QuDue: The Essay Support Tool

A dissertation submitted in partial fulfilment of

The requirement for the degree of

MASTER OF SCIENCE in Software Development

in

The Queen's University of Belfast

By

Michael Clarke

21/09/2016

SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING and COMPUTER SCIENCE

CSC7057 – INDIVIDUAL SOFTWARE DEVELOPMENT PROJECT

A signed and completed cover sheet must accompany the submission of the Individual Software Development dissertation submitted for assessment.

Work submitted without a cover sheet will **NOT** be marked.

Declaration of Academic Integrity

Before signing the declaration below please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project Handbook
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)
3. Does not exceed the specified page limit
4. Is clearly presented and proof-read
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance (2xCopies).
6. Software and files are submitted on a memory stick (2xMemory Sticks labelled with Name & Student No.).
7. Journal has been submitted (1xCopy).

I declare that I have read both the University and the School of Electronics, Electrical Engineering and Computer Science guidelines on plagiarism - http://www.qub.ac.uk/schools/eeecs/Education/StudentStudyInformation/Plagiarism/ - and that the attached submission is my own original work. No part of it has been submitted for any other assignment and I have acknowledged in my notes and bibliography all written and electronic sources used. I am aware of the disciplinary consequences of failing to abide and follow the School and Queen's University Regulations on Plagiarism.

Name: (BLOCK CAPITALS) _____

Student Number: _____

Student's signature _____ *Date of submission* _____

Acknowledgements

I would like to thank my project supervisor, John Busch, for his continued advice and guidance throughout this project.

I would also like to give thanks to Eve Wilson for her constant patience and support, and to Niall Clarke for his valuable feedback and opinion.

Abstract

This software deliverable is an academic essay support tool aimed primarily at new students joining Queen's University Belfast. The application aims to help students manage their time by providing them a tool to upload and view details of their upcoming essays. This software comes in a desktop and mobile version so that the user can share information across their devices. The completed solution ensures that users always have a timetable of their upcoming assignments readily available.

Contents

Chapter 1
Problem Specification

1.1 Perceived Problem

The transition from secondary school to university can be a challenging one, with new students suddenly finding themselves more independent and reliant on their own organisational skills. A survey conducted in the year 2000 discovered that after a full semester into their first year of university, over a third of new students did not believe they had adapted to independent learning (Kantanis, 2000). This result is tied to the fact that, with any university degree, students can be expected to submit a number of academic essays. The writing and managing of these essays can prove to be a difficult task, especially since students of any university come from a variety of different backgrounds, schools and even countries, where teaching standards invariably differ. Many students often suffer academically as a result of these changes, however, unknown to them, poor time management is a major contributing factor to their difficulties (Hirsch, 2013). As such, better time management could help ease their problems and ensure the transition is much smoother. Looking specifically at Queen's University of Belfast, students are given support and tutorials from when they first start in the area of essay writing and time management ("Queen's University Belfast | Time Management Resources"). However, despite these resources, the university fails to provide a practical tool that helps students manage their time in relation to their essay assignments.

1.2 - Target Demographic

The users targeted by this project are students of Queen's University Belfast. This incorporates a wide range of ages and backgrounds as the university has over 17,000 students from 120 different countries ("International Students - Queen's University, Belfast"). Specifically then, this will target students who have recently joined the university and aren't yet accustomed to having to write multiple essays over the course of a short period. Additionally, studies have found that on average, students in Northern Ireland work 17.7 hours per week at part-time jobs on top of studying full-time (McKernan & McQuade, 2004). Clearly there is a market for this application then, as it can be used by all students, at any level of the university, who need a tool that assists them with timetabling their essay deadlines.

The benefit of a focused and narrow target demographic means that the project can be built with specific requirements in mind. Since it is required that every user must be a student of Queen's University, there will be consistent and unique data bank (student number, student email) which can be incorporated into the data models and database design. This also applies to the courses and modules offered by the university. Furthermore, Harvard Referencing is generally accepted throughout the university so the project can focus on this referencing style in the development phase.

1.3 - Problem Domain

Queen's Online (QOL) is each student's virtual learning environment whilst at the university. Currently, if a student wants to retrieve information about any essay deadlines they have upcoming, they would have to sign into QOL using a web browser and navigate to the 'Modules' section. Here, there is an Assignment tab where the student can upload their essay. It is not necessary for the tutor to provide any additional information about the essay here besides the due date. Essays will only appear in this 'Assignment' window if the tutor has selected this method of submission for the essay. If the essay is to be submitted via email or in person then this portal will be unused and instead the student will have to navigate to 'Resources' and hope that the tutor has included a file in there detailing information about the essay. Currently, this process is unnecessarily complex and time-consuming, with the lack of a standardised approach making it difficult for students, who, under the time-pressure of meeting deadlines, need this information to be readily accessible.

On the note taking side, popular existing applications include Microsoft's OneNote and Google's Keep. Both these applications allow the user to take notes on the go and then have these notes shared across the user's devices. However, these notes are not directly linked to specific essays and thus require additional organisation by the student. The design and usability of both these applications will be taken into account during the design phase of this project, and whilst some features will be incorporated, this software solution hopes to create a unique tool that makes it as easy as possible for users to add notes directly related to each essay.

1.4 - Modifications to Original Specification

Originally there was a desire to have the solution assist the user in writing their essays. This functionality would have been incorporated alongside the timetabling element and included algorithms applied to the essay text to decipher:

- A consistent tense and point-of-view was used throughout the text;
- Sentence and paragraph length fell within certain boundaries;
- That certain words/phrases were not repeated too often;
- That essay paragraphs were constructed appropriately;
- Other language-based metrics.

This approach was changed as it required the application to become a text-editing tool rather than an essay management tool. It was instead decided that the development would focus on an improved note-taking functionality, one which would give users the ability to attach notes and sources to each specific essay.

Chapter 2
Proposed Solution and Justification of the Development Model

2.1 - Proposed Solution

This project feels that that the current process for students to retrieve and access details of their essays is overly complex. Instead it proposes a new tool that allows each student to self-organise, timetable and manage their own essays. The Academic Essay Support Tool will be designed with three main components: a platform-independent desktop application, an android mobile application and an online server. These technologies will communicate and work in tandem to provide the user with the means to manage their essays easily, efficiently, and most importantly, whilst on the go.

The tool will allow a first time user to register and log in to the system. Users will be able to add details of any upcoming essays, including the essay title, word limit and due date. Essay content can also be imported so that the essay's progress (word count) can be tracked. This project aims to achieve these functions in a manner that will be efficient and user-intuitive, so that as little time and effort on the user's part is required. In terms of improving on the existing problem domain, the solution will make essays available to view as soon as the user has logged in, allowing immediate access to an up-to-date timetable of all upcoming assignments (*Figure 1.1*). Additionally, to assist the user during the writing process, notes and sources can be added specifically to each essay. This feature will keep the user's resources organised and in one location, ensuring instant ease of access.

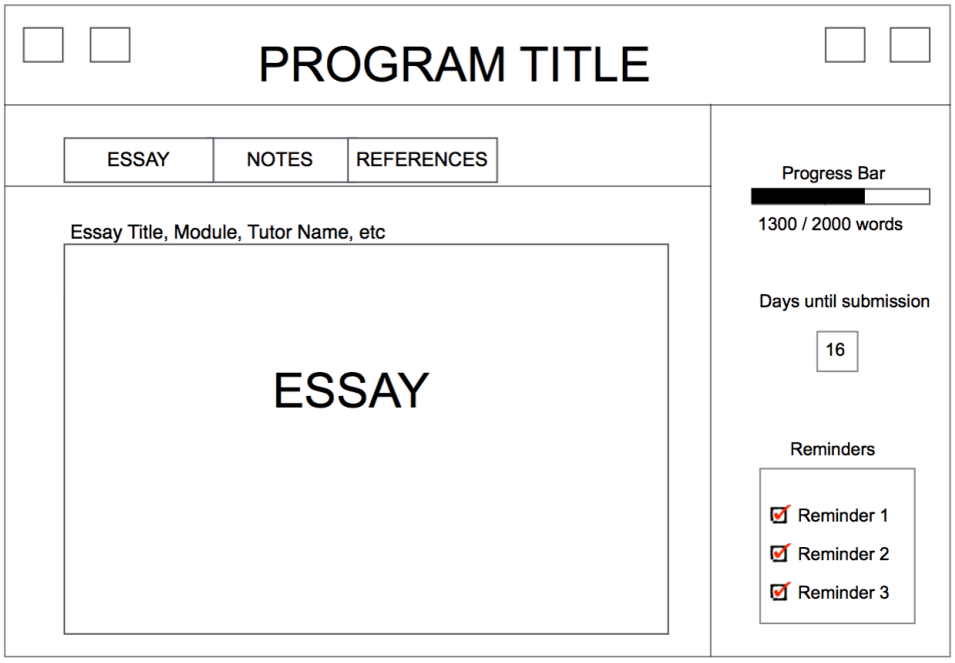

Figure 2.1: Early concept design of the Essay View screen for the desktop application

The user will also be able to download the android mobile version of the software. The mobile application will communicate with the desktop application via a web server, so that the data will be consistent between the two devices. The main function of the mobile application will be to provide the user with a timetable of their current essays and deadlines whilst on the go (*Figure 1.2*). This is an extremely important aspect of the solution because, as detailed in the problem specification, the main difficulties faced by students can be countered by more efficient time management. Having their essay timetable available instantly, addresses this problem and also prescribes to the idea that we are now living in a digital world where smartphones are part of everyday life and consumers have grown accustomed to getting their information on demand from multiple sources simultaneously (Ryan, 2014). Students do not want to have to log into QOL and navigate through serval folders and files to find out information that should be readily accessible to them. Instead, this solution proposes that the information should be able to be retrieved from their smartphone within moments.

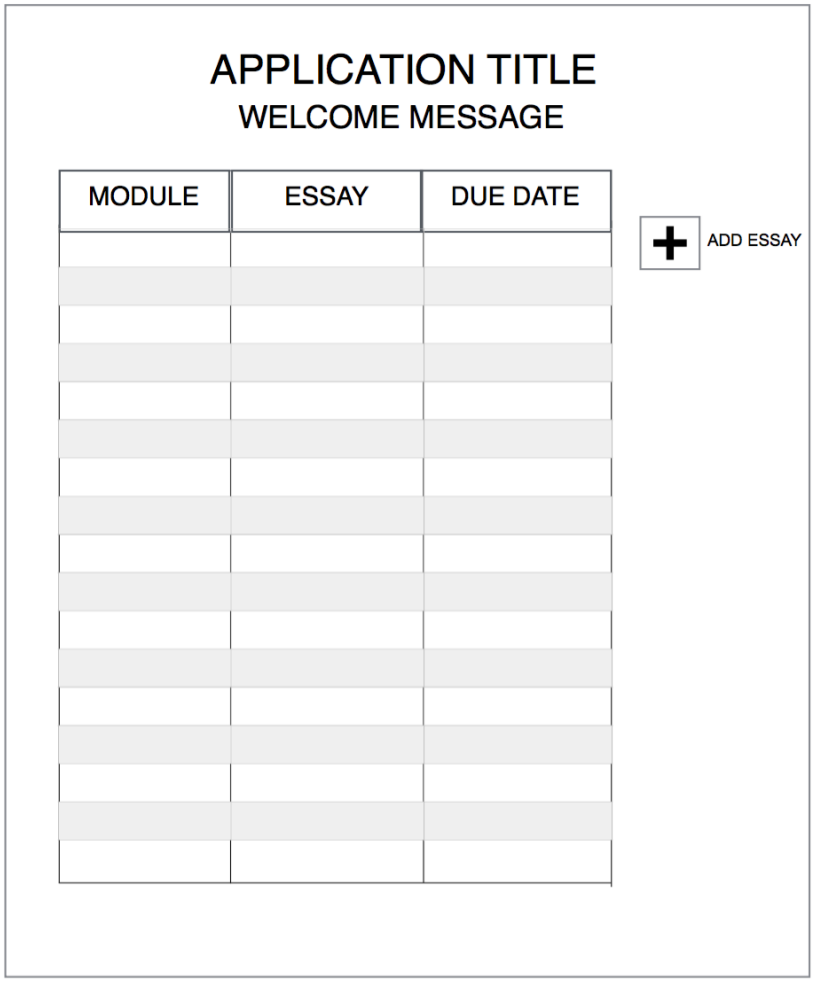

Figure 2.2: Early concept design of the Essay Overview screen for the mobile

2.2 - Potential Problems

This project includes several potential problem areas. The solution's success relies on the ability to share information across several devices so that the data is available to the user wherever and whenever they need it. This requires the creation of a web-server to host the data. Several services, such as Amazon Web Services (AWS), offer the framework to achieve this, however, the setting up process can be time-consuming. Additionally, Android, unlike Java, does not natively communicate with MySql databases. Thus an intermediary script, most likely written in PHP, will be required to complete the connection. Time will need to be allocated to research this method and learn the basics of PHP language.

Other potential problems include:

- The file type which the user attempts to import their essay in. Research into how to extract text from different types of files will be necessary.
- Essays which include images in their body may require more complex containers.
- If the user wants to upload images or audio files to the program, there will have to be a online storage facility set up to manage this.

2.3 - Development Strategy

2.3.1 Kanban Methodology

The objectives of the project are to deliver the outlined software components on-time and ensure they are completed to a satisfactory level of production. To achieve these goals an agile software development approach will be used, with a simplified Kanban model appropriated to suit the needs of the one-person development team. Kanban differs from the Scrum approach in that instead of fixed sprints the production is a continuous cycle where changes can happen at any time (*Figure 1.3*). A board is used to track the development cycle of the project, and usually includes three categories: 'To Do', 'In Progress' and 'Done'. Each task to be carried out by the developer is written onto a card. These cards are placed in the 'To Do' column. The developer then chooses a card and as they begin to work on it, it is transferred to the 'In Progress' column. Once completed it can be moved to the 'Done' column and the developer can be assigned a new card. David J. Anderson (2010), describes this approach as beneficial to the overall deliverable as the simple act of limiting work-in-progress (by only focusing on one card or work item at once) encourages a greater performance from the developer resulting in a higher quality final product.

