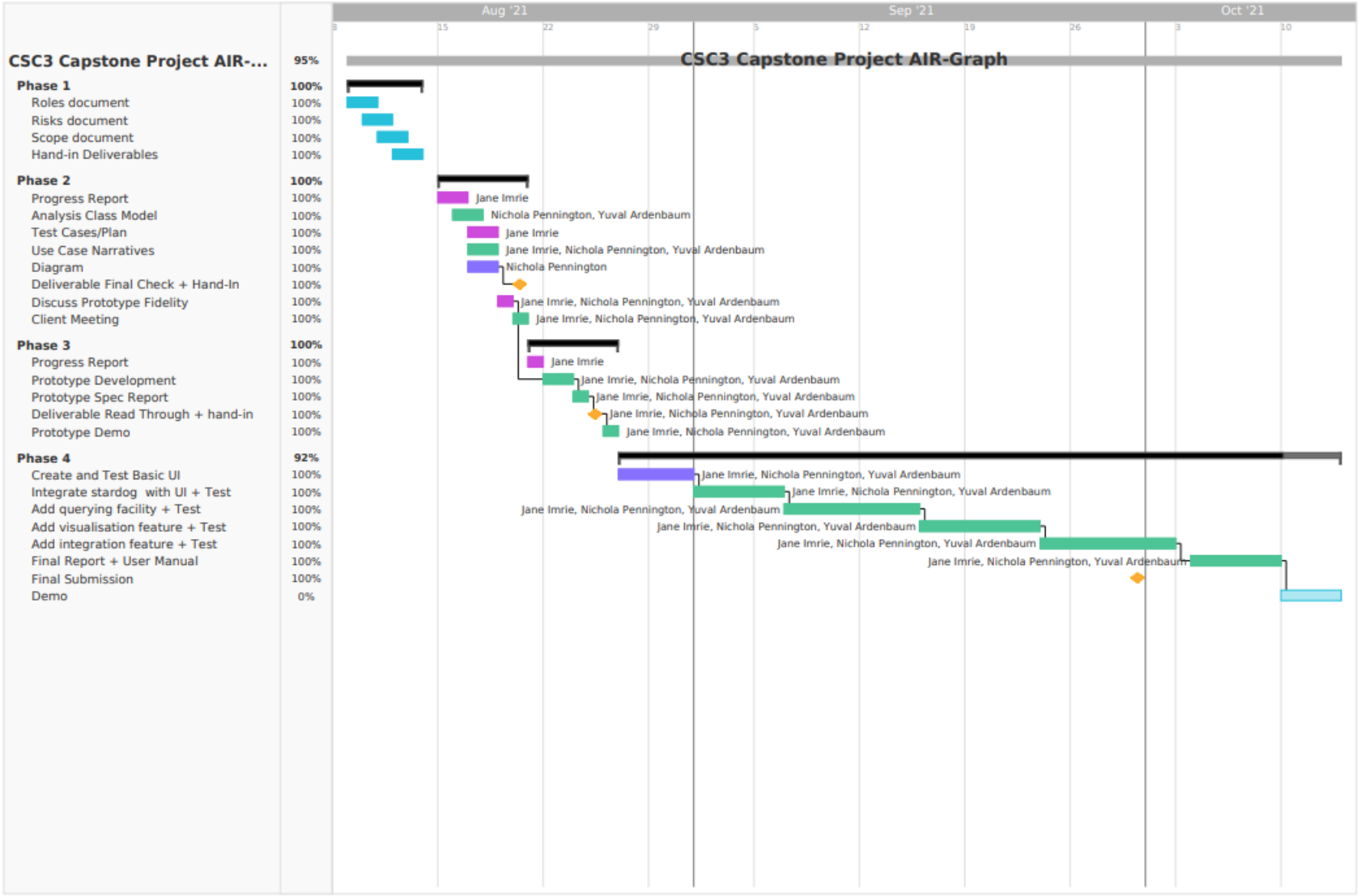
## 2.4 Analysis Artefacts

Stages 1-3: these stages are where the use case narratives, prototyping and class diagrams were planned.

https://drive.google.com/drive/folders/19NKJCi8gUH8A6--RXfupig0LlmH2Jb9u?usp=sharing

A change in submission dates and requirements facilitated the need to review and redo some aspects of the 4th phase of the project. Consequently, a new Gantt chart was drawn up to account for these dates.
This new Gantt Chart has been included below. The final component that would lead to 100% project completion is the demo, which the team will complete in the scheduled timeslot and day.

*Figure 1: Gantt chart with updated details and deadlines*



# 3. Design Overview

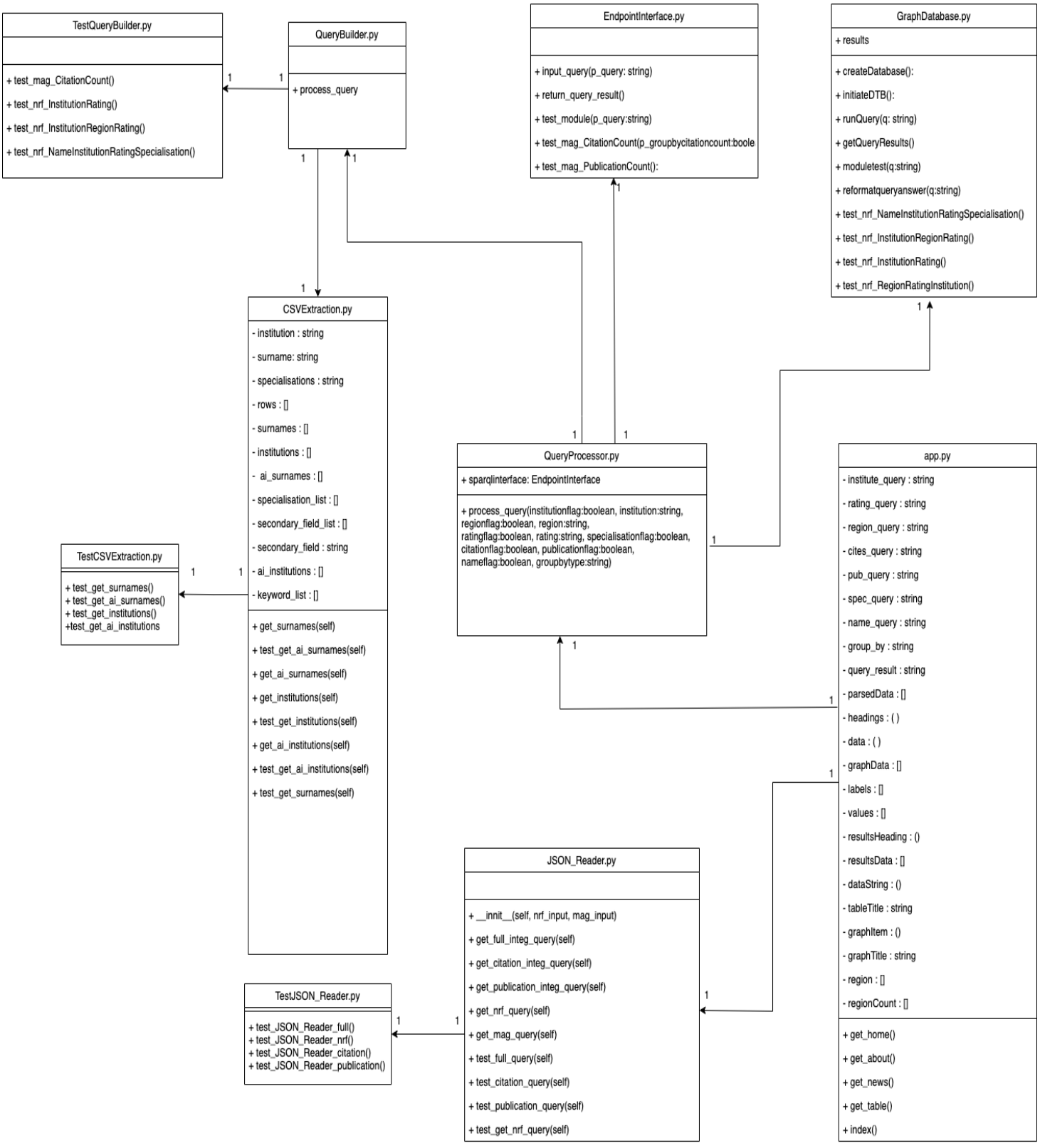
## 3.1 Design Justification

The AI Research Graph application is expected to provide a way of assessing the state of the AI community in South Africa. It is expected to provide users with a way of querying data relating to the AI community and be provided with results in the form of visualizations, such as charts and tables. These results can be used to gain insight into the various aspects of the AI community in the country – such as what researchers are specializing in, in terms of their research, as well as what institutions are conducting research in AI.

For these reasons, the application needed to be designed in such a way that it was modular, and easy to expand – so that additional data and functionality can be added as the user requires it. Additionally, the application must be flexible, as changes to the NRF data and schema for the MAG can happen at any point, without warning.

