COMP4905 HONOURS PROJECT

Relay
An Application for Creating and Viewing Articles

Theophilus Jeffrey Ali
100868083

Supervised By: Tony White

15 December 2017
TABLE OF CONTENTS

1.0 MOTIVATION
   1.1 Application Rationale
   1.2 Technology Rationale
   1.3 Project Goals and Objectives

2.0 METHODOLOGY
   2.1 Technologies Used
   2.2 Functional Requirements
   2.3 Non-Functional Requirements
   2.4 Use Cases
   2.5 Approach
      2.5.1 The Backend
      2.5.2 Creating and Posting an Article
      2.5.3 Viewing an Article

3.0 RESULTS
   3.1 Product Walkthrough
   3.2 Application and Future Works
   3.3 Summary

4.0 REFERENCES
1.0 MOTIVATION

1.1 Application Rationale

When smartphones were first introduced, they instantly took the world by storm particularly in the way society consumed information. Nowadays, a person’s smartphone is a worthy complement to courses, books, and computers when it comes to cultivating knowledge. There are hundreds of mobile applications that exist today with the purpose of aiding one’s learning experience. Relay is an Android mobile application that aims to do exactly that with a different approach.

Currently, there are existing mobile applications that share some of the functionalities and purpose of Relay. Medium for example, is an application, existing in both web and mobile form, that is used for publishing and reading articles. Topics range from business to programming to cooking and can be general or specific since users can use existing general tags or create their own specific tags. Another popular application is Chegg, an application specifically targeted towards students, also existing in web and mobile form. Students can post notes, exam papers, assignments, and even questions in hopes for a solution to that question. Chegg however is not free and requires a monthly payment for continued usage of its various services.

However, there exist problems with these two mentioned applications. Medium works well for general to specific topics but has too broad of a target audience. This makes Medium not entirely suitable for creating a student community within the application’s environment since students in college or university take a wide array of courses and articles with these topics in mind are not covered well on Medium. It is very important for a successful application or platform to have a strong user community as this helps form the foundation of the application itself. Without the user, the application is basically useless. The need of a strong community for an application will be further elaborated in the next section.
Chegg on one hand is a platform specifically designed with students in mind but on the other hand is not free and is not a suitable environment for learning. Most students would only use Chegg for solutions to questions, most of the time not even understanding the solution. This also falls within the realms of plagiarism and dishonest learning. Relay is therefore a hybrid of these two applications and aims to alleviate these problems by taking on a different but familiar structure.

1.2 Technology Rationale

One of the biggest reasons why React Native was chosen for this project was because of how easy it is for a front-end web developer already familiar with React to begin developing a mobile application. As a person who has never done any mobile development, but has done web development using React, creating my first mobile application with React Native was trivial. Initializing a React Native application also takes very little effort with the command line utility, `create-react-native-app`. React Native is also perfect for creating quick prototypes on the go and with tools such as Expo, distribution of early prototypes amongst the development team is made simple with the use of QR codes and links.

Although professional mobile developers still prefer to use native tools such as Swift and Objective C for iOS or Java and Kotlin for Android, it is no secret that the popularity of React Native is growing rapidly every year especially among programmers just starting out with mobile development. As of today, according to the list of most starred repositories openly available on GitHub, React Native is the 14th most starred repository on GitHub with 1,554 contributors and 57,216 stars. This rise in popularity since it was open-sourced in March of 2015, is evident that programmers, beginners and professional alike, see a potential in the capabilities and application of this cross-platform tool.

There is no denying that when it comes to overall performance of large scale applications, cross-platform applications such as React Native can never achieve what native applications can. However, there is also no denying that the performance gap between the two is getting narrower and narrower over time. An article by Weblab Technology was published in
November this year highlighting the theoretical as well as practical aspects of using React Native in comparison with Swift [Knyga and Hayat, 2017]. Two almost identical applications were developed using Swift and React Native respectively and then compared based on CPU, GPU, and memory consumption. Swift had better results in the CPU and GPU usage while React Native in memory consumption. The results from this comparison does not definitively conclude the performance of both tools but it does shed some light on the potential for React Native to be a legitimate tool for developing mobile application.

