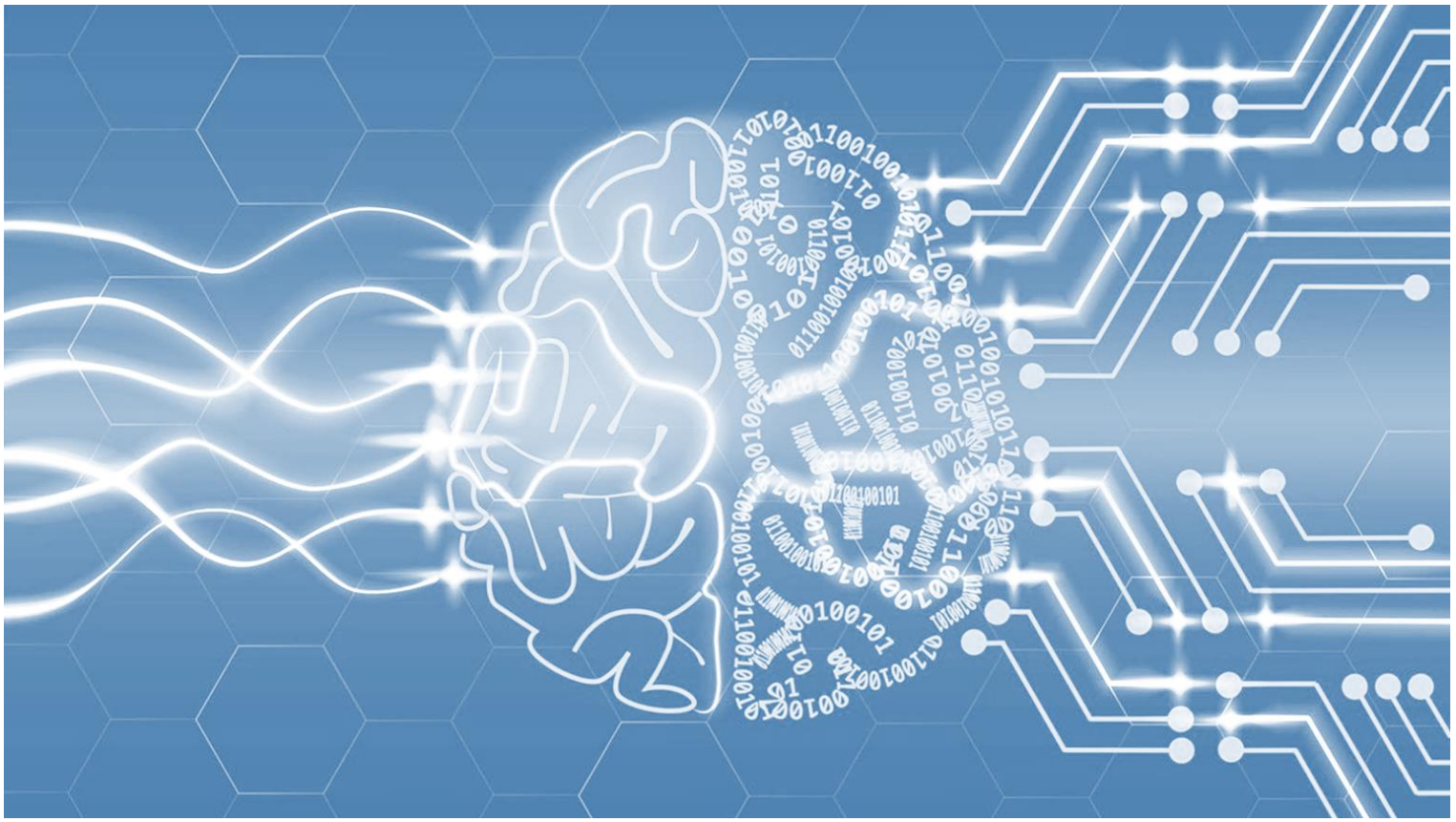


AI WEB APPLICATION

CSC3003S

Kauthar Orrie, Shreeya Khoosal and Rauseenah Upadhey

ORRKAU001, KHSSHR001 and UPDRAU001

Table of Contents

Introduction

Artificial intelligence is a rapidly growing scientific field that is constantly pushing the envelope with innovation and research. The AI research community in South Africa spans a wide range of institutions and concerns a plethora of sub-disciplines and research topics. In light of this, the primary purpose of this project was to develop a platform through which users can meaningfully engage with a centralised repository of AI related data. By performing basic search queries and viewing trends and statistics within the data, users can quickly and efficiently glean an understanding of the state of AI research in South Africa. Multiple data sources were assimilated in order to produce an integrated network of AI data whose structure was optimised for accessibility.

The development of this project followed an agile approach, as multiple iterations of the application were cycled through, as the project sprints progressed. Given that the development environment and project specification were relatively new terrain for all team members, the project progress fluctuated around our ability to leverage and adapt to these new frameworks. For this reason, it was necessary to adopt an agile methodology which catered for this level of flexibility.

The project was initiated with requirements gathering, in which we attempted to achieve consensus around the scope with the project supervisor (Professor Deshen Moodley), however this set of requirements was constantly adjusted in order to remain feasible. Before the project development officially commenced, it was necessary to carry out in-depth research regarding the suitability and compatibility of different database management systems and frameworks. Despite conducting this research, our group did need to pivot strategies after finding our original choice of frameworks (the MERN stack) to be inviable. The development was guided by a test-driven approach, as we optimised our MVP to meet the requirements of the first demonstration and continued to align and measure our development in accordance with the expected functionality.

The evolutionary prototype developed for this demonstration, made use of Spring Boot which was carried through to the end of the project deadline. The entirety of the project implementation was guided by the overarching objective of creating a user friendly, informative web application that could effectively report on and make sense of the wealth of AI research data available within a South African context.

Abstract

The culmination of this project was a web-based application that allows users to query a repository of AI related data. Within this application, end-users can filter the data by making

use of a predefined set of categories or by entering custom text data (such as a researcher's surname).

This functionality was developed in addition to another core component of the platform- visualising trends within the data. By creating a dynamic set of bar graphs, pie charts and line graphs, users are able to navigate through a comprehensive overview of trends within the AI research environment in a highly accessible and engaging manner.

This was enabled by deploying a number of pre-existing JavaScript libraries, as well as a DBMS framework such as Spring Boot in order to decrease the friction of working with an underlying database. As a result of this, the majority of the platform was developed using java, JavaScript and html. The data used for the project was extracted from two primary sources (the NRF and Web of Science) in order to produce an integrated network of AI research.

2.2 Requirements Captured

Requirements elicitation was an ongoing and feedback responsive process. The initial requirements for the project were developed from the assignment brief, and were later expanded on during client meetings, in which the scope was refined. Finally, the requirements were adjusted in accordance with the rate of team progress as well as the feedback received from preliminary demonstrations of the working software.