## 3.1.1. Class Diagram

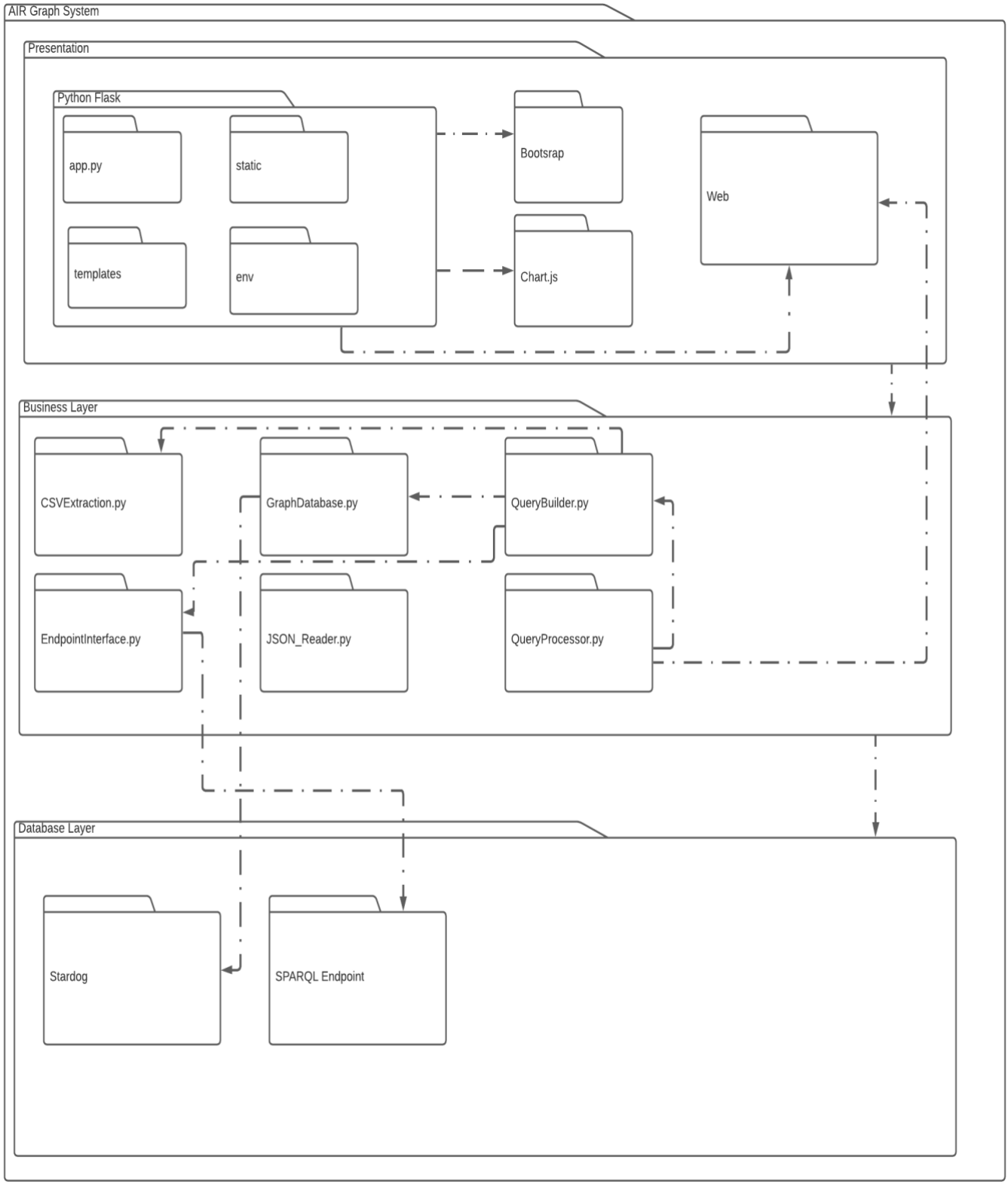
*Figure 2 Class Diagram for AIR Graph System*



## 3.2 System Architecture

### 3.2.1 Architecture Diagram

*Figure 3 Architecture Diagram*



## 3.3 Architecture Overview

A Layered Architecture was used in developing this application. A number of factors were taken into consideration when deciding on which architecture to use – these included the nature of the project (a web-based application), the various data structures need, as well as what good software engineering principles could be fulfilled by using a specific pattern.

A Layered Architecture offers a number of advantages – it allows for the separation of concerns, as components within a layer need only worry about being able to provide a specific functionality, and the modularity of such an architecture allows for easy modifying of a layer/layers. So long as a layer provides the functionality as what it needs to, many internal changes can be made to it, without disrupting the functionality of the application as a whole. An example of this would be if a decision was made to change the graph database software from Stardog to TripleGraph – the system would be able to accommodate it. Furthermore, new layers can be added to provide additional features to the system. Our system provides a *strict* layered architecture – i.e., a layer can only request services from a layer directly beneath it, and thus cannot circumvent one layer to reach a lower one. This application is relatively small, and so can be divided into three main layers – Presentation, Business and Database.

Another reason for using this architecture was the ease of testing. A layer can be emulated, and thus testing can be done in isolation for each layer.

### Presentation Layer

The presentation layer that the user interacts with – in this project, this is done through navigation menus, checkboxes, radio buttons and drop-down menus. Using Flask, various CSS and HTML templates are rendered to the browser. Graphs are rendered to the webpage using Javascript.

### Business Layer

This layer is responsible for using rules to determine the operating of the application. For example, “If the user selects the Specialisations option, then the NRF database needs to be queried”. This layer receives the input from the presentation layer, processes it and handles various interactions with the databases. QueryProcessor receives the input from App and uses a set of criteria to determine which methods must be called from QueryBuilder to build the correct SPARQL query. QueryProcessor uses supplementary classes such as JSON_Reader and CSVExtraction when building these queries.
Once the query has been created, then QueryProcessor determines which endpoint(s) to send the query to. Queries are either sent to GraphDatabase, which handles queries meant for the locally hosted RDF database, or to the EndpointInterface – the MAG SPARQL (SPARQL Protocol and RDF Query Language) endpoint hosted remotely. Query results are then retrieved or returned from these databases, cleaned, standardized, and finally, returned to the presentation layer.

### Database Layer

This layer contains the data managed and accessed by the application. The data that was acquired from the NRF is hosted locally in a database, which is accessible via an instance of a Stardog server. Raw data was acquired from the National Research Foundation – it was then converted into an RDF compliant that is “queryable” using SPARQL.
The data from the Microsoft Academic Graph Knowledge Graph is hosted on remote servers and is accessible via it’s SPARQL endpoint.

## 3.4 Algorithms and Data Organization

### 3.4.1 Data Organisation

This section provides details on the graph databases, the Microsoft Academic Graph, it's schema, problems with aforementioned schema and the NRF schema (that the team designed)

#### *Graph Databases*

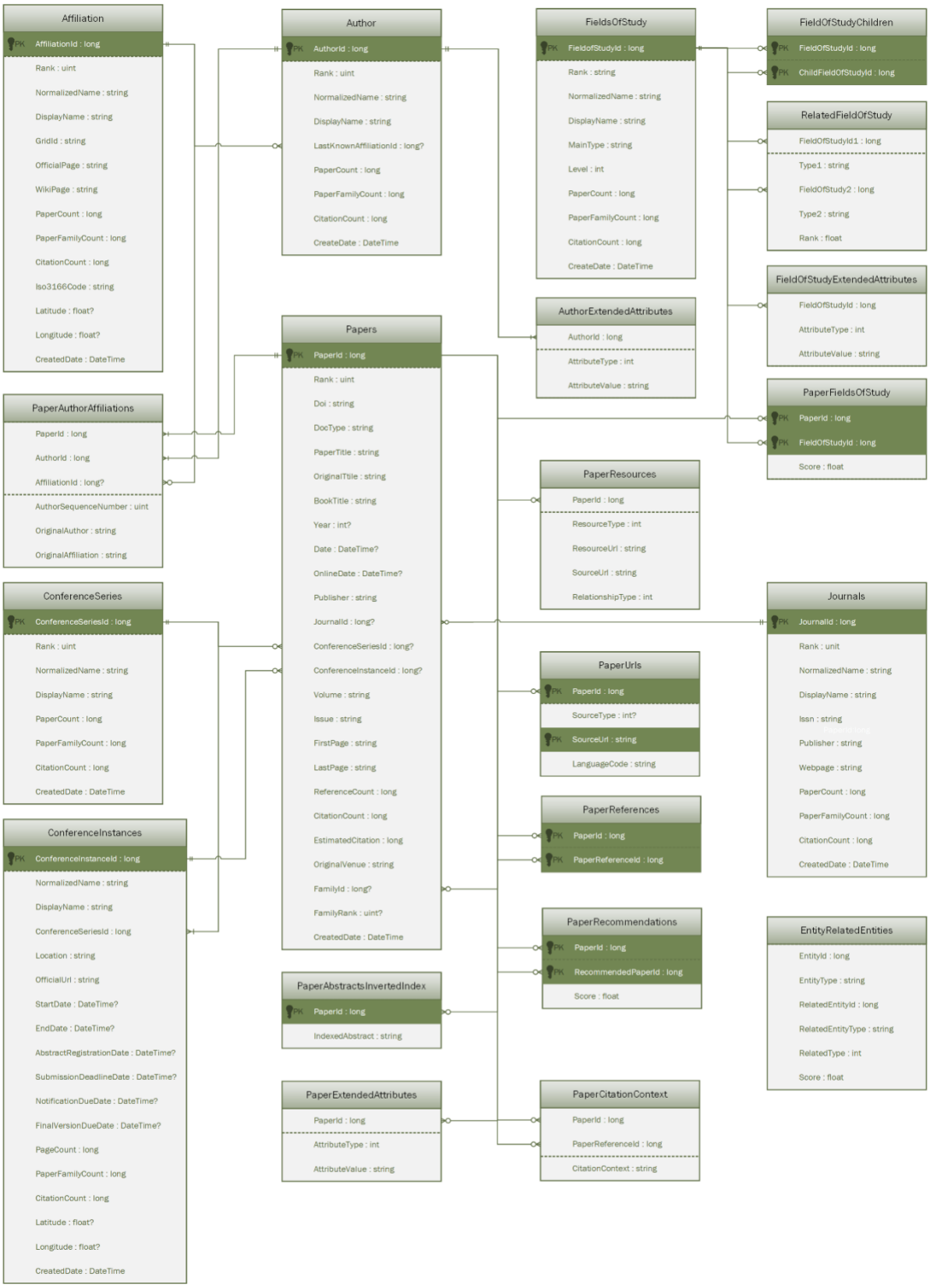
Simply, a graph database is "a database that uses graph structures for semantic queries with nodes, edges, and properties to represent and store data." [2] The relationships between data are treated as being equal (in terms of importance) to the data itself – edges are used to represent these relationships.[3]

#### *Microsoft Academic Graph*

The MAG is an RDF database that stores Triples. These triples are in the form of subject-predicate-object and can be used to show or discover relations between entities. The MAG schema is highly complex and interconnected, with several different class. The current MAG schema, as shown on the Microsoft website is this:

[2] https://en.wikipedia.org/wiki/Graph_database
[3] https://neo4j.com/developer/graph-database/

