Grade prediction and performance application

Part A – Requirements

1. *Functional requirements summary*

<R1>The software allows the user to build a course.</R1>
<description>The user is able to build a course in order to find out the outcome of his degree, real or by predicting marks.</description>

<R2>The software shall allow the user to record the name of the course.</R2>
<description>The student is able to add the course name. The user should be able to change the name of the course, for example, when a student decides to switch to a different course.</description>

<R3>The software shall allow the user to add modules to the course.</R3>
<description>The student can create modules and add them to the course.</description>

<R4>The software shall allow the user to add a credit value to each module.</R4>
<description>Each module has a credit value. Most of them are 15 credits but there are also modules with 30 credits. There is also a final year project worth 45 credits. Therefore, the user should be able to add a credit value to the each module.</description>

<R5>The software shall allow the user to input assessments(tutorial, coursework, exam) for each module.</R5>
<description>The user can add assessments to each module, including a tutorial exercise, coursework, exam et al. Each assessment has name and weight.</description>

<R6>The software should show the new module added.</R6>
<description>After each module entry, if everything is correct, the module will be shown in the software. However, some information can be deleted in case some of errors. For example if the assessments do not add up to 100% or invalid weight for assessments.</description>

<R7>The software shall allow the user to add modules until the course is complete.</R7>
<description>A course should be complete when the sum of 120 credits per year has been reached in all three years of study.</description>

<R8>The software shall allow the user to save the status of the course.</R8>
<description>The user should be able to save manually the status of the course using a save button. The software will also save automatically any changes on exit.</description>

<R9>The software shall continue from the last edit version.</R9>
<description>If the software is not at its first run, then it should use the existing data to build the interface and resume to the last edit stage.</description>

<R10>The software shall save the file changes in the background.</R10>
<description>The software will save changes in the background to simulate writing to a database.</description>

<R11>The software shall update any changes in the summary tab (level summary and main one) as the user edits any module details for the purpose of predicting the outcome of his degree.</R11>
<description></description>

<R12>The software shall allow the user to add modules for 3 years of study.</R12>
<description>The user can add modules for 3 years of study. Each module will have a credit amount. All modules should add to 120 credits per year.</description>

<R13>The software shall show results on each level summary.</R13>
<description>On a level summary the user can see the outcome of each module. For example: User Experience Design: 45% Pass or 35% Referral or 25% Fail.</description>

<R14>The software shall have a main Summary tab where the user can see the final outcome for each level of study and the final outcome of their degree.</R14>
<description>The main summary tab will cover all three levels of study with each percentage and the final outcome for the user's degree. For example: level 4: 44%, level 5: 60% , level 6: 75%. Outcome: second upper class degree.</description>

<R15>The software shall allow the user to delete a course.</R15>
<description>The user is able to delete a course. The user might want to change the course. Or to create the final outcome for a friends' degree without deleting his own data.</description>

<R16>The software shall allow the user to add the name of the student taking the course.</R16>
<description>The student or a friend of the student might want to find the outcome of their degree. They need to have a different name so that it could become distinctive on the table.</description>

<R17>The software shall allow the user to reset a year of study.</R17>
<description>When changing the course or for any other reason, the user shall be allowed to reset the year of study to make the data input easier. This will reset the year of study to 0 modules added.</description>

<R18>The software shall allow the user to delete the module and its assessments.</R18>
<description>If the student decided to drop a module, the student might want to delete the module and all it contains.</description>

<R19>The software shall allow the user to delete an assessment.</R19>
<description>The user might input data by mistake or might have added too many assessments. The user shall be allowed to delete an assessment. </description>

<R20>The software shall save changes to a database.</R20>
<description>The software shall save changes to a database either manual or automatically.</description>

2. *Non functional requirements*

<NF1>The software should not crash under any user input.</NF1>
<description>The user is asked to input lots of data. The software should be aware of the possibility of having negative numbers, instead of positive, strings instead of integers, et al.</description>

<NF2>The software should be able to recover from previous versions in case of a crash.</NF2>
<description>In case the software doesn't manage all the errors, it should recover from the previous version without having to input all the data again.</description>

<NF3>The capacity of the app should not be limited.</NF3>
<description>The user can create more than one course entry. In this case it can be used by lecturers and university staff as well.</description>

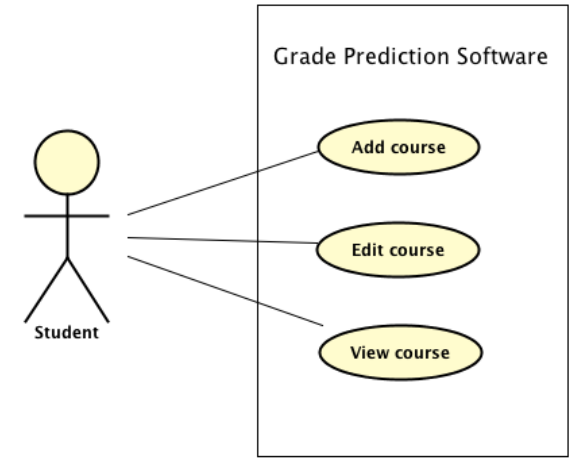
<NF4>The application should be secure.</NF4>
<description>The application holds personal information given by the user, so it should be secure in order to conform with the data protection act.</description>

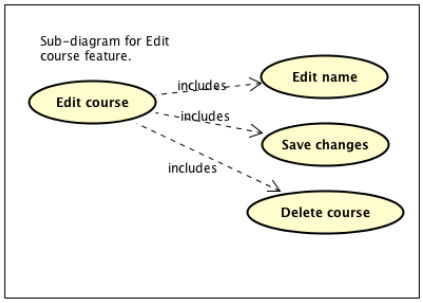
<NF5>The response time should be minimum.</NF5>
<description>The software should be optimised in order to minimize the time response.</description>

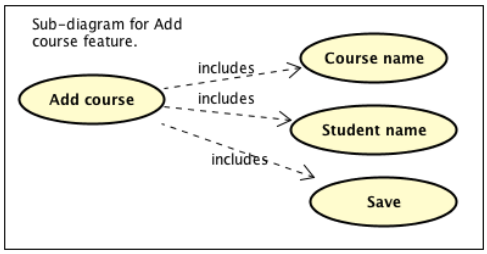
<NF6>The software should be managed easily.</NF6>
<description>The software should be easily managed by any user with no prior experience with it. Should also be intuitive.</description>

<NF7>The software should be run on a machine with internet connection.</NF7>
<description>The software should be used on a machine with internet connection to enable reading and writing from and to a database.</description>

Part B – Use Case Diagrams







Use case scenario 1

Use Case: Add course
Id: UC-001
Description: The user opens the Grade Prediction Software. There are no courses added. The user adds a course as Multimedia Computing for the Eduard student and saves it.
Level: High Level Summary
Primary actor: Student – Eduard
Supporting actors: Course leader
Stakeholders and Interests: University
Pre-conditions:

- The system needs to have a stable internet connection.
- The user needs to know the Course title and the Student's name.

Post conditions:

- In order to create a valid course entry, the user needs to input valid characters for a course and student names.
- To be able to create a valid course entry, the user needs to enter values that have not been used before in the same combination (same course and same name twice – not accepted. Same name used but on a different course – accepted).

Trigger: The user wants to create a Grade prediction for his own course.

Main success scenario

1. The user has opened the Grade prediction software.
2. Using a stable internet connection, the software has connected to the database and found that this is the first time the user has opened the software. There are no previous versions to which the software should be restored. The connection to the database has been closed.
3. The user decided to create a new course.
4. The software asks for the Course title and Student name.
5. The user added entries for both course and student names and saved.

6. The software validated the entries and saved the course entry.

7. The software connects to the database, saves the changes and closes the connection to the database.

8. The software updates the summary view.

Variations

- not required

Use case scenario 2

Use Case: Edit course
Id: UC-002
Description: The user has opened the Grade Prediction Software. There is at least one course added. The student has changed degrees, so the user wants to change the course name.
Level: High Level Summary
Primary actor: Student
Supporting actors: Course leader
Stakeholders and Interests: University
Pre-conditions:

- The system needs to have a stable internet connection.
- At least one course entry needs to be created already.
- The user needs to know the Course title.

Post conditions:

- In order to create a valid course entry, the user needs to input valid characters for a course and student names.
- To be able to create a valid course entry, the user needs to enter values that have not been used before in the same combination (same course and same name twice – not accepted. Same name used but on a different course – accepted).

Trigger: The student changed his previous course to a new one.

Main success scenario

1. The user has opened the Grade prediction software.

2. Using a stable internet connection, the software has connected to the database and found that this is not the first time the user has opened the software. There is a previous version to which the software is being restored. The connection to the database has been closed.

3. The user decided to edit the course.

4. The software asks for the Course title and Student name.

5. The user edited the entry for the course, left the student name as it was and saved.

6. The software validated the entries and saved the course entry.

7. The software connects to the database, saves the changes and closes the connection to the database.

8. The software updates the summary view.

Variations

not required

Use case scenario 3

Use Case: View course
Id: UC-003
Description: The user has opened the Grade Prediction Software. There is at least one course added. The student wants to view the course to add extra details, to check the status of their degree or to start a prediction.
Level: High Level Summary
Primary actor: Student
Supporting actors: Course leader
Stakeholders and Interests: University
Pre-conditions:

- The system needs to have a stable internet connection.
- At least one course entry needs to be created already.

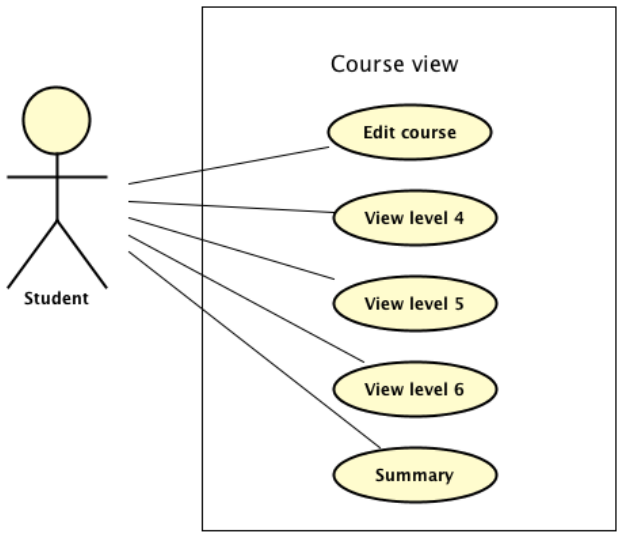
Post conditions: none
Trigger: The student wants to view the course to add extra details, to check the status of their degree or to start a prediction.