Functional Requirements

1. Users should be able to perform queries of both the Researcher and Article data set, with results returned and displayed to the user via the interface
2. Users should be able to view a variety of trends and statistics within the AI research community in a visually accessible and dynamic manner
3. The platform should track a comprehensive set of metrics that describe the size and nature of the researcher community, these metrics should be accessible to the user
4. The user should be able to view individual entities in the data such as both researchers and articles
5. End users of the platform should be able to filter/sort the data by selecting from a predefined drop-down menu

Non-Functional Requirements

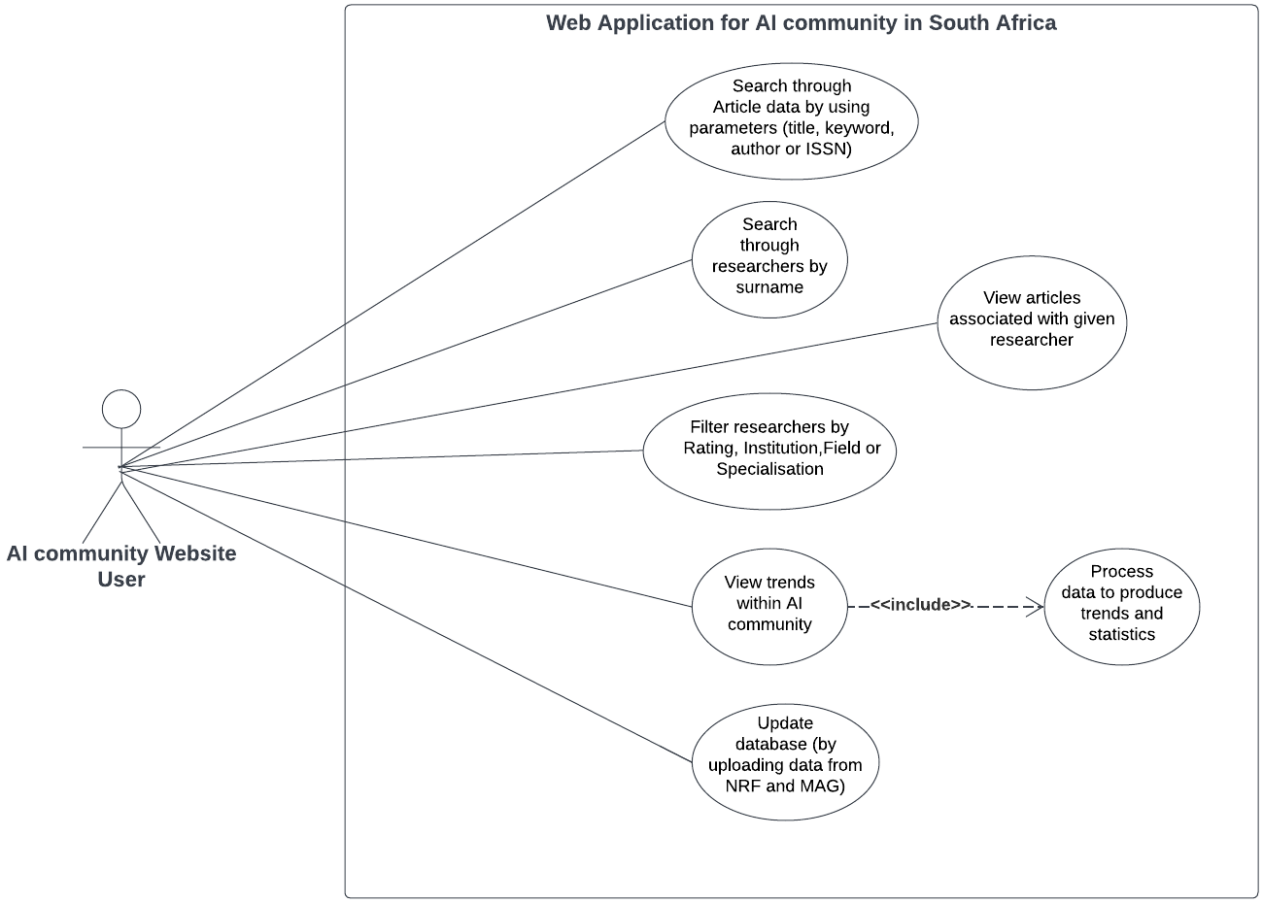
6. Updates to the data should be facilitated in order to maintain the reliability and relevance of the information displayed on the platform
7. Queries to the dataset should function with an average response time of 200 milliseconds to 1 second in order to prevent the user from experiencing lags/delays
8. The logical layer of functionality should be abstracted and separated from the presentation layer. This prevents malicious edits to the code by minimising/constraining

the capacity of the user to create errors which strengthens the overall security and safety of the platform

Usability Requirements

9. All webpages should have clear and consistent navigation as well as instructions that explain the different design elements and how to interact with them
10. The implementation of the functionality should be hidden from the user through the use of a comprehensive interface in order to increase the level of usability
11. Data validation with explicit feedback and error messages should be provided in order to assist the user in making correct use of the platform

Use Case Diagram of Implemented System Functionality



Use Case Narratives

1. Search through article data

   Actor: AI platform user

The user is able to search through the database of articles by selecting the search field type (such as Author name or ISSN number) from a dropdown menu and entering the relevant text String. Once the user presses submit, this text will be processed as an input parameter and a search query will be performed. The relevant records will be returned from the backend and displayed on the user interface. If no match has been found, or the user enters an invalid input into the search bar, a corresponding error message will be displayed.

2. **Filter Researcher data**

   Actor: AI platform user

The user is able to filter the data by selecting from a pre-specified set of categories within a drop-down menu. Once the user has selected their chosen filter values and pressed “Load filters”, a backend query will be performed. This query will return the relevant Researcher data and produce a refined data set. The table on the front end will be updated in order to reflect this. Given that this functionality does not require the user to enter any form of textual data, there is a narrower margin for error and no alternative path needs to be specified.

3. **Access Articles Associated with a Given Researcher**

   Actor: AI Platform user

By clicking on a record within the researchers table, users are able to view the articles/publications associated with a given researcher. Once a researcher has been selected, the user will be redirected to a table of their articles. Individual records within this table can additionally be selected, in order to view more information regarding a specific publication. If there are no articles associated with a researcher, the Articles table will be left blank. Given that no user input is required, an alternative sequence of events need not be outlined.

4. **Update Data that the Platform Runs On**

   Actor: AI Platform user

Once the application is run, upon start-up, the user is prompted via the terminal regarding whether they would like to use the “default” data or update the file. If “d” for default is entered, the application runs using the August 2022 NRF Researcher data (the most recent file at the time of development). However, if “U” for update is entered, the user is prompted to enter the date of the new file they would like to run the platform on. If there is a match with the filename that is inputted by the user, the statistics and trends will dynamically adjust according to the new data that is provided. Alternatively, if the user inputs neither “D” nor “U”, an error message is displayed, and they are prompted to re-enter a valid option.

Analysis of Artefacts

Over the course of the project, several deliverables were developed in order to track the project progress as well as to plan the design of the system and its scope.

The OneDrive folder linked to below contains these artefacts from Stage 1 – 3.

[Project Artefacts]

2.3 Design Overview

2.3.1 Justification for design architecture

The main objective of creating a web application for the AI community was to provide users with an easy way of viewing and retrieving information about the community. When deciding on an architecture for our system, we thought it would be best to follow a layered architecture. A layered architecture design allows for easy communication between classes. Not only did we require easy communication between classes, but we also needed a way to create a database, retrieve data from it and display or manipulate the data in a sizeable and user-friendly manner on a user interface. As shown in figure 1 we chose to implement the Spring Boot architecture as it best suited our application needs.

