SOFTWARE ENGINEERING PROJECT

GOLDEN OLDIES

INC.

AW Schoombie – 2011130198
WR Boltman – 2011107444
JC Venter - 2017063245
DC McDonald – 2018035978
ML Gomes – 2018315255

Table of Contents

Table of Figures

Project Proposal

Client Feedback

After a thorough review of the system requested, all features were deemed achievable and deliverable in a reasonable time frame, and within a reasonable budget.

However, a potential problem identified in the program is **redundancy**. Separate sections were listed that can instead be combined to both better the user experience and reduce the development time for creating separate sections.

The ‘photos, miscellaneous’ section can be combined with the ‘record audio’ and the ‘record video’ sections to reduce redundancy per module. These can be combined as simply an ‘upload’ section under the **notes** of each module.

The sections of ‘assignment vault’, ‘test vault’, and ‘exam vault’ can be combined as a subsection inside a single ‘vault’ category to better the user experience with less clutter.

Thus, the new layout per module will be as follow:

- Notes
  - Text notes
  - Uploads
- Vault
  - Assignment vault
  - Test vault
  - Exam vault
- Marks

Students will still be able to organise the material in the ‘**notes**’ section.

To further reduce redundancy, the ‘to-do list’ can be combined with the ‘diary’ section as a single ‘calendar’ feature. The execution and implementation of this will be discussed in more detail in question 2.

Additional Features

After a full review of the project description, we have identified a few additional features that we believe would improve the quality of the application.

Calendar:

Users will be able to add **To-Do**'s, **schedule upcoming tests**, and **assignment due dates** on the calendar. This will be a massive help to keep their studies organized and stay on top of what is happening and when it is happening.

Address book:

Having an address book for each module, where students can *save* **lecturer contact information**, as well as **fellow student contact information** will be a useful addition. Having all your lecturers' contact information accessible in a central location (this application) makes it easier to find the information and quicker. And having fellow classmates' information is useful if you ever missed a class and need to get notes, want to organize a group study session, need to make groups for an assignment, etc.

Media Upload and Playback:

Instead of being able to just record audio and video from inside the application, users will also be able to upload existing video, audio recordings and pictures that are on the device. Users will also be able to playback the media from within the application and not have to open an external application to view it.

Digital Flashcards:

One of the best ways to study is using Active recall, which can be achieved by using flashcards. This would be a section under **Module**/Flash Cards. Digital flashcards are extremely convenient to have since you can learn a few topics anytime during the day when you have a few spare minutes.

Flashcards show a definition on one side of the card, and the user must name the term. User taps to continue and then sees the term to know if they are correct. This can also be reversed to show a term and the user must explain the definition.

The flashcards can be sorted into stacks of cards, where the user can create, edit, and delete cards in a stack; and create, edit, and delete stacks. Users can pick a finished stack to study from, and cards will be shown at random.

Pomodoro Timer:

Another great way to study is through interleaving, and we thought a good way to use it would be to combine it with a pomodoro timer.

Interleaving is the process where students mix, or interleave, multiple subjects or topics while studying.

Pomodoro is a time management method where you study for **x** minutes, and take a break for **y** minutes, and repeat.

We could combine the two into the application where a user sets their study/rest times for the pomodoro timer, and then picks 2+ modules, or enters 2+ topics of their choice that they want to study. The timer will then randomly choose one of the modules/topics for each study session for the user. This will be able to run in the background while other parts of the application are used as the user can study from inside the application and doesn't have to use a different application or way to study.

Descriptions

We plan to make a mobile app to assist the user in managing their academic resources and aid in studying and organisation.

The landing page will give you a view of your modules for the year, allowing you to access previous years from a dropdown menu.

It also allows you to add new modules.

See figure 1

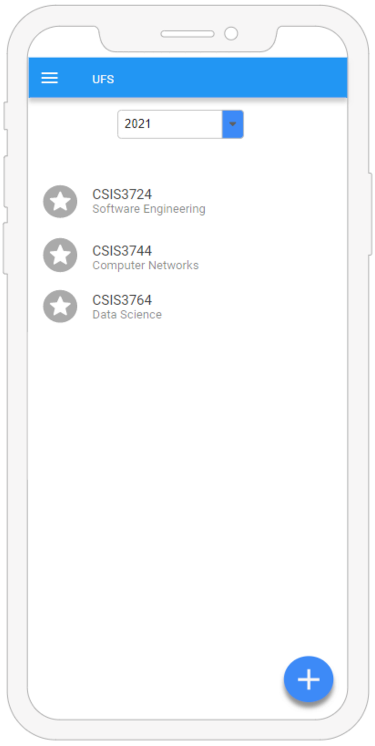


Figure 1: Landing Page

Each module has a view allowing the user to access the following:

- **Study material**: material given by lecturers in different formats such as pdf, videos, and text.
- **Notes**: allowing for creation and editing of notes; creation and uploading of photos, audio, and videos.
- **Assessment Vault**: a place to add any completed assessments like assignments, tests, and exams. Each type of assessment has their own section in the vault.
- **Marks**: a place for the user to add in marks for any assessments.
- **Flash cards**: a system to create and view flash cards for the module, to assist in studying.

  See figure 2

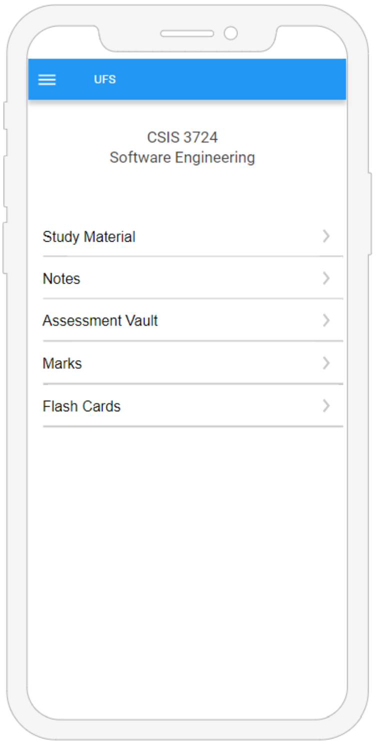


Figure 2: Module page

There is also a global menu for the user to access the following:

- **Statistics**: a system to view mark statistics and set up mark allocations, for an easy view of your current score.
- **Calendar**: Combines the To-Do list and diary, also notifying the user upcoming assessments and events.
- **Timetable**: Allows the user to view and edit their weekly timetable, for a basic schedule.
- **Address Book**: Allows the user to keep contact information of lecturers and other students this includes things like email, phone, office, or addresses.
- **Pomodoro Timer**: Subjects or topics (whichever is specified by the user) are randomly selected for the user to work on. This will work in conjunction with the flash cards.
  The user is timed for each subject or topic, according to the Pomodoro technique.

  See figure 3.

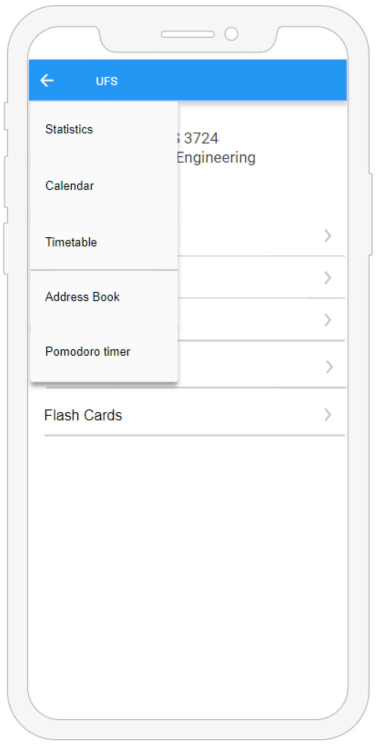


Figure 3: Global Menu

Specification Document

Student Career Portfolio

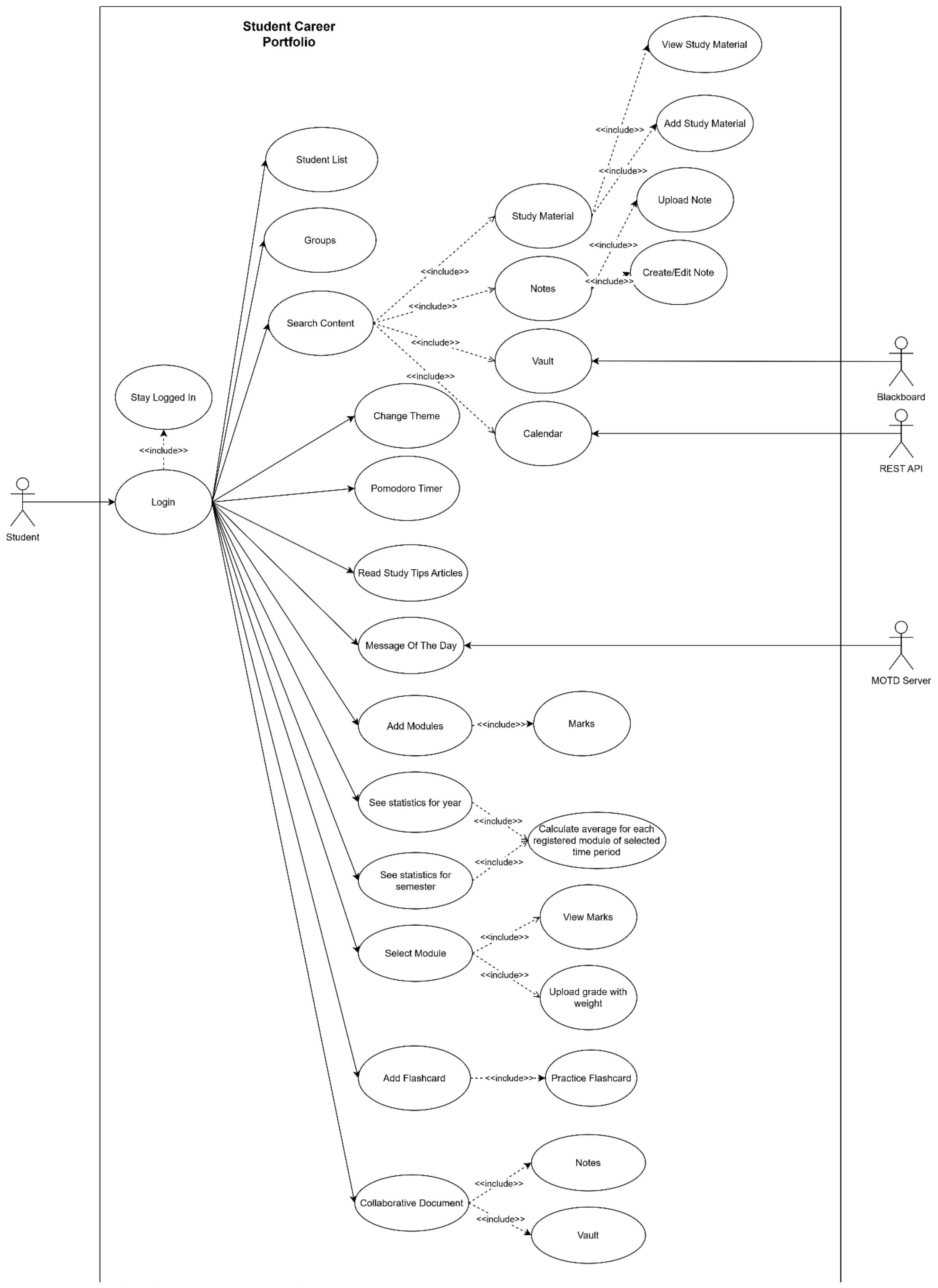


Figure 4: Completed System Use Case diagram

Login

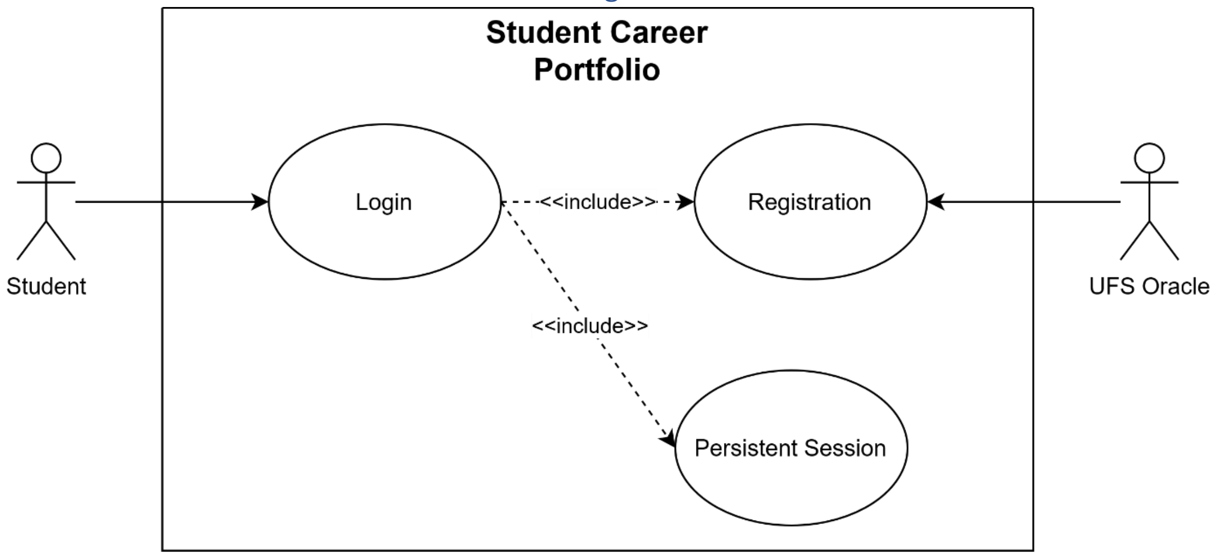


Figure 5: Login Use Case diagram

Brief Description

The Login use case enables the student to login and access the system.

Step-by-Step Description

1. On first start the student is presented with a login page.
2. The login page asks for their student number and password which were registered from the UFS Oracle system.
3. A checkbox asking whether the student wants to stay logged in is also displayed.
4. Once the student clicks login the system will check their credentials, if the credentials are correct the system creates a session for the student and the student is allowed access to the system.

Best-case scenario

1. Peter opens the application.
2. Application presents Peter with login page.
3. Peter enters their correct credentials and clicks the login button.
4. The system checks the credentials, they are correct, so the system grants a session to Peter.
5. Peter is now in the app.

Worst-case scenario

1. Ben opens the application.
2. Application presents John with login page.
3. John inserts SQL injection into their username and password.
4. The system sees this pitiful attempt and responds with a message saying the details are wrong.
5. John is disappointed that his hack didn’t work.

Alternative scenario

1. Sarah opens the application.
2. Application presents Sarah with login page.
3. Sarah doesn’t fill in details and presses login.
4. The system responds by specifying the fields that need to be filled in.

Search

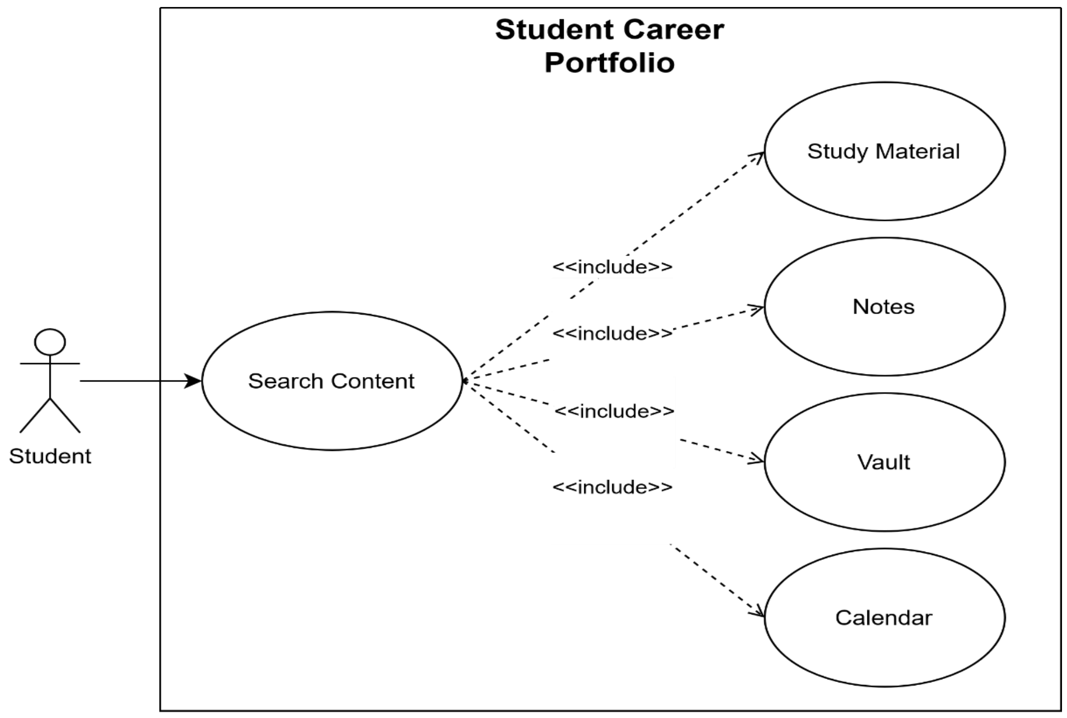


Figure 6: Search Use Case diagram

Brief Description

The Search Content use case allows the user to search the content added on the platform.

Step-by-Step Description

1. Student presses the search button available in any module view and in the dashboard.
2. When selected it will allow the student to search all content under the menu they are in.
3. The search will allow the student to type what they're looking for and will update live, showing relevant files/notes/events.

Best-case scenario

1. Peter presses the search button.
2. System responds with a prompt for what to search.
3. Peter types something.
4. System searches as Peter is typing and displays relevant search results.

Worst-case scenario

1. Ben presses the search button.
2. System responds with a prompt for what to search.
3. Ben types very fast to overload the live search.
4. System only sees the quick requests and responds once Ben calms down.

Alternative scenario

1. Sarah presses the search button
2. System responds with a prompt for what to search.
3. Sarah types a query that isn’t in the system.
4. System responds by saying there are no results.

Study Material

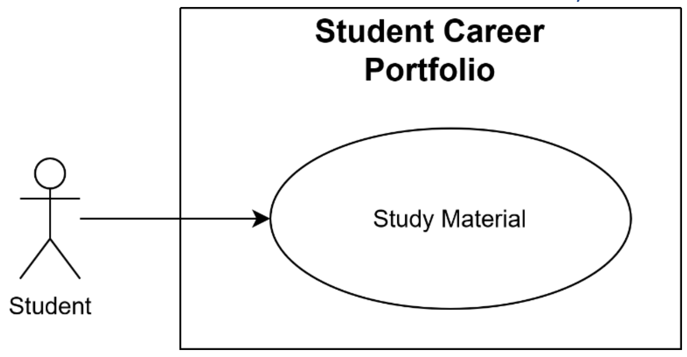


Figure 7: Study Material Use Case diagram

Brief Description

The Study Material use case enables the student to add and review study material for different modules on the system.

Step-by-Step Description

1. Student navigates to the relevant module.
2. Student goes into the Study Material section, where existing study material is shown.
3. Student clicks the + button and is able to upload any type of content eg. PDF, mp3, PPT, mp4, etc... or create a new folder to add content to.

Best-case scenario

1. Peter goes into the Study Material for a module.
2. System presents Peter with his study material, if any.
3. Peter clicks to add new study material.
4. System asks Peter for the document to be uploaded.
5. Peter selects a new pdf with slides to upload.
6. System uploads the document and shows Peter his study material, with newly added content.

Worst-case scenario

1. Ben goes into the Study Material for a module.
2. System presents Ben with his study material, if any.
3. Ben clicks to upload new study material.
4. System asks Ben for the document to be uploaded.
5. Ben selects a very large file that won’t fit in the systems storage limit.
6. Storage responds by saying the file is too large and takes Ben back to his Study Material.
7. Ben is now disappointed.

Alternative scenario

1. Sarah goes into the Study Material for a module.
2. System presents Sarah with her study material, if any.
3. Sarah clicks to add new study material.
4. System asks Sarah for the document to be uploaded.
5. Sarah cancels by pressing back.
6. System takes Sarah back to the Study Material view for the module.

