Academic Misconduct Management System

Introduction to Software Engineering

Group Coursework Team 49

Shae McFadden (k20072607), Ahmed Banko (k20071320), Inigo Cojuangco (k19007748), Zakariya Ahmed Mohamed (k20008985), Suhayb Yones (k20044202), Shermiaya Raymond (k20046722)

Table of Contents

Team Management

Meeting 1 – in notes

- Sunday 21 February 2021 16:00 (GMT)
- All members attended
- Coursework part two requirements were reviewed. Ahmed and Shermiaya were assigned to create the State Machine Diagram. Inigo and Zakariya were assigned to do the refactored class diagram. Shae and Suhayb refactored use cases.

Meeting 2

- Sunday 7 March 2021 16:00 (GMT)
- All team members attended
- Work assigned from the previous meeting was reviewed. Inigo and Ahmed were assigned to clean up their diagrams from the previous week. The rest of the team were assigned Sequence Diagrams. Shae (Submit Summary and Case Registered) Suhayb (Send Evidence and Declare Case Type), Shermiaya (Create & Complete Form and Schedule) and Zakariya (Report Case and Review Reported Case)

Meeting 3

- Sunday 14 March 2021 16:00 (GMT)
- All team members attended
- Work from the previous meeting was reviewed. State Machine Diagram, as well as Submit Summary and Case Registered Sequence diagrams were completed the rest of the diagrams required more time, so team agreed to give them another week. Inigo also assigned to write refactored class diagram description

Meeting 4

- Sunday 28 March 2021 16:00 (GMT)
- All members attended
- Diagrams were reviewed and approved. Ahmed and Shae were assigned to compile the report while Inigo, Zakariya, Shermiaya and Suhayb work on testing plan.

Meeting 5

- Sunday 2 April 2021 16:00 (GMT)
- All members attended
- Testing plan was reviewed and added to the report completing the report.

