# Planner

The greatest thing to happen to study planners since the Gantt Diagram


## Benjamin Dickson - 100140882


https://trello.com/b/BkMATIIC

# Planner

The greatest thing to happen to study planners since the Gantt Diagram


## Andrew Odintsov - 100125464


https://trello.com/b/BkMATIIC

# Planner

The greatest thing to happen to study planners since the Gantt Diagram


## Bijan Ghasemi Afshar - 100125463


https://trello.com/b/BkMATIIC

# Planner

The greatest thing to happen to study planners since the Gantt Diagram


## Zilvinas Ceikauskas - 100122076


https://trello.com/b/BkMATIIC

# Table of Contents

# Introduction

## About the Project

We have been tasked with producing a Study Planner for use by students studying at the UEA. It is a single user system, installed locally on their machine and will be updated using files sent from the Hub.

The purpose of this document is to analyse the requirements of the system, determine the use cases and come up with a class structure and system architecture ready to begin the development of the implementation.

## Who We Are

Our team consists of 4 members:

### Benjamin Dickson - Team Leader and Architecture Engineer

Ben is a mature student, who spent 15 years working as a freelance Web Developer before starting his degree at the UEA. Due to the fact he is the most experienced developer we decided he should be team leader, and took the lead on the

### Andrew Odintsov - GitHub Manager

Andrew's knowledge of using GitHub, as well as his talent for quickly learning and implementing new technologies made him the ideal person for taking on this role, which included implementing Travis.

### Bijan Ghasemi Afshar - Testing Co-ordinator

Bijan's attention to detail, and persistence meant that he was able to champion writing and rolling out our tests in JUnit.

### Zilvinas Ceikauskas - Chief Architecture and UI Designer

As the most experienced Java programmer on the team, Zil took on the responsibility of overseeing majority of internal architecture,

# System Analysis & Use Cases

# Similar Systems

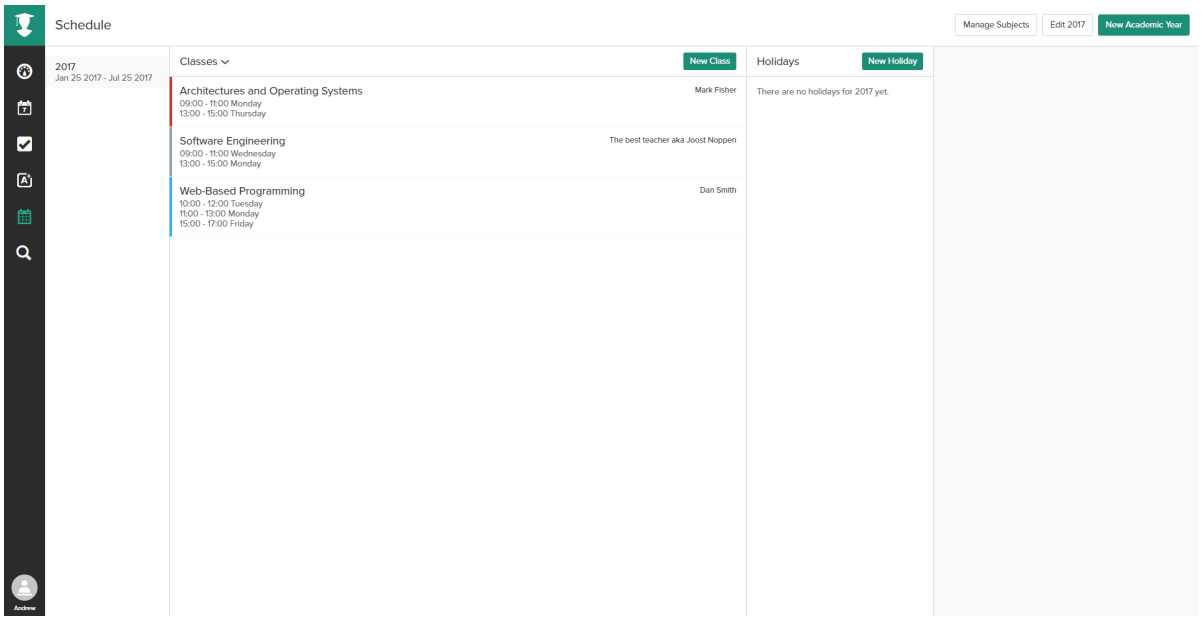
At this stage of the design phase we undertake a similar system analysis which can assists us in our design decisions by analysing the features of similar systems with the same purpose. We achieve this by analysing four similar systems and listing the similar and different features that they offer, and at the end make a decision on our system design based on the performed analysis.

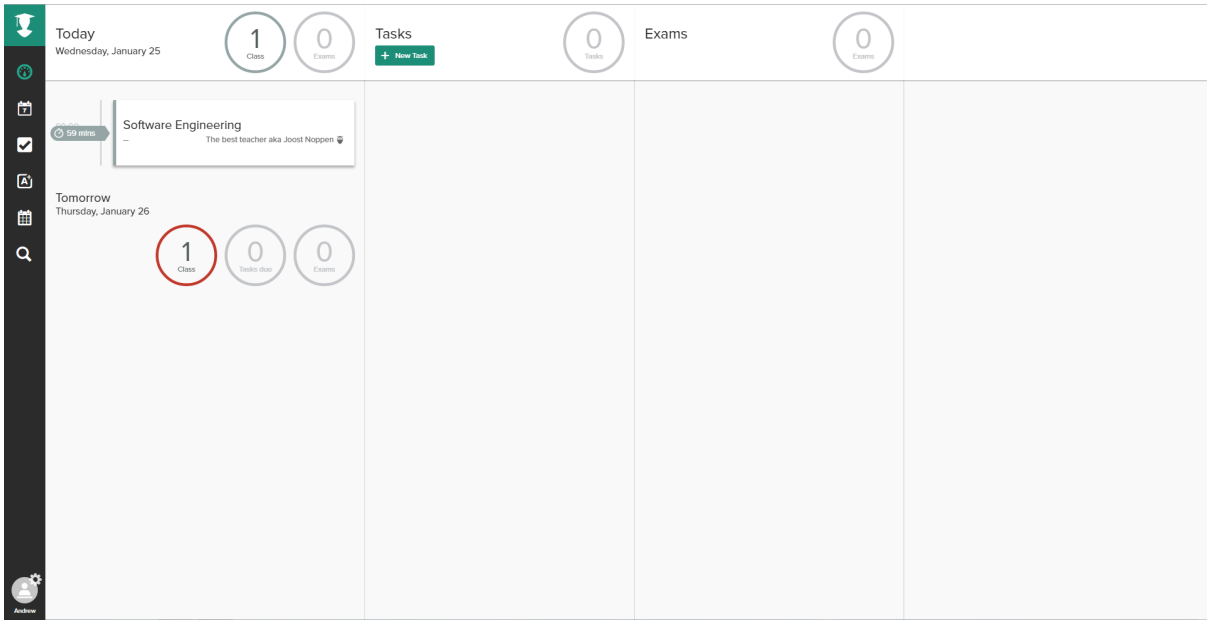
## Mystudylife.com

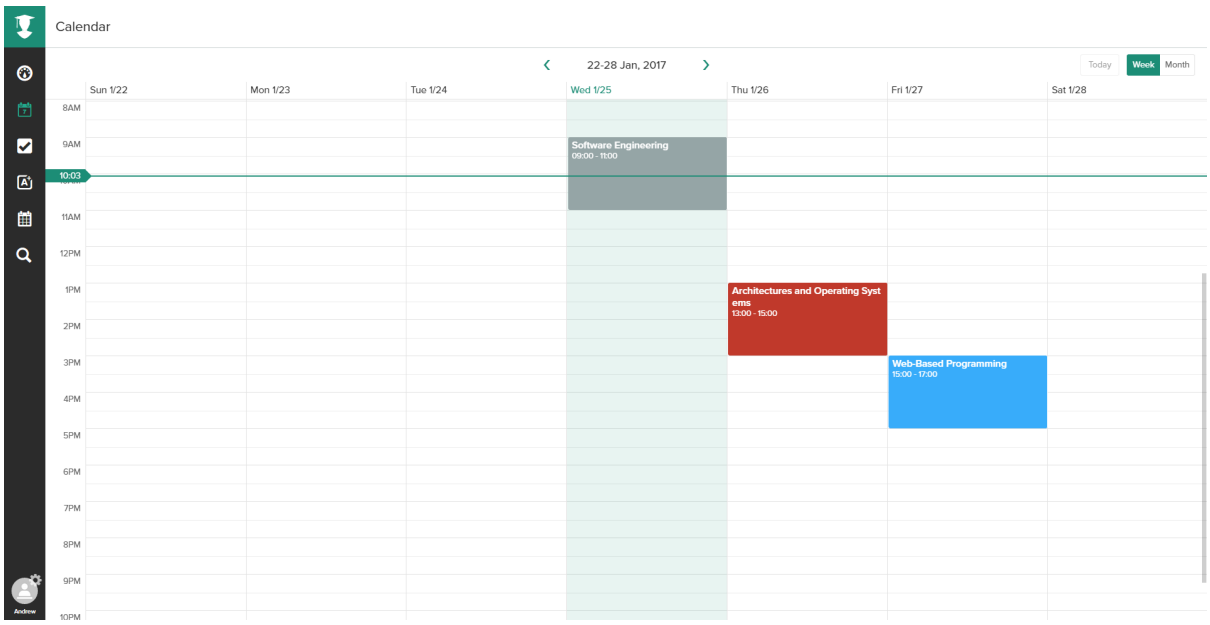
Mystudylisfe.com is a web application to organise timetables for lectures, assignments and exams. Displays all the information on the dashboard. Also has a calendar view of events.

