2022

COIT13230

APPLICATION AND SOFTWARE DEVELOPMENT CAPSTONE PROJECT

ASSIGNMENT 3 – PROJECT REQUIREMENTS SPECIFICATION AND DESIGN

DUE DATE: 29/08/2022

WORK INTEGRATED LEARNING MANAGEMENT APPLICATION (WILMA)

NICHOLAS MCGUFFIN

NATHAN DOWNES - 12046570

RORY ALLEN - 12149026

NATHAN SNOW - 12060962

Table of Contents

A. System Requirement Specification

i. Introduction

The development of a Work Integrated Learning (WIL) system will deliver a web-based application that allows Central Queensland University (CQU) to collaborate with industry partners to provide a work placement service for its stakeholders – Industry business leaders, CQU educators and students. The implementation of this system will allow our client, CQU, to help its students transition more efficiently and effectively into the workforce whilst also helping Australian businesses prosper. The proposed basic functionality of the WIL application includes:

- Industry partners can register their interest in the placement program and can publish job and placement opportunities to be viewed by educators, students, and other placement providers.
- CQU Educators can monitor, manage, and share placement and job opportunities between students and placement providers, and can also communicate with other users on the platform.
- Students can view job vacancies and register a specific interest in selected disciplines. They can also apply for placement and job opportunities through the application and can upload their resumes for employers to review and critique. Finally, students can also communicate with educators and industry partners using a forum to post questions and answers regarding placement and work opportunities.

Our client's goals align directly with the functionality of our proposed system, making it a noteworthy candidate as a solution. The software development firm taking ownership of the project is Wingin' IT Solutions, and the final product being delivered to our client is named the **W**ork **I**ntegrated **L**earning **M**anagement **A**pplication (WILMA)

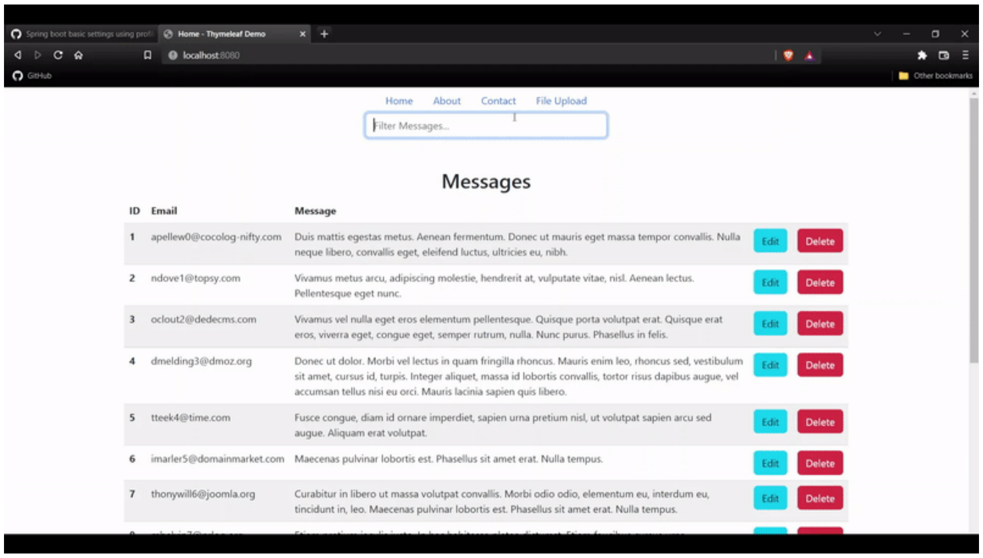
ii. Schedule and Task Allocation

Project task completion has been tracking well according to our running project schedule, and there have been no changes to the team regarding members, however some of the scheduled tasks needed to be renamed to map to their respective assessment tasks more accurately.

Changes to Project Scope

Project scope has been staying mostly within its initial bounds; however, we have decided to include a couple of minor additions that we believe are significant "value adders" for the client and all system users without introducing too much scope creep.

- Dynamic search/filter functionality for data table content (e.g.jobs, placements, pending publications (admin view)) that will update on every key press rather than a simple "submit" style search like the example below.

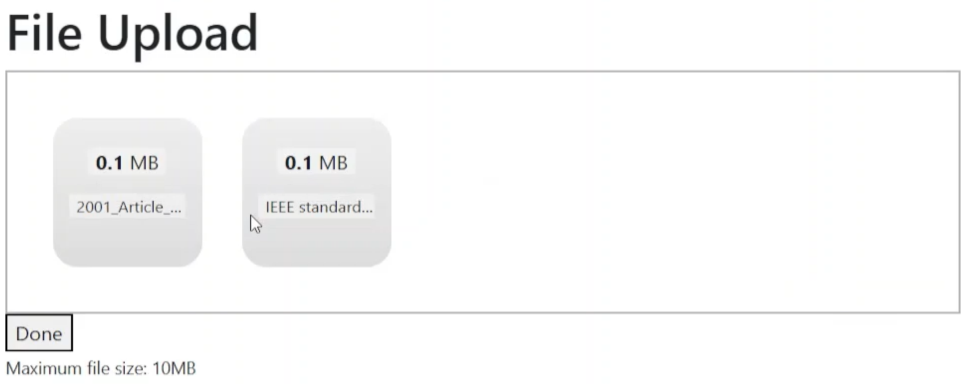


Messages

ID	Email	Message		
1	apellew0@cocolog-nifty.com	Duis mattis egestas metus. Aenean fermentum. Donec ut mauris eget massa tempor convallis. Nulla neque libero, convallis eget, eleifend luctus, ultricies eu, nibh.	Edit	Delete
2	ndove1@topsy.com	Vivamus metus arcu, adipiscing molestie, hendrerit at, vulputate vitae, nisl. Aenean lectus. Pellentesque eget nunc.	Edit	Delete
3	oclout2@dedecms.com	Vivamus vel nulla eget eros elementum pellentesque. Quisque porta volutpat erat. Quisque erat eros, viverra eget, congue eget, semper rutrum, nulla. Nunc purus. Phasellus in felis.	Edit	Delete
4	dmelding3@dmoz.org	Donec ut dolor. Morbi vel lectus in quam fringilla rhoncus. Mauris enim leo, rhoncus sed, vestibulum sit amet, cursus id, turpis. Integer aliquet, massa id lobortis convallis, tortor risus dapibus augue, vel accumsan tellus nisi eu orci. Mauris lacinia sapien quis libero.	Edit	Delete
5	tteek4@time.com	Fusce congue, diam id ornare imperdiet, sapien urna pretium nisl, ut volutpat sapien arcu sed augue. Aliquam erat volutpat.	Edit	Delete
6	imarler5@domainmarket.com	Maecenas pulvinar lobortis est. Phasellus sit amet erat. Nulla tempus.	Edit	Delete
7	thonywill6@joomla.org	Curabitur in libero ut massa volutpat convallis. Morbi odio odio, elementum eu, interdum eu, tincidunt in, leo. Maecenas pulvinar lobortis est. Phasellus sit amet erat. Nulla tempus.	Edit	Delete

Figure 1: Dynamic Filtering

- Drag and drop multi-file upload for students to upload multiple resumes or cover letters



-

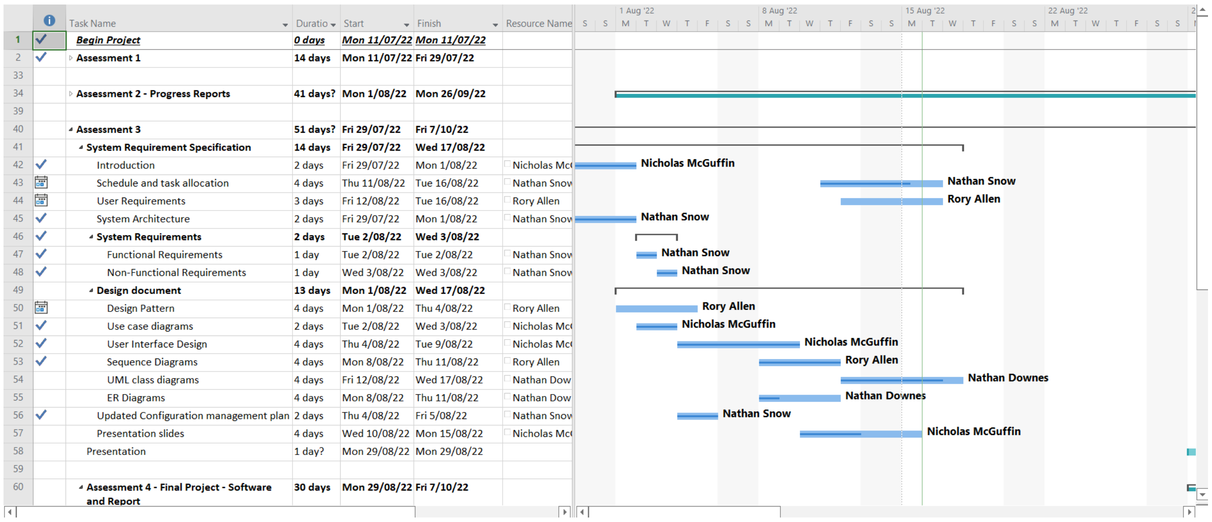
Figure 2: File Upload Drop Zone

- The ability for users to open PDF or various image files directly in the browser, or other file types in their respective applications (.docx in MS Word, .xlsx in MS Excel etc)
- Being able to download or delete files such as resume's and cover letters will ensure each user has maximum control over their files.

Current Project Status

The following task dependency table shows the teams' most recent progress status (as at 16/08/2022), each task being allocated a team member responsible for its completion.

Care has been taken to ensure that team resources (members) have not been overallocated, and project tasks have been distributed effectively throughout the available team.



		Task Name	Duration	Start	Finish	Resource Name
1	✓	***Begin Project***	***0 days***	***Mon 11/07/22***	***Mon 11/07/22***	
2	✓	**Assessment 1**	**14 days**	**Mon 11/07/22**	**Fri 29/07/22**	
33						
34		**Assessment 2 - Progress Reports**	**41 days?**	**Mon 1/08/22**	**Mon 26/09/22**	
39						
40		**Assessment 3**	**51 days?**	**Fri 29/07/22**	**Fri 7/10/22**	
41		**System Requirement Specification**	**14 days**	**Fri 29/07/22**	**Wed 17/08/22**	
42	✓	Introduction	2 days	Fri 29/07/22	Mon 1/08/22	Nicholas Mc
43	📅	Schedule and task allocation	4 days	Thu 11/08/22	Tue 16/08/22	Nathan Snow
44	📅	User Requirements	3 days	Fri 12/08/22	Tue 16/08/22	Rory Allen
45	✓	System Architecture	2 days	Fri 29/07/22	Mon 1/08/22	Nathan Snow
46	✓	**System Requirements**	**2 days**	**Tue 2/08/22**	**Wed 3/08/22**	
47	✓	Functional Requirements	1 day	Tue 2/08/22	Tue 2/08/22	Nathan Snow
48	✓	Non-Functional Requirements	1 day	Wed 3/08/22	Wed 3/08/22	Nathan Snow
49		**Design document**	**13 days**	**Mon 1/08/22**	**Wed 17/08/22**	
50	📅	Design Pattern	4 days	Mon 1/08/22	Thu 4/08/22	Rory Allen
51	✓	Use case diagrams	2 days	Tue 2/08/22	Wed 3/08/22	Nicholas Mc
52	✓	User Interface Design	4 days	Thu 4/08/22	Tue 9/08/22	Nicholas Mc
53	✓	Sequence Diagrams	4 days	Mon 8/08/22	Thu 11/08/22	Rory Allen
54		UML class diagrams	4 days	Fri 12/08/22	Wed 17/08/22	Nathan Dow
55		ER Diagrams	4 days	Mon 8/08/22	Thu 11/08/22	Nathan Dow
56	✓	Updated Configuration management plan	2 days	Thu 4/08/22	Fri 5/08/22	Nathan Snow
57		Presentation slides	4 days	Wed 10/08/22	Mon 15/08/22	Nicholas Mc
58		Presentation	1 day?	Mon 29/08/22	Mon 29/08/22	
59						
60		**Assessment 4 - Final Project - Software and Report**	**30 days**	**Mon 29/08/22**	**Fri 7/10/22**	

Figure 3: Schedule & Task Allocation

Overall Progress

Project progress overall (at 16/08/2022) is tracking at 48% complete which translates accurately given we are approximately halfway through the total project duration.

As can be seen in the "% Work Completed" chart below, assessment 2 is well underway but tracking slowly due to its distributed submission dates, and assessment 3 is rolling along nicely and on schedule.

PROJECT OVERVIEW

MON 11/07/22 SAT 8/10/22

% COMPLETE

48%

% WORK COMPLETE

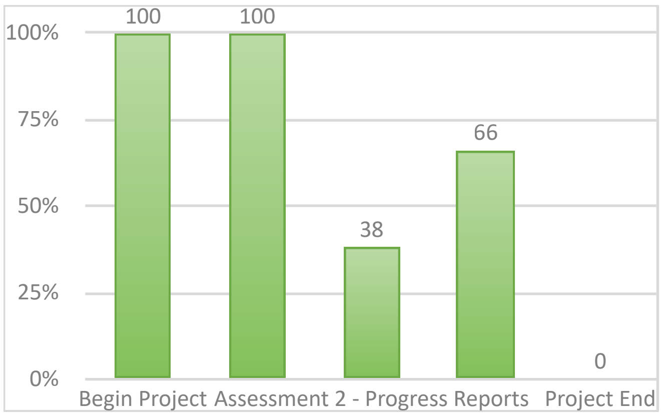


Figure 4: Work completed at at16/08/2022

Upcoming Changes

The team is expecting to make some minor modifications to the project schedule and task dependencies table during execution of the final software and report (assessment 4) which will be mostly around task renaming. Currently the tasks are named to align with the assessment instructions, but as we progress through the tasks, we expect to add some more detailed sub-tasks, particularly under the task “Implementation and Testing.”

iii. User Requirements

The WILMA system is designed to facilitate the placement of tertiary students in the Work Integrated Learning (WIL) program offered by the School of Engineering and Technology at the University of Central Queensland. The system will be used to manage WIL placement information between three types of users: students seeking internship positions, industry partners offering those positions, and educators within the school to manage those placements.

Students will be able to create an account using their @cqumail.com email address. Once verified, they can manage their personal profile including name, contact information, and disciplines, and can upload their resumes. They can also post and reply in the forums and can view and apply for positions posted by industry partners. They will receive notifications of new positions that match the disciplines they have set in their profile.

Industry partners can create their account and, once verified, can post and reply in the forums and manage their business profiles. They can also submit the details of any positions that they wish to offer appropriate students. These positions need to be reviewed by an educator before they become available for students to view. They can also view Expressions of Interest (EOIs) submitted by educators. They will receive notification of new applications to their submitted position vacancies, and of new EOI’s that match the disciplines/categories they have set in their business profile.