Main success scenario

1. The user has opened the Grade prediction software.
2. Using a stable internet connection, the software has connected to the database and found that this is not the first time the user has opened the software. There is a previous version to which the software is being restored. The connection to the database has been closed.
3. The user decided to view the course.
4. The software shows the course details.

Variations

not required



Use case scenario 4

Use Case: View level
Id: UC-004
Description: The student is viewing the course details. In order to add more details, to check the degree status or to make a prediction, the user will click to view one of the levels (4,5 or 6).
Level: High Level Summary
Primary actor: Student
Supporting actors: Course leader
Stakeholders and Interests: University
Pre-conditions:

- The system needs to have a stable internet connection.
- At least one course entry needs to be created already.

Post conditions: none
Trigger: The student wants to view a particular level (4,5 or 6) to add extra details, to check the status of their degree or to start a prediction.

Main success scenario

1. The software shows the course details.
2. The user selected a level to view.
3. The software shows the level details.

Variations

not required

Use case scenario 5

Use Case: Summary
Id: UC-005
Description: The student is viewing the course summary. This means he can see everything that the course holds: assessments with names and weights under modules, modules with names and assessments under levels, levels with names and modules under course name, average per level, average per module, total course outcome.
Level: High Level Summary
Primary actor: Student
Supporting actors: Course leader
Stakeholders and Interests: University
Pre-conditions:

- The system needs to have a stable internet connection.
- At least one course entry needs to be created already.

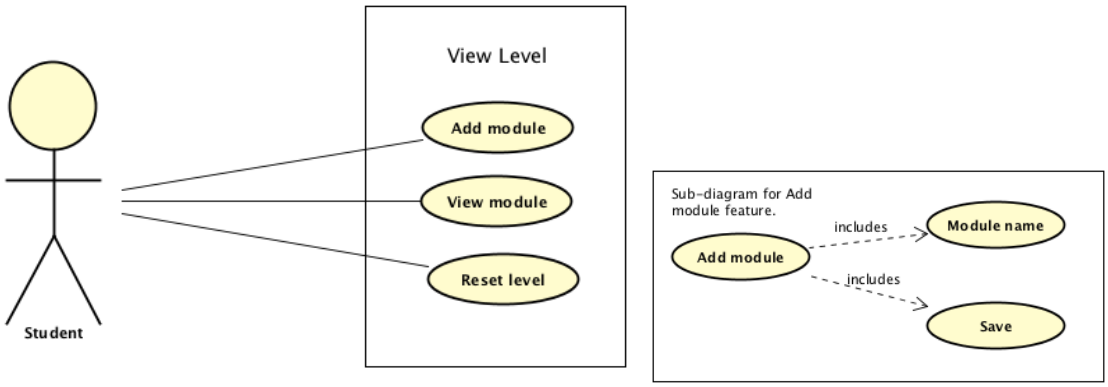
Post conditions: none
Trigger: The student wants to view a the summary of the course in order to check the status of their degree or to check the prediction.

Main success scenario

1. The software shows the course details.
2. The user selected clicked on the summary button.
3. The software shows the summary view.

Variations

not required



Use case scenario 6

Use Case: Add module
Id: UC-006
Description: The student is viewing the level details. The user wants to add a new module.
Level: High Level Summary
Primary actor: Student
Supporting actors: Module leader
Stakeholders and Interests: University
Pre-conditions:

- The system needs to have a stable internet connection.
- The user needs to know the module title.

Post conditions:

- In order to create a valid module entry, the user needs to input valid characters for a module name.

Trigger: The user wants to create a new module in order to build the degree and get an outcome or prediction.

Main success scenario

1. The student is viewing the level details.
2. The user decided to create a new module.
3. The software asks for the module title.
4. The user added an entry for module name and saved.
5. The software validated the entry and saved the module entry.
6. The software connects to the database, saves the changes and closes the connection to the database.
7. The software updates the summary view.

Variations

- not required

Use case scenario 7

Use Case: View module
Id: UC-007
Description: The student is viewing the level details. The user wants to view a module details.
Level: High Level Summary
Primary actor: Student
Supporting actors: Module leader
Stakeholders and Interests: University
Pre-conditions:

- The system needs to have a stable internet connection.

Post conditions: none
Trigger: The user wants to view a module details.

Main success scenario

1. The student is viewing the level details.
2. The user decided to view a particular module.
3. The software shows the module details.

Variations

- not required

Use case scenario 8

Use Case: Reset module
Id: UC-008
Description: The student is viewing the level details. The user wants to reset the level and all its modules to make it easy to edit a course when the student changes degrees.
Level: High Level Summary
Primary actor: Student
Supporting actors: Course leader
Stakeholders and Interests: University
Pre-conditions:

- The system needs to have a stable internet connection.

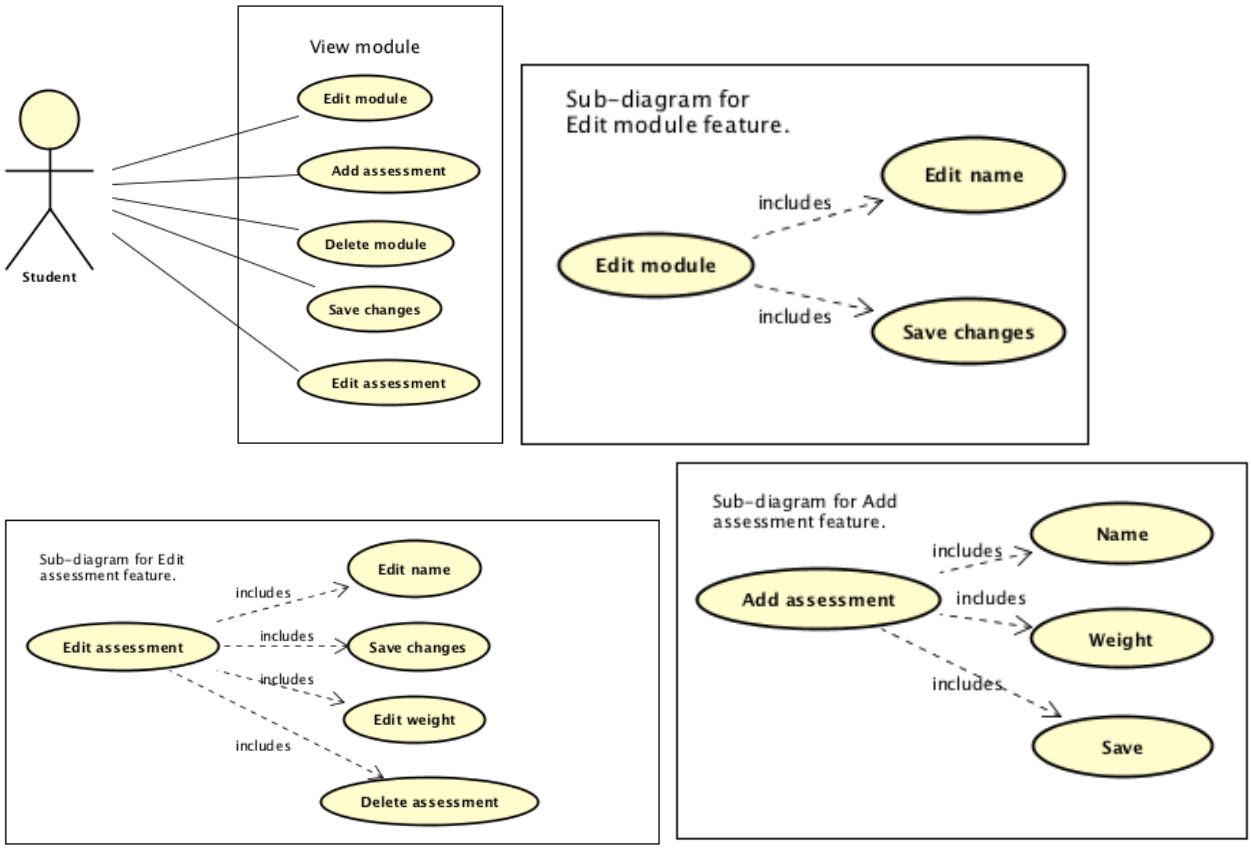
Post conditions: none
Trigger: The user wants to reset the level and all its modules.

Main success scenario

1. The student is viewing the level details.
2. The user decided to reset the level with all its modules.
3. The software asks for if the user is sure.
4. The user confirms the reset of that particular level.
5. The software resets the level (the modules are being dropped with all assessments connected to them) and saves the changes.
6. The software connects to the database, saves the changes and closes the connection to the database.
7. The software updates the summary view.

Variations

- not required



Use case scenario 9

Use Case: Edit module
Id: UC-009
Description: The student has requested to change an optional module with another. The user will change the name of that certain module. This feature can also be used when a module title is mistyped.
Level: High Level Summary
Primary actor: Student
Supporting actors: Course leader
Stakeholders and Interests: University
Pre-conditions:

- The system needs to have a stable internet connection.
- At least one course entry needs to be created already.
- At least one module entry needs to be created already.
- The user needs to know the new module title.

Post conditions:

- In order to create a valid module entry, the user needs to input valid characters for a module name.

Trigger: The student has requested to change an optional module with another or the student wants to correct mistyping errors.

Main success scenario

1. The user decided to edit the module.
2. The software asks for the module title.
3. The user edited the entry for the module and saved.
4. The software validated the entry and saved the module.
5. The software connects to the database, saves the changes and closes the connection to the database.

6. The software updates the summary view.

Variations

- not required

Use case scenario 10

Use Case: Add assessment
Id: UC-010
Description: The student is viewing the module details. The user wants to add a new assessment.
Level: High Level Summary
Primary actor: Student
Supporting actors: Module leader
Stakeholders and Interests: University
Pre-conditions:

- The system needs to have a stable internet connection.
- The user needs to know the assessment details.

