South East Technological University

Department of Science

Bachelor of Science (Honors) in Computer Forensics and Security

Bachelor of Science (Honors) in Computer Science

(Computer Forensics and Security)

**Software Engineering Practice**

**Final Documentation**

Under-Graduate Students:

Seán Murphy - 20101936

Alan Zahria - 20102420

Joshua Webster

Stephen McGrath - 20103121

Table Of Contents

[**Product Vision** 2](#_Toc164856385)

[**Initial Feature List** 2](#_Toc164856386)

[**User Personas** 3](#_Toc164856387)

[**User Scenarios** 3](#_Toc164856388)

[**User Stories** 5](#_Toc164856389)

[**Revised Feature List** 6](#_Toc164856390)

[Our Applications - Initial Features List 6](#_Toc164856391)

[Driver page Guest, Student, and Faculty sign-in 6](#_Toc164856392)

[Student Services 6](#_Toc164856393)

[Campus Map 7](#_Toc164856394)

[Extracurricular 7](#_Toc164856395)

[Work Experience 7](#_Toc164856396)

[Exam/Transcripts 7](#_Toc164856397)

[Counselling 7](#_Toc164856398)

[Documentation 7](#_Toc164856399)

[**Prototype** 8](#_Toc164856400)

[**Junit Tests** 8](#_Toc164856401)

[**Git Hub and Trello Link** 8](#_Toc164856402)

# **Product Vision**

**App 1**

**Moodle**

**- Pros**

\* Able to view all modules that the student is enrolled in.

\* Download and upload content required either for study or submitting assignments.

\* Calendar with submission dates/time and class attendance.

\* Notifications linked with emails.

\* Editor for how each user wants their home page to be set up.

\* Ability for lecturers to add students to groups for classes or otherwise.

\* Members of groups can be given different permissions.

**- Cons**

\* Calendar only has submission and attendance markers does not contain a timetable for classes and

doesn’t contain national holidays.

\* Only online on a webpage no functional application.

\* No functionality for campus maps, timetables, etc.

\* Messages sent through the platform take a while to be received.

**What can we take from it to better our application**

The main application is solid but there are some areas which can be improved. The calendar, group functionality, notification system and uploading and downloading facilities provide a solid foundation for any application of this type.

**FOR, WHO:**

Moodle is a web application for students and lecturers to manage class content and assignment submission.

**Product category and Product Name**

Moodle is a digital learning and information tool.

**THAT:**

Available online free, employed by universities.

**UNLIKE:**

Unlike Moodle, applications like Canvas are known for their strong mobile abilities allowing users to access course materials on a variety of different devices.

**OUR PRODUCT:**

An app which creates a more suitable and comfortable user experience for students and lecturers to communicate, designate and complete work.

---------------------------------------------------------------------------------------------------------------------------

**App 2**

**Slack**

**- Pros**

- Commonly used by lecturers and is a great way to communicate with them, More easier to use than emails and the ability to discuss with the full class.

**- Cons**

- Unfriendly user interface, web appilcation is pushed over to the mobile appliactions (No design for mobile users) and is difficult to set up as in it does not save information across(Setup up again on pc and mobile)

**- What can we take from it to better our own application**

- With our analysis of slack, we found that it can benefit from both an overhaul and dedicated apps for both PCs and mobile devices.

**FOR, WHO:**

Slack is an app for students and lecturers to communicate online,

**Product category and Product Name**

Slack is a digital communication tool.

**THAT:**

Available on mobile devices, free, employed by universities.

**UNLIKE:**

For communication, applications like discord could be used better to communicate with groups and individuals.

Other apps for employment and universities such as Moodle or workday tends to more commonly used by students and employees.

**OUR PRODUCT:**

An app which creates a more suitable and comfortable user experience for students and lecturers to communicate, designate and complete work.