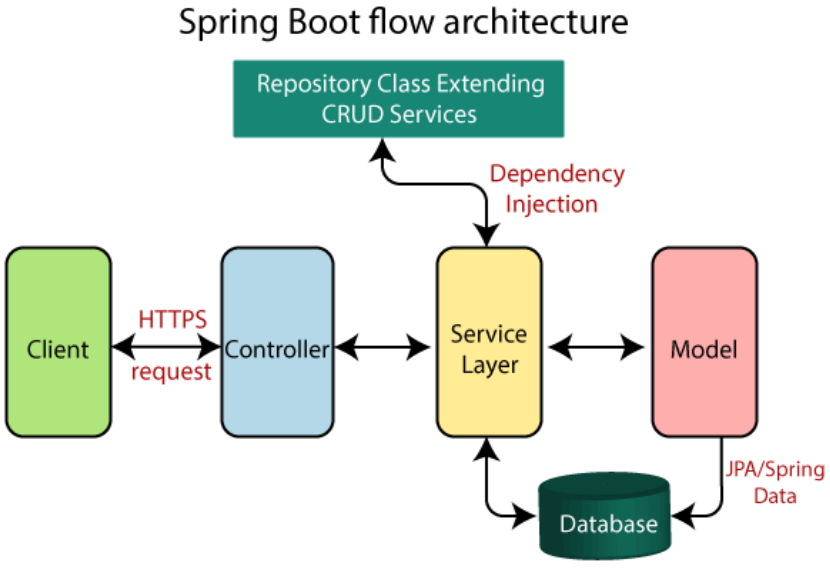
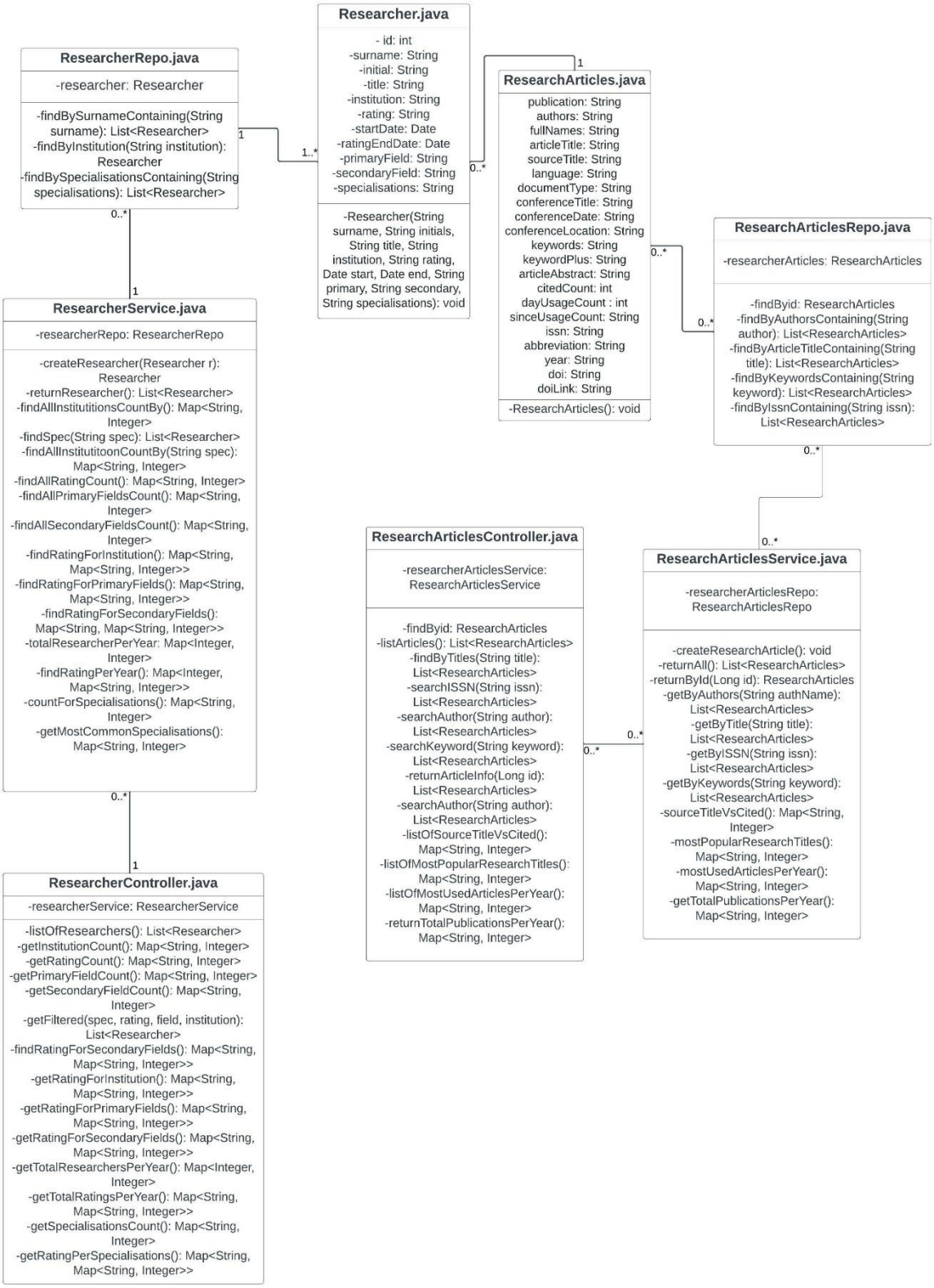


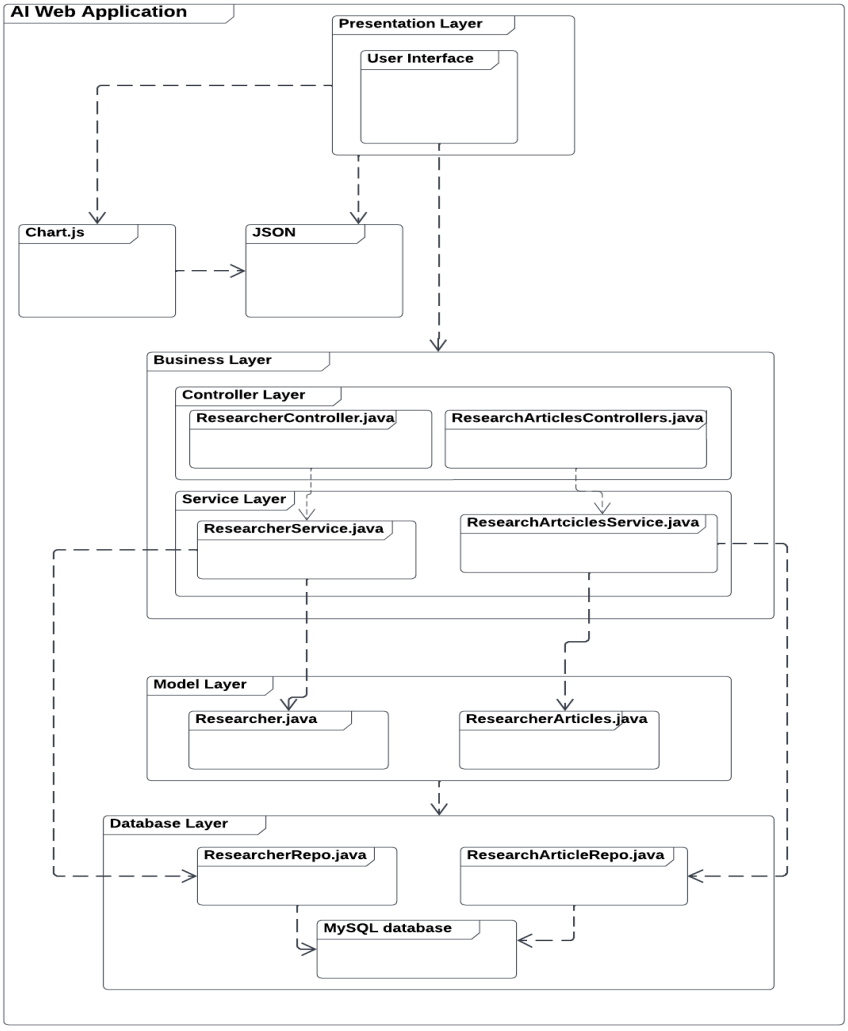
Figure 1

2.3.2 Class diagram



Above is an analysis class diagram showing the breakdown of the classes within the system.

