

**ARLA – Alumni Registration and Linking Application**

**B.Sc. (Hons) in Software Development**

**Student: Ciaran Roche**

**Project Partner: Pavel Antonov**

**Supervisor: Damien Costello**

**Submission Date:**

**Table of Contents**

1. Introduction
2. Methodology
3. Technology Review
4. System Design
5. System Evaluation
6. Conclusion
7. References
8. Appendices

**Introduction**

The project that was undertaken is called “Alumni Registration and Linking Application” also known as Arla. What is Arla? It is an application that was originally intended to be developed on behalf of the college itself. The original idea was that GMIT was to use it on their website. Arla is where users can sign up and login to the application using their google email address. They will then be able to register their details such as their name, the course that they studied in GMIT and the year in which they studied it, their interests and where they are now. The users of the application, once logged in, can then view the connections to a course on the graph page. On this page it will have every user that has linked to said course. Users will then be able to click on a person they wish to connect to. When doing this a popup will appear and a button which will bring them to a messenger page of the user they wish to contact. From there the users can now message each other.

The application is designed so that it is very user friendly to people of all ages and all experiences of using technology. For example, it was intended to make sure that someone who has never used a computer before in their life has as much ease of using the application as someone who uses it every day. The user will also have as much flexibility as they like when it comes to the information that people are able to access about them on the website. The user may decide they don’t want to have any connections and not upload any information to the application. Perhaps they just want to have a browse on the application without actually connecting to people.

It is a good project to develop at level 8 as there is so many different ways the project can be implemented with, given enough time, many cool features can also be implemented. Another reason as to why it was a good project for a final year project is because of the complexity and workload of it. These technologies used were extremely complex and took a lot of research and reading of documentation to fully understand and implement.

The front end of the application was developed using React. The graphical part of the application was implemented using D3.js. There were some new and exciting libraries that were implemented into the application also. These were chatengine.io and react popup as well as some libraries that were intended for use but could not properly use for one reason or another. These technologies and libraries will be discussed some more in the technology review section of the document. The application is fully responsive on the front end that can only be accessed if the user has logged in. It is also set up on Heroku so that it can be accessed publicly. Below at the end of the introduction are links to the application hosted on Heroku along with the GitHub repositories where the work was stored.

The backend had many issues in the development process which will be discussed more in the methodology section and outline the course of action that my supervisors advised to take. The backend connected to the new technology Neo4J which is a graph-based database where all the information passed from the front end was to be stored. The graph on the React front end was essentially the Neo4J database superimposed using D3.

There were many goals set at the beginning of the project before development initially took place, and it was hoped that as many as possible if not all of them were implemented and completed by the end of the project life cycle. The goals that were set were as follows:

The first goal of the project was to create a fully responsive web application that is hosted on a middleware site such as Heroku or AWS that allows for users to connect to each other and message each other via a messenger styled page on the application.

The second goal was to accommodate all people who wish to use their website, make it extremely simple, efficient and user friendly for both a user with no previous experience of using technology and someone who has high levels of experience in using technology.

The next goal in developing the project was to allow the user to enter as much or as little detail as they wish and to decide how much of this information would be visible to other users of the application. Along with this, it was also a goal to make sure that the user can adjust or remove any details they may have inadvertently added to their profile.

Another goal that was set for the project was to create a way in which users can create their own groups and contact each other. Perhaps along with this create their own little mini graph and show all the people within the group. They could then have a group chat on the messenger page for example.

The next goal that was planned was to gain a greater understanding of the technologies that were going to be used in development and to learn some new and exciting technologies such as D3 and Neo4J in the process.

Along with learning new Technologies there were some new things that have not been thought on the course before and it was exciting and intriguing to see can it be done for this project. These were: to create a google login application and understand the mechanics of what is behind this and understand how to implement this for future projects. Another of these is to have a dynamic home page for the user and understanding how to implement this. In other words, when the user logs in, I want it to show only their individual details on their home page, so that it would be different for every user of the application. The next goal was to understand how to draw graphs and essentially understand the basic concepts that were needed for this application using D3. It was also preferable to improve any testing skills throughout the duration and at the completion of programming the application.