An educator can create an account using their @cqu.edu.au email address. Once verified, they can post and reply in the forums and manage their profiles. Educators can submit EOIs to be viewed by industry partners and can also review and accept or deny job positions submitted by industry partners. They can manage (update/delete) the available positions and their status as well as the set of disciplines that categorise the job placements on offer. They can also manage (edit/delete) forum posts/replies and block user’s access to the system.

Once a student has applied for a position, the placement can be accepted by the industry partner. When a student has completed their placement, the industry partner can submit a report. The report can then be reviewed by an educator, before the placement is ultimately closed.

The system is designed with reliability and performance in mind. It will be initially scaled to handle the load of 1000 concurrent users, and the database will be backed up daily during off-peak usage hours. The distributed system has a projected response time of 0.2 seconds so that, from a user perspective, it will be indistinguishable from local operations, and it should operate identically under scaled user load.

The system is designed to be secure, with system data only accessible to appropriate parties. Users must be verified through an email verification process before using the system, and functionality is then limited by user role appropriately. User data is to be stored using trusted hash and salt algorithms, and the site will implement SSL (Secure Sockets Layer) certificate security. The system will not, however, extend to automated domain/ABN verification of industry partners, i.e., they will have to be manually authenticated and managed by users assigned the educator role.

The system is designed to be extensible, with many of the features being adaptable to potential future contexts. Within the system, users with elevated roles (such as educators) can modify the set of disciplines by which the content is categorised, to allow them to adapt to similar (WIL) programs run by other departments. The system can also be adaptable to entirely different clients as its’ implementation of UI templating can be easily modified to align visually with other brands. Furthermore, the use of exposed RESTful web service endpoints will allow for the system to be seamlessly deployed through additional UI modalities (e.g., native mobile applications mobile).

The system is designed to be highly usable, adhering to established and accepted UX heuristics. Additionally, cloud deployment through AWS means the system will experience less than 2% downtime, and maintenance will be carried out outside of regular business hours to minimise service interruptions. Following handover, up to 15 hours of support per month (for 2 years) can be provided to the client, with additional support charged at a nominal rate. Periodic service updates will be provided for 2 years, with the option for extended service maintenance negotiable.

iv. System Architecture

The WILMA system is split into numerous monolithic modules in contrast to the popular microservices architecture, based on maximising portability of the project particularly during the early stages.

Although these are fully expected to increase in number, initial project modules include:

- **Configuration**
  Application configuration including authorisation, authentication, and project-wide settings. Architecturally, the configuration module spans the API and Service application layers because it will contain settings that will be injected into both layer's APIs during runtime activities.

- **Entity**
  The entity module (residing on the data access layer) will contain all entity classes that will be used throughout the application including object relational mapping (ORM).

- **Service**
  The service module will contain all of the application's business logic as service classes (beans) and make up a major part of the service layer providing essential abstraction. These services will be called by the application controllers and then perform some logic with or without accessing any of the application repositories.

- **Repository**
  Repositories consist of JPA repository interfaces that contain logic for handling persistence of application entities. These will be accessed by services only, which will help implement the last layer of positive abstraction required by the system.

- **Web**
  The web module will house the application controllers, exception handlers, and web templates that are responsible for handling both website navigation and REST endpoints as shown in the API layer.
  This module is also the main driver module for the WILMA system and contains basic runtime configuration for items such as the active profile, server port, and database connection settings.

Whilst not in its own module per se, user access to the WILMA system will be managed via the Spring Security framework for both user authentication and 'role-based' authorisation. Once authenticated, users will be able to access the system with the appropriate access for their given role/s.

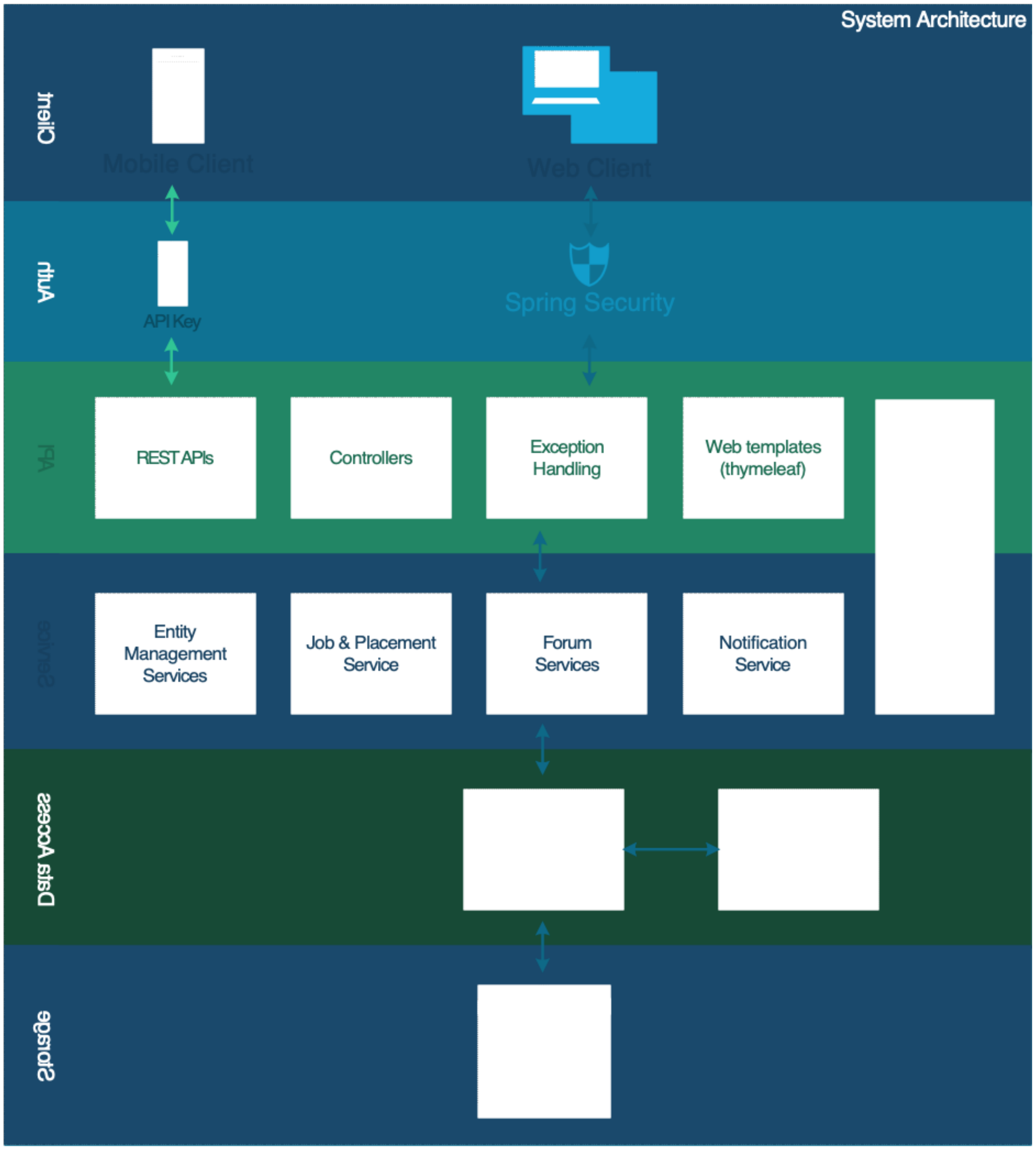


Figure 5: System Architecture

v. System Requirements

The requirements been configured in a way that allows developers and categories to be defined by prefix or initials respectively, and these prefixes and initials are then used to construct a unique code. Having this code format allows us to easily identify the the core details of the requirement.

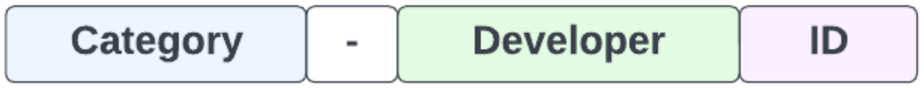
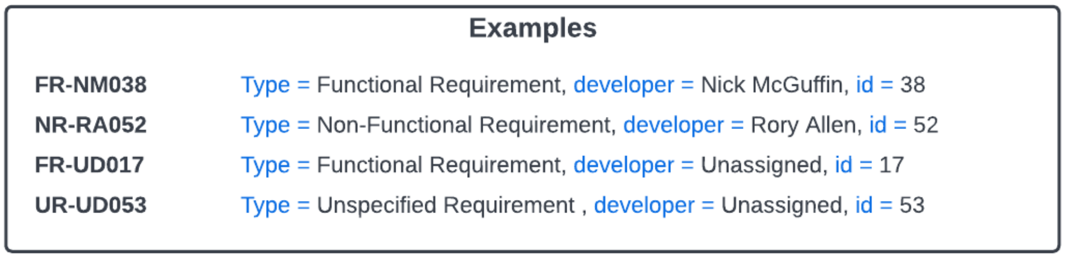


Figure 6: Requirement Code Format

The structure of a requirement code can be broken down into the following segments as per the image above (*note there is no delimiter between the developer initials and identifier)*:

1. The category prefix
    a. FR - Functional Requirement
    b. NR - Non-Functional Requirement
    c. UR - Unspecified Requirement
2. A predefined delimiter character (in this case “-“)
3. The developer Initials
    a. ND - Nathan Downes
    b. RA - Rory Allen
    c. NM – Nicholas McGuffin
    d. NS - Nathan Snow
    e. UD – Unassigned developer
4. The unique requirement identifier (padded to a minimum of 3 digits E.g. 25 = 025)

Below illustrates some examples of valid requirement codes and how they can be interpreted.



Examples

FR-NM038	Type = Functional Requirement, developer = Nick McGuffin, id = 38
NR-RA052	Type = Non-Functional Requirement, developer = Rory Allen, id = 52
FR-UD017	Type = Functional Requirement, developer = Unassigned, id = 17
UR-UD053	Type = Unspecified Requirement , developer = Unassigned, id = 53

Figure 7: Requirement Code Examples

Functional Requirements

The identified functional requirements have been broken down into four broad categories, and each category assigned to a designated developer.

The four defined categories are:

1. User authentication, authorisation, and notifications – Nathan Snow
2. Educator specific roles and responsibilities including admin features – Nicholas McGuffin
3. Industry partner roles and responsibilities, including application submission – Rory Allen
4. UI layout, content, organisation, and user profiles – Nathan Downes

Breaking up these requirements into categories not only helps to allocate tasks amongst the available development team, but also allows the developers keep more of a targeted mindset rather than having 'everyone doing everything'.

FR-NS001	Users can create a new account	Nathan Snow
FR-NS002	Roles and authorities are assigned to a new user account based on their account type	Nathan Snow
FR-NS003	Users can log in/out	Nathan Snow
FR-NM004	Educators can manage required job/placement opportunities	Nicholas McGuffin
FR-NM005	Educators can manage available jobs/placements	Nicholas McGuffin
FR-NM006	Educators can create and manage discipline groups	Nicholas McGuffin
FR-ND007	Users can manage their profile information (including disciplines)	Nathan Downes
FR-RA008	Industry partners can add a description about their business and/or plans	Rory Allen
FR-ND009	Jobs/placements can be categorised by discipline groups	Nathan Downes
FR-NM010	Educators can manage expressions of interest to supply a job/placement	Nicholas McGuffin
FR-ND011	Display and filter job/placement thumbnail/overview	Nathan Downes
FR-NS012	Notify users of recent activity based on their selected disciplines	Nathan Snow
FR-ND013	Show further details for a particular job or placement	Nathan Downes
FR-RA014	Handle job/placement application submissions	Rory Allen
FR-RA015	Notify industry partners of new applications for their job/placement	Rory Allen
FR-RA016	Allow users to publicly ask and answer questions (Q&A style forum like StackOverflow)	Rory Allen

Figure 8: Functional Requirements Snapshot

Non-functional Requirements

Like assigning the functional requirements, the non-functional requirements were broken up into the same 4 categories, however not all requirements were assigned to any one developer due to their implicit nature.

For example, (1) ensuring system uptime and (2) sufficient concurrent users, are not just tasks that one member will undertake alone, but rather tasks that will be determined by external sources and/or by the group as a whole.

Developer-category alignment is the same as with the functional requirements as some task crossover is expected during development. It is hoped that keeping developers on the same track across the project will reduce the likelihood of any tasks clashing and holding up other team members from their allocated development tasks.

NR-NS017	User passwords only to be stored using a secure and trusted hash and salt algorithm	Nathan Snow
NR-NS018	Must only supply access according to a user's roles & authorities	Nathan Snow
NR-NS019	Jobs/Placements will only be visible to registered users	Nathan Snow
NR-ND020	Students must register using an email having a predefined domain (E.g. cqumail.com)	Nathan Downes
NR-NM021	Educators must register using an email having a predefined domain (E.g. cqu.edu.au)	Nicholas McGuffin
NR-NS022	Email addresses must be verified before successfully registering	Nathan Snow
NR-UD023	System must allow for future remote web UI integration via API	
NR-NM024	All educators are given a moderator role by default	Nicholas
NR-NM025	Q&A forum content moderated only by users with the moderator role	Nicholas
NR-RA026	Required job/placement opportunities only visible to educators and industry partners	Rory Allen
NR-NS027	User login can be via (1) Username & Password or (2) Email & Password combinations	Nathan Snow
NR-ND028	Disciplines can only be created by users with the admin role	Nathan Downes
NR-ND029	User profile name and email fields are mandatory	Nathan Downes
NR-RA030	Jobs and placements must have at least one discipline tag	Rory Allen
NR-RA031	Jobs and placements can have many discipline tags	Rory Allen
NR-NM032	Only educators (with the moderator role) can manage industry partner expressions of interest	Nicholas McGuffin
NR-ND033	Clicking on a job/placement preview/thumbnail will open a page showing further details	Nathan Downes
NR-RA034	Job/placement applications will include details the student's profile details	Rory Allen
NR-RA035	Job/placement applications will include a message (cover letter)	Rory Allen
NR-RA036	Job/placement applications will optionally include attachments (CV, Certificates etc)	Rory Allen
NR-RA037	Submitted applications will remain on a student's record, but get archived after 1 year	Rory Allen
NR-NS038	Applications will be emailed to industry partners at a set time each day via email (batch send)	Nathan Snow
NR-NM039	Q&A forum content must be approved by a moderator before it is published	Nicholas
NR-UD040	System maintenance, updates, or upgrades should occur during quietest web traffic times (preferably Sunday) with user notice	
NR-UD041	System must be operational and online during weekdays 4am – 12pm (expected peak usage)	

NR-UD042	Cloud hosting will be used to minimize the risk of service downtime, with a minimum overall target of 98% uptime	
NR-UD043	User experience should be maximised through use of a cached content delivery network (such as Cloudflare)	
NR-UD044	Site MUST have a current SSL (Secure Sockets Layer) certificate	
NR-UD045	Platform should be scaled initially to handle a load of 1000 concurrent users	
NR-UD046	Persisted data will be stored in a cloud hosted MySQL relational database	
NR-UD047	Database content must be backed up daily	
NR-UD048	Codebase must be segregated using a monolithic modular design (Repository, Service, Configuration etc)	
NR-UD049	Application must be able to deploy to a Linux or Windows server	
NR-UD050	Application codebase must be able to run on Linux, Windows, and Mac OS	
NR-UD051	Source code to include adequate Javadoc comments so that online documentation can be generated	
NR-UD052	System must expose sufficient REST endpoints such that the system could implement an external/distributed frontend UI	

Figure 9: Non-functional Requirement Snapshot

B. Design Document

i. Design Pattern

WILMA will follow the Model-View-Controller (MVC) design pattern. Figure X below depicts how the components of the system interact.