	SCRUM	KANBAN
Cadence	Regular fixed length sprints (ie, 2 weeks)	Continuous flow
Release methodology	At the end of each sprint if approved by the product owner	Continuous delivery or at the team's discretion
Roles	Product owner, scrum master, development team	No existing roles. Some teams enlist the help of an agile coach.
Key metrics	Velocity	Cycle time
Change philosophy	Teams should strive to not make changes to the sprint forecast during the sprint. Doing so compromises learnings around estimation.	Change can happen at any time

Figure 2.3 - Diagram from the Atlassian website ("Kanban | The Agile Coach") detailing the differences between Scrum and Kanban

The Kanban system is being used for this project because the current requirements are in a fluid state, so changes are expected throughout the development lifecycle. Kanban is set up to accommodate changes occurring at any time throughout the lifecycle, more so than a Scrum approach because software deliverables are not expected at the end of each sprint (Anderson, 2010). Additionally, because of the small development team, this approach will ensure that each work item will receive an equal amount of attention. This will allow a focused and high-intensity development to occur on each software requirement and will ensure each element will be finished to completion, as a new work item cannot be started until the previous one is done.

2.3.2 MoSCow Prioritisation

Because of the potential scope of this project, the development will utilise the MoSCoW method, a prioritisation technique used in software development which assigns a level of importance to each requirement (Stephens, 2015). If the main requirements, the Must-Haves and Should-Haves, are all achieved successfully then the development can focus on the Could-Have features (extra functionality).

A project timetable has been included (*Appendix A*) which gives an indication of the time scale of the development life cycle. The timetable clearly indicates several distinct phases which could be interpreted as sprints. However, because the Kanban method is implemented here, the development for this project will be expected to occur in one continuous cycle instead of fixed sprints. The timetable then is just a presentation of expected targets to be completed by each date but will not be strictly followed or adhered to.

Chapter 3
Data Model, Requirements Analysis and Design

This chapter will provide high-level descriptions of the data model, requirements, software system and user interface design. As the Kanban methodology was used in the development strategy of this project, the low-level details were not realised until the development phase.

3.1 Data Model

The main entities of this software solution included in the database design are:

- Student
- Essay
- Note
- Module
- Tutor
- Course

The structure of these entities can be clearly defined in the following entity relationship diagram (*Figure 3.1*).

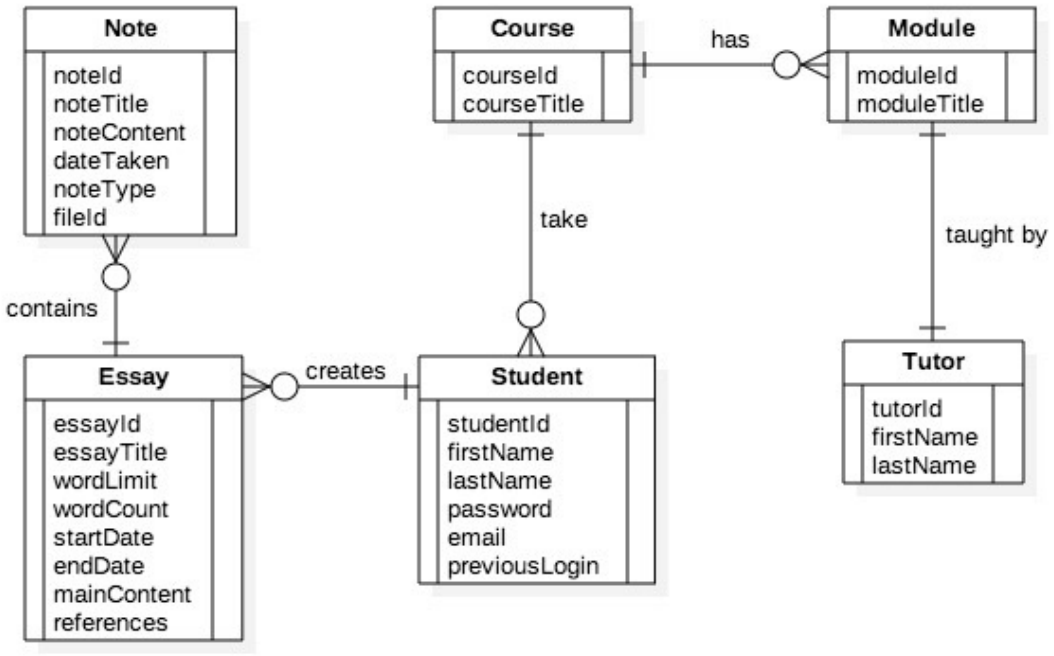

Figure 3.1 - ER diagram relating to the database design

The current ER diagram states the following assumptions:

1. Each Module is taught by a single Tutor.
2. Modules are not shared by multiple Courses.
3. Students cannot co-write Essays.

Additionally, dependencies not labelled by the diagram are as follows:

- Student *must* be signed up to a Course
- Course *must* contain one or more Modules
- Modules *must* be taught by a Tutor
- Student *does not* have to create Essays
- Essay *does not* have to contain Notes

3.2 Requirements Analysis

This section includes the high-level requirements of the software system. A full list of the requirements, including a complete set of function definitions and details of error conditions, will be available in the appendix (*Appendix B*).

3.2.1 Functional Requirements

This section includes the requirements that specify all the fundamental actions necessary in order for the software system to work as expected. The information labelled in the closing parenthesis indicates the version of the software which the requirement relates to.

FR1 - User shall be able to register to the system (Android & Desktop).

FR2 - User shall be able to login to the system using their unique credentials (Android & Desktop).

FR3 - User shall be able to add an essay to the system (Android & Desktop).

FR4 - User shall be able to add content to each essay (Desktop).

FR5 - User shall be able to add text notes to each essay (Android & Desktop).

FR6 - User shall be able to add sources to each essay (Desktop).

FR7 - User shall be able to add notes as images (Android).

FR8 - User shall be able to add notes as audio recordings (Android).

FR9 - User shall be able view a timetable of their essays (Android & Desktop).

FR10 - User shall be able to view their account details and change their password (Desktop).

FR11 - User shall be able to reset a forgotten password via email (Desktop).

FR12 - User shall be able to view their account details and change their password (Desktop).

FR13 - User shall receive notifications of upcoming deadlines (Android).

FR14 - User shall be able to logout of the system (Android & Desktop).

3.2.2 Non-Functional Requirements

This section includes the requirements that specify all the non-functional requirements necessary in order for the software system to work as expected.

3.2.2.1 Usability

NFR1 - The software solution will promote ease of use through a clear and well designed user interface

NFR2 - The user's devices will be connected to the internet at all times when using the software

3.2.2.2 Availability

NFR3 - The Android Mobile version of the software will be readily available to download

NFR4 - The Desktop version will be readily available to download and come with installation instructions

NFR5 - The online server can be accessed anytime by either device to allow instant send and pull requests

3.2.2.3 Performance

NFR6 - The software solution will not crash outside of extraordinary conditions

NFR7 - Whilst accessing the server, system responses should be no more than 4 seconds per request

NFR8 - Multiple users can operate the software simultaneously

NFR9 - Users can access their account on multiple devices simultaneously

3.3 Software System Design

This project uses a combination of several different design patterns to build an overall architecture. The main architectural style is the Client Server model which describes the relationship between a client and a server. In this case, the clients are the Desktop and Mobile devices, which as a result of the Client Server model, never directly talk to each other despite sharing the same data. This data is instead stored in the server which can be accessed by each application through server requests. The model is suited to this software solution because it is used when you want to centralise data storage between different client types and different devices ("Microsoft Developers Guide | Chapter 3: Architectural Patterns And Styles"). This ensures that there is consistent data available to the user across all their devices, a major requirement of this project's specification. The Client Server architecture, in relation to this solution, is presented in the digram below which offers a simplified version of how this system works (*Figure 3.2*).

Figure 3.2 - Client Server model

Although in order for this design to actually work in practice, the pattern needs a bit more complexity. The database is hosted by the server, so each database query must go through the server. The server, if it is provided by Amazon Web Services, will not connect natively to Android as it does with Java (the language used for the desktop application). To solve this problem, an intermediary PHP Script will have to be introduced to connect these two components whenever the Mobile Application wishes to send or retrieve data to and from the server. Additionally, because audio and image files can be saved onto the system by the user, these too must be stored online to ensure they are accessible by both devices. This means that a cloud storage facility will be included in the system's design. The resulting model still achieves the Client Server model but now it has introduced a Layered Architectural Pattern (*Figure 3.3*).

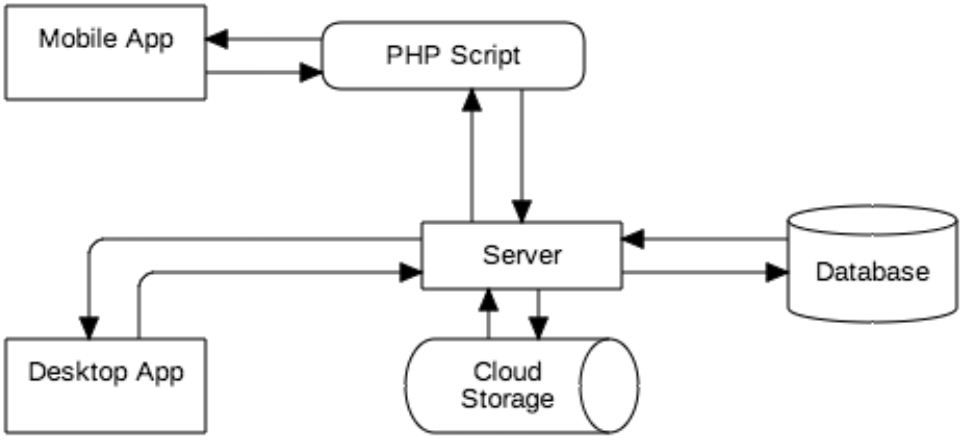

Figure 3.3 - More complex Client Server model

The purpose of the Layered Architectural pattern is to organise the structure of a system into distinct layers of related responsibilities, with a clear separation of concerns between each layer to encourage the decoupling of components (Larman, 2005). This has already been achieved by decoupling the Android application and the Server through the use a the PHP script. The Android application can only access the server via the PHP script and vice-versa ensuring these concerns are decoupled. However, if we want to achieve this pattern across the whole system architecture, then more layers have to be introduced.

The server, which hosts the database and cloud storage, is contained in a layer called Core Services. Technically the PHP scripts are hosted by the server as well but, because of their unique functionality, we can place them into their own layer called Technical Services. The remaining layers can be found inside each of the Android and Desktop application's source code. Both versions of the software solution will be designed in similar ways as they both will include User Interface, Application and Domain layers. These layers consist of classes that fall into each respective category. The User Interface classes handle the creation and display of the what the user will see. The Application layer handles the workflow, such as the creation of data for presentation, while the Domain layer is responsible for handling the application's data. An event, usually trigged by an the user's action in the user interface, will send a request of data to the server. This data must pass through each layer sequentially to ensure that the layered pattern has been achieved properly and that a decoupling of components has taken effect (*Figure 3.4*).

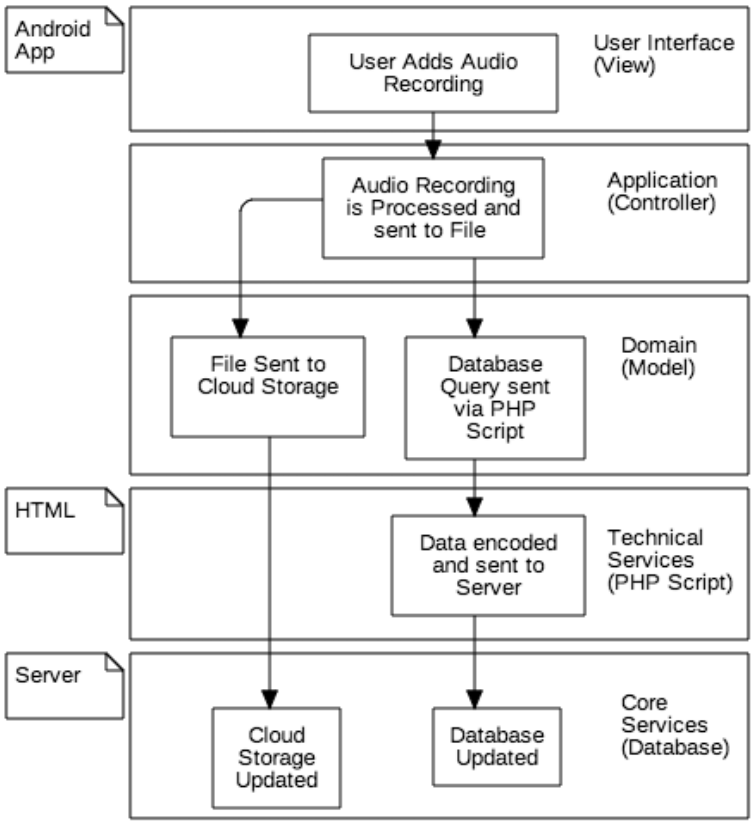

Figure 3.4- Flow of data through the layers of the system

Furthermore, the three layers found inside each application, can be incorporated into the Model-View-Controller (MVC) pattern, a pattern that enforces the separation of the model (Domain) from the view (User Interface). It is not unusual to use the Layered Architectural Style in tandem with the MVC pattern and the benefits of doing so as such are :

- **Abstraction** - Layers allow changes to be made at an abstract level.
- **Isolation** - Allows you to isolate upgrades to individual layers in order to reduce risk and minimise impact on the overall system.
- **Manageability** - Separation of core concerns helps to identify dependencies, and organises the code into more manageable sections.
- **Performance** - Distributing the layers over multiple physical tiers can improve scalability, fault tolerance, and performance.
- **Reusability** - Roles promote reusability. For example, in MVC, the Controller can often be reused with other compatible Views.
- **Testability** -Increased testability arises from having well-defined layer interfaces.

("Microsoft Developers Guide | Chapter 3: Architectural Patterns And Styles”)

These qualities should ensure that when it comes to the extension and maintenance of the software solution, those tasks will be much more easily carried out. In general terms, The Client Server model qualifies the system design to meet the project specifications which state that the user’s data must be available to the user on any device at anytime. And the Layered pattern, in tandem with the MVC pattern, ensures that the development team can focus on one layer of the application at a time, because each layer will be isolated from the other, which is keeping in line with the Kanban methodology chosen for the software development lifecycle.

3.4 User Interface Design

The design of this software relates to the project specification, which determines that information should be communicated to the user as quickly and efficiently as possible. This is achieved by using a minimal and refined style, with the primary purpose of presenting data in a clear and concise manner.

3.4.1 General Design

This section focuses on the overall theme or style of the software deliverable. The style was designed to be consistent across all devices on which the application appears, in order to successfully promote the software's brand. The design's two main objectives, outside of helping the software meet the requirements, are to affiliate the software with Queen's University Belfast, and to convey the purpose of the application.