### Main features

- Basic scheduling
- Tasks checklist
- Ability to have different lecture scheduling (odd/even weeks lectures)
- Mobile App






## Differences

- Does not have any link with the outside world (schedule import/sharing)
- Does not have task dependencies
- Activity visualisation implemented using a calendar, but not a Gantt chart

## Verdict

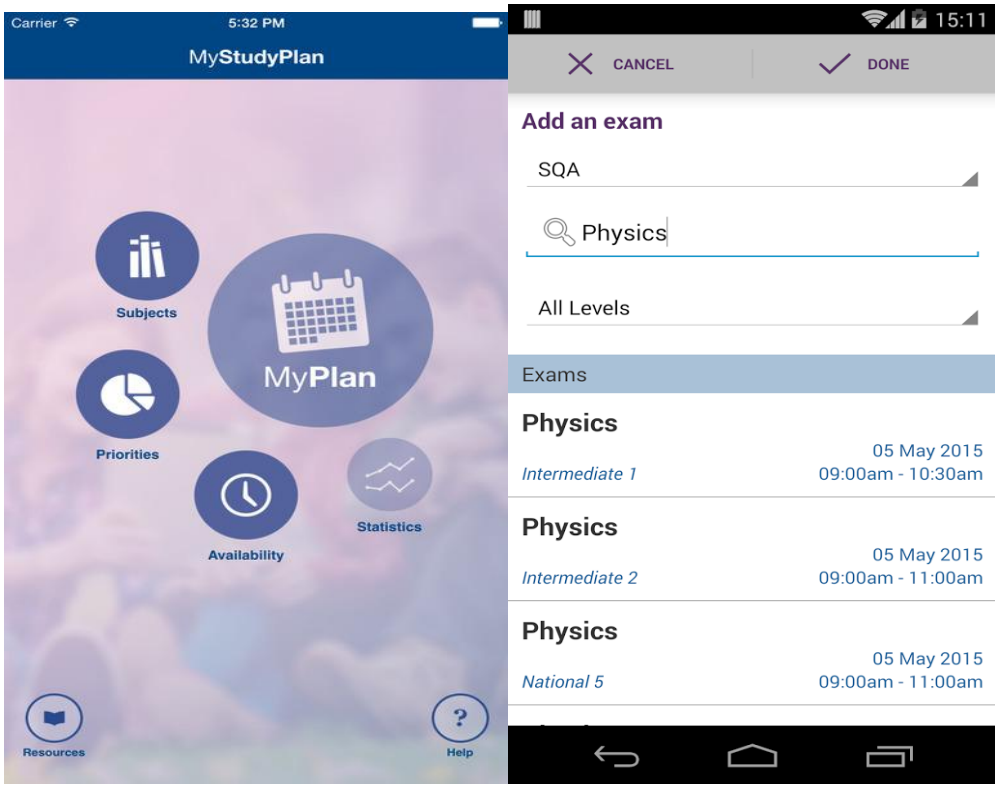
It is very simple, but a functional enough system. It doesn't have any import/export functionality. Some design ideas such as dashboard layout could be used in our future system.

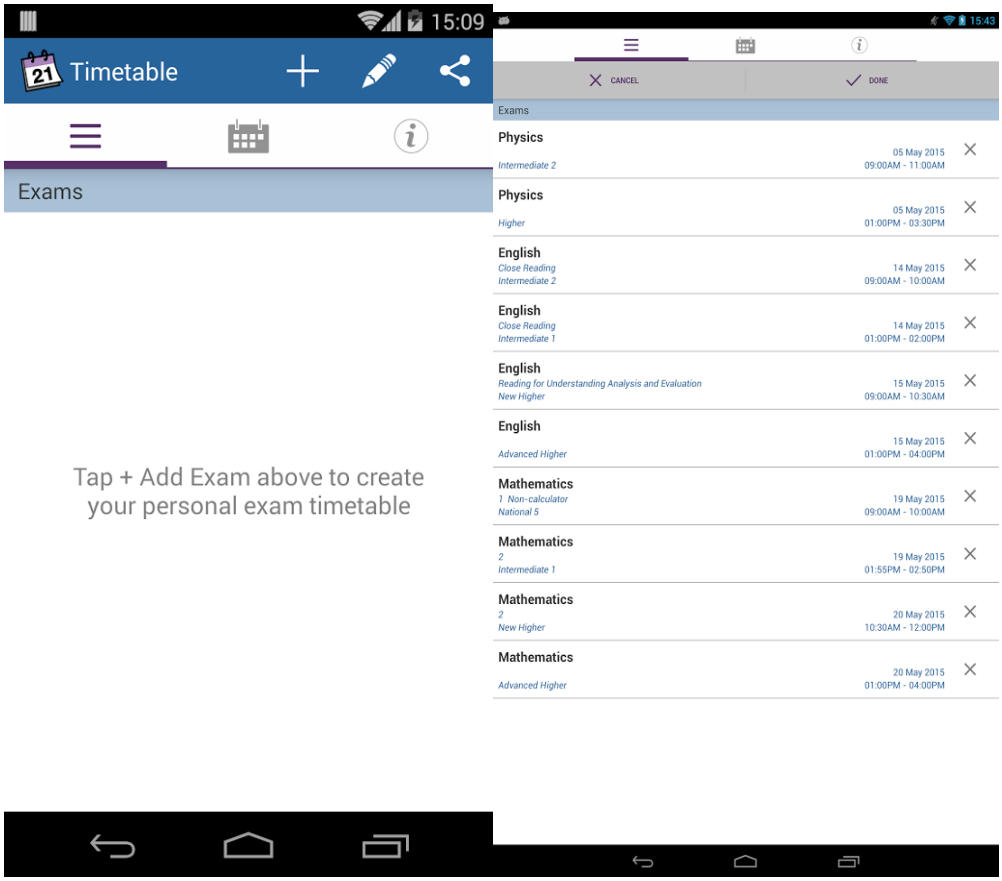
# SQA My Study Plan

Mobile app that allows students to create a study plan based on their exam dates, their importance and when they want to study. Developed by Scottish Qualifications Authority (SQA)

**Similar features:**

- Adding subjects and exams.
- Setting your preferred study period for each subject
- Editing your plan.
- Prioritisation of subjects
- Ability to mark off and monitor completed study periods
- Importing exams






**Different features:**

- Automatic generation of study plan based on your preferences
- Share your plan
- Print your plan

## Verdict

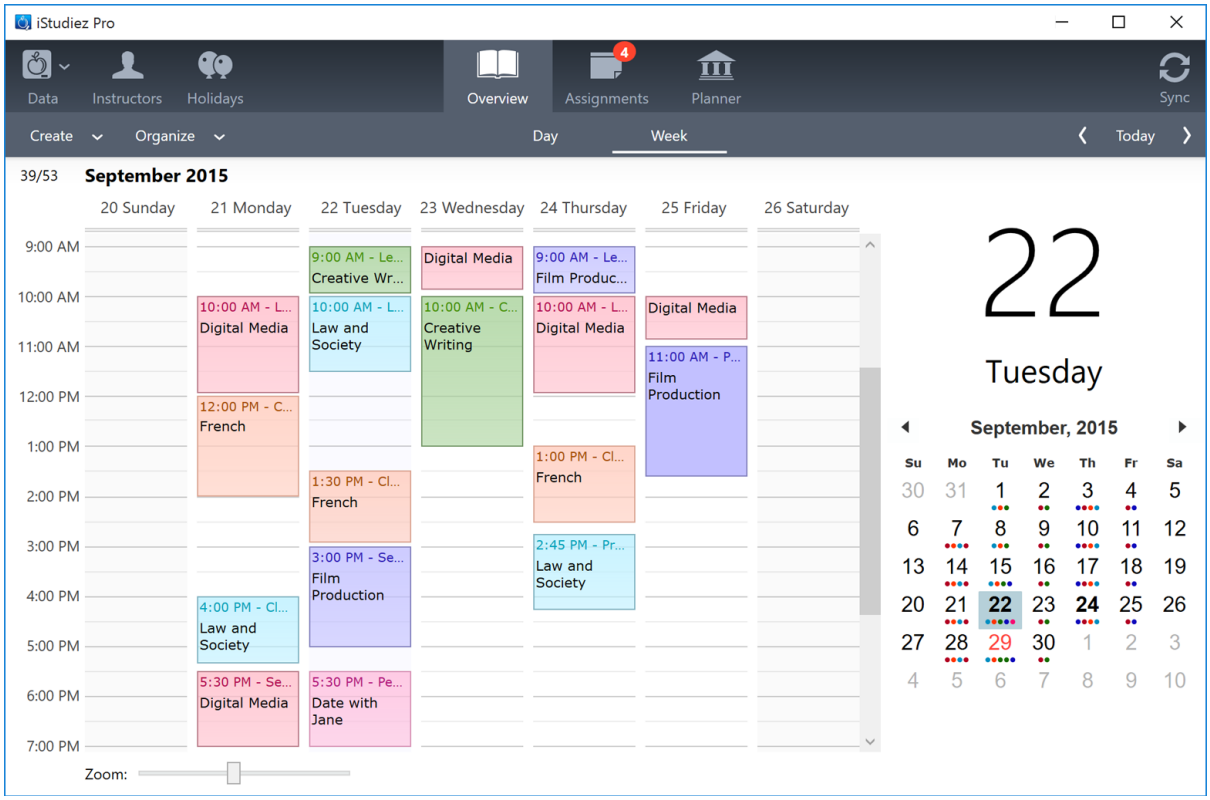
Because of its mobile nature it is a simple study planner with a lot of the required features missing.