The View consists of user facing components such as ThymeLeaf Templates and external UI components (e.g., optional native mobile user interfaces). View components have no reference to the Model components; interaction is passed between the View components and their relevant Controller components.

Controller components mediate interaction between the UI and the data model. These components include Web Controllers for handling web pages, redirects, etc., and REST Controllers for returning JSON responses. Controllers interact with the Model through injected references to Services in the Model.

The Model consists of Service, Entity, and Repository components. Entities are the data structures of the Model. Controllers interact with these Entities through Service components within the Model. Repositories interact directly with the database, and Services mediate interaction between the Controller components and the Repositories.

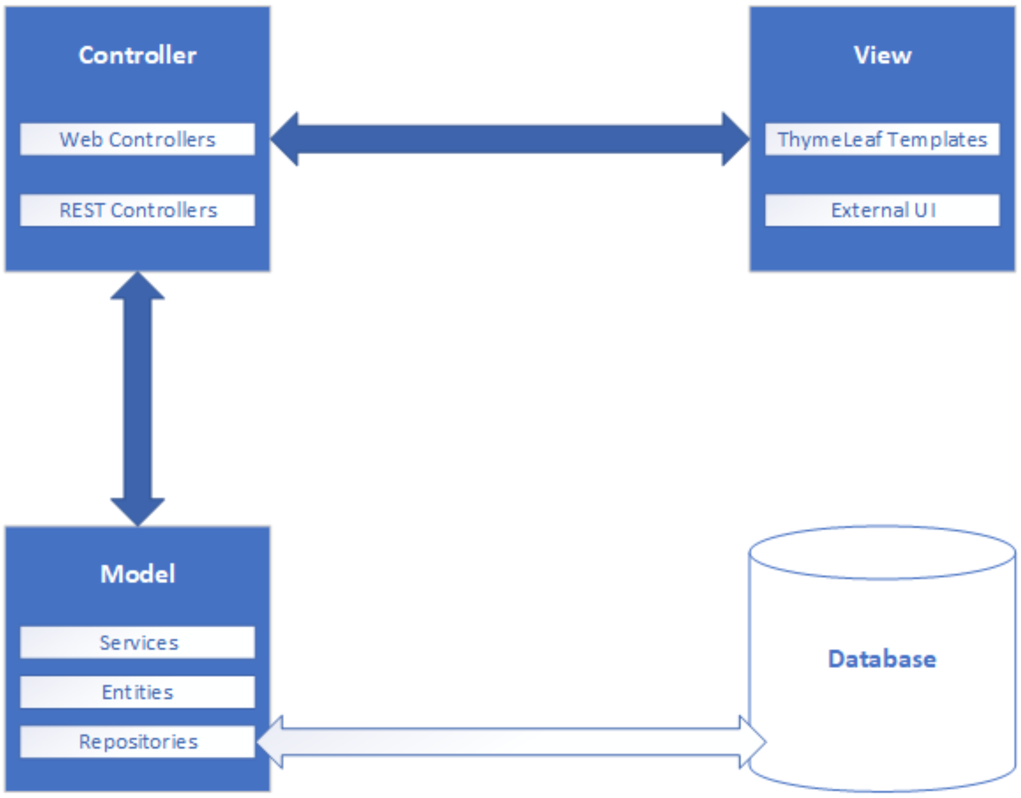


Figure 10: MVC design pattern as implemented in the WILMA project

ii. Use Case Diagrams

The level one use cases shown in this section give an abstract, high-level overview of the use cases for selected system processes.

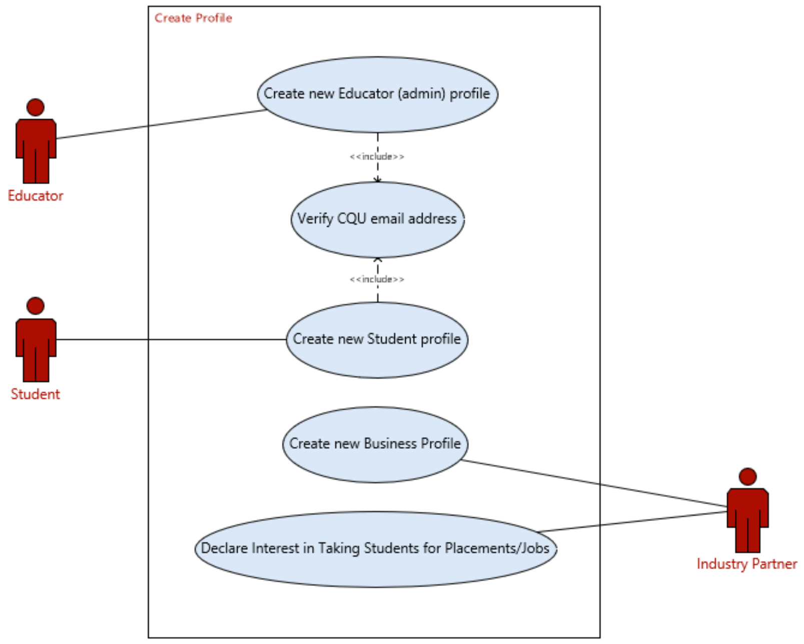

Create Profile
Create new Educator (admin) profile
<<include>>
Verify CQU email address
<<include>>
Create new Student profile
Create new Business Profile
Declare Interest in Taking Students for Placements/Jobs
Educator
Student
Industry Partner

Figure 11: The Create Profile Level 1 Use Case

The *Create Profile* use case as shown in Figure 11 outlines the profile creation process and the associated use cases for each stakeholder.

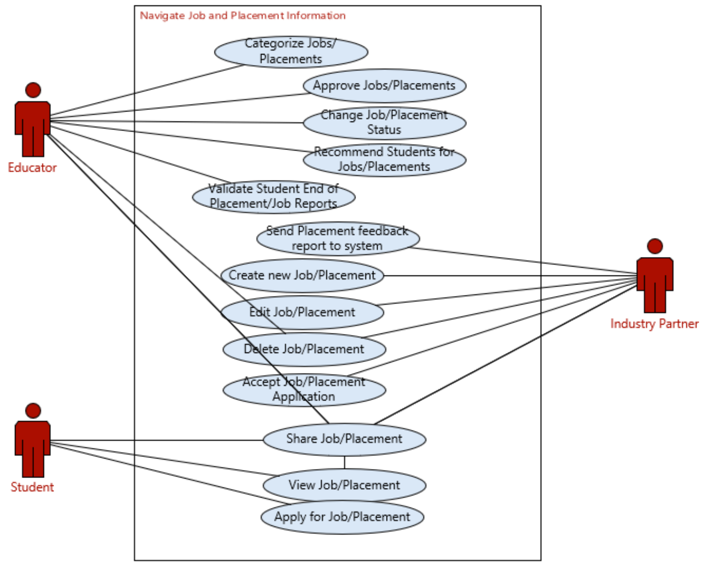

Navigate Job and Placement Information
Categorize Jobs/ Placements
Approve Jobs/Placements
Change Job/Placement Status
Recommend Students for Jobs/Placements
Validate Student End of Placement/Job Reports
Send Placement feedback report to system
Create new Job/Placement
Edit Job/Placement
Delete Job/Placement
Accept Job/Placement Application
Share Job/Placement
View Job/Placement
Apply for Job/Placement
Educator
Industry Partner
Student

Figure 12: The Navigate Job and Placement Information Level 1 Use Case

The *Navigate Job and Placement Information* use case as shown in Figure 12 outlines the job and placement information navigation process and the associated use cases for each stakeholder.

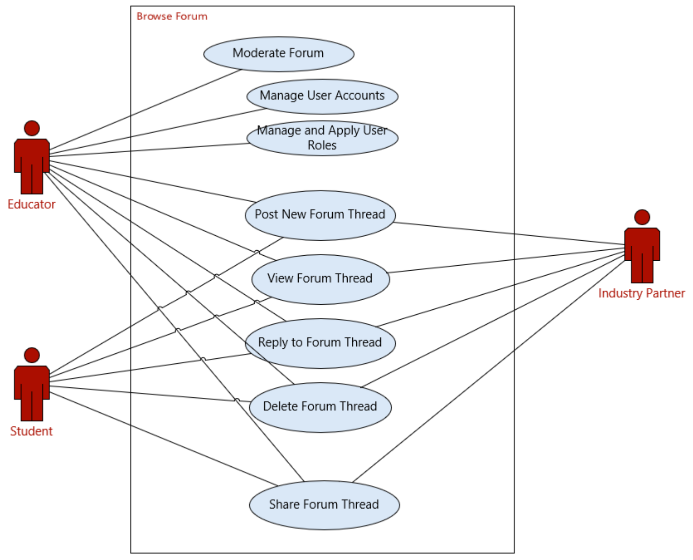


Figure 13: The Browse Forum Level 1 Use Case

The *Browse Forum* use case as shown in Figure 13 outlines the forum browsing process and the associated use cases for each stakeholder.

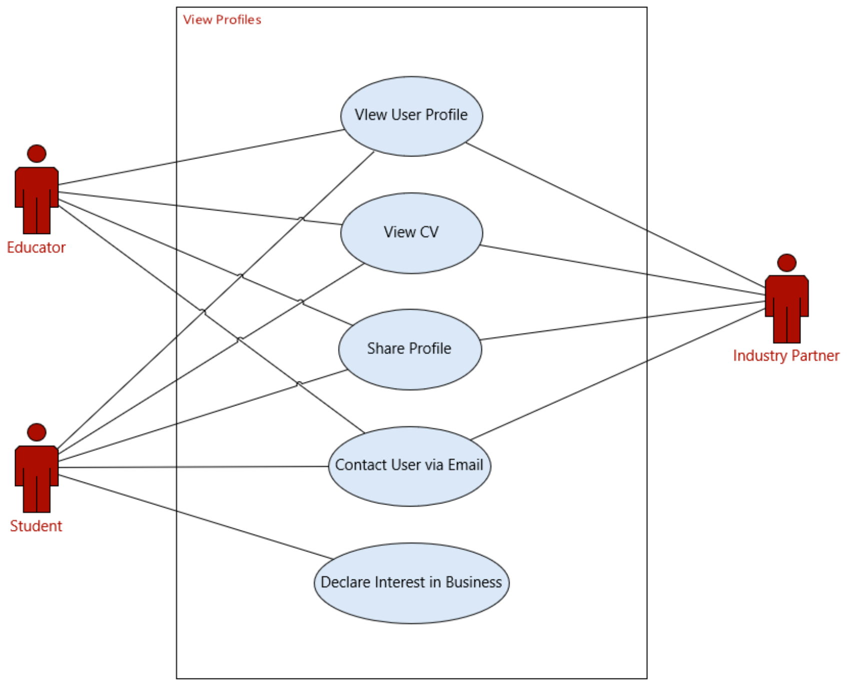


Figure 14: The View Profiles Level 1 Use Case

The *View Profiles* use case as shown in Figure 14 outlines the profile viewing process and the associated use cases for each stakeholder.

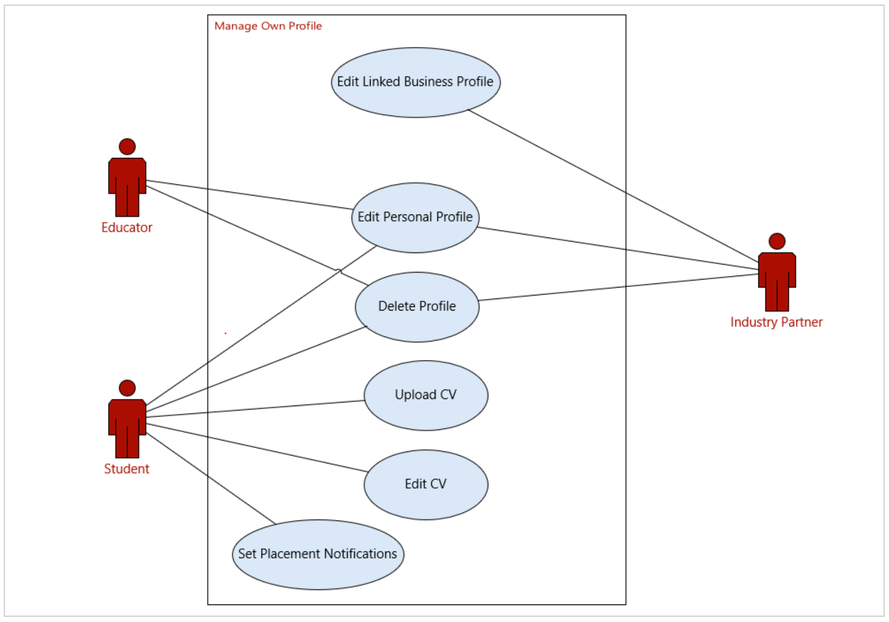
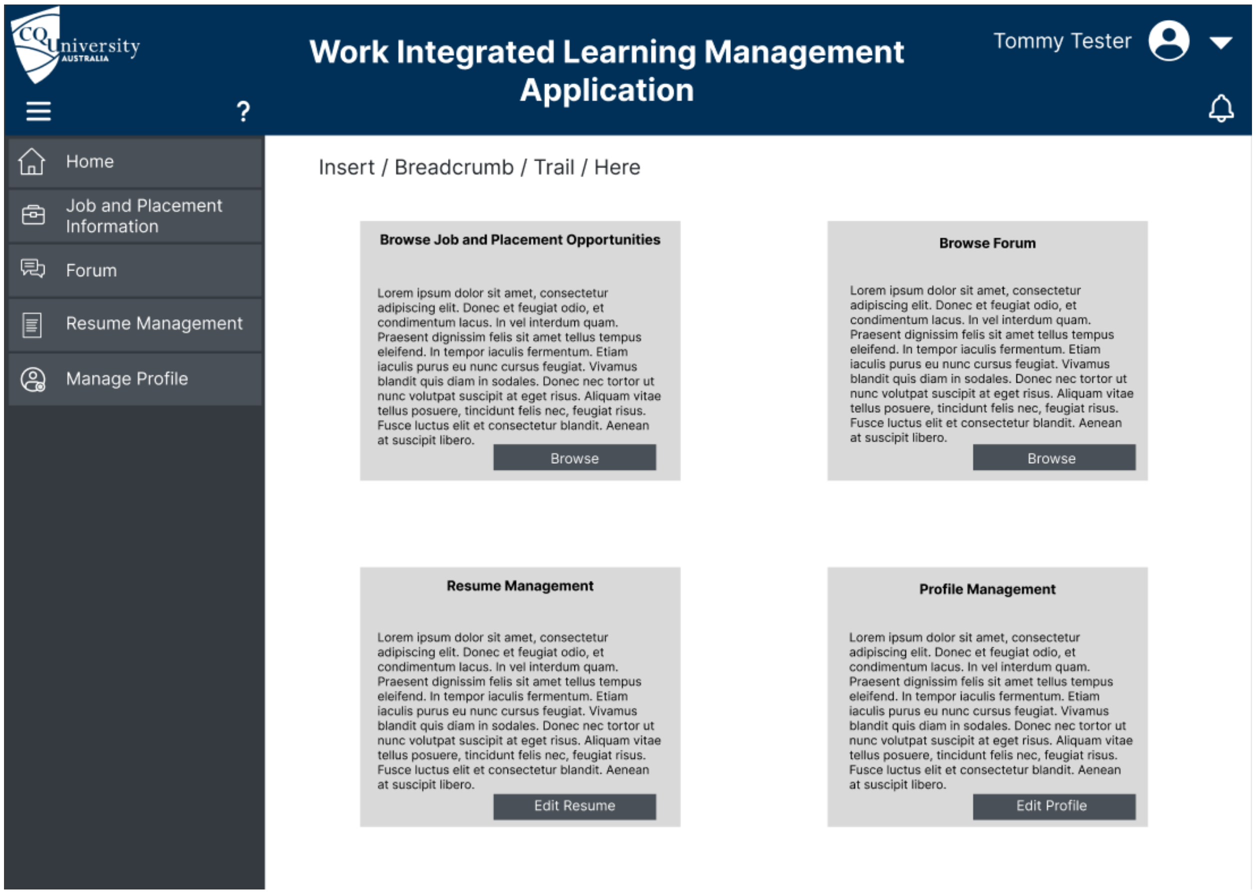