Despite the fact that this solution is not officially a registered product of Queen's University Belfast, it will be built to with the aim to be used exclusively by students of the university. It was therefore important to link the application with Queen's University through design choices. This was done by adapting colours used on the Queen's University seal (*Figure 3.5*) and making them the primary colour scheme of the design. Additionally a free-to-use font, Quicksand, was sourced to help recreate the distinctive font used by Queen's in their logo (*Figure 3.6*).

Figure 3.5 - Queen's University seal

Figure 3.6 - Queen's University logo

The software solution's name, *QuDue*, was chosen for it's connotation to the university, with the name itself is an abbreviation and concatenation of 'Queen's University' and 'Due Date'. The new logo was created which clearly references the source material but at the same time creates a stylish and unique emblem to represent the *QuDue* brand (*Figure 4.3*). The 'Q' symbolises the 'Q' in Queen's University and the 'Pen' icon conveys the essay tool component of the application.

Figure 4.3 - QuDue logo

Just as the pen in the *QuDue* logo hints that this application involves writing or editing, it is important that the style choices used throughout the design reflect the type of application that the software is. *QuDue* is an academic tool aimed at student's who generally fall within the 18-30 years old demographic. There was no need to include bright, primary colours or playful graphics and fonts in the design. Formal language is used throughout, alongside a minimal and practical design. The main graphical image used is a tiled background which appears in the title bar across the desktop application (*Figure 3.7*). This graphic (created specifically for this application) reinforces the academic nature of *QuDue* whilst maintaining the refined style.

Figure 3.7 - Title bar design displaying title and background graphic

It was also the responsibility of the design to ensure that the application falls in line with the ethos of Queen's University Belfast, considering that the software will be, indirectly or not, promoting and representing the academic institution. Thus there was an avoidance of using inappropriate language and images in the design. As the development process begins, each component will include their own design phase in keeping with the agile approach used. These design choices represent the style or theme which should be used throughout each of these components.

Chapter 4
Development

In this chapter, several work items will be used as case studies to provide details of the development process. Each work item will include it's own requirements analysis, design and implementation sections. This is done to represent the workflow followed by the agile approach used during the software development lifecycle of this project.

4.1 Desktop Application

The Desktop Application was developed in Java using the Eclipse integrated development environment (IDE), version 4.6.0 (Neon), and the JavaFX software platform. The application was tested using a MacBook Pro (Mid 2012) running on OS X El Capitan (version 10.11.1).

4.1.1 Essay View Window

The Essay View Window is the main display shown when a user has selected an essay from the Home Screen in the Desktop Application. This window operates a tabbed navigation display, with each selectable tab changing the view to represent a different section of the essay.

4.1.1.1 Requirements Analysis

The requirements relate to the problem specification, and specifically how the original solution makes it difficult for users to find the information they want without the need to navigate through several pathways of a software system. This window solves the problem by making everything available to the user with just on the one screen thus reducing complexity and maximising efficiency. The Essay View Window meets the following functional requirements (the full requirements, including complete function definitions and error conditions can be found in the appendix):

EVFR1 - User shall be able to navigate to the Home Screen.
EVFR2 - User shall be able to open the Add Essay window.
EVFR3 - User shall be able to open the Account window.
EVFR4 - User shall be able to select any unselected tab to change the window's view.
EVFR6 - User shall be able to Logout.

4.1.1.2 User Interface Design

In order to retain the minimal, clean style of QuDue, the main focus of the desktop design was to maximise the space without making the screen feel cluttered. It was also to be designed in such a way that the screen became compartmentalised so that, dependent, on the user's actions, only certain elements of the view would change, instead of creating a brand new view. The Essay View Screen (*Figure 4.1*) shows how these ideas were put into practice. This screen has been designed in such a way that there is a lot of functionality available to the user with just a single button click, while the main role of the page (i.e. the functional requirements related to the selected tab) is still at the forefront of the display.

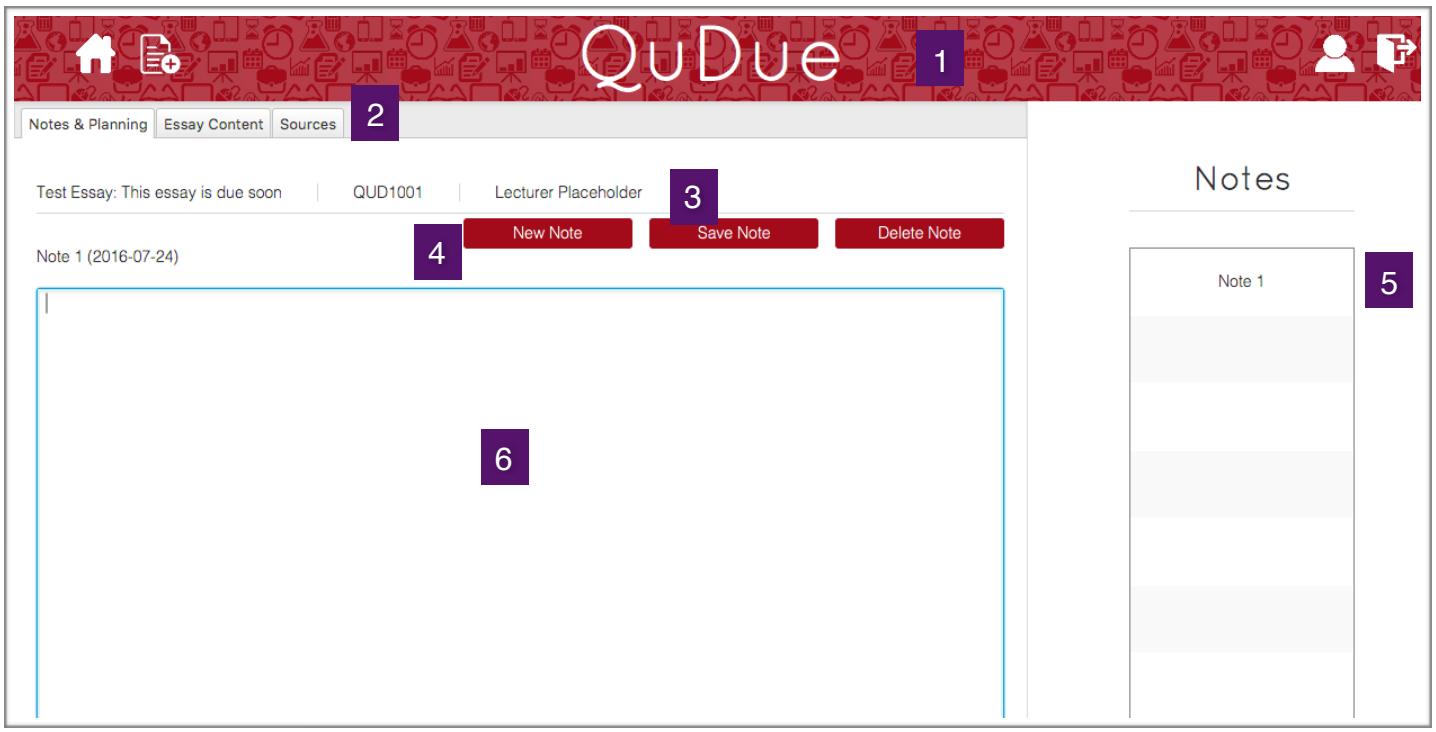

Figure 4.1 - Essay View Screen with annotations

The Essay View Screen has six main components:

1. **Toolbar** (*Figure 4.1.1*) : The Toolbar or Title Bar is consistent throughout the application so that at any time the user can view the title or access any of the Toolbar icons. From left to right these icons are; *Home* (navigates the user to the homepage), *Add Essay* (displays the Add Essay window), *Account* (allows the user to view their account details and change their password) and *Logout* (allows the user to log out of the application).
2. **Tabs** (*Figure 4.1.2*) : These tabs allow the user to quickly navigate to the other elements of the essay (*Essay Content* and *Sources*).
3. **Essay Information Bar** (*Figure 4.1.3*): Independent of the selected tab, information will be available detailing the essay title, essay module and module tutor. Having this information bar as a static element of the view removes the need to search the application for this information.
4. **Note Action Buttons** (*Figure 4.1.4*): These buttons, which are displayed when the Notes tab is selected, allow the user to add, save and delete notes. If an action is not available (i.e. no note is selected or current note is not editable) then one or more of these buttons will become disabled and will appear greyed out. This allows the application to communicate, in a very simple way, to the user which actions can and cannot be taking at any time.
5. **Tab Information Panel** (*Figure 4.1.5*): The right hand panel of the screen is made up of the page's title and a display which changes depending on the selected tab. If the Notes tab is currently selected, the right hand panel is populated with the Note List View display; a selectable list of all the notes available for this essay.

6. **Main Display** (*Figure 4.1.6*): The Main Display takes up the majority of the page in order to focus the user's attention, as this is where the main functionality is incorporated. The Main Display is dependent of the currently selected tab and, if the Notes tab is selected, the currently selected note. If either one of the Essay Content tab or Sources tab are selected then this display will be populated by a Text Area, a view where the user can enter and edit text. If a Text Note is selected while the user is on the Notes tab then this editor will also appear. The editor will be in focus (a blue highlight) and the cursor will appear to let the the user know that they can write in this area. If either a Photo Note or Audio Note is selected then this view will be modified to display images or playback audio respectively.

4.1.1.3 Software Design

To adhere to good Object-Oriented Programming (OOP) principles, the code to create the Essay View Window should promote loose coupling by defining a controlling object which encapsulates how the Tab objects in the window view interact with each other. The Mediator design pattern is best suited to this problem as it allows the code to be designed in such a way that the Tab objects never communicate with each other explicitly, thus allowing the program to vary their interaction independently. A pure fabrication object, in this case *TabManager*, must be created in order to control these interactions.

Each Tab object is defined by the *EssayViewTab* data model (*Figure 4.9*). The *TabManager* class instantiates each *EssayViewTab* object and provides methods to register and select each tab (*Figure 4.10*).

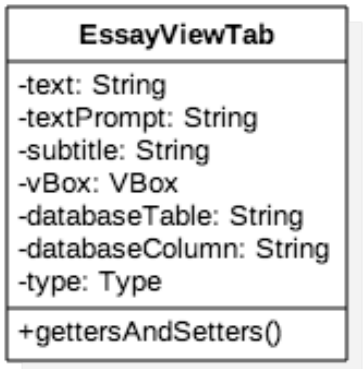

Figure 4.2 - EssayViewTab object

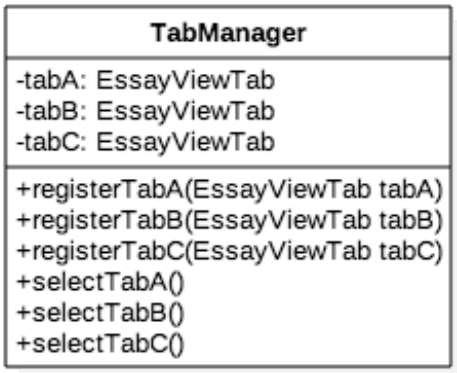

Figure 4.3 - TabManager class diagram

The *EssayView* class can then call *TabManager* to register and select each Tab. This ensures that the Tab objects never directly interact with each other, preventing unnecessary coupling between each object. The main benefit of this design pattern is that, in the future, if the requirements were to change and a new tab was to be added, then there would be much less refactoring of the code as a new Tab object could be created without having to change any code belonging to the existing Tab objects.

4.1.1.3 Implementation

As we can see from the requirements and the design, the Essay View Window has two main functions:

1. Allow the user to operate the selected tab's functionality.
2. Allow the user to quickly access the other functionality of the system.

Both these features depend on the tabbed navigation display implemented by this window. The tabbed navigation view allows the user to clearly see the components available to them and easily switch between these components. During the software design for this work item, the Mediator design pattern was identified as being able to achieve this implementation. The *EssayView* class is the class which provides the code to create and display the Essay View Window. This class is responsible for the tabbed navigation element of the program and as such uses the *TabManager* class to manage the Tab objects. The following code (*Figure 4.4*) focuses on the Main Display component of the window view as this is where the view dynamically changes dependent on which tab is selected.

```
/**
 * Sets the central display for the window.
 * @return TabPane with the four tabs set up
 */
private TabPane display() {

    // set up new TabPane
    tabPane = new TabPane();
    tabPane.setId("border-pane");

    // Set up tabA - Notes
    tabA = new Tab("Notes & Planning");
    tabA.setClosable(false);
    tabManager.registerTabA(essayViewTab);
    tabManager.selectTabA();
    tabA.setContent(tabContent());
    tabPane.getTabs().add(tabA);
```

Figure 4.4 - Method in the EssayView class to create the main display

Here a new *TabPane* object is created. This object creates the tabbed navigation element of the view. Tabs, of the type *javafx.scene.control.Tab*, are used by this TabPane, so our code has to link this type of tab with the *EssayViewTab* object which we want the tabs to appear as. The *TabManager* is used to achieve this. Firstly, we register an empty *essayViewTab* object to the *TabManager,* calling it *TabA.* Then we select this *TabA*. The method, *selectTabA*(), takes the empty *essayViewTab* (*Figure 4.5*) and sets the object's attributes based on what we want this tab to display, in this case the Notes Tab View. The *setVBox* method calls the *Notes()* method which is responsible for creating the display for the Tab Information Panel (*Figure 4.1.5*).

```
public void selectTabA() {

    tabA.setDatabaseTable("Notes");
    tabA.setDatabaseColumn("note_content");
    tabA.setSubtitle("Notes");
    tabA.setType(Type.NOTEVIEW);
    tabA.setVBox(Notes());
}
```

Figure 4.5 - selectTabA() method in TabManager

The line of code in *Figure 4.4* which sets the tab's content, calls a new method in the *EssayView* class called *tabContent()*. This method creates a generic GridPane, which has it's main display created by the called *tabDisplay()* method (*Figure 4.6*).

```
/**
 * Creates either a textArea or tableView depending on selected tab
 * @return node - textArea or tableView
 */
private Node tabDisplay() {

    Node node = null;

    // if selected tab type = TEXTAREA then creates textArea
    // if selected tab type = TABLEVIEW then creates tableView
    switch (essayViewTab.getType()) {
    case TEXTAREA:
        // calls method to create textArea
        node = makeTextArea();
        break;
    case NOTEVIEW:
        // calls method to create tableView
        node = makeNoteView();
        break;
    default:
        node = makeTextArea();
        break;
    }

    return node;
}
```

Figure 4.6 - tabDisplay() method in EssayView class

This method switches on the enum *Type* which is defined in the *essayViewTab* object. This object was created by the *TabManager* class. As *TabA* was the last tab to be called by the *TabManager* then the *Type* of the *essayViewTab* is the *Type* of *TabA*. In this case, *TabA* is the Notes tab and the *Type* equals *NOTEVIEW,* so as a result, the *makeNoteView()* method is called. The Node created is then returned through the method call stack to eventually populate the main display of the window. An example of the retuned view can be seen in *Figure 4.1.* Overall, the tabbed navigation allows the view to change dynamically dependent on which tab is selected, allowing the user to easily navigate the software solution. However, aside from the navigation, the implementation must also allow the user to fulfil certain requirements.