Modules

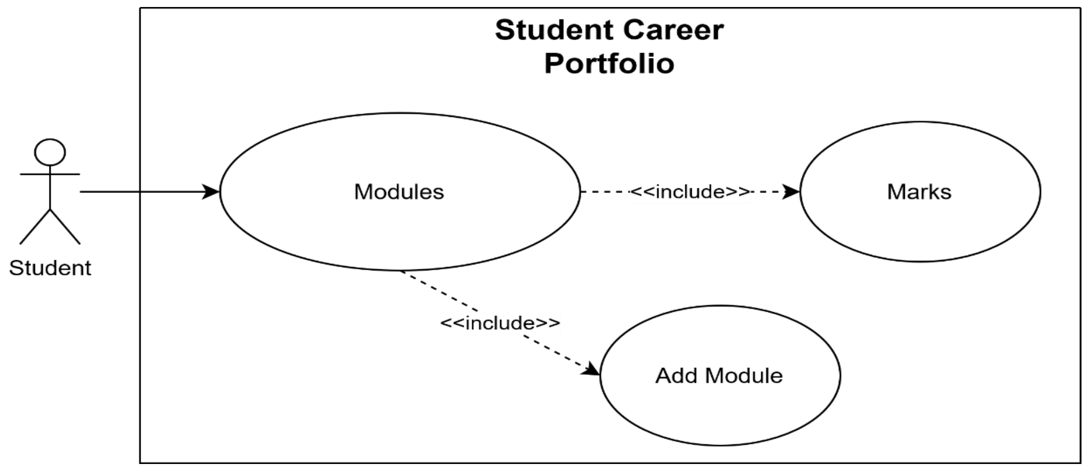


Figure 8: Modules Use Case diagram

Brief Description

The Modules use case enables the student to add new modules to their current academic year, or to link old ones (re-doing the module), it also displays previews of the students marks for the module.

Step-by-Step Description

1. In the main landing page (dashboard) the student will be presented with their current modules for the selected year (specified in a dropdown).
2. A preview of the marks for each module will also be displayed in color-coded UI elements, green for good marks (80+%), yellow for fine marks (50-80%), and red for failing marks.
3. The student can then click the + button to either add a new module or link an existing one.
4. In the case that the user adds an existing module, the system will give the option to import the content added from previous years. (The content is kept in the previous years, for archival purposes)

Best-case scenario

1. Peter clicks to add a module.
2. System responds by asking module code.
3. Peter enters module code and name and clicks add.
4. System adds the module to Peter's dashboard.

Worst-case scenario

1. Ben clicks to add a module.
2. System responds by asking module code.
3. Ben enters module code and name and clicks add.
4. System adds the module to Ben's dashboard.
5. Ben repeats this until his dashboard is full of random modules.
6. System allows Ben to scroll through his mountain of modules.

Alternative scenario

1. Sarah wants to see her marks.
2. The system shows Sarah all of her modules with their calculated marks.
3. Sarah is satisfied and closes the application.

Message of the Day

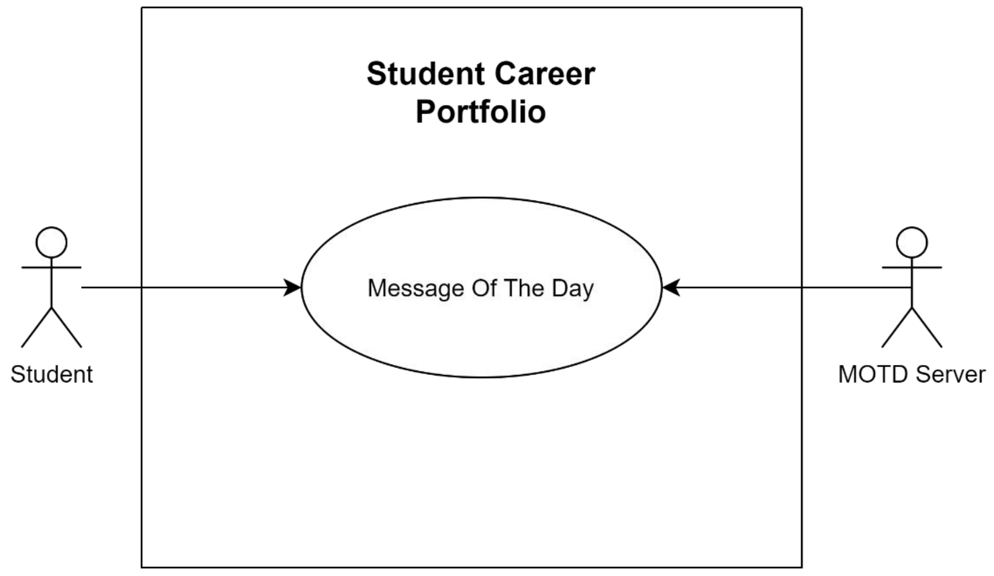


Figure 9: Message of the Day Use Case diagram

Brief Description

The Message of the Day (MOTD) use case enables the Message of the Day Server to daily send an inspirational quote to the application so that it can be displayed on the dashboard for the student to see. The student can customise the Message of the Day.

Step-by-Step Description

1. Student enters the app and is presented the dashboard.
    a. The system checks the student's settings to see if the MOTD should be displayed. If not skip remaining steps.
    b. The system checks if the MOTD should be changed (if it is a new day and MOTD is enabled). If so, the system gets a new MOTD from an external MOTD Server through their APIs.
    c. The system displays the appropriate MOTD on the dashboard.

2. The enable/disable MOTD option will automatically be set to enabled.
    a. If the student sets the option to disabled, the current message of the day will remain on the dashboard and will not change.
    b. If the student later sets the option to enabled, the message of the day will start changing daily, starting from the following day. If the MOTD visible/hidden option is set to hidden, it will be set to visible.

3. The MOTD visible/hidden option will automatically be set to visible.
    a. If the student sets the option to hidden, the message of the day will not be displayed. If daily updates to message of the day is set to enabled, it will be set to disabled.

b. If the student later sets the option to visible, the message of the day will be visible on the dashboard again.

4. If the student selects the option to edit the MOTD, he/she can type a new message that will be displayed on the dashboard for that day.

   a. If the MOTD enable/disable option is set to disabled the new custom message of the day will remain on the dashboard indefinitely.

Best-case Scenario

1. Peter sees the Message of the Day on his dashboard.
2. Peter edits the Message of the Day. He enters a new message. This message is displayed on the dashboard.
3. Peter opens his application the following day. He sees a new message on the dashboard.
4. Peter disables the Message of the Day. Peter is informed that: "Message of the day will no longer be updated daily."
5. Peter opens his application the following day. The message has not changed since the previous day.
6. Peter sets the Message of the Day to hidden. The message is not displayed on the dashboard.
7. Peter sets the Message of the Day to enabled. The message of the day is automatically set to visible, and it is displayed on the dashboard. Peter is informed that: "Message of the day will be updated daily. You will receive a new message starting tomorrow."
8. Peter opens his application the following day. He sees a new message on the dashboard.

Worst-case Scenario

1. Sally sees the Message of the Day on her dashboard.
2. Sally edits the Message of the Day. She enters nothing.
3. The Message of the Day on the dashboard is updated to display a blank message.

Alternate Scenarios

<u>Alternative Scenario 1</u>

1. Thomas sees the Message of the Day on his dashboard.
2. He sets the Message of the Day to disabled. Thomas is informed that: "Message of the day will no longer be updated daily."
3. Thomas edits the Message of the Day to display a custom message.
4. Thomas opens his application the following day. He sees his custom message.
5. Every subsequent day, Thomas still sees this same custom message on his dashboard.
6. Thomas sets the Message of the Day to enabled. Thomas is informed that: "Message of the day will be updated daily. You will receive a new message starting tomorrow."
7. Thomas sets the Message of the Day to hidden. It is no longer displayed on his dashboard. Message of the Day is also automatically set to disabled. Thomas is informed that "Message of the Day has been disabled."

Statistics

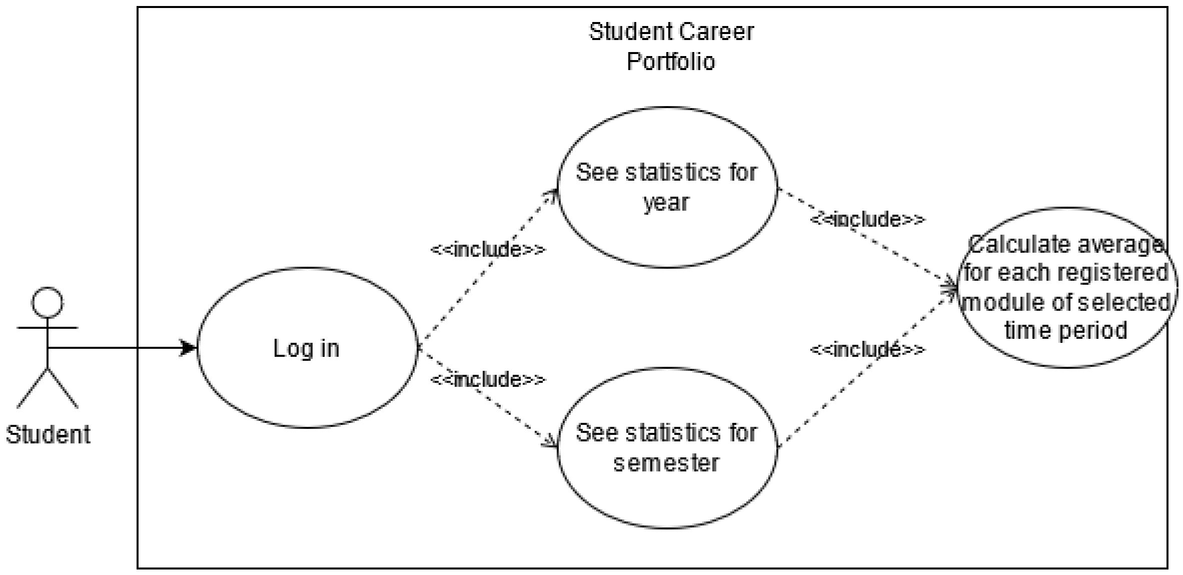


Figure 10: Statistics Use Case diagram

Brief Description

The user logs in and an average for the semester or year is displayed on the dashboard.

The time period can be selected of which to calculate the average from.

Step-by-step A: Description

1. Student logs in to the dashboard.
   1.1. Statistics (average of all marks) are calculated using all the marks and weights uploaded in the 'upload marks' use case for the entire year up to that point.
   1.2. Statistics (average for all marks) are displayed as a percentage for a year time period by default on the dashboard in a coloured button (dark green for 80%+, yellow for 60%+, and red below 50%) that gradually shifts with the percentage values.
2. Student clicks on the coloured button and a dropdown list appears with the available options of 'Semester 1' and 'Semester 2'.
3. Student selects semester 1 to display the average for all registered first semester modules.
   3.1. The average for semester 1 is calculated using all the marks and weights uploaded in the 'upload marks' use case under the first semester.
   3.2. Statistics (average for all marks) are displayed as a percentage for the first semester time period by default on the dashboard in a coloured button (dark green for 80%+, yellow for 60%+, and red below 50%) that gradually shifts with the percentage values.

Step-by-step B: Description

1. Student logs in to the dashboard.
   1.1. Statistics (average of all marks) are calculated using all the marks and weights uploaded in the 'upload marks' use case for the entire year up to that point.
   1.2. Statistics (average for all marks) are displayed as a percentage for a year time period by default on the dashboard in a coloured button (dark green for 80%+, yellow for 60%+, and red below 50%) that gradually shifts with the percentage values.
2. Student clicks on the coloured button and a dropdown list appears with the available options of 'Semester 1' and 'Semester 2'.
3. Student selects semester 2 to display the average for all registered first semester modules.

    3.1. The average for semester 2 is calculated using all the marks and weights uploaded in the 'upload marks' use case under the first semester.
    3.2. Statistics (average for all marks) are displayed as a percentage for the first semester time period by default on the dashboard in a coloured button (dark green for 80%+, yellow for 60%+, and red below 50%) that gradually shifts with the percentage values.

Step-by-step C: Description

1. Student's current selected time period is either semester 1 or semester 2.
    1.1. Statistics (average of all marks) are calculated using all the marks and weights uploaded in the 'upload marks' use case for that specific time period as stipulated in Step-by-step A or B.
    1.2. Statistics (average for all marks) are displayed as a percentage for a year time period by default on the dashboard in a coloured button (dark green for 80%+, yellow for 60%+, and red below 50%) that gradually shifts with the percentage values.
2. Student wants to revert to the year calculation and clicks the button.
3. A dropdown list appears with the options 'year' and either 'semester 1', or 'semester 2', as an opposite of what is currently selected.
4. Student selects 'year'.
    4.1. The average for the year is calculated using all the marks and weights uploaded in the 'upload marks' use case under both the first semester and second semester up to that point of what has been uploaded.
    4.2. Statistics (average for all marks) are displayed as a percentage for the first semester time period by default on the dashboard in a coloured button (dark green for 80%+, yellow for 60%+, and red below 50%) that gradually shifts with the percentage values.

Best-case Scenario

1. Johan opens the application and logs in.
2. Johan clicks on the 'average' button to change the calculation displayed from 'year' to 'semester 1'.
3. Application calculates all the uploaded marks and displays his average for the 1st semester in the correct format.

Worst-case scenario

1. James opens the application.
2. The average doesn't display at all; because marks uploaded were not in the correct data format.
    2.1. The application has validation checks when the student uploads marks to ensure all data is in the correct format, prompting the student to re-enter marks with a note on correct data types.

Alternate Scenarios

<u>Alternate Scenario 1:</u>

1. Lucas opens the application
2. Average is calculated incorrectly due to a logic error – Lucas swapped the 'grade' with the 'weight' when uploading his marks.
3. He sees an incorrect average for the 'year'
    3.1. In the 'upload marks' section for each module, Lucas can edit each mark and weight entered after it has been uploaded.

Upload marks

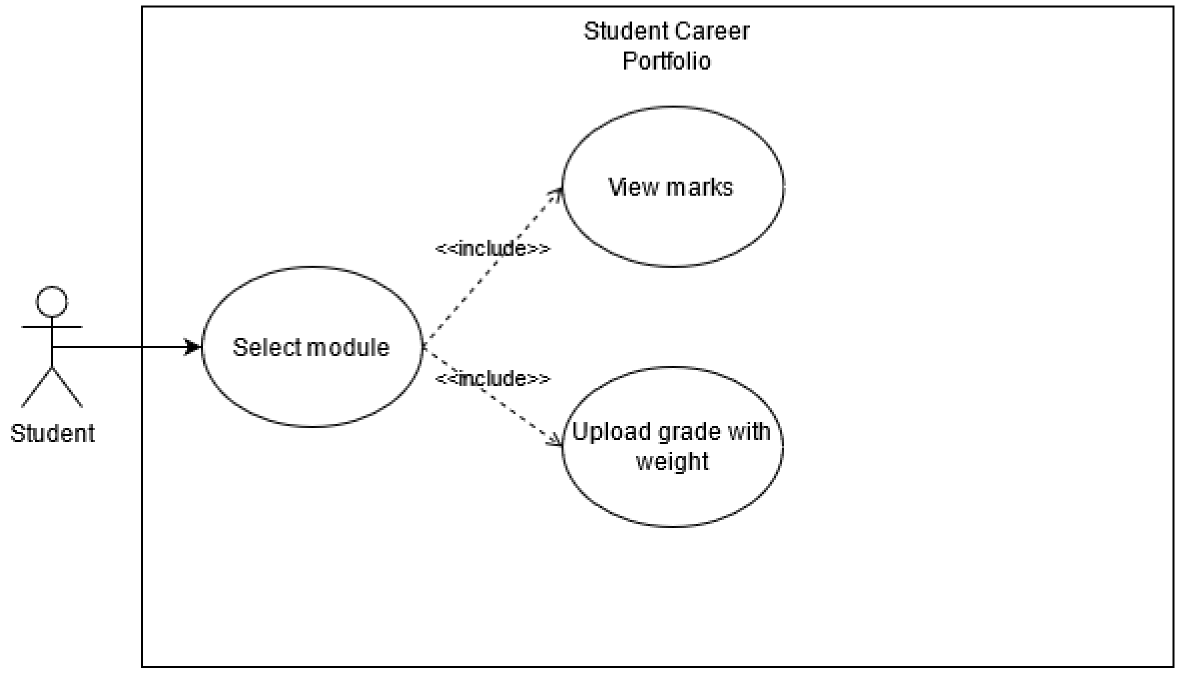


Figure 11: Upload Marks Use Case diagram

Brief Description

The student can upload marks, the system then calculates the grade average for each module from a button on the module page. The module page displays the average in a coloured rectangle.

Step-by-step Description

1. Student logs in to the application
2. Student selects a module from the dashboard.
3. Student then taps on the 'marks' button to upload a mark.
4. The student then:
   4.1. Input the weight for the test/assignment.
   4.2. Input the mark for the test/assignment.
   4.3. Input the name for the test/assignment.
   4.4. Uploads the marks and returns to the module screen.
5. The average is then calculated based on the weights, displaying a percentage on the marks button with a corresponding button colour (dark green for 80%+, yellow for 60%+, and red below 50%) that gradually shifts with the percentage values.
6. The average is then displayed on the module's home screen.

Best-case Scenario

1. Louis logs in.
2. He selects a module from the dashboard.
3. He clicks on the 'marks' button.
4. He enters a mark, a weight, and a name for an assignment.
5. The average is then updated and displayed on the module's home screen as a coloured button with the correct percentage average in it.

Worst-case scenario

1. Jaco logs in.
2. He selects a module from the home screen.
3. He selects the 'marks' button to upload marks.
4. He uploads a null value as a weight and a test mark.
    4.1. Application has validation checks when the student uploads marks to ensure all data is in the correct format and not null and prompts the user to enter the marks again with a note on what data types are accepted.

Alternate Scenarios

<u>Alternate Scenario 1:</u>

1. Amy logs in.
2. She selects a module from the dashboard.
3. She notices the marks displayed are incorrect for that module.
4. She clicks on the 'marks' button to upload marks.
5. She edits each mark and weight previously enter to the correct values.
6. Module mark is now displayed correctly.

Study Tips

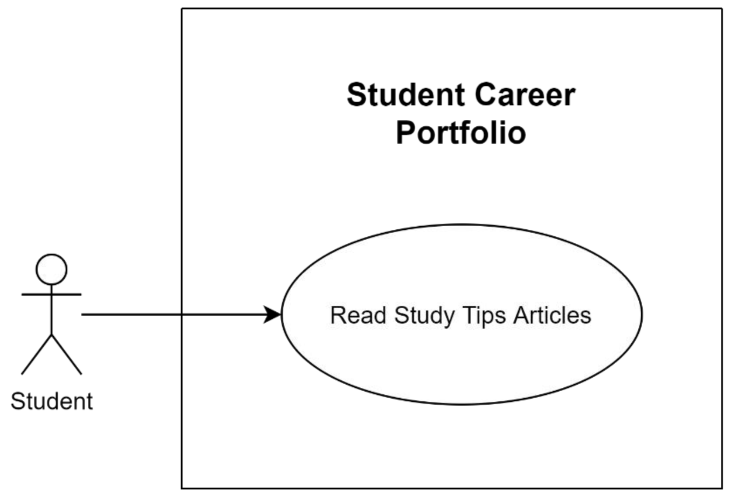


Figure 12: Study Tips Use Case diagram

Brief Description

The Read Study Tips Articles use case enables the student to select an article relating to study tips and methods and read it in the application.

Step-by-Step Description

1. A list of articles about study tips and methods will be displayed to the student.
2. The student can select one of these articles and read it in the application.

Best-case Scenario

1. Many articles have been uploaded by CTL.

2. Peter sees the list and selects one of the articles.

3. The article is displayed to him.

Worst-case Scenario

1. No articles have been uploaded by CTL yet.

2. Sally sees a message stating “No articles uploaded yet. Please check back soon!”

Pomodoro timer

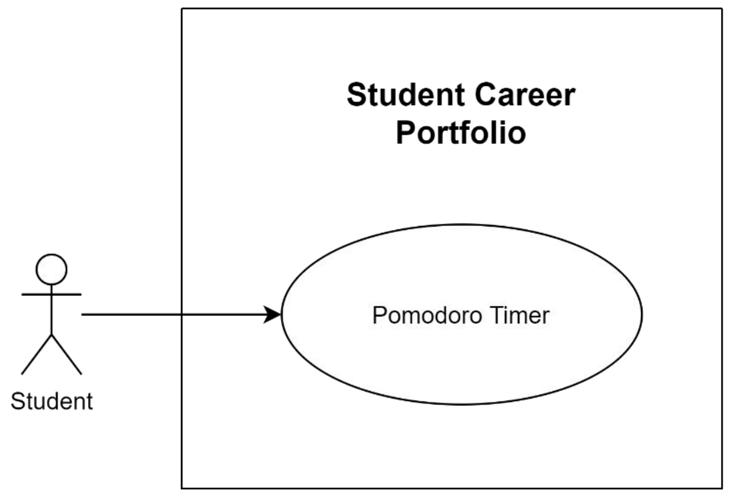


Figure 13: Pomodoro Timer Use Case diagram

Brief Description

The Pomodoro Timer use case enables the student to break study material into chunks of 25 minutes and then time himself/herself using the Pomodoro method.