2.3.3 System Architecture



2.3.4 Architecture overview

The development of this project followed the same design as the Spring Boot architecture. Spring boot follows a layered architecture for communication between classes. The system has four layers: Presentation, Business, Persistence and Data Layer. All layers communicate with each other as shown in Figure 2.2

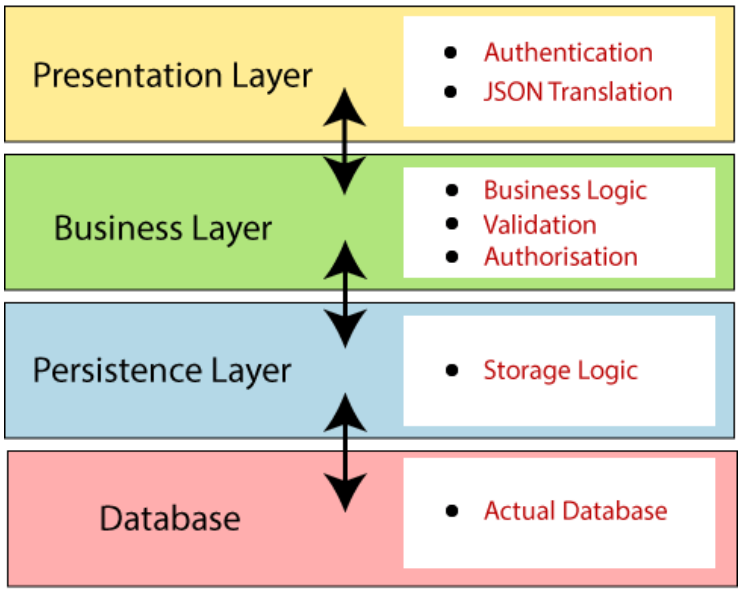


Figure 2.2

Due to the nature of the web application, we needed a pipeline from backend to frontend. The most challenging part of the development process was integrating the frontend with the backend. Using a layered architecture provided us with the ability to send and receive data easily since all layers follow a pipeline and linear way of communicating with each other.

Presentation Layer:

- The user interacts with the presentation layer to perform queries, view graphs and filter the data from the NRF data and web of science data.
- The presentation layer consists of all HTML, CSS and JavaScript files which renders buttons, filters and graphs
- The library Chart.js is used to create all graphs
- The presentation layer interacts with the business layer to fetch and receive all necessary data.

Business Layer

- The business layer is where all business logic takes place.
- All services and controller classes reside in this layer
- The business layer receives data from the MySQL database and performs queries on them.
- The business layer receives queries from the presentation layer and then outputs the selected query back to the presentation layer
- The business layer acts as a pipeline from the user interface to the database and vice versa

Database Layer

- The database layer is where all the data is being stored within the database
- The database saves two relational database tables, one for the NRF data called Researchers and another for the web of science data called ResearchArticles and these tables are created from a model classes which creates objects to insert into the database

2.3.5 Algorithms and the data organisation used

The development of the project required the use of databases therefore storing the NRF data and web of science data within a database. MySQL was used to create relational databases.

2.4 Implementation

User Interface Implementation

The front end is implemented using multiple HTML files. The HTML files make use of two CSS style sheets. The mystyle.css file contains the main CSS code and phone.css style sheet contains containers and sizes adjusted by size. However, the page setup selected suited all screens so a low level of adjustment/addition to the code was required. All HTML files make use of various JavaScript files. These files handle the majority of the front end logic code in which all front-end processing takes place. JS files fetch the data from the backend. The JS files gather queries from the user interface and fetch the relevant data from the backend. After the data is fetched it is processed and displayed to the user.

General Class Structure and Significant Methods			
Class Name	**Associated Layer**	**Function of the Class**	**Significant Methods**
ResearcherService.java	Business Layer	This class uses the repository interface to interact with the database. Methods within this class grabs hold of the data needed from the database. Methods within this class calculates, manipulates and performs queries with the data in the database.	- **createResearcher():** uses the repository classes to add the Researcher to the database - **returnResearcher():** return all researchers in the database - **findSurname():** a method to return a list of researchers by a specific surname - **findAllInstitutionsCount():** returns all researchers from the database and calculates a count for all institutions [similar methods for rating, primary fields and secondary field queries] - **totalResearhersPerYear()**: calculates the total number of researchers per year. This method is used for graphs on the interface
ResearcherController.java	Business Layer	This class acts as a "middleman" between the user interface (frontend) and the backend. The interface interacts with this class when it fetches data from the backend. The data being sent from the backend to the frontend is controlled by the controller class. When the frontend requests data the controller class interacts with	- **listOfResearchers():** returns a list of all researchers within the database. - **getName():** returns the researcher by the requested surname - **getInstitution():** returns the count per institution [similar methods were created for rating, primary and secondary fields and specialisations] - **getFiltered():** accepts queries from the frontend and then returns the filtered data - **getTotalRatingsPerYear():** returns data for the graph queries (similar methods for all graphs can be found within this layer)

		the service class to perform requested query. All methods within this class have an associated URL. This is used so that the frontend can interact with the backend.	
ResearcherRepo.jav	Database Layer	This is an interface, and it extends the JPA Repository implemented by the Spring Boot architecture. The interface has built in functions which makes it easy to perform SQL select queries. This interface is the only part of the system that can directly interact with the database to insert and receive data.	- **findSurnameContaining():** return a list of researchers containing a given surname - **findBySpecialisations():** return a list of researcher objects with specified specialisation
Researcher.java	Model	This class is used to create researcher objects with specified parameters	- Constructor - Getters and setters for all necessary parameters
trend.js	Presentation Layer	Display article graphs and support article interface page functionality needed.	**changeScreen()** This is to set all graphs visibility to hidden except the two that are meant to be shown on page 1 and set the label to page1/4. **rightswap()**

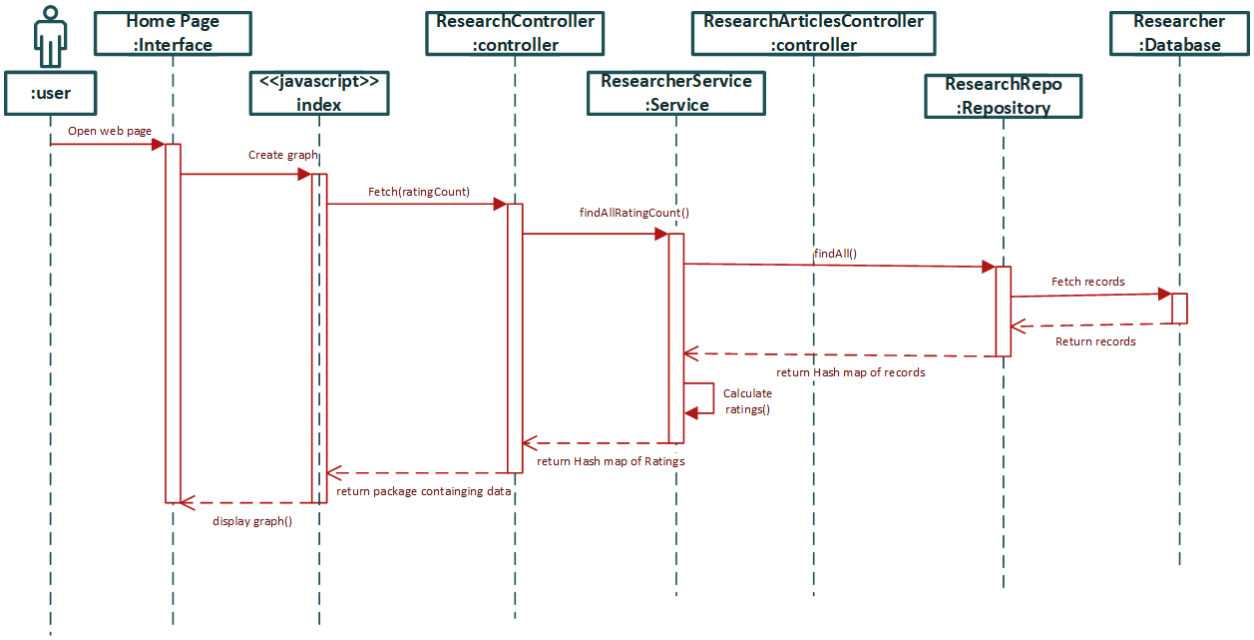
			This is to set the current two graphs being displayed to hidden and to change the next two graphs that need to be shown to visible as well as increment the page count on the label. **leftswap()** This is to set the current two graphs being displayed to hidden and to change the previous two graphs that need to be shown to visible as well as decrement the page count on the label.
ViewAllArticles.js	Presentation Layer	Display articles table and support article interface page functionality needed.	**ViewAll()** This reloads the table to display all the records in the database on the server side via a fetch command. Inside this method is another method that makes all the rows clickable and when clicks it fetches the id variable for that row and then prompts the articleInfo page to open and passes the id to it through a greetingValue variable. **LoadAll()** This method is called and then loads the dropdown for the search bar and then alls the viewAll() method so that the records are displayed when the page is loaded. **addObj(array,object)** This method adds the objects to the array. **DeleteRows()** This would empty the table so that it can be repopulated with the desired data.
graph.js	Presentation Layer	Display researcher graphs and support researcher interface page functionality needed	multiple fetch data functions that automatically load on the webpage opening up. This data is either used to populate a table or to generate a graph. **changeScreen()** This method is used to swap what is displayed on the screen between either two graphs or one table.
ViewAllResearchers.js	Presentation Layer	Display researchers table and support researcher interface page functionality needed.	async function loadSpec() This fetches the specialisations and adds it to an array and then loops through it adding it to the dropdown so that it can be displayed.

