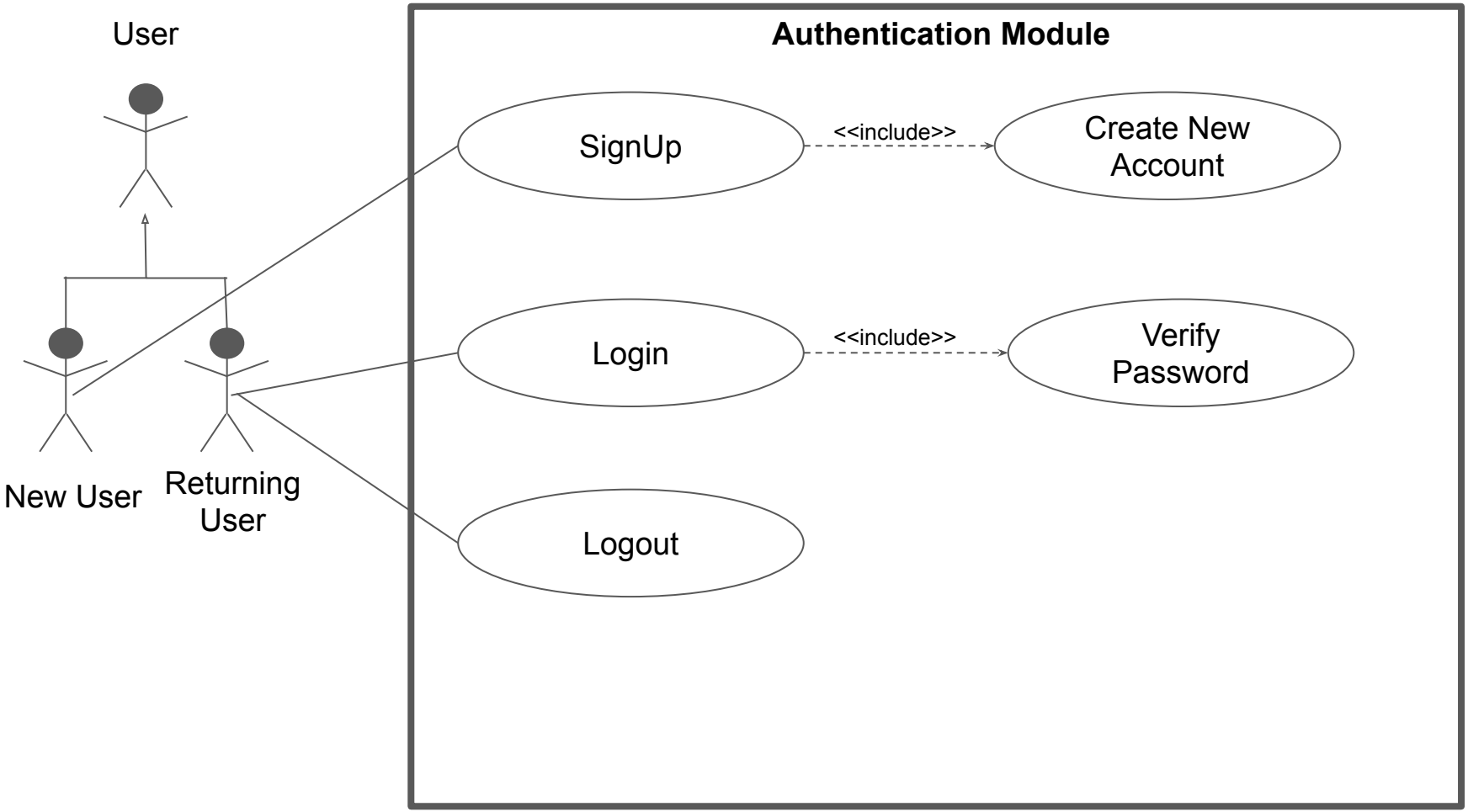
DESIGN OF THE SYSTEM

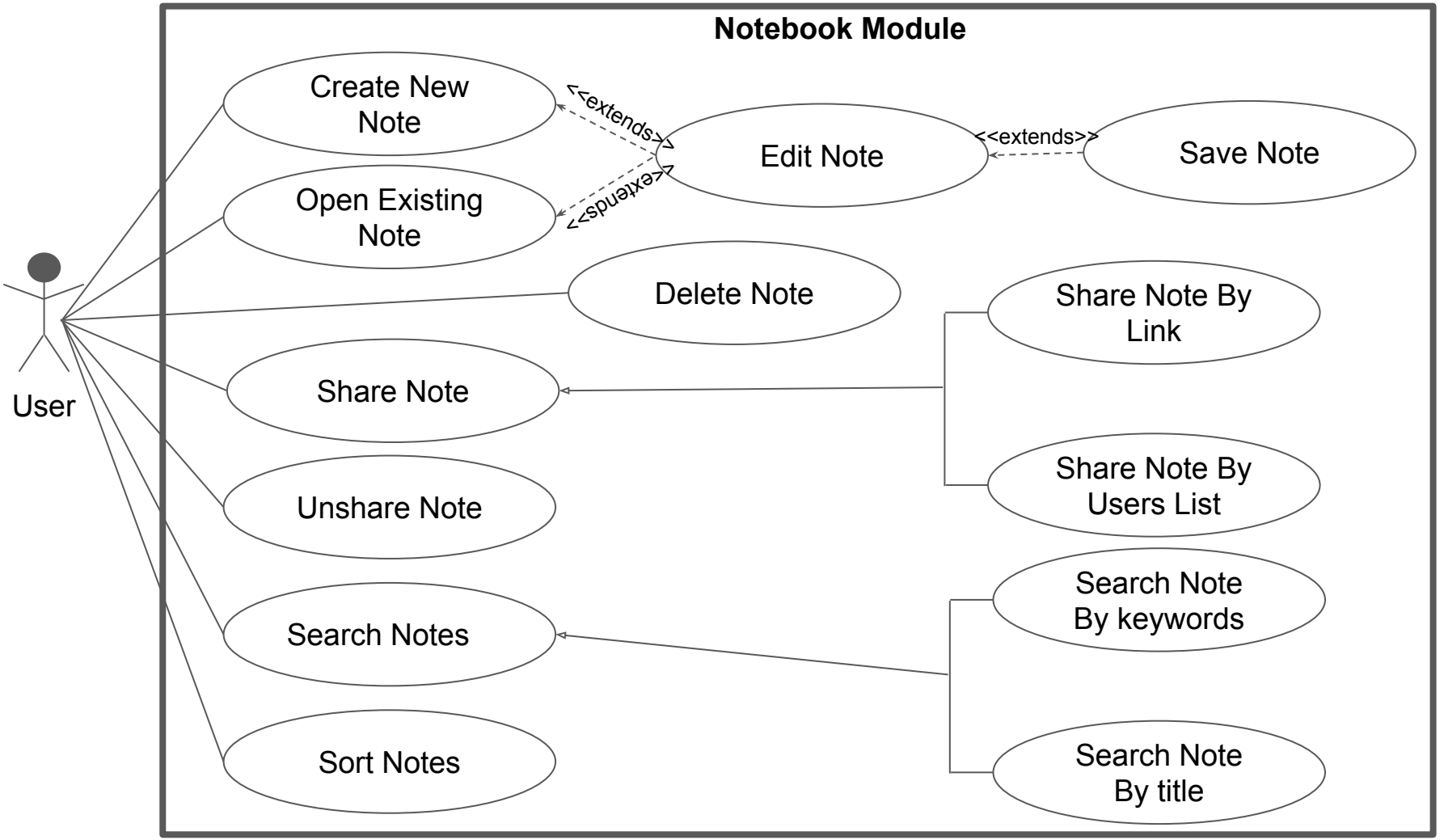
GROUP-19

Aditya Rajesh Patil	180101004
Kartikeya Saxena	180101034
Kushal Sangwan	180101096
Milind Prabhu	180101091
Sudesh Chaudhary	180103077

USE-CASE DIAGRAMS

Mainline and Alternative Sequence


User
Authentication Module
SignUp
<<include>>
Create New Account
Login
<<include>>
Verify Password
Logout
New User
Returning User


Notebook Module
Create New Note
Open Existing Note
Edit Note
Save Note
Delete Note
Share Note
Share Note By Link
Share Note By Users List
Unshare Note
Search Notes
Search Note By keywords
Search Note By title
Sort Notes
User
<<extends>>
<<extends>>
<<extends>>

SignUp User
MainLine Sequence

1. User: Selects register user option
2. System: Displays prompt to enter username,email,password
3. Customer: Enters the data.
4. System: Displays the message that the user has been successfully registered. Next displays the home page

Alternative Seq1: at step 4
4.System: Displays message that user is already registered.

Alternative Seq2: at step 4
4. System: Displays the message that some input value wasn't entered. Displays prompt to enter missing value.

Login User
MainLine Sequence

1. User: Selects login option
2. System: Displays a prompt to enter username, password.
3.Customer: Enters the data.
4.System: Displays the home page.

Alternative Seq1: at step 4
4.System: Displays message that entered credentials are wrong.

Alternative Seq2: at step 4
4. System: Displays the message that some inputs are invalid.

Logout User
MainLine Sequence

1. User: Select Logout option
2. System: leads to login page

Create New Note
Mainline Sequence

1. User: Selects create new note option.
2. System: Prompts the user to enter note title.
3.Customer: Enters the title
4.System: Displays a blank page as the new note.