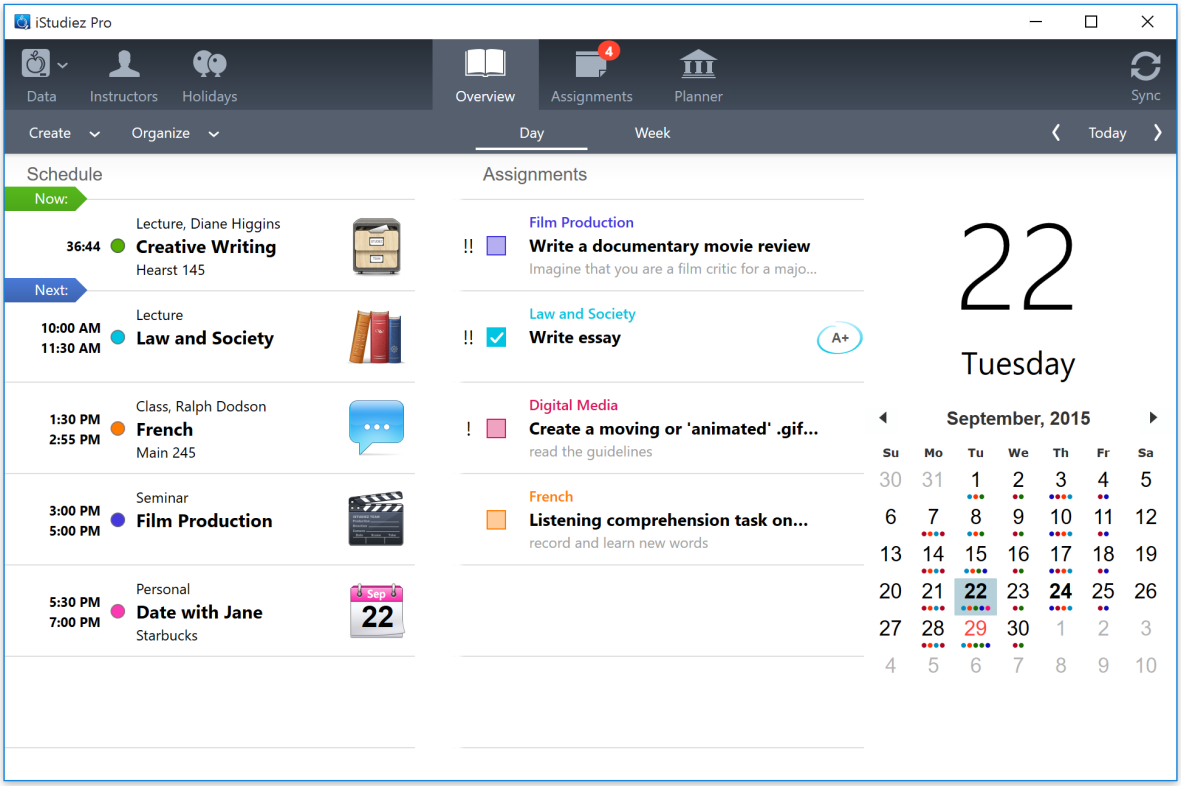
# iStudiez Pro

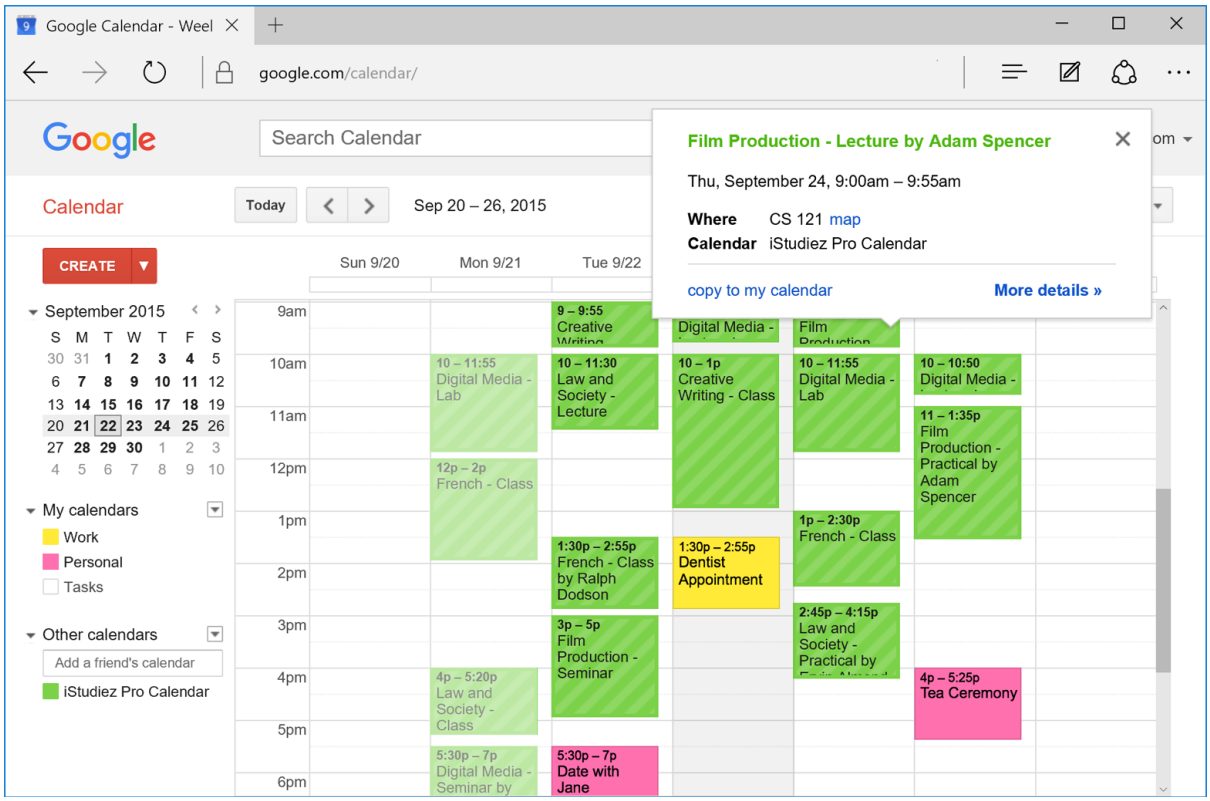
A multi-platform application helping students to schedule and keep track of their study activities.

**Similar features:**

- Quick Overview of Daily Schedule and Tasks
- Managing assignments
- Plan and manage courses and classes details, locations, instructors info, holidays and grades






**Different features:**

- Calculate grades
- Two-way integration with Google Calendar

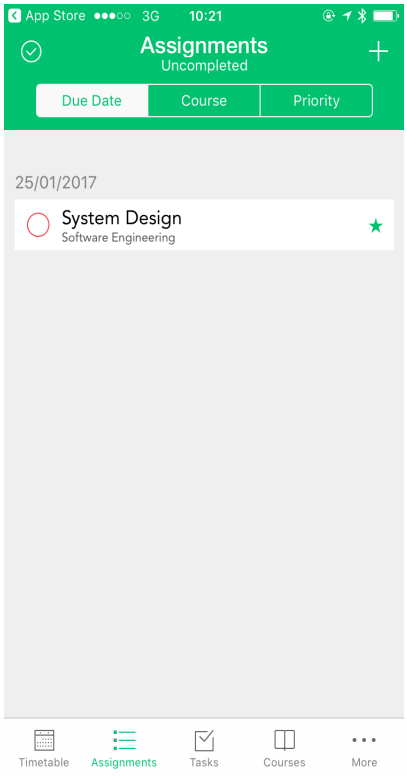
## Verdict

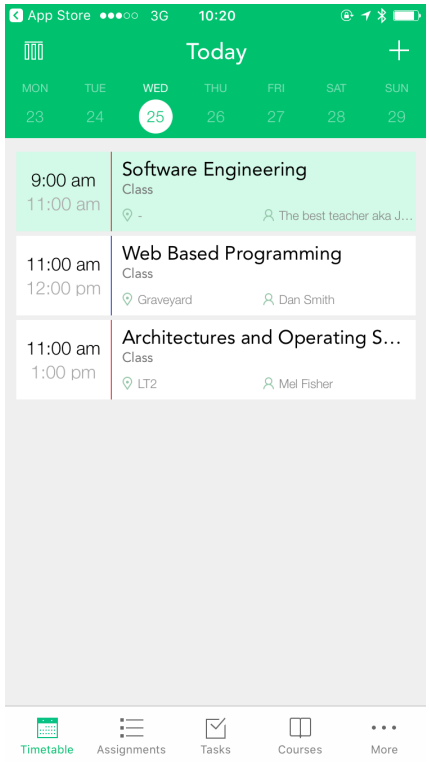
A very interesting application that can be used to decide on our features.

# Pocket Schedule

## Main features:

- Term by term timetable scheduling
- Very basic task checklist
- Assignment priority
- Ability to sync the data between different devices




## Differences

- No task dependencies and contribution options
- No import from file (impossible on iOS)

## Verdict

It is very good looking app with some basic functionality for a mobile study planner. It is missing a lot of required features, but some good ideas like assignment priority could be implemented in our system.

# Conclusion

In conclusion, this analysis assisted us on defining our main and extra features. By analysing these systems we decided what is essential for a study planner system and what is an extra feature. At the end notification feature has been added to our system design which has been influenced by these similar systems.

# MoSCoW Analysis

After reviewing the specification and seeing what was expected of the project, and comparing it to the features of other systems we came up with a MoSCoW analysis.

We also had several ideas and features that we wanted to implement, both features that we felt would be nice for the user, and others that we felt would improve the system - however as these were not detailed in the spec they were not essential deliverables, so they did not go in the most critical part.

## Must Have

- Load module, coursework and deadline information from a defined file format
- Create a study profile
- Ability to define study tasks, details, milestones and deadlines
- Ability to record study activities that contribute towards completing study tasks and milestones
- Visualise activities, dependencies, intermediate milestones and deadlines in a Gantt chart representation as well as a study progress dashboard that highlights upcoming deadlines, progress towards completing milestones and time spent for each module

## Should Have

- Creating/editing an account
- Applying for an extension
- Notifications
- Version Control

## Could Have

- Checksum for validating files
- Comprehensive documentation
- UI customization
- Native notifications support
- Mobile OS support (e.g. iOS, Android)
- Search feature

## Won’t Have

- Option to create your own study profile without the file from the Hub
- Network integration
- Coursework submission functionality

Automatic software update system

•

# List of Identified Use Cases

We quickly identified many use cases, however our initial model included the Hub as an actor. After contacting our client they clarified they only wished for the student to be considered as actor so we focussed our model focusing on their needs and interactions.

A lot of our use cases branched out to several smaller use cases. Some use cases had triggers from different points (eg, adding an activity from a module view would allow you to add all tasks from that module, whereas triggering it from a specific assignment would give you only tasks from that assignment) For those cases, where possible, we have grouped the use cases together (explaining the smalls differences) to avoid mass repetition.

## Use Case for Student

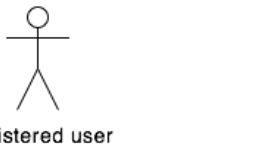
- Create account
- Edit account
- Create a study profile
- Add activity
- Apply semester update
    - *Receive an update file*
- Apply for an extension
- View study profile
    - *View module*
    - *Manage milestones*
- View study dashboard
- View module
    - *View assignment*
    - *Manage milestones*
    - *Manage activities*
- View assignment
    - *Manage milestones*
    - *Manage activities*
    - *Manage tasks*
    - *Manage requirements*
- Manage tasks (view/add/edit/remove)
    - *Manage requirements*
    - *Manage activities*
- Manage milestones (view/add/remove/edit)
- Manage activities (view/add/remove/edit)
- Manage requirements (view/add/remove/edit)
- Add/Remove dependency
- Receive notifications
- Generate Gantt Chart

# Use Case Diagrams

To help visualise our use cases, and to show how they relate to each other we produced the following use case diagrams. Due to the high number of use cases we broke our system down into multiple diagrams, with the main diagram having nodes which are explained in more detail in later diagrams.

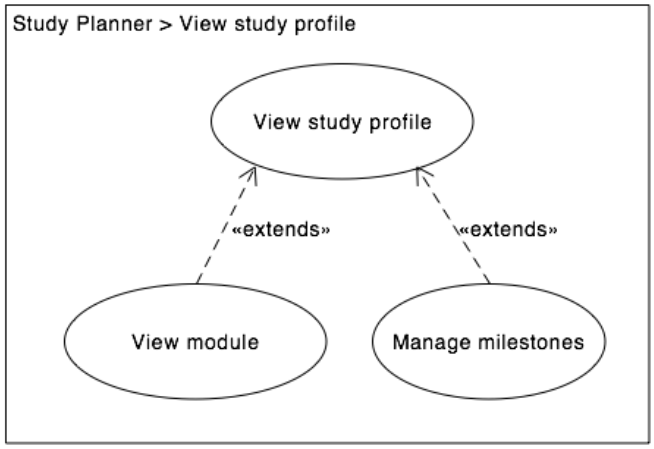
## Main Diagram

## View Study Profile



Registered user

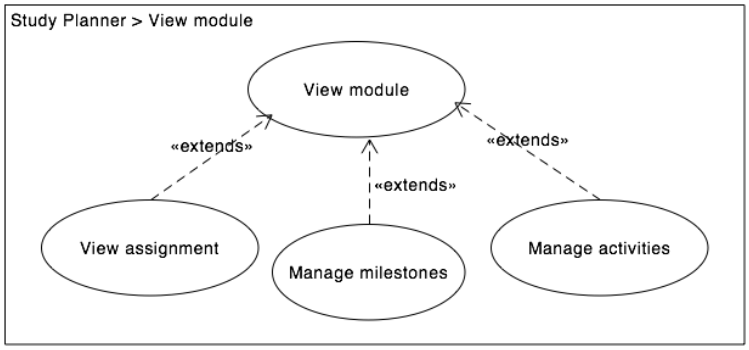
All use cases in this diagram apply to this actor