Step-by-Step Description

1. The student will enter a list of subjects, tasks or topics which he/she aims to accomplish. Each entry must consist of a task/topic or part of a task/topic that can be completed in 25 minutes.
2. Once the student starts the timer, the current task with the amount of time left of the allotted 25 minutes will be displayed on the dashboard.
3. Once the 25 minutes have run out, it will be indicated on the dashboard that the student has a break of 5 minutes.
4. Once the 5 minutes has run out, the next topic with its allotted 25 minutes will be displayed on the dashboard.
5. The 25 minute study time and 5 minute break time cycle will continue 4 times. After this, the student will receive a 25 minute break.
6. The whole process from step 2 to step 5 will be repeated until all the tasks as entered by the student has been completed.
7. The student can reset the timer at any time.

Best-case Scenario

1. Peter enters 5 tasks he wants to complete.
2. He starts the timer. The name of the first task with a timer counting down from 25:00 minutes is displayed on the dashboard.
3. After 25 minutes, the word “Break” with a timer counting down from 05:00 minutes is displayed on the dashboard.
4. After 5 minutes, the name of the second task with a timer counting down from 25:00 minutes is displayed on the dashboard.
5. The 25 minute work slot and 5 minute break slot are repeated 4 times.

6. When Peter has completed the 25 minutes assigned to the fourth task, the word “Long Break” with a timer counting down from 25:00 minutes is displayed on the dashboard.
7. After this, the name of the fifth (last) task with a timer counting down from 25:00 minutes is displayed on the dashboard again.
8. Upon completion of the last task, the words “Well done! All tasks are completed!” will appear on the dashboard.

Worst-case Scenario

1. Sally enters 6 tasks she wants to complete.
2. She starts the timer. The name of the first task with a timer counting down from 25:00 minutes is displayed on the dashboard.
3. After a few minutes, Sally closes the application.
4. A message appears asking her to confirm her action. She is warned that the timer will completely restart, and all of the entered data will be lost.
5. Sally confirms that she wants to close the application.
6. When she opens the application and navigates to the Pomodoro timer again, no entries appear.

Alternate Scenarios

<u>Alternative Scenario 1</u>

1. Thomas enters 6 tasks he wants to complete.
2. He starts the timer. The name of the first task with a timer counting down from 25:00 minutes is displayed on the dashboard.
3. After 25 minutes, the word “Break” with a timer counting down from 05:00 minutes is displayed on the dashboard.
4. After 5 minutes, the name of the second task with a timer counting down from 25:00 minutes is displayed on the dashboard.
5. Thomas navigates to the list of tasks he entered. He presses the “Reset” button.
6. A message appears asking him to confirm his action. He is warned that all tasks will be discarded.
7. Thomas confirms his action. All tasks are discarded, and no timer is displayed on the dashboard anymore.
8. Thomas can now enter new tasks.

Themes

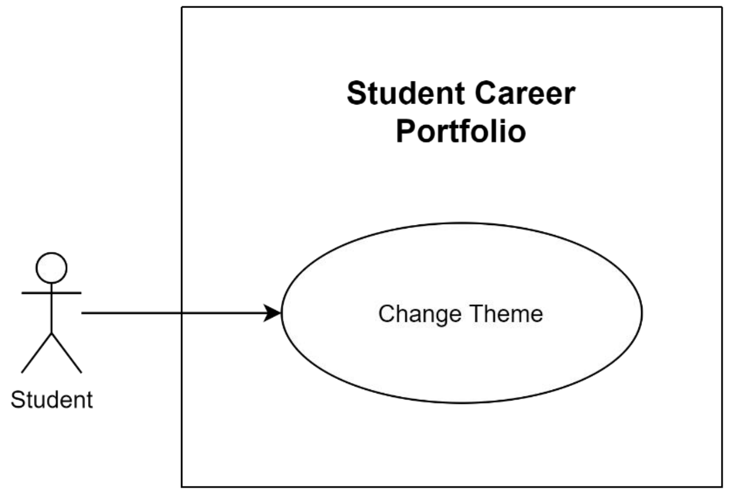


Figure 14: Themes Use Case diagram

Brief Description

The Change Theme use case enables the student to change the theme of the application to a theme he/she has already unlocked.

Step-by-Step Description

1. Display all four theme names, its status (locked or unlocked) and what task the student needs to perform to unlock it. (One of the themes is the default theme and will always be unlocked).
2. Check which themes are still locked and disable selection of these themes.
3. Allow the user to select only one of the unlocked themes and confirm the selection.
4. Change all font types, font size, background images and dashboard colours in the application according to the specified theme.

Best-case Scenario

1. Peter has unlocked all the themes.

2. Peter selects one of the themes.

3. Peter confirms his selection.

4. All font types, font size, background images and dashboard colours in the application change according to the specified theme.

Worst-case Scenario

1. Sally has two unlocked themes.

2. Sally selects one of the themes.

3. Sally does not confirm her selection.

4. The theme does not change. The list of themes is displayed to Sally again.

Alternate Scenarios

<u>Alternative Scenario 1</u>

1. Thomas has three unlocked themes.

2. Thomas attempts to select one of the locked themes.

3. Nothing happens as Thomas is not allowed to select a locked theme.

Upload Note

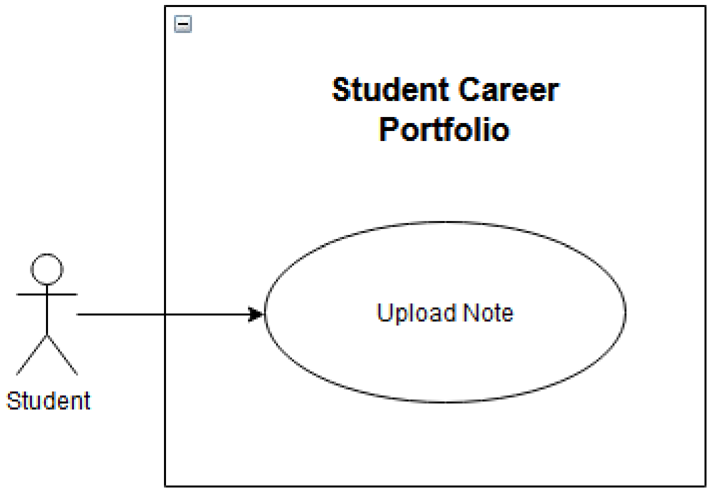


Figure 15: Upload Note Use Case diagram

Brief Description

The *Upload Note* use case enables a student to upload notes that are in any file format such as docx, pdf, txt, etc. and any media format such images, videos, and audio.

Step-by-Step Description

1. Student taps on upload file.
2. Student picks if they are uploading a document, audio, or media from gallery.
3. Depending on a selection, a different GUI will open for each, with documents and audio showing a list with file names, and media from gallery showing the images/videos as they would appear in the phone’s gallery.
4. Student selects the file/files they want to upload.
5. Student confirms they selection and the file(s) are uploaded.

Best-case scenario

1. Peter taps upload file.
2. Peter selects a document to upload.
3. Peter confirms that he wants to upload the document.
4. The document is uploaded to the app and saved.

Worst-case scenario

1. John taps upload file.
2. John selects a file that is not supported by the app.
3. John confirms that he wants to upload the file.
4. A pop-up is displayed telling John that the selected file format is not supported, and to contact support.

Alternate scenarios

1. Thabiso taps upload file.
2. Thabiso selects a document to upload.
3. Thabiso confirms he wants to upload the file.
4. While the document is being uploaded or saved, Thabiso closes the app.
5. The file finishes uploading and saving in the background.

Create/Edit Note

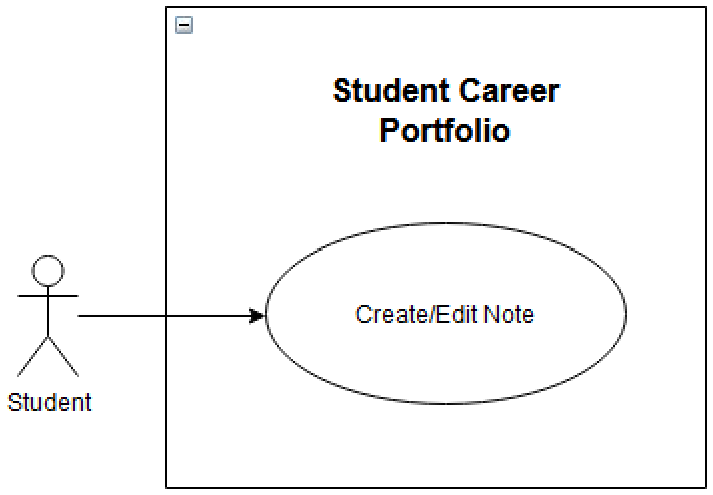


Figure 16: Create/Edit Note Use Case diagram

Brief Description

The *Create/Edit Note* use case allows a student to create, open and edit, and view the note.

Step-by-Step Description

<u>A – View Note</u>

1. Student taps on a note.
2. Note opens for editing if text file, viewing for any images or documents, and plays back for video and audio files.
3. When the student exits the note, any unsaved changes are saved.

<u>B – Create Note</u>

1. Student taps on the "create note" button.
2. Student selects Text File.
3. A new text file is automatically created and opened; student must enter a note name before continuing.
4. Student can type in the text editor to make changes to their note.
5. When the user exits the note, any unsaved changes are saved.

<u>C – Create Flashcard Deck</u>

1. Student taps on the "create note" button.
2. Student selects Flashcard Deck.
3. Student inputs a name for the deck.
4. The deck is saved, and the student is returned to the Notes GUI.

Best-case scenario

<u>A – View Note</u>

1. Khumbudzo taps on a note.
2. Note opens editor/view/playback depending on file format.

3. Khumbudzo exits the note.
4. If any changes were made, the changes are saved.

B - Create Note

1. Priaska taps on “create note” button.
2. Priaska selects Text File.
3. Priaska enters “First Note” into the file name field and continues.
4. Priaska types “This is my first note” into the editor.
5. Priaska exits the note.
6. The note is uploaded to the app and saved.

C - Create Flashcard Deck

1. Robert taps on “create note” button.
2. Robert selects Flashcard Deck.
3. Roberts enters “First deck” into the name field.
4. The deck is uploaded and saved to the app.

Worst-case Scenario

A - View Note

1. Khumbudzo taps on an existing text note.
2. Note opens and Khumbudzo makes changes.
3. Khumbudzo exits the app.
4. The changes are saved in the background before the app fully closes.

B - Create Note

1. Priaska taps on “create note” button.
2. Priaska selects Text File.
3. Priaska enters “Second Note” into the file name field and continues.
4. Priaska types “This is my second note” into the editor.
5. Priaska exits the app.
6. The note is uploaded and saved to the app in the background before it fully closes.

C - Create Flashcard Deck

1. Robert taps on “create note” button.
2. Robert selects Flashcard Deck.
3. Robert enters “Second deck” into the name field and continues.
4. Robert exits the app before the deck can be saved.
5. The deck is saved in the background before the app fully closes.

Alternative scenarios

B - Create Note

1. Priaska taps on “create note” button.
2. Priaska selects Text File.
3. Priaska tries to continue without entering a note name.
4. An error is displayed on screen that a note name is required.

5. App stays on same screen where Priaska must enter a note name until a name is entered or cancel is tapped.

<u>C - Create Flashcard Deck</u>

1. Robert taps on “create note” button.
2. Robert selects Flashcard Deck.
3. Robert tries to continue without entering a flashcard deck name.
4. An error is displayed on screen that a deck name is required.
5. App stays on same screen where Robert must enter a deck name until a name is entered or cancel is tapped.

Note Menu

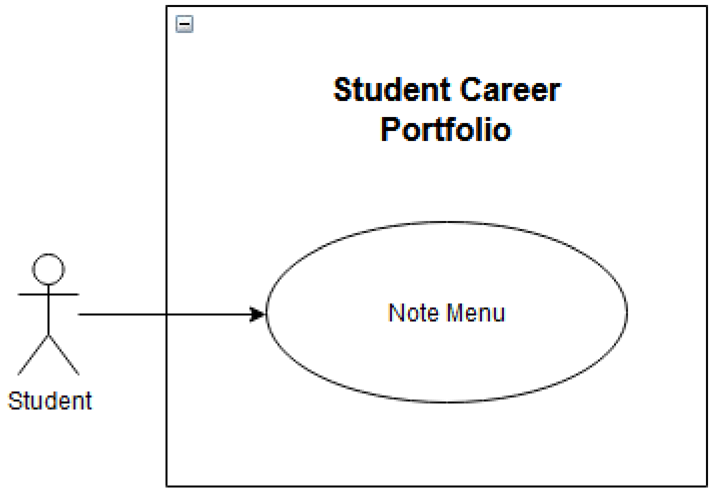


Figure 17: Note Menu Use Case diagram

Brief Description

The *Note Menu* use case enables a student to open a menu of additional actions that can be done with their notes, and perform those actions

Step-by-Step Description

1. Student long taps on a note.
2. A menu is displayed, showing the additional actions.

<u>A - Delete</u>

1. Student selects delete note.
2. Note is deleted and student is returned to Notes section.

<u>B - Rename</u>

1. Student selects rename note.
2. Student inputs a new name for the note and clicks OK.
3. Note is renamed and student is returned to the Notes section.

Best-case scenario

<u>A – Delete</u>

1. Shira long taps a note.
2. The note menu is displayed, Shira taps Delete.
3. The note is removed from the app and Shira is returned to the Notes section.

<u>B – Rename</u>

1. Kagiso long taps a note.
2. The note menu is displayed, Kagiso taps rename.
3. Kagiso types “changed note” into the rename field.
4. The note is renamed to “changed note” and saved.

Worst-case scenario

<u>B – Rename</u>

1. Kagiso long taps a note.
2. The note menu is displayed, Kagiso taps rename.
3. Kagiso clears the rename field and tries to continue.
4. An error is displayed that a file can’t have no name and Kagiso stays on the rename screen until a valid name is entered or cancel is tapped.

Add Flashcard

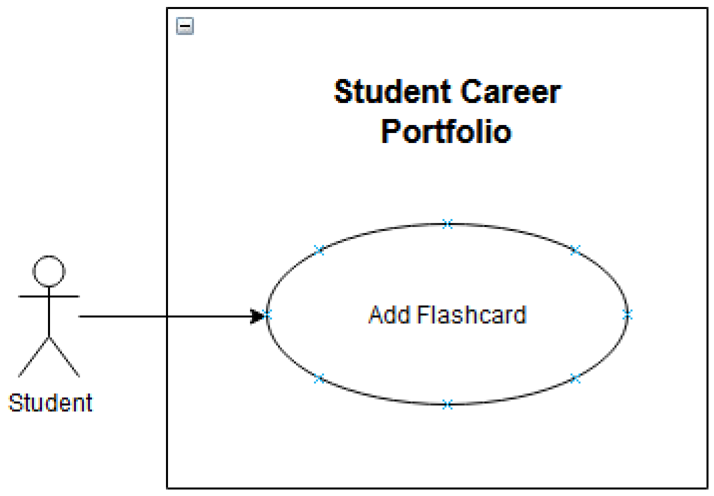


Figure 18: Add Flashcard Use Case diagram

Brief Description

The *Add Flashcard* use case enables a student to add flashcards to an existing flashcard deck.

Step-by-Step Description

1. Student opens note menu on a flashcard deck.
2. Student selects Add Flashcards.
3. Student enters front and back values of the flashcard.
4. Student taps save for the card to be saved.
5. Fields are cleared and a new card can be added.
6. Steps 3 - 5 are repeated until the student clicks back to return to the Notes section.

Best-case scenario

1. Dean long taps on a flashcard deck.
2. Dean selects Add Flashcards.
3. Dean enters “Q1” into the front and “Answer 1” into the back field, and taps save.
4. The flashcard is saved, and the fields are cleared.
5. Dean enters “Q2” into the front and “Answer 2” into the back field, and taps save.
6. The flashcard is saved, and fields are cleared.
7. Dean exits the flashcard adding GUI and is returned to the Notes section.

Worst-case scenario

1. Aiden long taps on a flashcard deck.
2. Aiden selects Add Flashcards.
3. Aiden does not fill in any fields and clicks save.
4. Error pops up that all fields need to be filled and app won’t continue until fields are filled or Aiden cancels adding a card.

Practice Flashcards

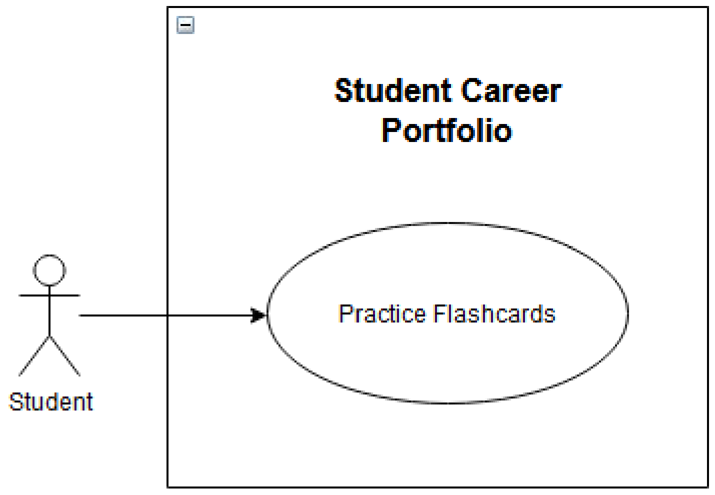


Figure 19: Practice Flashcards Use Case diagram

Brief Description

The *Practice Flashcards* use case enables a student to open and practice an existing flashcard deck.

Step-by-Step Description

1. Student taps on a flashcard deck.
2. Front of a random card from the deck is shown to the student.
3. Student taps on "show answer" to see the back of the flashcard.
4. Student taps on "next card" for the next random card to be shown.
5. Steps 2 - 4 are repeated until all cards have been drawn from the stack, or the student clicks on back to return to the Notes section.

Best-case scenario

1. Lungi taps on a flashcard deck.
2. A random card is selected from the deck and the front is shown to Lungi.
3. Lungi taps "show answer" and the card is flipped to show the back of the flashcard.
4. Lungi taps "next card", and a random card is show again, without replacement.
5. Lungi repeats steps 2 – 4 until he reaches the end of the deck.
6. Lungi is returned to the notes section.

Worst-case scenario

1. Suzy taps on a flashcard deck.
2. A random card is selected from the deck and the front is shown to Suzy.
3. Suzy clicks the back button.
4. Suzy is returned to the Notes view page.

Alternative scenarios

1. Suzy taps on a flashcard deck.
2. A random card is selected from the deck and the front is shown to Suzy.
3. Suzy taps "next card" instead of "show answer", and a random card is show again, without replacement.

Student List

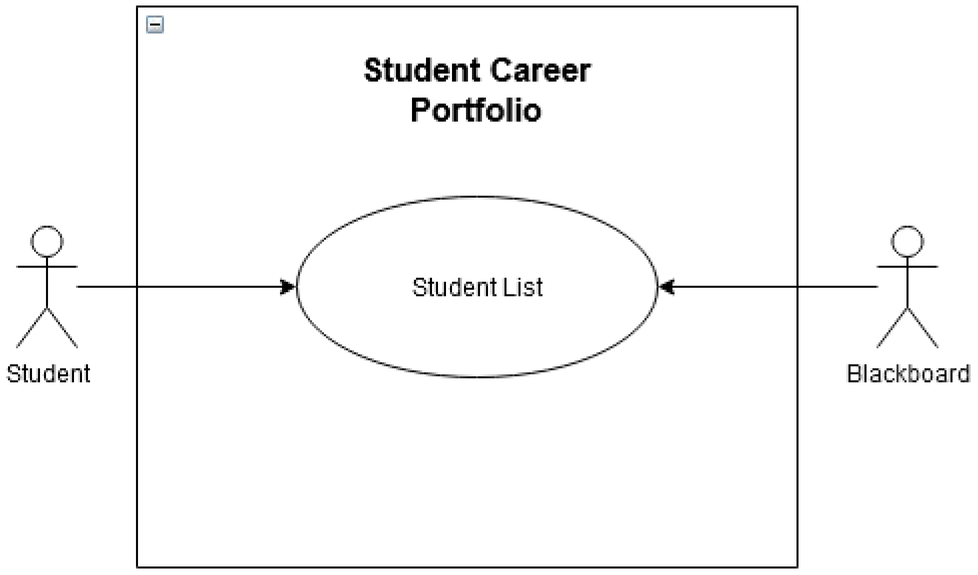


Figure 20: Student List Use Case diagram

Brief Description

The *Student List* use case enables a student to open a list of all other students enrolled in the module.

Step-by-Step Description

1. Student taps on “Student List”.
2. Application gets a list of all students enrolled for the module from Blackboard.
3. List of enrolled students is displayed to the student.

Best-case scenario

1. Quinton taps on “Student List”.
2. App requests a list of all the students enrolled for the module from Blackboard.
3. Blackboard returns a list of all the students to the application.
4. List of enrolled students is shown to Quinton.