Alternate Sequence: at step 4
4. System: displays message note already exists.

Open Existing Note
MainLine Sequence

1. User: Selects the note name in the hierarchy.
2. System: Displays the note.

Share Note
Mainline Sequence

1. User1: Selects the share option for the note
2. System: Prompts the user to enter mode of sharing.
3. User1: Selects share by url option.
4. System: Displays shareable link.

Alternative Sequence1: at step 3
3. User1: Selects share by email.
2. System: Displays prompt to enter email ID of user.
3.User1: Enters the email ID.
4.System: Displays a success message.

Alternative Sequence2: at step 4 of alternative sequence 1
4. System: displays a message that email is invalid.

Unshare Note
Mainline Sequence

1. User1: Selects Unshare option.
2. System: Display Editor's List and a message to select one user
3.User1: Enters the required input.
4.System: Displays a success message.

Alternative Sequence: at step 4
4. System: Displays message that you are not the owner and don't have permissions to remove users from the list.
.

Delete Note

1. User: Select delete note option for the note
2. System: removes the note from the list of notes

Search Notes
Mainline Sequence

1.User:Selects option to search note.
2.System: Displays prompt to enter string to be search
3.User: Enters the required input
4.System: Highlights the occurences of the text in the notes containing the search results.

Alternative Sequence: at step 4
4.System: Displays message pattern not found.

Sort Notes
Mainline Sequence

1.User: Selects sort note option.
2.System: Displays prompt to enter the field to sort by (Modification Time, Creation Time or Title).
3.User: Enters the field to sort by.
4.System: Displays the notes sorted by the selected field.

Edit Note
Mainline Sequence

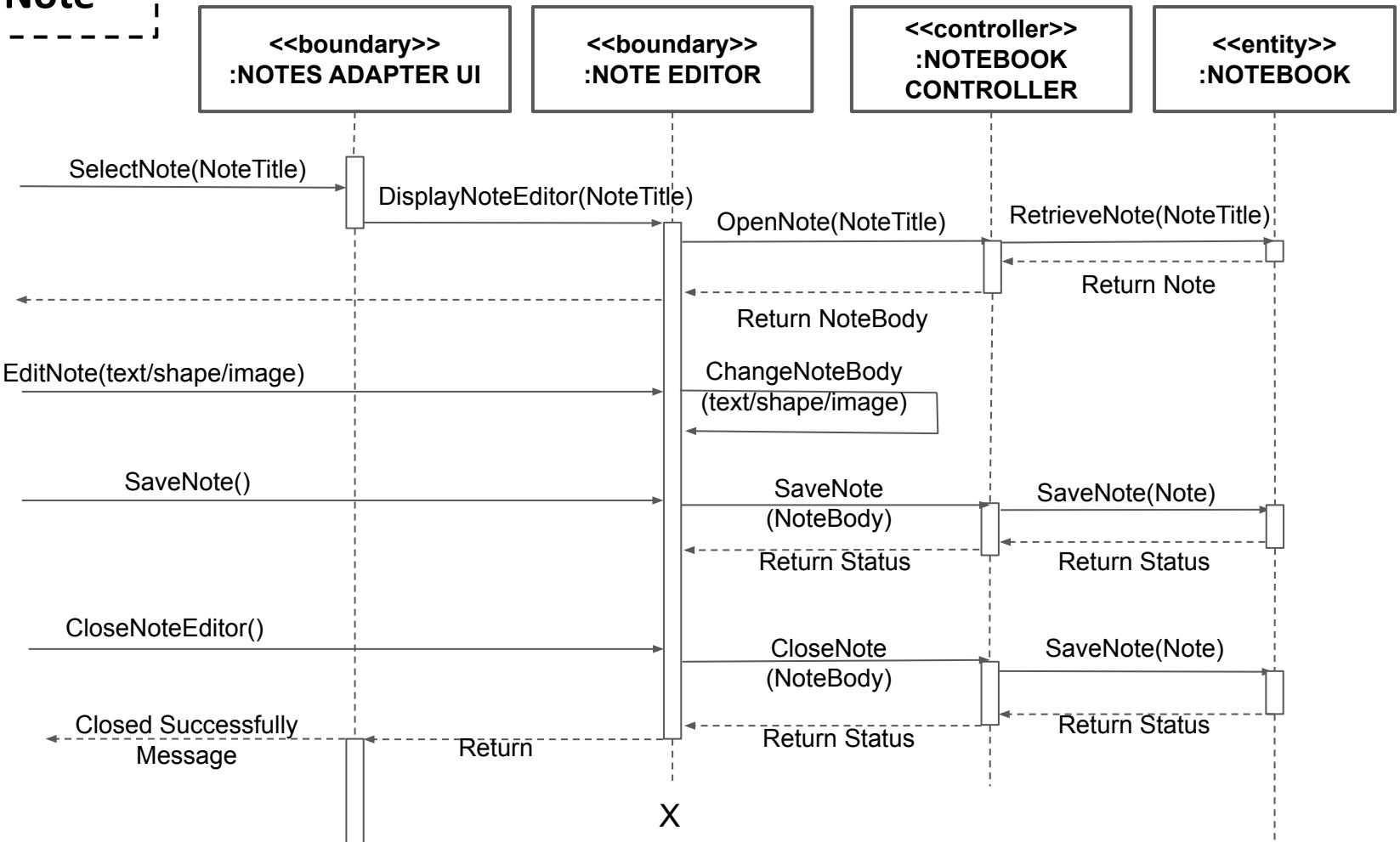
1.User: Selects the location to edit and the edit to make.
2.System: Displays the edits.