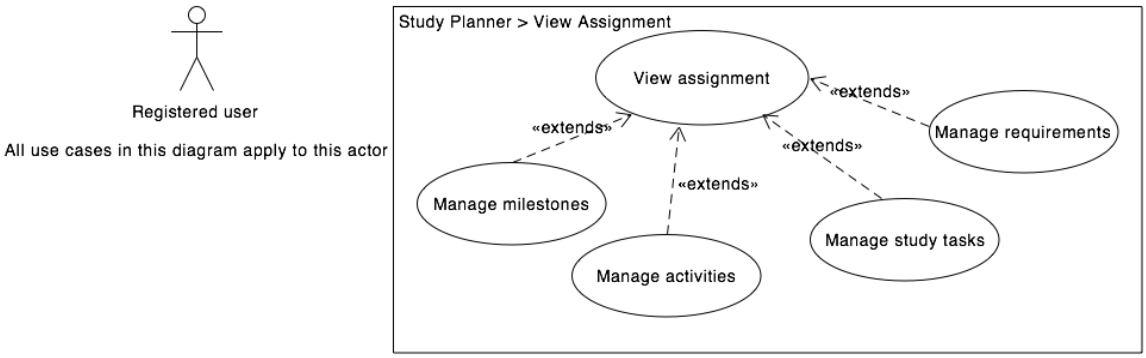

## View Module

Registered user

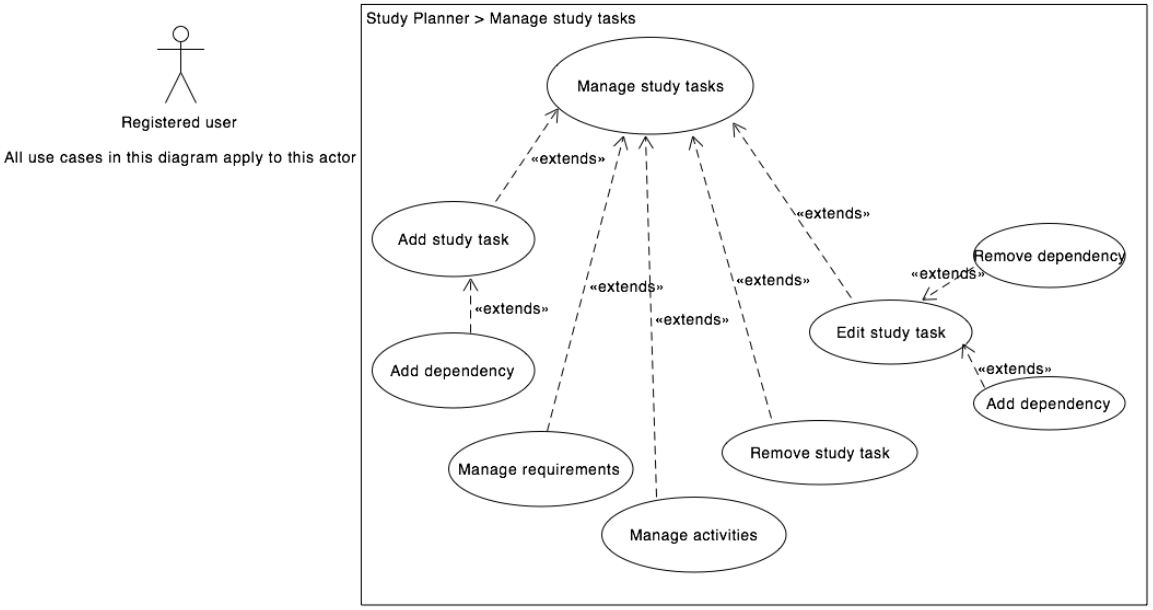
All use cases in this diagram apply to this actor



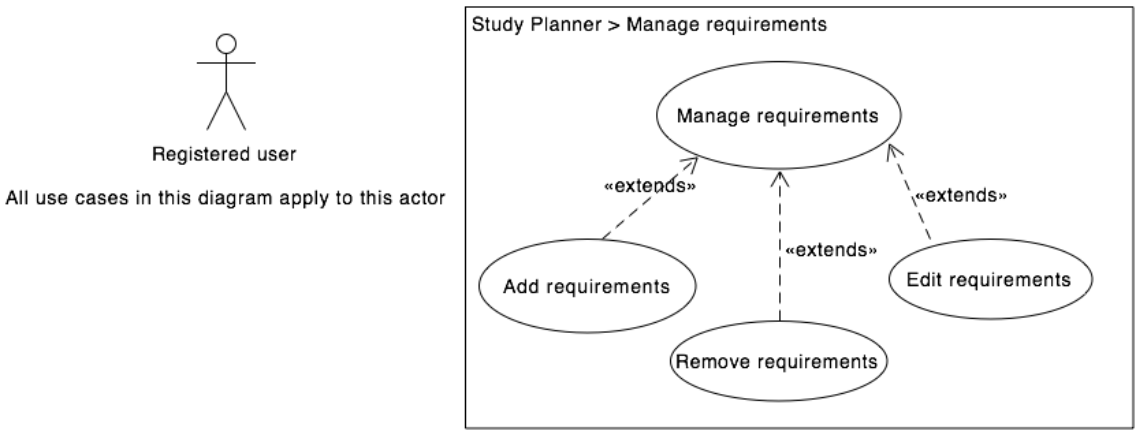
# View Assignment



## Manage Study Tasks



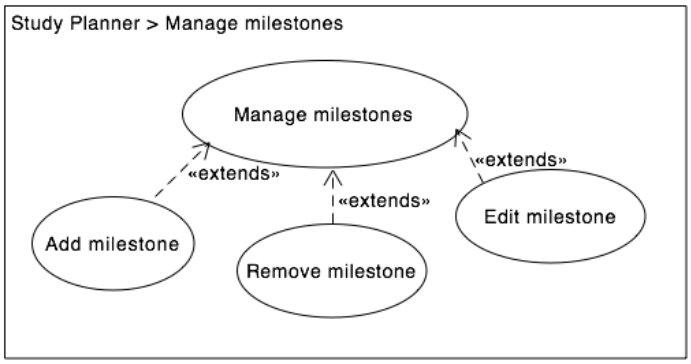

## Manage Requirements



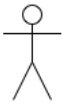
## Manage Milestones

Registered user

All use cases in this diagram apply to this actor

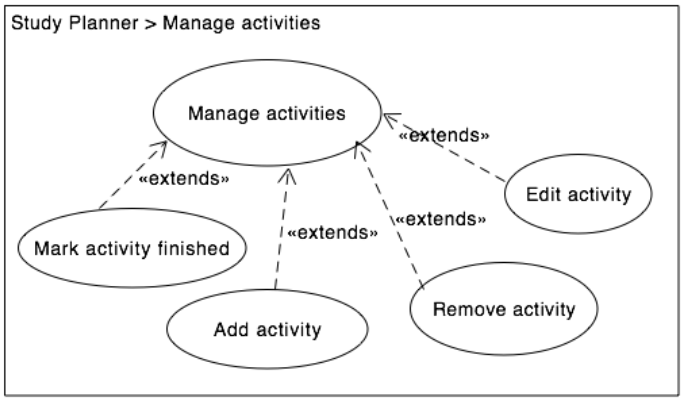



## Manage Activities



Registered user

All use cases in this diagram apply to this actor



# Object-Oriented Analysis & Initial Class Diagram

# Key Points and Ideas

One of the first things we identified was the need for version control. This is because over the course of a semester the hub will be sending out revisions to the module, such as coursework deadline changes, staffing or room changes, or if a student gets an extension request approved. In order to prevent a user from overwriting a newer update with an older one, we realised we would need a version control interface or inherited class.

We also, decided to follow a principle used by the Royal Navy's Nuclear Submarines for keeping their databases up to date. Submarines are not allowed to transmit information back when they are deployed in order to keep their location secret, yet they have maintain up to date information. As a result, they cannot inform the server of their latest database state. Therefore, the servers have to broadcast an instruction set of complete changes to the database since the last time they were in port, that will change it to the latest state from any potential state it could have been in from other received updates as they cannot confirm which ones have been received.

As the specification for this project only requires one way communication (from the hub to the user) it is a similar problem as consequently the version control system will have to consider the possibility a user has not received / installed all previous updates when receiving the next update. Therefore the version controller will include all the changes made since the initial set up of a particular study profile.

Another important point we noted was the flexible interaction of datatypes. For example. tasks were contained in many different places (Milestones, Assignments, Activities) there would be many different types of events (Assignment deadlines, exam events activity sessions) so having a robust and logical inheritance system would be crucial.

With this in mind, we set up two main parent classes, Event and ModelEntity from which the majority of our model elements were based on. A secondary parent class of VersionControl extended from ModelEntity to be applied to updates to any elements (such as an assignment) received from the hub.

# Initial List of Classes

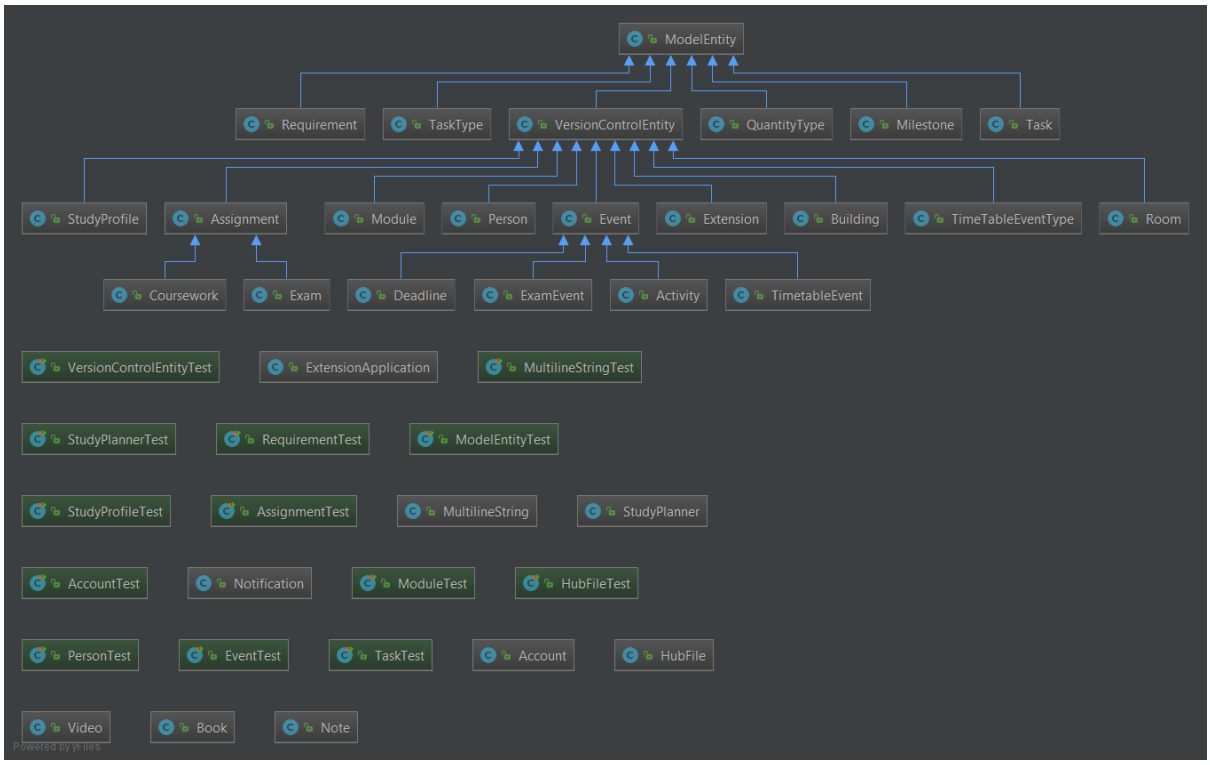
After going through the spec for our system and our MoSCoW analysis we came up with the following list of classes for managing our system.