1.3 Project Goals and Objectives

*Relay* is designed with college and university students in mind and has tags or interests matching the typical courses a student of a certain program would take. With *Relay*, users can share their knowledge on any topic specific to a course by creating an article with stylable texts, images and links. It could be an article containing the 20 important terms in DNA replication, tips on how to solve a statically indeterminate beam problem, or a helpful analogy for understanding pointers in programming. This application aims to create a portable and accessible platform where users can read, write, and share their knowledge. Whether it’s sitting on the train, during a lunch break, or waiting between classes, *Relay* can be used for filling in those wait times productively as opposed to just going through social media or playing mobile games. An article released by the news group Hacker Noon and a company Text Request, highlighted results from research conducted by several companies that specialize in data analytics for media consumption [Hacker Noon and Text Request, 2017]. The article highlighted that the average time per day spent on social media platforms such as YouTube, Facebook, Snapchat, Instagram, and Twitter are 1 hour, 56 minutes and 66% of that time takes place on smartphones.

The application is also not trying to replace conventional learning resources but rather complements them by strengthening a user’s familiarity and understanding of a subject. While conventional learning resources attempt to convey information to the general population, articles created among fellow students can be more effective due to peer
relatability. This also in turn instills a reading culture among students even when they are on their smartphone.

The mobile and portable nature of Relay can promote users to create short and concise articles without having to sacrifice quality. This forces the user to carefully select the essential points of a concept and structuring it in an effective manner instead of just including everything and writing a wall of text. A quote often attributed to Albert Einstein goes; “If you can’t explain it simply, you don’t understand it well enough”. Although this quote has never been properly sourced, the inverse also stands true; “If you want to understand something well, try to explain it simply”. This is the fundamental idea of the Feynman Technique, a mental model named after a Nobel Prize-winning physicist Richard Feynman [Thomas, 2017]. This technique usually involves selecting a concept and writing it down, slowly elaborating on every detail associated with that concept. This principle is the main idea behind creating articles in Relay and will be explored further in the methodology section.

Although very similar in terms of execution to what Medium does, Relay is different in terms of what it represents and the type of network of users that will form around the application. Universities around the world such as Stanford, Waterloo, and Wollongong in Dubai all have different program requirements for their computer science bachelor’s degree program but each of those programs have common courses like data structures and algorithms, database, object oriented programming, calculus, discrete structures and so on [Wollongong n.d., Stanford n.d., Waterloo n.d.]. Mechanical Engineering for example, will have mechanics of solids, thermodynamics, fluid mechanics, study of materials, calculus, algebra and so on. By creating a platform where students from different academic institutions can share their common knowledge on a course and have other students receive and convey it to other students, this creates a perpetuating cycle of knowledge exchange amongst users.

Although very much designed with students in mind, Relay can certainly still be used by non-student users provided they do not mind having more specific topics of interests to read or write about.
2.0 METHODOLOGY

2.1 Technologies Used

Relay is developed using the JavaScript library, React Native. The app is built upon the create-react-native-app boilerplate which only handles the configuration to build native code. This boilerplate is far from being the finished app since the UI, flow of data, and all functionalities of the app had to be implemented. Besides handling the configuration, create-react-native-app also comes with a framework called Expo.

Expo comes with a set of APIs for the smartphone’s camera, video, contact, and more. When a React Native (RN) application is started, a QR code is created and users can use the Expo mobile application on their smartphones to scan the QR code and have their RN application bundled and loaded onto their smartphone. This QR code, which is generated from a single codebase, can work on both Android and iOS provided no native libraries specific to any OS was used in the development of the application. However due to the limited time allocated to this project only Android devices will be tested with the application.

Backend processes are performed through Firebase, a mobile and web application development platform offering various backend services.

Devices used to test the application’s features are an Android smartphone and an Android tablet, both with Android 6.0 and 7.0 respectively.