Figure 15: The Manage Own Profile Level 1 Use Case

The *Manage Own Profile* use case as shown in Figure 15 outlines the profile management process and the associated use cases for each stakeholder.

iii. User Interface Design

The user interface of the WILMA project is loosely based off the Central Queensland University Moodle design to maintain similarity and improve user transition between CQU applications.


Work Integrated Learning Management Application
Tommy Tester
?
Home
Job and Placement Information
Forum
Resume Management
Manage Profile
Insert / Breadcrumb / Trail / Here
Browse
Browse
Edit Resume
Edit Profile

Browse Job and Placement Opportunities

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec et feugiat odio, et condimentum lacus. In vel interdum quam. Praesent dignissim felis sit amet tellus tempus eleifend. In tempor iaculis fermentum. Etiam iaculis purus eu nunc cursus feugiat. Vivamus blandit quis diam in sodales. Donec nec tortor ut nunc volutpat suscipit at eget risus. Aliquam vitae tellus posuere, tincidunt felis nec, feugiat risus. Fusce luctus elit et consectetur blandit. Aenean at suscipit libero.

Browse Forum

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec et feugiat odio, et condimentum lacus. In vel interdum quam. Praesent dignissim felis sit amet tellus tempus eleifend. In tempor iaculis fermentum. Etiam iaculis purus eu nunc cursus feugiat. Vivamus blandit quis diam in sodales. Donec nec tortor ut nunc volutpat suscipit at eget risus. Aliquam vitae tellus posuere, tincidunt felis nec, feugiat risus. Fusce luctus elit et consectetur blandit. Aenean at suscipit libero.

Resume Management

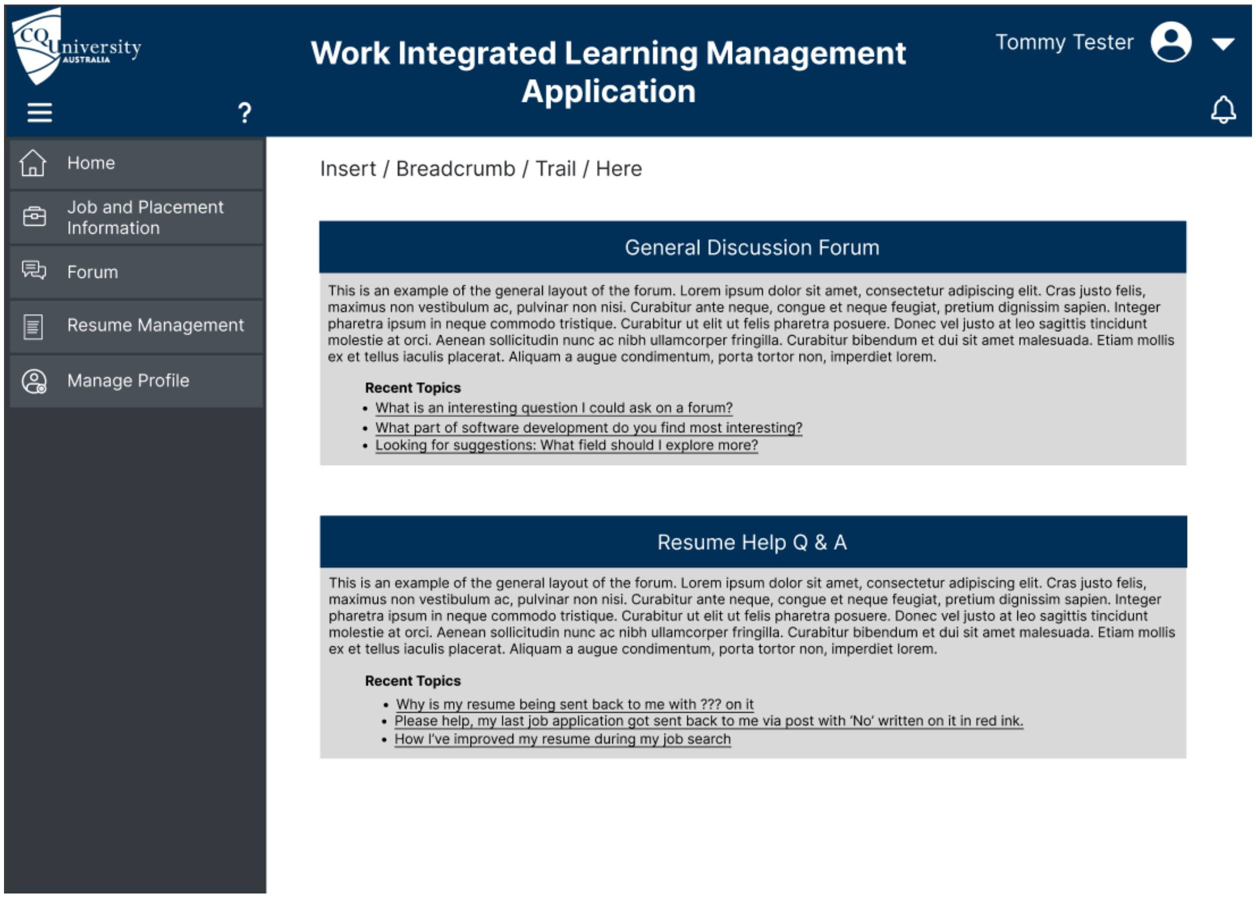
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec et feugiat odio, et condimentum lacus. In vel interdum quam. Praesent dignissim felis sit amet tellus tempus eleifend. In tempor iaculis fermentum. Etiam iaculis purus eu nunc cursus feugiat. Vivamus blandit quis diam in sodales. Donec nec tortor ut nunc volutpat suscipit at eget risus. Aliquam vitae tellus posuere, tincidunt felis nec, feugiat risus. Fusce luctus elit et consectetur blandit. Aenean at suscipit libero.

Profile Management

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec et feugiat odio, et condimentum lacus. In vel interdum quam. Praesent dignissim felis sit amet tellus tempus eleifend. In tempor iaculis fermentum. Etiam iaculis purus eu nunc cursus feugiat. Vivamus blandit quis diam in sodales. Donec nec tortor ut nunc volutpat suscipit at eget risus. Aliquam vitae tellus posuere, tincidunt felis nec, feugiat risus. Fusce luctus elit et consectetur blandit. Aenean at suscipit libero.

Figure 16: WILMA Homepage UI Design

Figure 16 shows a conceptual design for the home page user dashboard, allowing users to easily navigate through the website and never being more than a few clicks away from the home page.



This is an example of the general layout of the forum. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras justo felis, maximus non vestibulum ac, pulvinar non nisi. Curabitur ante neque, congue et neque feugiat, pretium dignissim sapien. Integer pharetra ipsum in neque commodo tristique. Curabitur ut elit ut felis pharetra posuere. Donec vel justo at leo sagittis tincidunt molestie at orci. Aenean sollicitudin nunc ac nibh ullamcorper fringilla. Curabitur bibendum et dui sit amet malesuada. Etiam mollis ex et tellus iaculis placerat. Aliquam a augue condimentum, porta tortor non, imperdiet lorem.

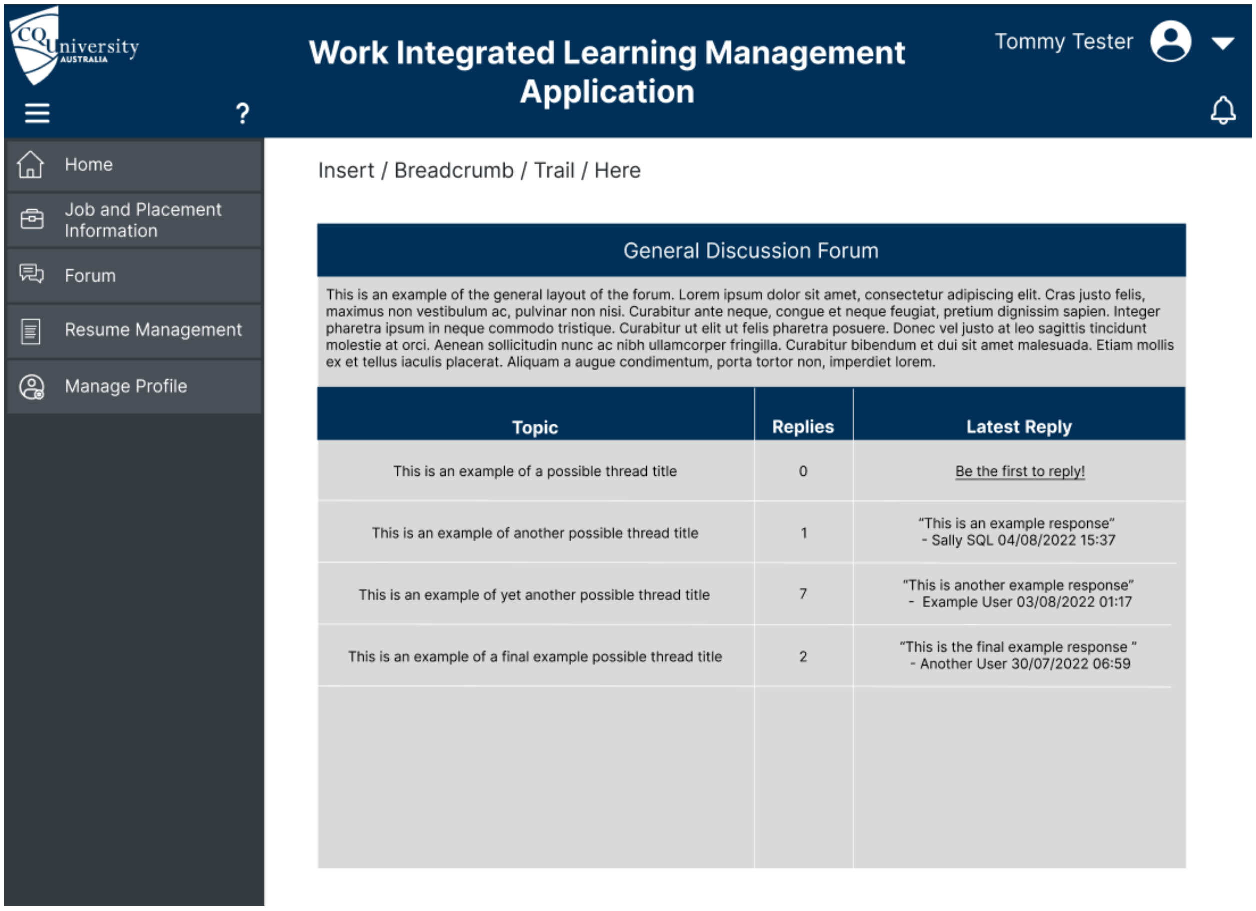
Recent Topics
- What is an interesting question I could ask on a forum?
- What part of software development do you find most interesting?
- Looking for suggestions: What field should I explore more?

This is an example of the general layout of the forum. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras justo felis, maximus non vestibulum ac, pulvinar non nisi. Curabitur ante neque, congue et neque feugiat, pretium dignissim sapien. Integer pharetra ipsum in neque commodo tristique. Curabitur ut elit ut felis pharetra posuere. Donec vel justo at leo sagittis tincidunt molestie at orci. Aenean sollicitudin nunc ac nibh ullamcorper fringilla. Curabitur bibendum et dui sit amet malesuada. Etiam mollis ex et tellus iaculis placerat. Aliquam a augue condimentum, porta tortor non, imperdiet lorem.

Recent Topics
- Why is my resume being sent back to me with ??? on it
- Please help, my last job application got sent back to me via post with 'No' written on it in red ink.
- How I've improved my resume during my job search

Figure 17: WILMA Forum Home Page UI Design

Figure 17 examines the forum home page, giving a brief overview of each forum section – General Discussion Forum, Resume Help Q & A, Software Development, Network, Business Analysis, and Project Management. Splitting the forum into sections allows for simpler topic management and makes it easier for users to find the information they need. Recent topics are also shown, and users can enter each forum section from this point to view it in its entirety.