1. MainController
2. DataController
3. StudyPlannerController
4. EventController
5. FileSystem
6. EventController
7. StudyPlannerController

1. HubFile
2. ExtensionApplication
3. StudyPlanner
4. ModelEntity
    a. VersionControlEntity
        i. Extension
        ii. Module
        iii. Assignment
            1. Exam
            2. Coursework
    b. Task
    c. Requirement
        i. Book
        ii. Video
5. StudyProfile
6. QuantityType
7. TimeTableEventType
8. TaskType
9. Milestone
10. Event
    a. Deadline
        a. Milestone
    b. TimetableEvent
    c. ExamEvent
    d. ActivityEvent
11. Person
12. Building
13. Room
14. Note
15. Account
16. UIManager
17. UIView
18. UIElement

19. UIEvent
Notification

# Initial Class Diagram




*Please note, this is available for download from our Trello board.*

# Architectural Design & Diagram

# Initial Analysis

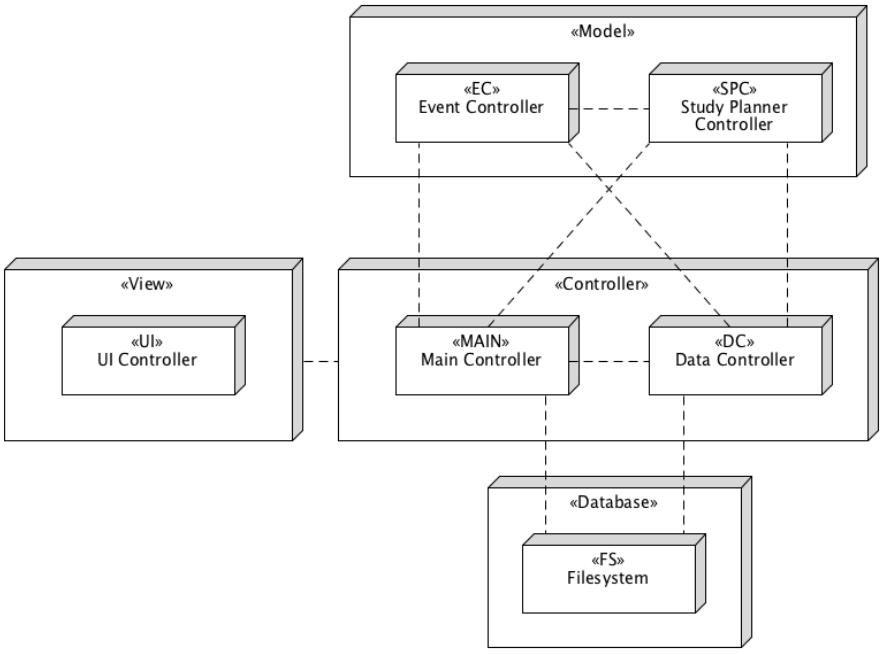
Our class system features a very detailed inheritance system for all of the elements involved in organising the study planner profiles. With many different types of interactions between the data (eg, scheduling conflicts of events, managing task dependences etc) it is very important that our system manages this efficiently and correctly. Also, as something to be used by computer users of all skills levels, it would require a user friendly design, meaning an easy to use GUI.

# Gut Instinct: Model/View/Controller

When we sat down to discuss which major architecture to use, we all went into the the meeting with the same idea: it should be MVC. As both Zil and Andrew had used it before, they felt most comfortable working with it, and both Ben and Bijan had concluded during the lectures that it felt the best fit for what we were doing.

The Model fits well with our strong data modelling of the elements, the control would coordinate the possible conflicts and organisation, and the view would handle UI.

We quickly produced the following loosely modelled controller based diagram on how we felt the system would interact, however, we felt it important to consider whether there were alternative systems.

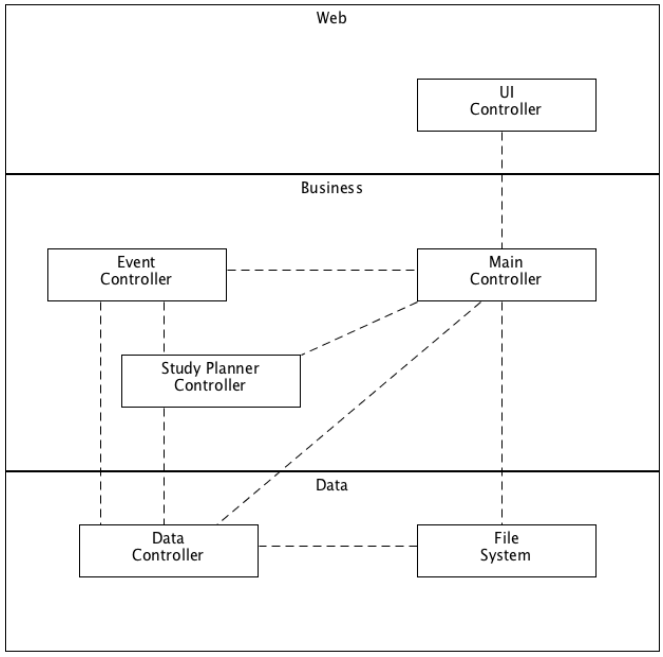



*Note - very simple representation*

# Nice Alternative: Web/Business/Data

As it turned out, our system could also be handled using a Web/Business/Data model. This could be a useful system, if we were developing a web app or if we decided to move our system to also be online. It has good user interaction, and the business layer can handle all of the scheduling/conflicts mentioned earlier. The data layer is separated from the user meaning it would have good integrity.

We produced the following diagram, modifying our MVC prototype so show how the system fits in a Web Business Data architecture.



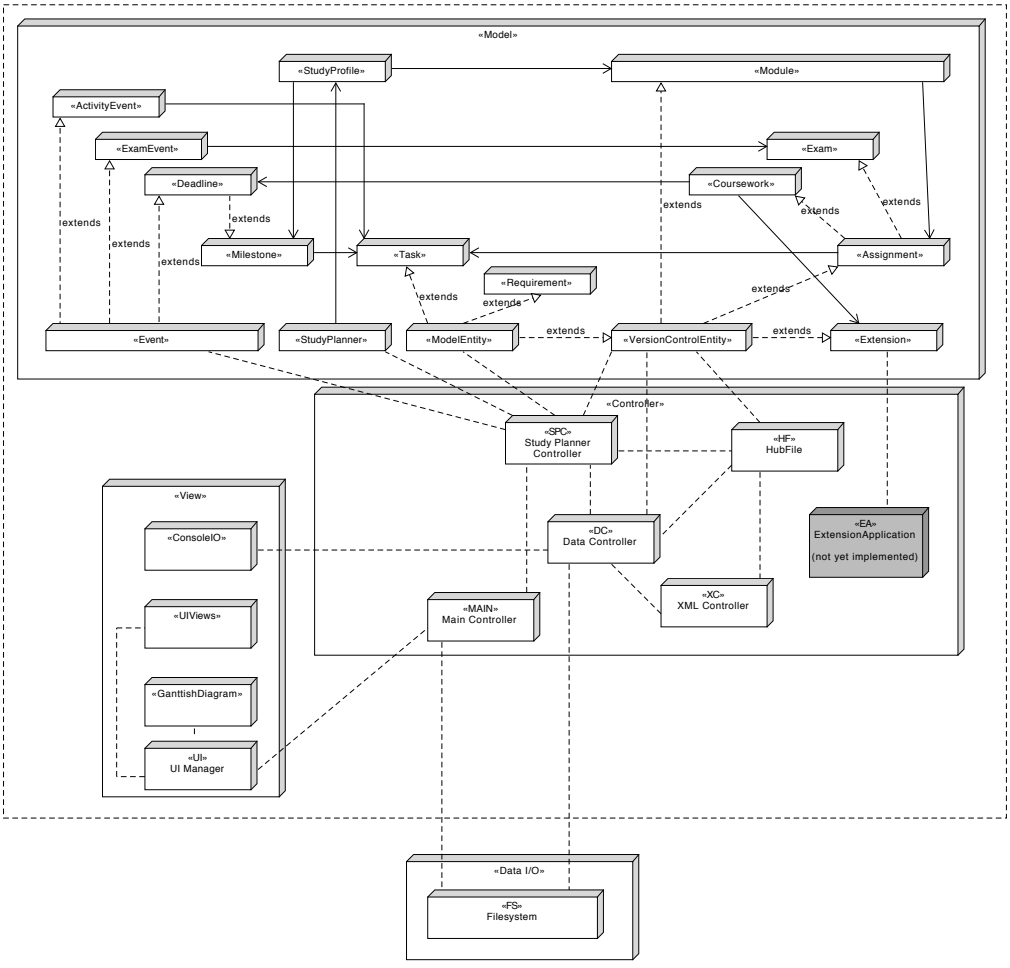

*Note - very simple representation*

# MVC is the best option

However, in the end we felt that MVC was the better choice. For one thing, our team already has experience, but more importantly, it better manages the relationships between the entities than Web/Business/Data. While WBD is good for web apps, MVC also works for this use case.

## Detailed MVC Diagram

Here is a detailed diagram showing our architecture and how the classes are connected. Our  MVC shows an additional section - Data I/O - this was added so the system can be upgraded for alternative methods of data storage in the future - eg network access, downloading from the internet, etc…



# Object-Oriented Design & Detailed Class Diagram

# Applying Detail to the Madness

Once we had settled on our architecture and our classes it was time to refine how they interacted. The first stage was to go through our model class and determine how the data was set and retrieved. As we had many attributes required get methods set methods and constructor methods we went through all of our attributes listing in short hand whether they required a get method, a set method, or various constructor methods: argument in constructor, optional argument in constructor or generated in constructor (eg, uid)

After we had done this, we went through each part of the process and determined how they would interact based on the relationships determined in the initial class diagram and the architecture diagram. This was later refined when we made our sequence diagrams and realised we needed some methods not yet defined, and we are also aware that during the implementation phase more methods will be added, so this is not complete, but a work in progress.