2.2 Functional Requirements

The functional requirements listed in Table 1 below are the minimum functionalities of the application which I deduced must be in the application for it to meet the goals and objectives outlined in section 1.3. Each of these functionalities will be elaborated to highlight the rationale
behind its existence. Any addition of functionality to this list will certainly contribute to a better user experience but is not necessary for the sake of meeting the goals of the project.

**Table 1: Functional Requirements**

<table>
<thead>
<tr>
<th></th>
<th>A user must be able to register for the application using email</th>
</tr>
</thead>
<tbody>
<tr>
<td>F-1</td>
<td>• This is how the application will ensure only authenticated users can use the application and maintains user state across the application (user is either signed in or out at any given time).</td>
</tr>
<tr>
<td>F-1-1</td>
<td>A user can select the interest tags they want added to their profile before registering for an email</td>
</tr>
<tr>
<td></td>
<td>• By having an interest page for users to select their interest tags before they register, this hopefully keeps them motivated in wanting to try out the application.</td>
</tr>
<tr>
<td>F-2</td>
<td>A user must be able to sign in to the application using email</td>
</tr>
<tr>
<td>F-3</td>
<td>An authenticated user must be able to personalize their profile.</td>
</tr>
<tr>
<td></td>
<td>• Giving the user the ability to personalize their profile is a way to maintain a virtual identity and presence across the application.</td>
</tr>
<tr>
<td></td>
<td>This includes:</td>
</tr>
<tr>
<td>F-3-1</td>
<td>Changing their profile picture</td>
</tr>
<tr>
<td>F-3-2</td>
<td>Changing their username</td>
</tr>
<tr>
<td>F-3-3</td>
<td>Changing their interest tags</td>
</tr>
<tr>
<td>F-4</td>
<td>An authenticated user must be able to create an article.</td>
</tr>
<tr>
<td></td>
<td>• This feature is one of the main features of this application. As mentioned in section 1.3, a user can test and strengthen their understanding of the topic they choose to write about by writing down what they know from memory and based on their personal understanding.</td>
</tr>
<tr>
<td></td>
<td>This article creation feature must have the following:</td>
</tr>
<tr>
<td>F-4-1</td>
<td>Behave like a standard WYSIWYG editor</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
| **F-4-2** | **Basic editing operations such as editing, deleting, and inserting between paragraphs**  
- It is important to note that in the context of this application, an image or a link is considered a paragraph on its own. This will be explained in the Approach section later. |
| **F-4-3** | **Able to add text, images, and links. Text must have various styling options to allow for better text segmentation**  
- These are the fundamental types of data a typical WYSIWYG should have. Besides text, images complement textual information well and can help aid the visualization of information being conveyed in an article. Links are useful for providing external information to the user that would otherwise clutter the already limited screen size of a mobile application. |
| **F-4-4** | **Able to add tags from a list of default tags as well as own custom tags** |
| **F-5** | **An authenticated user must be able to view articles on the home page, on the list of articles created by them, and on the list of bookmarked articles**  
- When a user views their own articles, they must be able to acknowledge that the article being viewed looks the same as it did during its creation. This feature will be tested and presented in the results section. |
| **F-6** | **An authenticated user must be able to bookmark articles created by other users**  
- Bookmarking articles is a great way for users to be able to revisit an article or to save the article for future reading. |
| **F-7** | **An authenticated user must be able to edit any of their posted article**  
- The article being edited must be loaded into a WYSIWIG editor and have the same functionalities as outlined in F-4. |
| **F-8** | **An authenticated user must be able to delete any of their posted article**  
- When an article is deleted, it should not appear anywhere else in the application; home page, bookmark page, and own articles page. |
| **F-9** | **An authenticated user must be able to log out of the application** |
2.3 Non-Functional Requirements

The non-functional requirements of this application outline the technical constraints and standards the application must meet to ensure optimal performance, usability, and reliability. Besides those three aspects, the application must also be constrained in other types of aspects such as the interface, implementation, and security.