*Figure 4: MAG ERD schema*

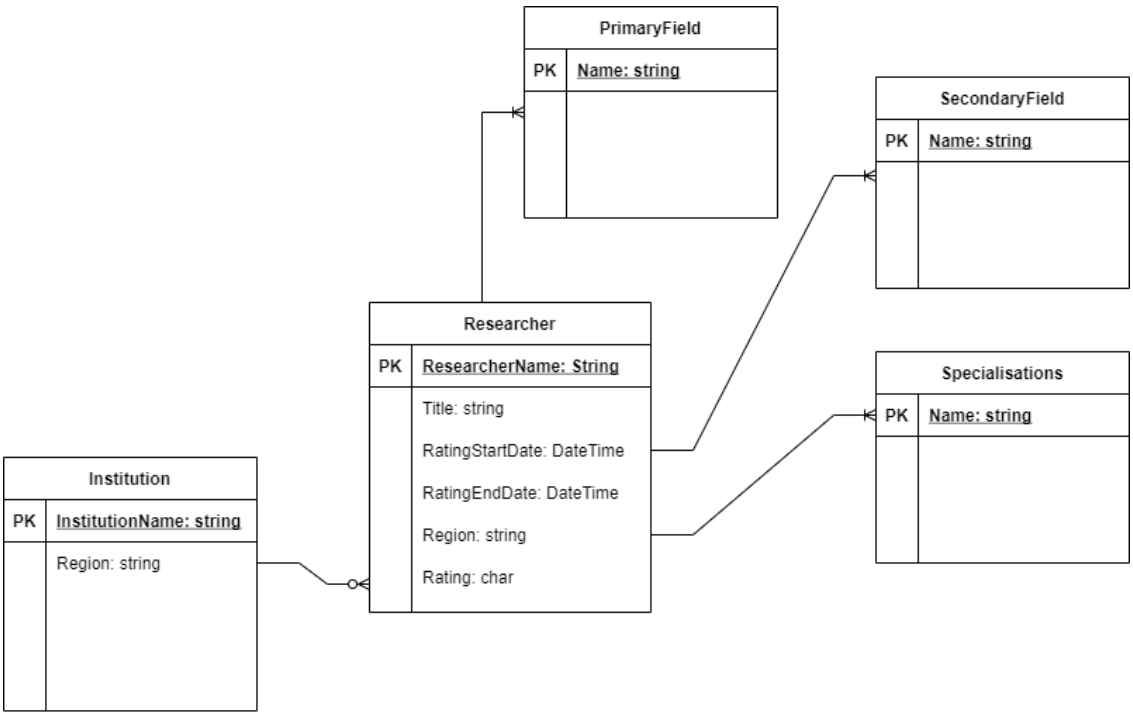


However, there exists some discrepancies between the MAG schema presented on the website, and the ontology present on the Microsoft Academic Knowledge Graph (MAKG), which hosts the SPARQL endpoint. These discrepancies are apparent in the ontology file present on the site. MAG shows a class called FieldsOfStudy – which shows a traversable subclass called FieldsOfStudyChildren. However, upon examining the ontology for the MAKG – it shows that no class was created for FieldOfStudyChildren[5]. The consequences of this being that is no longer possible to find the subtopic of research that a researcher is concerned with. Additionally, the ambiguity of a given field of study, e.g. “Segmented regression"[6], renders is near impossible to determine whether a researcher, given a field of study, is involved in artificial intelligence or not.

Given these constraints, we have decided to focus on just the citation count and publication count for a given author – both attributes that are accessible from the author class.

### *Locally Hosted Database and the NRF Schema*

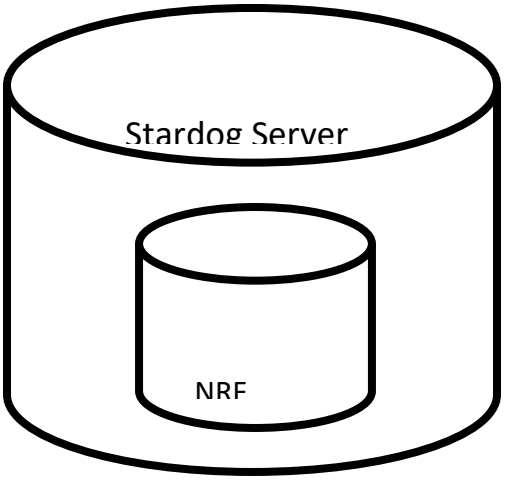
*Figure 5 Entity-Relationship Diagram for NRF Graph Database Schema*




The RDF formatted NRF data is held within a database called NRFDatabase – the data was converted into a format that the database could process. This database runs on an instance of a Stardog server.

[5] https://makg.org/ontology.owl
[6] https://makg.org/samples/2018-11-09/FieldsOfStudy_s.nt

*Figure 6: Diagram showing simplified version of database*



# 4. Implementation

## 4.1 Classes Structure, Hierarchy and Function

The application was created using Python – this language is "module" based and is not optimized for OOP compared to other programming languages. However, its dynamic, flexible and modular nature is well suited to this kind of project, and the modules are referred to as classes in this document for the sake of clarity.

*Table 1: Table of classes, their functions and methods*

| Class Name | Layer | Function | Significant Methods |
|---|---|---|---|
| **App** | Presentation | The app.py class acts as the presentation layer of the interface and includes all methods needed to render and display the html interface. In addition to this, it also includes the functionality needed to capture the user's query inputs from the web -interface, process the query using the relevant call and display either a tabulated or graphed result by redirecting the user to the relevant rendered HTML page. | **def index():**<br>• This method captures the user's input in the GUI's components. This is done by declaring the method as a 'POST' method and thereafter, capturing the elements from the index.html page's form. The form in the index.html page acts as a capturing mechanism for the html GUI components.<br>• Once the user's input is captured, it will compute the query, using a call to the relevant graph database and query classes. Once the data is queried it is processed depending on certain criteria and displayed either as a table or graph. Thus, either the table.html or update.html pages are rendered sending in the relevant parameters that are used to display the results<br><br>**def get_home():**<br>• Renders and formats all elements on the index.html page, ready to be displayed for the user. This is the home page containing the GUI to capture the user's query as well as some<br><br>**def get_about():**<br>• Renders and formats all elements on the about.html page, ready to be displayed for the user. The about page gives a summary of the project and team members. |

| | | | |
|---|---|---|---|
| | | | **def get_news():**<br>• Renders and formats all elements on the update.html page, ready to be displayed for the user. This page displays a graphic result to the user's query.<br><br>**def get_table():**<br>• Renders and formats all elements on the table.html page, ready to be displayed for the user. This page displays a tabulated result to the user's query. |
| **CSVExtraction** | Business | Extracts lists of researchers and institutions involved in AI in South Africa, using a curated list of keywords. These lists will be used in SPARQL queries to create the necessary filters when querying the databases. | **def get_surnames():**<br>• Returns a list of the surnames of all researchers found in the NRF rated-researcher dataset.<br>**def get_ai_surnames():**<br>• Returns a list of the surnames of researchers found in the NRF rated-researcher dataset publishing in AI-related fields of study or specializations.<br>**def get_institutions():**<br>• Returns a list of the publishing academic institutions with NRF-rated researchers.<br>**def get_ai_institutions():**<br>• Returns a list of academic institutions affiliated with NRF-rated researchers publishing in AI-related fields. |
| **EndpointInterface** | Database | Provides a way of interacting with a remote SPARQL endpoint hosted by the Microsoft Academic Graph using Virtuoso. Accepts SELECT type queries, interfaces with the endpoint, submits the queries and retrieves the results set, which is returned in the JSON format | **def input_query(self, p_query):**<br><br>• This method accepts a SPARQL query formatted as a string and returns the results as a JSON object. It sends the query to the SPARQL endpoint via an HTTP request and retrieves the results. |

| | | | |
|---|---|---|---|
| **GraphDatabase** | **Database** | **Provides a way of interacting with a locally hosted graph database. Accepts correctly formatted SPARQL select-type queries, submits it to the database and receives the results set – returned in JSON format.** | **def initiateDTB():**<br>• **This method connects to the database using a set of account details. It flushes the old data from the database and then adds new data. If the database is successfully started, it notifies the user of it.**<br><br>**def createDatabase():**<br>• **This method is only run once, when the program is first started, to create a locally hosted database (NRFDatabase).**<br><br>**def runQuery(q):**<br>• **This method accepts a SPARQL query formatted as a string and returns the results as a JSON object. The query is sent to a locally hosted database.** |
| **JSON_Reader** | Business | | **def get_nrf_query():**<br>• Returns a list of dictionaries, with each dictionary representing an individual query result and containing all the attributes queried from the NRF dataset for that given result.<br>**def get_mag_query():**<br>• Returns a list of dictionaries, with each dictionary representing an individual query result and containing all the attributes queried from the MAG dataset for that given result.<br>**def get_full_integ_query():**<br>• Returns a list of dictionaries, wherein each dictionary represents an individual query result including the full suite of integrated MAG and NRF properties for a given researcher (ResearcherName, Institution, Rating, CitationCount and NumberOfPublications).<br>**def get_citation_integ_query():**<br>• Returns a list of dictionaries, wherein each dictionary represents an individual query result with the CitationCount property from |

| | | | |
|---|---|---|---|
| | | | the MAG dataset and some NRF properties for a given researcher (ResearcherName, Institution and Rating).<br>**def get_publication_integ_query():**<br>• Returns a list of dictionaries, wherein each dictionary represents an individual query result with the NumbrOfPublications property from the MAG dataset and some NRF properties for a given researcher (ResearcherName, Institution and Rating). |
| **QueryBuilder** | Business | A highly specialized class can create a string representation of a SPARQL query, given a set of criteria. For example, Rating and Institution. | **def nrf_NameInstitutionRatingSpecialisation(p_institution, p_rating):**<br>• Returns the appropriate concatenated string of SPARQL statements needed to select name, institution and rating and specialisation from the NRF graph database<br>**def nrf_InstitutionRegionRating(p_institution, p_region, p_rating, p_groupbyregion):**<br>• 2. Returns the appropriate concatenated string of SPARQL statements needed to select name, institution and rating and region from the NRF graph database. Allows for a groupby clause to be added.<br><br>**def nrf_InstitutionRating(p_institution, p_rating, p_groupbyrating):**<br>• 3. Returns the appropriate concatenated string of SPARQL statements needed to select name, institution and rating and region from the NRF graph database. Allows for a groupby clause to be added.<br>**def nrf_RegionRatingInstitution(p_region, p_rating, p_institution, p_groupbyinstitution):**<br>• Returns the appropriate concatenated string of SPARQL statements needed to select region, institution and rating from the NRF graph database. Allows for a groupby clause to be added.<br>**def mag_CitationCount(p_groupbycitationcount):**<br>• 5. Returns the appropriate concatenated string of SPARQL statements needed to select region, institution and rating from the NRF graph database. Allows for a groupby clause to be added. |

| | | | |
|---|---|---|---|
| **QueryProcessor** | **Business** | **Receives a series of criteria and determine which endpoints need to be used to fulfill those criteria. For example, if a request is made for all institutions and the number of citations per researcher at that institution, this requires the use of the MAG endpoint. If a user specified rating and institution, this requires the use of the Stardog database.** | **def process_query(institutionflag, institution, regionflag, region, ratingflag, rating, specialisationflag, citationflag, publicationflag, nameflag, groupbytype):**<br>• **This method uses a combination of boolean flag parameters to determine which is the database that is required to fulfill a query. The correct endpoint is then chosen and the results set retrieved and piped back up to the calling method/class.** |
| **TestJSON_Reader** | NA – Unit Tests | A Unit Test class that tests whether function outputs are instances of the expected type. | **def test_JSON_Reader_full():**<br>• Tests that the get_full_integ_query() function returns the correct type of output.<br>**def test_JSON_Reader_citation():**<br>• Tests that the get_citation_integ_query() function returns the correct type of output.<br>**def test_JSON_Reader_publication():**<br>• Tests that the get_publication_integ_query() function returns the correct type of output.<br>**def test_JSON_Reader_nrf():**<br>• Tests that the get_nrf_query() function returns the correct type of output. |
| **TestQueryBuilder** | NA – Unit Tests | A class that runs a series of unit tests for the Query Builder class the ensure that the functions in the class are returning the correct type of output. | **def test_mag_CitationCount(self):**<br>• Tests that the mag_CitationCount() function returns the correct type of output<br>**def test_nrf_InstitutionRating(self):**<br>• Tests that the nrf_InstitutionRating() function returns the correct type of output<br>**def test_nrf_InstitutionRegionRating(self):** |

| | | | |
|---|---|---|---|
| | | | • Tests that the nrf_InstitutionRegionRating() function returns the correct type of output<br>**def test_nrf_NameInstitutionRatingSpecialisation(self):**<br>• Tests that the nrf_NameInstitutionRatingSpecialisation()function returns the correct type of output |