As well as software development goals, there were some goals that involved project management. The project cycle was ideal to improve any ability to use the Jira application and learn how it can be incorporated it into the project. This is so that it improves efficiency when planning and implementing a project during a project life cycle. Other applications that were ideal to incorporate fully and improve usage of were OneNote for documentation and GitHub for the programming aspect of the project.

Finally, in terms of goals that were set out for the project, it was ideal and preferable to improve soft skills. Improving any skills such as teamwork and communication by having regular meetings with the supervisor and teammate. This is ideal to constantly keep people up to date with the project as well as making sure it is on the right track.

Methodology:

The methodology chapter in this paper is where the projects methodology will be discussed and essentially how the project was carried out and implemented. The development approach that was used will be discussed under topics such as, how it faired and an evaluation of the style that was adopted. As well as this the planning stages of the project will be discussed. An in-depth look will be taken at what technologies that were used in the early days to set a plan out for the project, how this developed as time progressed and how changes were dealt with. A thorough and in depth look at the technologies that were used for documentation, planning and designing of the project. The tools that were used to develop the project will be discussed and analysed, for example, GitHub. The research that was conducted in the planning stages of the project will be thoroughly explained, dissected and analysed. Also, a delve into the weekly meetings and communication throughout the project will take place.

Technology Review:

The technology review section is where an in-depth look will take place at all the different technologies that were used in the development process of the project. Some of these technologies will include react, D3, chatengine.io and all the different libraries that were either implemented successfully or implemented unsuccessfully for one reason or another. A brief look will be taken as to why said technologies were not implemented and what alternatives can be or were used instead. Each technology will be analysed to the fullest extent.

System Design:

The system design section is the part in which a detailed explanation of the overall system architecture will be given. This is essentially the HOW of the project. It is where the knowledge gained from research is implemented. Also, it will look at each aspect of the application and give a detailed overview and in-depth analysis of different components of the system and how they work together.

System Evaluation:

The system evaluation section is where an evaluation of the system is conducted. For example, what is good, what is bad, what needs more work and what could be done to make this even better. An evaluation of goals will be made, where they met, if not why?

Finally, a conclusion to the project will be given and an overall opinion of how the project went. Analysis will take place to see what can be done to improve the project as well as performances of members. An evaluation will be made on what was learned during the project cycle. Below are links to the front-end GitHub repository, the backend GitHub repository and the link to the Heroku hosted application.

[**https://github.com/CiaranRoche203/Arla-App-FrontEnd**](https://github.com/CiaranRoche203/Arla-App-FrontEnd)

[**https://github.com/CiaranRoche203/Arla-App-Backend**](https://github.com/CiaranRoche203/Arla-App-Backend)

[**https://arl-application.herokuapp.com/**](https://arl-application.herokuapp.com/)

**Methodology**

There was a lot of planning, research and decision making that went into the early days approach to the project. The first and most important aspect was to set out a stable plan that the team members can follow. The first step in planning that took place before all else was to research heavily and thoroughly. Research is a common practise that took place in the duration of the project.

The first stage of research involved which type of methodology suited the project and the team members best. There was a general idea of which methodology suited the project best before any research was done but it was important that this was done correctly as it would benefit the planning and overall flow of the project. The 2 cycles that were researched were the Waterfall and Agile.

After much deliberation between team members and with advice from the supervisor, it was decided that an Agile approach benefitted the project the most. The reason for this is due to Agiles’ flexibility. It also was a preferred methodology to follow as it allowed for consistent testing of the application as well as receiving consistent feedback. Waterfall could and probably would have caused a lot of issues overall if it were the methodology that was pursued. The reasoning for not taking on the waterfall model is that it is too rigid. In order to move on to the next step, i.e., design to implementation, the design phase must be one hundred per cent complete.