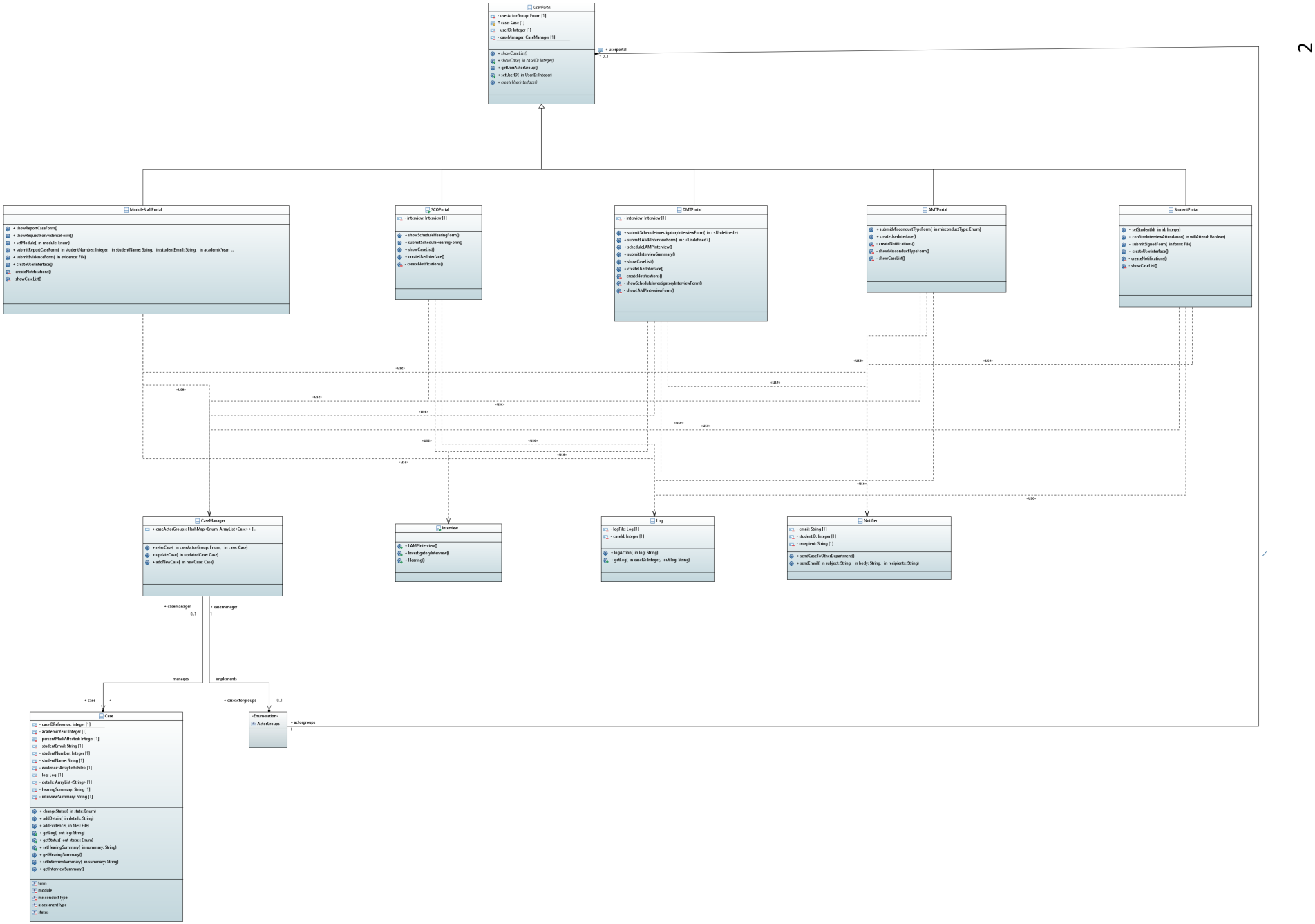
Refactored Class Diagram

Design pattern:

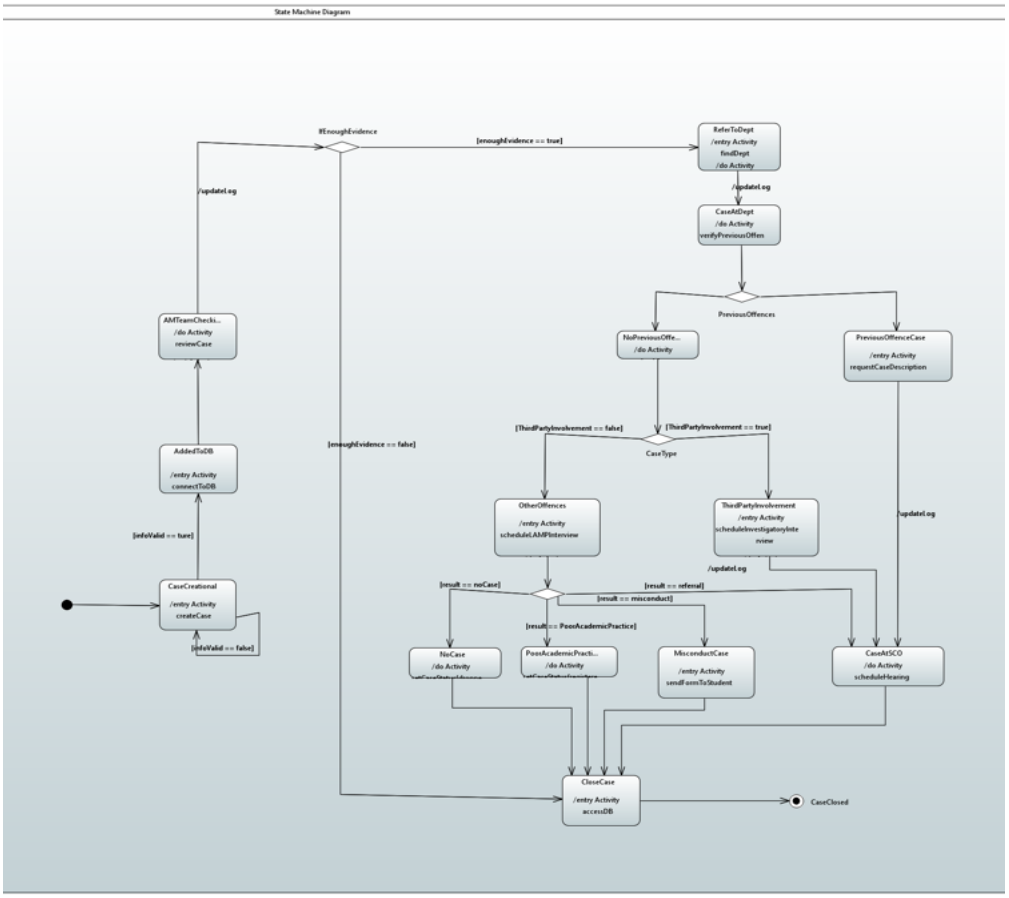
Abstract Factory

Explanation:

We decided to use an abstract factory, because it is designed around creating different things (such as in a factory line), via different factories. The entire system is designed around the creation and modification of a case (through the case manager), log, interview or notification. From the initialisation of the system till the end, every class relates to the class **CaseManager**, which handles the creation, modification and viewing of any and all cases, **Interview**, which represents a scheduled interview between two users, **Log**, representing the logged information, or **Notifier**, which notifies any party as necessary. The main factory is the **UserPortal**, which represents the abstract generalised factory, from which all other portals inherit. From the aforementioned UserPortal, five portal inherit, which are the **ModuleStaffPortal**, **SCOPortal**, **DMTPortal**, **AMTPortal**, and **StudentPortal**, all of which *manufacture* one of the previously mentioned *products*.


UserPortal
- userActorGroup: Enum [1]
case: Case [1]
- userID: Integer [1]
- caseManager: CaseManager [1]
+ showCaseList()
+ showCase(in caseID: Integer)
+ getUserActorGroup()
+ setUser(ID(in UserID: Integer)
+ createUserInterface()
+ userportal
0..1
ModuleStaffPortal
+ showReportCaseForm()
+ showRequestForEvidenceForm()
+ setModule(in module: Enum)
+ submitReportCaseForm(in studentNumber: Integer, in studentName: String, in studentEmail: String, in academicYear: ...
+ submitEvidenceForm(in evidence: File)
+ createUserInterface()
- createNotifications()
- showCaseList()
SCOPortal
- interview: Interview [1]
+ showScheduleHearingForm()
+ submitScheduleHearingForm()
+ showCaseList()
+ createUserInterface()
- createNotifications()
DMITPortal
- interview: Interview [1]
+ submitScheduleInvestigatoryInterviewForm(in : <Undefined>)
+ submitLAMPInterviewForm(in : <Undefined>)
+ schedule(LAMPInterview()
+ submitInterviewSummary()
+ showCaseList()
+ createUserInterface()
- createNotifications()
- showScheduleInvestigatoryInterviewForm()
- showLAMPInterviewForm()
AMITPortal
+ submitMisconductTypeForm(in misconductType: Enum)
+ createUserInterface()
- createNotifications()
- showMisconductTypeForm()
- showCaseList()
StudentPortal
+ setStudentId(in id: Integer)
+ confirmInterviewAttendance(in willAttend: Boolean)
+ submitSignedForm(in form: File)
+ createUserInterface()
- createNotifications()
- showCaseList()
«use»
CaseManager
+ caseActorGroups: HashMap <Enum, ArrayList<Case>> |...
+ referCase(in caseActorGroup: Enum, in case: Case)
+ updateCase(in updatedCase: Case)
+ addNewCase(in newCase: Case)
Interview
+ LAMPInterview()
+ InvestigatoryInterview()
+ Hearing()
Log
- logFile: Log [1]
- caseId: Integer [1]
+ logAction(in log: String)
+ getLog(in caseID: Integer, out log: String)
Notifier
- email: String [1]
- studentID: Integer [1]
- recipient: String [1]
+ sendCaseToOtherDepartment()
+ sendEmail(in subject: String, in body: String, in recipients: String)
+ casemanager
0..1
+ casemanager
1
manages
implements
+ case
*
+ caseactorgroups
0..1
Case
- caseDReference: Integer [1]
- academicYear: Integer [1]
- percentMarkAffected: Integer [1]
- studentEmail: String [1]
- studentNumber: Integer [1]
- studentName: String [1]
- evidence: ArrayList <File> [1]
- log: Log [1]
- details: ArrayList <String> [1]
- hearingSummary: String [1]
- interviewSummary: String [1]
+ changeStatus(in state: Enum)
+ addDetails(in details: String)
+ addEvidence(in files: File)
+ getLog(out log: String)
+ getStatus(out status: Enum)
+ setHearingSummary(in summary: String)
+ getHearingSummary()
+ setInterviewSummary(in summary: String)
+ getInterviewSummary()
term
module
misconductType
assessmentType
status
«Enumeration»
ActorGroups
+ actorgroups
1

State Machine Diagram



Refactored Use Cases

While reviewing for part two of the coursework we realized our use cases required some refactoring. Below is a list of the new use cases each of which list which of the old use cases they combine (If a use case has the same name as an old one in implicitly includes that as well).