In this case, where the *Notes & Planning* tab has been selected, these requirements include viewing, adding, editing and deleting notes. These functions can be called by accessing the different components of the user interface. A note can be selected from the Note List View, on the right side panel of the window. Once a note is selected here, the Main Display of the Essay View Window will dynamically change, depending on the type of note selected. This process follows on from the *makeNoteView()* method call shown in *Figure 4.6*. This method creates a new BorderPane which will be used to fill the main display. If no note is currently selected, the centre of the border will contain a placeholder label telling the user to select a note from the note list. If a note is selected, the selected note is passed into the *NoteView* class (*Figure 4.7*).

```
/**
 * Method to dynamically populate the note view
 * display based on the note selected
 * @param note
 * @return the Node to be displayed
 */
public static Node makeNoteSelectedView(Note note) {

    //set a string to equal the note type
    String type = note.getType();

    //instantiate the new Node
    Node node = null;

    //switch on note type
    switch (type) {
    case "TEXT":
        //call TextNoteView
        node = TextNoteView.getView(note);
        break;
    case "PHOTO":
        //call PhotoNoteView
        node = PhotoNoteView.getView(note);
        break;
    case "AUDIO":
        //call AudioNoteView
        node = AudioNoteView.getView(note);
        break;
    }

    return node;
}
```

Figure 4.7 - makeNoteSelectedView() method in NoteView class

The *makeNoteSelectedView()* method switches on the Note type of the passed in Note. Dependent on the note type, different methods are called to create a view. This view is returned to the EssayView window and used to populate the BorderPane which in turn is used to populate the Main Display of the window. In *Figure 4.1,* it is clear that a Text Note has been selected because the Note Editor appears. This Note Editor is created in the *getView*() method in the *TextNoteView* class. The Note Editor view consists of a TextArea where the selected Note's content appears. This content can be edited and then saved. An important element of this method is the listener attached to the TextArea (*Figure 4.8*).

```
// add change listener
textArea.textProperty().addListener(new ChangeListener<String>() {

    // if the text changes then..
    @Override
    public void changed(ObservableValue<? extends String> observable,
            String oldValue, String newValue) {
        //enabled the save button
        EssayView.saveButton.setDisable(false);
        //update the selected note's value
        EssayView.selectedNote.setContent(newValue);
    }

});
```

Figure 4.8 - Property listener attached to the TextArea in the getView() method

This code recognises when the content in the TextArea displayed in the Essay View Window is changed. If the text is edited then two things occur; the Save Button becomes enabled, indicating to the user that this option is now available to select, and the selected Note's content is updated, ensuring that when the user selects the Save Button they are sending the most up to date information to the class which handles updating the database.

4.1.1.4 Testing

As an agile software development methodology was used in the development of this project, testing occurred throughout the software life cycle. In terms of the Kanban method, product quality is dependent on the skills of the developer and the immediate correction of any errors (Linz, 2014). This means that testing takes place on each work item as the development is ongoing. As each work item represents the lowest level of the development, the corresponding testing level is Component Testing. This test level consists of searching for defects in the classes and methods of the executable code. This can be achieved via Black-box (Functional) and White-box (Structural) testing techniques. In this project, due to the one-person team, this testing was carried out by the developer.

The White-box testing was carried out in ad-hoc fashion while the developer was writing the code. Once a specific method or piece of code was complete, the developer would execute the code and if an incident occurred, they would locate the defect, fix the defect and then re-run the code to ensure the problem was solved. This testing technique was carried out using the IDE's debugging tool and involved no formal incident management.

The Black-box testing was carried once the work item was complete. A set of test conditions and cases were created from the requirements analysis of each work item and a Test document was produced which identified the steps required to test the component. In the cases where a component was tested before other related components had been completed, it was necessary to introduce testing stubs and drivers.

This functional testing strategy was carried out on the Essay View Window and the following documentation represents the test area involving the Notes Tab and how the view responds when a each Note type is selected .

Test Basis (General for the Essay View Window)
Verify the user can navigate to the Home Screen by selecting the *Home* icon in the toolbar.
Verify the user can open the Add Essay window by selecting the *Add Essay* icon in the toolbar.
Verify the user can open the Account window by selecting the *Account* icon in the toolbar.
Verify the user can logout by selecting the *Logout* icon in the toolbar.
Verify that the user can select any unselected tab to open that tab's view.

Test Basis (Specific to the Note Tab being selected)
Verify the user can select a note from the Note List View.
Verify the user can delete a selected note.
Verify the user can add a new Text Note.
Verify the user can edit a Text Note.
Verify the user can save an edited Text Note.
Verify the user can view the image of a Photo Note.
Verify the user can playback the audio of an Audio Note.

Test Conditions
1. When no note is selected the Main Display is empty aside from the placeholder text and the only button available to select is the Add Note button.
2. When a Text Note is selected the view changes to the Text Note View and the Delete button becomes available.
3. When a Photo Note is selected the view changes to the Photo Note View and the Delete button becomes available.
4. When an Audio Note is selected the view changes to the Audio Note View and the Delete button becomes available.
5. When a Text Note is edited the Save button becomes available.
6. When Add Note button is selected, new note is added and becomes available to select in the Note View List.
7. When Delete Note button is selected, selected note is deleted and removed from the Note View List.
8. When Save button is selected, the Text Note's content is saved.

Test Cases		
ID	**Test Data**	**Expected Result**
N1	No selected note	Main Display empty aside from placeholder text Add Note button is available Delete Note button is unavailable Save Note button is unavailable
N2	Text Note selected from Note View List	Main Display shows Text Note View Add Note button is available Delete Note button is available Save Note button is unavailable
N3	Photo Note selected from Note View List	Main Display shows Photo Note View Add Note button is available Delete Note button is available Save Note button is unavailable
N4	Audio Note selected from Note View List	Main Display shows Audio Note View Add Note button is available Delete Note button is available Save Note button is unavailable
N5	Text Note edited in the Text Note View	Save Note button is available

Test Results		
Test Case ID	**Actual Result**	**Pass/Fail**
N1	Main Display empty aside from placeholder text Add Note button is available Delete Note and Save Note buttons unavailable	PASS
N2	Main Display shows Text Note View Add Note button is available Delete Note button is available Save Note button is unavailable	PASS
N3	Main Display shows Photo Note View Add Note button is available Delete Note button is available Save Note button is unavailable	PASS
N4	Main Display shows Audio Note View Add Note button is available Delete Note button is available Save Note button is unavailable	PASS
N5	Save Note button is available	PASS

4.2 Android Mobile Application

The Android Application was developed using the Android Studio IDE, version 1.4. The minimum SDK version of the application is 19 (Android 4.4 - KitKat). The target and complied SDK version is 23 (Android 6.0 - Marshmallow). The application was tested on a Nexus 5 device, running Android 6.0.1.

4.2.1 Home Screen

The Home Screen is the first screen the user will see once they log into the mobile application. This screen should display a timetable of the user's current essays as well as provide an option to a add new essay.

4.2.1.1 Requirements Analysis

The high level requirements for this software solution state that the user shall be able view a timetable of their essays on both the Android and Desktop device (Chapter 3.2.1 - *FR9*). This will make it possible for the user to view their essays anytime and anywhere, provided they have installed the application onto their mobile device. The Home Screen should provide instant access to the user's essay timetable, enabling them to see their upcoming deadlines with just a quick glance. This ensures that the following functional requirements (detailed in full in *Appendix B*) are successfully met:

HSFR1 - User shall be able to view a full timetable of their essays.
HSFR2 - User shall be able to select anyone of the essays from the essay list.
HSFR3 - User shall be able to open the Add Essay screen.
HSFR4 - User shall be able to Logout.

4.2.1.2 User Interface Design

Despite working with a much smaller screen, the android version of the software solution aims to provide much of the same functionality of the desktop version. Therefore it was to be designed in such away that this functionality could be displayed in full without compromising the stylistic choices of the overall design.

As per the requirements, the purpose of the Android Application is to allow the user to view their upcoming essays whilst on the go. It is then important, in terms of the design, that the user gets immediate access to this information and also that this information is presented in a very clear manner with emphasis placed on the most important details. This is achieved with the Home Screen User Interface (*Figure 4.9*).

This user interface is split into two components; the Toolbar and the Essay List View. The toolbar displays the *Title, Add Essay* and *Menu* icons. The *Title* icon reinforces the QuDue brand, the *Menu* icon enables the user to log out of the application and the *Add Essay* icon opens a new page. These icons are placed in the Toolbar so that the functionality that they offer is readily available to the user when using the application but the icons themselves take up relatively little space on the screen. The *Add Essay* and *Menu* icons are not labelled with text as their functionality is already implicit enough to be intuitively understood by the user.

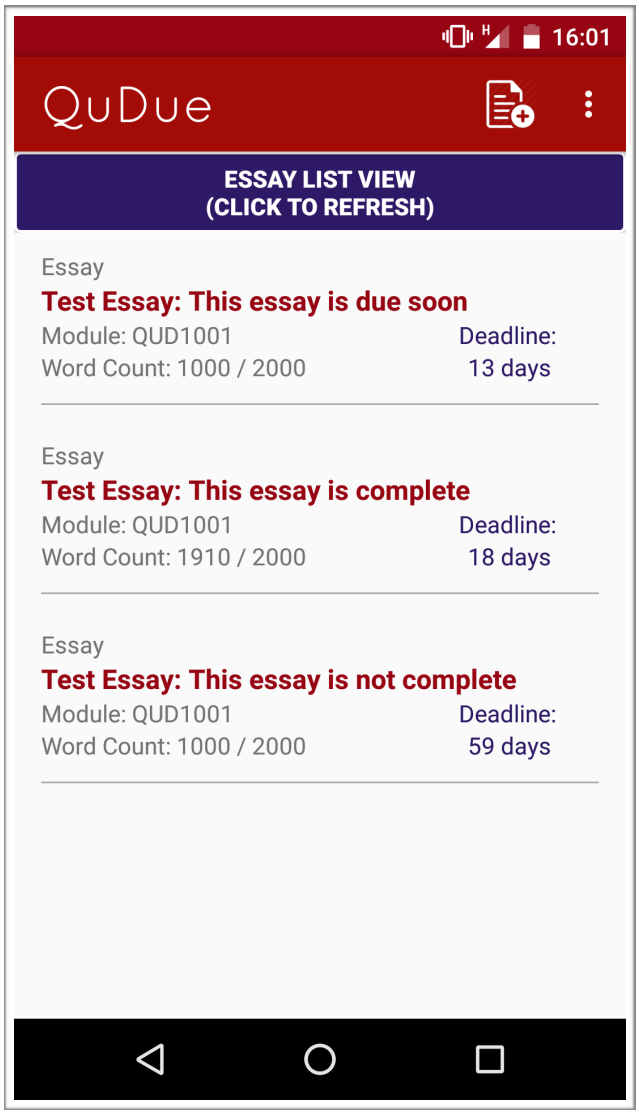

Figure 4.9 - Android Mobile Application Home Screen User Interface

The Essay List View takes up the main display on this screen as this is where the user's focus should be directed to. Here we see a Title Bar (*Essay List View*) which doubles as a Refresh Button. Below it, the Essays themselves are listed in a RecyclerView. The RecyclerView allows the application to list customised objects. In this case each row in the list displays the Essay Title, Module, Word Count and Deadline for each Essay. Already the user has all the basic information they require. The Essay List View is scrollable (if more than five essays are present) and is customised to sort the Essays by due date so the user can clearly see their next deadline.

4.2.1.3 Software Design

As detailed in the high level software system design, this development focuses on the implementation of a layered architecture with specific emphasis on the separation of concerns between roles. This separation of concerns involves decoupling User Interface (UI) objects with non-UI objects. To achieve this design pattern, the class which creates the Home Screen UI contains no code inside it's main thread which communicates with a class found in the Domain layer of the system's architecture. The call to the Domain layer is instead handled by an intermediary class. This controller class populates the essay list found in the Home Screen view, thus ensuring a loose coupling between layers is maintained. To illustrate this in terms of the software structure, *Figure 4.10*, demonstrates how the packages in the Android Applications java folder have been set up.

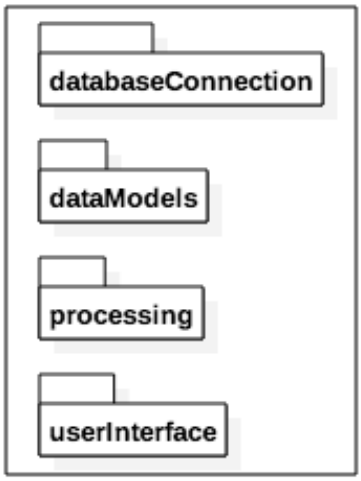

Figure 4.10 - Package Diagram for the Android Application

The *databaseConnection* folder equates to the Domain or Model layer, the *userInterface* package represents the View layer and the *processing* package is where the Controller/Application code is kept. The *HomeScreen* class will be placed inside the *userInterface* package as this class contains the code to create the new activity. To ensure that Domain and View layers remain decoupled, a *DataService* class will be created in the *processing* package. This class will be called by any *userInterface* class when they wish to send or retrieve any information from the server (i.e. access the Domain layer). The class which relates to the specific send or pull request can be found in the *databaseConnection* package. The information returned by this class is sent to a class in the *processing* package which is ready to handle this information. In this case, that class is called *EssayData.* The *EssayData* class places the retuned information inside a container which the *HomeScreen* class can handle, before sending it back to the user interface class to be displayed. This is an example of the Model-View-Controller design pattern in practice and ensures that a separation of concerns is maintained in this project's structure.