### 4.1.1 Sequence Diagrams

The Sequence Diagrams depict different sequence of activities that can occur in the AI Research Graph System. The two specific cases, among many, that have been depicted are:

- User Accessing the AI Research Graph Web Interface
- Querying and Displaying Query Results

User Accessing the AI Research Graph Web Interface

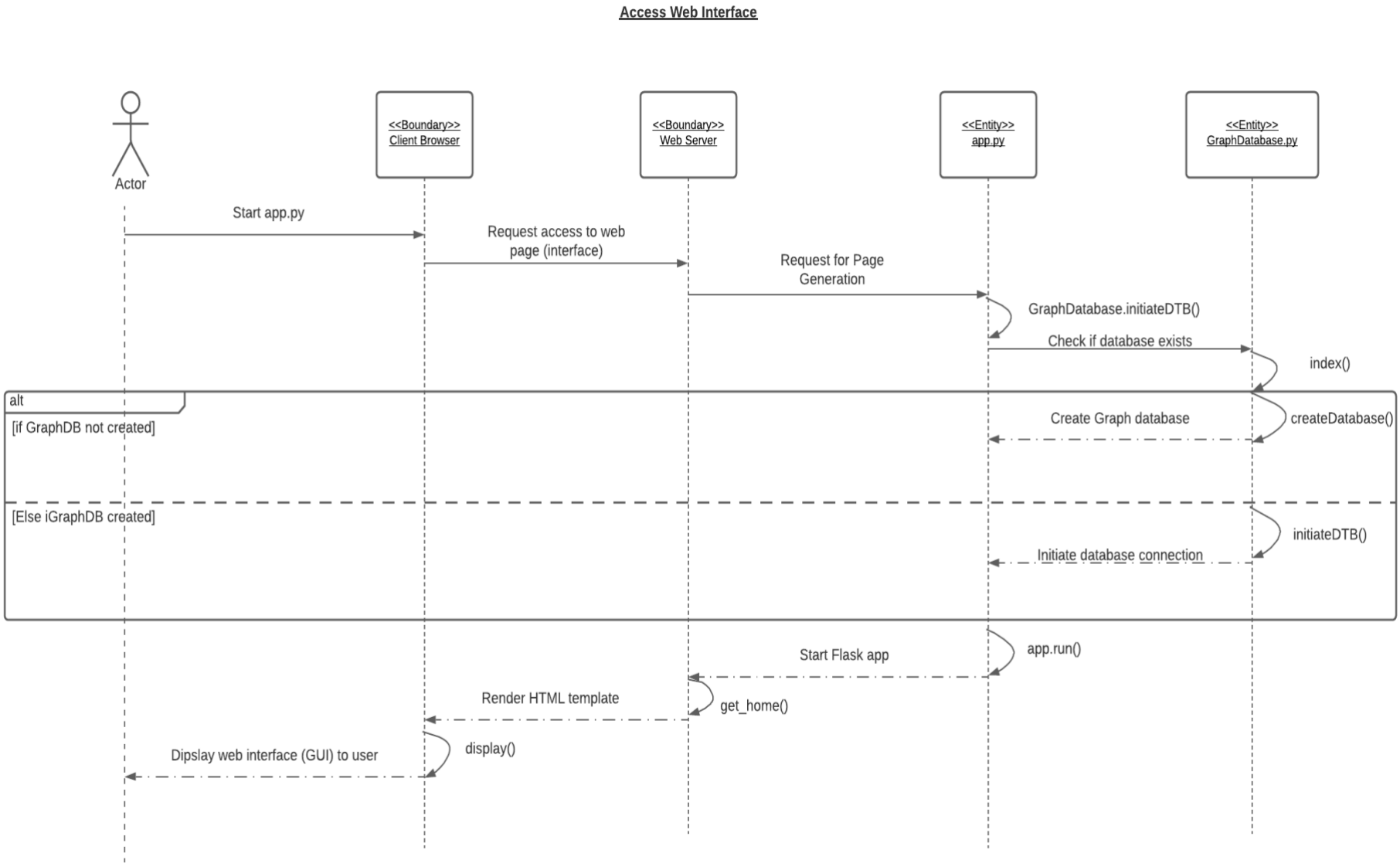
In this use case the use will start off by running or executing the python app.py class. Once running, the class will give the user a local host port on which the user can access and interact with the web interface. When the web page is accessed, there is a check perform to see if the necessary graph database has been created. If not created, the system will create it (the connection is initiated during this), otherwise if the database already exists the system will initiate the connection with the database. Thereafter, the user will be displayed the web interface which they can interact with to query and display result graphs and tables.

Querying and Displaying Query Results

This sequence diagram depicts the sequence of events and processes that occurs once a user has inputted query parameters and sends in the query to the system to be processed(by clicking the “Find Research” Button). When this button is clicked it the index () method in app.py is used to capture the user’s input from index.html’s form. Once the input is captured into the respective variables it is sent to the Query Processor class to be processed. This class send the query on further to be processed more by other classes, eventually returning result to app.py to be formatted, rendered and displayed either as a table or graph.

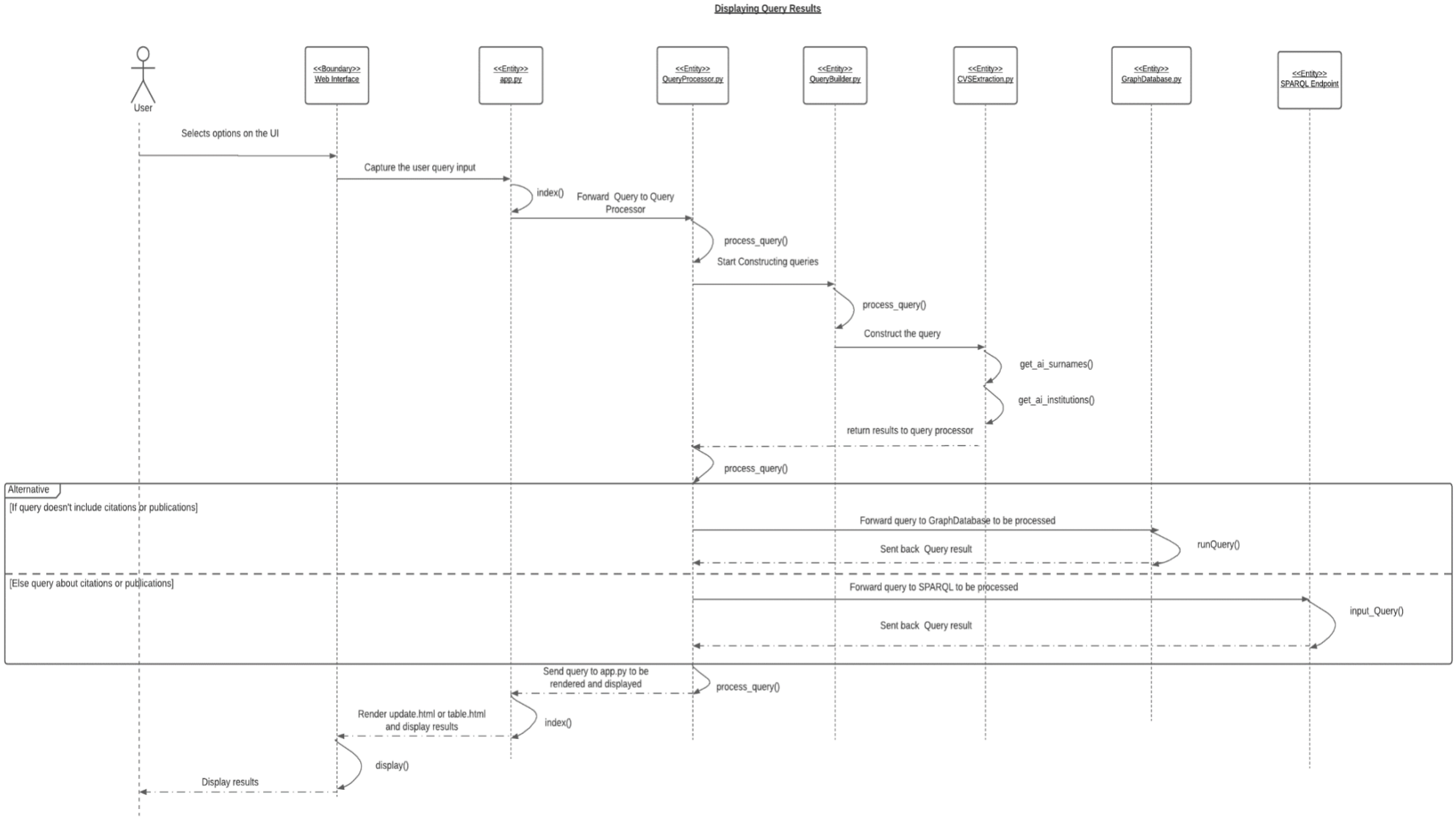
# Sequence Diagram for Accessing the Web Interface

*Figure 7 Accessing Web Interface Sequence Diagram*



# Sequence Diagram for Querying and Displaying Results

*Figure 8: Displaying query & Results Sequence Diagram*



## 4.2 Special Relationships Between Classes

### Friend Classes

A friend class can access the private and protected members of the class in which it's declared a friend. This breaks the object-oriented programming principle of information/data hiding. While designing the classes and methods, and following the Layered Architecture design pattern, the team tried to maximise cohesion, whilst minimising coupling. Following this, there are no friend classes present in the code.