Post conditions:

- In order to create a valid assessment entry, the user needs to input valid characters for an assessment name and weight.

Trigger: The user wants to create a new assessment in order to build the module and get an outcome or prediction.

Main success scenario

1. The student is viewing the module details.
2. The user decided to create a new assessment.
3. The software asks for the assessment name and weight.
4. The user added an entry for assessment name, weight and saved.
5. The software validated the entry and saved the assessment entry.
6. The software connects to the database, saves the changes and closes the connection to the database.
7. The software updates the summary view.

Variations

- not required

Use case scenario 11

Use Case: Delete module
Id: UC-011
Description: The student is viewing the module details. The user wants to delete the module in order to add a new one in its place. In order for this to work, the user has to save the changes.
Level: High Level Summary
Primary actor: Student
Supporting actors: Module leader
Stakeholders and Interests: University
Pre-conditions:

- The system needs to have a stable internet connection.

Post conditions: none
Trigger: The user wants to delete the module in order to add a new one in its place.

Main success scenario

1. The student is viewing the module details.
2. The user decided to delete the module.
3. The software marks the module to be deleted.
4. The user saves the changes.
5. The software deletes the module and all the assessments that it holds.
6. The software connects to the database, saves the changes and closes the connection to the database.
7. The software updates the summary view.

Variations

- not required

Use case scenario 12

Use Case: Edit assessment
Id: UC-012
Description: The student is viewing the module details. The user wants to edit one of the existing assessments for the prediction reasons or for correcting mistypes.
Level: High Level Summary
Primary actor: Student
Supporting actors: Module leader
Stakeholders and Interests: University
Pre-conditions:

- The system needs to have a stable internet connection.
- The user needs to know the assessment details.

Post conditions:

- In order to create a valid assessment entry, the user needs to input valid characters for an assessment name and weight.

Trigger: The user wants to edit one of the existing assessments for the prediction reasons or for correcting mistypes.

Main success scenario

1. The student is viewing the module details.
2. The user decided to edit an existing assessment.
3. The software asks for the assessment name and weight.
4. The user added an entry for assessment name, weight and saved.
5. The software validated the entry and saved the assessment entry.
6. The software connects to the database, saves the changes and closes the connection to the database.
7. The software updates the summary view.