Worst-case scenario

1. Quinton taps on “Student List”.
2. App requests a list of all the students enrolled for the module from Blackboard.
3. Request times out.
4. App tells Quinton that it is currently unable to retrieve a list of currently enrolled students.

Alternative scenarios

4. n/a

Student Menu

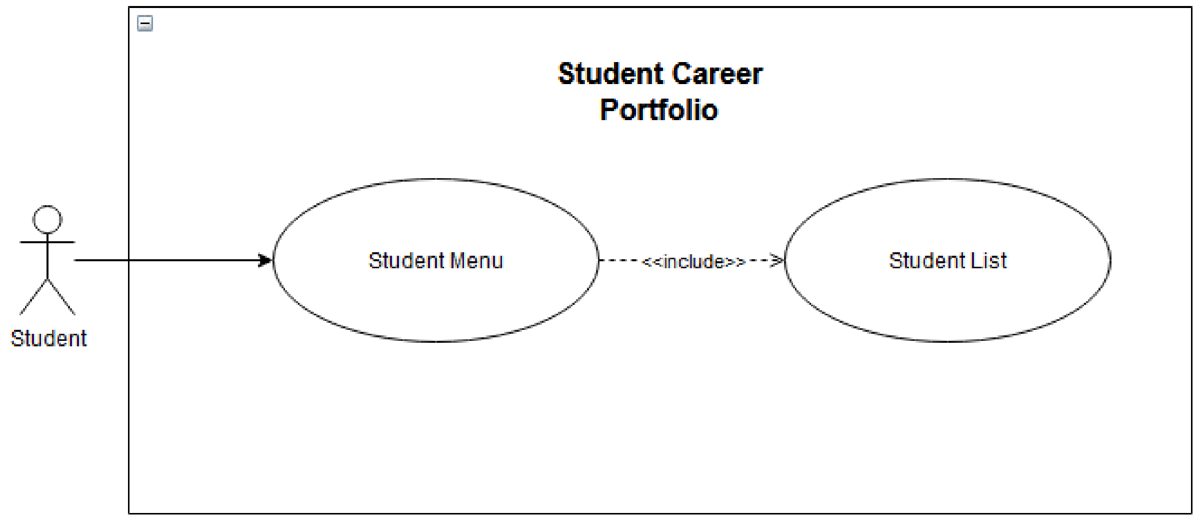


Figure 21: Student Menu Use Case diagram

Brief Description

The *Student Menu* use case enables a student to open a list of other actions.

Step-by-Step Description

1. Student opens student List.
2. Student long taps another student.
3. A menu is displayed, showing the additional actions.

<u>A – Block Student</u>

1. Student taps "Block {studentname}".
2. Selected student is blocked and can no longer communicate with the user student.

<u>B – Unblock Student</u>

1. Student taps "Unblock {studentname}".
2. Selected student is unblocked and can communicate with the user student again.

<u>C – Create Group</u>

1. Student selects "Create group".
2. Student names the group, only the user student and the selected student are in the group discussion.

<u>D – Add to Group</u>

1. Student selects "Add to group".
2. A list of groups that the user student is part of is shown
3. User student selects which group they want to add the student to
4. Selected student is added to the group.

Best-case scenario

<u>A – Block Student</u>

1. Tabraiz opens student list and long taps on Rassie.
2. Tabraiz taps "Block Rassie".
3. Rassie tries to add Tabraiz to a group.
4. Tabraiz is not added to the group and Rassie gets a message that Tabraiz cannot be added since Tabraiz blocked Rassie.

<u>B – Unblock Student</u>

1. Tabraiz opens student list and long taps on Rassie.
2. Tabraiz taps "Unblock Rassie".
3. Rassie tries to add Tabraiz to a group.
4. Tabraiz is added to the group as normal.

<u>C – Create Group</u>

1. Tabraiz opens student list and long taps on Rassie.
2. Tabraiz selects "Create Group".
3. Tabraiz enters "Group A" for the name for the group.
4. "Group A" is created and the initial participants in the group are Tabraiz and Rassie.

<u>D – Add to Group</u>

1. Tabraiz opens student list and long taps on David.
2. Tabraiz selects "Add to group".
3. A list of groups that Tabraiz is part of is shown, Tabraiz selects "Group A".
4. David is added to "Group A".

Worst-case scenario

<u>C – Create Group</u>

1. Tabraiz opens student list and long taps on Rassie.
2. Tabraiz selects "Create Group".
3. Tabraiz enters does not enter a group name and tries to continue.
4. An error message is displayed that a group name is needed and Tabraiz cannot continue until a group name is entered or he cancels.

Alternative scenarios

5. n/a

Groups

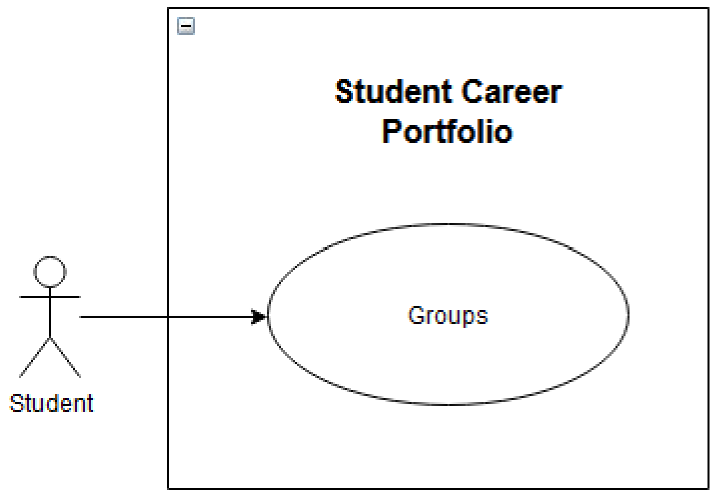


Figure 22: Groups Use Case diagram

Brief Description

The *Groups* use case enables a student to list what groups they are part of and the additional actions that can be done

Step-by-Step Description

1. Student taps on “Groups”.
2. A list of all the groups the student is part of is shown to them.
3. Tapping on one of the groups will open the group chat.

<u>A – Start Group Call</u>

1. Inside the group chat, a user taps on “Group Call”.
2. A group call is started with all the users in the group.
3. Users can leave the call, and the call will end when all users have left.

<u>B – Leave Group</u>

1. Student long taps on a group.
2. Student selects “Leave group”.
3. Student is then removed from the group.

Best-case scenario

<u>A – Start Group Call</u>

1. Andile taps on groups and a list of groups he is part of is shown.
2. Andile opens one of the groups he is part of.
3. Andile taps on “Group Call”.
4. A group call is started with all other students in the group. They need to accept call before they join the call.
5. Once all the students leave, the call is ended.

<u>B – Leave Group</u>

6. Andile long taps on a group he is part of.

7. Andile selects “Leave Group”.
8. Andile is removed from the group and no longer part of it. He cannot interact with it unless someone adds him again.

Worst-case scenario

<u>A – Start Group Call</u>

1. A group call is started.
2. Andile never accepts the call
3. The call notification disappears after a minute.

Alternative scenarios

<u>A – Start Group Call</u>

1. A group call is started.
2. Andile accepts the call.
3. Andile closes the application.
4. The call stays connected in the background.

Collaborative Document

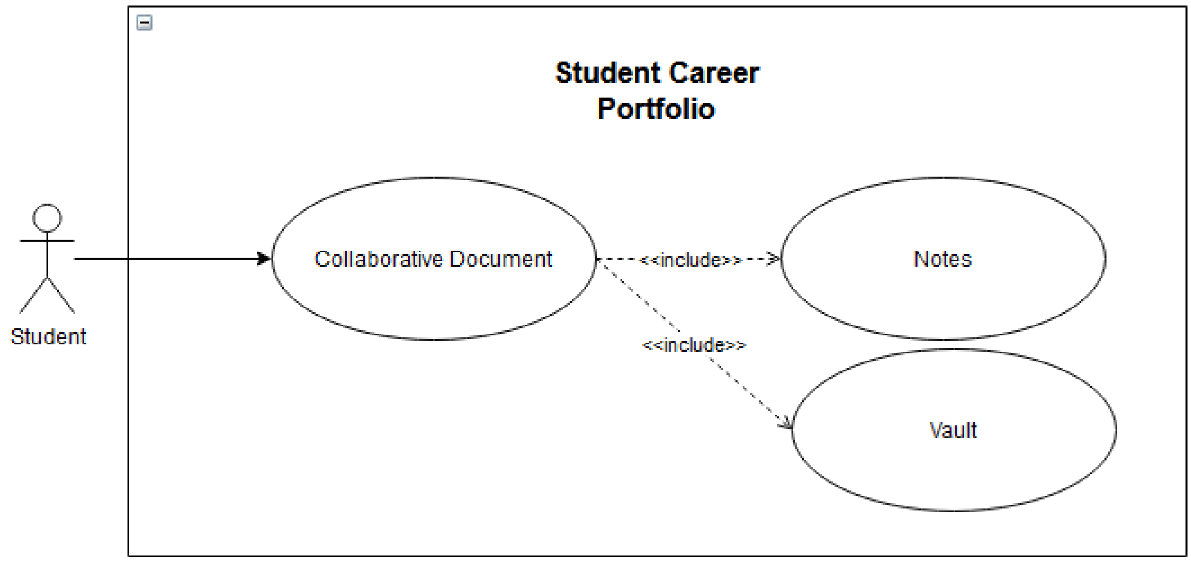


Figure 23: Collaborative Document Use Case diagram

Brief Description

The *Collaborative Document* use case enables a student to share a note or file from the vault for all the users in the group to collaborate on and work on at the same time.

Step-by-Step Description

1. Student opens file menu.
2. Student taps "Collaborate with Group".
3. A list of groups that the student is part of is shown to the student.
4. Student selects which group they would like to share the document to.
5. The document is accessible to all the student in the groups.

Best-case scenario

1. Bjorn opens Notes.
2. Bjorn opens a Notes menu and selects "Collaborate with Group".
3. A list of groups that Bjorn is part of is show, he selects which group he wants to share the note to.
4. Everyone in the group can open it at the same time and do live edits to it.

Worst-case scenario

1. Bjorn opens Notes.
2. Bjorn opens Notes menu and selects "Collaborate with Group".
3. A list of groups that Bjorn is part of is shown, he clicks back.
4. Bjorn is returned to the Notes view.

Alternative scenarios

- n/a

Calendar

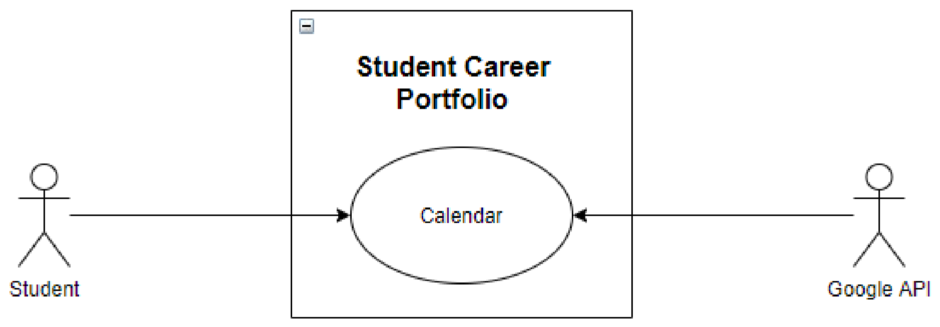


Figure 24: Calendar Use Case diagram

Brief Description

The *Calendar* use case enables a student to use a variety of functionalities, namely, Diary entries, To-do list, add/edit/delete events, add/edit/delete event categories and switching between views.

Step-by-Step A: Description

4. Student selects *Calendar*
5. Application establishes connection to Google API and opens the Google Calendar API
6. *Calendar* is shown with default view being the daily view with the To-do list visible.

Step-by-Step B: Description

1. Student selects *Calendar*
2. Application establishes connection to Google API and opens the Google Calendar API
3. Student selects Monthly view
4. Student is displayed a monthly view with the current month as focus.

Step-by-Step C: Description

1. Student selects *Calendar*
2. Application establishes connection to Google API and opens the Google Calendar API
3. Student selects To-do list view
4. Student is displayed the To-do list.
5. Student can now add/edit/mark as complete their To-do list.

Step-by-Step D: Description

1. Student selects *Calendar*
2. Application establishes connection to Google API and opens the Google Calendar API
3. Student selects to add an event.
4. Event pop-up displays student options to fill in
5. Student selects the date, time, title, and category (if relevant) for event.
6. Student is asked whether there should be an event alert/notification.
7. Student then gets asked if this event should be repeated, and if yes, how frequently.
8. Event is then added to the corresponding date and time.

Step-by-Step E: Description

1. Student selects *Calendar*
2. Application establishes connection to Google API and opens the Google Calendar API
3. Student selects to edit an event.
4. Student selects the date, time, title, and category (if relevant) for event.
5. Student then gets asked if this event should be repeated, and if yes, how frequently.
6. Event is then saved.

Step-by-Step F: Description

1. Student selects *Calendar*
2. Application establishes connection to Google API and opens the Google Calendar API
3. Student selects to add an event category.
4. Event category pop-up displays student options to fill in.
5. Student then selects a title, and colour scheme for the category.
6. Category is then added.

Step-by-Step G: Description

1. Student selects *Calendar*
2. Application establishes connection to Google API and opens the Google Calendar API
3. Student selects to add a diary entry.
4. Student is displayed the diary section.
5. Student can start typing, or start a voice recording.

Best-case Scenario

1. Peter logs in to the application.
2. Peter selects the *Calendar.*
3. System loads data from *Google Calendar REST* API and presents it to Peter.
4. Peter can proceed to use the *Calendar* as intended.

Worst-case Scenario

1. Sally logs in to the application.
2. Sally selects the calendar option.
3. System fails to contact Google Calendar REST API due to a network error.
4. System tells Sally about the error and tells her to try again later.
5. Sally is disappointed, but now knows what happened.

Alternate Scenarios

1. Thomas logs in to the application.
2. Thomas selects the calendar.
3. System fails to contact Google Calendar REST API due to a network error.
4. System tells Thomas about the error and tells him to try again later.
5. Thomas can't access their calendar.

Vault

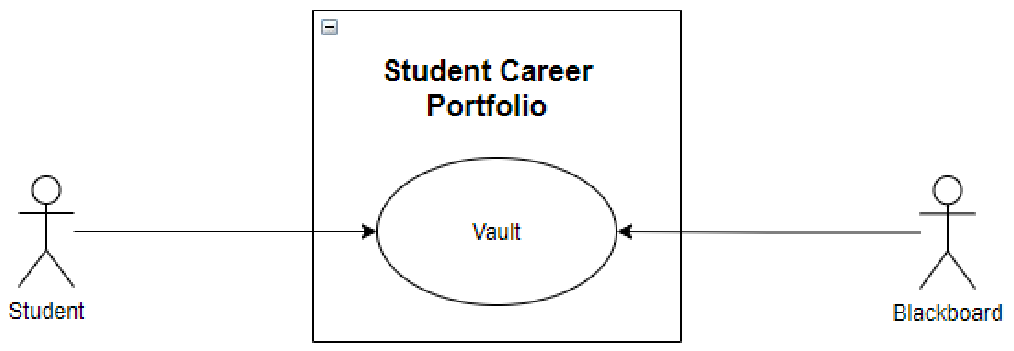


Figure 25: Vault Use Case diagram

Brief Description

The *Vault* use case enables a student to upload/download/delete their past assessment documents and import their marks from Blackboard or enter it in themselves if Blackboard does not have it.

Step-by-Step Description

1. Student selects *Vault*.
2. A list of modules will be shown.
3. Student selects the appropriate module.
4. Student then chooses assessment type, either assignment, test or exam.
5. Student is then taken to the corresponding page.
6. Student can then select to upload a past assessment.
   a. Student gets prompted to select the file they want to upload.
   b. Student selects the document they want to upload.
   c. Student then gives the upload a title and input the mark achieved.
7. Student can select to download a past assessment.
   d. Student selects the assessment they want to download.
   e. Student gets prompted to select where to save it to.
   f. File is downloaded and saved to selected destination.
8. Student can Delete a past assessment.
   g. Student selects assessment to be deleted.
   h. Student gets prompted “Are you sure?”
   i. Student selects Yes or No.
   j. IF Yes was selected, the file is removed and deleted.
   k. IF No was selected, the prompt goes away, and the file remains.
9. Student can choose to Import marks and documents from Blackboard where appropriate.
   l. Student ticks the checkbox to import assessments for that module from Blackboard
   m. Application then sends a request to Blackboard for the module
   n. All available assessment information and marks are then stored categorically.

Best-case Scenario

1. Peter opens the application.
2. Peter selects the vault.
3. Peter selects the module that he wants.
4. Peter selects the assessment type.
5. Peter uploads the files he wanted.
6. Files are uploaded successfully.
7. Files are shown under module.

Worst-case Scenario

1. Sally opens the application.
2. Sally selects the vault.
3. Sally selects the module that they want.
4. Sally selects the assessment type.
5. Sally uploads the files they wanted.
6. Files are uploaded unsuccessfully due to a network error.
7. System tells Sally to try again later.

Alternate Scenarios

1. Thomas opens the application.
2. Thomas selects the vault.
3. Thomas selects the module that they want.
4. Thomas selects the assessment type.
5. Thomas uploads the files they wanted.
6. Files are uploaded successfully.
7. Files are not loaded from the system due to a network error.
8. System lets Thomas know that the files could not be retrieved and tells him to try again later.

Software Project Management Plan

1. Overview

1.1 Project summary

1.1.1 Purpose, scope, and objectives

The objective of this project is to develop a system to help students keep track of their studies and record everything they do at the university.

1.1.2 Assumptions and constraints

- The deadline must be met.
- The product must be developed within the given budget.
- The product must be user-friendly

1.1.3 Project deliverables

The project must be completed **900 hours** after the project commences, that is:

- The system software
- Integration with blackboard
- User cloud storage

1.1.4 Schedule and budget summary

Requirements workflow

- Five team members, 22 hours at R180 per person/per hour

Analysis workflow

- Five team members, 39 hours at R180 per person/per hour

Design Workflow

- Five team members, 40 hours at R180 per person/per hour

Implementation workflow

- Five team members, 800 hours at R180 per person/per hour

Testing workflow

- Done throughout other workflows and accounted for in their hours

Total development time: ~901 hours

Total cost estimated at R1 060 900

2. Reference materials

Schach, S.R., 2011. *Object-Oriented and Classical Software Engineering*. 8th ed. New York: McGraw-Hill.

3. Definitions and acronyms

Client – The University of the Free State

System software – The software responsible for executing the overall software product, including:

4. Project organization

4.1 External interfaces

The external software that the system will interact with is built by Blackboard, Inc. The modules that will handle the interfacing with Blackboard's modules will be built by the in-house development team, namely DC McDonald, ML Gomes, AM Schoombie, WR Boltman, and JC Venter.

Client meetings will be done monthly by DC McDonald, ML Gomes, AM Schoombie, WR Boltman, and JC Venter to ensure client satisfaction throughout the project and to ensure project meets all the client demands.

All components developed will be integrated with each other by the development team, consisting of DC McDonald, ML Gomes, AM Schoombie, WR Boltman, and JC Venter.

4.2 Internal structure

The internal development team will consist of DC McDonald, ML Gomes, WR Boltman, and JC Venter. This team will be managed by AM Schoombie.

Client interaction team will consist of DC McDonald, ML Gomes, AM Schoombie, WR Boltman, and JC Venter.

The module integration team will consist of DC McDonald, ML Gomes, AM Schoombie, WR Boltman, and JC Venter.

Testing will be done throughout by each developer, testing the other's work.

4.3 Roles and responsibilities

All workflows will be managed by AM Schoombie. Furthermore, the entirety of the team consisting of DC McDonald, ML Gomes, WR Boltman, AM Schoombie and JC Venter will be responsible for all artefacts of the life cycle workflows.

- DC McDonald will be responsible for all artefacts pertaining to the calendar and fault functions of the system, including interfacing an importing module from Blackboard.
- AM Schoombie will be responsible for customization modules, and Study Tips modules, building the interface module linking the system with CTL's study tips, as well as the integration of the Pomodoro timer.
- ML Gomes will be responsible for all artefacts pertaining to the search function, the login functionality, study material functionality (including uploading and viewing), and adding the modules.
- JC Venter will be responsible for all artefacts pertaining to notes, as well as the flashcard deck functionality. Furthermore, he will be responsible for the collab functionality, building the chat platform between students and integrating it with the rest of the system.
- WR Boltman will be responsible for the statistics and uploading marks functionality. He will integrate these two functionalities to communicate with each other as well as the main screen user interface modules.

Testing will be done throughout the development process to ensure that quality will continuously be up to standard. It is each member's own responsibility to ensure that their modules can integrate with the other team members' modules.