### Aggregation, Composition and Inheritance

Classes are placed within different layers in order to fulfill some function. Data is passed from the top (presentation) layer, then formatted and finally passed to the bottom (database) layer. Data is then returned from this layer and passed back up, cleaned, formatted and then finally presented to the user. Given the nature of the application, and the choice of architecture, there was no need to use inheritance, aggregation or composition when designing the classes.

## 4.3 User Interface Implementation

The user interface was implemented using HTML, CSS and Flask. The main home page, index.html, is used to introduce the user to the web-interface as well as capture the user's input. By using a form component in index.html's code, we are able to capture and store the user's query input in app.py by declaring a 'POST' method for that route.

Several components have been added to index.html, so that when rendered users can input query choices through. These components include option boxes (dropdown style), checkboxes and radio buttons. There are no text-input elements on the GUI to prevent incorrect data being inputted as this can be detrimental to the system and cause errors.

Furthermore, the use of option boxes limits the input values the user can select, since users are only presented with a predefined set of valid options known to be in the database. This further, prevents errors and system crashes and the values chosen are known to be valid.

In terms of styling the HTML pages, a combination of Cascading style sheets (CSS) and Bootstrap was used. Each HTML page had a link reference to a CSS style sheet used to format the component of the page such as the colour, alignment or size. Some Bootstrap components were used to allow for easier incorporation and styling of elements with flask such as the options element and the navigation (nav) bar.

## 4.4 Additional Libraries, Microservices and Proprietary Software

### Stardog Free

This is proprietary software that is needed to store the NRF data in a graph database. The NRF data was converted into a format that was compatible with the software – in this case, it was converted into a ttl (Terse RDF Triple Language) file. Using this software allows for a graph database to be locally stored and accessed via a localhost port number.

### Pystardog

Stardog created a Python module which can be used to interface with the graph database with a python class. This module can create, drop (delete) and interact with database, using a series of provided methods. Using the module allows for the database to be started once the application is opening, and for new data to be uploaded to a database

### SPARQL Wrapper

This is a freely available external module that any developer can import in their project. This allows a developer to connect to any SPARQL endpoint. A SPARQL endpoint is an HTTP-based query service that accepts SPARQL formatted queries for sending these queries to a Relational Database Management System. In the case of our application, we made use of the MAG SPARQL endpoint, hosted at https://makg.org/sparql.

### OpenRefine

A Google-maintained software utility for deep processing and faceting of large datasets, as well as reconciliation of datasets with existing online databases. This software also facilitates the creation of functional database schemas in various formats from raw data files, for instance converting an excel file into a Turtle file to create an RDF graph database schema.

## 4.5 Schema, Ontologies and the Graph Database

Given that two distinct datasets can be queried by the web application, two database schemas are relevant to the project. Namely, the established MAG schema, which is queried through a remote SPARQL query endpoint directly to reduce required computing resources, and the NRF schema, which our team created from the NRF rated researcher excel repository. In order to construct an RDF-compliant ontology from this data, each column (node) was given a prefix URI and a property name. The result is a ttl (Turtle) format that adheres to RDF specifications and can be queried with SPARQL once loaded into a graph database endpoint (in our case a local Stardog server instance).

The RDF specification is a means of representing highly interconnected data within the ‘semantic’ web, showing deeper relationships between entities. An ontology is representation of data that shows the inter-relatedness of elements, hence the application to graph database systems.

# 5. Program Validation and Verification

## 5.1 Software Testing Plan

The development of this application followed the principles of Test-Driven Development. Each time a method or additional piece of functionality was added, a test was written for it, and then the method was tested to ensure it performs as expected. The other layers were then tested to ensure they could still use the current layer's services and the correct functionality was still provided by that layer. The team felt this was the best course of action, given that we needed to ensure that the layers integrated smoothly, and that data could flow smoothly between layers.

The application's layered architecture, as well as various classes, were taken into consideration when creating the testing plan. Initially, the back end and front-end were developed in isolation, so that a bare-bones, minimally functional prototype could be developed. Once the correct functionality was developed on both sides, they were connected. Development and testing thus follow an incremental design approach, as mentioned first. Design, then code, create tests, run tests, and then debug until all test cases have passed.

The systems architecture allows for easily splitting the classes in the different layers into groups to be tested. For classes that are classified into the business layer, Unit Tests were found to be the most appropriate. Unit tests ensured that each method with a class works as expected – this is important due to the nature of SPARQL queries and the exact syntax required to formulate them, we needed to make sure that the formulated queries were precise and error-free.

If any query has an error in it, the database will not be able to formulate a result set and the classes will raise and exception. Unit tests provided a finer-grained level of control when it came to testing, as it could test specific methods, one after another, and provide specific output as to which passed, and which failed. This control level was needed due to the sensitivity of the graph database software – e.g. some methods can only accept input of type string, otherwise the methods will raise an exception and halt the running of the program.

Random testing was used as the primary method of integration testing. Once the backend and front were connected, random testing was done to ensure that input from the UI could be correctly extracted and piped to the backend. We also need to test that the output could be compiled, cleaned and piped back up to the top level. "Cleaned" in this context refers to reformatting and removing additional data sent back from the database that would make graphing such data difficult. Heavy use was made of version control software, namely Git and GitLab, given that teammates had to work remotely (due to the pandemic). Use of this software allowed us to rollback when new features behaved unexpectedly.

Validation tests were conducted via means of frequent meetings with the client and tutor. Prototypes of the final product were shown at these meetings, in order to ensure that the correct requirements were being met. As the application was able to provide more functionality, external users (i.e. no-one on the development team) were brought in to do some use-case based tests on the system. Their experiences are documented in the appendix.

A variety of system testing was also carried out. Security was added to the database by means of a password and username (shared privately between the group members) – this provided strict access control. Once the application is started, these login credentials are automatically inputted and sent to

the database. However, should the incorrect login details be given, an exception will be thrown and the database will not be able to accept queries. Random testing, by given the incorrect login details, was done to ensure secure access. Stress testing, using random testing techniques, was also done. Specific methods in certain classes were created and then scheduled to run consecutively in order. This was done using two different machines, running different operating systems, with different hardware specifications. These informal tests showed the higher-end devices tended to execute queries faster. As the project neared completion, more user tests were done to ensure that the system remained easy to use (when fully integrated).

*Table 2: A table of tests carried out, their techniques and results*

| **Process** | **Technique** |
|---|---|
| **1. Class Testing: test methods and state behaviour of classes**<br>**(a) For all classes that fell into the Business Layer**<br>**(b) For all classes that fell into the Presentation Layer – test functionality of the UI and its components** | (a) **White-Box tests – Unit tests**<br>Due to the nature of the application, i.e. that queries need to be *perfectly* formatted, or otherwise the database will not be able to run them, *all* unit tests needed to be passed i.e. the minimum acceptance standard was a 100% pass for all tests conducted here.<br>(b) **User tests** – with Kirsten, Sabine and Rachel<br>Documentation can be found in the Appendix |
| **2. Integration Testing: test the interaction of sets of classes** | **Random Testing**<br><br>For each Sprint, additional functionality was added to the existing codebase for the project. At each increment, team members re-tested to ensure that the application was still functional, and still behaved as expected. The new functionality was then also tested.<br><br>Once thoroughly tested, with a series of random inputs, then the author added their code to the existing codebase. |
| **3. Validation Testing: test whether customer requirements are satisfied** | **Acceptance tests and Use-case based black box tests**<br><br>Met with the client during the halfway meeting to show some black-box testing using random query parameters (16 September 2021) – this was a customer acceptance test.<br><br>Use-case based black box tests was also performed by Nichola to ensure that the application functions on a different platform (i.e. Apple MAC). |

| | |
|---|---|
| **4. System Testing: test the behaviour of the system as part of a larger environment** | **Recovery, security, stress and performance tests**<br><br>**Security**<br>Security tests were performed within the GraphDatabase class to ensure that access to the database was restricted with a username and password. An incorrect username/password would not allow a connection with the database to be setup – thus queries would not be able to be run.<br><br>**Stress**<br>Stress tests were performed, whereby multiple queries were consecutively run and submitted to the database.<br>It was determined that less complex queries, which had less FILTER clauses, or involved no GROUPBY clauses, performed better.<br>Test were done on a lower-end hardware laptop, and a higher end MAC. Queries run on the MAC performed faster than on the lower-end device, however, the more complex queries still ran slower (compared to simpler ones) on the MAC.<br><br>**User Tests**<br>User tests were performed by recruiting non-expert, first-time users to access the web app and perform basic queries, as well as explore the app's functionality. Each development team member found an individual who had not been involved in development and gave them a series of instructions to engage with the app.<br>This was both to determine whether these functionalities would be robust enough for regular use, without causing bugs or errors, as well as to establish if the UI was user-friendly enough to facilitate ordinary usage. |

# 5.2 Test Cases

**Test Case Table**

*Table 3 Test Cases for AI Research Graph System*

| **Data Set and reason for its choice** | | | | **Test Cases** | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Normal Functioning (Unit Tests | | Extreme Boundary Cases | | | Invalid Data (program should not crash) | |
| **Unit Test: QueryBuilder** | | | | Passed | | NA | | | Passed | |
| **Unit Test: CSVExtraction** | | | | Passed | | N/A | | | Passed | |
| **Unit Test: JSON_Reader** | | | | Passed | | N/A | | | Passed | |
| **User Tests** | | | | | | | | | | |
| **User's Name** | Database is created/initiated | Able to interact with UI components, like dropdowns, radio buttons, etc | User reset selection | User can submit query to back-end using correct button | Able to view filters that they (user) selected | Able to view a visualization – bar graph, table – of their data | User is able to interact with the visualizations (click the columns, view the counts) | Able to navigate back to home page to make new selection | User can submit more queries | User is able to view About page |
| **Kirsten** | Passed | Passed | Passed | Passed | Passed | Passed | Passed | Passed | Passed | Passed |
| **Sabine** | Passed | Passed | Passed | Passed | Passed | Passed | Passed | Passed | Passed | Passed |
| **Rachel** | Passed | Passed | Passed | Passed | Passed | Passed | Passed | Passed | Passed | Passed |
| **Nichola (team member)** | Passed | Passed | Passed | Passed | Passed | Passed | Passed | Passed | Passed | Passed |

# 6. Conclusion

This project entailed a significant learning curve for all team members. All software frameworks explored and chosen for use were completely new to us, as was the concept of a graph database. As we moved from conceptualizing the system on paper to implementing, there were several decisive moments that made us realize we may not be equipped to meet the initial requirements specified by our client. However, by communicating clearly and proactively with the stakeholders outside the development team, we were able to refine what would actually be possible for us to achieve within the timeframe. This compromise was negotiated and ultimately approved by our client, and the final system produced can be considered a proof-of-concept to show the utility of an AI knowledge graph to all individuals interested in or participating in the emerging AI ecosystem in South Africa.