--------------------------------------------------------------------------------------------------------------------------

1) FOR - Students and Faculty of SETU

2) WHO - No such application for the College

3) THE (product name) -

- Name Ideas

\* SetuScholar

\* Setu(Campus)Companion

\* **SetuCompass** **--Final name--**

\* SetuPortal

\* SetuConnect

4) IS A – web-based education platform

5) THAT – Everything students and lecturers need at your finger

6) UNLIKE - Moodle and Slack

7) OUR PRODUCT - Combination of those mentioned above with extra features

**Our Application**

1) Driver page Guest, Student, and Faculty sign-in

\* Guest to view newsletters and courses - possible future students

2) Timetable

\* Students and Lectures have their accounts and can access their own timetables

3) Courses

\* Catalog of available courses (Guest and Student)

\* Assignment submissions (Students and Lectures)

4) Student Services

\* Student Union

\* Campus Map

\* View of all college campuses (Waterford, Carlow, and Wexford)

\* Clubs & Societies

5) Alumni

\* Events and Networking

6) Career Services

\* Job postings

\* Internships

# **Initial Feature List**

**Driver page Guest, Student, and Faculty sign-in**

Guest to view newsletters and courses - possible future students Timetable

Students and Lectures have their accounts and can access their own timetables Courses

Profile :

- Name

- Course

- Student No. / Employee No.

- Contact Info

Catalog of available courses (Guest and Student)

Assignment submissions (Students and Lectures)

**Student Services**

Student Union

Student and College organised events - Eg. Computing Week and student organised balls

Timetable for classes

Calendar

- Public holidays

- Assingment Deadlines

Ability to create group chat for student

Computing & Maths Centre

- Booking

Anonymous report system and direct message to students class representitiive

**Campus Map**

- View of all college campuses

- Waterford

- Carlow

- Wexford

**Extracurricular**

- Clubs

- Volunteer Work

- Events

**Work Experience**

- Career Services

- Internships

- Part-time jobs

**Exam/Transcripts**

Viewing of all taken exams

**Counselling**

(Non) Anonymous chat with a Counsellor / Psychotherapist

Booking time with them help with college life

# **User Personas**

A user persona is a fictional character that fits the description of an intended end user made for the purpose of testing a system through methods such as user scenarios and stories. These people are members of the product’s target market and can help designers in creating a system that will suit their actual target audience’s needs.

A screenshot of a personal background

Description automatically generatedA screenshot of a purple and white website

Description automatically generatedA screenshot of a computer

Description automatically generatedA person smiling at the camera

Description automatically generated

# **User Scenarios**

A user scenario describes a specific situation a user may find themselves in, and how they would interact with the system to achieve the goal they want. Examples include navigating to a specific help

section of an application or logging in with a third-party account. They can be used to evaluate the usability of a system, determining if it is efficient and testing its consistency with what the user expects.

Rebecca wants to access her timetable through the central information area of the application.

She first logs into the service using her username and password.

Once she is logged in, she navigates to the information area from the menu bar at the top of the screen.

The system uses Rebecca's information and the groups she's a member of to load all of the relevant sections she can access.

She clicks the tab for timetable information.

The system loads a new interface, where it displays her lectures, practical’s, tutorials and breaks, any changes to her timetable since the previous week, any notices for cancelled classes, any tests or exams during the upcoming week and any classwork or assignments that are due.

A screenshot of a cell phone

Description automatically generatedA screenshot of a social media post

Description automatically generatedA person's profile on a purple and pink card

Description automatically generated

# **User Stories**

User stories are written from the perspective of a potential user of the system and are written in informal language. For example, suitable user stories for our application come from students and lecturers. They describe the user’s wants, needs and expectations, giving a development team greater insight into their system and potentially showing cracks in it such as missing features or poor accessibility.