5. Managerial process plans

5.1 Start-up plan

5.1.1 Estimation Plan

The estimation was done using *expert judgement by analogy,* where the team of software developers compared previous similar projects and combined estimates to an agreed upon amount of R 850 000. This estimate was reached through a consensus based on time of each artefact's man-hours required to build, considering all internal costs.

5.1.2 Staffing plan

The entire team of DC McDonald, ML Gomes, AM Schoombie, WR Boltman, and JC Venter. Will be involved throughout all workflows of the project.

The team of 5 will be managed by AM Schoombie, who will ensure that the project will meet all deadlines within budget.

The rest of the team, consisting of DC McDonald, ML Gomes, AM Schoombie, WR BOLTMAN, and JC Venter. Will ensure that for all their assigned modules, the analysis, design, and implantation workflows are correct and well-integrated.

5.1.3 Resource Acquisition plan

All Hardware required to implement the software will be acquired from *Wootware,* this includes all computers that will be used during the development of the application. Testing will be done on mobile devices acquired from *Mobicell,* iOS, *Windows* and *Android* devices will be used.

Cloud storage will be utilizing the University of the Free State's server network, to keep costs to a minimum.

Software used will be:

- *Visual Studio 2019* Acquired from Microsoft. Each team member will Enterprise version.
- *SQL Server 2014 Management Studio* to set up all database operations.
- *Blackboard, Inc.'s* web-based virtual learning environment for integration of the system.

5.2 Work plan

5.2.1 Activity Schedule

Requirements – Five members for a total of 22 hours. Members meets the client and elicits requirements using interviews. The requirement artefacts were then reviewed to ensure client satisfaction.

Analysis workflow – Five members for a total of 39 hours. The Analysis workflow is refined to a specification document that is sent for client approval. Upon client confirmation, deposit amount (50% of agreed upon amount) is paid to the development company and design workflow commences.

Design Workflow – Five members for a total of 40 hours. Each developer designs their module's artefacts and sets up test cases, to ensure that the module will work as intended. Each member will review the other member's test cases to ensure that all test cases are up to standard. Each member will update their own module's documentation. At the end of the design workflow, all documentation will be combined in a single document.

Implementation Workflow – Five members for a total of an estimated 1200 hours. Each member will implement their module and test it using their custom test cases. Each member will then ensure that the other members' test cases return the correct results. Implementation will be compared with initial specification document.

Testing Workflow – Testing will be done continuously throughout the entire SDLC and do not have a separate activity schedule. All hours mentioned above include testing.

5.2.2 Resource Allocation

Each team member will work independently on a separate workstation with their own copies of all the software mentioned in the resources section.

Only a single version of Blackboard is required to test integration, as members will run test cases on the current online version utilized by the University of the Free State.

Mobile phones will be utilized and given to each member during the implementation workflow to ensure modules appear correctly on all mobile formats.

5.2.3 Budget Allocation

Requirements workflow	R 19 800
Analysis workflow	R 35 100
Design workflow	R 36 000
Implementation workflow	R 720 000
Resources	R 250 000
General Expenses	R 62 060
	___________
Total	R 1 122 960

5.3 Quality Control plan

Testing will be continuously done throughout all workflows; the specification document will serve as a grounds for functionality testing. The entire team will also be responsible for ensuring that their modules will function correctly against all their test cases.

AW Schoombie will be responsible for the overall quality of the project as team leader. She will review all test case results throughout the project.

5.4 Risk management plan

Backups will be made throughout the project development, where each version of the software will enter a testing phase before it is accepted as a new branch. All previous versions will also be stored on a version control platform, such as GitHub to ensure a rollback is possible, should a critical failure be identified in a later version.

Extra workstations will be obtained to ensure the team can function on an optimal level, should a workstation give in. This will minimize turnaround time to get a team member set up on a new environment.

5.5 Project close-out plan
Currently no other software projects are in development, thus no staff reassignments will be required after the completion of the project. After project completion, the entire team will focus on customer support to ensure optimal customer satisfaction.

6. Technical process plans
6.1 Process model
The Unified Process methodology and Iterative-and-Incremental Life-cycle model will be used.

6.2 Methods, tools, and techniques
The workflows of the Unified Process will be applied. The product will be implemented in Java, C#, and Swift. The database and database management system will be implemented using Microsoft SQL Server.

6.3 Infrastructure plan
The project will be developed using Visual Studio 2019, and SQL Server 2014. Running under Windows on personal computers, and iOS and Android on personal mobile devices. The system software will run on Linux OS on a server located within ICT services.

6.4 Product acceptance plan
Acceptance of the product by the client will be achieved by following the steps of the Unified Process.

7. Supporting process plans
7.1 Configuration management plan
Git will be used throughout for all artifacts.

7.2 Testing plan
Testing will be conducted according to the activities of the testing workflow of the Unified Process.

7.3 Documentation plan
Project documentation will be produced as specified in the Unified Process.

7.4 Quality assurance plan
AM Schoombie, DC McDonald, ML Gomes, WR Boltman, and JC Venter will each test their assigned artifacts and each other’s code (unit testing). Integration and product testing will be conducted by all five development team members. AM Schoombie will also oversee the integration and product testing to ensure adherence to the specifications document and compliance with the quality standards and constraints outlined therein.

7.5 Reviews and audits plan
Not applicable.

7.6 Problem resolution plan
Any major problems will be reported during the daily meetings.

7.7 Subcontractor management plan
Not applicable.

7.8 Process improvement plan
Not applicable.

8. Additional plans

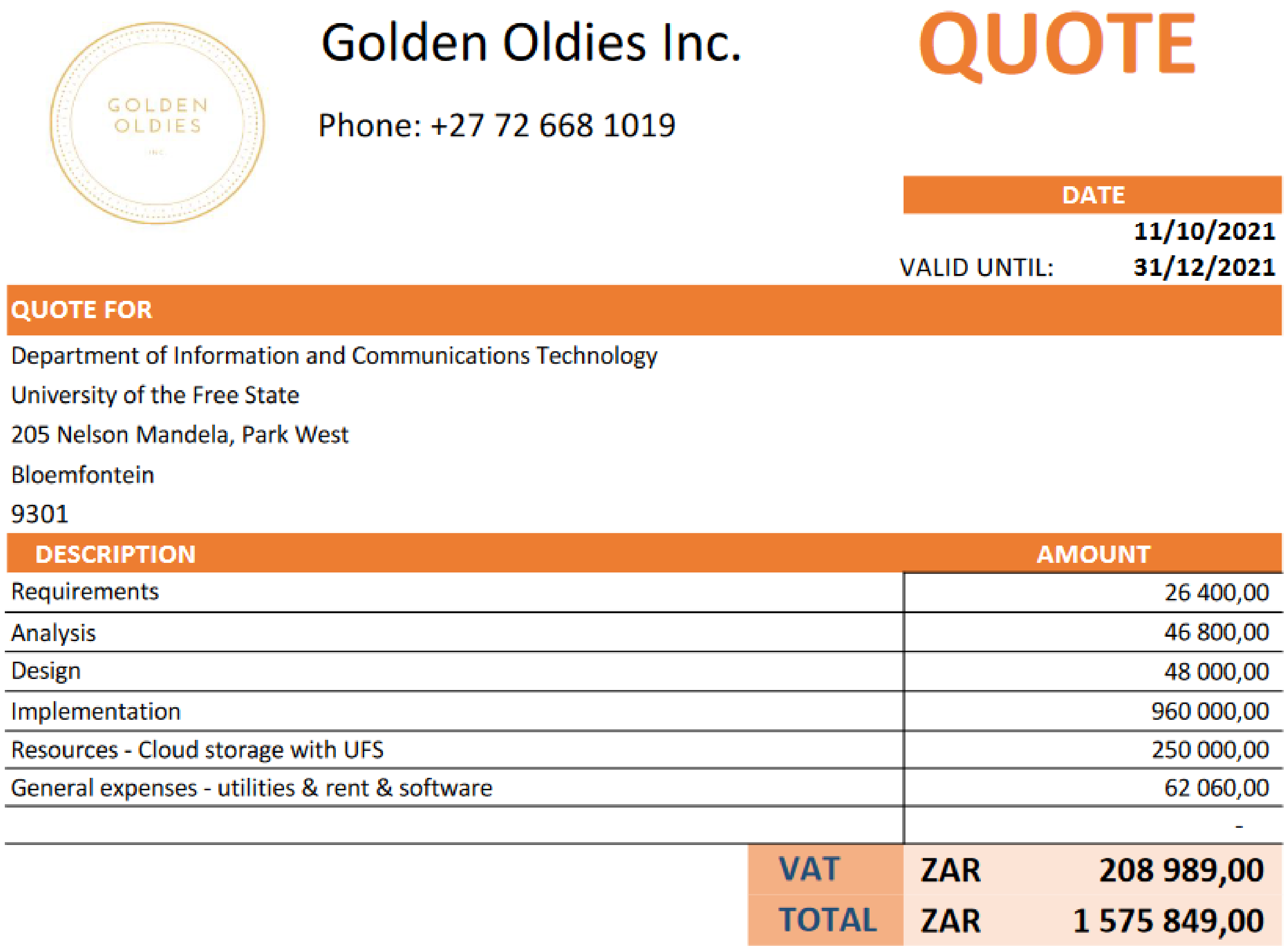
8.1 Security

Credentials of a valid current student will be required to log in and access the system and its functionalities.

8.2 Maintenance

Corrective maintenance on the software components will be performed by the team at no cost for 6 months.

Quotation



Golden Oldies Inc.

Phone: +27 72 668 1019

QUOTE

	DATE
	11/10/2021
VALID UNTIL:	**31/12/2021**

QUOTE FOR

Department of Information and Communications Technology
University of the Free State
205 Nelson Mandela, Park West
Bloemfontein
9301

DESCRIPTION	AMOUNT
Requirements	26 400,00
Analysis	46 800,00
Design	48 000,00
Implementation	960 000,00
Resources - Cloud storage with UFS	250 000,00
General expenses - utilities & rent & software	62 060,00
	-

VAT	**ZAR**	**208 989,00**
TOTAL	**ZAR**	**1 575 849,00**

Figure 26: Quotation

Noun Extraction

The student logs in to his portfolio. He can select which year's modules to display on the dashboard. The student can search for specific content and a daily message is also displayed on the dashboard. From the dashboard, the student can navigate to statistics of his marks, his calendar, study tips, a Pomodoro timer and themes for the portfolio. For each module, the student can access study material, notes, previous assessments, marks, flash cards and collaborative groups.

Classes:

- **Portfolio** (Includes functions: search content, add modules to a year, delete modules from a year, edit modules in a year, add StudyTipsArticles, delete StudyTipArticles
  Includes properties: list<Year>, DailyMessage, Calendar, list<Theme>, CalculateStatistics, PomodoroTimer, list<StudyTipsArticles>)

- **DailyMessage** (includes properties: message itself, all settings selected by user – see use cases and CRC cards)
- **Calendar** (includes properties and functions: see use cases and CRC cards)
- **Static PomodoroTimer** (includes properties and functions: see use cases and CRC cards)
- **Theme** (includes properties and functions: see use cases and CRC cards)
- **StudyTipArticles** (includes properties: see use cases and CRC cards)
- **Year** (Includes properties: list<Module>, CalculateStatistics)
- **Static CalculateStatistics** (includes functions: all calculations on marks – see use cases and CRC cards)

- **Module** (includes properties: list<StudyMaterial>, list<Notes>, list<Assessment>, list<Marks>, list<Flashcards>, list<Group>)
- **StudyMaterial** (includes properties and functions: see use cases and CRC cards)
- **Notes** (includes properties and functions: see use cases and CRC cards)
- **Assessment** (includes properties and functions: see use cases and CRC cards)
- **Marks** (includes properties and functions: see use cases and CRC cards)
- **FlashCards** (includes properties: name of flashcard deck, list of card names, includes functions: see use cases and CRC cards)
- **Group** (includes properties and functions: see use cases and CRC cards)

CRC Cards

Statistics

Class: Statistics	
Responsibilities	**Collaborators**
• View average marks. • Calculate average marks over specific period. • Separate average marks depending on selected time period.	• Module Marks

Calendar

Class: Calendar	
Responsibilities	**Collaborators**
• To-do list (add, edit, delete) • Reminders • Events • Monthly view • Daily view • Test/Assignment/Exam dates	• Portfolio

Flashcard

Class: Flashcard	
Responsibilities	**Collaborators**
• Create, edit, delete cards • Add front and back info on cards	• FlashcardStack

ModuleMarks

Class: ModuleMarks	
Responsibilities	**Collaborators**
• Upload, edit, and delete marks for assignments/exams/test. • Calculate average module mark.	• Module

PomodoroTimer

Class: PomodoroTimer	
Responsibilities	**Collaborators**
• List<Topic Names> • Add topic names • Start timer • Reset the timer & remove all topic names • Timer (4 repetitions of 25 minutes study and 5 minute break. Then 25 minute break. Then repeat whole process again until all topics covered)	• Portfolio

Note

Class: Note	
Responsibilities	**Collaborators**
• Upload notes • Add, edit, rename, delete notes • View notes • Playback media notes	• Module

Collaboration

Class: Collaboration	
Responsibilities	**Collaborators**
• Create group • Add other users to group • Block/unblock users • Group call • Share documents	• Module

StudyTipArticle

Class: StudyTipArticles	
Responsibilities	**Collaborators**
• Article Name • Article Date Posted • Article Content • Name of CTL member who uploaded the article	• Portfolio

Portfolio

Class: Portfolio	
Responsibilities	**Collaborators**
• Allow year selection • Let themes know what the user does to allow the themes to unlock.	• Year • Theme

DailyMessage

Class: DailyMessage	
Responsibilities	**Collaborators**
• Message • Enabled/Disabled (does not change daily) • Hidden/Visible • Edit the message • Communicate with Message of the Day server • Update message daily	• Portfolio

Assessment

Class: Assessment	
Responsibilities	**Collaborators**
• Upload, edit, delete previous assessments • Download assessments • Categorise the assessments into relevant modules	• Module

AcademicYear

Class: AcademicYear	
Responsibilities	**Collaborators**
• List all modules under that year • Allow linking to existing modules	• Module

StudyMaterial

Class: Study Material	
Responsibilities	**Collaborators**
• Allow submission of electronic files • Keep electronic files • Allow deletion of electronic files	• Module

Theme

Class: Theme	
Responsibilities	**Collaborators**
• Theme name • Theme status: locked/unlocked • Description of task required to unlock theme • Is theme currently in use? • Font type • Font size • Font colour • Background image • Dashboard colour • Check if task has been completed to unlock this theme & unlock it if so	• Portfolio

FlashcardStack

Class: FlashcardStack	
Responsibilities	**Collaborators**
• Add, rename, delete stacks • Practice stack • Add cards to stack	• Note • Module • Flashcard

Module

Class: Module	
Responsibilities	**Collaborators**
• Module Name • Module Code • List all notes • Add/remove/edit notes • List all study material • Add/remove/edit study material • List assessments • Add/remove/edit assessments • List marks • Add/remove/edit marks • List flashcards • Add/remove/edit flashcards • Link module groups • Add/remove groups	• AcademicYear • Study Material • Notes • Assessment • Marks • Flashcards • Groups

State Chart

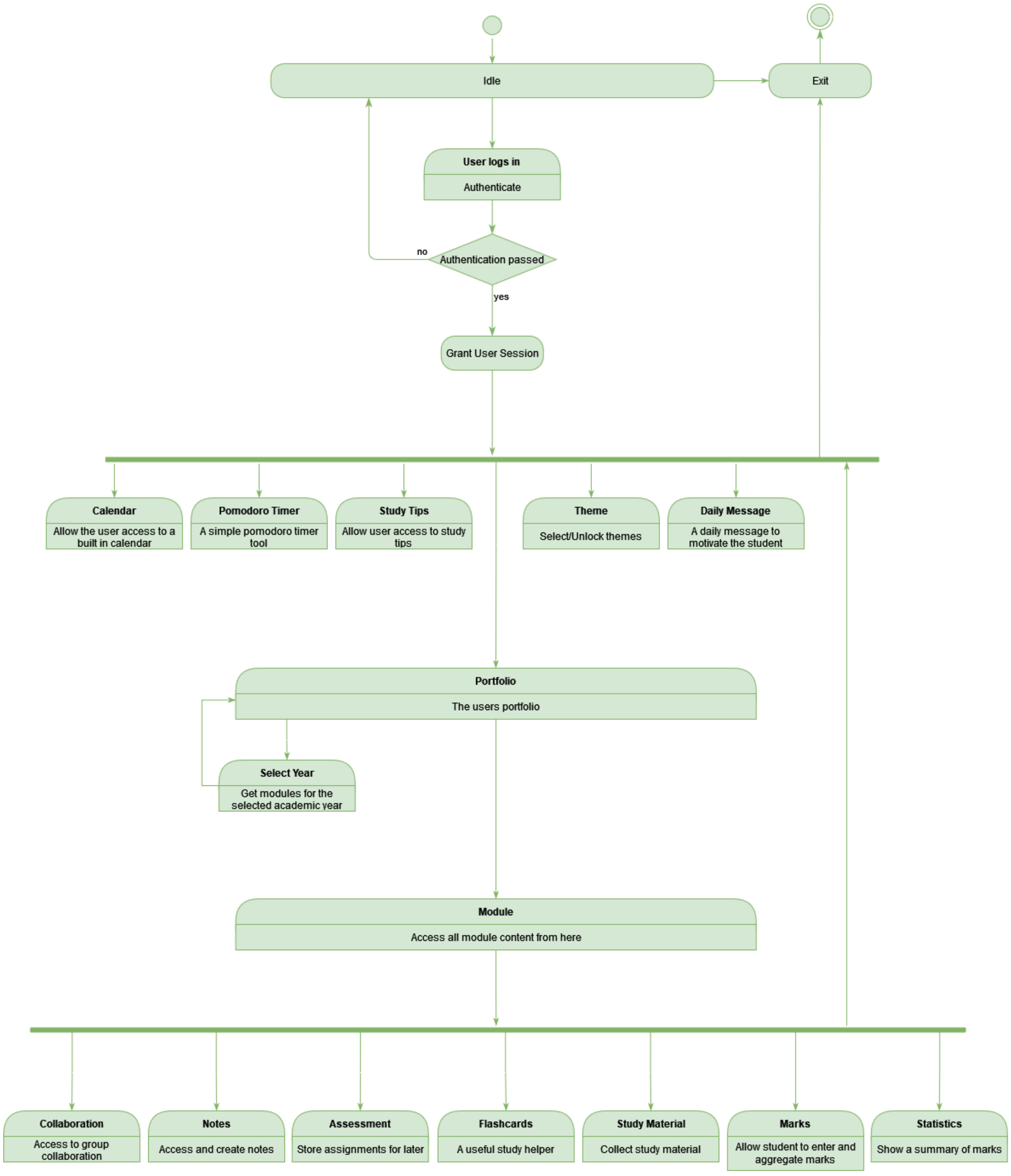


Figure 27: State Chart

Sequence Diagrams

Message of the Day

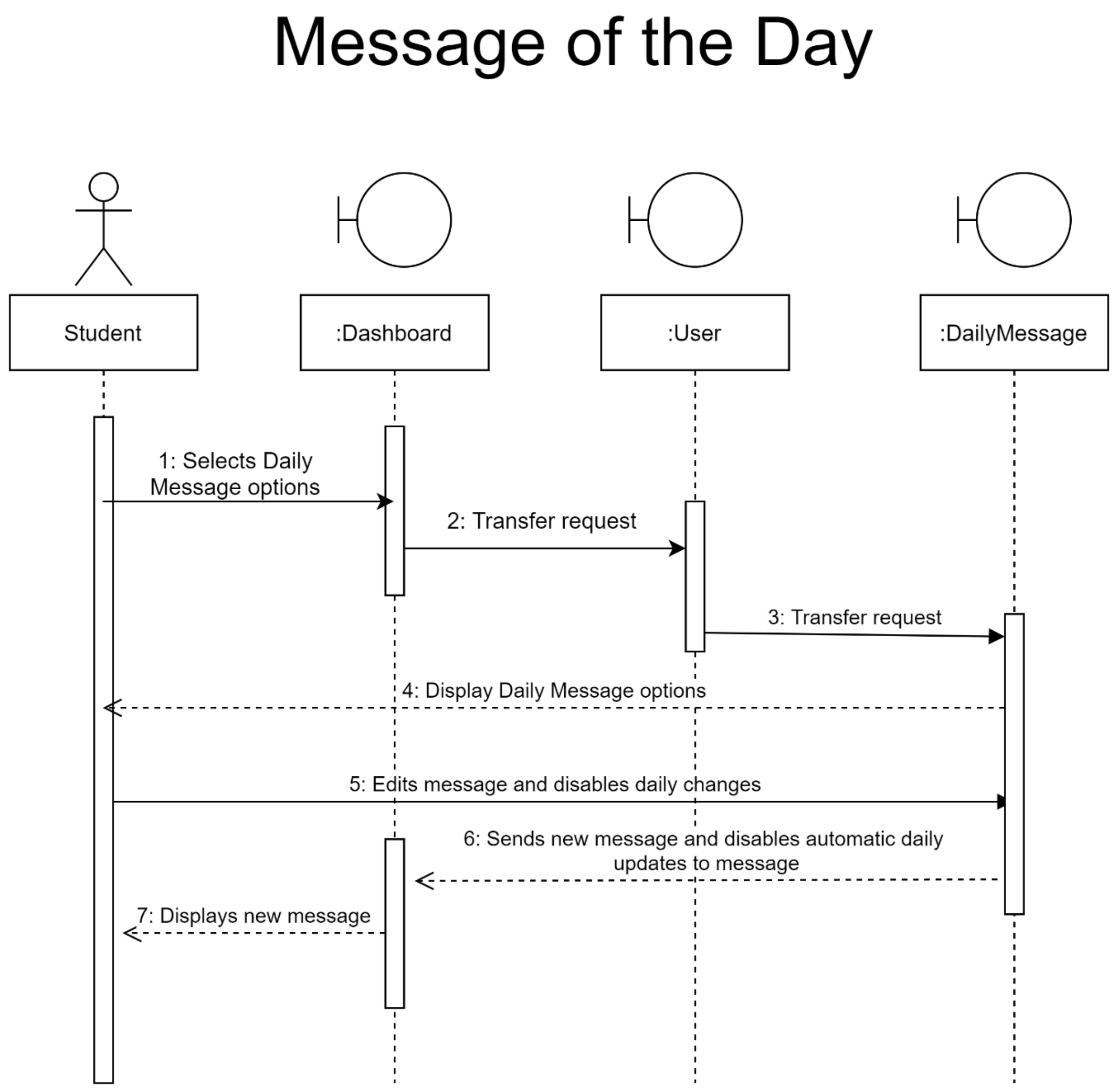

Message of the Day
Student
:Dashboard
:User
:DailyMessage
1: Selects Daily Message options
2: Transfer request
3: Transfer request
4: Display Daily Message options
5: Edits message and disables daily changes
6: Sends new message and disables automatic daily updates to message
7: Displays new message