CQUniversity AUSTRALIA
Work Integrated Learning Management Application
Tommy Tester
?
Home
Job and Placement Information
Forum
Resume Management
Manage Profile
Insert / Breadcrumb / Trail / Here

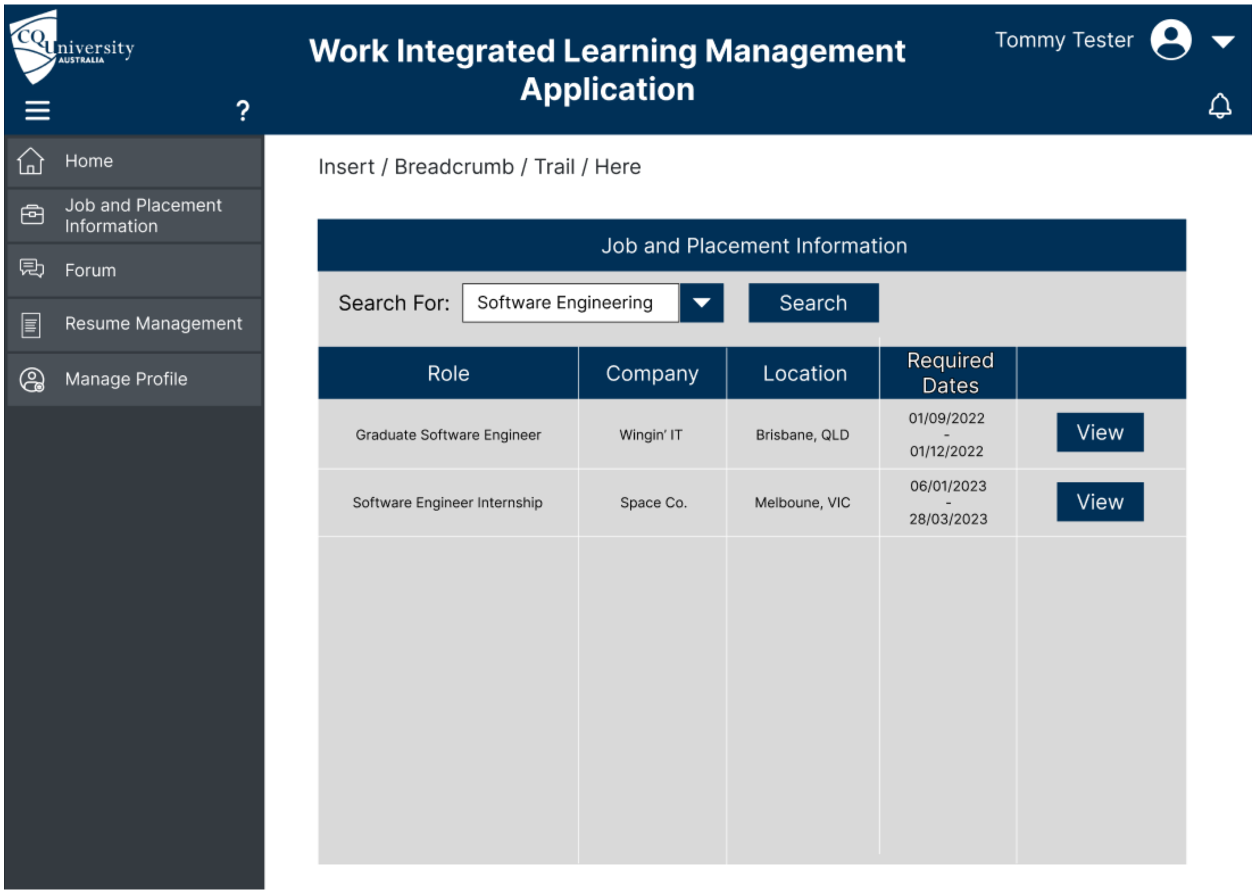
General Discussion Forum

This is an example of the general layout of the forum. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras justo felis, maximus non vestibulum ac, pulvinar non nisi. Curabitur ante neque, congue et neque feugiat, pretium dignissim sapien. Integer pharetra ipsum in neque commodo tristique. Curabitur ut elit ut felis pharetra posuere. Donec vel justo at leo sagittis tincidunt molestie at orci. Aenean sollicitudin nunc ac nibh ullamcorper fringilla. Curabitur bibendum et dui sit amet malesuada. Etiam mollis ex et tellus iaculis placerat. Aliquam a augue condimentum, porta tortor non, imperdiet lorem.

Topic	Replies	Latest Reply
This is an example of a possible thread title	0	Be the first to reply!
This is an example of another possible thread title	1	"This is an example response" - Sally SQL 04/08/2022 15:37
This is an example of yet another possible thread title	7	"This is another example response" - Example User 03/08/2022 01:17
This is an example of a final example possible thread title	2	"This is the final example response " - Another User 30/07/2022 06:59

Figure 18: WILMA Specific Forum UI Design

Figure 18 examines what a specific forum page could look like. A basic description of the forum sits at the top of the page, followed by a list of topics, the number of replies and a brief look at the latest reply. Users can click to enter a specific thread, where they can view the original post followed by every reply. They are also able to add responses to threads where required. The CQU educators are expected to be the leaders of the forum and have the added task of moderating the forum, which can include things like warning or banning users for inappropriate content, deleting posts, manage user accounts and apply user roles where applicable.


Work Integrated Learning Management Application
Tommy Tester
?
Home
Job and Placement Information
Forum
Resume Management
Manage Profile
Search For:
Search
View

Insert / Breadcrumb / Trail / Here

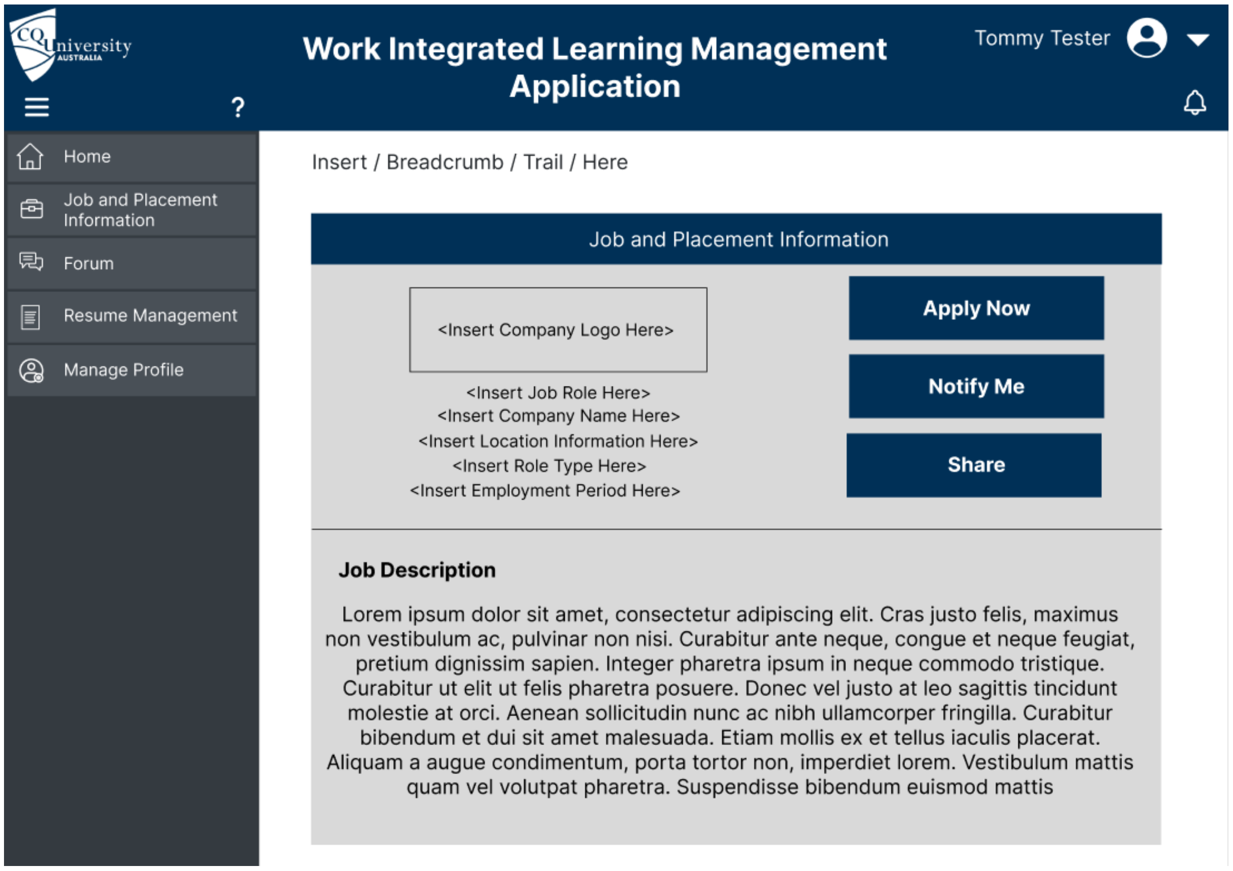
Job and Placement Information

Search For: Software Engineering

Role	Company	Location	Required Dates	
Graduate Software Engineer	Wingin' IT	Brisbane, QLD	01/09/2022 - 01/12/2022	View
Software Engineer Internship	Space Co.	Melboune, VIC	06/01/2023 - 28/03/2023	View

Figure 19: WILMA Job and Placement Information Home Page UI Design

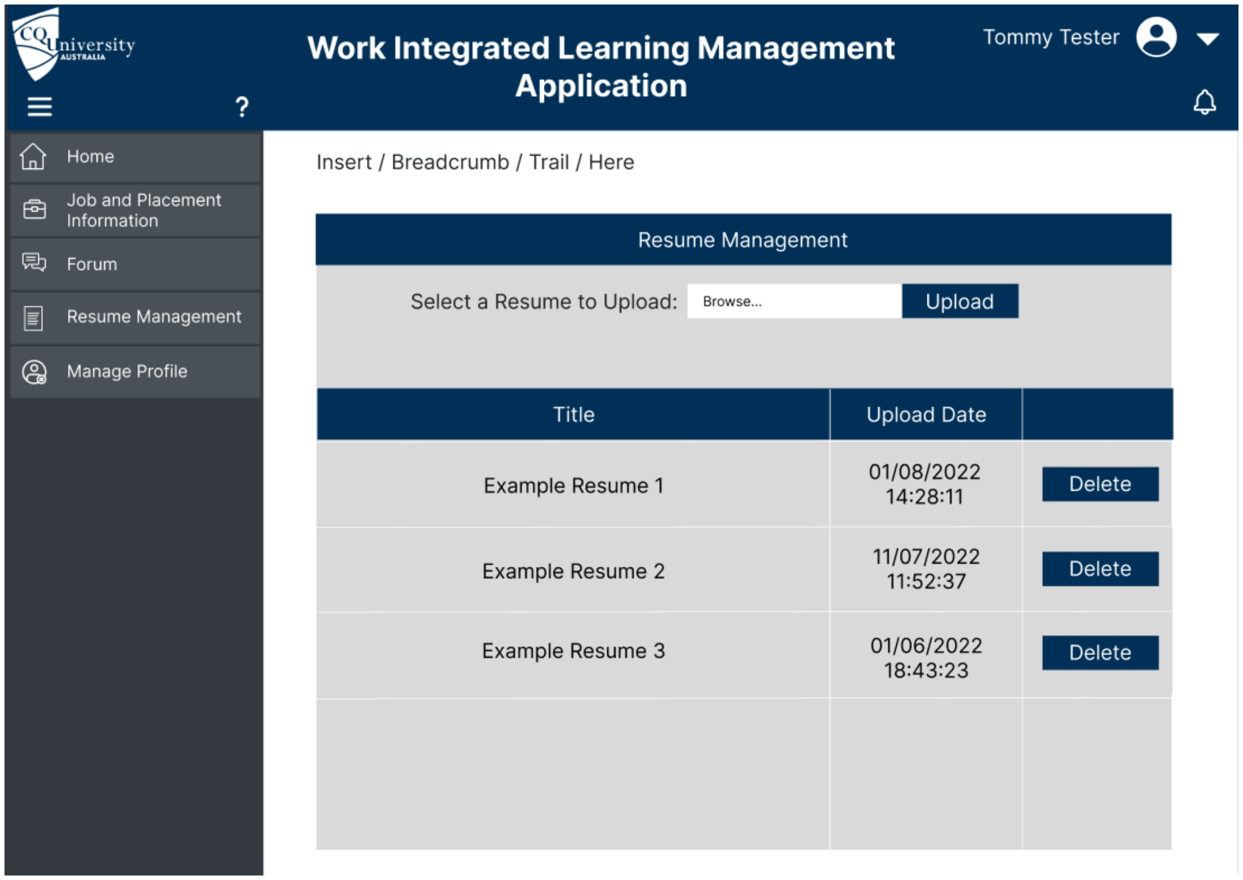
Figure 19 shows a basic Job and Placement Information page design. Users are able to search for Jobs or Placements by discipline. In the example, Software Engineering has been chosen. Users can then view a specific job opportunity by clicking the view button.


Work Integrated Learning Management Application
Tommy Tester
Home
Job and Placement Information
Forum
Resume Management
Manage Profile
Insert / Breadcrumb / Trail / Here
Job and Placement Information
<Insert Company Logo Here>
<Insert Job Role Here>
<Insert Company Name Here>
<Insert Location Information Here>
<Insert Role Type Here>
<Insert Employment Period Here>
Apply Now
Notify Me
Share
Job Description

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras justo felis, maximus non vestibulum ac, pulvinar non nisi. Curabitur ante neque, congue et neque feugiat, pretium dignissim sapien. Integer pharetra ipsum in neque commodo tristique. Curabitur ut elit ut felis pharetra posuere. Donec vel justo at leo sagittis tincidunt molestie at orci. Aenean sollicitudin nunc ac nibh ullamcorper fringilla. Curabitur bibendum et dui sit amet malesuada. Etiam mollis ex et tellus iaculis placerat. Aliquam a augue condimentum, porta tortor non, imperdiet lorem. Vestibulum mattis quam vel volutpat pharetra. Suspendisse bibendum euismod mattis

Figure 20: WILMA Job and Placement Information – Selected Job - UI Design

Figure 20 depicts a basic job and placement information page. Users can view all of the information associated with the job or placement. They are then able to receive notifications about similar roles, share, or apply for the role.


CQUniversity Australia
Work Integrated Learning Management Application
Tommy Tester
?
Home
Job and Placement Information
Forum
Resume Management
Manage Profile
Insert / Breadcrumb / Trail / Here
Resume Management
Select a Resume to Upload:
Browse...
Upload

Title	Upload Date	
Example Resume 1	01/08/2022 14:28:11	Delete
Example Resume 2	11/07/2022 11:52:37	Delete
Example Resume 3	01/06/2022 18:43:23	Delete

Figure 21: WILMA Resume Management UI Design

Figure XX examines the resume management page. Students can add a new resume in PDF or docx format to the page and a list of existing resume uploads will be shown. It is also possible for them to delete resumes where necessary. Having a resume management page lets students upload different resumes for different job opportunities and disciplines and is an important aspect of the resume management process. This page will only be visible to students – the resumes that have been uploaded will be visible on a user's page, which can be accessed by clicking on their name or profile picture when they apply for a position or post on the forum.