			There are two other methods exactly the same except the one is named loadInc() and it handles the institutions, the other is named loadField() and it handles loading all the primary fields. ViewAll() This resets the dropdowns to its default variable ALl and reloads the table to display all the records in the database on the server side via a fetch command. Inside this method is another method that makes all the rows clickable and when clicks it fetches the needed variables for that row and then prompts the researchInfo page to open and passes the needed variables to it through a greetingValue variable. sendRequest() This fetches the selected variable in the dropdowns and then combines them into a url to fetch all matching records in the database on the server side. Inside this method is another method that makes all the rows clickable and when clicks it fetches the needed variables for that row and then prompts the researchInfo page to open and passes the needed variables to it through a greetingValue variable. LoadAll() This method is called and then loads the dropdowns this means calling the loadSpec() method, loadIns(), loadField(), viewAll() method so that the records are displayed when the page is loaded. then it then loops through the rating array and adds its objects to the dropdown to be displayed. - **DeleteRows():** This would empty the table so that it can be repopulated with the desired data.

2.7 Sequence Diagram

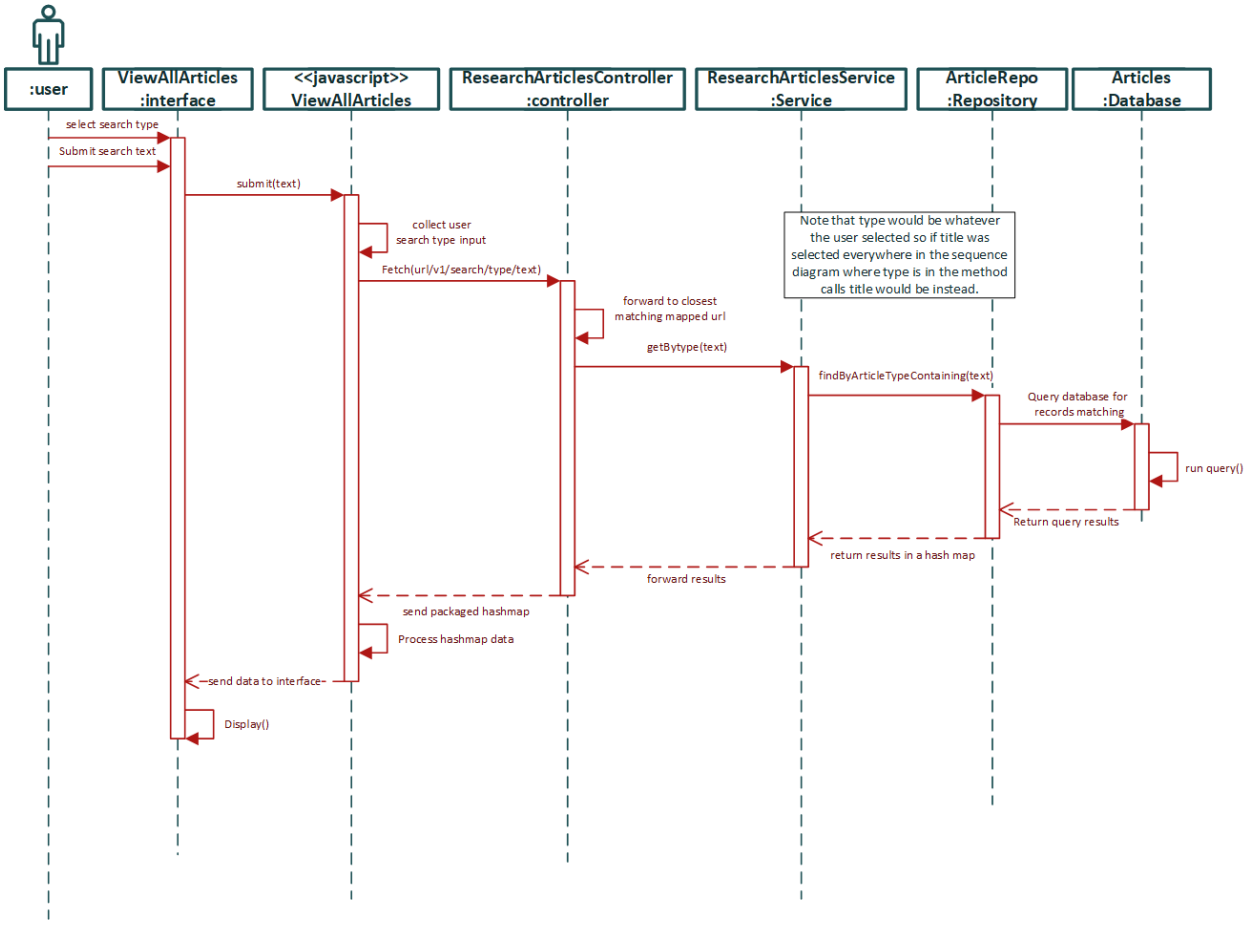
This is a sequence diagram representing the sequence of steps that occur when the homepage is opened.