It was then decided that Agile would be the development methodology to be followed. Agile allowed for the project to be developed while minimizing risk when adding new functionality. Some of the risks that it minimized where bugs and changing of requirements. The consistency of testing the application that would take place with Agile would cause less issues in the long run and would not delay development as much as waterfall potentially could. The flexibility was a key factor for our choice of Agile. The ability to make continuous changes and additions to the application coupled with the fact that these new additions and changes would be tested consistently immediately after implementation was a major factor in the decision.

The next stage in the development of the project was to plan. The plan is crucial as it sets a basis of what needs to be done, by when and by who. The plan is a fundamental necessity to this or any project as without one, it will be a disaster so to speak. The first stage of planning was to hold a meeting with the supervisor. In this meeting the overall project was discussed with the objectives and tasks being set out from the beginning. An idea was given as to what the final project should eventually look like. This was used as a guideline, but changes could always be made to add cool new features to the project. The next stage of planning involved designing the system. It was imperative to get an understanding of what would be happening with the application, what sort of data would be passed from front to back end, where it will be stored etc. A simple architecture of the solution was drawn up (see system design). Following this, some front-end design was drawn up to get a basic understanding of what a final product may look like (see system design diagrams also). Discussion between team members and supervisor was had over the potential design and some changes and additions were made accordingly. The final stage of the first phase of planning was to set up a Jira board. On this Jira the following were set up:

* Roadmap

Graphical user interface

Description automatically generated

The roadmap is a useful tool that allows for a visual representation of the project progress. It is essentially a Gannt chart. The sprints could be adjusted accordingly as needed. For example, if sprint one was taking longer than initially planned, simply moving the bar on the chart allowed for the sprint to continue until the new date it was moved to.

* Backlog

Graphical user interface, text, application

Description automatically generated

The backlog was where any tasks that were incomplete in the sprint were held. It was easy to access and see what is holding the project progress up and who has been assigned the task.

* Board

Graphical user interface, application, Teams

Description automatically generated

The board is a cool feature that basically has 3 categories as seen above. It categorises what needs to be done, what is in progress and what has been completed. Once all tasks have been completed the sprint can be completed and the next sprint will begin.

Tasks were separated into sprints. There was an idea to have three sprints. Getting each sprint done and moving on to the next sprint was ideal, however there was also flexibility. Should a task or feature take longer than expected, it was agreeable that sprints could be adjusted accordingly, or the task could be moved into the new sprint.

Sprint 1:

It was decided that during sprint 1 the following would be attempted to be completed:

* Login page implementation
* Register page Implementation

Sprint 2:

It was decided that during sprint 2 the following would be attempted to be completed:

* Home page design
* Graph pages display
* Pop up with user info

Sprint 3:

It was decided that during sprint 3 the following would be attempted to be completed:

* Home page dynamically display data
* Messaging application

The sprints were adjusted as the project progressed, but this was how it was planned originally.

After a plan had been organised, the next step was vital. It was important to research the technologies that were going to be used in the development of the web application. Some of the research that was conducted was to do with Neo4J and how to use it as this was going to be a new language that had to be learned. The next bit of research that had to be done was with the graphical side of the application. There are many different tools that can be used to superimpose a graph onto the front end, but it was important to find the best one for this project. Some of these tools researched was D3.js, Neovis and VivaGraphJS among many others.

Meetings were held on a weekly basis. This was decided on in the early days of the project. The importance of meetings were vital in order to ensure the project was being kept on track and to ensure every team member was doing as much as they could to the best of their ability. The supervisor was a big help in the advice that they gave as well as the reassurance of the work that was done was correct and done to a high standard. Feedback was very important to the progress of the project.

GitHub was used for the storage and collaboration of the project. This is where any work was stored and continuously updated with any changes and or issues that may have arisen. The commit history was of particular use as there it could be seen what stage the project was at and if an error needed fixing. OneNote from Microsoft was used for documenting any progress or issues as well as minute meetings throughout the duration of the project. The other tool for project management was Jira which was discussed in detail above.