Ultimately, we feel we have mostly satisfied the re-negotiated requirements set, and provided a competent proof-of-concept, upon which a comprehensive system could be built. The greatest emphasis was placed by our client on having a local graph database instance to represent the NRF researcher dataset, while also offering users access to the MAG. The ability to make queries to both datasets through the same UI was considered crucial. We have ensured that this functionality is possible, while providing an easy-to-navigate experience for the user. Furthermore, we have integrated several forms of results visualization that depend on the type of query submitted by the user. Each query type has been associated with a particular graphic, varying from tables to bar graphs and depending on what representation gives the most intuitive feel for the results. Though, we were unable to implement measures of closeness centrality or betweenness centrality in our database query results at this stage, we feel that we have created a robust foundation on which further functionality and metrics could be implemented.

While our final application is not completely free from bugs or vulnerabilities, it performs the expected operations, and prevents users from exploiting or accidentally incurring any issues by showing query terms as a series of dropdown boxes rather than allowing the user to enter free-form queries. This reduces friction for the user by clearly showing all possible queries that can be made and preventing them from breaking the backend.

Though our focus was on backend functionality, we ensured that our UI was aesthetically-designed and improved the overall user experience, given that we would hope to attract members of the broader public to learn about the state of AI in our country, and avoid intimidating the non-expert portion of our target audience.

Finally, we were able to implement a limited number of options that demonstrate integrated queries. In other words, a user can submit a single query that produces a result set about an entity with properties from both the NRF schema we created and the MAG database. In doing so, we have proven that a project to integrate several additional sources (like AMiner or Open Graph) and deeper integration of query functionality could provide yet greater insights for the end user.

# Appendix

## Appendix 1: Test Cases and Unit Tests

### Unit Tests

*Table 4 Unit Tests*

| |
|---|
| QueryBuilder<br>Testing started at 09:54 ...<br>Launching unittests with arguments python -m unittest C:/Users/Jane/Desktop/Courses/2021/2nd SEM/CS/Capstone/Code/ai-graph/TestQueryBuilder.py -v in C:\Users\Jane\Desktop\Courses\2021\2nd SEM\CS\Capstone\Code\ai-graph<br>C:\Users\Jane\Desktop\Courses\2021\2nd SEM\CS\Capstone\Code\ai-graph<br>Ran 4 tests in 0.144s<br>OK<br>Process finished with exit code 0 |
| CSVExtraction<br>C:\Python39\python.exe "C:\Program Files\JetBrains\PyCharm 2021.2.1\plugins\python\helpers\pycharm_jb_unittest_runner.py" --target TestCSVExtraction.TestCSVExtraction<br>Testing started at 10:53 ...<br>Launching unittests with arguments python -m unittest TestCSVExtraction.TestCSVExtraction in C:\Users\ardyuv001\PycharmProjects\air-graph-capstone-project<br>C:\Users\ardyuv001\PycharmProjects\air-graph-capstone-project<br>Ran 4 tests in 0.003s<br>OK<br>Process finished with exit code 0 |
| JSON_Reader<br>C:\Python39\python.exe "C:\Program Files\JetBrains\PyCharm 2021.2.1\plugins\python\helpers\pycharm_jb_unittest_runner.py" --target TestJSON_Reader.TestJSONReader<br>Testing started at 11:18 ...<br>Launching unittests with arguments python -m unittest TestJSON_Reader.TestJSONReader in C:\Users\ardyuv001\PycharmProjects\air-graph-capstone-project<br>C:\Users\ardyuv001\PycharmProjects\air-graph-capstone-project<br>Ran 4 tests in 0.015s<br>OK<br>Process finished with exit code 0 |

# User Tests

*Table 5 User Tests*

| |
|---|
| **User test 1**<br>**User's name:** Sabine Ellis<br>**Computer literacy level:** high<br>**Knowledge of Artificial Intelligence:** low<br><br>**Results**<br>User was able to easily interact with various UI components – could select options from the dropdown, use radio buttons with ease, etc.<br>Initially, user didn't know what they were selecting e.g. "Selecting UCT in relation to what?"<br>User was able to submit a query<br>User commented and praised the clarity of the visualization.<br>User enjoyed the overall aesthetic of the site<br><br>**Items that could be addressed:**<br>Adding an additional webpage that gives the user some background about the project<br>Adding some tooltips to assist users/provide clarity on certain interface components |
| **User test 2**<br>**User's name:** Kirsten Pennington<br>**Computer literacy level:** High<br>**Knowledge of Artificial Intelligence:** Low<br><br>**Results**<br>User was able to easily interact with various UI components – could select options from the dropdown, use radio buttons with ease, etc.<br>User knew what they were selecting as they had prior knowledge on the project.<br>User was able to submit a query<br>User commented and praised the clarity of the visualization.<br>User enjoyed the overall aesthetic of the site and theme of the site<br><br>**Items that could be addressed:**<br>Adding some help tools and usability tips. |
| **User test 3**<br>**User's name:** Rachel Brodkin<br>**Computer literacy level:** High<br>**Knowledge of Artificial Intelligence:** Moderate<br><br>**Results**<br>The user interacted easily and intuitively with the UI and its core components such as radio buttons and dropdown selections provided.<br>User had to receive basic instructions as to the nature of the queries offered by each dropdown selection, and the information they would provide. Context and some background explanation made the process more seamless. |

The user was able to submit a query.
The user was able to glean the relevance of the results set to their query, and found the visual helpful in gaining an intuitive sense for the data.
The user appreciated the 'sandboxed' nature of the UI – all possible queries were already laid out in the selections, thus providing guidelines on query possibilities the user might not have independently thought of.

**Items that could be addressed:**
The user suggested that 'scroll-over' information icons with a brief guideline next to each query parameter could improve overall user experience.

# References


1. https://aiindex.stanford.edu/report/
2. https://www.nrf.ac.za/information-portal/nrf-rated-researchers
3. https://en.wikipedia.org/wiki/Graph_database
4. https://neo4j.com/developer/graph-database/

# User Manual

## Introduction

Welcome to the AIR Graph project! It's our aim here to provide you, the user, with as much insight as possible about the state of AI research in South Africa. With artificial intelligence and all its emerging sub-disciplines becoming trendier and more relevant than ever, we thought it would be valuable to provide a central source of information to all who are interested!

With so much of the conversation around artificial intelligence happening behind closed doors, or simply too loaded with technicalities to wrap one's head around, our goal is to provide a clear idea of what experts in AI are actively interested in, and the big questions challenging our brightest minds daily.

To this end, we have created a graph database of academic researchers and their affiliated institutions who have published in AI and related fields. Our web application allows you to submit queries to discover useful information about this group of researchers, their institutions, their credentials and what their work specializes in.

Since this is a web app, the resources required from your system are low, and the most important requirement from your system is that you are using the most up to date version of your browser of choice available to you! See below for additional details.

## System Requirements

<table>
<tr><td>RAM:<br>- 4GB</td><td>OS:<br>- Windows 10 or later<br>- macOSx High Sierra or later</td></tr>
<tr><td>CPU:<br>- Intel Pentium Gold G6400 or equivalent</td><td>GPU:<br>- Onboard graphics</td></tr>
<tr><td colspan="2">Web Browser:<br>- Chrome 70.x.x +<br>- Firefox 52 +<br>- Microsoft Edge Anaheim or later<br>- Safari 11.x +<br>- Stable Internet connection</td></tr>
</table>

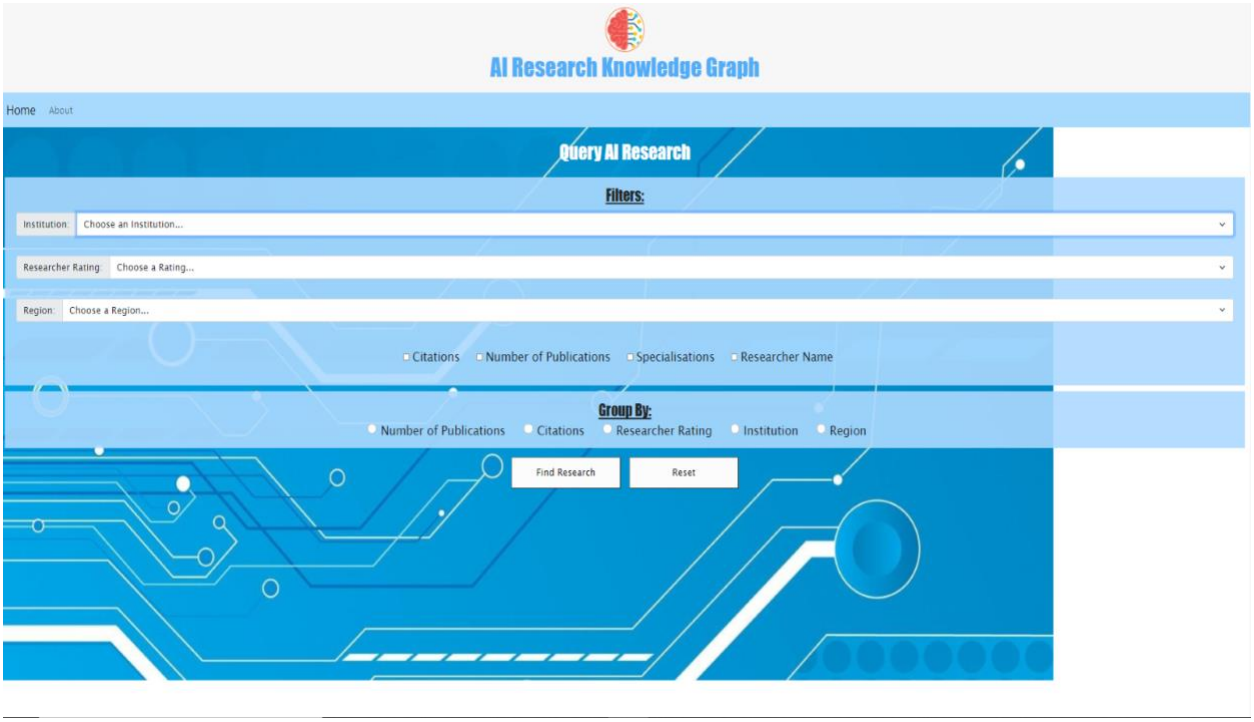
## The Datasets

With the housekeeping out the way, let's talk about the information we are giving you access to! The two powerful and comprehensive datasets at your disposal through the AIR Graph web-app, are the NRF-rated-researcher set and the Microsoft Academic Graph. The NRF is a governmental agency in South Africa that provides a rating system for publishing South African academic researchers, and records data about their research. The Microsoft Academic Graph is a mass-scale knowledge graph of academic research in numerous disciplines globally.