Diagram 1: Home page Sequence Diagram



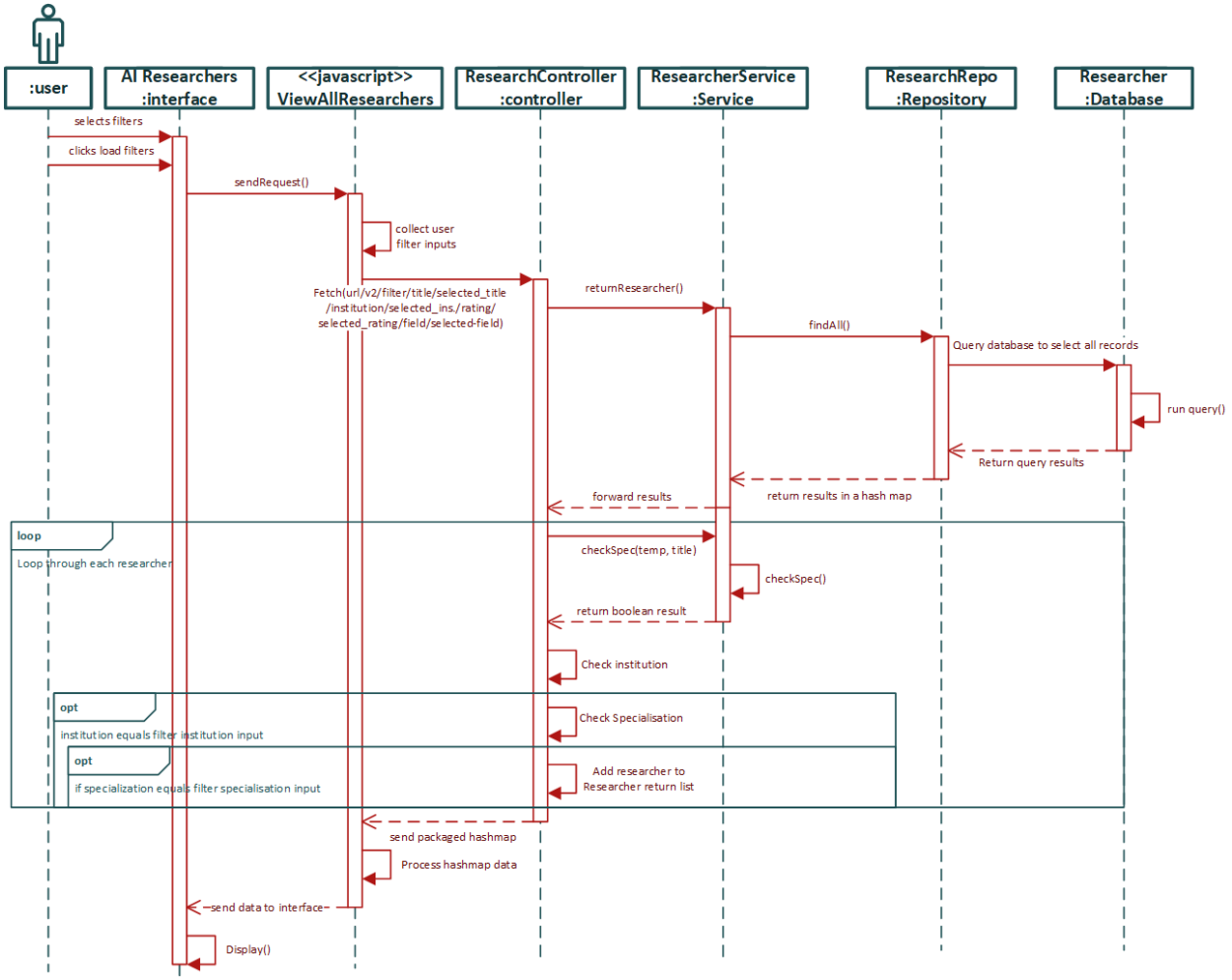
This is a diagram displaying the sequence that occurs when a user searches on the view all articles page.

Diagram 2: View All Articles search Sequence Diagram



This is a sequence diagram representing the sequence that takes place when the user selects dropdowns and clicked the load filter buttons.

Diagram 3: Load Filter Sequence Diagram



2.5 Program Validation and Verification

The development of this project resulted in using a Test-Driven Development approach and the development of the project was done in an Agile manner. This meant that with each method or feature that was added, tests were rolled out with every iteration. Each component of functionality was implemented independently focusing on completing frontend and backend all together before moving on to a new feature. When creating a feature (for example a search query) the backend functionality was implemented first. Before pushing the backend to the frontend, we decided to test our functionality first by writing up simple tests like printing the results to the console/terminal or using the Spring Boot implementations to ensure the backend was working as expected. Once backend testing was completed, we integrated the front and backend. Upon successful integration, random testing took place on the user

interface to ensure that the correct data was being fetched by the frontend. This method of testing was consistent throughout the project development. All team members agreed that this would be the best way of development for the expected requirements and required functionalities.

The structure of the classes allowed for white box testing to take place. For each class in the system architecture, unit testing occurred. As shown in Appendix A, Figure 1.1 the tests were conducted by testing the repository interfaces to ensure that data can be inserted into the database and fetched from the database. Tests were also conducted on the Service and Controller classes to ensure that queries can be performed successfully from the database. Given the system architecture, white box testing was the most ideal form of class testing because it provided a better view of the inner workings of the system, how the components integrated with one another, as well as the ability to examine the overall structure of the database queries. This additionally enabled us to reveal bugs within the system that were not picked up by the random testing during development.

The frontend will fetch data from the backend and the backend will perform the query and send the data back to the frontend. Should the data from the frontend not be recognised by the backend, no query will be performed, and empty data will be sent back to the frontend which results in an error and will be displayed on the user interface. The structure of the classes ensures that the frontend does not interact directly with the database. This is done by having a Controller class acting as a "middleman" while interacting with the Service classes which are the only classes able to interact with the database. This creates an additional layer of security and prevents SQL injections from the frontend.

We set up frequent meetings with our supervisor (tutor) to ensure that user requirements were met. This was done by developing an evolutionary prototype, checking in with the tutor halfway during the project and engaging in client meetings to ensure that the scope was correct and core functionality was understood.

Table 1: *Summary Testing Plan.*

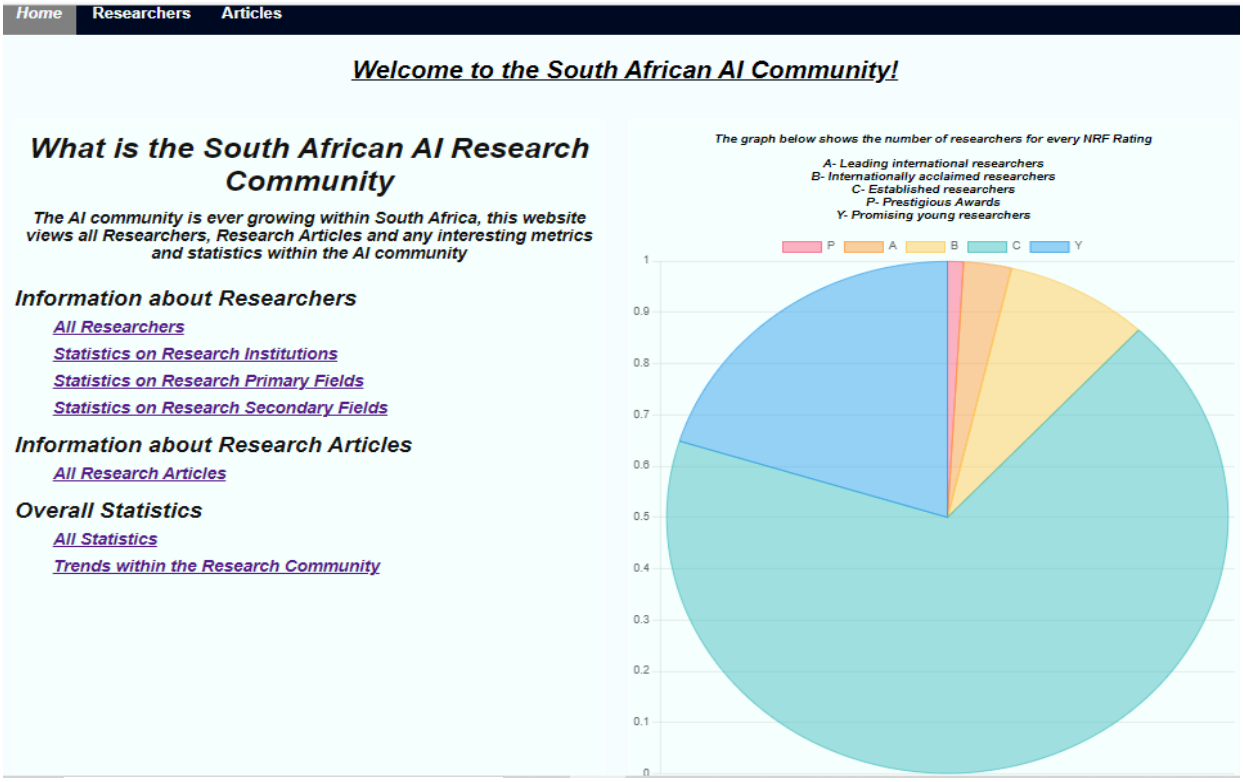
Process	Technique
1. **Class Testing:** - All classes within the Service and Model classes were tested to ensure queries are being dealt with correctly - Repo interface was tested to ensure that data was stored and retrieved from the database correctly	Random and White-Box Tests - White box testing: All classes were tested as units because of the structure and interaction with the database. All testing took place via IntelliJ IDE and the proof can be found in Appendix A.
2. **Integration Testing:** test the interaction of set of classes	Random and Behavioural Testing - Random testing was done frequently with every functionality implementation. For every query implemented the frontend was