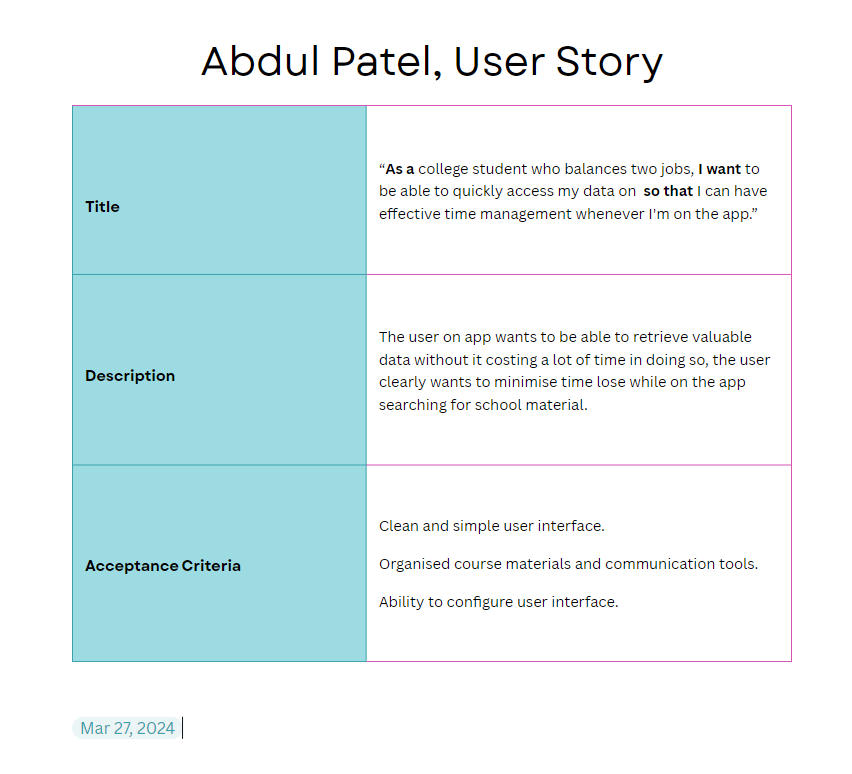
Seán Murphy – User Story

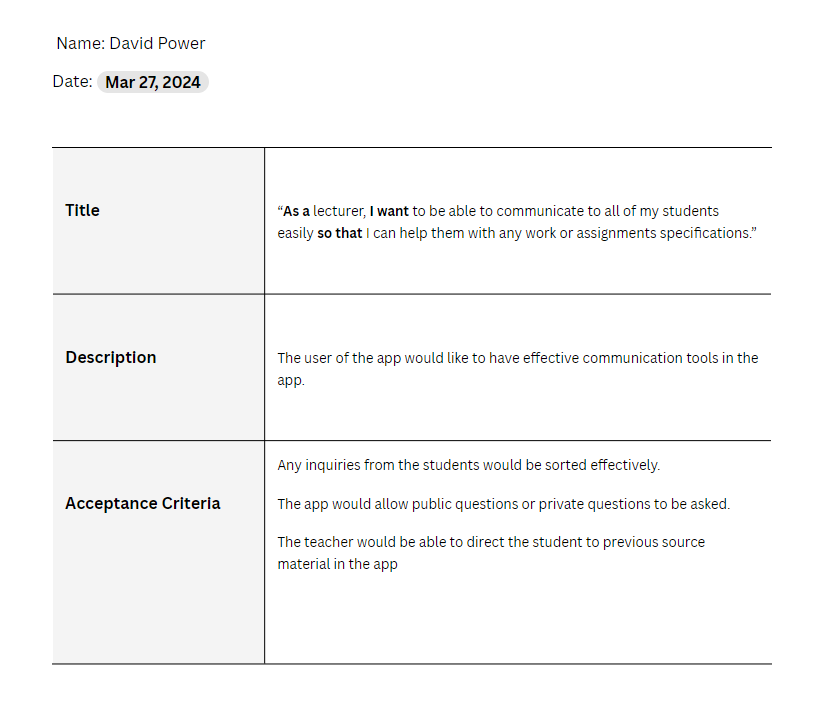
A person smiling for a picture

Description automatically generated



Alan Zaharia – User Story





Joshua Webster – User Story





Stephen McGrath – User Story

As a student, I want to be able to see all information for my timetables, exam information, notices from the college or lecturers and more in one central location. Other services I have used have had multiple redirects and unclear interfaces, causing confusion and incovenience for me, which I would prefer not to have to deal with during my time at college.

As a student, I want to be able to easily instantly message lecturers and classmates. I want to be able to set up group messaging with classmates to organise assignments and group work. It can be difficult to find contact information and get in contact with lecturers quickly, as platforms like Moodle impose a cooldown before a message is actually sent. Having easy instant messaging would eliminate these problems.

# **Revised Feature List**

**Driver page Guest, Student, and Faculty sign-in**

Guest to view newsletters and courses - possible future students Timetable

Students and Lectures have their accounts and can access their own timetables Courses

Profile :

- Name

- Course

- Student No. / Employee No.