Table 2: Non-Functional Requirements

| NF-1 | **Usability**: The application must present a different authentication process for new and returning users |
| NF-2 | **Usability**: The system should be easily navigable |
| NF-3 | **Usability**: Error messages should be descriptive and suggest appropriate solutions |
| NF-4 | **Usability**: Proper messages should be displayed when an action is successful |
| NF-5 | **Performance**: Users should be registered, signed in, or logged out within 5 seconds |
| NF-6 | **Performance**: Articles should take less than 20 seconds to be posted |
| NF-7 | **Performance**: Articles should take less than 5 seconds to open when viewed |
| NF-8 | **Performance**: All articles should be loaded onto the home page within 5 seconds |
| NF-9 | **Reliability**: Articles should be saved to the backend server and posted on the home page 99% of the time with 0 faults |
| NF-10 | **Reliability**: Users should be successfully authenticated and have the correct profile information loaded into the application 99% of the time with 0 faults |
| NF-11 | **Reliability**: Viewed articles must resemble how it appeared during its creation 99% of the time |
| NF-12 | **Interface**: The interface of the application must be graphical in nature |
| NF-13 | **Interface**: The interface must be presented clearly with appropriate, nonconflicting colours and readable font size |
| NF-14 | **Interface**: The use of graphical icons must be appropriate and accurate to the functionality it is tied to |
| NF-15 | **Implementation**: All source code must be written in JavaScript and compiled in a single command |
**2.4 Use Cases**

The use cases outlined in *Figure 1* below shows the type of operation a user can perform with Relay. Each operation is also elaborated further in *Table 3* below:

![Use case diagram for Relay](image)

*Figure 2: Use case diagram for Relay.*

<table>
<thead>
<tr>
<th><strong>Table 3: Use cases for Relay</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UC-1</strong></td>
</tr>
<tr>
<td>A first-time user registers for the application using email. A message will be displayed saying ‘Welcome to Relay!’ to indicate a successful registration. The user will be presented with the home screen where the</td>
</tr>
<tr>
<td>UC</td>
</tr>
<tr>
<td>------</td>
</tr>
</tbody>
</table>
| UC-2 | RegistrationError | An error will be displayed with the following possibilities:  
1. An existing account with the same email already exists  
2. Password must be more than 6 characters long |
| UC-3 | SignIn            | An existing user signs into the application using email and a message will be displayed saying ‘Welcome back’ to indicate a successful sign-in. The user will be presented with the home screen where the articles will be displayed. |
| UC-4 | SignInError       | An error will be displayed with the following possibilities:  
1. There is no existing account associated with the entered email  
2. Incorrect password entered for the email provided |
| UC-5 | LogOut            | The authenticated user logs out and is brought to the logo screen with the options to register as a new user or sign in |
| UC-6 | PersonalizeProfile| The user can personalize their profile and can see these changes occur |
| UC-7 | ChangeProfilePicture | The user presses the hamburger icon or swipes from the left edge of the screen to bring up the side menu. Here, their profile picture will be displayed at the top of the menu. Upon clicking on it, a modal dialog with the option to use the camera or select pictures from the gallery appears. After a picture is selected, the profile picture will be set after a few seconds. |
| UC-8 | ChangeUsername    | The user presses the hamburger icon or swipes from the left edge of the screen to bring up the side menu. Here, their username will be displayed at the top of the menu. The default username will be the first section of their email before the “@” symbol. Clicking the username will prompt a modal dialog to be displayed with a text input. Typing in their new username here |
and pressing the “Apply” button will close the modal dialog and a message of ‘Username changed’ will be displayed to the user.