Also, as this is the first time any of us is doing GUI with Java, we are not familiar with the concepts or frameworks yet, so these elements are still quite vague and generic, but based on our general understanding and knowledge of UI in other systems.

## Key for lists

attributesAreInBlack (variableTypes)

**gc** = generated in constructor (ie date stamp or uid)
**ac** = argument in constructor (ie passed as an argument)
**oc** = optional argument constructor
**ogc** = option argument -> generated if missing constructor
**s** = set methods
**g** = get methods (for arrays this can return a specific element)
**e** = edit methods (eg array handling / objects)

methodsAreInRed(argumentTypes) (returnType)

constructorsAreInBlue

+public
-private
#protected

# Model Classes with Methods/Attributes

Classes:

1. StudyPlanner
    a. Generic
        i. -account (Account) **g,e**
        ii. -quantityTypes (ArrayList QuantityType) **g, e**
        iii. -taskTypes (ArrayList TaskType) **g, e**
        iv. -studyProfiles (ArrayList StudyProfile) **g, e**
        v. -activityList (array ActivityEvent) **g,e**
        vi. -timeTableEventTypes (array TimeTableEventType) **g,e**
        vii. -calendar (array Event) **g,e**
        viii. -hubFiles (ArrayList FileLocations of Serialised HubFile) **g,e**
        ix. +loadFile (String filePath) (HubFile) loads files into HubFile object, and moves a copy to the program data folder (throws error if invalid)
        x. +processHubFile(HubFile newHubFile) (void) - takes HubFile - if new, creates StudyProfile parsing newHubFile to the constructor, else updates existing StudyProfile, throwing error if no such StudyFile has been created (ie, user attempts to load an update file, before creating initial version)
2. ModelEntity
    a. Generic
        i. #name (String) **ac,g,s**
        ii. #details (ArrayList String) // used for description too **oc,g,s**
        iii. #notes (ArrayList Note)
3. ModelEntity.VersionControlEntity
    a. Generic
        i. #version (int) **ac,g**
        ii. #uid (String) **gc, g**
        iii. +update(VersionControlEntity receivedVCE) (boolean) - if receivedVCE is newer than self, update and return true, else false
4. StudyProfile
    a. Generic
        i. -modules (array Module) **g**
        ii. -milestones (ArrayList Milestone) **g,e**
        iii. -name (String) **g,s**
        iv. -details (array String) **g,s**
        v. -year (int) **g**
        vi. -semesterNo (int) **g**
        vii. -version (int) **g,s**
        viii. +view (UIView) **STATIC,FINAL**
        ix. +milestonesCompleted() (int)  returns number of completed

milestones
x. +milestonesProgress() (double) returns proportion of completed milestones (0.0 to 1.0)
xi. CONSTRUCTOR(HubFile initialHubFile) - constructs a new study profile, from the initalHubFile)

5. VersionControlEntity.Module
    a. Generic
        i. -assignments (ArrayList Assignment) **g, e, oc**
        ii. -organizer (Person) **g,s**
        iii. -moduleCode (String) **g,s**
        iv. -timetable (ArrayList TimetableEvent) **g,e,oc**
        v. +view (UIView) **STATIC,FINAL**
6. VersionControlEntity.Assignment
    a. Generic
        i. #tasks(ArrayList Task) **g,e**
        ii. #requirements(ArrayList Requirement) **g,e,oc**
        iii. #weighting (int) **g,s,ac**
        iv. #setBy (Person) **g,s,ac**
        v. #markedBy (Person) **g,s,oc**
        vi. #reviewedBy (Person) **g,s,oc**
        vii. #marks (int) **g,s,oc**
        viii. #state (enum IN PROGRESS / DEADLINE PASSED / NOT STARTED)
        ix. +view (UIView) **STATIC,FINAL**
    b. Exam
        i. -resit (Exam) // null = no resit **g,s**
        ii. -timeslot (ExamEvent) **g,e**
    c. Coursework
        i. -startDate (Event) **oc,g,s**
        ii. -deadline (Deadline) **oc,g,s**
        iii. -submissionDate (DateTime) **g,s**
        iv. -extensions (arrayList Extension) //size 0 = no extension **g,e**
7. ModelEntity.Task
    a. Generic
        i. -dependencies (arrayList Task) **g,e**
        ii. -deadline (Deadline) **g,s,ac**
        iii. -requirements (array Requirement) **g,e**
        iv. -notes (ArrayList Note) **g,e**
        v. -checkedComplete (boolean) **s,g**
        vi. -weighting (int) **g,s**
        vii. -type (TaskType) **g,s,ac**
        viii. +view (UIView) **STATIC,FINAL**
        ix. +dependenciesComplete() (boolean) are all the dependent Tasks complete (returns true if no dependent tasks)
        x. +hasDependencies() (boolean) returns true if Task has and tasks in its dependencies
        xi. isComplete() (boolean) checks whether all requirements and

activities have been completed and the checkedComplete variable is true

xii. canCheckComplete() (boolean) checks where it's possible to check the task complete (ie all activities and requirements completed)

8. QuantityType
    a. Generic
        i. -name (String) **g,s, ac**
9. TimeTableEventType
    a. Generic
        i. -name (String) **g,s, ac**
10. TaskType
    a. Generic
        i. -name (String) **g,s, ac**
11. ModelEntity.Requirement
    a. Generic
        i. #checkedComplete(boolean) **g,s**
        ii. #estimatedTimeInHours (double) **oc,g,s**
        iii. #activityLog (ArrayList ActivityEvent) **g,e**
        iv. #initialQuantity (int) **g,ac**
        v. #remainingQuantity (int) **g,s, gc (from initialQuanitiy)**
        vi. #quantityType (QuantityType) **ac, g, s**
        vii. +view (UIView) **STATIC,FINAL**
        viii. isComplete() (boolean) has requirement been completed
        ix. requirementProgress (double) 0.0 - 1.0 representation of requirement progress (based on remainingQuantity and initialQuantity
    b. Book
        i. -Chapters (array String) **oc,g,s**
    c. Video
        i. -URL (String) **oc,g,s**
12. Deadline.Milestone
    a. Generic
        i. -tasks (ArrayList Task) **g, e**
        ii. -notes (ArrayList Note)
        iii. +view (UIView) **STATIC,FINAL**
        iv. +getName() (String) -> gets name from deadline.name
        v. +setName() (String) -> sets name to deadline.name
        vi. +isComplete() (Boolean) -> are all tasks complete?
        vii. +progressPercentage() (double) -> computes progress based on Tasks Completed and their weighting (between 0.0 and 1.0)
        viii. +tasksCompleted() (int) -> returns number of tasks completed
        ix. +size() (int) -> returns tasks.size()
        x. +totalWeighting() (int) -> returns the sum of all of the tasks weightings
13. Event
    a. Generic
        i. #date (Date) **ac, g, s**

ii. #name (String) **ac, g, s**
iii. #details (array String) **oc, g, e**
iv. #notes (ArrayList Note)

b. Deadline
i. -time (Time) **ac, g, s**

c. TimetableEvent
i. -time (Time) **ac, g, s**
ii. -room (String) **ac, g, s**
iii. -lecturer (Person) **oc, g, s**
iv. -timeTableEventType (TimeTableEventType) **oc,g,s**
v. -duration (int) // in minutes **ac, g, s**

d. ExamEvent
i. -time (Time) **ac, g, s**
ii. -room (Room) **ac, g, s**
iii. -duration (int) // in minutes **ac, g, s**

e. ActivityEvent
i. -tasks (array Task) **oc, g,e**
ii. -checkedComplete (boolean) **g,s**
iii. -start (Time) **ac, g, s**
iv. -duration (int) // in minutes **ac, g, s**
v. -description (String) **ac,g,s**
vi. -activityQuantity (int) **g,s,ac**
vii. -type (QuantityType) **g,s,ac**
viii. +view (UIView) **STATIC,FINAL**
ix. +markCompleted() marks the activity as complete
x. +isCompleted() (boolean) has the activity been completed?

14. Person
a. Generic
i. -firstName (String) **g,ac,s**
ii. -lastName (String) **g,ac,s**
iii. -salutation (String) **g,ac,s**
iv. -email (String) **g,ac,s**

15. Building
a. Generic
i. -name (String) **g,ac,s**
ii. -code (String) **g,ac,s**
iii. -longitude (double) **g,ac,s**
iv. -latitude (double) **g,ac,s**

16. Room
a. Generic
i. -name (String) **ac,g,s**
ii. -building (Building) **g,ac,s**
iii. -roomNumber (String) **g,ac,s**
iv. -details (array String) **g,ac,s**

17. Note
a. Generic
i. -title (String) **g,ac**

ii. -date (DateTime) **g,ac**
iii. -text (String) **g,ac**

18. VerisionControlEntity.Extension
    a. Generic
        i. -newDeadline (Deadline) **ac, g, s**
        ii. -circumstances (String) **ac, g, s**
        iii. -approvalStatus (enum PENDING / APPROVED / DECLINED) **ogc, g, s**
19. Account
    a. Generic
        i. studentDetails (Person) **g,e**
        ii. studentNumber (String) **g,s**

# View Classes with Methods/Attributes

1. UIManager
    a. Generic
        i. +displayView(UIView)
        ii. +updateTasks(ArrayList<Task>)
        iii. +validateCurrentForm()
        iv. +previousScreen()
2. UIView
    a. Generic
        i. -title (String) **ac,s, g**
        ii. -elements (ArrayList UIElement) **ac,e, g**
        iii. -events (ArrayList UIEvent) **ac,e, g**
        iv. +updateTasks(ArrayList<Task>)
        v. +validateCurrentForm()
        vi. +previousScreen()
3. UIElement
    a. Generic
        i. -title (String) **ac,s, g**
        ii. -events (ArrayList UIEvent) o**c,s, g**
        iii. -xPos (int) **ac,s, g**
        iv. -yPos (int) **ac,s, g**
        v. -width (int) **ac,s, g**
        vi. -height (int) **ac,s, g**
    b. Button
        i. -events (ArrayList UIEvent) **oc,e,g**
4. UIEvent
    a. Generic

20. Notification
    a. Generic
        i. -title (String) **ac, g**
        ii. -dateTime (DateTime) **ac,g**
        iii. -details (array String) **ac,g**
        iv. -read (boolean) **gc, g, s**
        v. -link (Object) // we will probably create a superObject for everything in here to sit under with basic info **ac, g**