- Contact Info

Catalog of available courses (Guest and Student)

Assignment submissions (Students and Lectures)

**Student Services**

Student Union

Student and College organised events - Eg. Computing Week and student organised balls

Timetable for classes

Calendar

- Public holidays

- Assignment Deadlines

Ability to create group chat for student

Computing & Maths Centre

- Booking

Anonymous report system and direct message to student’s class representative

**Campus Map**

View of all college campuses

- Waterford

- Carlow

- Wexford

**Extracurricular**

Clubs and Societies

**Work Experience**

Career Services

- Internships

- Part-time jobs

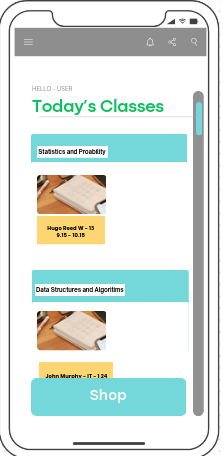
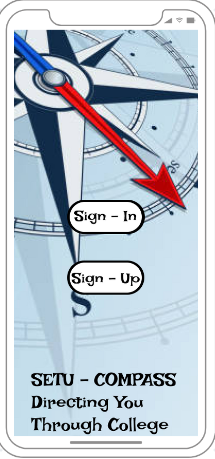
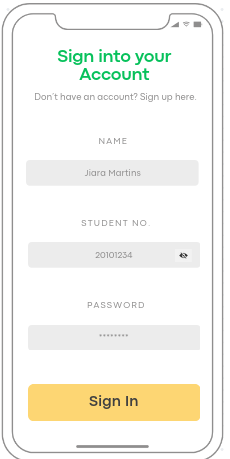
**Exam/Transcripts**

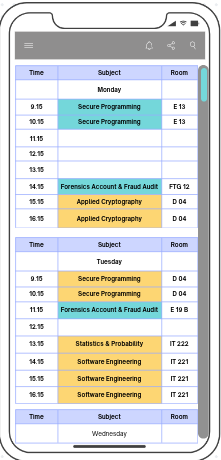
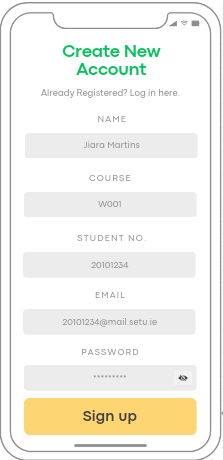
Viewing of all taken exams

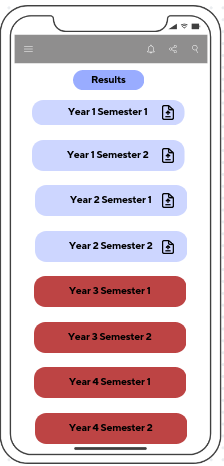
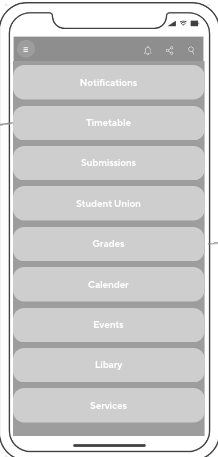
# **Prototype**

Prototyping is an important part of the development of an application. It is an early step in the development cycle, used to paint a rough idea of what the final product may look like once completed. It may change dramatically during the production cycle, but it creates a general goal for the team to work towards. Prototypes can also be shown to relevant people, such as investors or potential users, in order to get feedback so time is not wasted implementing unwanted features.

\*\*TODO\*\* Tool used to create Prototype (say if you found it easy to use or not briefly!!!)







# **Junit Tests**

**AccountTest.java**

package models;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.\*;

class AccountTest {

private Account account0, account1, account2, account3;

@org.junit.jupiter.api.BeforeEach

void setUp() {

account0 = new Account("John Doe","W104",20000001,"20000001@mail.setu.ie");

account1 = new Account("Testing names with exactly 34 char", "W13", 2000001, "20000000@mail.setu.ie");

account2 = new Account("Testing names with exactly 35 chars", "W143", 20000001, "20000000@mail.setu.ie2345");

account3 = new Account("Testing names with more than 35 chars","W1443",200002001,"20000000@mail.setu.ie2345");

}

@org.junit.jupiter.api.AfterEach

void tearDown() {

account0 = account1 = account2 = account3 = null;

}

@Test

void underConstructorTests() {

//Test under boundary's

assertEquals("Testing names with exactly 34 char", account1.getName());

//Value accepted under 35 chars

assertEquals("W13", account1.getCourse());

//Value accepted under 4 chars

assertEquals(20000000, account1.getId());

//Value 2000001 not accepted default to 20000000

assertEquals("20000000@mail.setu.ie", account1.getContactInfo());