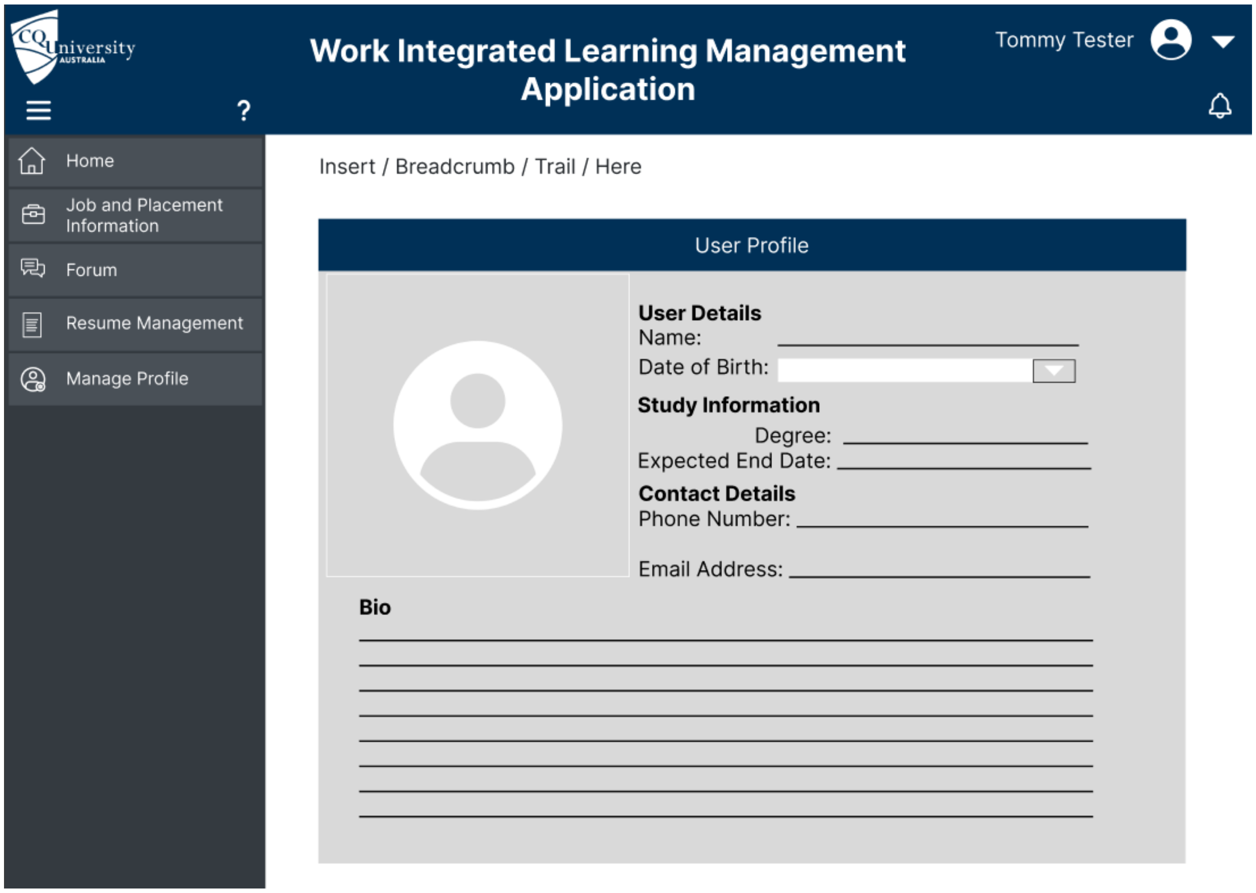


Figure 22: WILMA User Profile UI Design

Figure 22 shows an example of a possible User Profile design. All user information including a picture would be included and could be edited on the page where required.

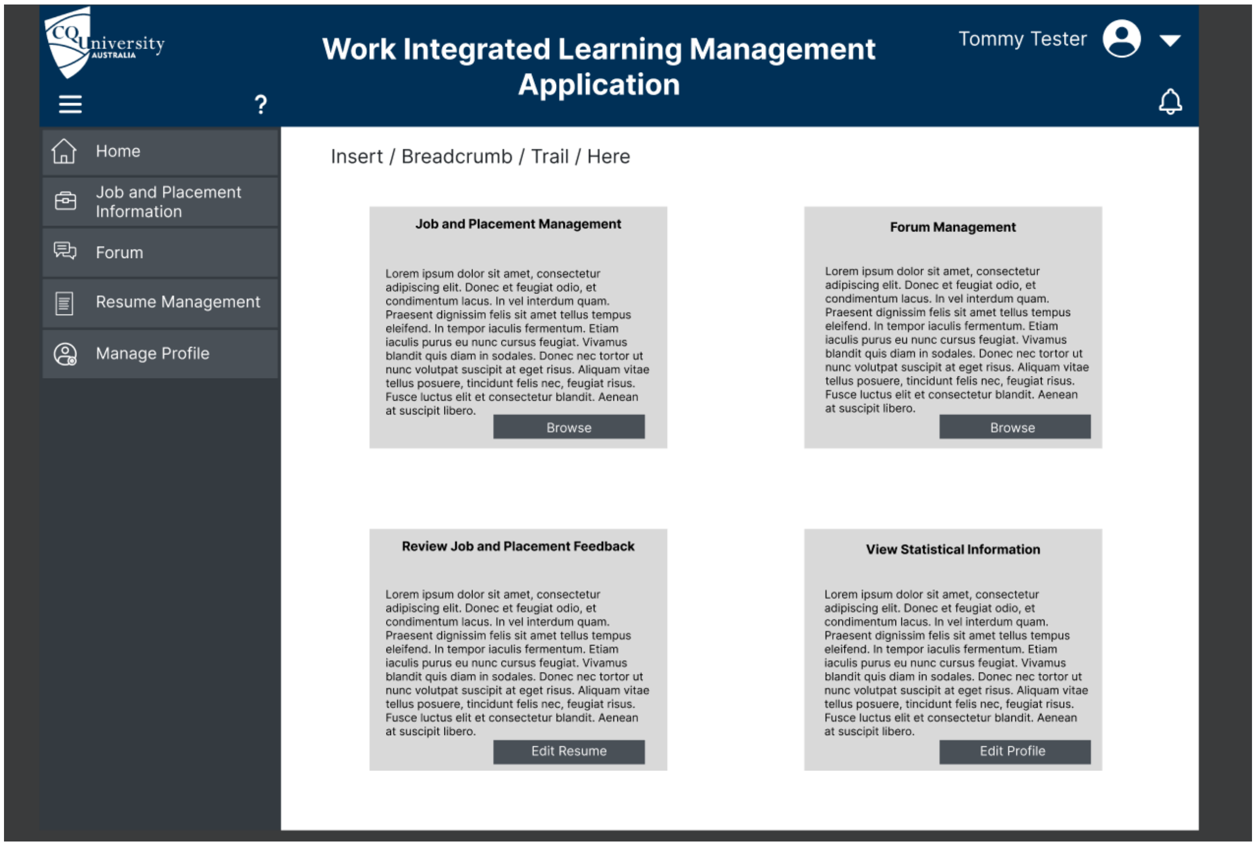

Work Integrated Learning Management Application
Tommy Tester
Home
Job and Placement Information
Forum
Resume Management
Manage Profile
Insert / Breadcrumb / Trail / Here
Job and Placement Management
Browse
Forum Management
Browse
Review Job and Placement Feedback
Edit Resume
View Statistical Information
Edit Profile

Figure 23: WILMA Educator Dashboard UI Design

Figure 23 shows a potential design for the Educator dashboard. This design is a slight variation from the home page dashboard to maintain a consistent system view for all users. It gives educators a dashboard that offers easy access to the management tasks they will require on a regular basis.

iv. Sequence Diagrams

The functional requirements of the WILMA project depend upon the interaction between multiple system components. Figure 24 below shows the sequence of messages passed between the components involved in the creation of a new user.

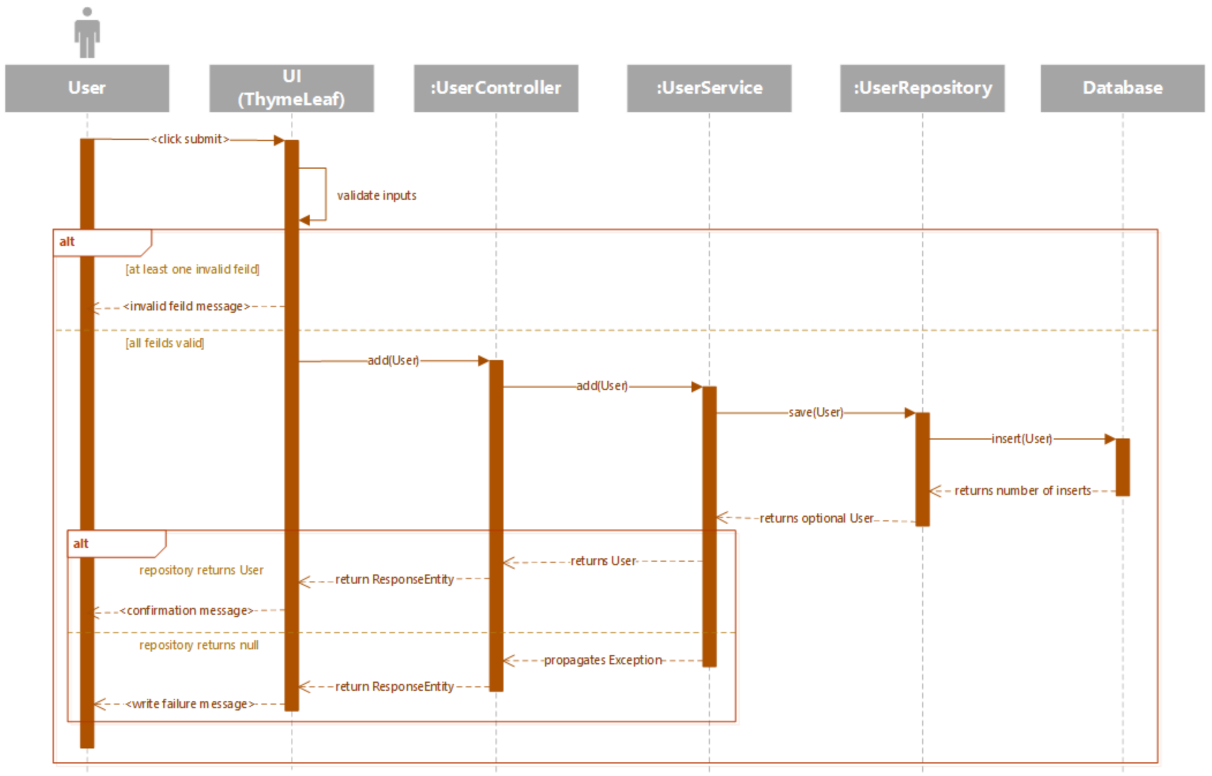


Figure 24: Sequence Diagram depicting new user creation process

Once the user has input their details into the new user creation form and clicked on the submit button, the values in the form are validated client-side. If any fields are invalid, an appropriate message is generated and presented. If, however, all fields are valid, the user data is passed to the UserController component via the add(User) method. The UserController calls the add(User) method of the injected UserService component, which, in turn, calls the save(User) method of the injected UserRepository component. This save(User) method generates the appropriate SQL statement, in this case an INSERT to update the Database, i.e., adds a new line (a new user) to the User table.

The prepared SQL statements returns the number of inserts. The repository will return an optional User entity at this point.

If the database operation returns 0, the repository will return *null*. In this case, the service propagates an exception back to the controller. The controller then generates a ResponseEntity containing an error message to return to the UI, And the UI displays an appropriate write failure message to the user.

If, however, the database returns 1 (the number of inserts), the repository returns the User entity that was just written. In this case, the service returns the User, and the controller returns a ResponseEntity set to *ok*. The UI then displays a success message to the user.

The example depicted in Figure 24 shows the flow of data down the stack, from the user to the database, and back up from the database to the user. This flow is representative all operations in the WILMA software project. The differences for other specific use cases are explained below.

When users update their details, the initial client-side validation is identical. Upon validation, however, update(User) is called on the controller and consequently on the service. The service still calls the save(User) method on the repository, which handles both add(User) and update(User) methods of the service, generating INSERT and UPDATE SQL statements, respectively. The return of an UPDATE statement, however, is the updated entity (as opposed to the number of inserts). This propagates back up the stack identically to the add new user process, resulting in either a success or failure message presented to the user.

Delete processes (such as users deleting their accounts, or as allowed to Educator/Admin roles for moderation of the forums and job opportunities) follow an almost identical sequence. In place of the client-side validation sequence, there is a simple confirmation prompt that, once confirmed, triggers a delete sequence down the stack, ending with deletion of an entry in the database. Success and failure are propagated back up the stack as per the add and update sequences.

Read sequences are similar, but they have no client-side validation or confirmation sequence. Parameters are passed down the stack through parametrised get() methods, ending in SELECT SQL statements generated by the appropriate repository. Appropriate entities are returned up the stack until the controller returns a ResponseEntity containing the appropriate data to be displayed to the user.

For example (see Figure 25 below), if a user filters forum entries by the tag “design patterns”, the controller will call the getPostsTagged(String tag) method of the ForumService. This will in turn call the appropriate repository method which will build the SQL SELECT statement to query the database. This will return all entries that match the filter criteria in the Posts table. The repository will return a set of Post entities back up the stack, until the ForumController generates and returns a ResponseEntity, and the results are displayed by the UI.

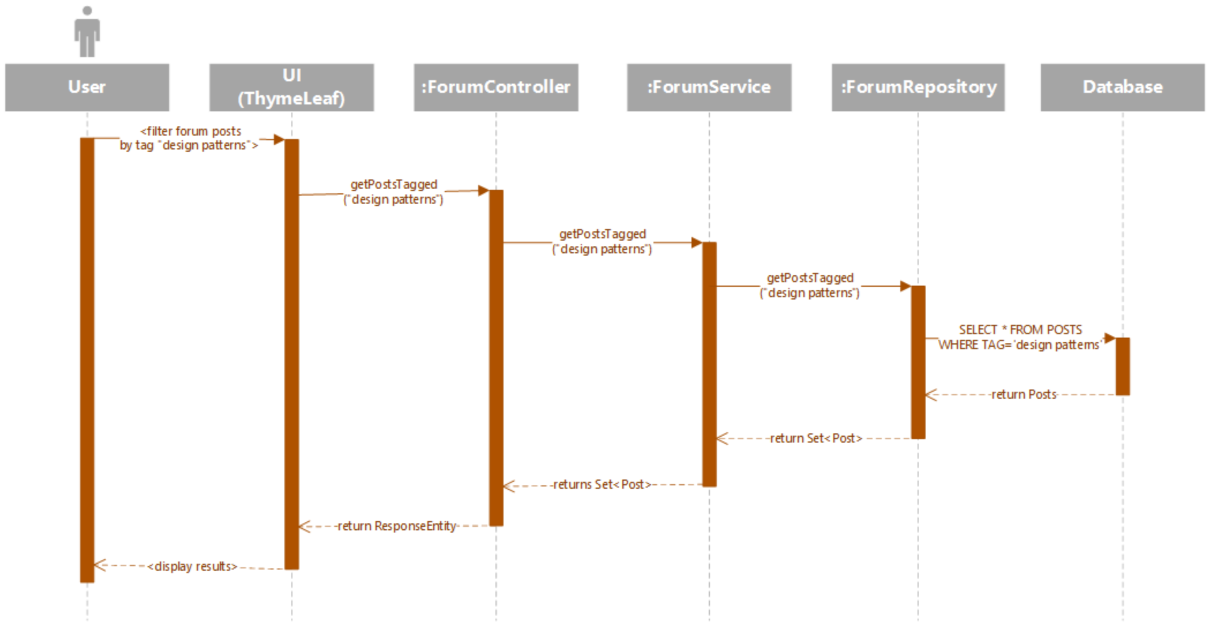

User
UI (ThymeLeaf)
:ForumController
:ForumService
:ForumRepository
Database
<filter forum posts by tag "design patterns">
getPostsTagged ("design patterns")
getPostsTagged ("design patterns")
getPostsTagged ("design patterns")
SELECT * FROM POSTS WHERE TAG='design patterns'
return Posts
return Set<Post>
returns Set<Post>
return ResponseEntity
<display results>

Figure 25: Sequence Diagram depicting filtering forum entries by tag process

One use case that cannot be covered with a sequence diagram is the user log in process. This is handled almost exclusively by the Spring Security framework. The framework provides a log in endpoint (*/login*) and manages user roles and access behind the scenes. User passwords are hashed using Bcrypt encryption, and encryption/decryption is handled by the framework.

v. UML Class Diagrams

Our project uses a UML (Unified Modelling language) Class Diagram. Class Diagrams show the structure of the system and how each class relates to each other. The classes contain attributes, operations and relationships.