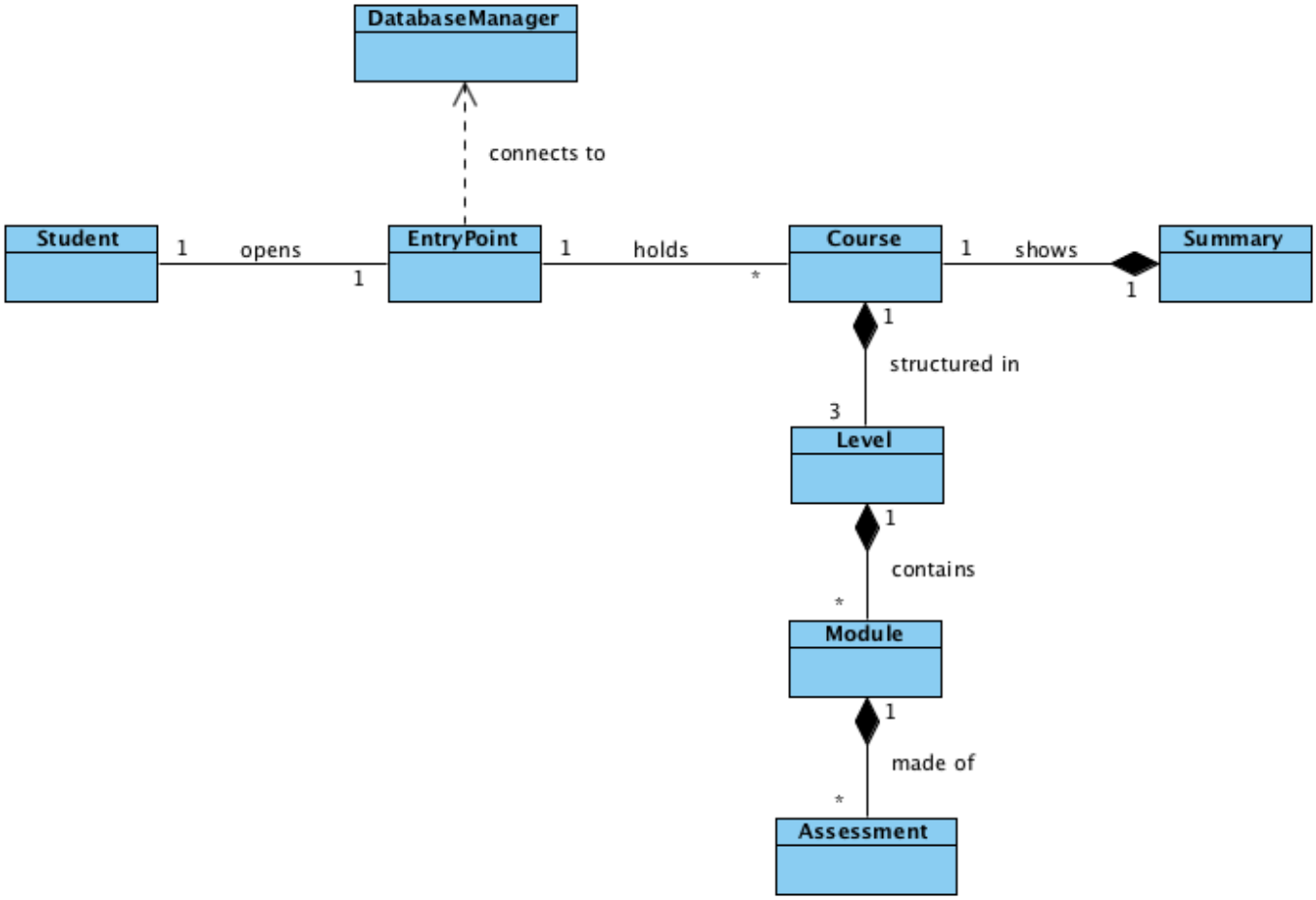
Variations

- not required

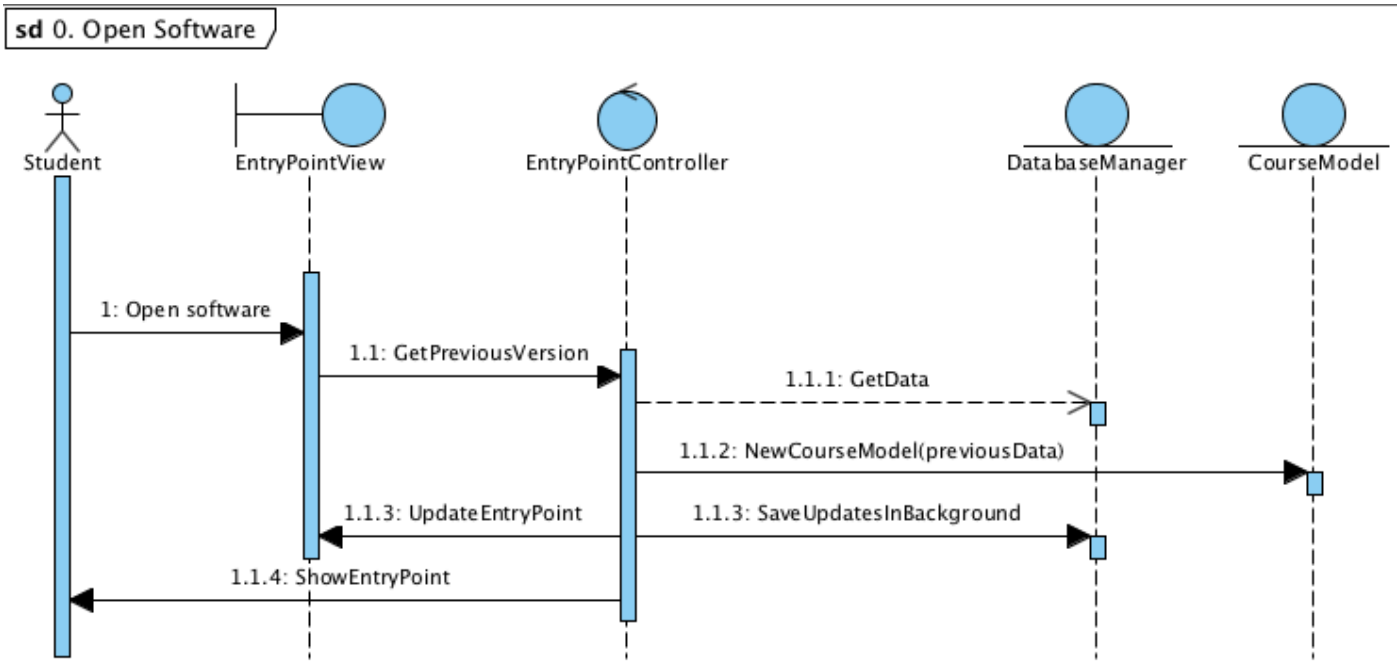
Part C – Classes

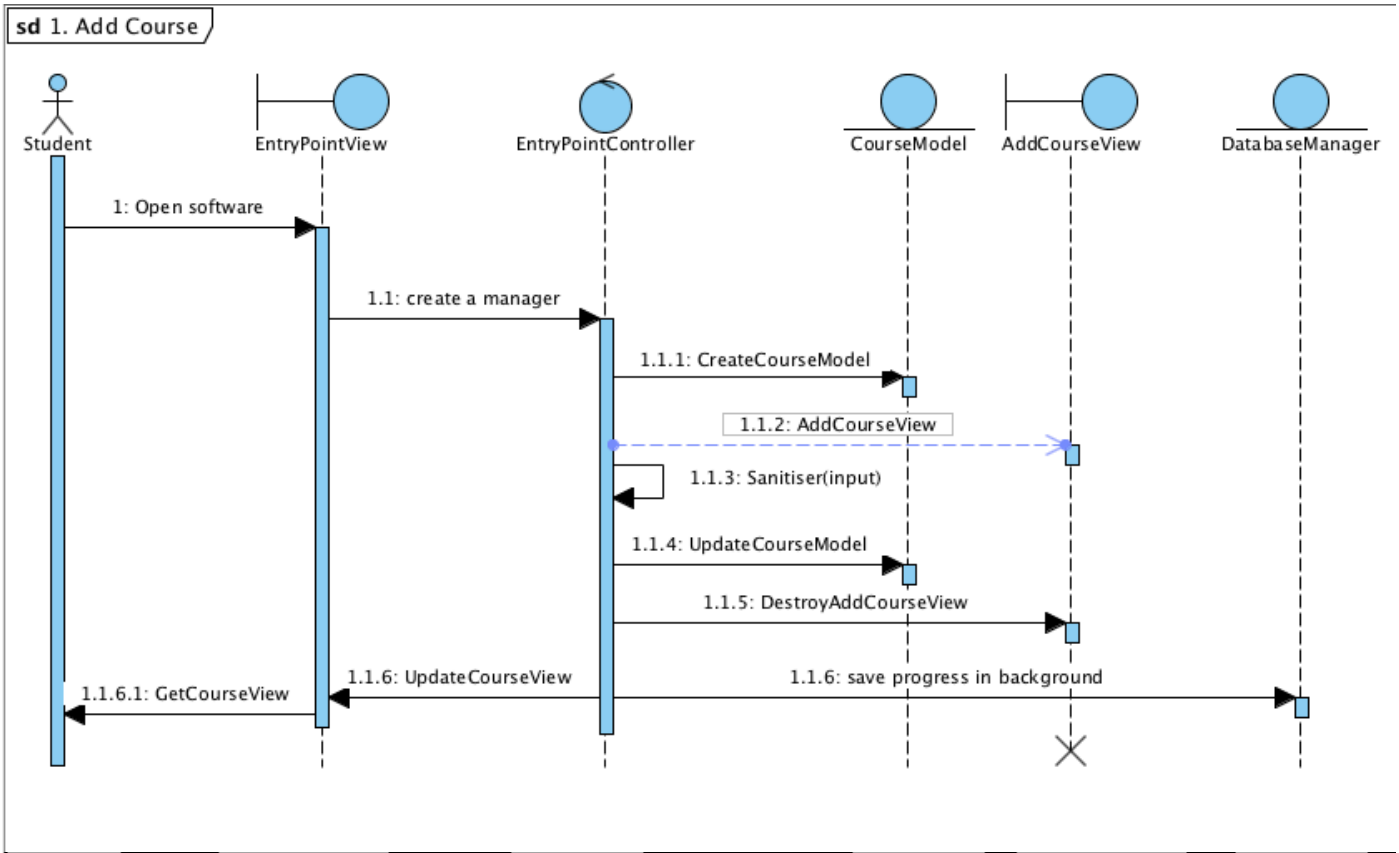
Class name	Type	Responsibility	Collaborations
1. EntryPointView	Form (View)	First view of the application. Holds all the courses.	EntryPointController, CourseController, Student
2. CourseView	Form (View)	Form opened when the user wants to view a course. Has labels with all 3 study levels with an average for each of them. Holds the final degree outcome.	EntryPointController, CourseController, LevelController, Student
3. LevelView	Form (View)	Form opened when the user wants to view a level. Has labels with all the modules a level has. Each module has an average. Holds the total average of the module.	EntryPointController, LevelController, ModuleController, Student
4. ModuleView	Form (View)	Form opened when the user wants to view a module. Has labels with all the assessments a module has with marks and weights. Holds the module average.	ModuleController, Student
5. AddCourseView	Form (View)	Form opened when the user wants to add a course. Has course name and student name fields.	EntryPointController, Student
6. AddModuleView	Form (View)	Form opened when the user wants to add a module. Has a module name as a field..	LevelController, Student
7. AddAssessmentView	Form (View)	Form opened when the user wants to add an assessment. Has an assessment name and weight as fields.	ModuleController, Student
8. EditCourseView	Form (View)	Form opened when the user wants to edit a course. Has the same fields as the course view.	CourseController, Student
9. EditModuleView	Form (View)	Form opened when the user wants to edit a module. Has the same fields as the module view.	ModuleController, Student
10. EditAssessmentView	Form (View)	Form opened when the user wants to edit an assessment. Has the same fields as the assessment view.	ModuleController, EditAssessmentController, Student
11. SummaryView	Form (View)	Form opened when the user wants to see the summary of the course in order to check the status of their degree or its prediction.	CourseController
12. CourseModel	Model	Model for the course. Keeps the data for each course.	EntryPointController, CourseController

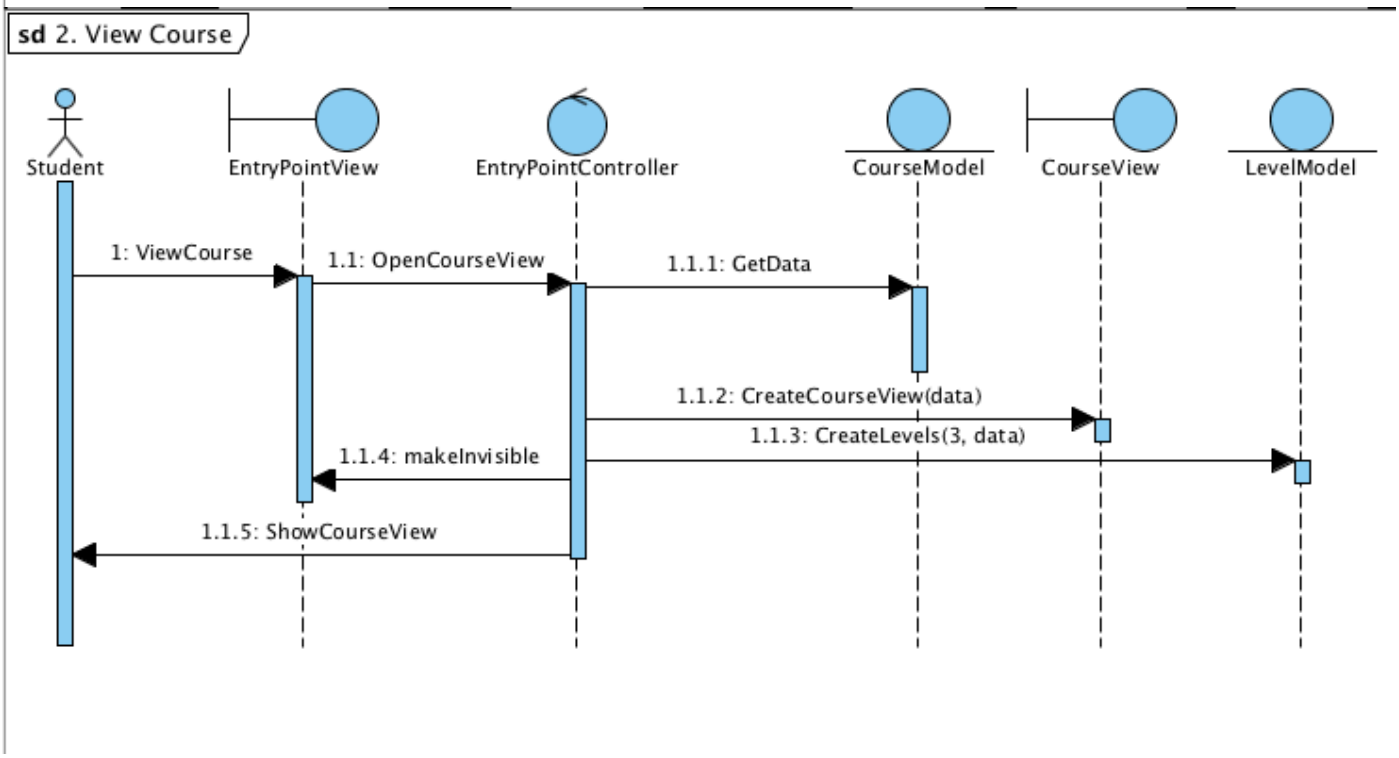
13. LevelModel	Model	Model for the level. Keeps the data for each level.	LevelController
14. ModuleModel	Model	Model for the module. Keeps the data for each module.	LevelController, ModuleController
15. AssessmentModel	Model	Model for the assessment. Keeps the data for each assessment.	ModuleController
16. EntryPointController	Controller	Controller for the first view of the application. Makes everything work properly in the first view of the software.	Student, EntryPointView, CourseView, LevelView, AddCourseView, CourseModel, DatabaseManager, Sanitiser
17. CourseController	Controller	Controller for the course. Makes everything work properly in the course view.	Student, EntryPointView, CourseView, EditCourseView, CourseModel, DatabaseManager, Sanitiser
18. LevelController	Controller	Controller for the level. Makes everything work properly in the level view.	Student, CourseView, LevelView, AddModuleView, LevelModel, ModuleModel, DatabaseManager, Sanitiser
19. ModuleController	Controller	Controller for the module. Makes everything work properly in the module view.	Student, ModuleView, LevelView, AddAssessmentView, EditModuleView, EditAssessmentView, ModuleModel, AssessmentModel, DatabaseManager, Sanitiser
20. EditAssessmentController	Controller	Controller when editing an assessment. Makes everything work properly in the edit assessment view.	EditAssessmentView
21. DatabaseManager	Model (can be controller as well)	Entity for a database manager that will get, save, update, delete data from a database.	EntryPointController, CourseController, LevelController, ModuleController
22. Sanitiser	Helper	This class will make sure there all the characters and integers are valid before using them in the software. Will help the controller do a better job.	EntryPointController, CourseController, LevelController, ModuleController