4.2.1.4 Implementation

The main component of the Home Screen is the essay list which acts as a timetable, displaying the user's upcoming deadlines. This list is created using a RecyclerView object. In the *HomeScreen* class a new activity is created with the following code used to set up the *recyclerView* (*Figure 4.11*). The *recyclerView* created here points to a view already created in the *home_screen.xml* layout file. The most important line is the last one which sets the adapter for the *recyclerView*.

```
//set up the recycler view
recyclerView = (RecyclerView) findViewById(R.id.recycler_view);
// use this setting to improve performance if you know that changes
// in content do not change the layout size of the RecyclerView
recyclerView.setHasFixedSize(true);
//set up the layout manager
RecyclerView.LayoutManager layoutManager = new LinearLayoutManager(context);
recyclerView.setLayoutManager(layoutManager);
//use the default animations
recyclerView.setItemAnimator(new DefaultItemAnimator());
//set the recycler views adapter to essayAdapter
recyclerView.setAdapter(essayAdapter);
```

Figure 4.11 - Code to set up the RecyclerView

The e*ssayAdapter* object is a public static object of the type *EssayAdapter*. The *EssayAdapter* class defines how each row will appear in the *recyclerView*. This object has been set before the *HomeScreen* activity is created to ensure that when the activity opens, the information will already be available to view. The information is set when the user successfully logs into the system. Upon user verification, a call will be made to the *DataServices* class, the controller class which handles UI calls to the Domain layer. A class called *EssayRepository* is called by *DataServices (Figure 4.12)*.

```
public void retrieveAllEssays(int id, String action) {
    essayRepository.execute(Integer.toString(id), action, retrieveAll);
}
```

Figure 4.12 - Calling EssayRepository from DataServices

EssayRepository, like all the classes located in the *databaseConnection* package, extends *AsyncTask*. This means that when the *execute*() method is called, this class will run as an asynchronous task (i.e. it will run in the background, outside of the main thread). The benefit of this is that the main thread does not get slowed down with these outside requests which take time to process. The program will run faster and more efficiently, with the the user not having to deal with the UI freezing up.

The first action taken when this async task is called, is a switch statement which switches on a String parameter passed in with the method call. In this case, the parameter indicates that the *login_url* variable is assigned a URL String which ends in "EssayData.php" (*Figure 4.13*).

```
@Override
protected String doInBackground(String... params) {

    //capture the first and second parameters passed in
    id = params[0];
    action = params[1];

    //switch on the third passed in parameter
    switch (params[2]){

        case "RETRIEVE_ALL":{
            mCase = params[2];
            //the url link to retrieve all essay data
            login_url = "http://ec2-52-50-204-225.eu-west-1.compute.amazonaws.com/EssayData.php";
            break;
        }
```

Figure 4.13 - Switching on the given parameter to assign the URL String

After a new HttpURLConnection has been opened with the given URL, the class is now ready to send information to the server-hosted database via an intermediary PHP script. The information we wish to send is encoded in the correct format and sent to the HTML page using a OutputStream object wrapped inside a BufferedWriter object (*Figure 4.14*).

```
//create a new output stream
OutputStream OS = httpURLConnection.getOutputStream();
//create a new buffered writer which in turn holds the output stream writer
//output stream writer specifies which decoding to use: UTF-8
BufferedWriter bufferedWriter = new BufferedWriter(new OutputStreamWriter(OS, "UTF-8"));
//set the data using URL encoder to ensure the Strings are in the correct format
//the date to be sent is the user's email and password
String data = URLEncoder.encode("id", "UTF-8") + "=" + URLEncoder.encode(id, "UTF-8");
//write the encoded data
bufferedWriter.write(data);
```

Figure 4.14 - Sending data to the server via a PHP script

The PHP script is necessary because Android does not communicate natively with MySql databases like Java does. This PHP script handles the database queries based on the information that it retrieves. It then returns the requested information my displaying data as HTML on the given URL. This data can then be read back into the application using an InputStream (*Figure 4.15*).

```
//create a new input stream
InputStream IS = httpURLConnection.getInputStream();
//create a new buffered reader which in turn holds the input stream reader
//input stream reader specifies which decoding to use: iso-8859-1
BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(IS, "iso-8859-1"));
String response = "";
String line;
//read in the response
while ((line = bufferedReader.readLine()) != null) {
    response += line; //the response is specified by the php script which created the html
}
```

Figure 4.15 - Reading data from the server via a PHP script

Once this operation is complete, the *postExecute*() method in the *AsyncTask* automatically runs. In this case, this method call the *EssayData* class and passes in the received information. The *EssayData* class is set up to handle and format this data. Firstly, because the data came from an HTML source, to split the data into each new entry the *.split()* function was used. The given String is

```
/**
 * Record audio using the AudioRecord class
 */
private void record() {
    //set the var isRecording to true
    isRecording = true;

    try {
        //open a new FileOutputStream wrapped in a BufferedOutputStream and DataOutputStream
        final DataOutputStream dos = new DataOutputStream(
                new BufferedOutputStream(new FileOutputStream(
                        outputFile))); //set the previously defined out file

        //create a new AudioRecord object, passing in the audio source, sample rate, channels,
        // encoding format and buffer size
        final AudioRecord audioRecord = new AudioRecord(
                MediaRecorder.AudioSource.MIC, RECORDER_SAMPLERATE,
                RECORDER_CHANNELS, RECORDER_AUDIO_ENCODING, bufferSize);
```

Figure 4.24 - Creating the DataOutputStream and AudioRecorder objects in the record() method

first split on “
”, as this syntax indicates a new line in HTML and each row received from the database table was printing onto it’s own line by the PHP script. Now that we have each line, we split on “$” because we know that this was the syntax used by the PHP script to distinguish between each attribute. An Essay object can now be created using the split data as long as we specify which attribute is which in the correct order. This whole process is shown in *Figure 4.16*.

```
String[] allEssaysString = result.split("<br/>");

for (int i = 0; i < allEssaysString.length; i++){
    String[] essayString = allEssaysString[i].split(Pattern.quote("$"));

    essayObj = new Essay();
    essayObj.setEssayId(Integer.parseInt(essayString[0]));
    essayObj.setEssayTitle(essayString[1]);
    essayObj.setWordLimit(Integer.parseInt(essayString[2]));
    essayObj.setWordCount(Integer.parseInt(essayString[3]));
    essayObj.setWordCountAndLimit();
    essayObj.setStartDate(essayString[4]);
    essayObj.setDueDate(essayString[5]);
    essayObj.setModuleCode(essayString[6]);

    allUsersEssays.add(essayObj);
}
```

Figure 4.16 - Formatting the retrieved data in the EssayData class

When the ArrayList of Essays is complete we can use the *Comparator* interface to sort this list in ascending order of the essay due date *(Figure 4.17*). The *compare* method from the *Comparator* interface, compares two objects and returns an integer. The integer will be less than zero if the first object is less than the second object, exactly zero if they are equal, and greater than zero if the second object is greater than the first.

```
Collections.sort(allUsersEssays, new Comparator<Essay>() {
    @Override
    public int compare(Essay e1, Essay e2) {

        Date date1 = CreateDateObject.create(e1.getDueDate());
        Date date2 = CreateDateObject.create(e2.getDueDate());


        return (int) ((date1.getTime() - date2.getTime()) / (1000 * 60 * 60 * 24));
    }

});
```

Figure 4.17 - Sorting the essay list by due date in the EssayData class

To get a value to compare we call the static *CreateDateObject.create()* method and pass in the essay object due date. In this method, the date String parameter, in the standard *dd-mm-yy* format, is converted into a java.util.Date object and retuned to the *EssayData* class. This Date object allows the *.getTime*() method to be called which returns the number of milliseconds since Jan. 1, 1970, midnight GMT. This process is applied to the two essay object's which we wish to compare. The millisecond value of the first object is then subtracted from the millisecond value of the second value. The difference is then divided by 86,400,00 (1000 * 60 * 60 * 24) to convert from milliseconds to days. The value returned equates to the number of days between deadlines of the first and second essays. They are then placed accordingly back into the array list with the essay due sooner placed first. This process is carried out on every essay object until the whole list is sorted. The finalised list, *allUsersEssays*, is used to instantiate the HomeScreen's public static *essayAdapter (Figure 4.18)*. Therefore, when the *HomeScreen* activity is opened the RecyclerView will be created using this *essayAdapter* which uses the *allUserEssays* list, which in turn displays all the user's essays in ascending order of due date thus fulfilling the functional requirement.

```
HomeScreen.essayAdapter = new EssayAdapter(allUsersEssays);
```

Figure 4.18 - Setting the HomeScreen's essayAdapter from the EssayData class

4.2.1.4 Testing

In order to test that this process was working correctly Black-box component testing was undertaken.

Test Basis
Verify that, if essays exist, they appear when the HomeScreen activity is opened. Verify that these essays are ordered by ascending due date.

Test Cases		
ID	**Test Data**	**Expected Result**
HS1	List of essays with different due dates	The essays will appear ordered by due date

Test Results		
Test Case ID	**Actual Result**	**Pass/Fail**
HS1	The essays appear ordered by due date	PASS

4.2.2 Note Taking Functionality

4.2.2.1 Requirements Analysis

The high level requirements state that the user shall be able use their android device to add text, image and audio notes to each essay using this application. This note taking ability is an important function as it allows the user to take notes whilst on the go and have them organised by essay. The user will then be able to refer to these notes when writing their essay by accessing either the android or desktop version of the app. Including this functionality with the software solution, means the user doesn't have to use a third-party application to take and organise their notes. Having all the information available on one system, and thus already organised, will save the user time and energy. To negate the usage of third-party application, it was important for the project to design and implement this functionality in a professional manner. This is why the ability to add audio and images was included in the software requirements. This additional functionality gives the user more options when taking their notes, as they will be able to quickly take a picture of a powerpoint slide in the middle of a lecture or record a short audio clip of their tutor giving important information. The functional requirements met by this component include:

FR5 - User shall be able to add text notes to each essay.
FR7 - User shall be able to add notes as images.
FR8 - User shall be able to add notes as audio recordings.

This requirements are listed in full in the appendix (*Appendix B*), with full function definitions and details of error conditions.

4.2.2.2 User Interface Design

The note taking functionality can be found in the *TabbedNavigation* activity of the android application. This activity is introduced whenever an essay is selected from the *HomeScreen*. The *TabbedNavigation* loads four different fragments, Overview, Notes, Content and Sources, with each one representing their respective section of the essay. These fragments are loaded by the user swiping either left or right. Once the user accesses the Notes tab they will be presented with another RecyclerView, with this one containing a list of all the notes related to the selected essay. Additionally, once this tab is accessed, a Floating Action Button (FAB) will appear in the bottom right corner of the screen (*Figure 4.19*). This FAB, iconised with a '+' symbol implies that a new note can be added. Once selected, a Floating Action Menu will appear offering the user three options; *Text Note*, *Photo Note* and *Audio Note*. These three options are each represented by their respective icons (*Figure 4.20*). The FAB's icon will transform into an 'x' symbol letting the user know that this button can now be re-selected to close the Action Menu.