//Value accepted under 25 chars

}

@Test

void exactConstructorTests() {

//Test exact boundary's

assertEquals("Testing names with exactly 35 chars", account2.getName());

//Value accepted max 35 chars

assertEquals("W143", account2.getCourse());

//Value accepted max 4 chars

assertEquals(20000001, account2.getId());

//Value accepted within boundary's

assertEquals("20000000@mail.setu.ie2345", account2.getContactInfo());

//Value accepted max 25 chars

}

@Test

void overConstructorTests(){

//Test over boundary's

assertEquals("Student Name Error",account3.getName());

//Value Testing names with more than 35 chars not accepted over 35 chars set to default

assertEquals("W001",account3.getCourse());

//Value W1443 not accepted over max 4 chars set to default

assertEquals(20000000,account3.getId());

//Value 200002001 not accepted set to default

assertEquals("20000000@mail.setu.ie2345",account3.getContactInfo());

//Value 20000000@mail.setu.ie2345 not accepted max 25 chars set to default

}

@Test

void gettersAndSetters(){

//Check the values

assertEquals("John Doe",account0.getName());

assertEquals("W104",account0.getCourse());

assertEquals(20000001,account0.getId());

assertEquals("20000001@mail.setu.ie",account0.getContactInfo());

//Set new values

account0.setName("Joey Blogs");

account0.setCourse("W105");

account0.setId(20000011);

account0.setContactInfo("20000011@mail.setu.ie");

//Check that it changed

assertEquals("Joey Blogs",account0.getName());

assertEquals("W105",account0.getCourse());

assertEquals(20000011,account0.getId());

assertEquals("20000011@mail.setu.ie",account0.getContactInfo());

}

@Test

void validateEquals(){

Account otherAccount = new Account("John Doe","W104",20000001,"20000001@mail.setu.ie");

assertNotSame(account0,otherAccount);

assertEquals(account0,otherAccount);

}

@Test

void toStringContainsAllFields(){

assertTrue(account0.toString().contains("John Doe"));

assertTrue(account0.toString().contains("W104"));

assertTrue(account0.toString().contains("20000001"));

assertTrue(account0.toString().contains("20000001@mail.setu.ie"));

}

}

**ClubsTest.java**

package models;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.\*;

class ClubsTest {

private Clubs clubs0, clubs1, clubs2, clubs3;

@org.junit.jupiter.api.BeforeEach

void setUp() {

clubs0 = new Clubs("Esport Club",15,"Compete for glory");

clubs1 = new Clubs("Testing names with exactly 34 char", 21,"This is a description of an event");

clubs2 = new Clubs("Testing names with exactly 35 chars", 15,"Testing text with exactly 35 char's");

clubs3 = new Clubs("Testing names with more than 35 chars",30,"Testing description's entry with more than 50 chars okiedokie");

}

@org.junit.jupiter.api.AfterEach

void tearDown() {

clubs0 = clubs1 = clubs2 = clubs3 = null;

}

@Test

void underConstructorTests() {

//Test under boundary's

assertEquals("Testing names with exactly 34 char", clubs1.getName());

//Value accepted under 35 chars

assertEquals(21, clubs1.getCost());

//Value accepted under 8 chars

assertEquals("This is a description of an event", clubs1.getDesc());

//Value accepted under 35 chars

}

@Test

void exactConstructorTests() {

//Test exact boundary's

assertEquals("Testing names with exactly 35 chars", clubs2.getName());

//Value accepted max 35 chars

assertEquals(15, clubs2.getCost());

//Value accepted max 4 chars

assertEquals("Testing text with exactly 35 char's", clubs2.getDesc());

//Value accepted max 25 chars

}

@Test

void overConstructorTests(){

//Test over boundary's

assertEquals("Club Name Error", clubs3.getName());

//Value Testing names with more than 35 chars not accepted over 35 chars set to default

assertEquals(10, clubs3.getCost());

//Value W1443 not accepted over max 4 chars set to default

assertEquals("Clubs Description", clubs3.getDesc());

//Value 20000000@mail.setu.ie2345 not accepted max 25 chars set to default

}

@Test

void gettersAndSetters(){

//Check the values

assertEquals("Esport Club", clubs0.getName());

assertEquals(15, clubs0.getCost());

assertEquals("Compete for glory", clubs0.getDesc());

//Set new values

clubs0.setName("Go Karting");

clubs0.setCost(20);

clubs0.setDesc("Changed Desc");