# Introduction to Interface

After accessing the site, the following screen should be visible:

*Figure 9*



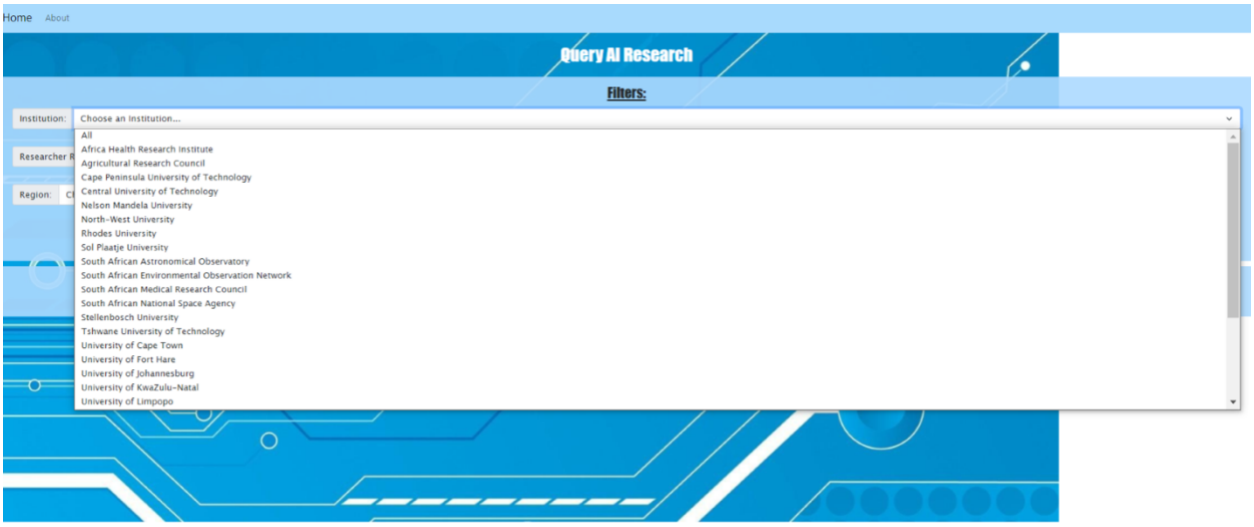

## *Filters*

The 'Filters' section consists of three dropdown boxes that will allow you, as the user, to customize the query you would like to submit to the database.

## *Institutions*

This filter provides a list of all academic institutions that are affiliated with NRF-rated researchers in South Africa. Selecting a given institution will query the number of rated researchers at that institution.

*Figure 10*




## *Researcher Rating*

This filter will specify the NRF rating for researchers that you wish to query. For example, selecting 'University of Cape Town' in the Institution filter, and 'Y' in the Researcher Rating filter will produce the number of 'Y'-rated researchers affiliated with the University of Cape Town. Note that the NRF rating works as follows:

A – Leading international researchers

B – Internationally acclaimed researchers

C – Established researchers

P – Prestigious Awards

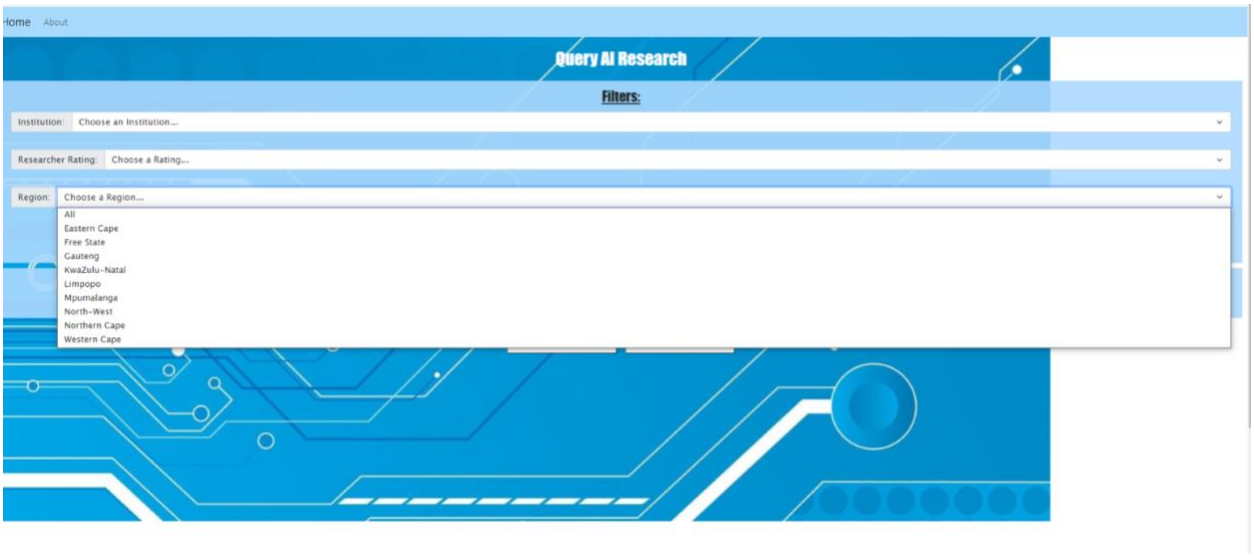
Y – Promising young researchers

*Figure 11*

## Region

This filter specifies the region a publishing researcher is based in. For example, selecting ‘All’ in the Institution filter, ‘All’ in the Researcher Rating filter and ‘Gauteng’ in the Region filter will produce a list of all NRF-rated researchers affiliated with institutions located in the Gauteng province.

*Figure 12*




### *Citations*

Selecting this tick-box will list the number of citations recorded for all researchers in the result set of your chosen query.

### *Number of Publications*

This selection returns the number of publications for each distinct researcher in a result set.

### *Specialisations*

This tick-box will return a list of all sub-specialisations of each researcher in a results set, which is a great indicator what particular problems that researcher is interested in within AI.

### *Researcher Name*

Naturally, selecting this option will ensure that the names of all researchers in the result set of your query are displayed distinctly on screen for your perusal.

### *Group By*

This function will allow you to group your query results set together logically, based on the parameter selected. For example, selecting ‘Group By: Citations’ will group all results based on the number of citations of each researcher in the set.

### *Submitting a Query*

Now let’s try submitting a sample query to show you what AIR Graph can do!

We are going to begin by querying the names and specialisations of all researchers at University of Cape Town. Therefore, we initiate this query by selecting ‘University of Cape Town’ within the Institution

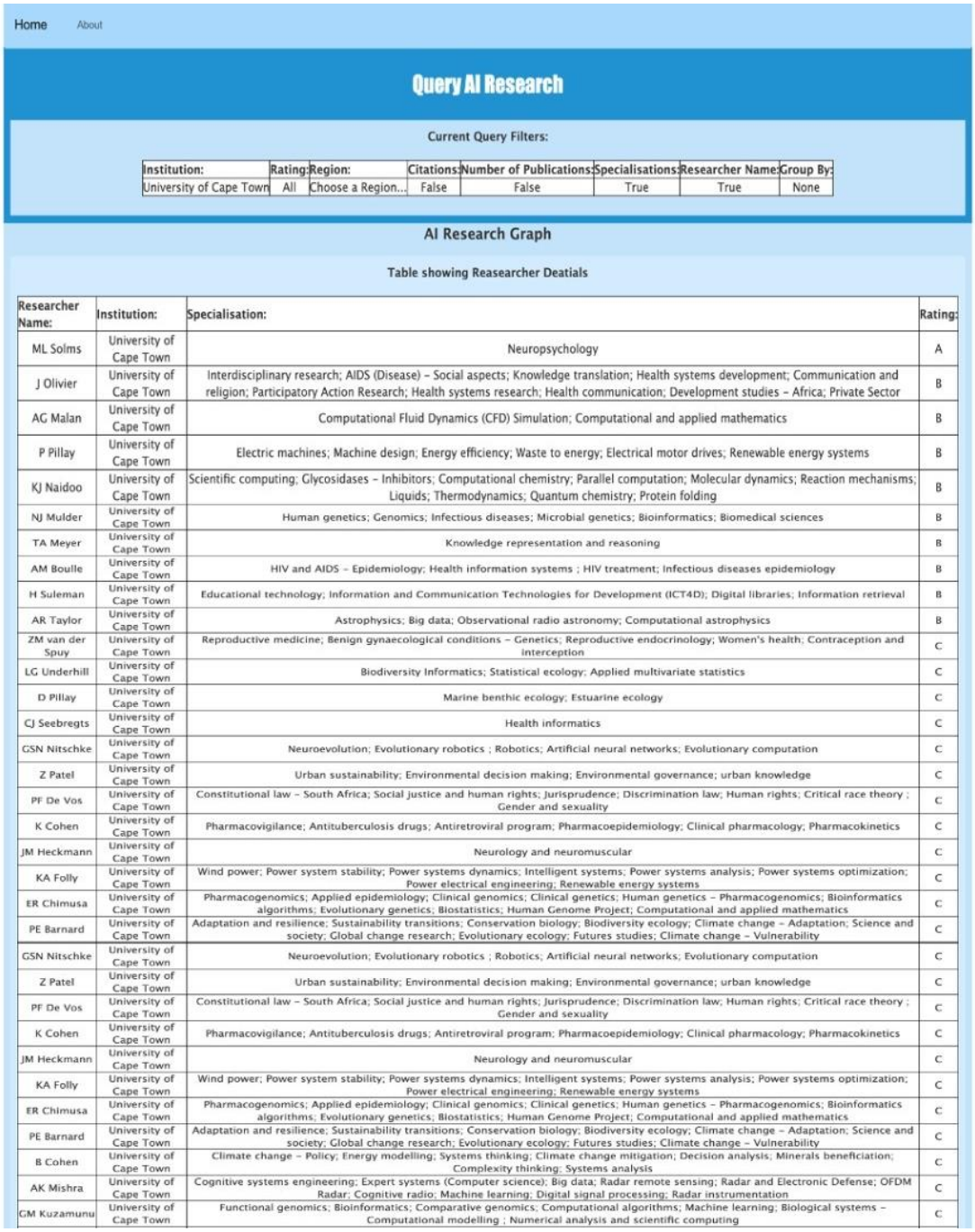
selection, ‘All’ from the Researcher Rating, and leaving the ‘Region’ selection as the default ‘Choose a Region’ field. Then we will select the ‘Specialisations’ and ‘Researcher Name’ fields. Now your screen should look like below:

*Figure 13*


Now there are two options available to you. If you selected the wrong fields for the query you are attempting to make, you can go ahead and click the ‘Reset’ button at the bottom. This will reset all query parameters for a fresh selection!