Figure 28: Message of the Day Sequence Diagram

Themes

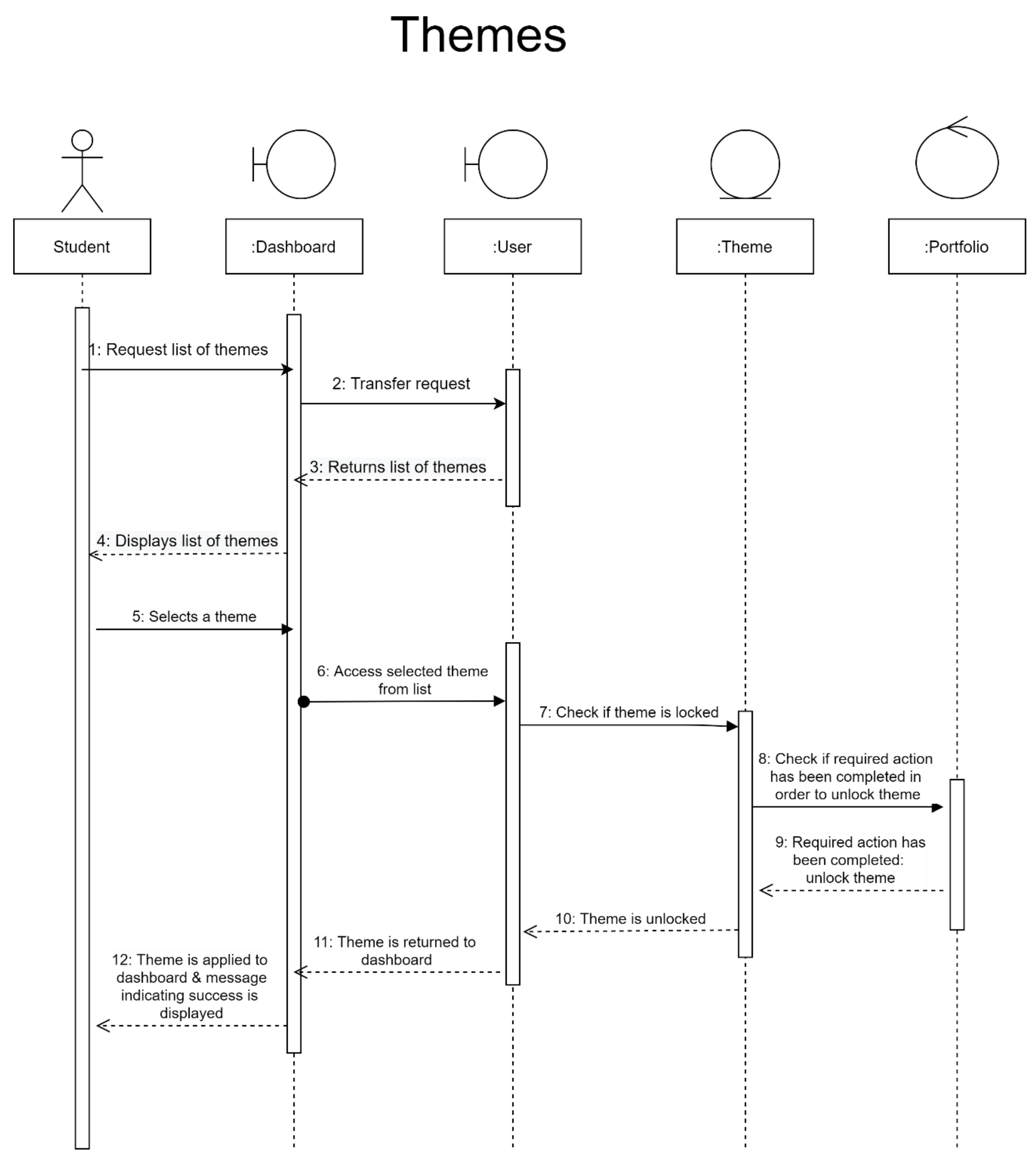

Themes
Student
:Dashboard
:User
:Theme
:Portfolio
1: Request list of themes
2: Transfer request
3: Returns list of themes
4: Displays list of themes
5: Selects a theme
6: Access selected theme from list
7: Check if theme is locked
8: Check if required action has been completed in order to unlock theme
9: Required action has been completed: unlock theme
10: Theme is unlocked
11: Theme is returned to dashboard
12: Theme is applied to dashboard & message indicating success is displayed

Figure 29: Theme Sequence Diagram

Study Tips

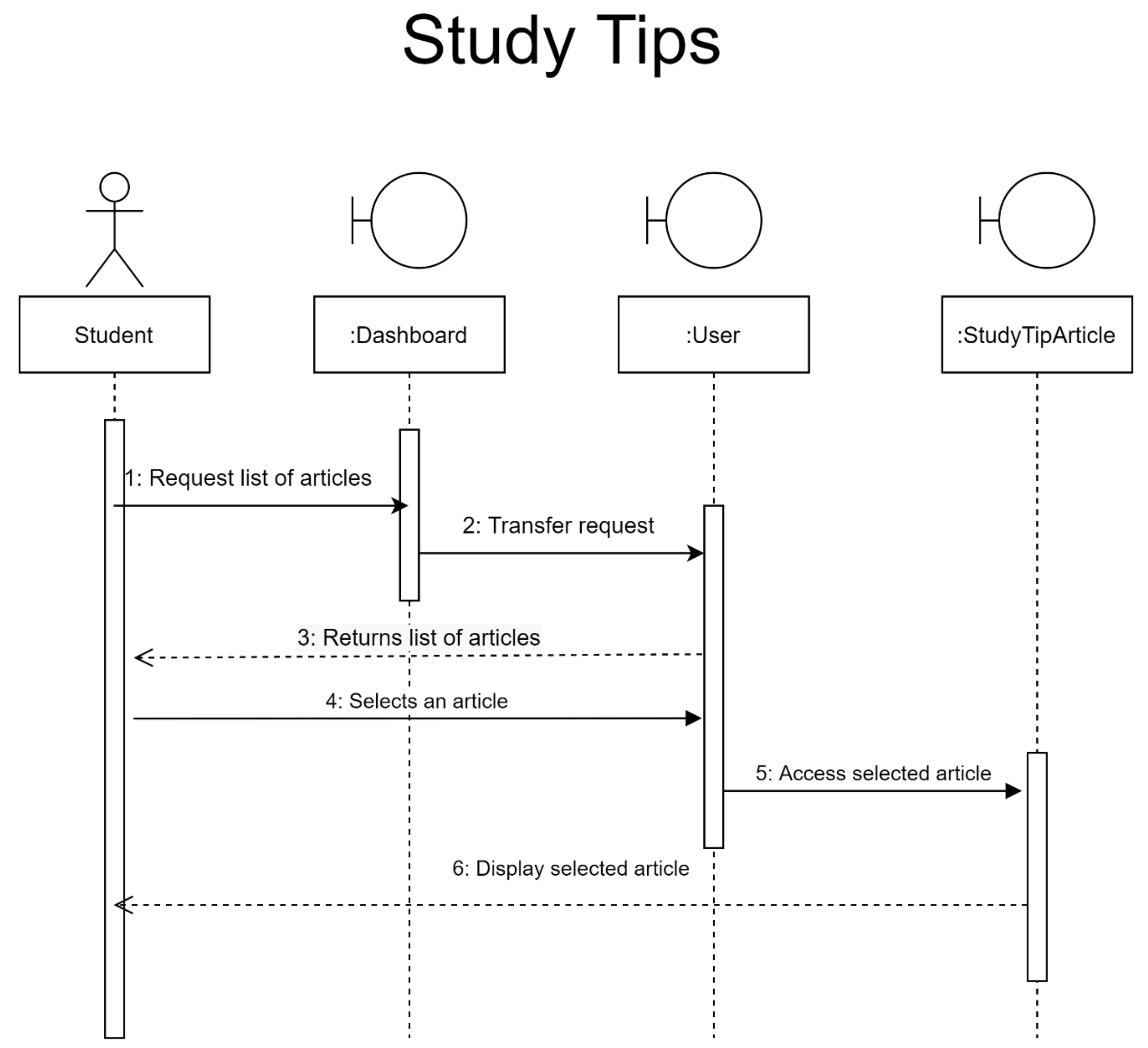
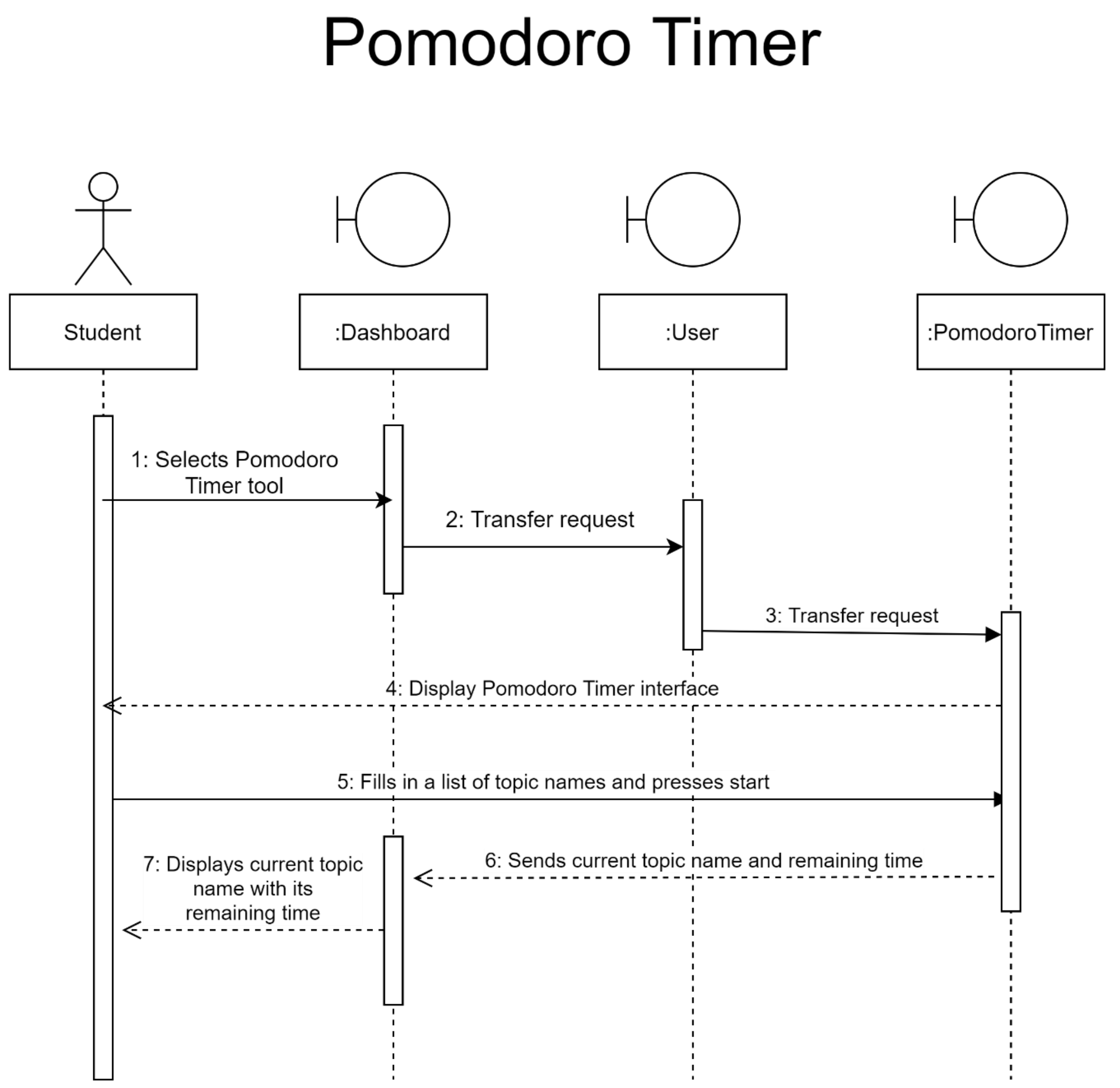

Study Tips
Student
:Dashboard
:User
:StudyTipArticle
1: Request list of articles
2: Transfer request
3: Returns list of articles
4: Selects an article
5: Access selected article
6: Display selected article

Figure 30: Study Tips Sequence Diagram

Pomodoro Timer



Pomodoro Timer
Student
:Dashboard
:User
:PomodoroTimer
1: Selects Pomodoro Timer tool
2: Transfer request
3: Transfer request
4: Display Pomodoro Timer interface
5: Fills in a list of topic names and presses start
6: Sends current topic name and remaining time
7: Displays current topic name with its remaining time