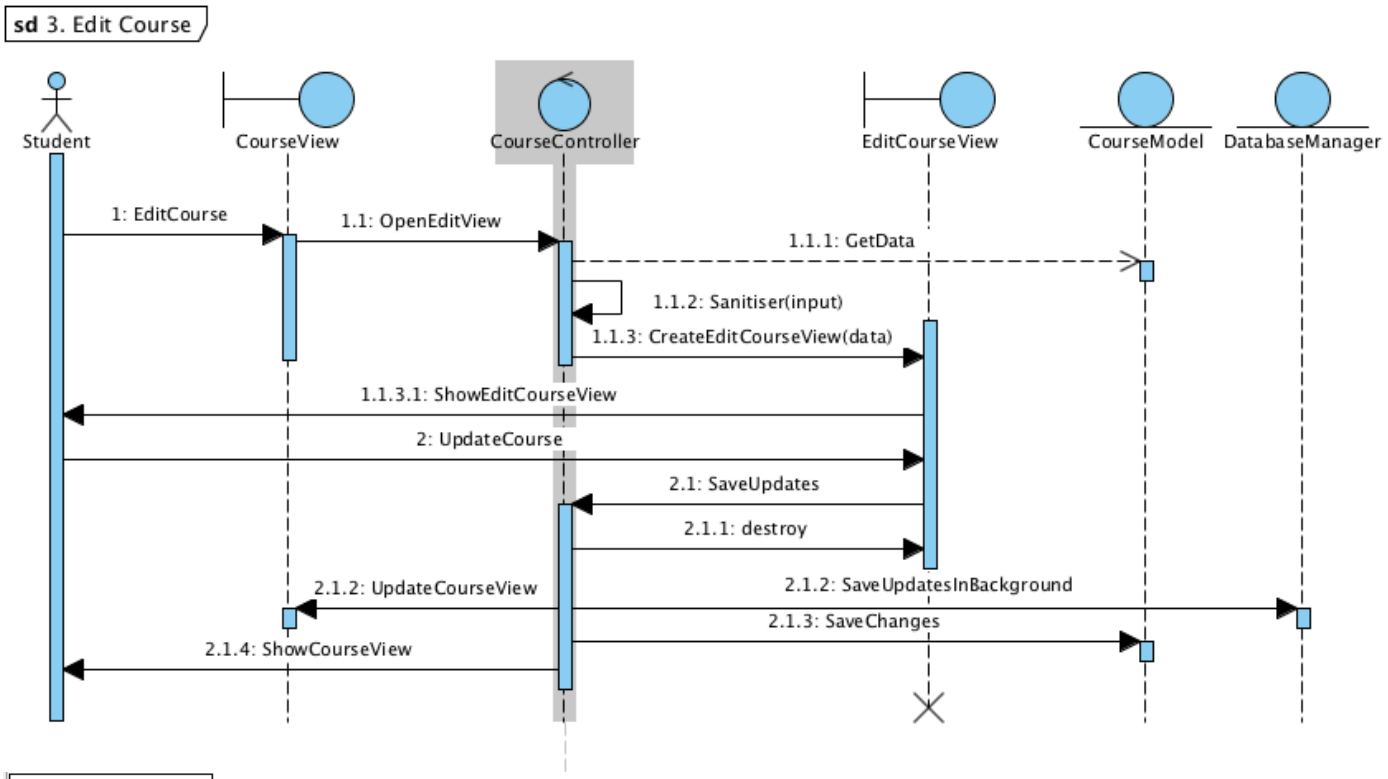


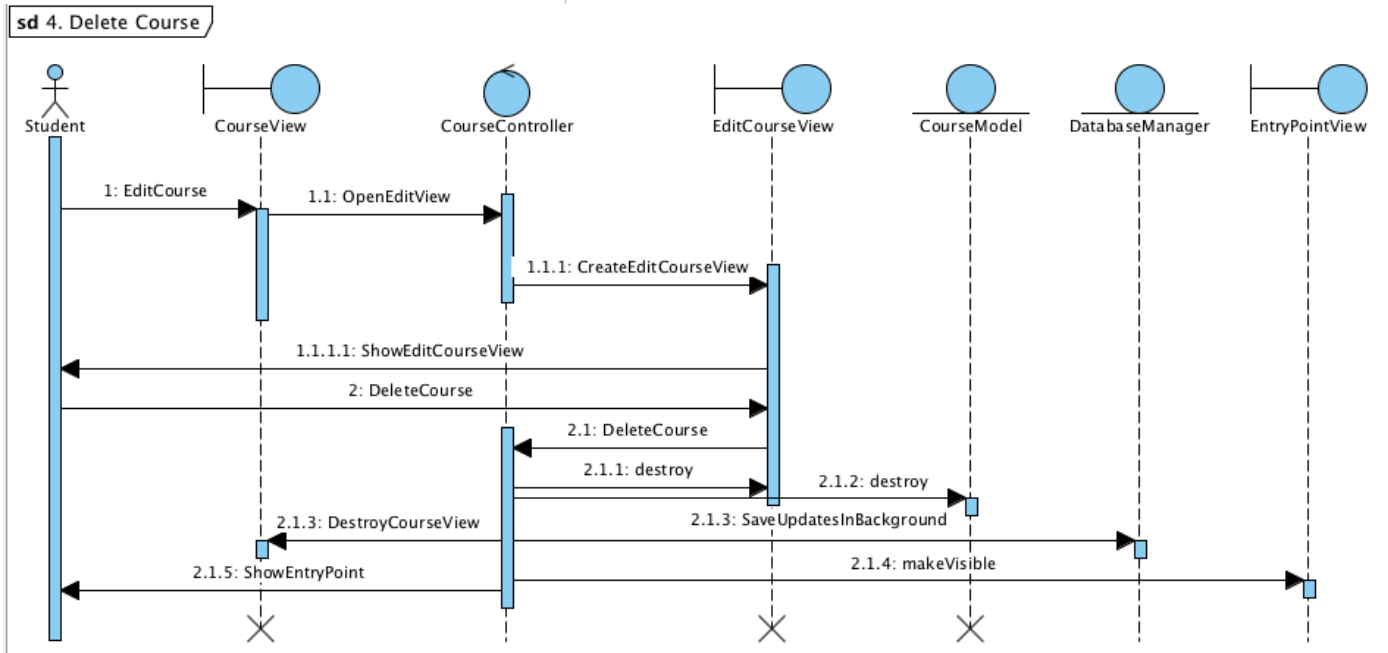
Part D – Collaboration

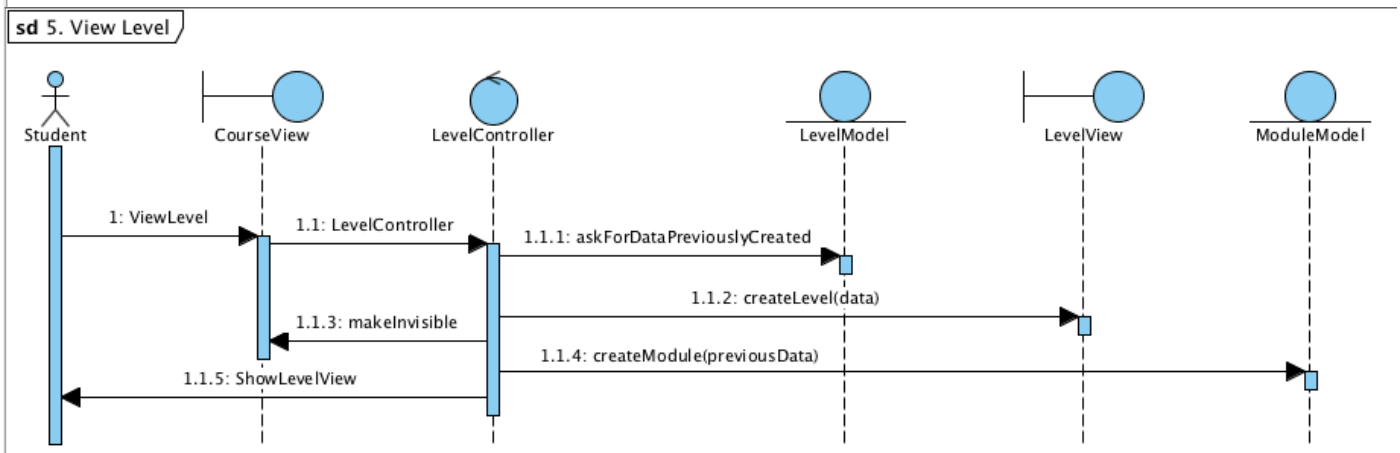


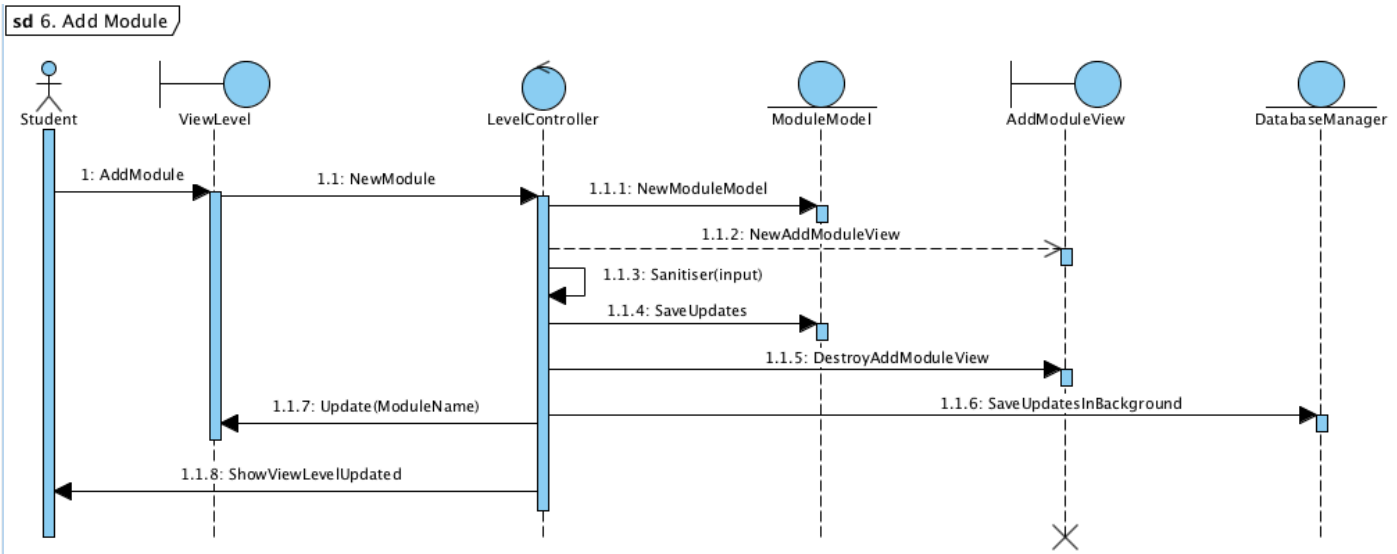

sd 1. Add Course
Student
EntryPointView
EntryPointController
CourseModel
AddCourseView
DatabaseManager
1: Open software
1.1: create a manager
1.1.1: CreateCourseModel
1.1.2: AddCourseView
1.1.3: Sanitiser(input)
1.1.4: UpdateCourseModel
1.1.5: DestroyAddCourseView
1.1.6.1: GetCourseView
1.1.6: UpdateCourseView
1.1.6: save progress in background