<table>
<thead>
<tr>
<th>UC-9</th>
<th>ChangeInterestTags</th>
</tr>
</thead>
<tbody>
<tr>
<td>The user presses the hamburger icon or swipes from the left edge of the screen to bring up the side menu. Then the user goes into the “Interests” option. Here they will be presented with the option to add more interests to their list of interest or remove from it. They can also add custom interests to their list of interests. After that the user presses “Apply” and a message will be displayed saying “Interests saved”</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UC-10</th>
<th>CreateArticle</th>
</tr>
</thead>
<tbody>
<tr>
<td>From the home screen, an authenticated user selects the “+” icon on the right-hand side of the header. This brings the user to the WYSIWYG editor. A user can perform several operations outlined in UC-11, UC-12, UC-13, and UC-14, as well as in section 2.5.2. When they are done creating the article, the user presses “Next” on the right-hand side of the header and a modal dialog appears with two text inputs; one for title and tags. After typing in a title and one or several tags, the user presses ‘Publish’. One or more messages saying ‘Uploading...’ will be displayed on the screen and the number of times this message shows up and the time it takes for an article to be successfully published depends on the number of images in the article and the quality of each image. After the article has been successfully published, a message saying, ‘Article published’ will displayed and the user is automatically taken back to the home screen.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UC-11</th>
<th>EditParagraph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regardless of the data type, each paragraph of text, image or link will be treated as a single paragraph. When a paragraph of text, image, or link is added into the viewing area, the user clicks on it. This opens a modal dialog with options to either edit, delete, insert above, or insert below the selected paragraph. The user selects the “Edit” option and the modal dialog closes and an instruction will appear at the top of the WYSIWYG editor with a ‘Cancel’ button beside it to</td>
<td></td>
</tr>
<tr>
<td>UC-ID</td>
<td>Function</td>
</tr>
<tr>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>UC-12</td>
<td>DeleteParagraph</td>
</tr>
<tr>
<td>UC-13</td>
<td>InsertAbove</td>
</tr>
<tr>
<td>UC-14</td>
<td>InsertBelow</td>
</tr>
<tr>
<td>UC-15</td>
<td>ViewArticle</td>
</tr>
<tr>
<td>UC-16</td>
<td>BookmarkArticle</td>
</tr>
<tr>
<td>UC-17</td>
<td>DeleteArticle</td>
</tr>
</tbody>
</table>
2.5 Approach

The mobile application Medium was the main influence for the functionality of Relay. Being careful to only treat Medium as an inspiration and not as a blueprint, Relay is structured around a typical university or college student’s program syllabus by using the interests tags that new users select before they register for the application. These tags are the tags that users can use when adding tags to their article and users can even add their own custom ones too. These interest tags are categorized by programs and for now the only programs listed are computer science and engineering.

The name of the application, Relay is very closely related to what the application aspires to achieve; as a platform for users to convey their knowledge and then for the receiver of that knowledge to relay that knowledge to other people. Initially the application was named “ToGoNotes” but was changed to “Relay” due to reasons mentioned earlier.

The approach to implementing this application can be broken down into four main parts; the backend, creating an article, posting an article, and viewing a posted article.

2.5.1 The Backend

The initial plan was to use Firebase, for all the backend processes. However, at the earlier stages of the project, there was a short attempt at creating a RESTful API from scratch using Node.js, Express, MongoDB, and Passport. Unfortunately, due to the number of intended functionalities outlined in Table 1 and a limited amount of time given for the project, the entire
endeavor was terminated. The API only got up to the email, and Google authentication stages. Firebase was therefore, put back into action.

*Relay* has several backend processes that are performed through Firebase. Although Firebase authentication supports several methods such as Google, Twitter, Facebook, GitHub and email, only registration and signing in with email was eventually chosen due to insufficient time. Despite this, the ability for users to be authenticated as outline in F-1 and F-2 of the functional requirements is still in effect. Adding other methods of authentication will certainly improve the user experience but just email alone is adequate for satisfying the required authentication functionalities.

A registered user can have a custom profile picture, username, a list of academic interests, created articles, and bookmarked articles saved onto Firebase. Firebase however does not store any of the images used in the application and this includes the user’s custom profile picture and images in articles. Amazon S3 had to be used for storing and retrieving images. This is due to the image format outputted by Expo’s image picker. The format of the image URI is incompatible with the way Firebase stores its images and after performing extensive research on the matter, a workaround was found. I would have to upload images onto a different cloud storage first, in my case Amazon S3, and then retrieve the URL for that stored image from a callback object that is returned from the function that uploads the image. That image URL would then be stored on Firebase as a string and can simply be retrieved and used as a source for image components.