//Check that it changed

assertEquals("Go Karting", clubs0.getName());

assertEquals(20, clubs0.getCost());

assertEquals("Changed Desc", clubs0.getDesc());

}

@Test

void validateEquals(){

Clubs otherClubs = new Clubs("Esport Club",15,"Compete for glory");

assertNotSame(clubs0,otherClubs);

assertEquals(clubs0,otherClubs);

}

@Test

void toStringContainsAllFields(){

assertTrue(clubs0.toString().contains("Esport Club"));

assertTrue(clubs0.toString().contains("15"));

assertTrue(clubs0.toString().contains("Compete for glory"));

}

}

**EventTest.java**

package models;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.\*;

class EventTest {

private Event event0, event1, event2, event3;

@org.junit.jupiter.api.BeforeEach

void setUp() {

event0 = new Event("Computing Ball","14/04/24","20:00","Computing Ball in the tower hotel at 8pm until 11pm");

event1 = new Event("Testing names with exactly 34 char", "21/03", "21", "This is a description of an event");

event2 = new Event("Testing names with exactly 35 chars", "15/02/24", "20:30", "Testing description's entry with exactly 50 chars okay");

event3 = new Event("Testing names with more than 35 chars","20/20/2004","19:30:55","Testing description's entry with more than 50 chars okiedokie");

}

@org.junit.jupiter.api.AfterEach

void tearDown() {

event0 = event1 = event2 = event3 = null;

}

@Test

void underConstructorTests() {

//Test under boundary's

assertEquals("Testing names with exactly 34 char", event1.getName());

//Value accepted under 35 chars

assertEquals("21/03", event1.getDate());

//Value accepted under 8 chars

assertEquals("21", event1.getTime());

//Value accepted under 5 chars

assertEquals("This is a description of an event", event1.getDesc());

//Value accepted under 25 chars

}

@Test

void exactConstructorTests() {

//Test exact boundary's

assertEquals("Testing names with exactly 35 chars", event2.getName());

//Value accepted max 35 chars

assertEquals("15/02/24", event2.getDate());

//Value accepted max 4 chars

assertEquals("20:30", event2.getTime());

//Value accepted within boundary's

assertEquals("Testing description's entry with exactly 50 chars okay", event2.getDesc());

//Value accepted max 25 chars

}

@Test

void overConstructorTests(){

//Test over boundary's

assertEquals("Event Name Error",event3.getName());

//Value Testing names with more than 35 chars not accepted over 35 chars set to default

assertEquals("00/00/00",event3.getDate());

//Value W1443 not accepted over max 4 chars set to default

assertEquals("00.00",event3.getTime());

//Value 200002001 not accepted set to default

assertEquals("Event Description",event3.getDesc());

//Value 20000000@mail.setu.ie2345 not accepted max 25 chars set to default

}

@Test

void gettersAndSetters(){

//Check the values

assertEquals("Computing Ball",event0.getName());

assertEquals("14/04/24",event0.getDate());

assertEquals("20:00",event0.getTime());

assertEquals("Computing Ball in the tower hotel at 8pm until 11pm",event0.getDesc());

//Set new values

event0.setName("Christmas Party");

event0.setDate("21/12/24");

event0.setTime("21:00");

event0.setDesc("Changed Desc");

//Check that it changed

assertEquals("Christmas Party",event0.getName());

assertEquals("21/12/24",event0.getDate());

assertEquals("21:00",event0.getTime());

assertEquals("Changed Desc",event0.getDesc());

}

@Test

void validateEquals(){

Event otherEvent = new Event("Computing Ball","14/04/24","20:00","Computing Ball in the tower hotel at 8pm until 11pm");

assertNotSame(event0,otherEvent);

assertEquals(event0,otherEvent);

}

@Test

void toStringContainsAllFields(){

assertTrue(event0.toString().contains("Computing Ball"));

assertTrue(event0.toString().contains("14/04/24"));

assertTrue(event0.toString().contains("20:00"));

assertTrue(event0.toString().contains("Computing Ball in the tower hotel at 8pm until 11pm"));

}

}

# **Git Hub and Trello Link**

GitHub

<https://github.com/SeanMurphy1479/Software_Engineering_CA>

Trello

<https://trello.com/invite/userworkspace39934289/ATTIf254ff9966b2c0b6e65d2b79e5e91e1e7B0BFB03>