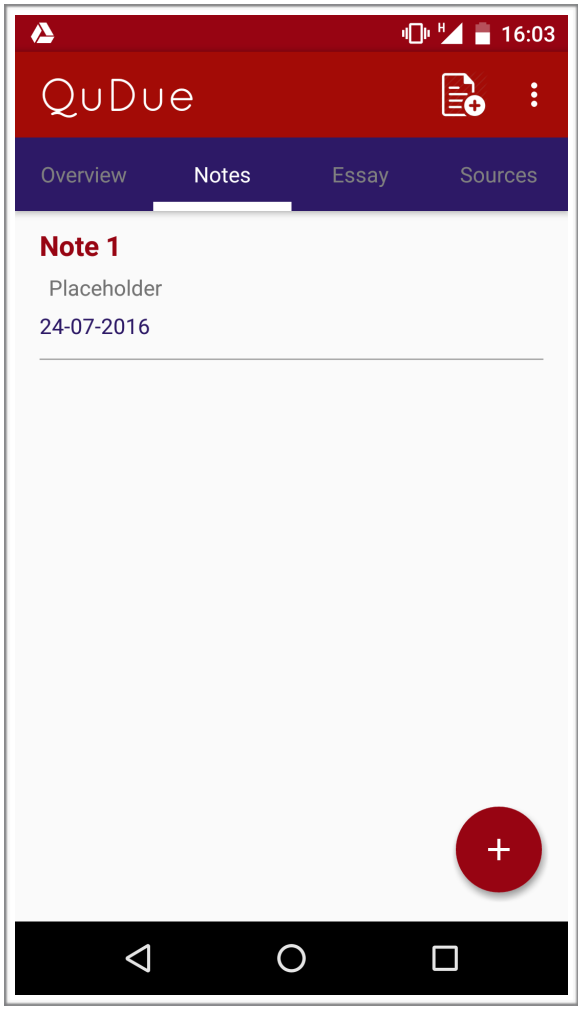

Figure 4.19 - Notes Tab with closed Floating Action Menu

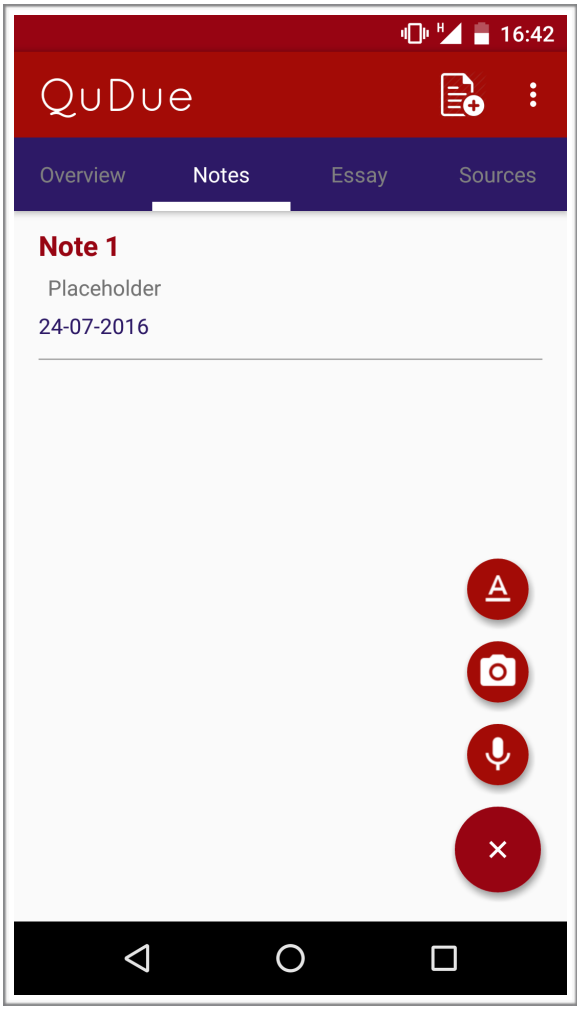

Figure 4.20 - Notes Tab with open Floating Action Menu

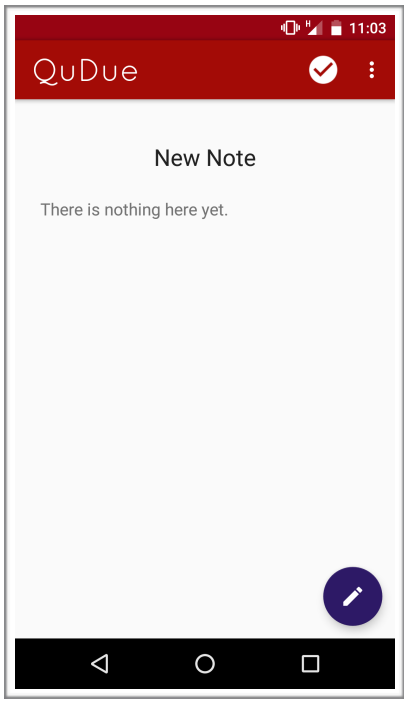

Figure 4.21 - TextNote View

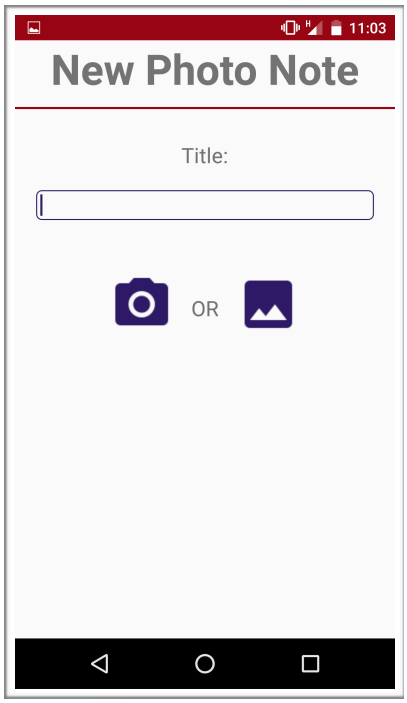

Figure 4.22 - PhotoNote View

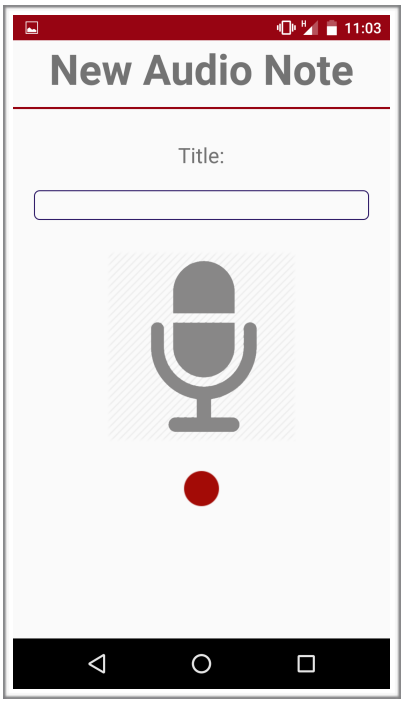

Figure 4.23- AudioNote View

This menu, along with the Note List View, enables the user to view and add new notes with extreme ease and efficiency. The FAB offers an intuitive and stylish way to access a menu without having to leave the screen, and the use of icons minimises the descriptive text, allowing the user instead to focus on their notes. Each option of the menu, when selected, will open a new activity related to the action of that specific button. The *TextNote* view offers a clean and effective way to write, edit and save text notes (*Figure 4.21*). The design here, mirrors the note taking design found in popular note-taking android apps, such as Google Keep. The PhotoNote (*Figure 4.22*) and AudioNote (*Figure 4.23*) views are similarly minimal and make use of icons over descriptive text. The idea is that the target demographic are proficient enough will mobile technology to understand these images and icons and, even if they are unfamiliar with android applications, the views should be designed in such a way to be user-intuitive .

The *TextNoteView* activity offers a large text area where the user can edit their note. The edit function is enabled by selecting the Edit Floating Action Button. Once selected, this button will disappear as it will become redundant. A cursor will appear signifying to the user that the text can be edited. The *Toolbar* is still active here as the *TextNoteView* activity is considered a main activity of the application and one which the user may spend a lot of time on.

The *PhotoNoteView* activity displays a text area where the user can enter a title for the note. Below this there are two icons which the user has to choose from. The option of the left, iconised by the *Camera* icon, will open the camera hardware on the mobile device allowing the user to take a picture which they can then save as a note. The option on the right, iconised by the *Gallery* icon, indicates that the user can select an image from their devices picture gallery. When an image is selected from either of these two views, a preview of it will appear in the empty space beneath the icons. This will be

accompanied with a Save button, which the user can select when they are satisfied with their photo note.

The *AudioNoteView* activity also displays a text area when the user can enter a title. Below this field there is a large *Microphone* icon. This icon can not be selected but instead signifies the purpose of this activity without the need for text labels. A red recording button is available beneath this, with this style a recognised indicator that recording can be enabled on selection. Once this button is selected, it will be replaced by a Stop button, iconised with the universal symbol for stop (a solid square). Selecting this when then lead to a Play button and Save button becoming available. The user will be able to play back their audio recording and then save it once they are satisfied.

4.2.2.3 Software Design

This software component is being designed within a tabbed navigation system. This layout reduces the number of activities that the user has to navigate through, making for a better and more streamlined user experience. Additionally, because each tab is loaded as a Fragment instead of a brand new Activity, we are able to modify the activity's appearance at runtime and preserve those changes in a back stack that's managed by the activity ("Fragments | Android Developers"). The *TabbedNavigation* class is the activity which hosts the fragments. This class will call the *TabsPagerAdapter*, the class which handles which fragment will open when each tab is selected. Here new instances of each tab fragment are opened when the specified tab becomes selected. This architecture successfully decouples the *TabbedNavigation* class from each of the fragments which populate it’s view. The purpose of doing so ensures that the code conforms to the Open-Closed Principle (OCP), one of the SOLID design principles for good Object-Orientated Programming.

The Open-Closed Principle (OCP) states that code should be open for extension and closed for modification, essentially meaning that new data-structures can be added to the system without the need to modify the base code (Knoernschild, 2002). The software design for this component ensures that if, in the future, a new tab was to be added to the tabbed navigation element then none of the previous code would need to be modified. Instead the new Fragment would be created in a new class and a call to this new class would be added to the *TabsPagerAdapter* class. Then when *TabbedNavigation* creates the new instance of *TabsPagerAdapter*, the new tab will be included without ever having to modify code in the *TabbedNavigation class*.

4.2.2.4 Implementation

This implementation section will only cover a specific section of code in order to provide a high level of detail about how a particular requirement was achieved. That requirement is the function to add a new note as an audio recording. This functionality occurs in the *AddNewAudioNote* class which is created when the Audio Note icon is selected from the Action Menu in the Notes Tab of the *TabbedNavigation* activity. Despite this code dealing with functions which should appear in the Application layer of the software structure, these functions have been included in this activity class because they affect the User Interface.

The *record*() method, which is called when the Record Button is selected, contains the code which handles the recording of the audio. Firstly, two objects have to be created to allow this to happen (*Figure 4.24*). The first is a FileOutputStream which is to be wrapped inside a BufferedOutputStream and a DataOutputStream. An output file, with the extension *.pcm,* must be specified as this is where the data will be saved to. PCM (Pulse Code Modulation) is a common method of storing and transporting uncompressed digital audio that most applications can read (Fries & Fries, 2005). This output stream is how the raw audio recording is saved to a file. The second object is an instance of the *AudioRecord* class. As per the class documentation, this class manages the audio resources for Java applications to record audio from the audio input hardware of the platform. In this instance we are using the mobile device's hardware to capture the audio. This is indicated by the *MediaRecorder.AudioSource.MIC* parameter. The other parameters included in the constructor for this method include the audio's sample rate (44100), the channel (mono), the audio format (16-bit, PCM) and the buffer size. The buffer size has been calculated earlier using the *AudioRecord.getMinBufferSize()* method. With the *AudioRecorder* object now set up, we can begin recording but first we change the UI to reflect that the audio recording is now taking place.
The changes to the User Interface include; changing the *Microphone* icon to a *Microphone-On* icon, replacing the Record Button with a Stop Button, and displaying a message to the user that audio recording is taking place (*Figure 4.25*). Once these changes have taken place we can start recording.

```
//show the new mic image view (with red dot)
mic.setImageResource(R.drawable.microphone_icon_record);
//hide the record button
record.setVisibility(View.INVISIBLE);
//show the stop button
stop.setVisibility(View.VISIBLE);
//show toast that recording has started
Toast.makeText(getApplicationContext(), "Recording started", Toast.LENGTH_SHORT).show();
```

Figure 4.25 - Changing the AddAudioNote UI to reflect audio recording is ongoing

The actual recording must take place in a new thread, outside of the main UI thread (*Figure 4.26*). This is because the device's hardware is being accessed. In this new thread, an array of data type short is created with length of the previously set buffer size. The short data type is used because it is a 16-bit type, the same audio format we are using. The *startRecording* method is then called on our *audioRecord* object. While the audio is being recorded we use the *audioRecord.read()* method to read the audio data into our short array. This method returns the total number of bytes read as an integer. We assign this integer to a new variable and then create a for loop with this new variable as the termination value. With each increment of this loop, we use our DataStreamOutput object's *.writeShort*() method to write each element of the short array to our output file. When this is finished the *audioRecord* object is stopped and the *dataOutputStream* closed. The audio is now saved to a file ready to be opened or transported to a different device.

```
//create a new thread (outside of main UI thread)
Thread recordingThread = new Thread(new Runnable() {
    public void run() {

        try {
            //create new array of data type short
            short[] buffer = new short[bufferSize];
            //start recording
            audioRecord.startRecording();

            while (isRecording) {
                //read audio data from hardware into buffer array
                int bufferReadResult = audioRecord.read(buffer, 0,
                        bufferSize);
                //loop over number of bytes read
                for (int i = 0; i < bufferReadResult; i++) {
                    //write each element of buffer to file
                    dos.writeShort(buffer[i]);
                }

            }
            audioRecord.stop();
            dos.close();
        } catch (IOException ex) {
        }
    }
});
recordingThread.start();
```

Figure 4.26- Recording the audio and writing the result to a file

4.2.2.5 Testing

The successful implementation of this method was the result of several failed attempts using different audio types and methods. The main difficulty was finding a combination that resulted in the media being successfully played back on the Desktop Version of the software as well as the Android version. The testing document displays which formats were trialled.

Test Basis
Verify that, once the Play Button is selected, the audio plays back the expected recording.

Test Cases				
ID	**Test Data**	**Android Result**	**Desktop Result**	**Pass/ Fail**
A1	MP3 data format	Audio did playback	Audio did not playback	FAIL
A2	3GP data format	Audio did playback	Audio did not playback	FAIL
A3	ACC data format	Audio did playback	Audio did not playback	FAIL
A4	MP4 data format	Audio did not playback	Audio did not playback	FAIL
A5	PCM data format	Audio did playback	Audio did playback	PASS

Chapter 6
Evaluation and Conclusion

6.1 Success of the Project

6.1.1 Meeting Requirements

In terms of meeting the requirements this project was mostly successful. The only requirement which the software solution failed to meet was the function which sent notifications to the user about upcoming deadlines via their mobile device (*Chapter 3.2.1 - FR13*). This requirement would of assisted in solving the problem specification but was not a high-priority and when categorised using the MoSCoW method, falls into the Could-Have category. The remaining requirements were all met fully and to a satisfactory level of completion. Where this project could of fared better was with improved requirements analysis before the design phase of the project began. Instead, requirements were left too open and subject to future changes. The agile approach used helped the development to manage these changes but with better requirements analysis in the first place, and a more thorough requirements elicitation process, these changes could of been avoided.

6.1.2 Design Evaluation

When using the software solution, it is clear that the User Interface was designed with an appreciation of the problem specification and target demographic in mind. A common stylistic theme is achieved throughout the design and the colour scheme, font choices and layouts all combine to present a professional and polished interface. In some cases, the design may rely too heavily on the user's own knowledge and intuition, and there was no clear effort to reach an audience of people who may struggle with mobile technology either through impaired hearing/sight or through a lack of previous knowledge and use.

The Software Architecture was designed with an attempt to follow good OOP principles. There is a clear effort to modularise the classes within different packages and the developer has tried to implement design patterns, such as the Client Server model, Layer Pattern and the Model-View-Controller pattern. This is successfully carried out in the Android Application but less so in the Desktop Application. More time and care would of been needed to ensure that these principles were consistent throughout the software system design.

6.1.3 Implementation Evaluation

The source code verifies that the developer put a lot of time and effort into ensuring that the solution could meet the requirements. Complex functions and methods are achieved and the developer shows good problem solving skills to ensure issues, which may affect the project's ability to solve the problem specification, were dealt with. A wide range of knowledge and techniques were used throughout the implementation, with the code to communicate the Android, Desktop and Server components, vital to the success of the project.

In hindsight, the code could of been written with more clarity and more comments to explain the processes used. A greater attempt to achieve good OOP principles could also of been made, with some of the code 'hacked together' in an attempt to get a working solution. There was a heavy reliance on web tutorials and forums such as StackOverflow (stackoverflow.com) to order to find solutions for problems and issues. The code also relies on quite a few external libraries, with the most notable being the library which provided the code necessary to implement the Floating Action Menu used in the Note Tab on the Android device ("Clans/Floatingactionbutton On Github"). However, these issues can be countered by the fact that the scope of the project was quite large, especially for a one-person development team, and a lot of the techniques were new and had to first be learned by the developer.

6.2 Future Work

There were several ideas which could of improved the software which were not included in the initial requirements analysis because they only become apparent during the development process. The main extension to the software, which would improve how it solves the problem specification, would be to engineer a pathway for Tutors to log on and access the system. This pathway would give power to the tutors to set a new essay specific to a module. Then, every student who was registered on that module would automatically be assigned to that essay. This would effectively remove any communication issues between the tutors and students regarding their essay deadlines.

Further future work could include code to implement the notification service outlined in the requirements. This service would automatically notify the students when an essay deadline was approaching as well as, if implemented alongside the above idea, when their tutor had uploaded a new essay. Additional work could be taken to improve the Text Editor tool with some of the ideas discussed in the changes to the original specification (*Chapter 1.4*). An essay import tool could also be introduced along side the current copy-and-paste method which is used to insert essays into the system. The Add Sources function could be extended to include different types of sources such as journals and websites.