For the WILMA System our entity classes are broken up into many packages, but this report will discuss the main three entity packages being, (1) Positions, (2) Forums, and (3) Accounts. These packages contain both superclasses and child classes to minimise code duplication through effective use of inheritance, each having their own related service and controller to control the flow of data throughout the stack which also helps increase code security through abstraction

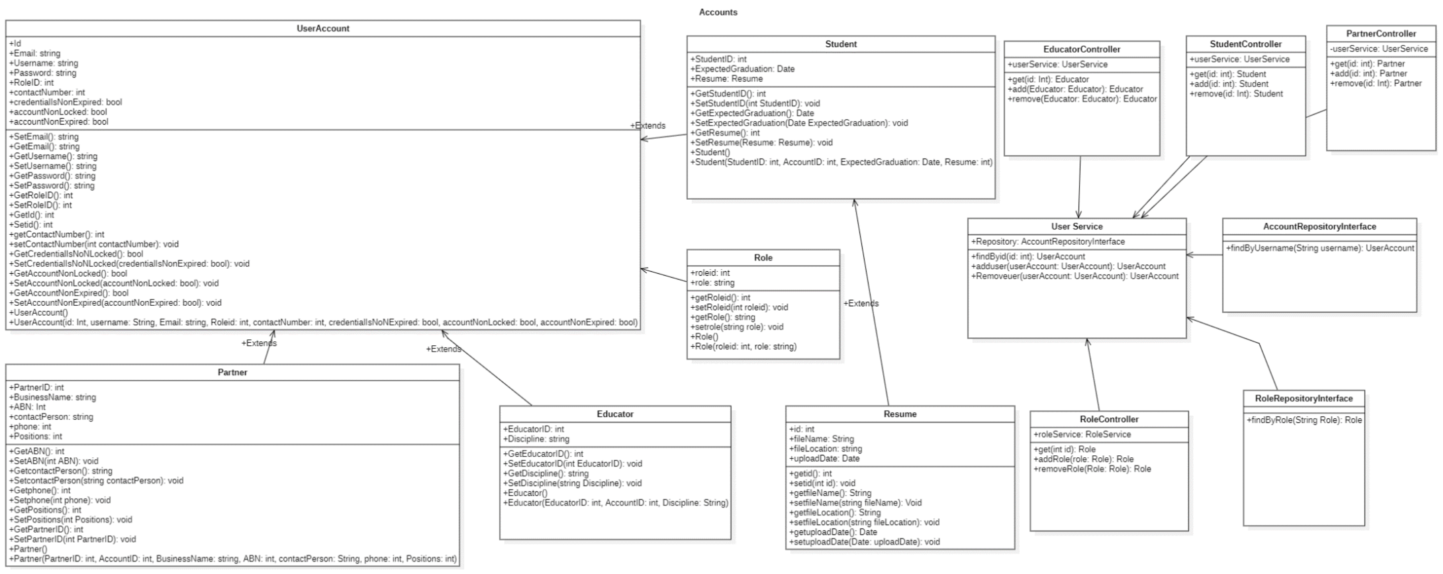
Our first Package, Accounts, contains a superclass of UserAccount and three base classes being Partner, Educator, Student. The superclass contains base information such as the users' credentials and account status, whilst Partner, Educator and Student contain variables that specifically relate to those classes such as Category for educators, Resumes for students, and business details for Partners. A one-to-many relationship is implemented between students and resumes allowing one student to have many resumes, but each resume belongs with just one student. The Role class contains Role information such the role and ID allowing for the creation of user roles allowing role based authentication and granted authorities, for example Educators will be given the role of 'ADMIN' which will grant them elevated administrator privileges. There is a UserService which, although in a different module, implements a repository interface and adds functionality applicable to all of the inheriting user groups. For instance the UserService has the ability to find Users by their ID and add and remove Users. Similarly, the UserController (REST API) contains endpoints that will make use of the UserService methods to expand the system functionality, particularly with regards to basic CRUD (create, read, update, and delete) tasks.

The Forum package consists of the ForumContent superclass which contains information relating to all forum discussions such as the time of a post or reply, and the author. The classes Reply and Post inherit from this superclass with Reply allowing users to reply to forum posts, and the Post class acts as a main discussion thread in the forum and is decorated with a title and an appropriate category and (optionally) a list of tags.

Similar to above, the ForumController provides an access point for performing CRUD operations on forum related objects, which relies on the ForumService to provide method implementation and business logic to get the tasks done.

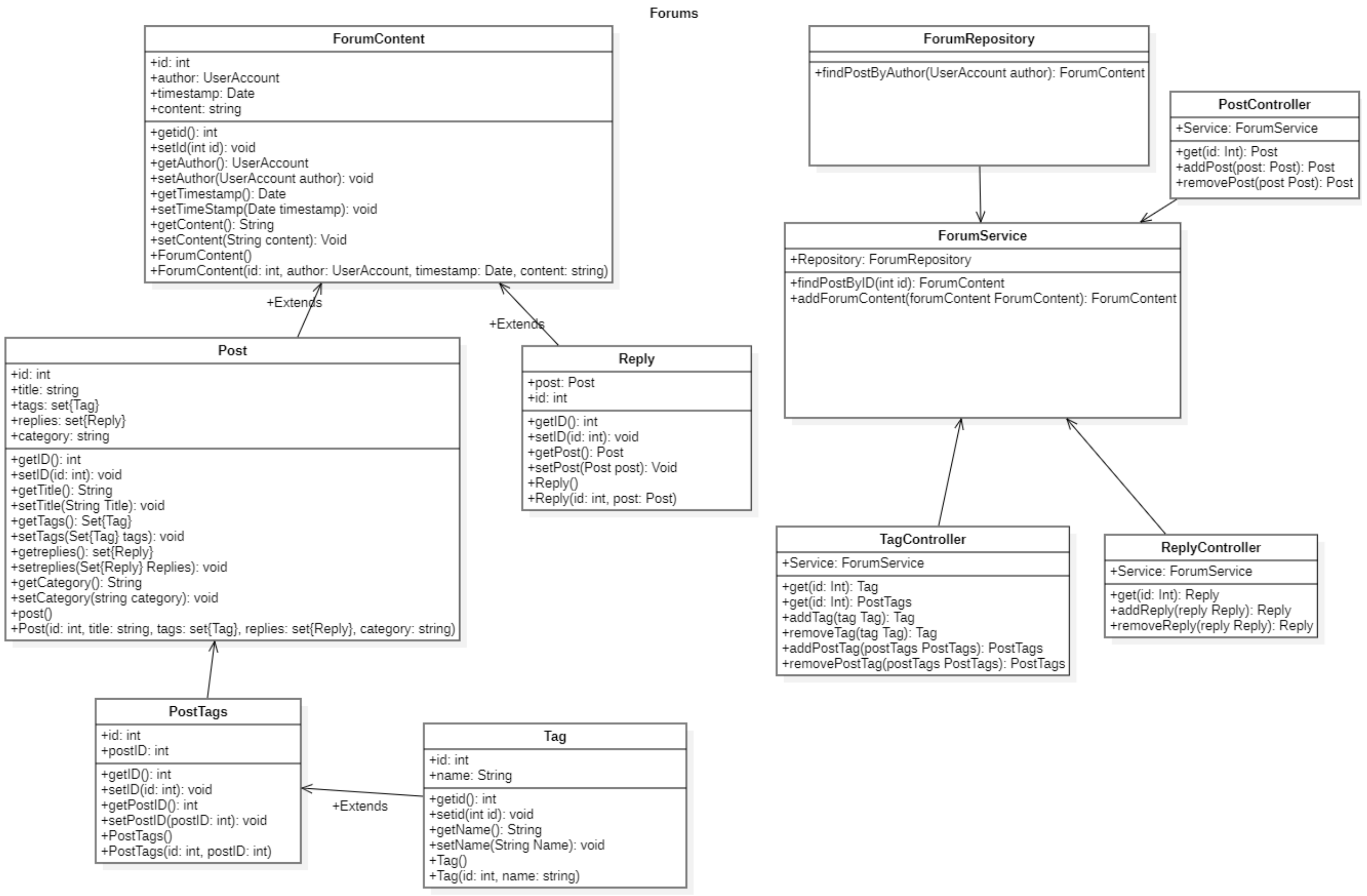
The Final Package to be discussed is the Positions Package. This contains a Position superclass containing all the relevant information to any position, such as the start and end dates (and/or duration period), a description of what the position requires, and the location at which the position is based to name a few. The Placement Class inherits from Position and defines additional fields such as the supervising educator, and a field to indicate it's completion. The Application Class contains the applicantID , ResumeID and the JobID. This class allows users to have applications for jobs. The Job Class allows for the creation of jobs and contains variables, such as Pay Rates and amounts. The Position Category class allows for Categories and extends Position, ExpressionOfInterest extends PositionCategory and allows the user to put up an Expression of Interest for a Position. The class contains variables to identify the author and category and whether they are a current student. The EOIResponse class allows for a response to the Expression of Interest to approve or deny them.

The PositionReport class contains variables for each of the 3 UserAccount classes (Educator, Partner, and Student) and enables the review of a student following a placement, allowing for feedback to be given from both the Partner and Educator about the Student's performance. This all ties into the Position Service via various position controllers, each of these controllers enabling CRUD operations of their respective child class.



Accounts
UserAccount
+Id
+Email: string
+Username: string
+Password: string
+RoleID: int
+contactNumber: int
+credentialIsNonExpired: bool
+accountNonLocked: bool
+accountNonExpired: bool
+SetEmail(): string
+GetEmail(): string
+GetUsername(): string
+SetUsername(): string
+GetPassword(): string
+SetPassword(): string
+GetRoleID(): int
+SetRoleID(): int
+GetId(): int
+Setid(): int
+getContactNumber(): int
+setContactNumber(int contactNumber): void
+GetCredentialIsNoNLocked(): bool
+SetCredentialIsNoNLocked(credentialIsNonExpired: bool): void
+GetAccountNonLocked(): bool
+SetAccountNonLocked(accountNonLocked: bool): void
+GetAccountNonExpired(): bool
+SetAccountNonExpired(accountNonExpired: bool): void
+UserAccount()
+UserAccount(id: Int, username: String, Email: string, Roleid: int, contactNumber: int, credentialIsNoNExpired: bool, accountNonLocked: bool, accountNonExpired: bool)
+Extends
+Extends
+Extends
+Extends
Partner
+PartnerID: int
+BusinessName: string
+ABN: Int
+contactPerson: string
+phone: int
+Positions: int
+GetABN(): int
+SetABN(int ABN): void
+GetcontactPerson(): string
+SetcontactPerson(string contactPerson): void
+Getphone(): int
+Setphone(int phone): void
+GetPositions(): int
+SetPositions(int Positions): void
+GetPartnerID(): int
+SetPartnerID(int PartnerID): void
+Partner()
+Partner(PartnerID: int, AccountID: int, BusinessName: string, ABN: int, contactPerson: String, phone: int, Positions: int)
Educator
+EducatorID: int
+Discipline: string
+GetEducatorID(): int
+SetEducatorID(int EducatorID): void
+GetDiscipline(): string
+SetDiscipline(string Discipline): void
+Educator()
+Educator(EducatorID: int, AccountID: int, Discipline: String)
Student
+StudentID: int
+ExpectedGraduation: Date
+Resume: Resume
+GetStudentID(): int
+SetStudentID(int StudentID): void
+GetExpectedGraduation(): Date
+SetExpectedGraduation(Date ExpectedGraduation): void
+GetResume(): int
+SetResume(Resume: Resume): void
+Student()
+Student(StudentID: int, AccountID: int, ExpectedGraduation: Date, Resume: int)
Role
+roleid: int
+role: string
+getRoleid(): int
+setRoleid(int roleid): void
+getRole(): string
+setrole(string role): void
+Role()
+Role(roleid: int, role: string)
Resume
+id: int
+fileName: String
+fileLocation: string
+uploadDate: Date
+getid(): int
+setid(int id): void
+getfileName(): String
+setfileName(string fileName): Void
+getfileLocation(): String
+setfileLocation(string fileLocation): void
+getuploadDate(): Date
+setuploadDate(Date: uploadDate): void
EducatorController
+userService: UserService
+get(id: Int): Educator
+add(Educator: Educator): Educator
+remove(Educator: Educator): Educator
StudentController
+userService: UserService
+get(id: int): Student
+add(id: int): Student
+remove(id: Int): Student
PartnerController
-userService: UserService
+get(id: int): Partner
+add(id: int): Partner
+remove(id: Int): Partner
User Service
+Repository: AccountRepositoryInterface
+findByid(id: int): UserAccount
+adduser(userAccount: UserAccount): UserAccount
+Removeuer(userAccount: UserAccount): UserAccount
AccountRepositoryInterface
+findByUsername(String username): UserAccount
RoleController
+roleService: RoleService
+get(int id): Role
+addRole(role: Role): Role
+removeRole(Role: Role): Role
RoleRepositoryInterface
+findByRole(String Role): Role