	extensively tested to ensure that data was being fetched correctly and that errors were dealt with accordingly. - Behavioural Testing was best suited for the backend functionality. Checking the behaviour of the backend ensured that the correct behaviour was outputted to easily integrate it with the frontend
3. **Validation Testing:** test whether customer requirements are satisfied	Use-case based black box and Acceptance tests - Acceptance testing was done by having frequent meetings with the client and tutor to ensure that user requirements and the core functionality was met.
4. **System Testing:** test the behaviour of the system as part of the larger environment	Recovery, security, stress and performance tests

2.7 User Manual

Home page

When you first open the web app you are faced with the *Home* page. The home page displays links to view All Researchers, All Research Articles and interesting trends within the South African AI community. You can also navigate to these pages using the navigation bar at the top of the page. The graph being displayed shows the overall NRF rated researchers for each NRF rating in a pie chart form.



Researchers Tab

We'll start off with the first tab, "Researchers". When clicking the tab, you get a drop-down list.

First on the drop-down list is "All Researchers" and the screenshot below shows what is displayed when choosing the drop-down item "All Researchers".

On this page you can perform multiple queries:

Search user by Surname

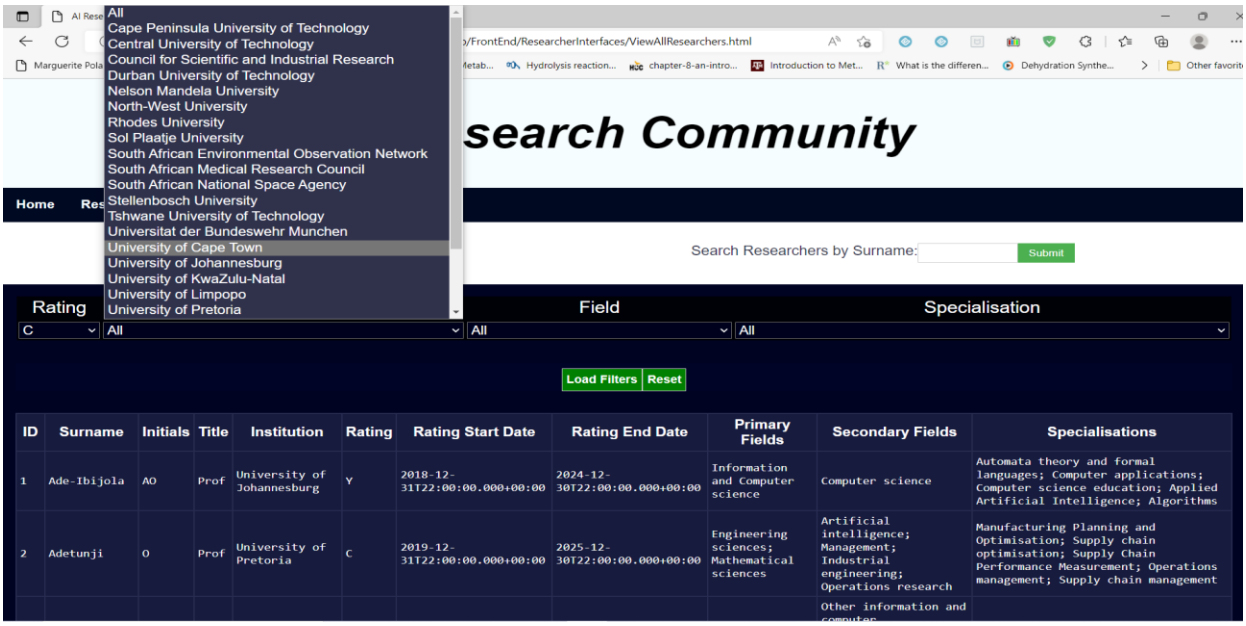
With this query, the user can perform a search query. When entering the surname of the interested researcher, the researcher's information will populate the table.

Reset

When selecting the “Reset” button you can go back to showing all the researchers in the table.

Filter the table by Rating, Institution, Field and Specialisations

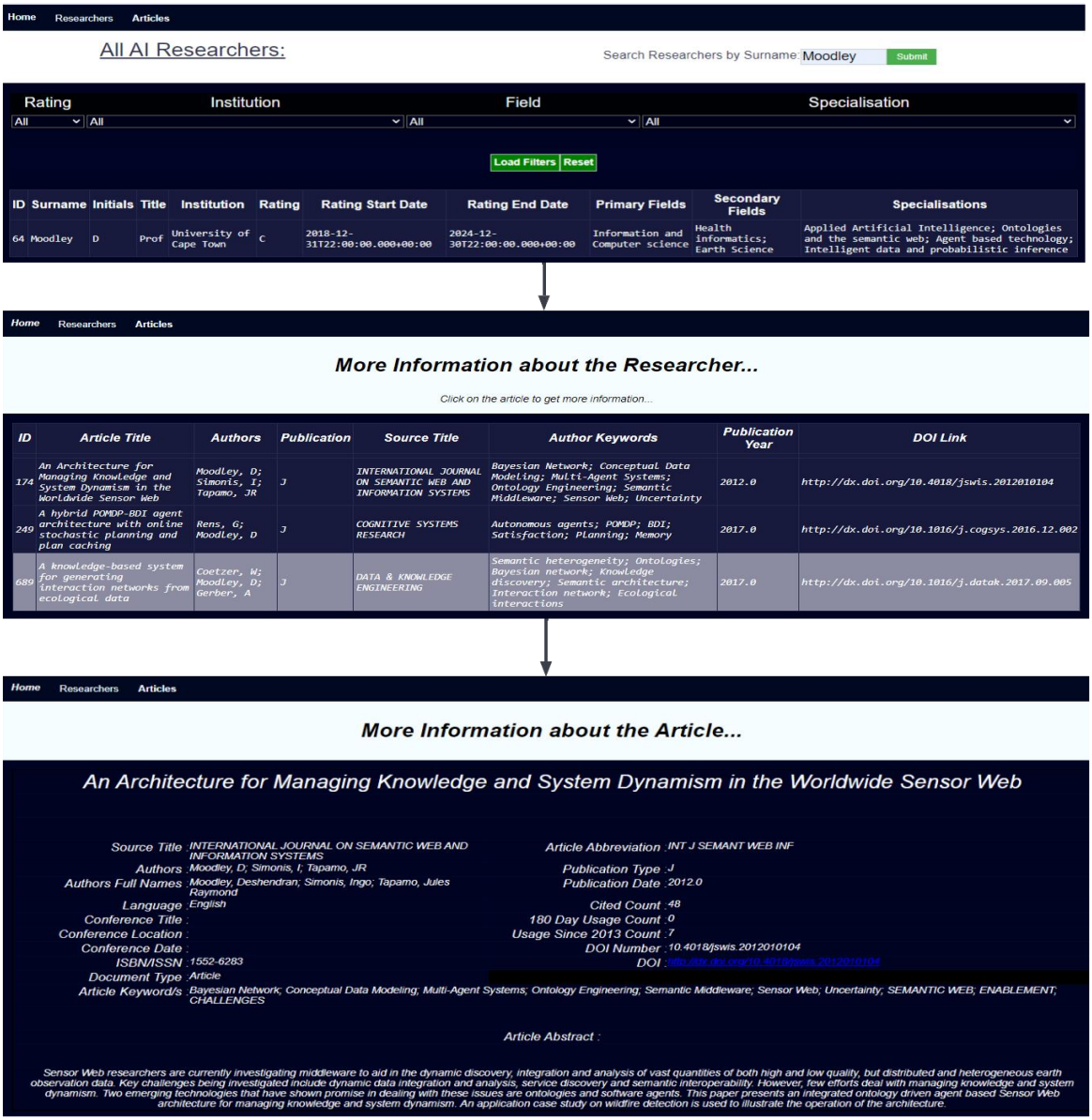
With the filter you can select multiple queries to filter out the table. An example of what the filters looks like is shown below. When pressing the “Load Filters” button the filtered information will display in the table.