The entire process of image uploading from the user’s device to Amazon S3 and finally to Firebase will be covered extensively in section 2.5.2.

2.5.2 Creating and Posting an Article

One of the most if not the most crucial element of the application is the ability to create articles. The initial plan was to create a WYSIWYG editor using Facebook’s Draft.js, a framework for creating WYSIWYG editors. This framework only creates editors for web applications, therefore the plan was to create a web version of the text editor and have the mobile application display a web view of the editor through React Native’s component, WebView. Data
between the web text editor and the mobile application would be exchanged using several of WebView’s built-in functions.

A web text editor was eventually created but did not work smoothly on the mobile application when displayed using WebView. The first and major problem was that the typing was very sluggish and seemed to be very buggy. The whole experience was very inconsistent, disconnected, and might even perform differently on other Android devices.

Therefore, the web based text editor was terminated, and a new implementation was engineered for the WYSIWYG editor. To maintain consistency of the text editor between different devices, the text editor had to be made within the app itself and not displayed separately using a web view.

~ The WYSIWYG Editor ~

![High-level Layout of Relay’s WYSIWYG editor.](image)

Figure 2: High-level Layout of Relay’s WYSIWYG editor.

There is currently no open-sourced WYSIWYG editor for React Native and one had to be made from scratch. Although this text editor will not be as dynamic and as flexible as a typical one found in applications like Medium or Evernote, it still has elements of “what you see is what you get” to it.

The idea is to have a WYSIWYG editor that takes advantage of React’s concept of “state”, which is an object that determines how a component renders and behaves [Simons, 2017]. At a high level, the WYSIWYG editor will have the following sections; an input area, buttons to style and insert data, and an output area.

From Figure 2 above, a user would enter their text in section A and may style it using the appropriate style buttons found in section B. Also, when a certain style is applied, say a header
style which gives a larger and bolded font, the text in section A will appear with that selected style. When the “insert text” button from section B is clicked, the text would then appear in section C with the same styling as applied in section A.

As for inserting images, clicking the “insert image” button will display an option to use the device’s camera or select from the device’s image gallery. Lastly for inserting links, a modal dialog will pop up in which the user can insert the URL to their intended webpage.

![Figure 3: Article Layout and Editing Modal Dialog](image)

Within section C of the WYSIWYG editor, each text, image or link inserted will form a paragraph and can be selected. Figure 2 above illustrates this. Upon clicking a paragraph, a modal dialog will appear with the option to edit, insert above, insert below, or delete the selected paragraph. With edit, the selected paragraph can be replaced with any type of data. The delete option will remove the selected paragraph from the output area. Insert above or below allows data to be inserted at the beginning, end, or between paragraphs. This adds more customizability to an article.
~ Implementation ~

For the implementation of the WYSIWYG editor, a React component named NoteEditor, that holds the entire code base for the text editor was created. The code base for this component can be separated into 8 sections as depicted in Figure 3 below.

The component level states section holds the states for the component. There are several states contained here but the most important state would be a state called editorState. This state is what stores the entire data for an article. This state is structured as an object with three properties; typeArray, dataArray, and styleArray.

Each of these properties will have an array as the value. Each array contains data corresponding to the name of the property. A typeArray property for instance will contain an array of data types stored as strings. An example of such an array would be ['text', 'image', 'text', 'link', 'image']. An example of an instance of editorState would be as such:

```javascript
editorState = {
    typeArr: ['text', 'image', 'link'],
    styleArr: [Styles.header, {}, Styles.link]
}
```

The styleArray only stores styles for texts and links and therefore only has an empty object for images since images do not need any styling. An important thing to note is that the item at index n of typeArray corresponds to that of dataArray and styleArray. This structure is intended and will be explained further in the upcoming sections.
The **data insertion** section contains functions that are responsible for inserting different types of data into the `editorState`. There are five functions in this section. The first three are `insertText()`, `insertImage()`, and `insertLink()`, all corresponding to the three data types handled when creating an article. In each of the three functions, three possible insert cases may occur; a normal insertion of data to the end of an article, an edit to a selected paragraph, or an insertion of paragraph above or below the selected paragraph. The other two functions in this section are the `insertAbove()` and `insertBelow()` both handling the state changes that indicates which type of insertion is occurring and at which index an insertion is at. These two functions do not handle the insertion of data but rather only initiates the action to do so.