Figure 26:Class Diagram #1

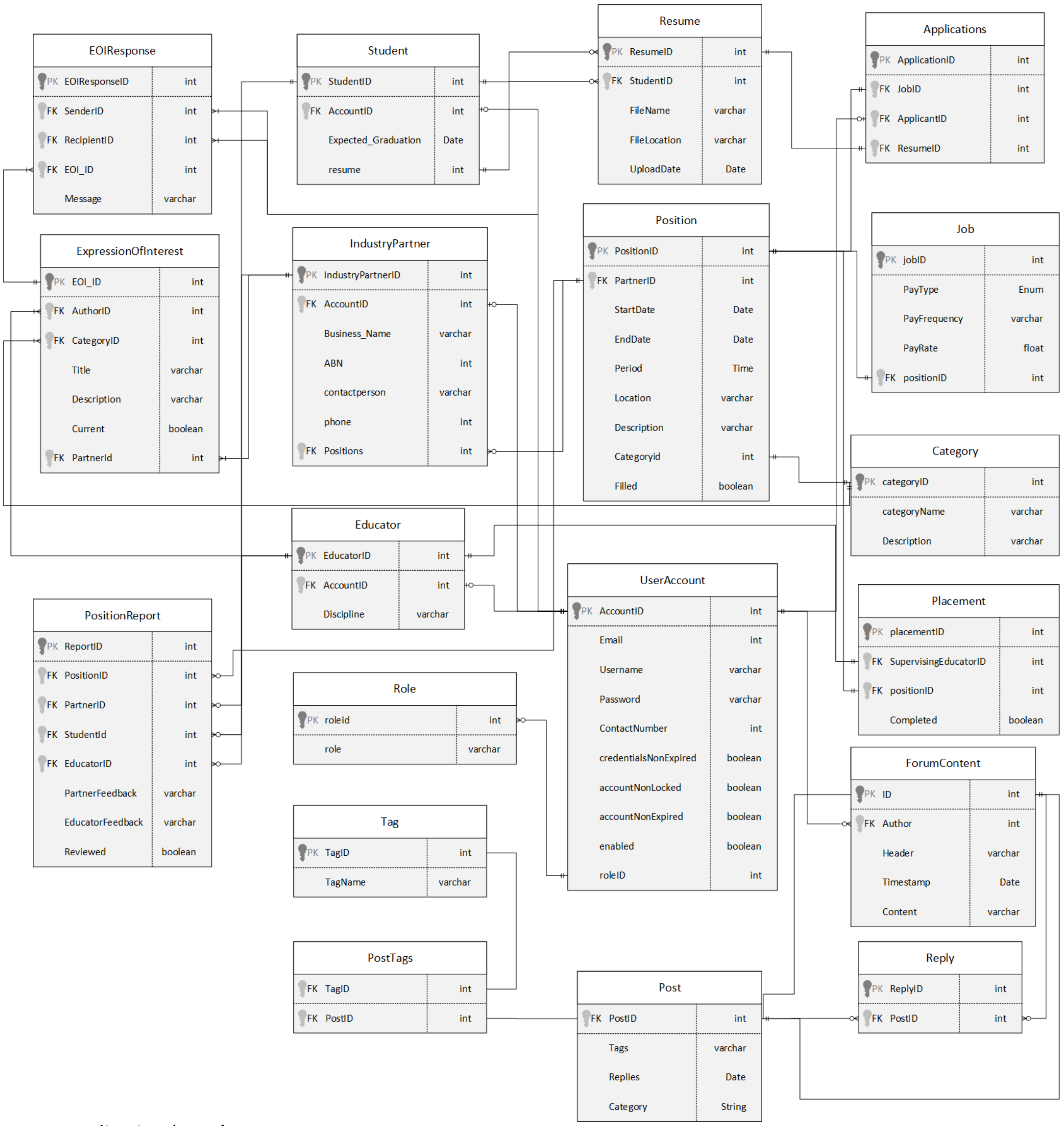


Forums
ForumContent
+id: int
+author: UserAccount
+timestamp: Date
+content: string
+getid(): int
+setId(int id): void
+getAuthor(): UserAccount
+setAuthor(UserAccount author): void
+getTimestamp(): Date
+setTimeStamp(Date timestamp): void
+getContent(): String
+setContent(String content): Void
+ForumContent()
+ForumContent(id: int, author: UserAccount, timestamp: Date, content: string)
+Extends
+Extends
Post
+id: int
+title: string
+tags: set{Tag}
+replies: set{Reply}
+category: string
+getID(): int
+setID(id: int): void
+getTitle(): String
+setTitle(String Title): void
+getTags(): Set{Tag}
+setTags(Set{Tag} tags): void
+getreplies(): set{Reply}
+setreplies(Set{Reply} Replies): void
+getCategory(): String
+setCategory(string category): void
+post()
+Post(id: int, title: string, tags: set{Tag}, replies: set{Reply}, category: string)
Reply
+post: Post
+id: int
+getID(): int
+setID(id: int): void
+getPost(): Post
+setPost(Post post): Void
+Reply()
+Reply(id: int, post: Post)
PostTags
+id: int
+postID: int
+getID(): int
+setID(id: int): void
+getPostID(): int
+setPostID(postID: int): void
+PostTags()
+PostTags(id: int, postID: int)
+Extends
Tag
+id: int
+name: String
+getid(): int
+setid(int id): void
+getName(): String
+setName(String Name): void
+Tag()
+Tag(id: int, name: string)
ForumRepository
+findPostByAuthor(UserAccount author): ForumContent
PostController
+Service: ForumService
+get(id: Int): Post
+addPost(post: Post): Post
+removePost(post Post): Post
ForumService
+Repository: ForumRepository
+findPostByID(int id): ForumContent
+addForumContent(forumContent ForumContent): ForumContent
TagController
+Service: ForumService
+get(id: Int): Tag
+get(id: Int): PostTags
+addTag(tag Tag): Tag
+removeTag(tag Tag): Tag
+addPostTag(postTags PostTags): PostTags
+removePostTag(postTags PostTags): PostTags
ReplyController
+Service: ForumService
+get(id: Int): Reply
+addReply(reply Reply): Reply
+removeReply(reply Reply): Reply

Figure 27:Class Diagram #2

vi. ER Diagrams

For our project we are using an ERD (Entity Relationship diagram) to illustrate the relation between entities in a database. The database is what contains all the information that we are submitting from the WILMA system, which is then used to hold and process information, effectively giving our



application 'state'.

In line with the examples given in the UML class diagram section, we will focus on the user accounts, forums, and positions sections.

Figure 28: Entity Relationship Diagram

The UserAccount entity holds various shard information across all user types (Partner, Educator, and Student) including values relative to their account status, authentication, and authorisation.
Subclasses then contain details relative to only them including a ‘discipling’ for educators, business details for IndustryPartners, an expected graduation date for Students and so on.

A relationship exists between a Student and a Resume whereby a student can be associated with many resumes, but each resume can only be associated to one Student, giving us a one-to-many relationship which we will map via a join table appropriately named “student_resumes”.
The forum consists of ForumContent, which encompasses both Posts and Replies. As a superclass, the ForumContent stores data about the author, timestamp, and content, which all forum discussions must have.
While Replies hold no data in addition to the content held by the ForumContent superclass beyond a PostID for the post they are replying to, Posts contain information such as a list of tags, a category so it can be appropriately grouped, a list of replies, and of course a title.

Similarly, the Position table has information regarding to the relevant position, such as the start and end date, location and position description.
A Position has a relation to an Industry Partner account who can post many positions. The Job Table contains information that is pay related and extends Position. Each Position maps to a Category which defines a name and description used to group similar positions.
Expressions of Interest also contain a category as can be noted in the ExpressionOfInterest table. This table allows Partners to express their interest towards providing a Placement and contains information relating to the category, the providing Industry Partner, and of course a description and title.

Industry Partners can then respond to an EOI via the EOIResponse table which holds information regarding the Partner, Educator and ExpressionOfInterest ID as well as a message that can be used to propose any changes or restrictions, for example regarding availability.
Students may also apply for a Job as mapped by the Applications table which stores data such as mapping to their resume, their profile and the specified job being applied for.
If a student gets accepted for a Placement, that information is submitted into the placement table.
The placement table consists of ID’s mapping to the Position and Supervising Educator as well as a boolean field to mark the completion of the Placement.
The Position Report table exists to enable feedback about a student following a successful Placement.
This table consists of ID’s for the Placement, Partner, Student and Educator. In addition to this, it also contains PartnerFeedback and EducatorFeedback, which contains a varchar of the feedback given, and a reviewed Boolean to allow for confirmation that the feedback has been reviewed.
The idea being such that a Partner can leave feedback about a student and their time during the placement, then the educator would review the feedback from the partner before releasing the final report to the student, having feedback from both the partner and the educator.

C. Updated Configuration Management Plan

i. Test-driven Software Development

Project development will follow a process that has evolved from the early concepts of extreme programming known as test-driven development (TDD) (Wikipedia 2022). TDD is a clear and predictable way to develop code as it promotes numerous programming best-practices such as (Beck 2002):

- It links design with code as you can use your user stories as test methods.
- The functions and results of unit tests dictate the code implementation.
- Provides a clear image of when coding is finished, and what remains to be done.
- It is intended to develop the least code possible to achieve a task.
- It is claimed that projects following a test-driven development process are likely to have a lower defect density of those that do not.
- Testing first ensures no code is left untested, and also keeps developers responsible for their own code and tests.

ii. Agile Processes

The WILMA SDLC (system development lifecycle) will follow the typical agile methodology, but including the TDD process into the development phase, specifically as shown in the cycle diagram below.

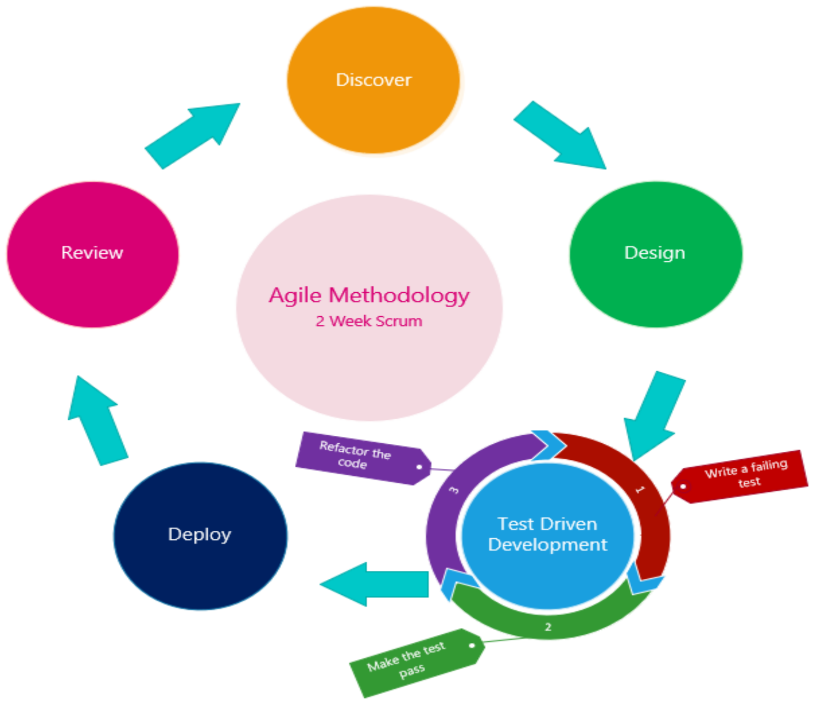


Figure 29: Agile Methodology Sprint Processes

1. Discover
   Work scope, stakeholders, and user stories will be defined.

2. Design
   UI/UX design, and any UML diagrams.

3. Development (TDD)
   Implementation will follow Beck's TDD mantra "red/green/refactor".

4. Deploy
   Deployment to our main production-ready GitHub branch and live cloud host (if applicable)

5. Review
   The development team will hold a sprint review meeting to discuss what went well as well as any challenges or suggestions for future sprints.

iii. Version Control

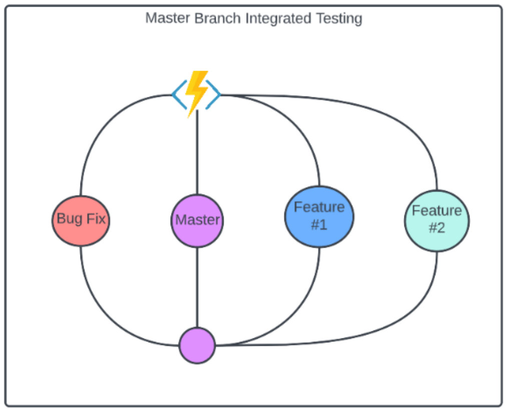
System source code will be versioned using the Git versioning system to trace and monitor changes, with the repository being housed on GitHub due to its widely accepted usage, integration, and community support & documentation.

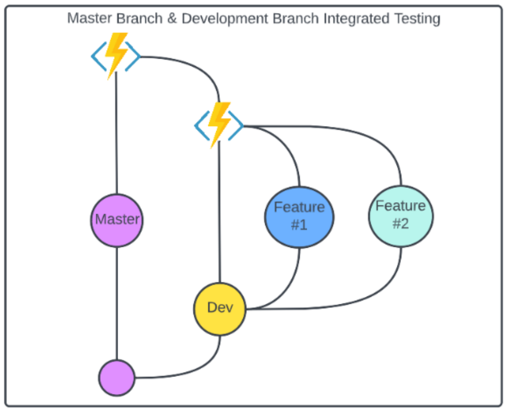
All collaborative changes made to the source code must be requested via GitHub pull requests, which will be reviewed and considered by the team. A **Pull Request Template** will be provided to ensure sufficient information is provided to the reviewer.

Similarly, issues can be created to bring a bug or requested feature to the team's awareness. Templates will also be provided for both **Bug Reports** and **Feature Requests** to help ensure adequate information is provided.

To help maintain the high standard of code and documentation, the repository will also feature a set of **Contributor Guidelines** which will outline expectations and answers to commonly/frequently asked questions. One such guideline will be to use the "feature branch" approach as shown below in figure 1, specifying that each proposed feature, for example *"user authentication"*, would be in a branch of its own and have pull request/s relating to just that feature.

In addition to the feature branch approach shown in figure 1, this project includes an additional failsafe in the form of a **development branch** (or *staging* branch) which can be visualised in the image below (right). All pull requests and/or issues will be merged with the development branch which is also equipped with automated testing via GitHub Actions. Once tests pass on the development branch and the team is satisfied with the changes made, the development branch can be merged into the master branch and subsequently become a "release". The master branch will also run automated build tests, both as a sanity check and to feed a status badge in the master branch readme file, as this branch [master] is the source that will, in future releases, feed a continuous integration & delivery pipeline.


Master Branch Integrated Testing
Bug Fix
Master
Feature #1
Feature #2


Master Branch & Development Branch Integrated Testing
Master
Feature #1
Feature #2
Dev

References

- Beck, K 2002, *Test-driven Development: By Example*, E-book, Addison Wesley, available at https://books.google.com.au/books?hl=en&lr=&id=CUlsAQAAQBAJ&oi=fnd&pg=PR7&dq=test+driven+development&ots=QChUZd0KLS&sig=gpNa2QBwaNeSO6enW2k4l0kI3xU#v=onepage&q=test%20driven%20development&f=false
- Wikipedia 2022, Test-driven development, viewed 10 August 2022, https://en.wikipedia.org/wiki/Test-driven_development