Figure 31: Pomodoro Timer Sequence Diagram

Calendar

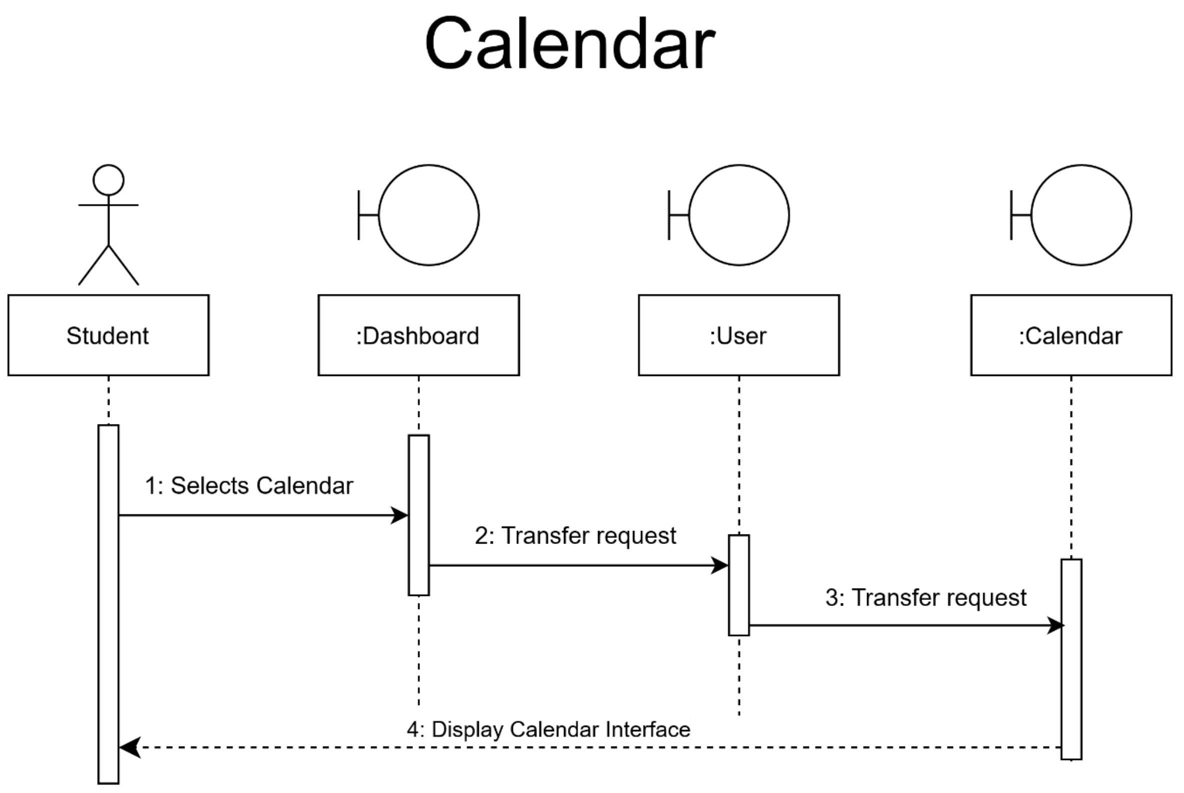


Figure 32: Calender Sequence Diagram

Assessment

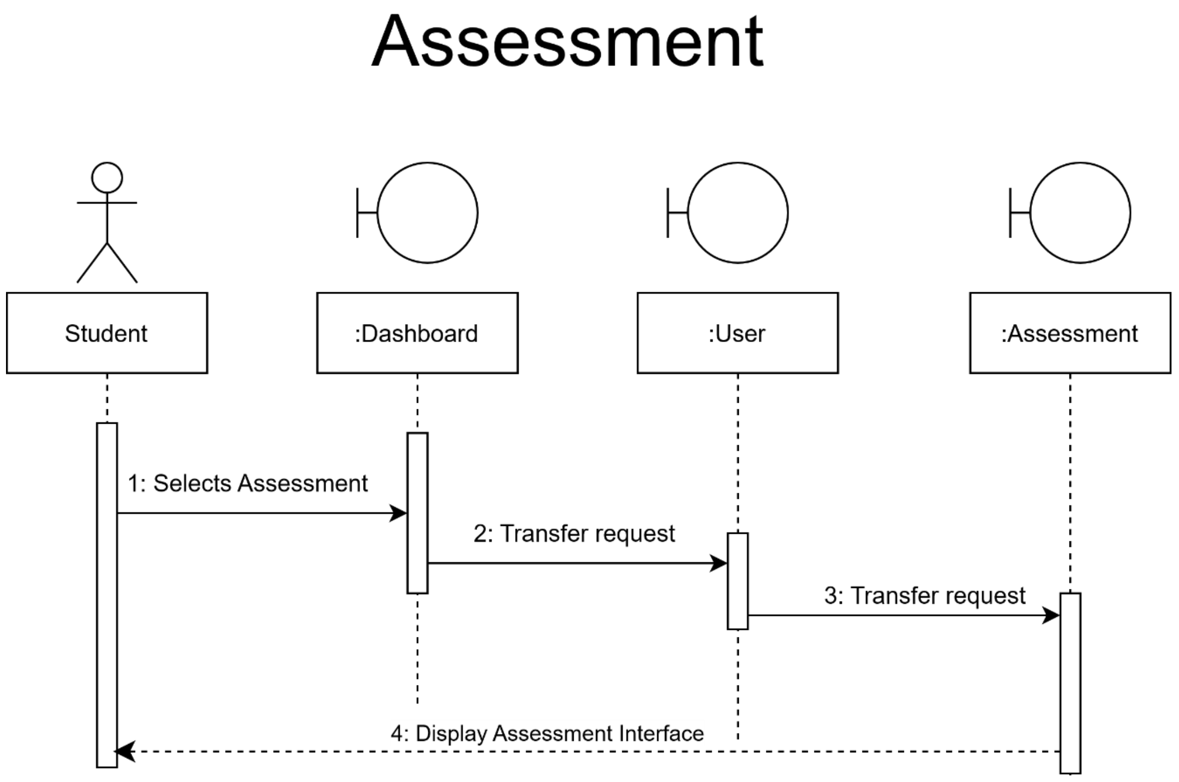


Figure 33: Assessment Sequence Diagram

Note

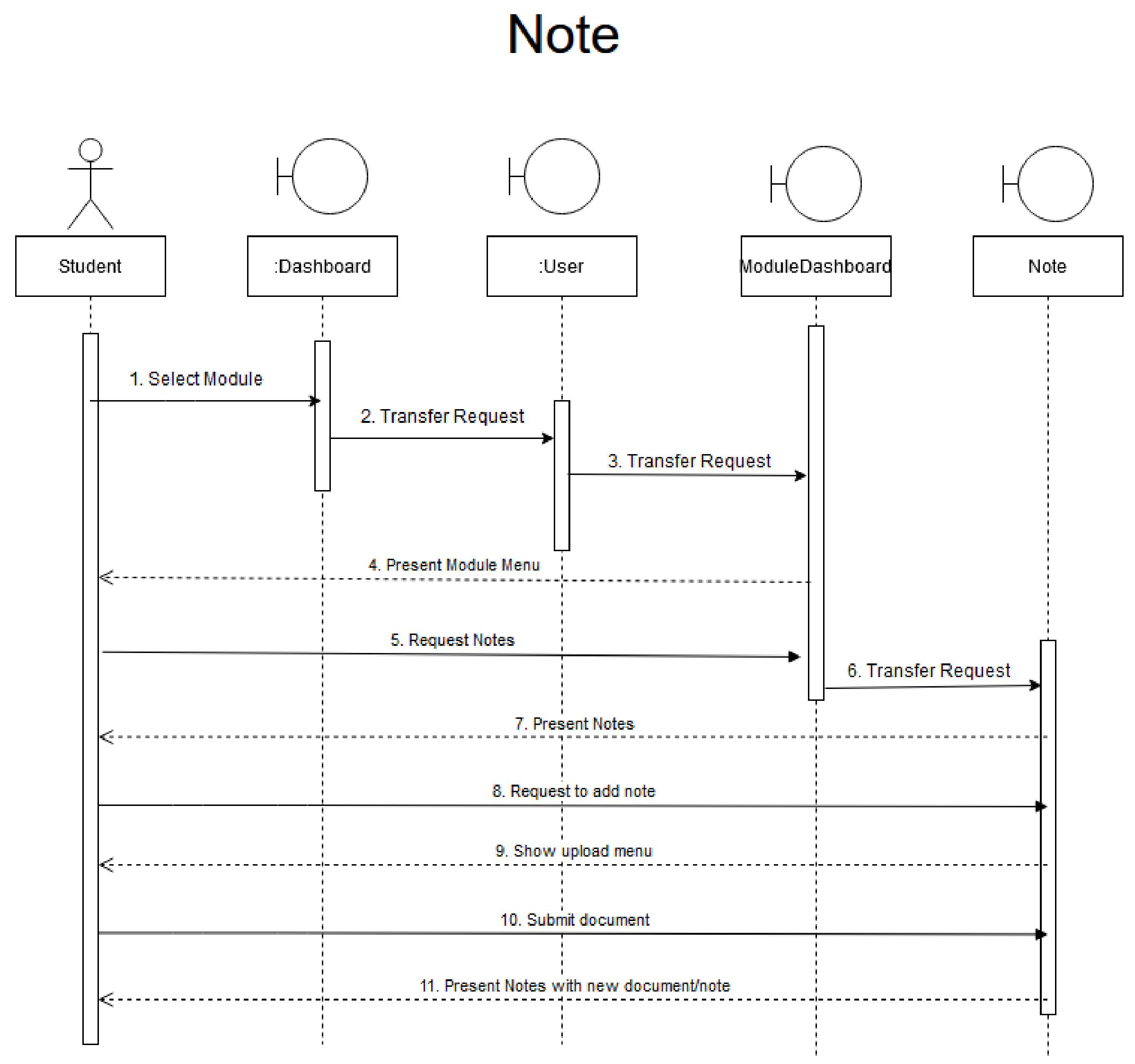


Figure 34: Notes Sequence Diagram

Flashcard

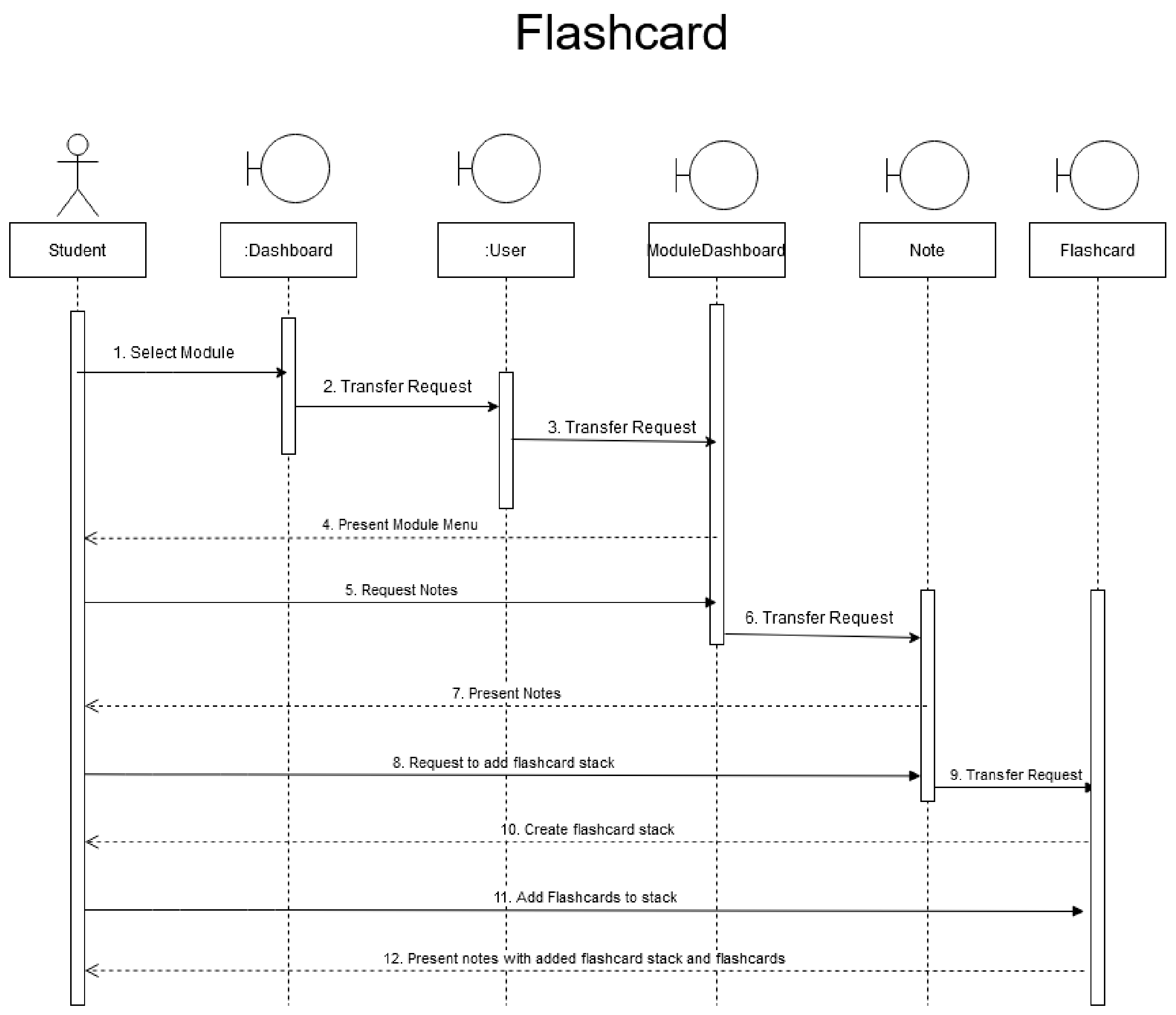


Figure 35: Flashcard Sequence Diagram

Collaboration

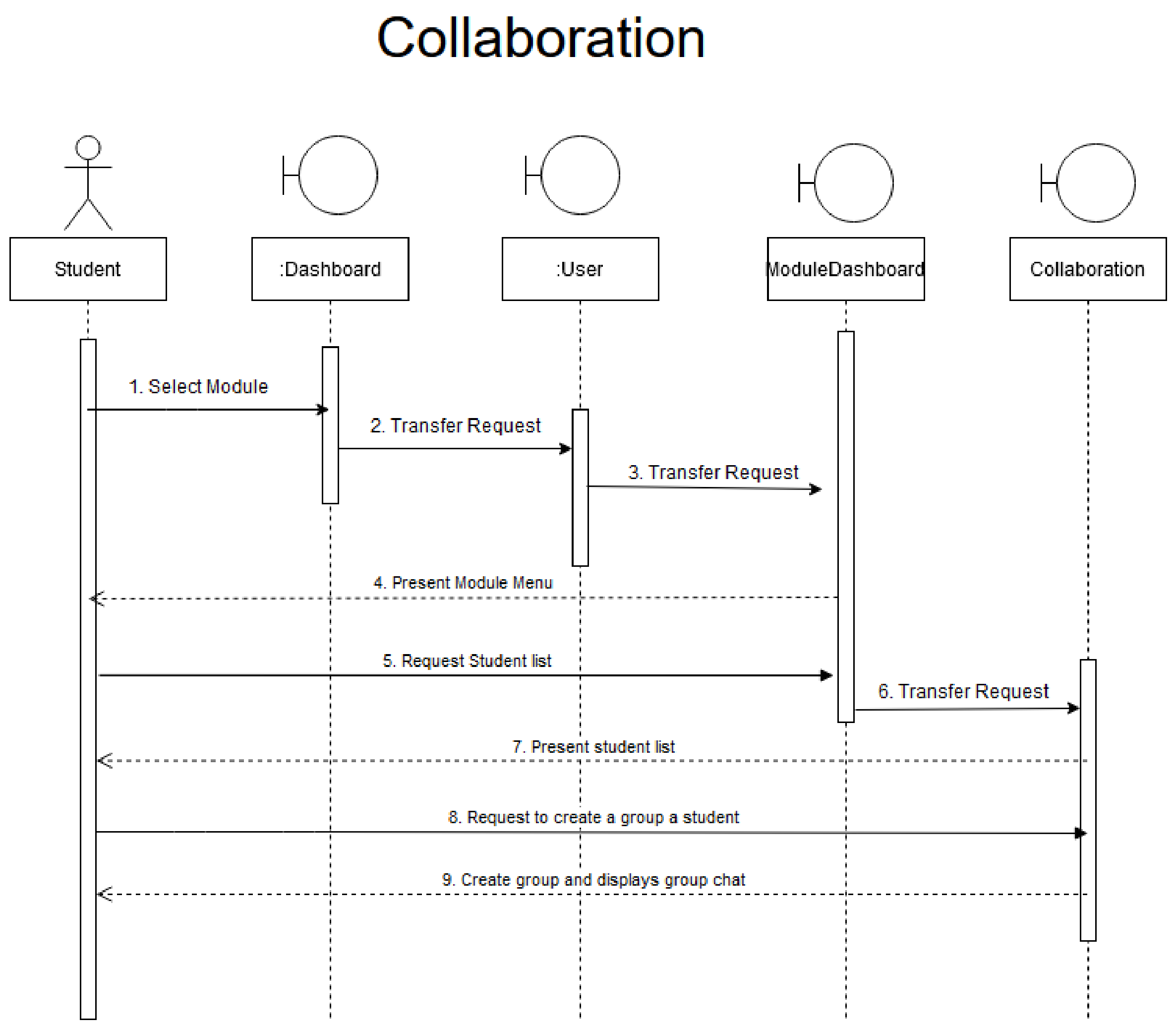
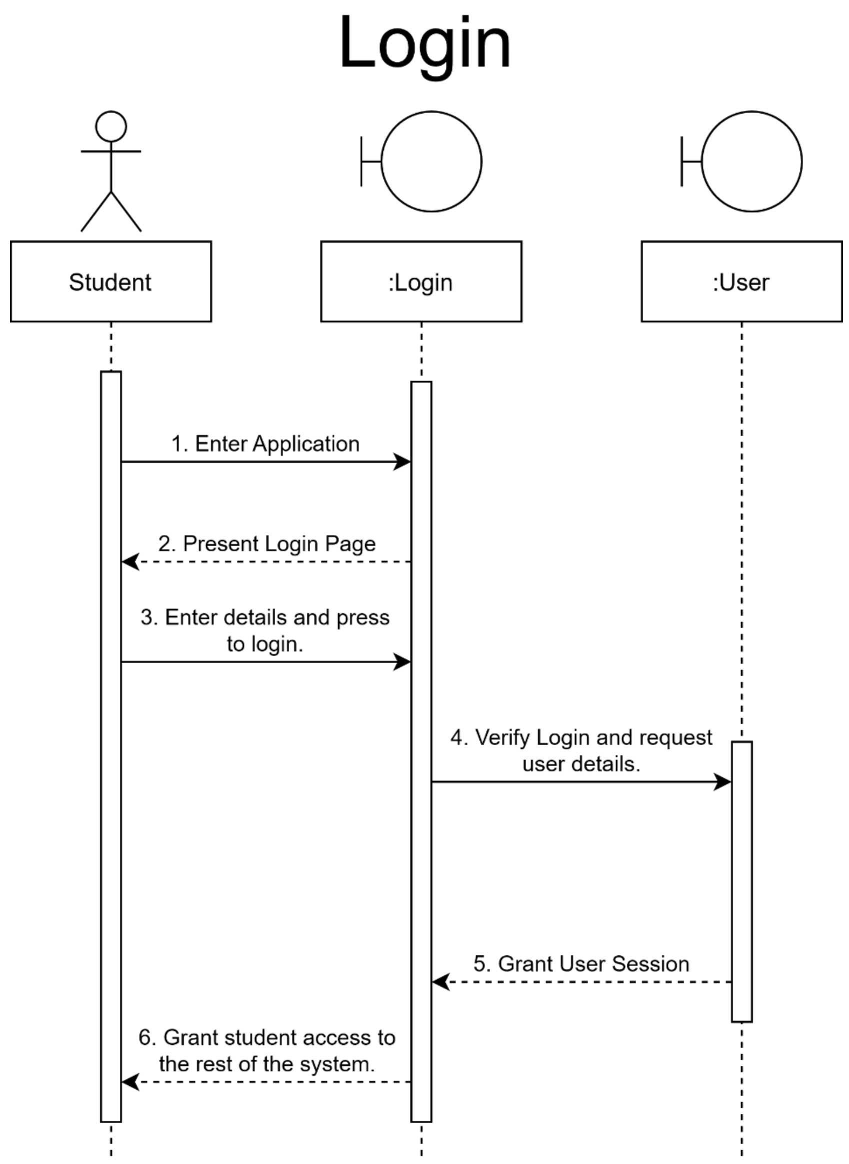


Figure 36: Collaboration Sequence Diagram

Login



Login
Student
:Login
:User
1. Enter Application
2. Present Login Page
3. Enter details and press to login.
4. Verify Login and request user details.
5. Grant User Session
6. Grant student access to the rest of the system.

Figure 37: Login Sequence Diagram

Search

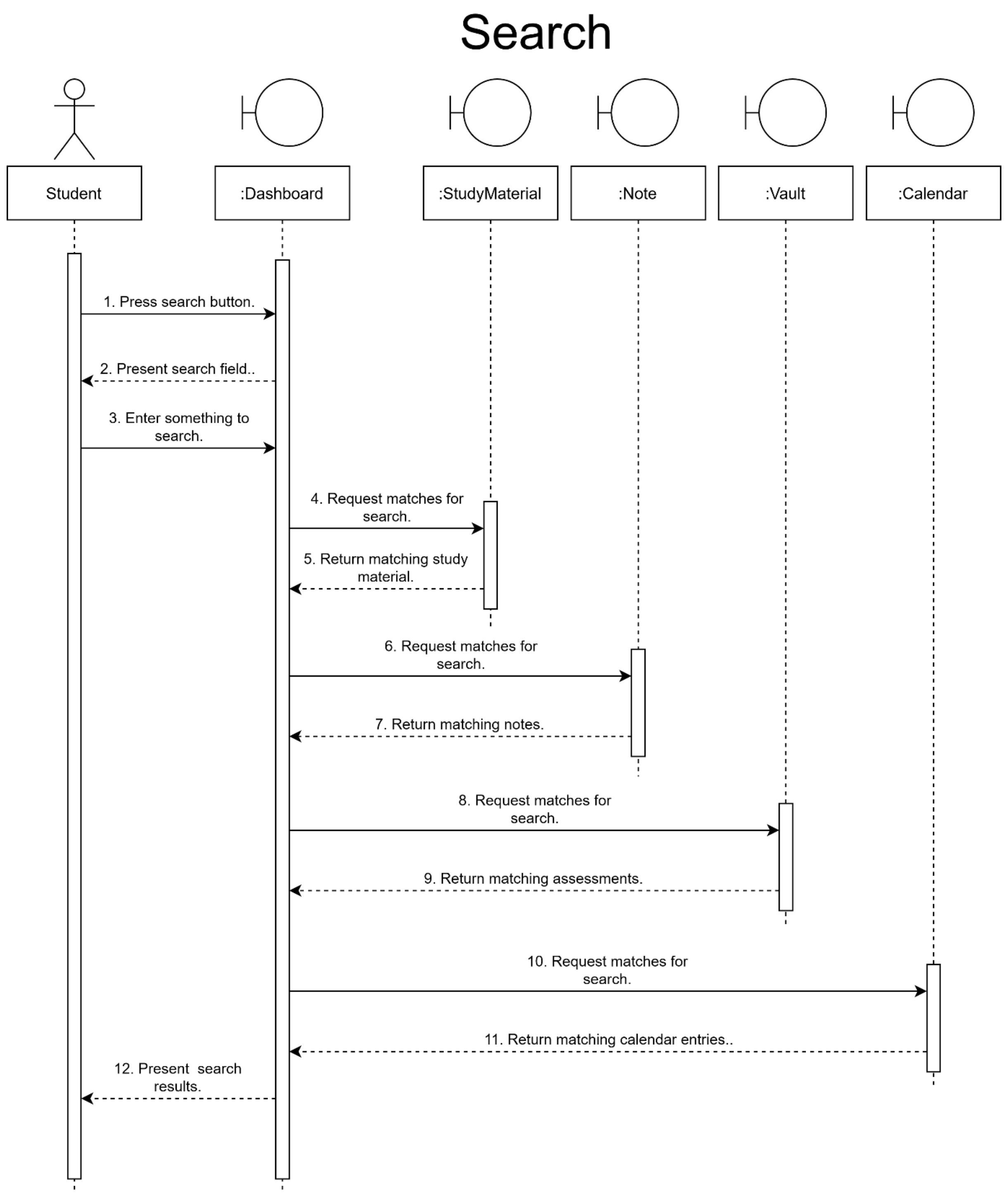

Search
Student
:Dashboard
:StudyMaterial
:Note
:Vault
:Calendar
1. Press search button.
2. Present search field..
3. Enter something to search.
4. Request matches for search.
5. Return matching study material.
6. Request matches for search.
7. Return matching notes.
8. Request matches for search.
9. Return matching assessments.
10. Request matches for search.
11. Return matching calendar entries..
12. Present search results.