More information on selected researcher

Below is a screenshot that shows the sequence of events when searching a researcher name. If you want to know more about the research articles for the selected researcher, you can click on the researcher record in the table and all the researchers' articles will be displayed in tabular format under “More Information about the Researcher” page. You are also able to select a research article and get more information about the selected article such as Conference Title, Conference data, ISSN, etc.

Note: This functionality is available for filtered tables as well as all the researchers in the table.



AI Research Articles Tab

When clicking the Articles tab on the navigation bar, you get a drop-down list.

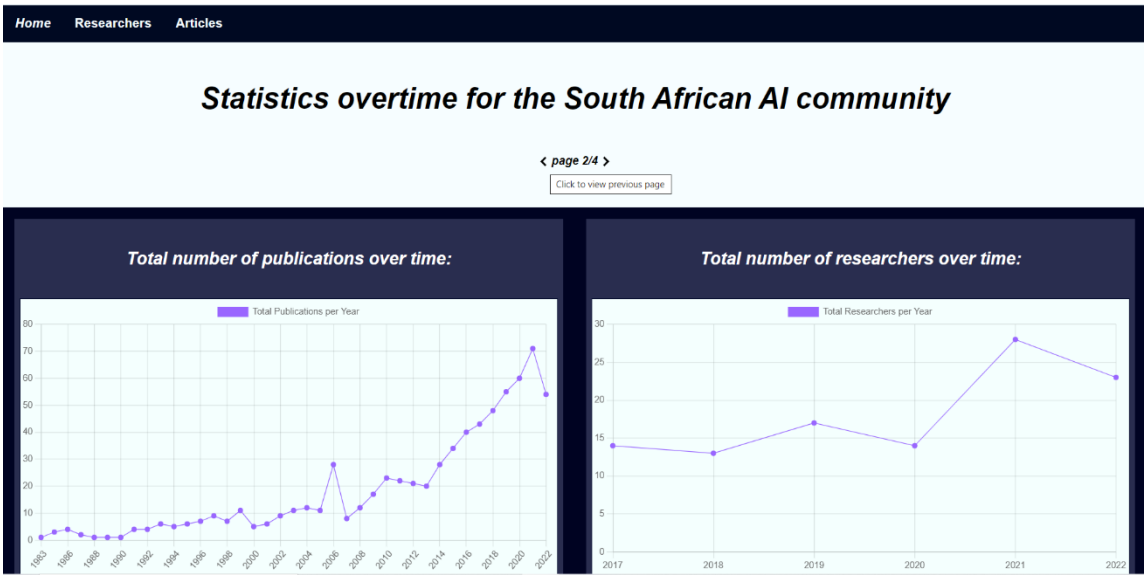
First on the drop-down list is “All Articles” and the screenshot below shows what is displayed when choosing the drop-down item “All Articles”. The articles page follows the same struture as the

Researcher Page.

ID	Article Title	Authors	Publication	Source Title	Author Keywords	Publication Year	DOI Link
1	Mammogram content-based image retrieval based on malignancy classification	Chikamai, K; Viriri, S; Tapamo, JR	J	INTELLIGENT DATA ANALYSIS	Mammography; microcalcifications; image processing; CBIR; machine learning; computer aided diagnosis	2017.0	http://dx.doi.org/10.3233/IDA-163101
2	A Monte Carlo simulation based approach for stochastic semi-infinite mathematical programming problems	Luhandjula, MK	J	INTERNATIONAL JOURNAL OF UNCERTAINTY FUZZINESS AND KNOWLEDGE-BASED SYSTEMS	stochastic; semi-infinite program; Monte Carlo simulation	2007.0	http://dx.doi.org/10.1142/S0218488507004662
3	Sentence analysis using a concept lattice	Serutla, L; Kourie, D	S	MACHINE TRANSLATION AND THE INFORMATION SOUP	concept lattice; machine learning; natural language processing; machine translation; example-based machine translation	1998.0	

Research Articles trends Tab

Under the Articles tab in the navigation bar, you are also able to view trends within the data. Multiple graphs are displayed, and you can page through to view more graphs as shown below.



Conclusion

This project was an incredibly instructive experience as all team members were forced to rapidly upskill in order to meet the project demands. Consistently, we struggled with the challenge of setting realistic goals and frequently had to scale back on our expectations as we encountered unforeseen setbacks within the implementation phase of the project. For this reason, the project was a valuable exercise in both the project management and software development arenas.

Throughout the course of the project, we had to redistill and realign on what was regarded as core functionality. However, by the end of the project, we were able to deliver on the majority of our initial scope. Users are able to perform full front-end to back-end queries of the data and are similarly able to access and view dynamic representations of the researcher and article data. In addition to this, community metrics were incorporated into the graphs that were created and extensive thought went into designing around the most meaningful variables to track. Once this core functionality had been developed, it was then possible for us to extend the scope of the project to “nice-to-have” features. This consisted of enabling updates to the data which contributed an additional layer of flexibility and relevance to the web application that was developed.

Each feature within the project specification was developed from the front-end and back-end concurrently, and thorough testing was carried out before the implementation of the feature was considered complete. A variety of both class and integration tests were completed that exhibit the performance of the system in a range of contexts, and with different input parameters. While the testing was not completely exhaustive, the application passed the majority of the tests and delivered the expected behaviour.

Compounding this, the system structure was created with the objective of being robust and invulnerable to security threats such as SQL injection attacks. The use of the Spring Boot framework assisted us in abstracting the core functionality of the database and separating this from the user's view. In addition to this, where possible, the design of the interface limits user input, however it is acknowledged that the system is not entirely invulnerable.

Another crucial aspect of the project which affected its success was the team dynamic. The skill sets of our team members were highly complementary and the team dynamic boosted our overall productivity. Members were cognisant of/accommodating towards one another’s schedules and thus we were able to maintain a relatively consistent level of progress throughout the project implementation. Despite this, there were additional/ambitious features that remained out of scope for this iteration of the project. However by and large we felt we were able to deliver on the objective of creating an interactive, informative application that allows end-users to meaningfully engage with AI related data within a South African context.

Appendix A

Tests conducted of the ResearcherService class.