The **data renderer** is responsible for taking the data from the `editorState` and rendering it into appropriate paragraphs. The function does this by mapping through the `typeArray` from the
editorState and uses a switch statement on the type of item; either a “text”, “image”, or “link”. Based on each item type, the appropriate case will be used and the corresponding JSX (a syntax extension to JavaScript that produces React elements) will be produced and pushed into an array. This array, let us call it the tagsArray, will contain the JSX code that will be rendered when the NoteEditor component first mounts and whenever it re-renders due to a change in state. Since the items in this tagsArray corresponds to each paragraph inserted into the article in the correct order, the correct ordering of paragraph will always be rendered on the output area of the text editor.

The text style section is responsible for giving styling capabilities to the text-editor. The text editor will have buttons that when clicked will apply a certain style to the text in the input area. These buttons can be toggled and only one style can be active at any given time. If a style button is active when a text is inserted, the style applied to the text will also be applied to the text in the output area. The styles are stored in one of the arrays in editorState.

The paragraph editing section, contains functions that are invoked when a paragraph is being edited. The functions here are mostly responsible for setting certain states of the component to initiate the beginning of an operation. In here, the index of the selected paragraph is kept in the state and is a very crucial piece of state for the operations performed in the edit, insert above, insert below, and delete process.

The index of the selected paragraph is used to select the correct data from each array in the editorState. A selected paragraph with index n will correspond to typeArray[n], dataArray[n], and styleArray[n] in editorState. Therefore, when a paragraph is selected, the appropriate data can be edited, deleted, or have a new paragraph inserted above or below it.

The modal section just contains functions for showing and hiding modal dialogs in the WYSIWYG editor while the helper section contains secondary functions that work with other functions from other sections.

The render() section contains all the JSX code that produces viewable elements in the WYSIWYG editor.
The state machine diagram below depicts the different states and operations an article can have during the entire creation process:

![State Machine Diagram for Creating an Article](image)

**Figure 5**: State Machine Diagram for Creating an Article

The entire NoteEditor component will then be encapsulated in a parent component called CreateNote where adding a title, tags and publishing occurs. *Figure 5* below depicts the flow of data between the two components and to the different backends.
One design pattern that stands out is the state design pattern and this is the main design pattern that React uses when it deals with state changes throughout a component. The behaviour of a component changes with state changes and this is especially evident in the
NoteEditor component where a change in state needs to be reflected to the user through the UI to provide some feedback mechanism.

Another design pattern used here is the façade design pattern. The CreateNote component externalizes all complex operations into the NoteEditor component and through those operations produces data for the article.

2.5.3 Viewing Posted Articles

Once articles are created and published, they would have to be viewable on the home page by every user. The articles they see on the home page should display the title of the article, the creator, the tags, and the date it was created. A user will also be able to bookmark an article and these will appear in the bookmarks section of the side menu. By reading or bookmarking an article posted by another user, the information is or will be conveyed and this covers the knowledge sharing aspect of the application.

One of the initial plans was to allow user to create private articles and share articles privately to other users. However, the plan to have these functionalities was discontinued as it may pose some unwanted outcomes. Firstly, having a user create an article privately is going against the main objective of this application, which is to share their knowledge with other users in the hopes of that knowledge spreading to more people. Private articles would be defeating the purpose of using the application in the first place. As for sharing articles privately with other users, this may instigate academic misconduct amongst users such as sharing solutions to assignments or giving away confidential information on exams. Although this feature may bring about some positive user experience, it is not worth the risk and even without this feature, the application will be able to keep its objectives in check.
3.0 RESULTS

3.1 Product Walkthrough

The results for the final product will be broken down into multiple sections, the authentication process, creating an article, posted article functionalities, and other secondary functionalities.