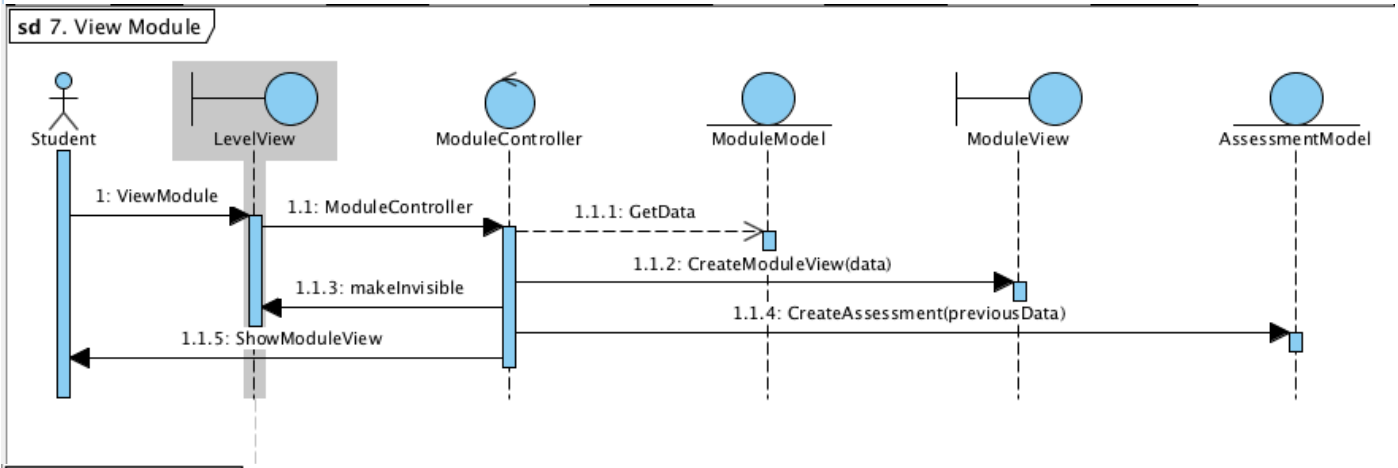

sd 2. View Course
Student
EntryPointView
EntryPointController
CourseModel
CourseView
LevelModel
1: ViewCourse
1.1: OpenCourseView
1.1.1: GetData
1.1.2: CreateCourseView(data)
1.1.3: CreateLevels(3, data)
1.1.4: makeInvisible
1.1.5: ShowCourseView

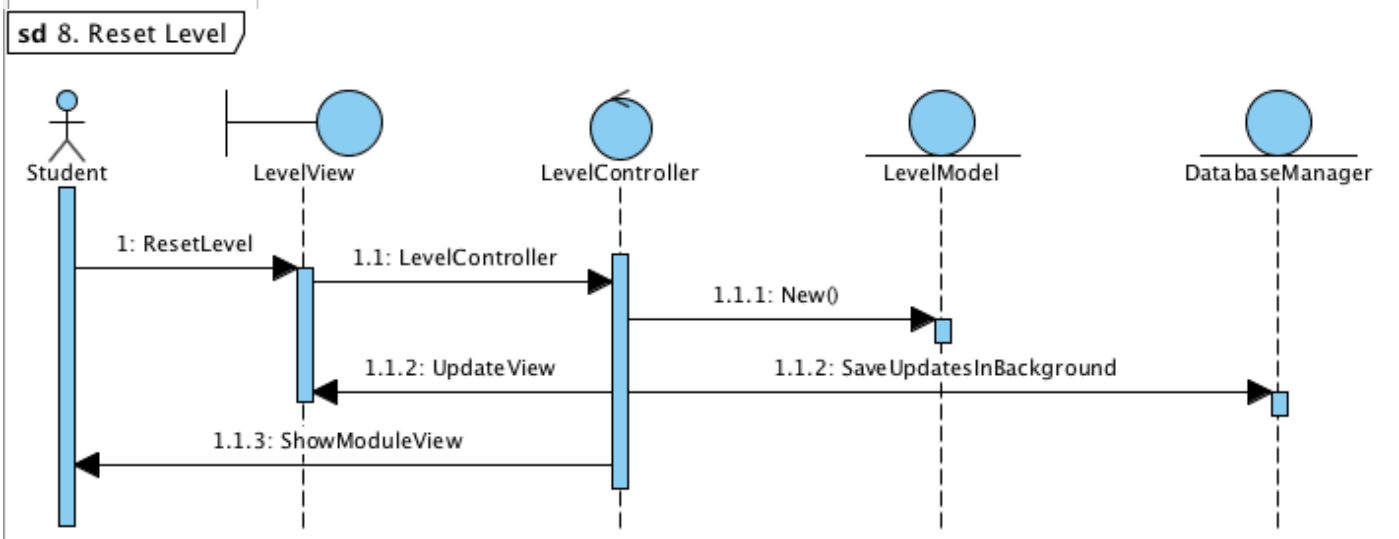

sd 3. Edit Course
Student
CourseView
CourseController
EditCourseView
CourseModel
DatabaseManager
1: EditCourse
1.1: OpenEditView
1.1.1: GetData
1.1.2: Sanitiser(input)
1.1.3: CreateEditCourseView(data)
1.1.3.1: ShowEditCourseView
2: UpdateCourse
2.1: SaveUpdates
2.1.1: destroy
2.1.2: UpdateCourseView
2.1.2: SaveUpdatesInBackground
2.1.3: SaveChanges
2.1.4: ShowCourseView

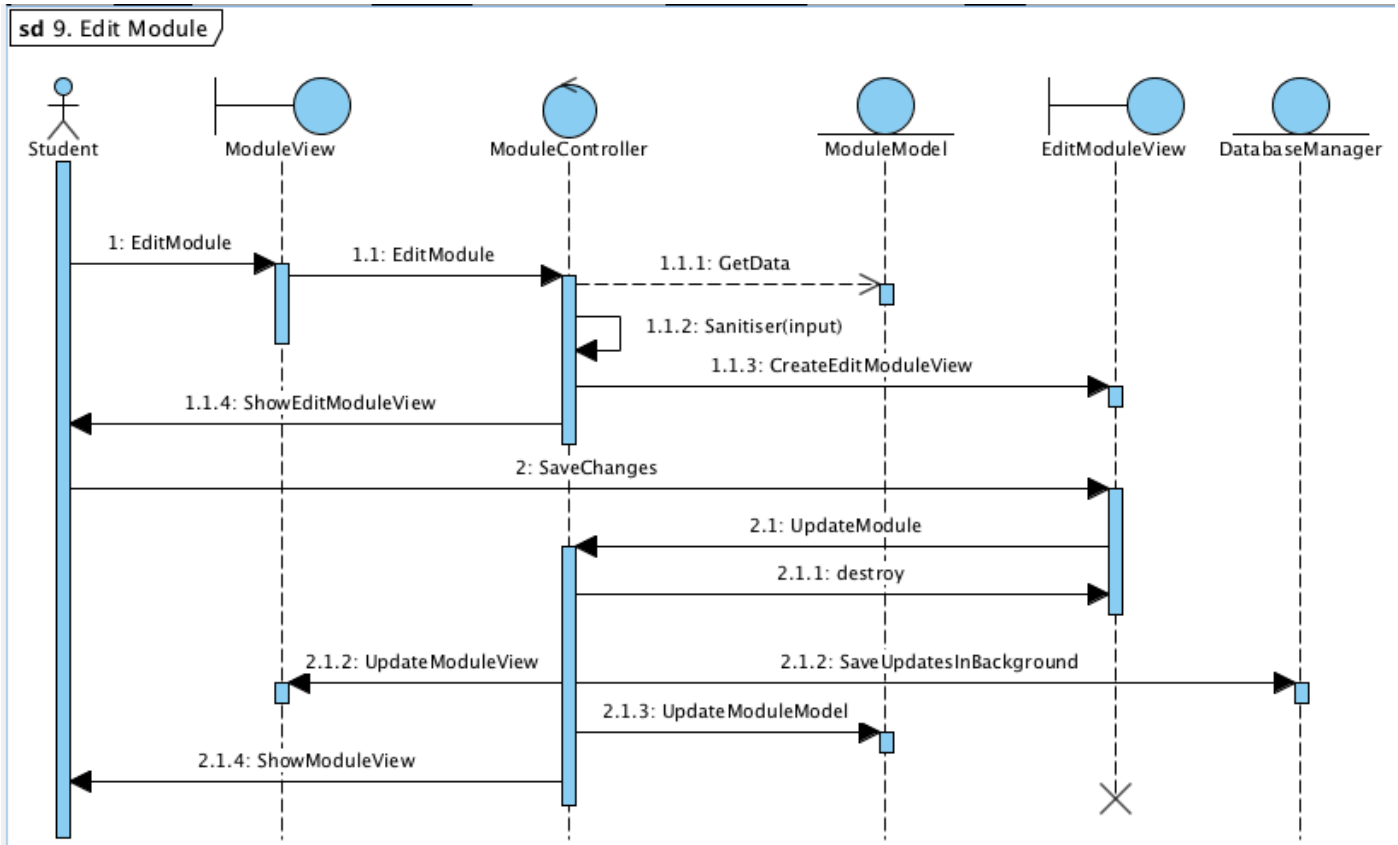

sd 4. Delete Course
Student
CourseView
CourseController
EditCourseView
CourseModel
DatabaseManager
EntryPointView
1: EditCourse
1.1: OpenEditView
1.1.1: CreateEditCourseView
1.1.1.1: ShowEditCourseView
2: DeleteCourse
2.1: DeleteCourse
2.1.1: destroy
2.1.2: destroy
2.1.3: DestroyCourseView
2.1.3: SaveUpdatesInBackground
2.1.4: makeVisible
2.1.5: ShowEntryPoint