In short, there are a lot of functions which could be added to improve this software. This does not mean the software was designed badly but instead proves that this was an open software solution which has the potential to be massive in scale. The current program solves the problem specification by implementing the basic requirements and a few additional requirements, but the scope of the project extends far beyond that.

6.3 Conclusion

Overall, this project has delivered a functional software solution which has been completed to a high level of production and successfully meets the requirements outlined in the requirements specification. Where the project struggled was with the scope of the overall solution. An improved requirements analysis could of narrowed this scope and allowed the development to focus on producing an even better solution on a smaller scale as the developer had issues with implementing the functionality laID out in the requirements. This resulted in some elements of the solution not being achieved (notification service, import tool) and lead to the code focusing more on a working solution rather than good programming principles. Additionally, more formal and thorough testing should of taken place instead of the ad-hoc approach which was used. The future work section highlights the extensions which could be made to the solution to improve how it meets the problem specification. These additional features would result in the software being implemented to an even higher level of production but, as the project stands, the developer can be extremely satisfied with the final solution

References

Print

Anderson, D.J. *Kanban: Successful Evolutionary Change for Your Technology Business.* ISBN: 9780984521401. Washington, WA: Blue Hole Press, 2010.

Brechner, E. *Agile Project Management with Kanban.* ISBN: 9780735698956. Washington, WA: Microsoft Press, 2015.

Fries, B & Fries, M. *Digital Audio Essentials*. ISBN: 9780596008567. California, CA: O'Reilly, 2005.

Hirsch, G. *Helping College Students Succeed: A Model for Effective Intervention.* ISBN: 1560328525 Philadelphia, PA: Routledge, 2001.

Knoernschild, K. *Java Design: Objects, UML, and Process*. ISBN: 9780201750447. Massachusetts, MA: Addison-Wesley, 2002.

Larman, C. *Applying UML and Patterns: An Introduction to Object-oriented Analysis and Design and Iterative Development.* ISBN: 9780910215015. Prentice Hall, 2005.

Linz, T. *Testing In Scrum : A Guide For Software Quality Assurance In The Agile World*. ISBN: 1492001538. Rocky Nook, 2014.

Kantanis, T. *The role of social transition in students': adjustment to the first-year of university.* Journal of Institutional Research, 9(1), pp.100-110, 2000.

Ryan, D. *Understanding Digital Marketing: Marketing Strategies for Engaging the Digital Generation.* ISBN: 9780749471033. 3rd ed. Philadelphia, PA: Kogan Page, 2014.

Stephens, Rod. *Beginning Software Engineering.* ISBN: 9781118969144. Indiana, IN: John Wiley & Sons, 2015.

Wiegers, K. & Beatty, J. *Software Requirements.* ISBN: 9780735679665. 3rd ed. Pearson Education, 2013.

Web

"Chapter 3: Architectural Patterns And Styles". Msdn.microsoft.com. N.p., 2016.

"Clans/Floatingactionbutton On Github". Libraries.io. N.p., 2015.

"Fragments | Android Developers". Developer.android.com. N.p., 2016.

"International Students - Queen's University, Belfast". Thecompleteuniversityguide.co.uk. N.p., 2016.

"Kanban | The Agile Coach". Atlassian. N.p., 2016.

"Queen's University Belfast | Time Management Resources". Qub.ac.uk. N.p., 2016.

Appendices

Appendix A
Project Timetable

Week	Date Commenci	Production Timetable
1	20th June	**Project Start. Commencement of Design Phase.** Meet with project mentor (21/06). Begin project journal. Create project timetable & Kanban chart. Have strong understanding of direction of project so design phase can begin. Research Software Design, System Architecture, Design Patterns. Design data models/entities, use case diagrams, project UML.
2	27th June	**Continuation of Design Phase.** Database design - schema, ER model, normalisation. DDL Script creation. Set up web-server to host the database. Project Specification/Plan.
3	4th July	**Commencement of Phase 2: Desktop Application** This phase will include the design and creation of the desktop application. Overall objectives: Build the data models based on database schema. Build the initial UI based on designs. Build the database and ensure it can connect to the UI via JDBC. Ensure testing (User and JUnit) is carried out throughout.
4	11th July	**HOLIDAYS**
5	18th July	**Continuation Phase 2: Desktop Application**
6	25th July	**Continuation Phase 2: Desktop Application**
7	1st August	**Commencement of Phase 3: Mobile Application** This phase will include the design and creation of the mobile application. Overall objectives: Build the UI based on designs. Connect the mobile application with the main database via the web-server database. Carry out user testing.
8	8th August	**Continuation of Phase 3: Mobile Application**
9	15th August	**Continuation of Phase 3: Mobile Application**
10	22nd August	**Commencement of Phase 4: Documentation/Dissertation** This phase will include the completion of the project documentation/dissertation. While the documentation should be an ongoing progress throughout the project, this time is dedicated to the compiling of different resources/research/project notes and ensure they are included in the final dissertation. Emphasis will also be placed on the correct formatting and style of the dissertation and that each chapter is complete and of the necessary standard.
11	29th August	**Continuation of Phase 4: Documentation/Dissertation**
12	5th September	**Commencement of Phase 5: Project Completion** If, and only if, each previous phase has been completed then this time can be used to add additional features to the project. However, this two week period will primarily be used as a safeguard in the event of any of the other project phases running overtime and to ensure that the project is ready to be presented.
13	12th Sept	**Continuation of Phase 5: Project Completion**
14	19th September	**Project End.** **Wednesday 21st - Submission of Dissertation, Code & Journal** **Thursday 22nd - Project Demo**

Appendix B

Functional requirements

This appendix includes the full list of requirements that specify all the fundamental actions of the software system.

Android Mobile Application

This section includes the requirements that specify all the fundamental actions of the Android Mobile application.

As specified by the Android Developer's Guide there has been an update to the User permissions required when using software on a Android Mobile application ("System Permissions | Android Developers"):

If the device is running Android 6.0 (API level 23) or higher, and the app's target SDK Version is 23 or higher, the app requests permissions from the User at run-time. The User can revoke the permissions at any time, so the app needs to check whether it has the permissions every time it runs.

The assumption will be made that the User agrees to these permission requests when and wherever they occur during the usage of the application and thus they will not be included in the functional requirements. If the User was to deny the permission requests, major functionality of the application would become defunct.

Functional requirement 1.1

USE CASE	**MFR1**
DESCRIPTION	Download Mobile Application
PRE-CONDITION	Mobile device runs on the Android OS. Device has a minimum SDK Version 19 (Android 4.4 - KITKAT).
NORMAL FLOW	User shall be able to download the mobile application through either an application store or similar service on the mobile phone.
EXCEPTIONS	-
POST-CONDITION	The User can open the mobile application on their device.

Functional requirement 1.2

USE CASE	**MFR2**
DESCRIPTION	Welcome Screen
PRE-CONDITION	User has completed MFR1. The application must be open and the device connected to the internet.
NORMAL FLOW	User shall be able to click the 'LOGIN' button.
EXCEPTIONS	-
ALTERNATE FLOW	The User clicks the 'REGISTER' button.
POST-CONDITION	The Login Screen will open.

Functional requirement 1.3

TITLE	MFR3
DESCRIPTION	User Registration
PRE-CONDITION	User has completed MFR2 - Alternate Flow.
NORMAL FLOW	The User shall be able to enter their information into the following fields and then click the 'REGISTER' button: 1. First Name 2. Last Name 3. Email 4. Password 5. Confirm Password 6. Select Course
EXCEPTIONS	If any of the following error conditions are met the User will remain on the same page and an error message will be displayed: 1. Any of the text fields are left empty 2. The email address does not comply with the required email address attributes 3. The 'Password' field and 'Confirm Password' field do not match 4. An account already exists with the given email address
POST-CONDITION	The User's details will be uploaded to the server database. The Login Screen will open.

Functional requirement 1.4

USE CASE	MFR4
DESCRIPTION	Login
PRE-CONDITION	User has completed MFR2 or MFR3.
NORMAL FLOW	The User shall be able to enter their information into the following fields and then click 'LOGIN' button: 1. Student Email 2. Password
EXCEPTIONS	If any of the following error conditions are met the User will remain on the same page and an error message will be displayed: 1. Any of the text fields are left empty 2. No account can be found with the given information
POST-CONDITION	The User details will be validated and they will be logged onto the system. The Home Screen will open.

Functional requirement 1.5

USE CASE	MFR5
DESCRIPTION	Select Essay
PRE-CONDITION	User has completed MFR4. Essays are available on the essay list view.
NORMAL FLOW	The User shall be able to select any essay from the essay list view.
EXCEPTIONS	-
ALTERNATE FLOW	1. The User clicks the ‘REFRESH BUTTON’; see MFR6 2. The User clicks the ‘ADD ESSAY’ icon; see MFR7 3. The User clicks the ‘MENU’ icon; see MFR8
POST-CONDITION	The EssayView page will open the selected essay.

Functional requirement 1.6

USE CASE	MFR6
DESCRIPTION	Refresh Essay List View
PRE-CONDITION	User has completed MFR5 - Alternate Flow 1.
NORMAL FLOW	A progress dialog will be displayed as the server is accessed.
EXCEPTIONS	-
POST-CONDITION	The essay list view shall be refreshed with the latest data from the server.

Functional requirement 1.7

USE CASE	MFR7
DESCRIPTION	Add New Essay
PRE-CONDITION	User has completed MFR5 - Alternate Flow 2.
NORMAL FLOW	The User shall be able to enter information into the following fields and click the ‘ADD ESSAY’ button: 1. Select Module 2. Essay Title 3. Word Limit 4. Start Date 5. End Date
EXCEPTIONS	If any of the following error conditions are met the User will remain on the same page and an error message will be displayed: 1. Any of the text fields are left empty 2. The selected start date is before the selected due date
ALTERNATE FLOW	1. The User clicks the ‘SELECT DATE’ button; see MRF9 2. The User clicks the ‘MENU’ icon; see MFR8
POST-CONDITION	The newly created essay will be uploaded to the server. The HomeScreen page will open.

Functional requirement 1.8

USE CASE	MFR8
DESCRIPTION	Menu Options Selected
PRE-CONDITION	User has selected the ‘MENU’ icon in the toolbar.
NORMAL FLOW	The User shall be able to click the ‘Log Out’ button.
EXCEPTIONS	-
POST-CONDITION	The User will be logged out of the application. The Login page will open.

Functional requirement 1.9

USE CASE	MFR9
DESCRIPTION	Date Picker
PRE-CONDITION	User has completed MFR7 - Alternate Flow 1.
NORMAL FLOW	The Date Picker dialog will appear with the current date highlighted. The User shall be able to select a date and click the ‘OK’ button.
EXCEPTIONS	-
ALTERNATE FLOW	1. The User clicks the ‘CANCEL’ button - the dialog will close.
POST-CONDITION	The selected date will be shown in the Add Essay Page.

Functional requirement 1.10

USE CASE	MFR10
DESCRIPTION	Essay View Tabbed Navigation - Overview
PRE-CONDITION	User has completed MFR5.
NORMAL FLOW	The Essay Overview page will be displayed. The User shall be able to swipe left to access the next tab.
EXCEPTIONS	-
ALTERNATE FLOW	1. The User clicks the ‘ADD ESSAY’ icon; see MFR7 2. The User clicks the ‘MENU’ icon; see MFR8
POST-CONDITION	The next tab will be selected.

Functional requirement 1.11

USE CASE	MFR11
DESCRIPTION	Essay View Tabbed Navigation - Notes
PRE-CONDITION	User has completed MFR10.
NORMAL FLOW	The Notes page will be displayed. The User shall be able to swipe left to access the next tab. The User shall be able to swipe right to access the previous tab.
EXCEPTIONS	-
ALTERNATE FLOW	1. The User clicks the ‘ADD ESSAY’ icon; see MFR7 2. The User clicks the ‘MENU’ icon; see MFR8 3. The ‘+’ floating action toolbar is selected; see MFR14 4. The User selects a Text Note from the list; see MFR20 5. The User selects an Audio Note from the list; see MFR21 6. The User selects a Photo Note from the list; see MFR22
POST-CONDITION	The next tab’s layout will be shown.

Functional requirement 1.12

USE CASE	MFR12
DESCRIPTION	Essay View Tabbed Navigation - Essay
PRE-CONDITION	User has completed MFR11.
NORMAL FLOW	The Essay Preview page will be displayed. The User shall be able to swipe left to access the next tab. The User shall be able to swipe right to access the previous tab.
EXCEPTIONS	-
ALTERNATE FLOW	1. The User clicks the ‘ADD ESSAY’ icon; see MFR7 2. The User clicks the ‘MENU’ icon; see MFR8
POST-CONDITION	The next tab’s layout will be shown.

Functional requirement 1.13

USE CASE	MFR13
DESCRIPTION	Essay View Tabbed Navigation - Sources
PRE-CONDITION	User has completed MFR12.
NORMAL FLOW	The Sources page will be displayed. The User shall be able to swipe right to access the previous tab.
EXCEPTIONS	-
ALTERNATE FLOW	1. The User clicks the ‘ADD ESSAY’ icon; see MFR7 2. The User clicks the ‘MENU’ icon; see MFR8
POST-CONDITION	The next tab’s layout will be shown.

Functional requirement 1.14

USE CASE	MFR14
DESCRIPTION	Add New Note Floating Action Menu
PRE-CONDITION	User has completed MFR11 - Alternate Flow 3.
NORMAL FLOW	The floating action menu will be displayed. The ‘+’ icon will transform into a ‘x’ icon. The User shall be able to select the ‘x’ icon.
EXCEPTIONS	-
ALTERNATE FLOW	1. The User clicks the ‘ADD ESSAY’ icon; see MFR7 2. The User clicks the ‘MENU’ icon; see MFR8 3. The User clicks the ‘TEXT’ icon; see MRF15 4. The User clicks the ‘CAMERA’ icon; see MRF16 5. The User clicks the ‘AUDIO’ icon; see MRF19
POST-CONDITION	The floating action menu will be closed.

Functional requirement 1.15

USE CASE	MFR15
DESCRIPTION	Add New Text Note
PRE-CONDITION	User has completed MFR14 - Alternate Flow 3.
NORMAL FLOW	The User shall be able to enter information into the following fields and click the ‘SAVE’ button: 1. Title 2. Note
EXCEPTIONS	-
POST-CONDITION	The Text Note will be saved to the server. The Essay View Tabbed Navigation - Notes page will be opened and the new note will appear on the note list.