Two options will be presented to the user when they first open the application. A “Get Started” option for new users and a “Sign in” option for returning users. Both options will guide the user through two different authentication processes.

~ Authentication Process ~

The authentication process can be split into two types; first time users and returning users.

Upon selecting the “Get Started” option, first time users will be directed to the interest page (Figure 6) and asked to select their program and then interests related to the courses they are currently taking.

Articles they see on the home page will match the interests they have selected and can be changed any time in the settings once a user signs in. Once the user has selected their interests, they will be brought to an authentication page containing registration options for Google, Facebook, Twitter, and Email (Figure 7).
A **returning user** will also see a page similar to the registration page in *Figure 7* when they select the “Sign In” option from the logo screen.

After successfully signing in, the user will be brought to the application’s home screen where they will be able to see all posted articles, and various options such as the side menu, creating an article, filtering and search.
Creating an Article

The completed WYSIWYG editor will be presented and discussed in the next few sections. *Figure 10* and *11* below shows the layout for the WYSIWYG editor which complies to the abstract layout illustrated in Section 2.2.2.

To add a styled text, a user would enter their text and select the style button of their choice. The “H1” and “H2” buttons will give the text a larger and bolded font. The “</>” would give the block of paragraph a light blue background for equations, code, or any text that requires a coloured background. The “quote” button will italicize the text. Once a text is ready, the “+ Text” would be pressed to insert the text into the output area. It is important to note that the “+ Text” inserts the text as a new paragraph. *Figure 12* shows how a new text inserted goes directly below the previous insertion. “+ Image” will simply add images to the article.

*Figure 12: Inserting a styled text*

*Figure 13: An inserted styled text*

*Figure 14: Text ordering*

*Figure 15: Inserted image*
Figure 16: Inserting a link and how it appears in the article.

Pressing the “+ Link” will bring up a modal where the user can add links to the article.

Besides adding consecutive texts, images, and links to the output area, the WYSIWYG editor also allows edit, delete, insert above and below a selected paragraph. When a paragraph is selected, a modal with these options will be displayed on screen as seen in Figure 15.

Figure 17: Selecting a paragraph (the duck image) will bring out the paragraph option. Selecting edit will allow the user to edit texts, replace images and links. In this instance, the image is being replaced with a new image.
When editing texts, the text value will be copied to the text input area to allow for an easier editing experience. Delete will simply remove the selected paragraph from the article as well as from the `editorState`.

For the insert above and insert below option, these options allow texts, images, or links to be inserted between paragraphs.

If the “insert above” option is selected, the user will be able to insert a text, image, or link above the selected paragraph and vice versa for the “insert below” option.

**Figure 18:** Inserting a text above another paragraph

**Figure 19:** Inserting an image below another paragraph

**Figure 20:** Title and tags modal

The title and tags modal will be displayed when the user presses the “Next” button on the right-hand side of the header. When a user is typing a tag, tags that matches the typed tag and are available in the system will appear below the text input. A user can also add their own custom
tags. Once this step is done, pressing the “Publish” button will initiate a post request to Firebase with the `editorState` as the data being posted.

**~ Posted Article ~**

![Home Screen](image)

Figure 21 above shows the home page. The list of articles can be seen here and when an article from the list is selected, the application will display the article in full. Figure 22 below shows the article when it is selected.
Object oriented programming

What is an instance of a class

Figure 22: A viewed article
4.0 REFERENCES


University of Wollongong Dubai, Bachelor of Computer Science Degree Requirements (n.d.) https://www.uowdubai.ac.ae/computer-science-and-engineering-programs/bachelor-of-computer-science-bcompsc-degree. Accessed 13 December 2017

University of Waterloo, Bachelor of Computer Science Degree Requirements (n.d.) http://ugradcalendar.uwaterloo.ca/page/MATH-Bachelor-of-Computer-Science-1. Accessed 13 December 2017

Stanford University, Computer Science Department Requirements, Core Courses (n.d.) http://csmajor.stanford.edu/Requirements.shtml#Core. Accessed 13 December 2017