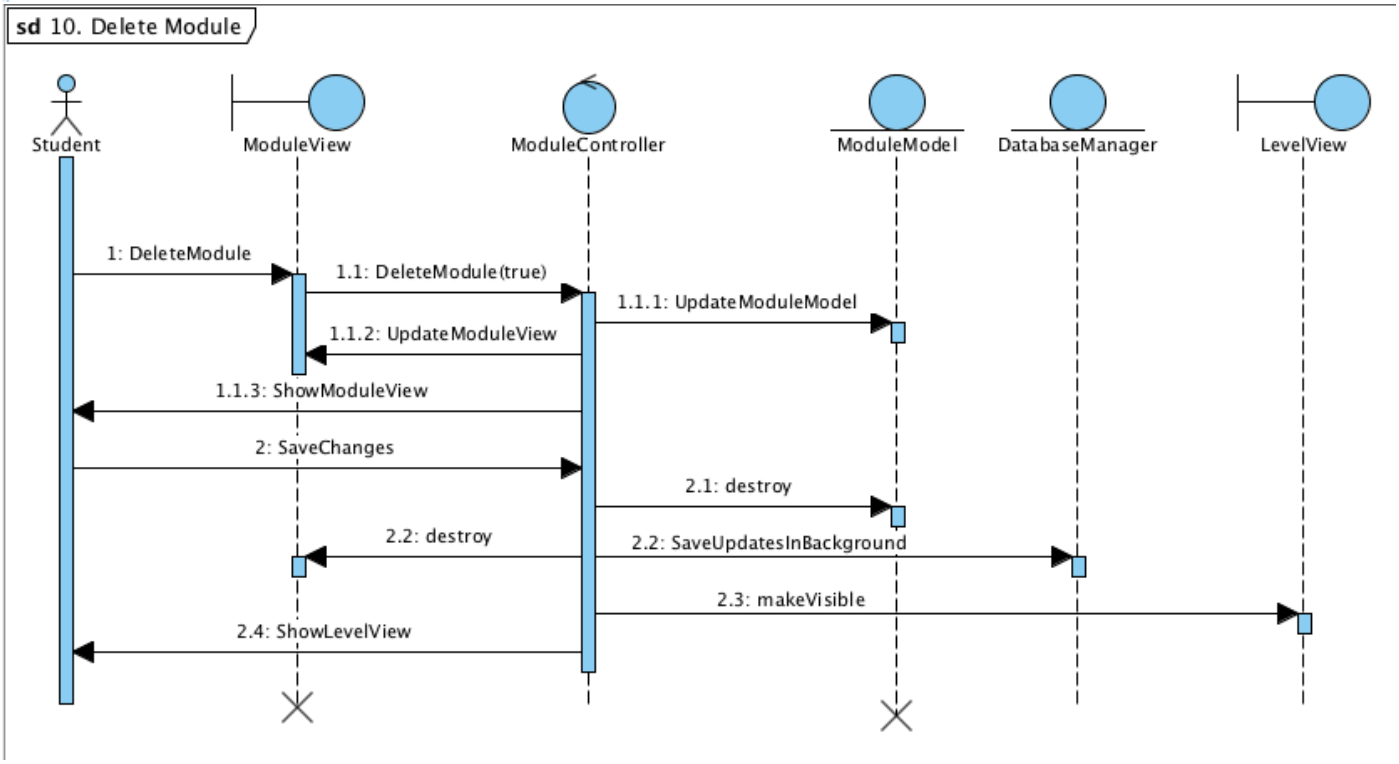

sd 5. View Level
Student
CourseView
LevelController
LevelModel
LevelView
ModuleModel
1: ViewLevel
1.1: LevelController
1.1.1: askForDataPreviouslyCreated
1.1.2: createLevel(data)
1.1.3: makeInvisible
1.1.4: createModule(previousData)
1.1.5: ShowLevelView

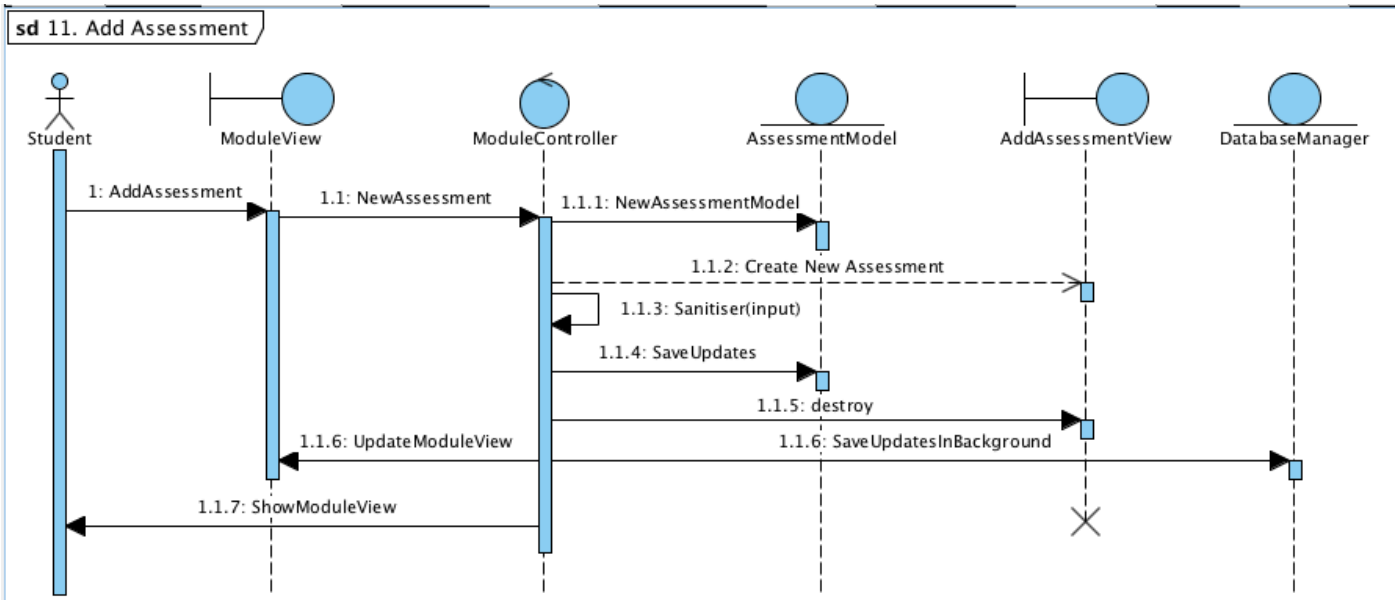

sd 6. Add Module
Student
ViewLevel
LevelController
ModuleModel
AddModuleView
DatabaseManager
1: AddModule
1.1: NewModule
1.1.1: NewModuleModel
1.1.2: NewAddModuleView
1.1.3: Sanitiser(input)
1.1.4: SaveUpdates
1.1.5: DestroyAddModuleView
1.1.7: Update(ModuleName)
1.1.6: SaveUpdatesInBackground
1.1.8: ShowViewLevelUpdated

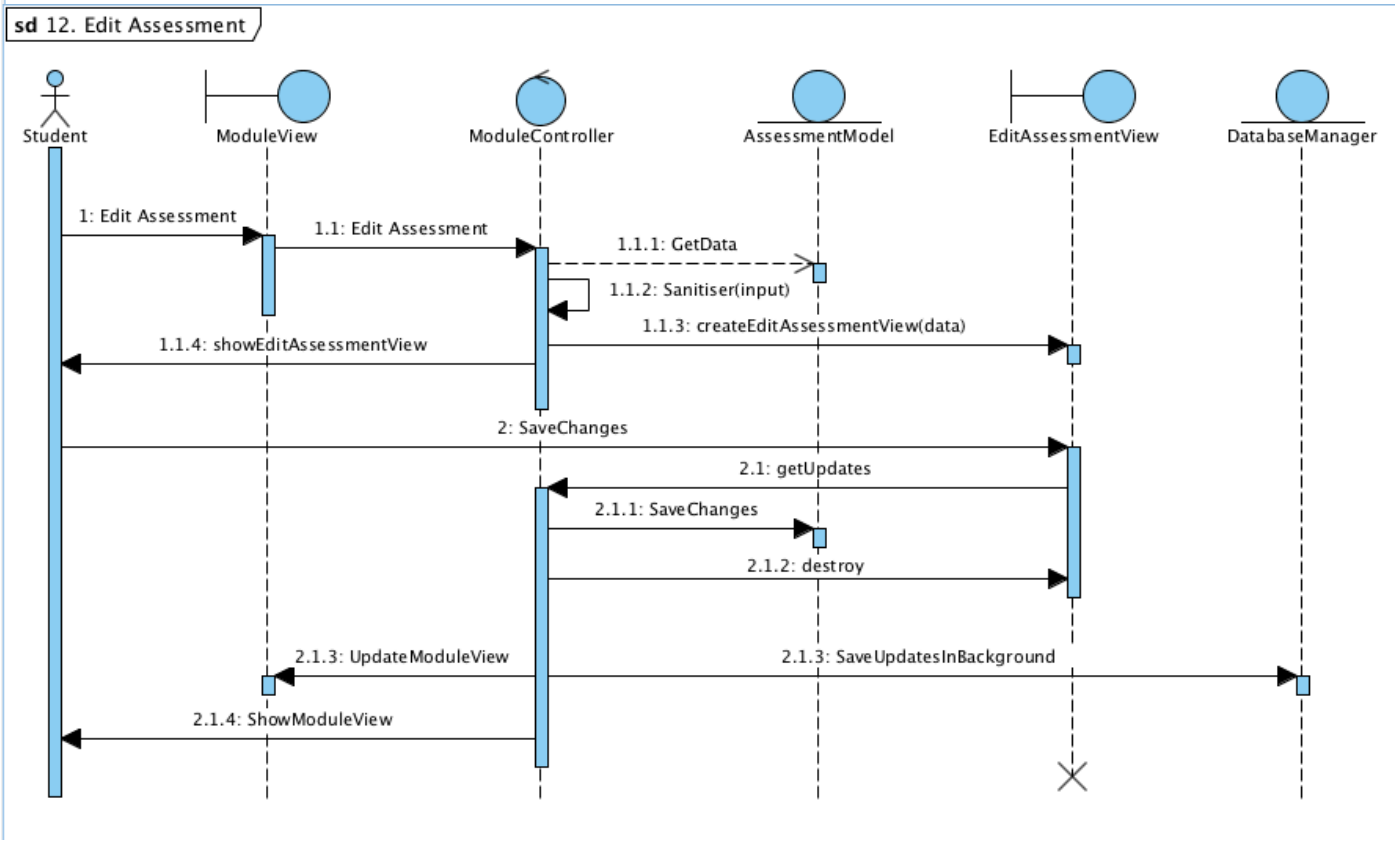

sd 7. View Module
Student
LevelView
ModuleController
ModuleModel
ModuleView
AssessmentModel
1: ViewModule
1.1: ModuleController
1.1.1: GetData
1.1.2: CreateModuleView(data)
1.1.3: makeInvisible
1.1.4: CreateAssessment(previousData)
1.1.5: ShowModuleView