References:

[**https://www.synopsys.com/blogs/software-security/top-4-software-development-methodologies/**](https://www.synopsys.com/blogs/software-security/top-4-software-development-methodologies/)

[**https://github.com/neo4j-contrib/neovis.js/**](https://github.com/neo4j-contrib/neovis.js/)

[**https://neo4j.com/developer-blog/graph-visualization-with-neo4j-using-neovis-js/**](https://neo4j.com/developer-blog/graph-visualization-with-neo4j-using-neovis-js/)

[**https://d3-graph-gallery.com/**](https://d3-graph-gallery.com/)

[**https://d3js.org/**](https://d3js.org/)

**Technology Review**

**System Design**

The how of the project.

The application was made using the React framework. React is a web application framework that is used for creating interactive user interfaces. In addition to the React framework, the graphical designs that were used to superimpose the graph from the Neo4J database was done using D3.js. The first step of the design was to plan it, as previously mentioned in the methodology section. Below are the sample designs that were drawn out and used as a basis for developing the front end of the application.

1. Login Page

A drawing on a piece of paper

Description automatically generated with medium confidence

1. Home Page

A drawing on a piece of paper

Description automatically generated with medium confidence

1. Messenger Page

A piece of paper with writing on it

Description automatically generated with medium confidence

1. Register Page

A picture containing text, whiteboard

Description automatically generated

It will be seen from the implementation of the project that the designs are not exactly the same. Some pages were changed with more additional changes added. Below there will be an in depth look at each of the pages on the application and there will be some noticeable differences from the design, but these differences are an improvement to the overall design of the application.

1. Login Page

The login page was the first point of design that was implemented on the front end of the application. This is the component that the user first sees as they access the website. The page has a design of the GMIT logo at the top on a blue background. There is a google login button to which the user can login to the application. A welcome message is included on the card. The user must login to the website using google. If they do not they will not be able to access the rest of the application as it is protected using protected routes.

1. Protected Routes

Below is an example of the Protected route in action. The user will not be able to access the components that are under the Protected route bracket unless they have successfully logged in.

A screenshot of a computer

Description automatically generated with medium confidence

The user must be authenticated. If the user provides valid details, the auth function is set to true. When the user logs out, the Boolean value is set to false. The getAuth function gets the state of the Boolean.

Text

Description automatically generated

A protected route component is set up. In this function the authentication value from getAuth is retrieved. If the user is authenticated then access to the site is allowed, otherwise, the user is redirected to the login page.

Text

Description automatically generated

1. Login Page functionality.

The login page makes use of the react google login library. In this component there is a client id, which is related to the google cloud console. An onSuccess and an onFailure method.

Text

Description automatically generated

The onSuccess method is basically everything that occurs when a user has valid credentials and the login is working correctly. An object is set up that is called googleresponse. This object contains the name, email, id and image URL of the user. The auth method is then called and set to true. The details are then posted to the backend using an axios post method. In order for the user to be able to access the website and to have dynamic data set up for every user, information is set using session storage. This will be elaborated on further in the home page design description. The onFailure method is just a simple message that is printed to the console.

Text

Description automatically generated

Finally, below is a screenshot of the login page in both browser and mobile format.

Graphical user interface, website

Description automatically generated

Graphical user interface

Description automatically generated

Home Page

The home page is essentially the landing page of the application. This is where the user, after logging in will be redirected to. The home page consists of two different designs depending on the platform the user is on. When the user is on a browser such as chrome, the home page consists of a card with the GMIT logo on it. When the user hovers over the card, it will rotate showing the back of the card. Here is where the user that is logged in details will appear. Below is an example of this

Before:

**A picture containing graphical user interface

Description automatically generated**

After**:**

Graphical user interface, application

Description automatically generated

When the user is accessing the page through a mobile page, there is no card. It is displayed in a simpler format. Below is an example of this.