Figure 38: Search Sequence Diagram

Study Material

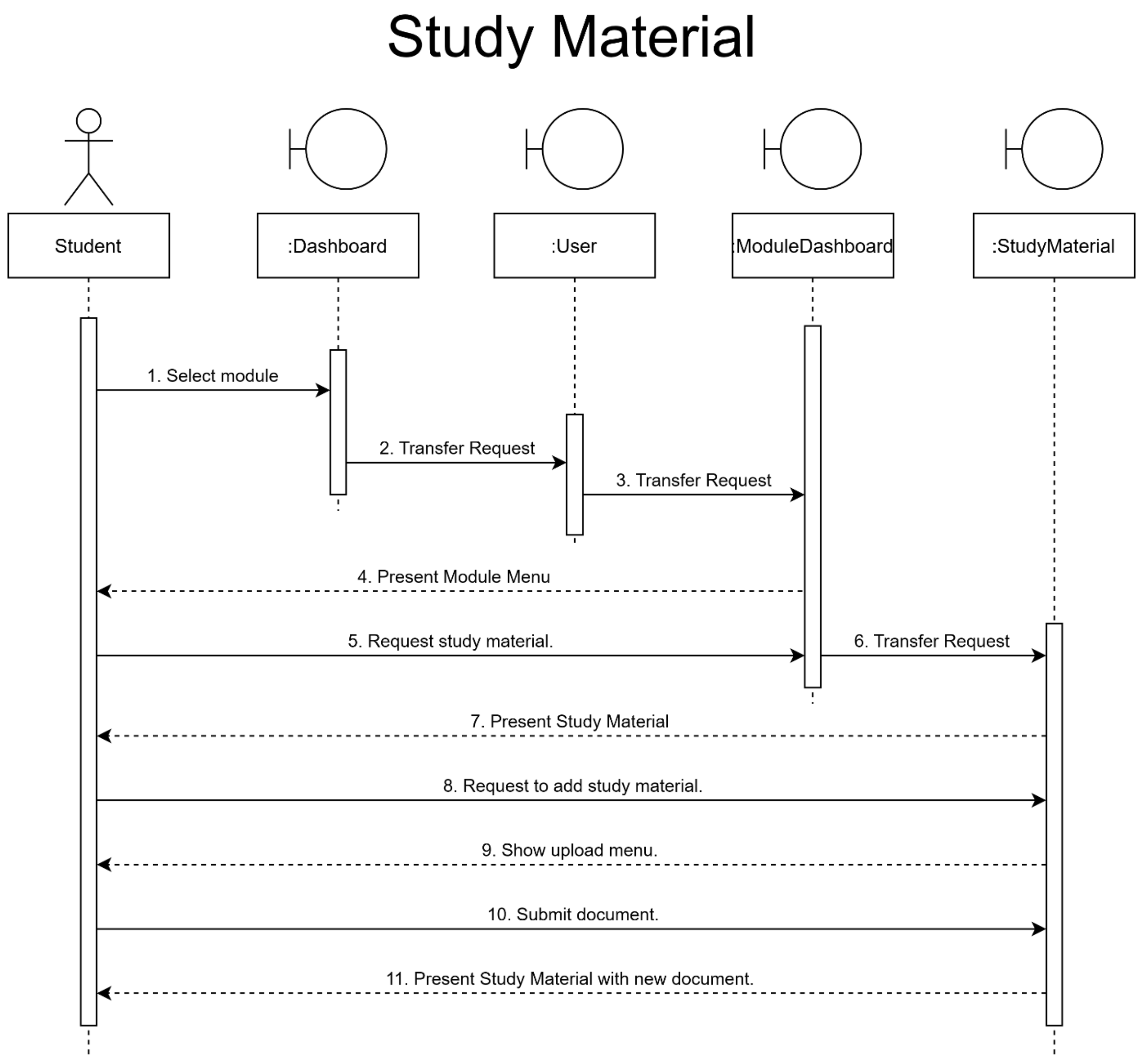


Figure 39: Study Material Sequence Diagram

Module

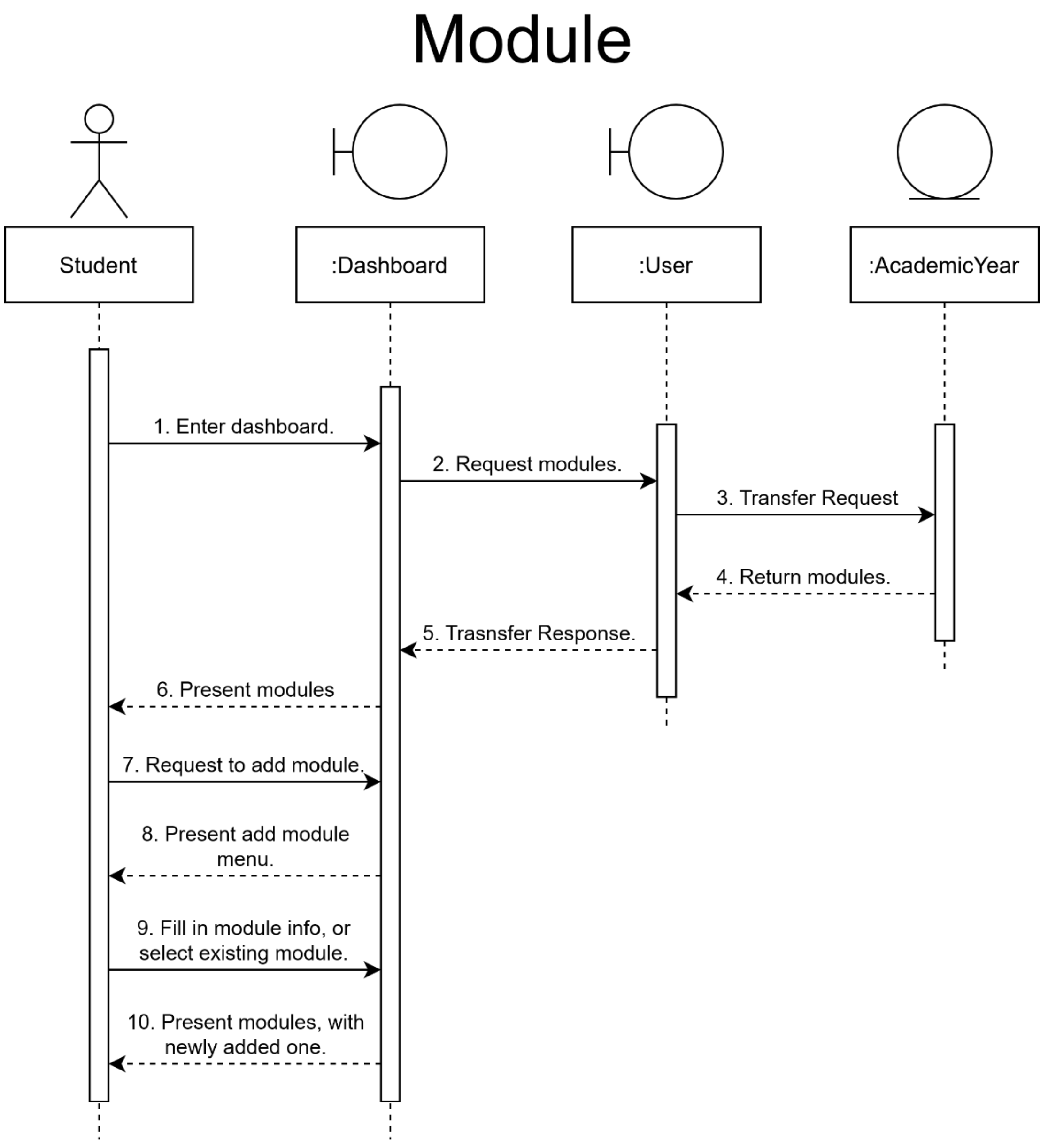


Figure 40: Module Sequence Diagram

Statistics

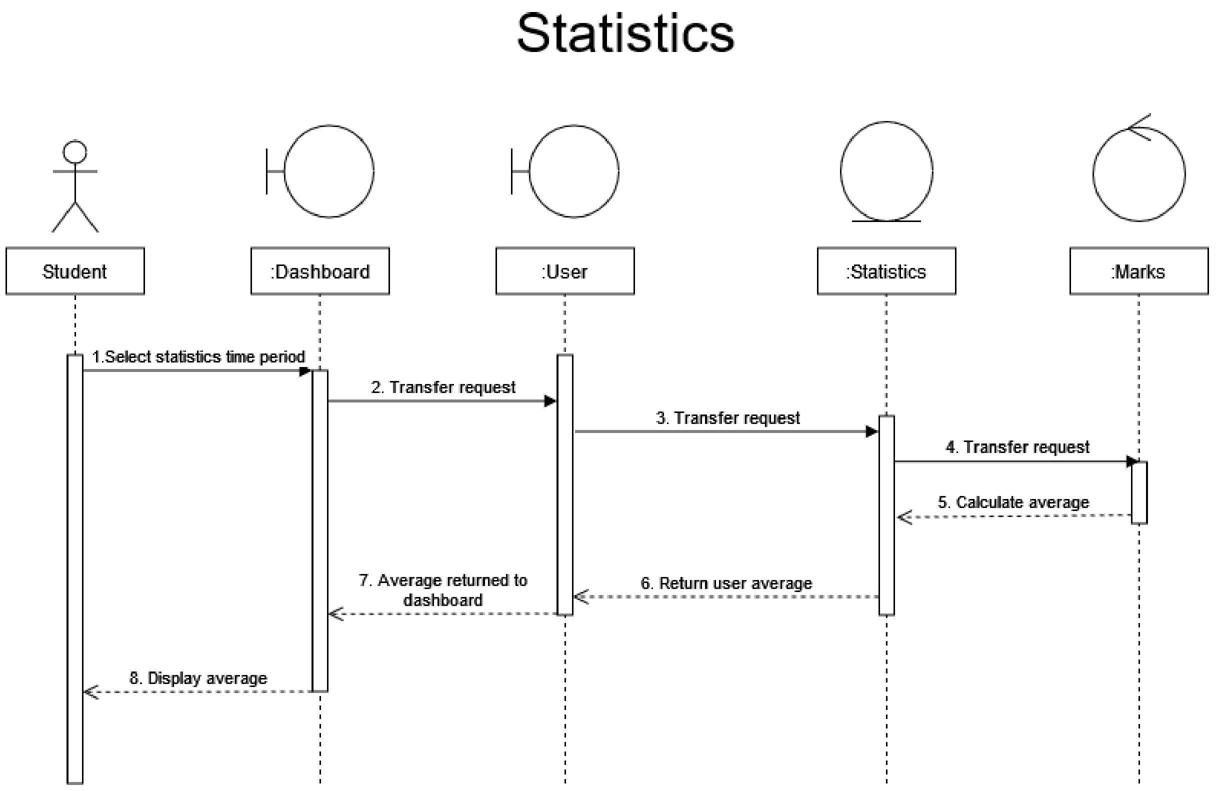
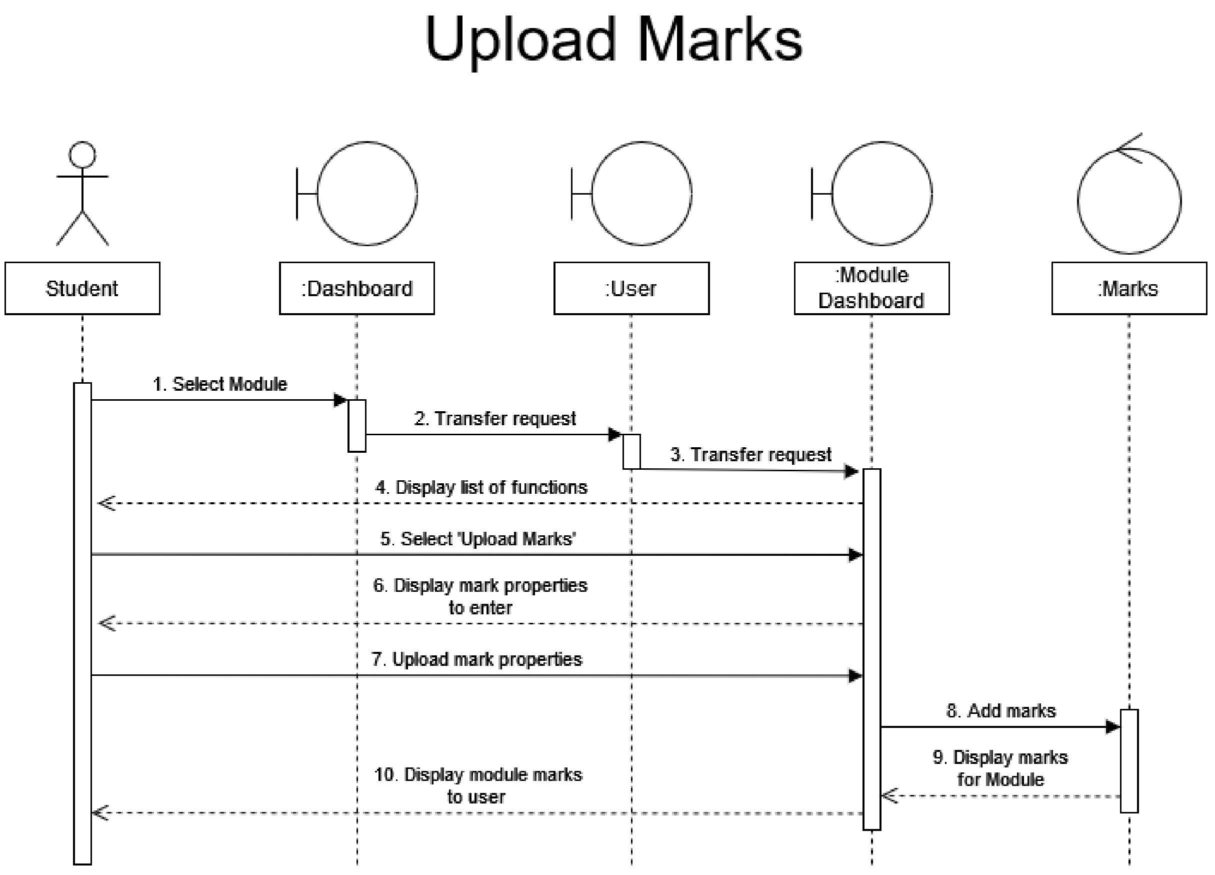


Figure 41: Statistics Sequence Diagram

Upload Marks



Upload Marks
Student
:Dashboard
:User
:Module Dashboard
:Marks
1. Select Module
2. Transfer request
3. Transfer request
4. Display list of functions
5. Select 'Upload Marks'
6. Display mark properties to enter
7. Upload mark properties
8. Add marks
9. Display marks for Module
10. Display module marks to user

Figure 42: Upload Marks Sequence Diagram

Design of Interface and Data Access Layers

The Interface layer – Each part of the system needs to be displayed to the user accurately and without clutter, that is why the system will have a very easy to use, simple design which will utilise Accessibility where applicable and allow users to customise their own application by means of themes, this will allow users to feel more personal with the system, in turn making them use it more and allowing the system to serve its purpose.

The Data Access layer –The data will be stored on campus servers, which would be running a database management system, to allow for easy access and utilisation of data/information. The system will access this layer whenever it needs to request data, or when it wants to write data to the servers. The system will also use a file server to store all the files and information uploaded, by students, and the Centre for Teaching and Learning (CTL), doing this will allow for users to access the information when they need, and upload it when they can.

UML diagram

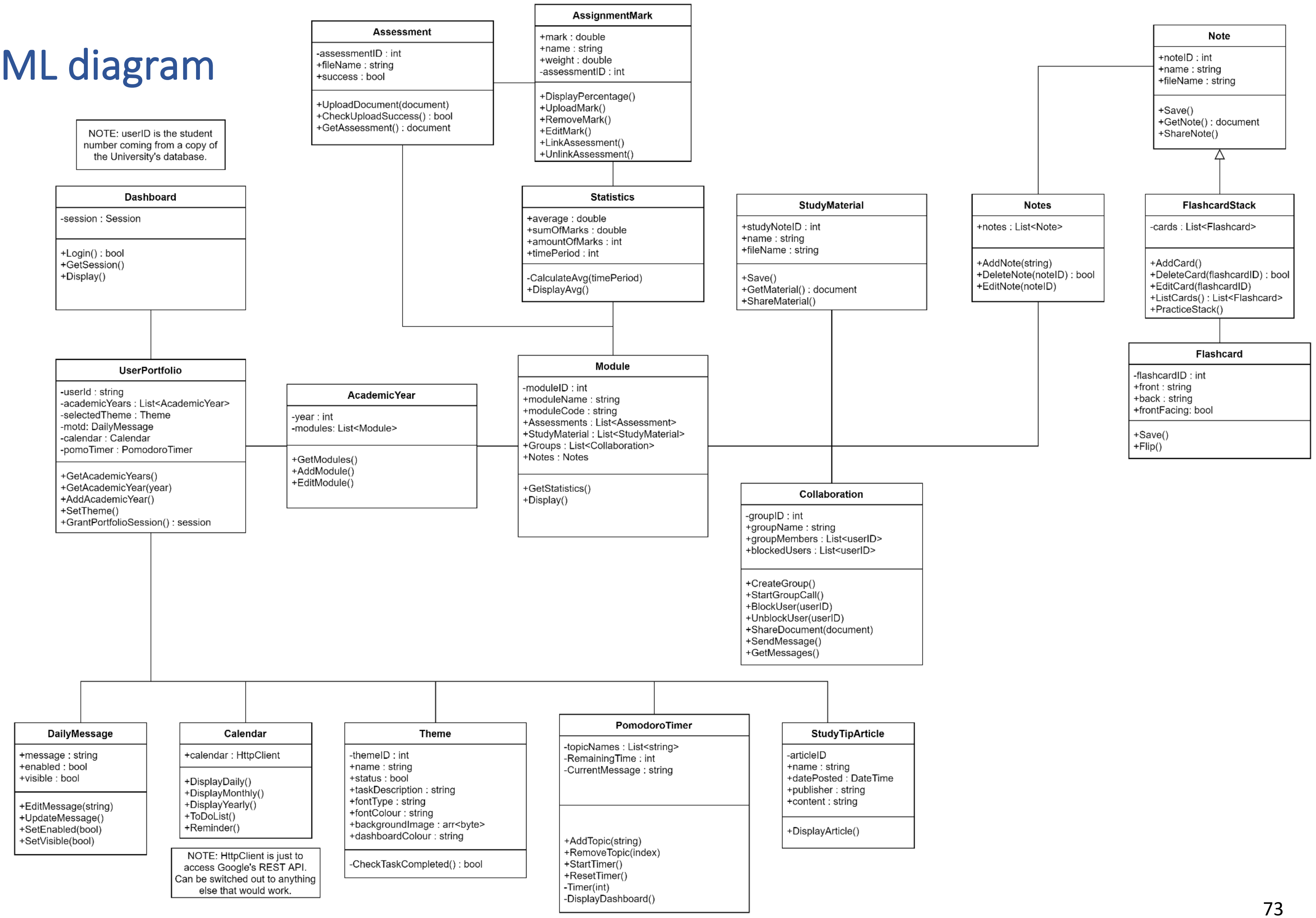


Figure 43: UML Diagram

Pseudo Code

Class: FlashcardStack - Method: PracticeStack

```
void PracticeStack()
{
        duplicate list of cards to temporary list
        while (length of temporary list > 0)
        {
                display front of random card from temporary list
                wait for user to continue
                flip card and show back of card to user
                wait for user to continue
                remove card from temporary list
        }

        return to last view
}
```

Class: FlashcardStack - Method: AddCard

```
void AddCard()
{
        while (True)
        {
                enter front face value of card
                enter back face value of card
                create flashcard object with the front and back values
                add flashcard object to stack’s list of cards
        }

        return to overview of stack view
}
```

Class: PomodoroTimer - Method: StartTimer

```
void StartTimer()
{
        for (int i = 0; i < topicNames.Length; i++)
        {
                Set CurrentMessage to topicNames[i];
                Timer(0);
                Timer(1);
                if (the remainder of i divided by 4 is equal to 3)
                        Timer(2);
        }
}
```

Class: PomodoroTimer - Method: Timer

```
void Timer(int MessageType)
{
        Instantiate a timer with an interval of 60000 miliseconds;

        if (MessageType == 0)
        {
                Set RemainingTime equal to 25;
                DisplayDashboard();
        }

        else if (MessageType == 1)
        {
                Set RemainingTime equal to 5;
                Set CurrentMessage to "Break";
                DisplayDashboard();
        }

        else
        {
                Set RemainingTime equal to 25;
                Set CurrentMessage to "Long Break";
                DisplayDashboard();
        }

        Let the IntervalElapsed event occur when an interval has elapsed;

}
```

Class: PomodoroTimer - Method: DisplayDashboard

```
void DisplayDashboard()
{
        Set displayed message on the dashboard to: CurrentMessage + RemainingTime +
                "minutes remaining"
}
```

Class: PomodoroTimer - Event: IntervalElapsed

```
event IntervalElapsed()
{
        Subtract 1 from RemainingTime;
        DisplayDashboard();
}
```

Overview of development effort

Requirements Workflow

The iterative and incremental model was applied as soon as the project commenced. At first, we produced a summary of the requirements as supplied by the client. After this summary was reviewed by the client, changes already had to occur. We thus applied the iterative and incremental model as the initial requirements underwent revision. It became apparent that we had to remove some functionalities and add others to meet the client's needs more effectively.

Our use cases also had to be revised after feedback was provided by the client. Initially our use cases were constructed too granularly. We adjusted it so that each use case only represented a single functionality.

Analysis Workflow

Few revisions occurred during noun extraction. However, the iterative and incremental model played a crucial role in the construction of the CRC cards. It was necessary to compile a complete list of the classes before creating the sequence diagrams. We then realised that more CRC cards had to be added as we extracted additional classes while compiling this list. In this way, the iterative and incremental model was applied in the construction of the CRC cards.

Design Workflow

Upon completion of the initial analysis workflow, the detailed UML diagram was designed. We then wrote pseudo code for two of the most complex classes. The process of writing pseudo code made it clear that some of the classes in the detailed UML diagram had to be changed. This further implied that changes had to occur in the CRC cards and use cases as well. The way in which changes to the documents in the current workflow results in corresponding changes to documents belonging to previous workflows is another demonstration of the application of the iterative and incremental model.

Total Real Hours Spent

Requirements	22 hours
Analysis	39 hours
Design	18 Hours
Implementation	n/a