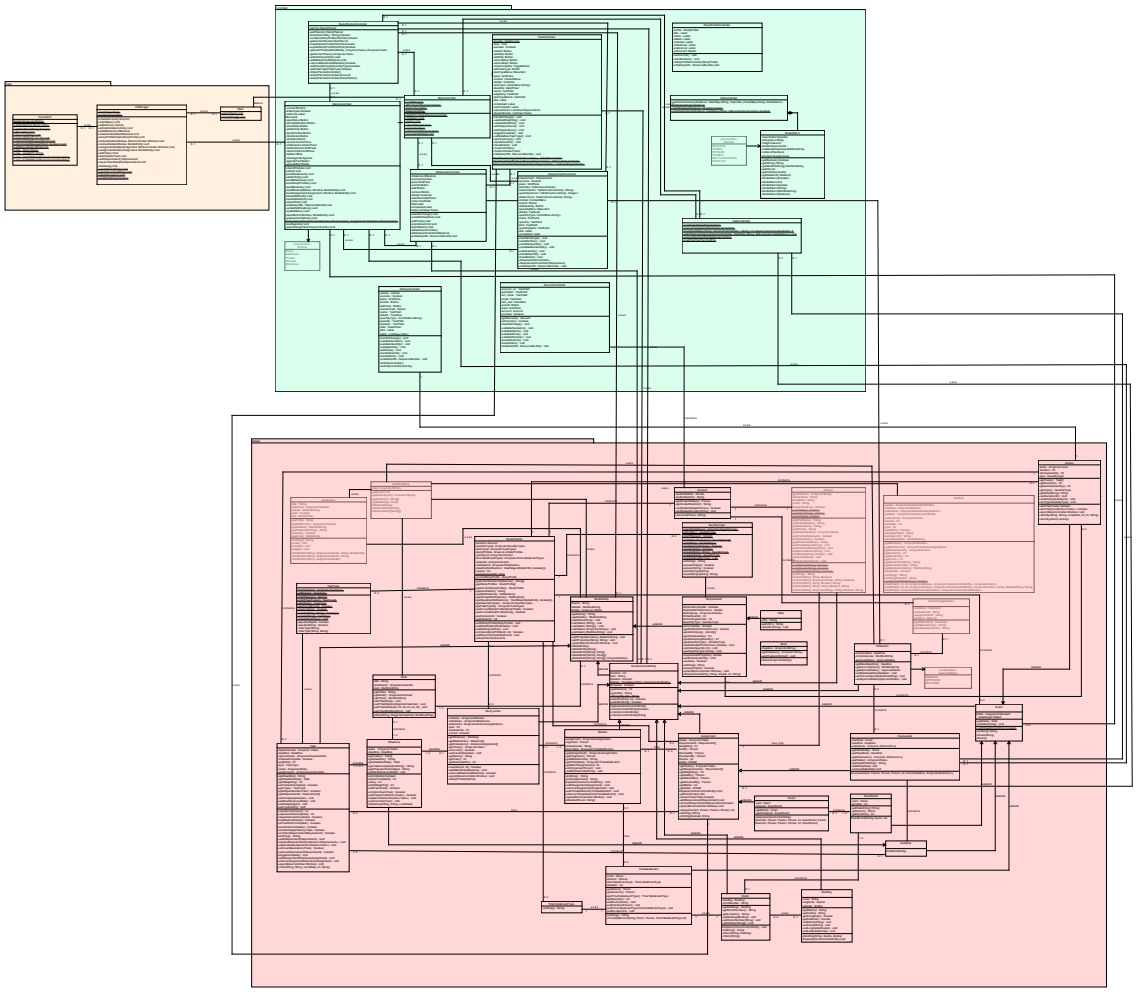
# Controller Classes with Methods/Attributes

1. MainController
    b. Generic
        i. initialise()
        ii. loadAccount()
        iii. saveAccount()
        iv. +submitFile(String) (boolean) passes a String filepath for validation and processing
        v. +addActivity(currentModelEntity) (UIView) instructs the Main Controller to prepare for adding new activity and returns a new UIView
        vi. +updateRelevantTasks(ModelEntity, ArrayList<Task)
        vii. +newActivityTasksAdded() (UIView)
        viii. +addActivity(UIView, ArrayList<Task>)
5. DataController
    a. Generic
        i. getTasks() (ArrayList Tasks)
        ii. +processFile(hubFile) (boolean) goes through the second step of file validation and if the file passes it, processes the file
        iii. getActivityQuantityTypes() (ArrayList<QuantityTypes>)
        iv. addActivity(Activity)
6. StudyPlannerController
    a. Generic
        i. +fileValidation(String) (boolean) first step for validating a file (format, structure, etc.)
        ii. +containsStudyProfile(hubFile) (boolean) checks whether the StudyProfile in this file already exists in the system
        iii. +getCurrentVersion(hubFile) (int) if this Study Profile already exists, check which version the system has loaded in
        iv. +createStudyProfile(hubFile) (void) creates a new Study Profile based on the provided file
        v. +updateStudyProfile(hubFile) (void) updates the study profile with the provided file
        vi. +getListOfTasks(ModelEntity, ArrayList<Task>) (ArrayList<Task>) add all valid Tasks associated with this ModelEntity to the given ArrayList
        vii. newActivity(UIView, ArrayList<Task>)
7. EventController
    a. Generic
        i. +checkConflict(Event) (boolean)
8. FileSystem
    a. Generic
        i. +saveDatabase()
9. EventController
    a. Generic
        i. eventConflicts(Event,ArrayList Event) (boolean)
10. StudyPlannerController
    a. Generic

        i. getListOfTasks(ModelEntity) (ArrayList Task)

21. HubFile
    a. Generic
        i. -modules (ArrayList Module) **gc, g**
        ii. -extensions (ArrayList ExtensionApplication) **oc, g**

# Detailed Class Diagram



*Please note, this is available for download from our Trello board.*

# Sequence Diagrams

# Introduction

When choosing our two sequence diagrams to show in detail we felt it important to choose two that highlight the strengths and features of our system, its flexibility, and the considerations we have given to possible points of failure.

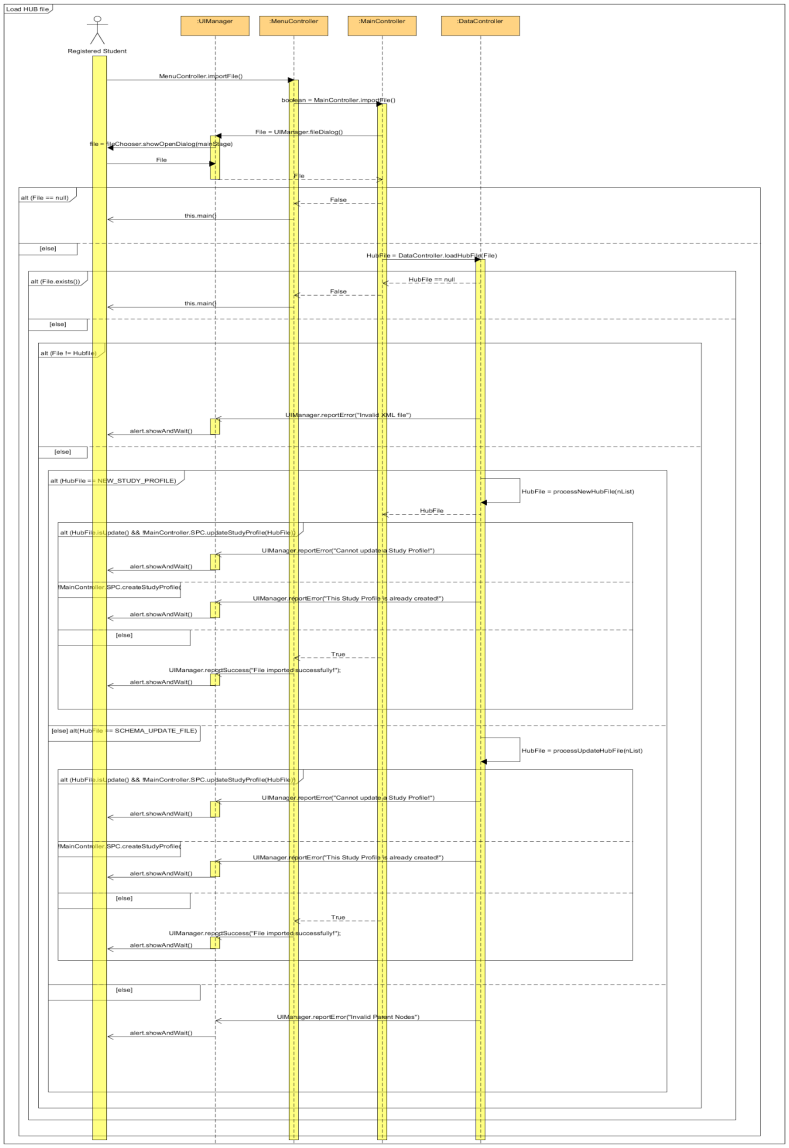
Loading a Hub File, utilises our version control system. It also deals with possible corrupt or incorrect files. It also has two success paths (the first is if the file is a new semester study profile, the second is if it is a semester update file).

Adding an activity is one of the most important tasks. This is because we feel it will probably be the most used part of the system (ie, a user may quickly log some work they have done, or may schedule something soon, and the most prolific users will be doing this multiple times a day).

# Sequence 1: Load a Hub File

Below is the sequence diagram for loading a hub file.

A few details to consider is that we intend to use an XML based file structure for sending the data from the hub. This file will also contain a checksum property (possibly just the number of bytes used) to ensure that the received file is complete. It's details like this that will be used to determine whether the file is valid.

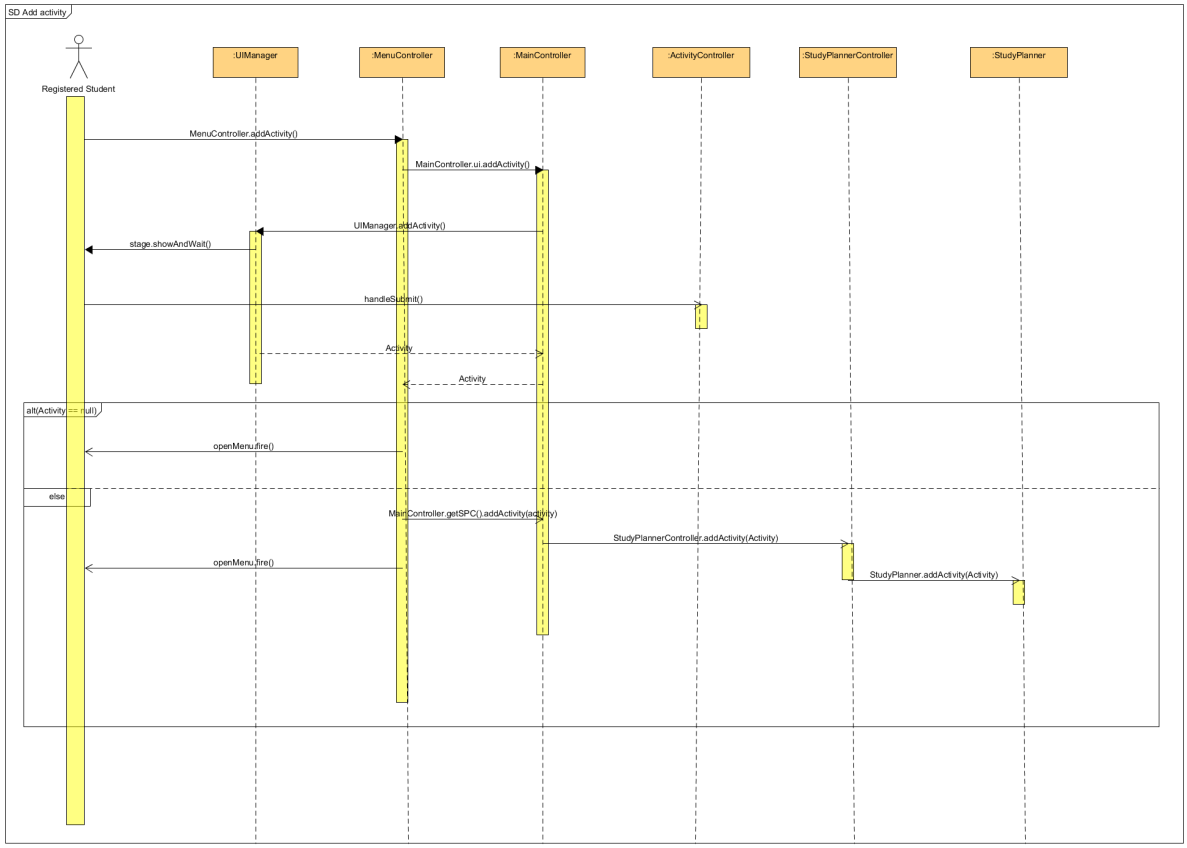



*Please note, this is available for download from our Trello board.*

# Sequence 2: Add an Activity

Below is the sequence diagram for adding an activity.