Graphical user interface

Description automatically generated with low confidence

The home page is where it will display the dynamic and unique information to of the user. It displays the users profile picture, their name, the course they did, where they are now, their interests and finally, hobbies. – session storage to get the user that is logged in, then use that to get dynamic info from backend – axios get request, display that info. CSS to make the card

**Text

Description automatically generated**

As seen from the screenshot above, the useStates and empty arrays are set. 2 variables called userLogged and userImage are created. The two respectively are assigned to the session storage userData and session storage userPic. This is how the home page for each user is created dynamically. useEffect is then used to call the getAllInformation method. The getAllIInformation method is where the application connects to the backend of the application and gets all the data from the backend. This is done using an axios get request. Displaying the profile picture is done by using the session storage as mentioned above. The data is retrieved from storage and then displayed in an image style as seen below.

A screenshot of a computer

Description automatically generated with medium confidence

The resulting information is then displayed on the card that was shown above if on a web browser. The secondary simpler style is used if on a mobile device.

**A screenshot of a computer screen

Description automatically generated with medium confidence**

Register Page

The register page is where the user can add details to their profile. On the logging in and the creation of the account, the user only has the google login details associated to them. Here is where the user can add their name and biography, course, the year they studied said course, interest and hobbies and where they are now. The user does not have to enter any details here at all, although they will not show up on the home page that has been detailed above. The idea around the application and the use of a no SQL database was to give the user as much flexibility in what they wanted to be shown on their profiles. The register page is divided into 4 cards. The user enters details and clicks on the buttons to add the information to the database. The cards are placed on a carousel, so in order to access the next card, the user must click on next as seen below.

Graphical user interface

Description automatically generated

Going to the next item on the carousel will bring up a new card where the user can then add that information to the profile. Once the user is satisfied with the information that is added to the profile, they can then see said information by navigating back to the home page.

Adding the information and posting it to the backend is done using useState.

Text

Description automatically generated

The addDetails function is created using an axios put request. The reason for this being a put rather than a post will be discussed a little further on in the backend section. This updates the name of the user and the biography of the user.

Text

Description automatically generated

The addCourse function is a post request to the backend and adds the course to the database if not already there. This is also the same for the add country and add interest functions. The add country method however adds the value of the label to the backend. This is because the way a user selects where they are now is by using a react select bar. The add interest method also allows for multiple interests to be passed to the backend of the application. All of these post requests are done by posting the data to the backend as an Object.

Text

Description automatically generated

The final step of creating a profile is to create links to each object. For example, creating a relationship or link from the user profile to the course. This done by posting the userLogged variable, where information is retrieved from session storage, along with the associated value, i.e. course posts the course value.

Text

Description automatically generated

Below you can also see an example of the card being displayed as a carousel item in code.

Text

Description automatically generated

The react select bar is created using the react-select-country-list package. It is a useful library that was made use of to create a list of countries rather than individually making the data itself.

Graphical user interface, application

Description automatically generated

Adding multiple interests is quite an interesting feature that was also implemented. This allows the user to click a button and it will create a new field where the user can enter another interest.

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated

Graph Page

The graph page, in essence, is a way of displaying the data from the graph database Neo4J and superimposing it onto the front end of the application for the user to see, use and interact with it. The user is first left with a choice of which course they wish to view the graph of. This is done in similar fashion to the react select bar that was used for countries, this time using the react-select package. After selecting a course, a dynamic page is then loaded with all the information from that specific course being loaded on to the screen. The graph is then drawn with D3.js. The user can interact with the graph where they can click on the user’s name. This will bring up a popup showing the users name and asking if they would like to connect with them in the messenger page. Below is an example of how the graph will look.

Graphical user interface

Description automatically generated with medium confidence

As more people register this course and link it to their profile, the more names or nodes will be added to the graph connecting to the main node which is the course name.

After much deliberation between the different technologies as mentioned in the Technology Review section, it was decided that D3.js would be the best option to superimpose the graph onto the front end. As well as the issues with the other technologies, those other technologies did not offer the same flexibility that D3.js was offering. D3.js had many benefits outweighing the others, the one issue with it was that it was a very difficult part to learn for the project.

An axios get request was used to get all the data for the specific course from the back end of the project. This data is then pushed into an array.