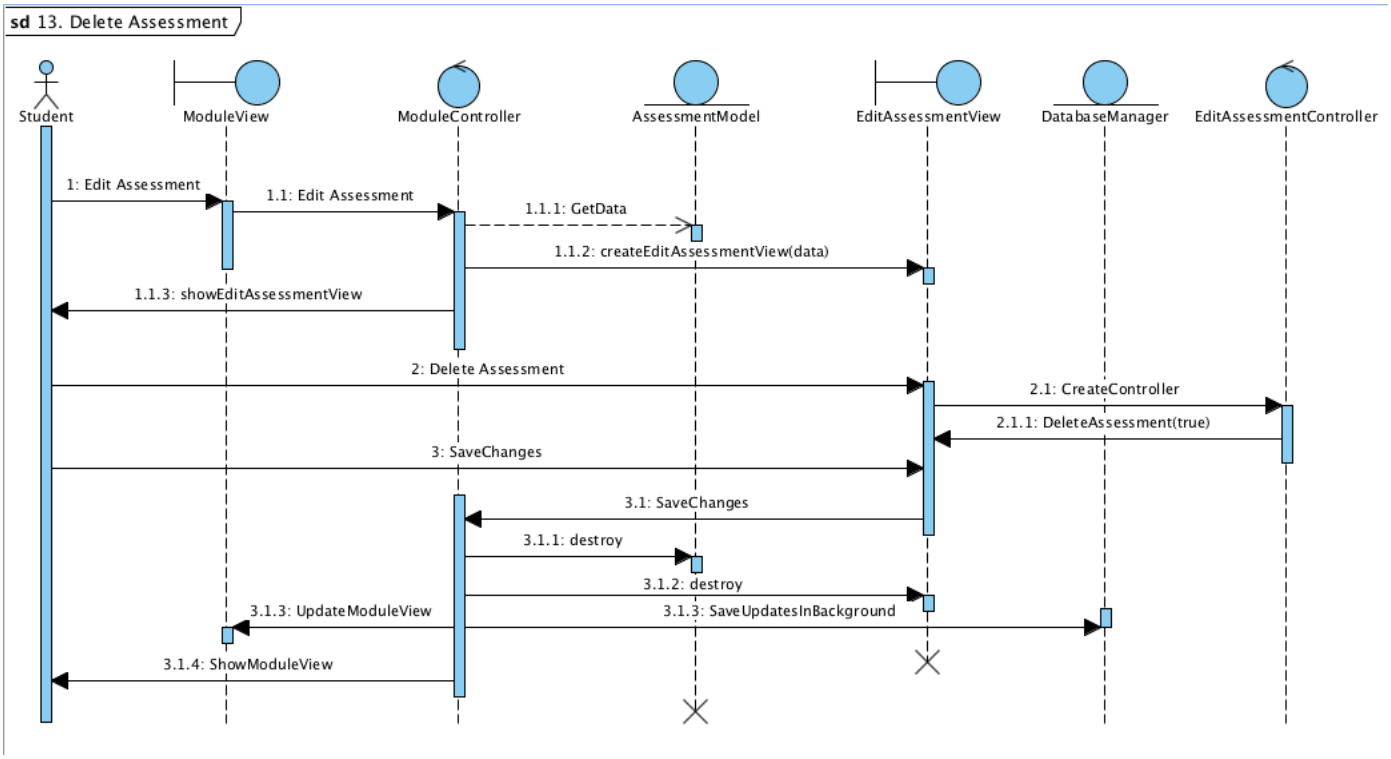

sd 8. Reset Level
Student
LevelView
LevelController
LevelModel
DatabaseManager
1: ResetLevel
1.1: LevelController
1.1.1: New()
1.1.2: UpdateView
1.1.2: SaveUpdatesInBackground
1.1.3: ShowModuleView

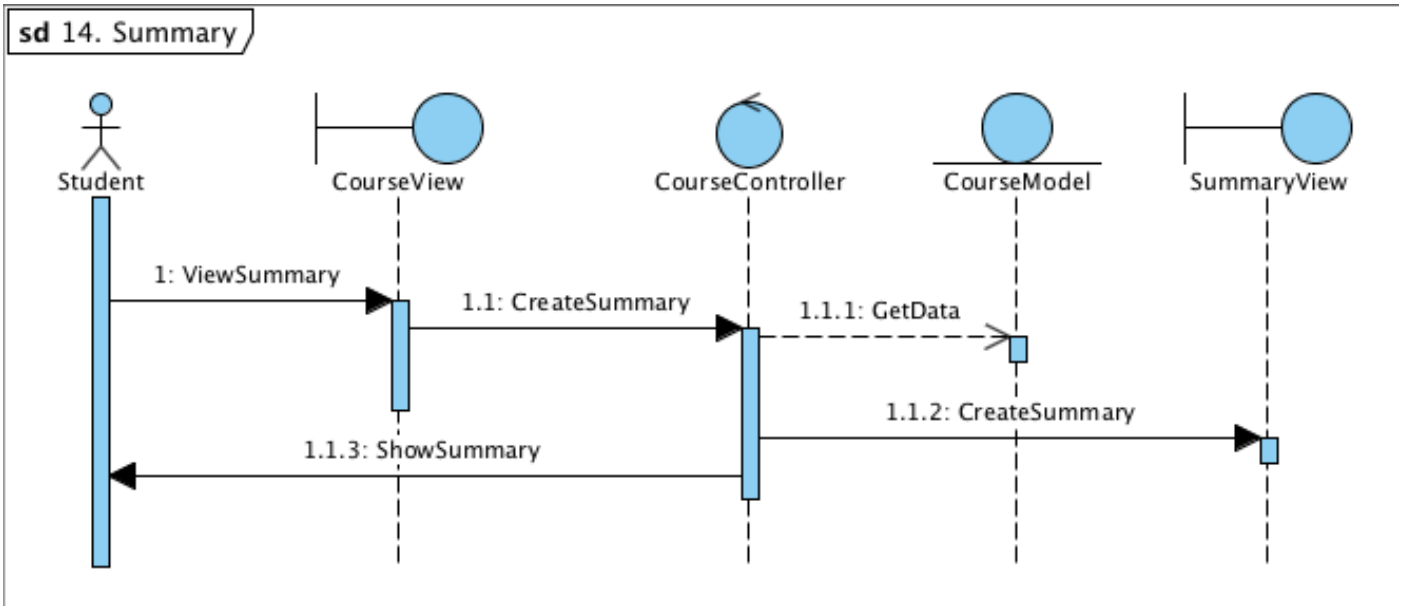

sd 9. Edit Module
Student
ModuleView
ModuleController
ModuleModel
EditModuleView
DatabaseManager
1: EditModule
1.1: EditModule
1.1.1: GetData
1.1.2: Sanitiser(input)
1.1.3: CreateEditModuleView
1.1.4: ShowEditModuleView
2: SaveChanges
2.1: UpdateModule
2.1.1: destroy
2.1.2: UpdateModuleView
2.1.2: SaveUpdatesInBackground
2.1.3: UpdateModuleModel
2.1.4: ShowModuleView


sd 10. Delete Module
Student
ModuleView
ModuleController
ModuleModel
DatabaseManager
LevelView
1: DeleteModule
1.1: DeleteModule(true)
1.1.1: UpdateModuleModel
1.1.2: UpdateModuleView
1.1.3: ShowModuleView
2: SaveChanges
2.1: destroy
2.2: destroy
2.2: SaveUpdatesInBackground
2.3: makeVisible
2.4: ShowLevelView


sd 11. Add Assessment
Student
ModuleView
ModuleController
AssessmentModel
AddAssessmentView
DatabaseManager
1: AddAssessment
1.1: NewAssessment
1.1.1: NewAssessmentModel
1.1.2: Create New Assessment
1.1.3: Sanitiser(input)
1.1.4: SaveUpdates
1.1.5: destroy
1.1.6: UpdateModuleView
1.1.6: SaveUpdatesInBackground
1.1.7: ShowModuleView


sd 12. Edit Assessment
Student
ModuleView
ModuleController
AssessmentModel
EditAssessmentView
DatabaseManager
1: Edit Assessment
1.1: Edit Assessment
1.1.1: GetData
1.1.2: Sanitiser(input)
1.1.3: createEditAssessmentView(data)
1.1.4: showEditAssessmentView
2: SaveChanges
2.1: getUpdates
2.1.1: SaveChanges
2.1.2: destroy
2.1.3: UpdateModuleView
2.1.3: SaveUpdatesInBackground
2.1.4: ShowModuleView


sd 13. Delete Assessment
Student
ModuleView
ModuleController
AssessmentModel
EditAssessmentView
DatabaseManager
EditAssessmentController
1: Edit Assessment
1.1: Edit Assessment
1.1.1: GetData
1.1.2: createEditAssessmentView(data)
1.1.3: showEditAssessmentView
2: Delete Assessment
2.1: CreateController
2.1.1: DeleteAssessment(true)
3: SaveChanges
3.1: SaveChanges
3.1.1: destroy
3.1.2: destroy
3.1.3: UpdateModuleView
3.1.3: SaveUpdatesInBackground
3.1.4: ShowModuleView


sd 14. Summary
Student
CourseView
CourseController
CourseModel
SummaryView
1: ViewSummary
1.1: CreateSummary
1.1.1: GetData
1.1.2: CreateSummary
1.1.3: ShowSummary

Part E – Activity