Report Case
- log action
- Display on Relevant user Portal

Send Evidence
- log action

Review Reported Case
- log action
- Request More Evidence (optional)
  - Display on Relevant user Portal

Declare Case Type
- log action
- check student past offences

Schedule
- log action

Confirm Attendance
- log action

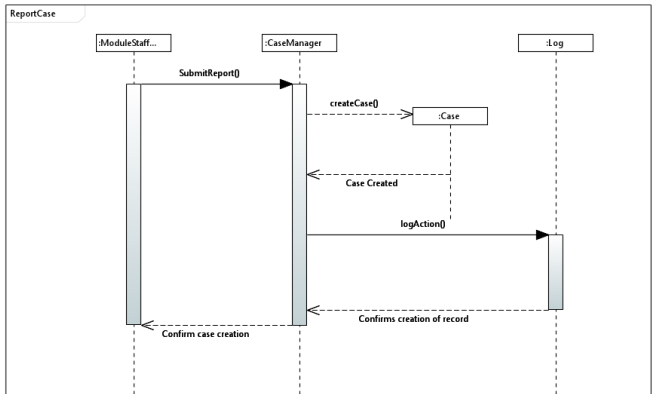
Create & Complete Form
- log action
- Send Email
- Display on Relevant user Portal

Case Registered
- log action
- case closed (optional)

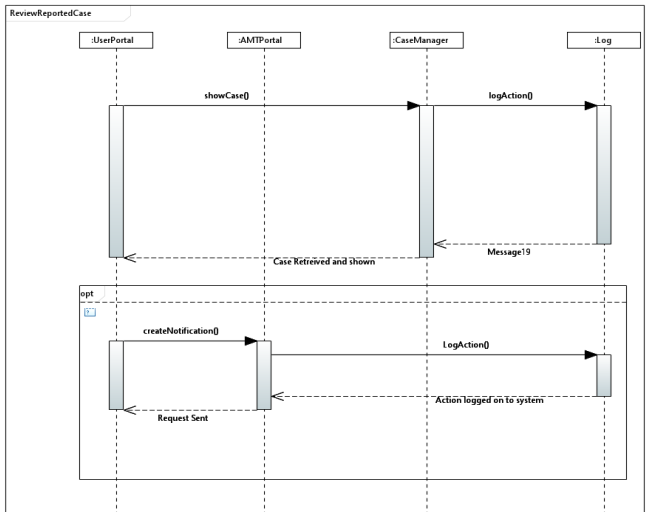
Submit Summary
- log action
- Submit Interview Summary (optional)
- Submit Hearing (optional)
  note directly tied to SCO now
- Refer Case (optional)
- Case Dropped (optional)
- Send Email (optional)
- Display on Relevant user Portal (optional)