Text

Description automatically generated

The next part is where the D3.js comes into play. The array is looped through to get all the different elements or names that are stored in it from the get request. These are then set as a target for the source. The source and the target are then connected together.

Text

Description automatically generated

The next part was to draw the graph itself. First the size of the graph is created and the intricate details such as distance between node and source is created. Text

Description automatically generated

The variable circle is now drawn and the relevant methods called. The on click method will set the name to appear in the popup and the visibility of the popup will be set to true. The sizing of the text method is also created, this is just simple formatting for any text that appears.



Text

Description automatically generated

Finally, the graph is drawn on the target space, the relevant functions are called, and the graph is created successfully. Text

Description automatically generated

There are some things that need to be fixed or that can be done to make the graph better or more user friendly which will be discussed in the System Evaluation section.

The popup was an intriguing idea that came to mind during one of the meetings with the supervisor. The idea about the popup was that the user can click on a node in the graph, it would display the persons details and have a button which would take the user to the messenger part of the application and they could start a conversation from there. This was done using the package prop-types.

The custom popup, again like much of the project, uses useState and useEffect. At the start, the useState is set to false and useEffect is used to update the visibility of the popup.

Text

Description automatically generated

The popup then can be used in any part of the application as it was set up as a component, but it was only used on the graph page. The popup contains, the name or title, a close button which links back to the closeHandler function, and the visibility is set to either true or false depending.

Text

Description automatically generated

Navbar

The navbar of the application is essentially the user’s tool to navigate the application with ease. Should the user get confused or unsure of where to go, this will be their go to option. The navbar appears in the top left-hand corner of the application, appearing as seen below.

Text

Description automatically generated

Upon clicking this icon, the following navigation bar is displayed onto the screen.

Graphical user interface, application

Description automatically generated

The options the user will have to navigate to are as follows:

* Home – this redirects the user to the Home Page.
* Add Info - this redirects the user to the Register page.
* Messenger – this redirects the user to the Messenger page.
* Login – redirects the user to the Login Page
* My Networks – redirects the user to the Graph selection Page
* Logout – Logs the user out of the application.

These pages can only be accessed by the user when they have successfully logged in. Sidebar.js is where the navbar is designed. useState once again is set as false at the beginning, as well as setSidebar being set to false. The showSidebar function is set to true when the relevant button is clicked. This so that the navbar does not load onto the screen until clicked. The onSuccess method is what is called when a user has successfully logged out and redirects to the login screen.

Text

Description automatically generated

When the navbar is selected, the data is then displayed. The items are taken from a page called SidebarData.js. These are displayed in a list with the relevant icon and title.

Text

Description automatically generated

The SidebarData.js script contains a simple const which contains a number of different items. Each one with its own dynamic URL, title and icon.

Text

Description automatically generated

Messenger

The messenger part of the application allows for users to connect with other registered users. The user can register for the application on the login screen with the ARLA Messenger login.

Graphical user interface, text, application, chat or text message

Description automatically generated

This will then authenticate the user with firebase. Allowing them to access and use the application. This is done by creating a function called auth using firebase.InitializeApp. This contains an api key, domain, project Id, senderId and appId. This leads to the AuthContext.js file. This is used to set the user. The user is set to null at first. useEffect is then called into play. Here the auth function from firebase.js is called which then sets the user. The value is then set to user.

A screenshot of a computer

Description automatically generated with medium confidence

On successful login, the user can then access the messenger part of the application. The user from here can set up new chats, add users to chats, delete chats and most importantly message other users.

At this point of the application, chatengine.io comes into play. An axios get request is used to get access to the messenger application. This uses the username and user secret as well as the project ID. Should the user not have logged in correctly they will be redirected to the login screen.

Text

Description automatically generated

Within the catch statement., an axios post request is performed adding a new user to chatengine.io. This is where the private key is posted.

Graphical user interface

Description automatically generated with medium confidence

Below is what the messenger application looks like after a successful login and two users have messaged each other.

**System Evaluation**

**Conclusion**

**References**

**Appendices**