An important detail to note is that this is an action that can be triggered from multiple places. For example, it could be triggered from the home page of the app, meaning the user is presented with a full list of possible tasks. Alternatively, it could be triggered from a specific assignment, thus presenting the user with just the tasks for that assignment. Therefore, the system will determine which tasks are relevant to the screen that the user triggers it, and this is detailed in the sequence diagram.




*Please note, this is available for download from our Trello board.*

# State Charts

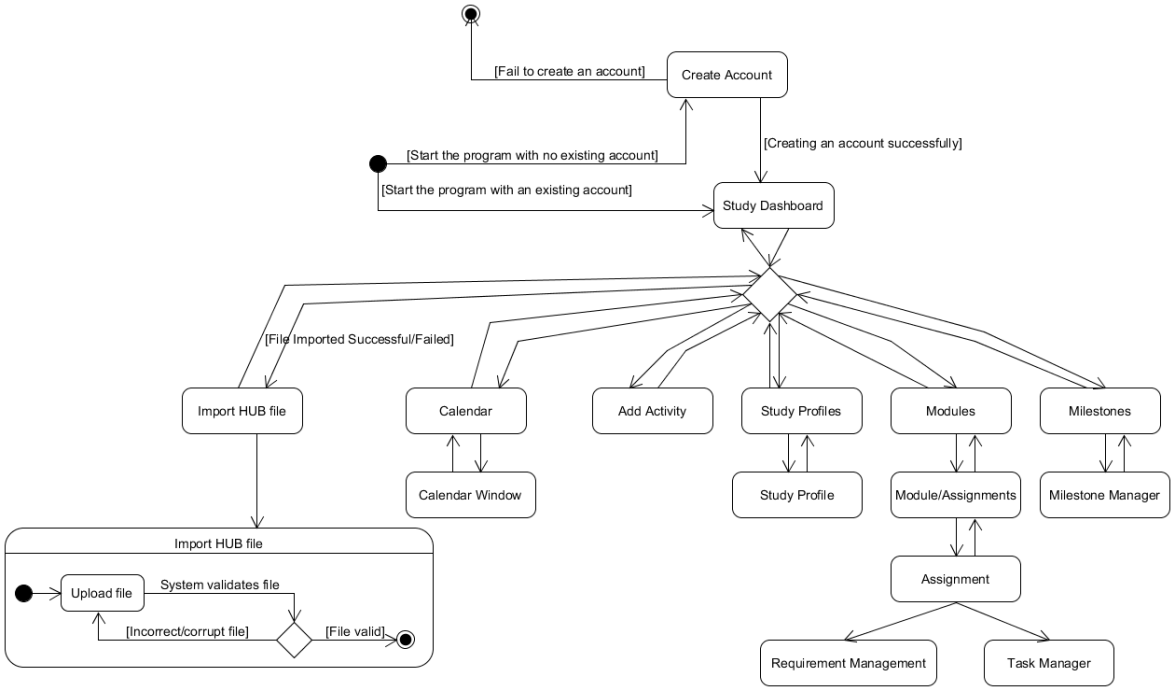
# Introduction

The GUI State Transition is the most important state diagram to get an understanding of the overall system as it shows all of the places the user can go, and how they get there.

Loading a Hub File was a good second choice, as we previously detailed the sequence diagram, and it demonstrates that we have carefully considered this important part of our system, from multiple different points of view.

# GUI State Transitions

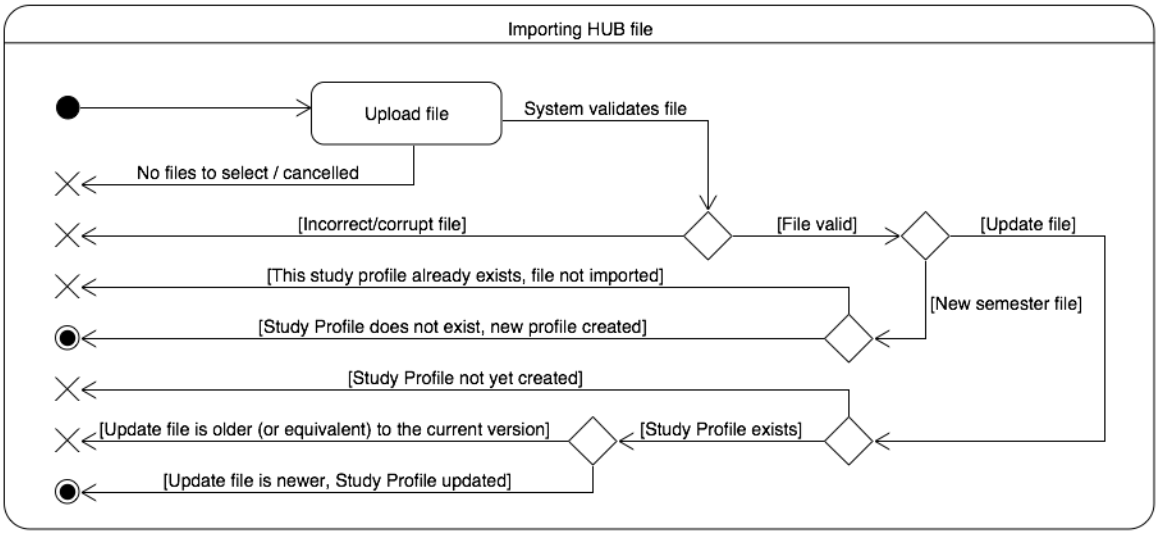
Below is our state transition diagram for the GUI State Transitions. An important property of this is that some views are accessed from multiple locations and therefore the return location is specified as the previous view.




*Please note, this is available for download from our Trello board.*

# Loading a Hub File Transitions

Below is our state transition diagram for loading a hub file. As previously mentioned, this can be compared to our sequence diagram, and our class diagram.




*Please note, this is available for download from our Trello board.*

# Conclusion

# Analysis of our Proposal

We are very pleased with the model we have come up with for handling the study planner, particularly the inheritance model and the version control. We feel this will be very robust and flexible, and we are very excited to begin on the implementation phase.

We are aware of our limited knowledge of working with GUI in Java, however as we have experience in using GUI in other languages and IDEs, we understand the principles so eagerly wait to discover how to do this in Java.

# What We Feel We Have Learned

One of the things we noticed while developing this project is how quickly things changed. While the general principles of our design stayed true, the detail could often change on a meeting by meeting basis, meaning it was important to stay on top of when and where these changes happened

Going forwards, we are planning to adapt our organisation so changes are communicated more efficiently.

However, we feel this was an important learning objective of the coursework, particularly if we were to work in an Agile environment. In the end, we feel we have presented a strong product that will continue to evolve as we proceed with implementation.

# Appendix

# Glossary

***Activity -*** an activity is a piece of work associated with a specific task or tasks and contributes to the completion of study tasks and milestones. Each activity has a start date, end date, tasks it relates to, time spent on it and any notes that the user chooses to add. Each activity is associated with a quantity that can be used to evaluate its contribution, such as having studied for three hours, created three coursework parts, etc. Each activity must be attached to at least one task, and the details of the activity contribute towards completing the task. For example if a task requires five hours of studying and a related activity lists that the user has studied for two hours, this means that in the task there are three hours of studying left. An activity can be attached to multiple tasks and thereby contribute to the completion of both. Activities, together with other information, can be visualised by means of a Gantt chart representation.

***Task -*** each task has its deadline and progress count, requirement criteria based on which progress and completion are assessed (such as time studied, book chapters covered), dependencies on other tasks and any notes added by the user. In addition, milestones can be associated with a task. Study tasks must belong to a specific assessment event of the module (coursework, exam) and capture the time that will be spent and the type of a task. Activities are attached to a task and contribute towards the completion of that task. Tasks contribute towards the completion of coursework assignments or preparation for the exam. The application can visualise the planned tasks and their dependencies by means of a Gantt chart representation.

***Dependency -*** tasks can have dependencies on each other, such as a task cannot be started before another has been completed. Other tasks can be done in parallel. Dependencies, together with the tasks they are associated with, can be visualised by means of a Gantt chart representation.

***Milestone -*** intermediate deliverables with their own deadline. Milestones must be related to the tasks that are required to be completed before the milestone is achieved. Milestones can be displayed in the Gantt chart representation.

***Requirement -*** associated with each task (and by inheritance a milestone) and specifies an amount of work (such as time studied, book chapters covered, etc.) to be done for the task to be completed.

***Study Progress -*** representation of the amount of work done or left to be done for a specific task, milestone or assignment. Progress is measured by the completion of activities which are defined by the user.

***Study Dashboard -*** *a user interface from which the user can quickly see a summary of important information (eg, upcoming deadlines, progress on milestones, time spent on each module, etc)*

They can also view a Gantt chart showing the individual module planning.

***Study Profile -*** a profile containing relevant information about a semester.

***Student -*** a student studying at the University on a course requiring use of the study planner. They will be an end user, only able to access their own User Account, and be able to make appropriate changes.

***Hub -*** the administration and contact point of a school within the university. They will have administrator access to all accounts for students studying on modules in their school.

***User Account -*** an account for accessing the system. These will have different levels, such as student and admin, able to do different things on the system.

***Deadline -*** a pair of date and time which is associated with an assignment, milestone or a task and indicates the date and time an assignment should be submitted or a milestone or task should be completed.

***Assignment -*** can either be a coursework or an exam. Holds relevant information such as a list of tasks, requirements etc.

***Module -*** a specific subject of study that is associated with a number of assignments and holds information about the module organizer, module code and timetable.

***Weighting -*** value of a task indicated by the user or an assignment (by percentage).

***Quantity -*** a unit that is used to measure the amount of work of an activity.

***Update file -*** a file provided by the HUB to the student, containing updated information about the changes to the timetable or extensions for assignments.