Functional requirement 1.16

USE CASE	MFR16
DESCRIPTION	Add New Photo Note
PRE-CONDITION	User has completed MFR14 - Alternate Flow 4.
NORMAL FLOW	The User shall be able to enter information into the ‘Title’ field. The User shall be able to to click the ‘CAMERA’ icon (see MFR17). The User shall be able to click the ‘GALLERY’ icon (see MFR18). User shall be able to click the ‘SAVE’ button.
EXCEPTIONS	The User will remain on the same page and an error message will be displayed if: 1. No input is added to the ‘Title’ field. 2. No image is added.
POST-CONDITION	The image will be saved to the server. The Essay View Tabbed Navigation - Notes page will be opened and the new note will appear on the note list.

Functional requirement 1.17

USE CASE	MFR17
DESCRIPTION	Add New Photo Using Camera
PRE-CONDITION	User has selected the ‘CAMERA’ icon during MFR16.
NORMAL FLOW	The device’s camera application is opened. The User shall be able to take a picture. The User shall be able to save the picture.
EXCEPTIONS	-
POST-CONDITION	The image will be saved. The Add New Photo Note page will be opened and the saved picture available to view.

Functional requirement 1.18

USE CASE	MFR18
DESCRIPTION	Add New Photo Using Gallery
PRE-CONDITION	User has selected the ‘GALLERY’ icon during MFR16.
NORMAL FLOW	The device’s gallery application is opened. The User shall be able to take a picture. The User shall be able to save the picture.
EXCEPTIONS	-
POST-CONDITION	The image will be saved. The Add New Photo Note page will be opened and the saved picture available to view.

Functional requirement 1.19

USE CASE	MFR19
DESCRIPTION	Add New Audio Note
PRE-CONDITION	User has completed MFR14 - Alternate Flow 5.
NORMAL FLOW	The User shall be able to enter information into the ‘Title’ field. The User can record audio: 1. The User shall be able to to click the ‘RECORD’ icon. 2. A message will appear : ‘Recording started’ 3. The ‘RECORD’ icon will be replaced by the ‘STOP’ icon. 4. The User shall be able to click the ‘STOP’ icon. 5. A message will appear : ‘Audio recorded successfully’. 6. The ‘STOP’ icon will be replaced by the ‘PLAY’ icon. 7. The ‘SAVE’ button will appear. 7. The User shall be able to click the ‘PLAY’ button. 8. The audio will be played back. 9. The User shall be able to click the ‘SAVE’ button.
EXCEPTIONS	-
POST-CONDITION	The audio will be saved to the server. The Essay View Tabbed Navigation - Notes page will be opened and the new note will appear on the note list.

Functional requirement 1.20

USE CASE	MFR20
DESCRIPTION	View Text Note
PRE-CONDITION	User has completed MFR11 - Alternate Flow 4.
NORMAL FLOW	The User shall be able to view the selected note. The User shall be able to click the ‘EDIT’ floating action button. The User shall be able to edit the title. The User shall be able to edit the note. The User shall be able to click the ‘SAVE’ icon in the toolbar.
EXCEPTIONS	-
ALTERNATE FLOW	1. The User clicks the ‘MENU’ icon; see MFR8
POST-CONDITION	The note will be saved to the server. The Essay View Tabbed Navigation - Notes page will be opened and the edited note will appear on the note list.

Functional requirement 1.21

USE CASE	MFR21
DESCRIPTION	Play Audio Note
PRE-CONDITION	User has completed MFR11 - Alternate Flow 5.
NORMAL FLOW	The audio note will playback. A message will appear : ‘Playing audio’.
EXCEPTIONS	-
POST-CONDITION	-

Functional requirement 1.22

USE CASE	MFR22
DESCRIPTION	View Photo Note
PRE-CONDITION	User has completed MFR10 - Alternate Flow 6.
NORMAL FLOW	The User shall be able to view the selected photo. The User shall be able to click the ‘BACK’ button to exit the Photo View window.
EXCEPTIONS	-
POST-CONDITION	The User is returned to the Essay View Tabbed Navigation - Notes page.

Desktop Application

This section includes the requirements that specify all the fundamental actions of the Desktop application.

Functional requirement 2.1

USE CASE	DFR1
DESCRIPTION	Download/Install Application
PRE-CONDITION	Device has the required storage/system requirements.
NORMAL FLOW	User shall be able to download and install the application.
EXCEPTIONS	-
POST-CONDITION	The User can open the application on their device.

Functional requirement 2.2

USE CASE	DFR2
DESCRIPTION	Login Screen
PRE-CONDITION	User has completed DFR1.
NORMAL FLOW	The User shall be able to enter their information into the following fields and then click the ‘LOGIN’ button: 1. Student Email 2. Password
EXCEPTIONS	If any of the following error conditions are met the User will remain on the same page and an error message will be displayed: 1. Any of the text fields are left empty 2. No account can be found with the given information
ALTERNATE FLOW	1. The User clicks the ‘Forgot Password’ link. 2. The User clicks the ‘Register’ link.
POST-CONDITION	The User detail’s will be validated and they will be logged onto the system. The Home Screen will open.

Functional requirement 2.3

USE CASE	DFR3
DESCRIPTION	Forgot Password
PRE-CONDITION	User has completed DFR2 - Alternate Flow 1.
NORMAL FLOW	The User shall be able to enter their information into the email field and then click the ‘SEND’ button.
EXCEPTIONS	If the email field is left blank then the User will remain on the same page and an error message will be displayed.
POST-CONDITION	The User will be sent an email with a randomly generated password which they will be able to log in using.

Functional requirement 2.4

TITLE	DFR4
DESCRIPTION	User Registration
PRE-CONDITION	User has completed DFR2 - Alternate Flow 2.
NORMAL FLOW	The User shall be able to enter their information into the following fields and then click the ‘REGISTER’ button: 1. First Name 2. Last Name 3. Email 4. Confirm Email 5. Password 6. Confirm Password
EXCEPTIONS	If any of the following error conditions are met the User will remain on the same page and an error message will be displayed: 1. Any of the text fields are left empty 2. The email address does not comply with the required email address attributes 3. The ‘Password’ field and ‘Confirm Password’ field do not match 4. The ‘Email’ field and ‘Confirm Email’ field do not match 5. An account already exists with the given email address
POST-CONDITION	The User’s details will be uploaded to the server database. The Home Screen will open.

Functional requirement 2.5

TITLE	DFR5
DESCRIPTION	Home Screen
PRE-CONDITION	User has completed DFR2 or DFR4. An essay is available in the essay list view.
NORMAL FLOW	The User shall be able to select an essay from the essay list view.
EXCEPTIONS	-
ALTERNATE FLOW	1. The ‘HOME’ icon is clicked. See DFR6. 2. The ‘ADD ESSAY’ icon is clicked. See DFR7. 3. The ‘ACOUNT’ icon is clicked. See DFR8. 4. The ‘LOGOUT’ icon is clicked. See DFR9.
POST-CONDITION	The EssayView Screen will open the selected essay.

Functional requirement 2.6

TITLE	DFR6
DESCRIPTION	Home Screen
PRE-CONDITION	User has completed DFR5 - Alternate Flow 1.
NORMAL FLOW	The User will be navigated to the Home Screen.
EXCEPTIONS	If the Home Screen is already open an error dialog will appear.
POST-CONDITION	The Home Screen will open.

Functional requirement 2.7

TITLE	DFR7
DESCRIPTION	Add Essay
PRE-CONDITION	User has completed DFR5 - Alternate Flow 2.
NORMAL FLOW	The User shall be able to enter information into the following fields and click the ‘ADD’ button: 1. Select Module 2. Essay Title 3. Word Limit 4. Start Date 5. End Date
EXCEPTIONS	If any of the following error conditions are met the User will remain on the same page and an error message will be displayed: 1. Any of the text fields are left empty 2. The selected start date is before the selected due date
POST-CONDITION	The essay will be added to the server database. The User will be returned to the previous screen.

Functional requirement 2.8

TITLE	DFR8
DESCRIPTION	My Account
PRE-CONDITION	User has completed DFR5 - Alternate Flow 3.
NORMAL FLOW	The User shall be able to click the ‘Edit’ button. The fields will become available to edit. The ‘Save’ button will become enabled. The User shall be able to enter their information into the following fields and then click the ‘Save’ button: 1. First Name 2. Last Name 3. Password 4. New Password 5. Confirm Password
EXCEPTIONS	If any of the following error conditions are met the User will remain on the same page and an error message will be displayed: 1. Any of the text fields are left empty 2. The ‘New Password’ and ‘Password’ fields do not match. 3. New password is the same as the old one.
POST-CONDITION	User details will be saved. The User will be returned to the previous screen.

Functional requirement 2.9

TITLE	DFR9
DESCRIPTION	Logout
PRE-CONDITION	User has completed DFR5 - Alternate Flow 4.
NORMAL FLOW	A dialog box will appear asking the User to confirm that they want to log out. The User can click on the ‘OK’ option.
EXCEPTIONS	If any of the following error conditions are met the User will remain on the same page and an error message will be displayed: 1. Any of the text fields are left empty 2. The ‘New Password’ and ‘Password’ fields do not match. 3. New password is the same as the old one.
ALTERNATE FLOW	The User clicks the ‘Cancel’ button and returns to the previous page.
POST-CONDITION	The User will be logged out. The Login Screen will open.

Functional requirement 2.10

TITLE	DFR10
DESCRIPTION	Essay View - Notes
PRE-CONDITION	User has completed DFR5.
NORMAL FLOW	User shall be able to select a Note from the Note List found in the right hand panel. The following functionality occurs for each Note type: A. Text Note - User shall be able to edit the text and then click the enabled ‘Save’ button to save their changes B. Photo Note - User shall be able to view the image C. Audio Note - User shall be able to click the ‘PLAY’ icon to hear audio playback
EXCEPTIONS	-
ALTERNATE FLOW	1. The ‘HOME’ icon is clicked. See DFR6. 2. The ‘ADD ESSAY’ icon is clicked. See DFR7. 3. The ‘ACOUNT’ icon is clicked. See DFR8. 4. The ‘LOGOUT’ icon is clicked. See DFR9. 5. The ‘New Note’ button is clicked. See DFR11. 6. The ‘Delete Note’ button is clicked. See DFR12. 7. The ‘Essay Content’ tab is selected. See DFR13. 8. The ‘Sources’ tab is selected. See DFR14.
POST-CONDITION	The main display will be populated with the Note’s title, date and content.

Functional requirement 2.11

TITLE	DFR11
DESCRIPTION	New Note
PRE-CONDITION	User has completed DFR10 - Alternate Flow 5.
NORMAL FLOW	User shall be to enter a Note Title in the dialog box. User shall be able to click the ‘Add’ button.
EXCEPTIONS	An error message will appear if the user does not enter anything into the text field.
ALTERNATE FLOW	User selects the ‘Cancel’ button and is returned to the previous page.
POST-CONDITION	A new Text Note will be added to the note list. The main display will show the empty note for the user to edit.

Functional requirement 2.12

TITLE	DFR12
DESCRIPTION	Delete Note
PRE-CONDITION	User has completed DFR10 - Alternate Flow 6. A Note is selected.
NORMAL FLOW	A dialog box will appear asking the User to confirm that they want to delete the selected note. The User can click on the ‘OK’ option.
EXCEPTIONS	-
ALTERNATE FLOW	User selects the ‘Cancel’ button and is returned to the previous page.
POST-CONDITION	A note will be deleted and removed from the note list.

Functional requirement 2.13

TITLE	DFR13
DESCRIPTION	Essay View - Essay Content
PRE-CONDITION	User has completed DFR5 - Alternate Flow 7 or DFR13 - Alternate Flow 8.
NORMAL FLOW	User shall be able to paste their essay into the text area.
EXCEPTIONS	-
ALTERNATE FLOW	1. The ‘HOME’ icon is clicked. See DFR6. 2. The ‘ADD ESSAY’ icon is clicked. See DFR7. 3. The ‘ACOUNT’ icon is clicked. See DFR8. 4. The ‘LOGOUT’ icon is clicked. See DFR9. 5. The ‘New Note’ button is clicked. See DFR11. 6. The ‘Delete Note’ button is clicked. See DFR12. 7. The ‘Notes & Planning’ tab is selected. See DFR10. 8. The ‘Sources’ tab is selected. See DFR14.
POST-CONDITION	The word count will be updated. The progress bar will be updated. Content will be added to the essay.

Functional requirement 2.14

TITLE	DFR14
DESCRIPTION	Essay View - Sources
PRE-CONDITION	User has completed DFR5 - Alternate Flow 8 or DFR13 - Alternate Flow 8.
NORMAL FLOW	User shall be able to paste their sources into the text area. User shall be able to click the ‘ADD SOURCE’ button - see DFR15.
EXCEPTIONS	-
ALTERNATE FLOW	1. The ‘HOME’ icon is clicked. See DFR6. 2. The ‘ADD ESSAY’ icon is clicked. See DFR7. 3. The ‘ACOUNT’ icon is clicked. See DFR8. 4. The ‘LOGOUT’ icon is clicked. See DFR9. 5. The ‘New Note’ button is clicked. See DFR11. 6. The ‘Delete Note’ button is clicked. See DFR12. 7. The ‘Notes & Planning’ tab is selected. See DFR10. 8. The ‘Essay Content’ tab is selected. See DFR13.
POST-CONDITION	Sources will be added to the essay.

Functional requirement 2.15

TITLE	DFR15
DESCRIPTION	Add Sources
PRE-CONDITION	User has selected the ‘ADD SOURCE’ button during DFR14.
NORMAL FLOW	The User shall be able to enter their information into the following fields and then click the ‘Add’ button: 1. Source Title 2. Author First Name 3. Author Last Name 4. Year 5. City 6. Publisher 7. Pages (optional)
EXCEPTIONS	If any of the following error conditions are met the User will remain on the same page and an error message will be displayed: 1. Any of the text fields are left empty aside from ‘Pages’
ALTERNATE FLOW	1. The ‘HOME’ icon is clicked. See DFR6. 2. The ‘ADD ESSAY’ icon is clicked. See DFR7. 3. The ‘ACOUNT’ icon is clicked. See DFR8. 4. The ‘LOGOUT’ icon is clicked. See DFR9. 5. The ‘New Note’ button is clicked. See DFR11. 6. The ‘Delete Note’ button is clicked. See DFR12. 7. The ‘Notes & Planning’ tab is selected. See DFR10. 8. The ‘Essay Content’ tab is selected. See DFR13.
POST-CONDITION	Sources will be added to the essay.