However, if you entered the query parameters correctly, then click ‘Find Research’, and wait while the query is made to the knowledge graph...

*Figure 14*




| Researcher Name: | Institution: | Specialisation: | Rating: |
|---|---|---|---|
| ML Solms | University of Cape Town | Neuropsychology | A |
| J Olivier | University of Cape Town | Interdisciplinary research; AIDS (Disease) - Social aspects; Knowledge translation; Health systems development; Communication and religion; Participatory Action Research; Health systems research; Health communication; Development studies - Africa; Private Sector | B |
| AG Malan | University of Cape Town | Computational Fluid Dynamics (CFD) Simulation; Computational and applied mathematics | B |
| P Pillay | University of Cape Town | Electric machines; Machine design; Energy efficiency; Waste to energy; Electrical motor drives; Renewable energy systems | B |
| KJ Naidoo | University of Cape Town | Scientific computing; Glycosidases - Inhibitors; Computational chemistry; Parallel computation; Molecular dynamics; Reaction mechanisms; Liquids; Thermodynamics; Quantum chemistry; Protein folding | B |
| NJ Mulder | University of Cape Town | Human genetics; Genomics; Infectious diseases; Microbial genetics; Bioinformatics; Biomedical sciences | B |
| TA Meyer | University of Cape Town | Knowledge representation and reasoning | B |
| AM Boulle | University of Cape Town | HIV and AIDS - Epidemiology; Health information systems ; HIV treatment; Infectious diseases epidemiology | B |
| H Suleman | University of Cape Town | Educational technology; Information and Communication Technologies for Development (ICT4D); Digital libraries; Information retrieval | B |
| AR Taylor | University of Cape Town | Astrophysics; Big data; Observational radio astronomy; Computational astrophysics | B |
| ZM van der Spuy | University of Cape Town | Reproductive medicine; Benign gynaecological conditions - Genetics; Reproductive endocrinology; Women's health; Contraception and interception | C |
| LG Underhill | University of Cape Town | Biodiversity Informatics; Statistical ecology; Applied multivariate statistics | C |
| D Pillay | University of Cape Town | Marine benthic ecology; Estuarine ecology | C |
| CJ Seebregts | University of Cape Town | Health informatics | C |
| GSN Nitschke | University of Cape Town | Neuroevolution; Evolutionary robotics ; Robotics; Artificial neural networks; Evolutionary computation | C |
| Z Patel | University of Cape Town | Urban sustainability; Environmental decision making; Environmental governance; urban knowledge | C |
| PF De Vos | University of Cape Town | Constitutional law - South Africa; Social justice and human rights; Jurisprudence; Discrimination law; Human rights; Critical race theory ; Gender and sexuality | C |
| K Cohen | University of Cape Town | Pharmacovigilance; Antituberculosis drugs; Antiretroviral program; Pharmacoepidemiology; Clinical pharmacology; Pharmacokinetics | C |
| JM Heckmann | University of Cape Town | Neurology and neuromuscular | C |
| KA Folly | University of Cape Town | Wind power; Power system stability; Power systems dynamics; Intelligent systems; Power systems analysis; Power systems optimization; Power electrical engineering; Renewable energy systems | C |
| ER Chimusa | University of Cape Town | Pharmacogenomics; Applied epidemiology; Clinical genomics; Clinical genetics; Human genetics - Pharmacogenomics; Bioinformatics algorithms; Evolutionary genetics; Biostatistics; Human Genome Project; Computational and applied mathematics | C |
| PE Barnard | University of Cape Town | Adaptation and resilience; Sustainability transitions; Conservation biology; Biodiversity ecology; Climate change - Adaptation; Science and society; Global change research; Evolutionary ecology; Futures studies; Climate change - Vulnerability | C |
| GSN Nitschke | University of Cape Town | Neuroevolution; Evolutionary robotics ; Robotics; Artificial neural networks; Evolutionary computation | C |
| Z Patel | University of Cape Town | Urban sustainability; Environmental decision making; Environmental governance; urban knowledge | C |
| PF De Vos | University of Cape Town | Constitutional law - South Africa; Social justice and human rights; Jurisprudence; Discrimination law; Human rights; Critical race theory ; Gender and sexuality | C |
| K Cohen | University of Cape Town | Pharmacovigilance; Antituberculosis drugs; Antiretroviral program; Pharmacoepidemiology; Clinical pharmacology; Pharmacokinetics | C |
| JM Heckmann | University of Cape Town | Neurology and neuromuscular | C |
| KA Folly | University of Cape Town | Wind power; Power system stability; Power systems dynamics; Intelligent systems; Power systems analysis; Power systems optimization; Power electrical engineering; Renewable energy systems | C |
| ER Chimusa | University of Cape Town | Pharmacogenomics; Applied epidemiology; Clinical genomics; Clinical genetics; Human genetics - Pharmacogenomics; Bioinformatics algorithms; Evolutionary genetics; Biostatistics; Human Genome Project; Computational and applied mathematics | C |
| PE Barnard | University of Cape Town | Adaptation and resilience; Sustainability transitions; Conservation biology; Biodiversity ecology; Climate change - Adaptation; Science and society; Global change research; Evolutionary ecology; Futures studies; Climate change - Vulnerability | C |
| B Cohen | University of Cape Town | Climate change - Policy; Energy modelling; Systems thinking; Climate change mitigation; Decision analysis; Minerals beneficiation; Complexity thinking; Systems analysis | C |
| AK Mishra | University of Cape Town | Cognitive systems engineering; Expert systems (Computer science); Big data; Radar remote sensing; Radar and Electronic Defense; OFDM Radar; Cognitive radio; Machine learning; Digital signal processing; Radar instrumentation | C |
| GM Kuzamunu | University of Cape Town | Functional genomics; Bioinformatics; Comparative genomics; Computational algorithms; Machine learning; Biological systems - Computational modelling ; Numerical analysis and scientific computing | C |

Here we have the results to your query, with all the researchers from University of Cape Town being listed in a table alongside their specialisations and their NRF rating! This should give you a good idea of all publishing researchers currently affiliated with UCT. Now to make another query, simply hit the 'Reset' button at the bottom of the screen again, which will simply return us to the home page with no selected query parameters again.

Let's make one more query, just to show you how powerful the visualization tool can be! We will now construct a query for the number of NRF-rated researchers at all institutions in the graph database. Select 'All' under Institution, 'All' under Researcher Rating, and select the 'Researcher Rating' tick-box under the 'Group By:' heading below the dropdown boxes. Your screen should now look like so:

*Figure 15*


Now click the 'Find Research' button, and off we go!

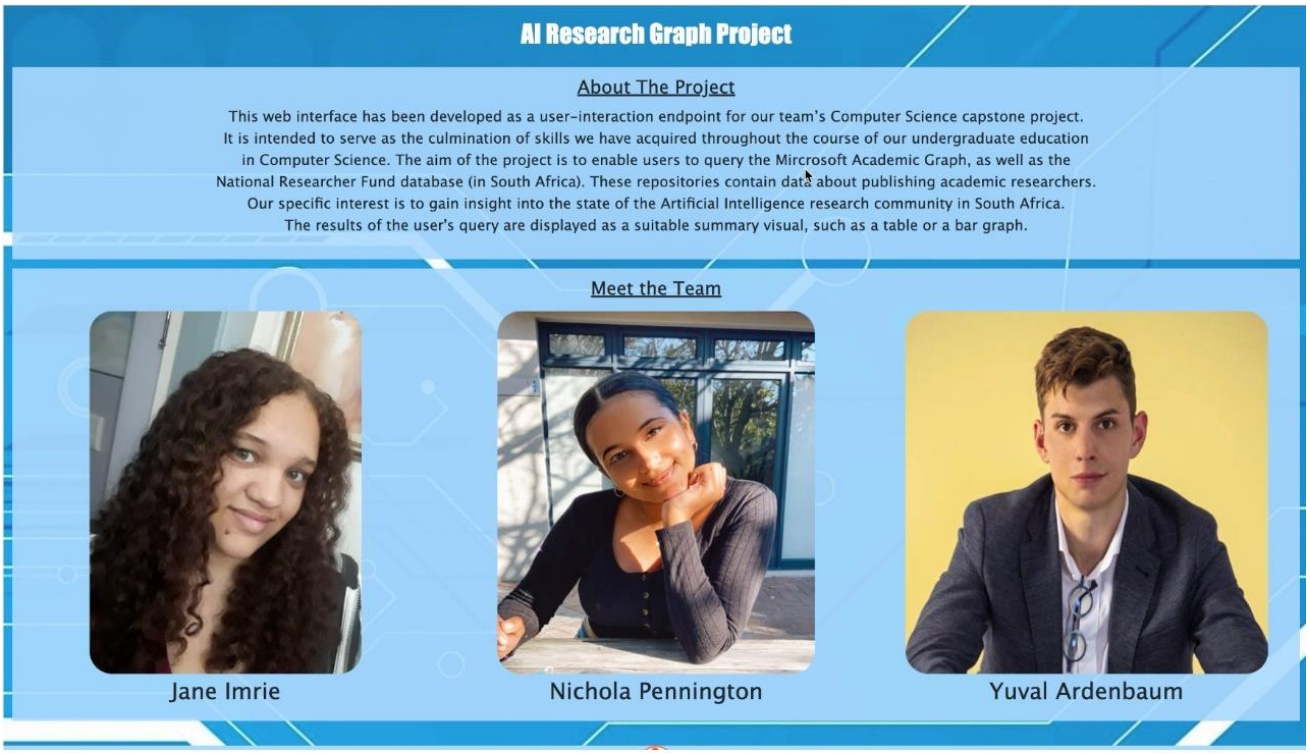
The results produced should look like below:

*Figure 16*


This fabulous graphic shows each academic institution that has NRF-rated researchers, with each bar showing the number of researchers of a particular rating. Some institutions appear multiple times, but each bar represents all researchers with a distinct rating (e.g. one bar might show 17 researchers with a C-rating, while another could show 5 researchers with an A-rating for the same institution). Another way to reset the app is to click the 'Home' button in the upper left corner, which once again returns us to the default home page with no query parameters selected.

Finally, from the home page, you can select 'About' in the upper left corner to learn a little more about our project, as well as check out each of our team members' LinkedIn profiles by clicking on their picture!

*Figure 17*




Thank you for exploring the South African AI research community along with us! We hope to add greater functionality to the AIR Graph project in future, and we look forward to the watching what emerges from this exciting field in the future!