Sequence Diagrams

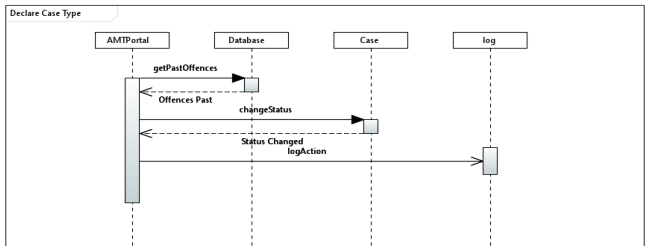
Report Case



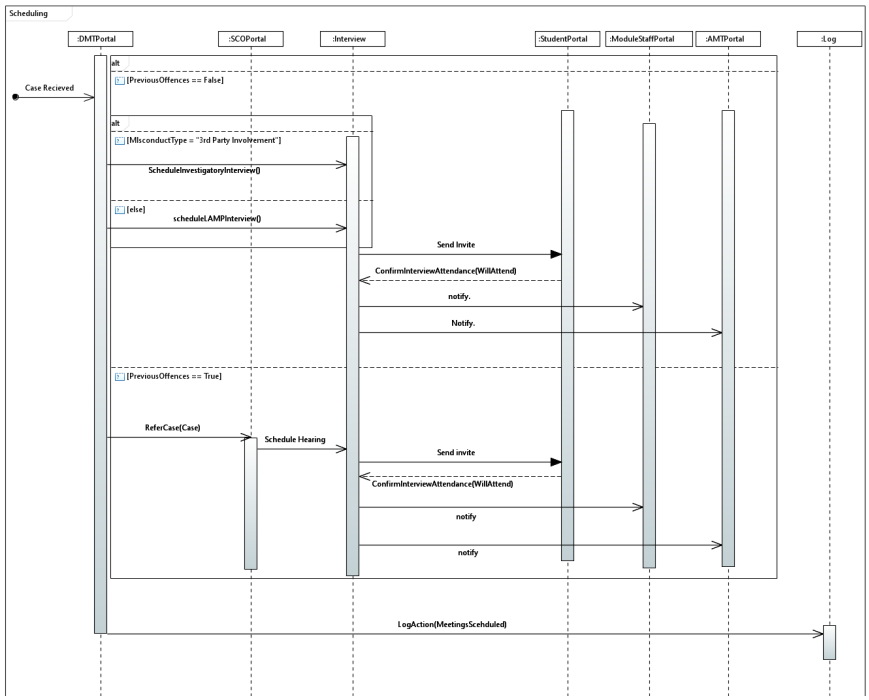
Review Reported Case



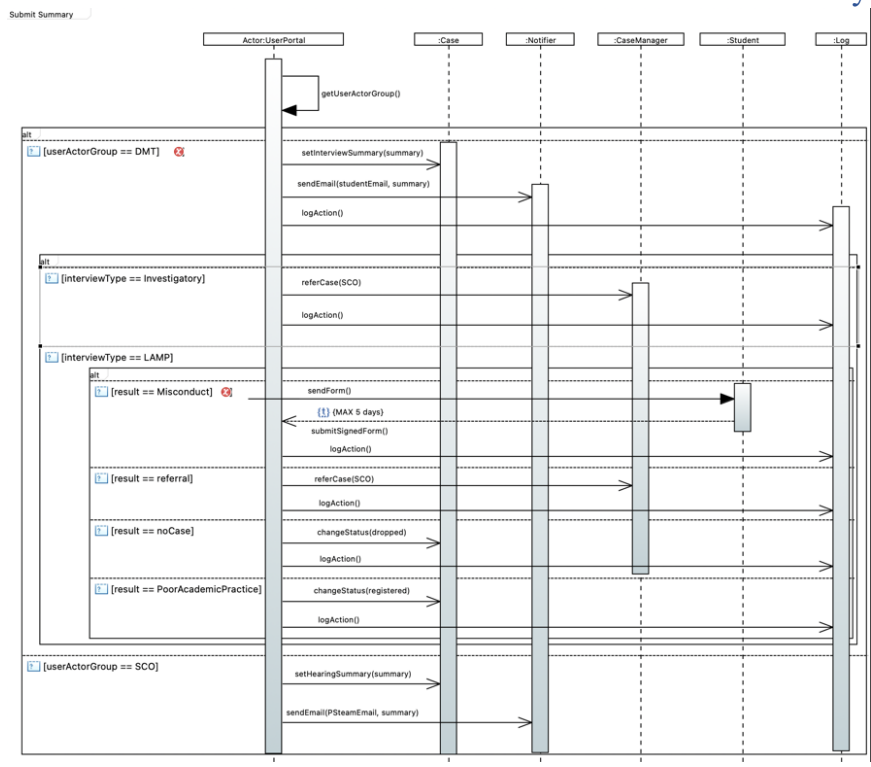
Declare Case Type



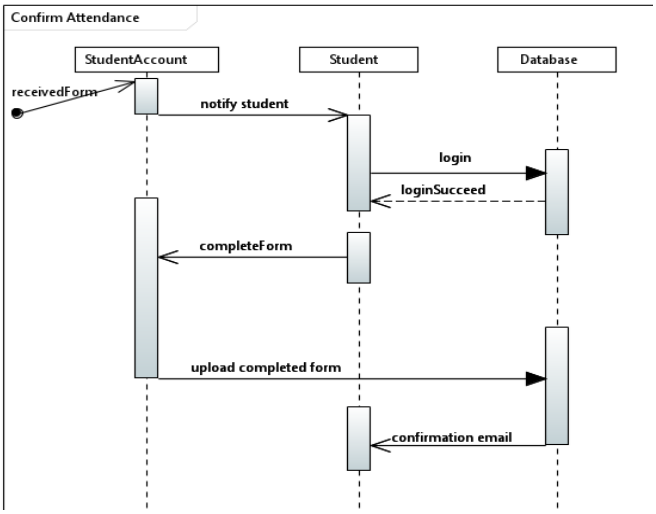
Scheduling



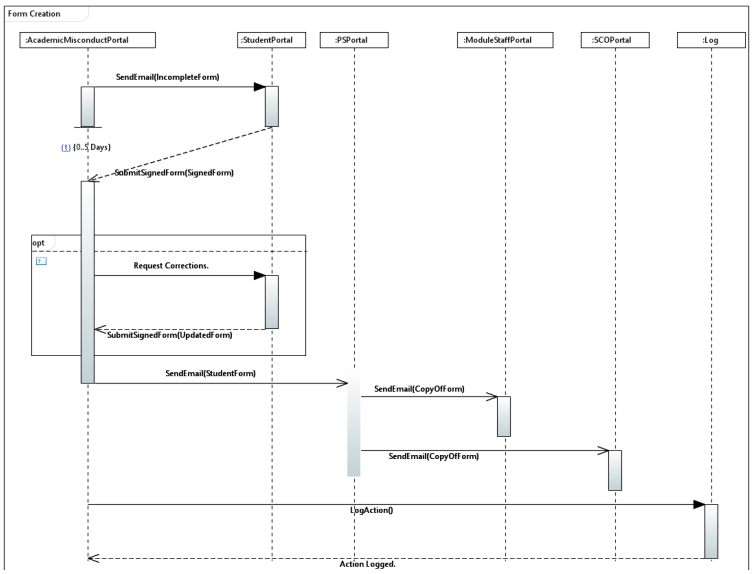
Submit Summary



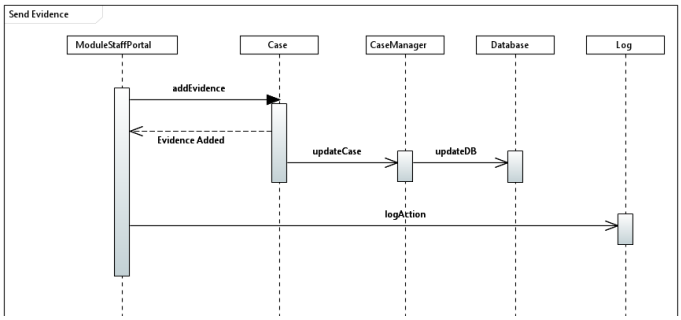
Confirm Attendance



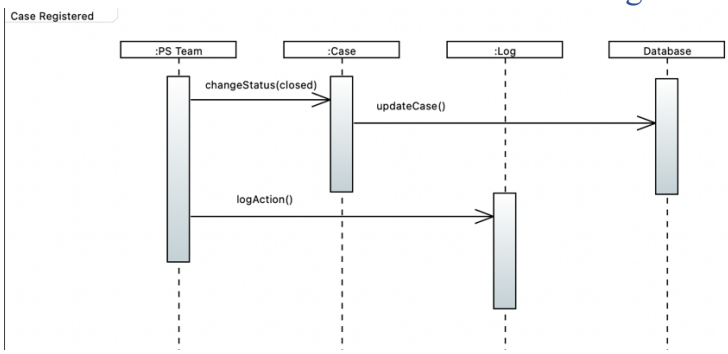
Form Creation



Send Evidence



Case Registered



Testing Plan

In order to test system functionality, we will be conducting system wide Whitebox testing. This is what allows us to test if the system is working and it is fit for purpose, with regards to the client's requirements. In addition, conduct Blackbox testing on the input fields of the program to assess that appropriate input validation has been implemented.

Scope:

In Scope:

The scope of the test will be the entire system, specifically looking to fulfill the following requirements:

- The client(s) can…
  - create a specific case or multiple cases
  - update a specific case or multiple cases
  - review a specific case or multiple cases
- The system can…
  - log all actions taken
  - send appropriate notifications (such as an email)
  - schedule relevant interview(s) and/or hearing(s)

Out Scope:

We have only been assigned the creation of the software system; therefore the following are outside of the scope of our client's requests:

- The security of the system (user authentication)
  - such as how a student cannot access anyone else's case and update any case
- The compliance of the cases and data with the GDPR 2016/679 and Data Protection Act 2018
- The stability and robustness of the system's hardware
  - such as how it must be able to be accessed by a multitude of users at asynchronous times, all at once
  - such as how it must allow usage at any time of day, by any member of the University

Quality Objective:

The quality objectives of this testing plan is to ensure that…

- our software fulfills all the requirements
- our software meets the quality needs of the client
- any bugs or issues are identified and removed before the submission of the system

Testing Methodology:

Unit testing will be conducted on the CaseManager class, using the following test plan.

What will be tested	Expected outcome(s)
The ability to add a case	A case is created and stored in the system
The ability to update a case	A previously created case is overridden by a newly created version of it, with updated information
The ability to refer a case	The previously stored case is referred to the relevant users (such as ModuleStaff or DMT)