Alternative Sequence: at step 2
2.System: Displays message that the user does not have edit access.

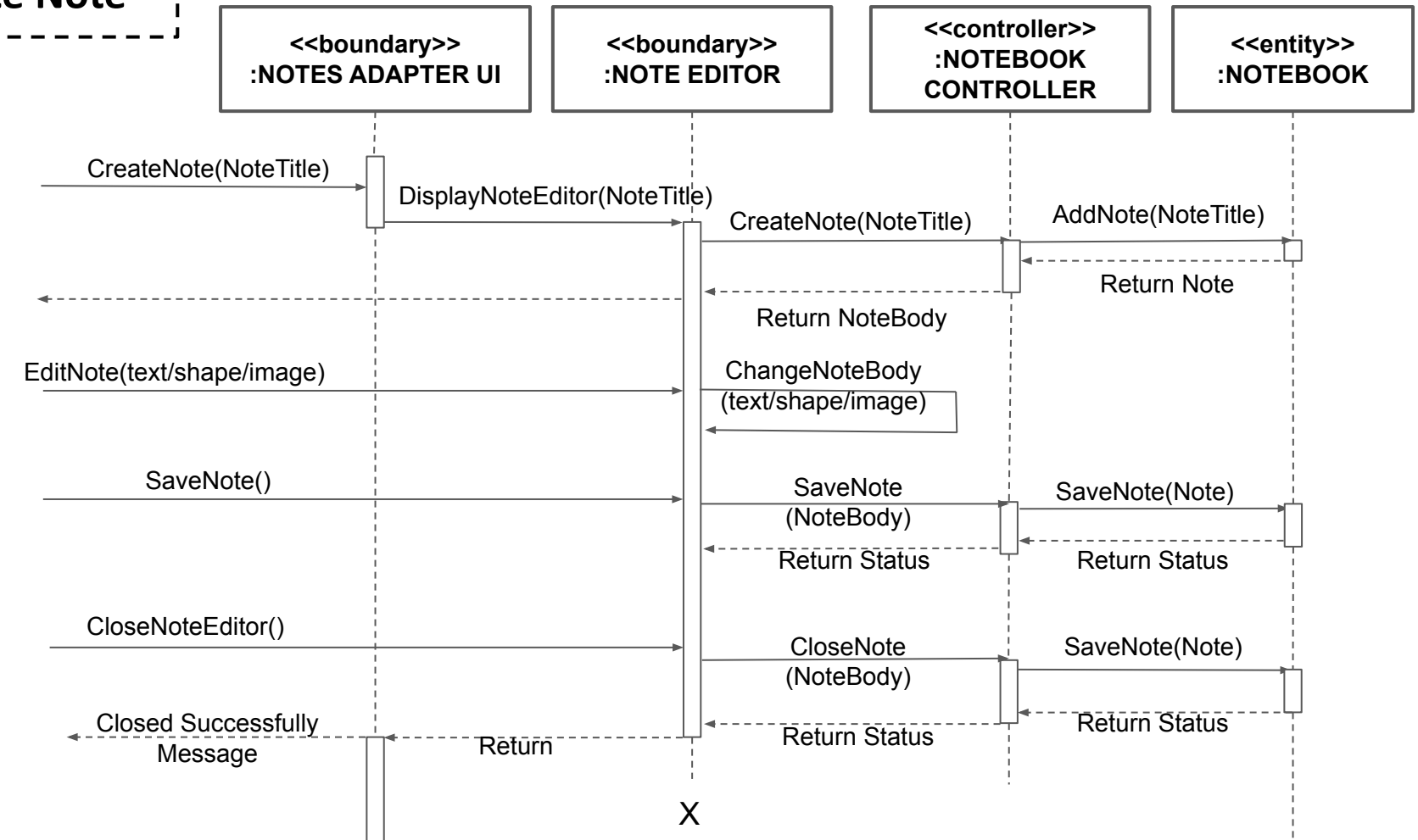
INTERACTION DIAGRAMS

Sequence Diagrams

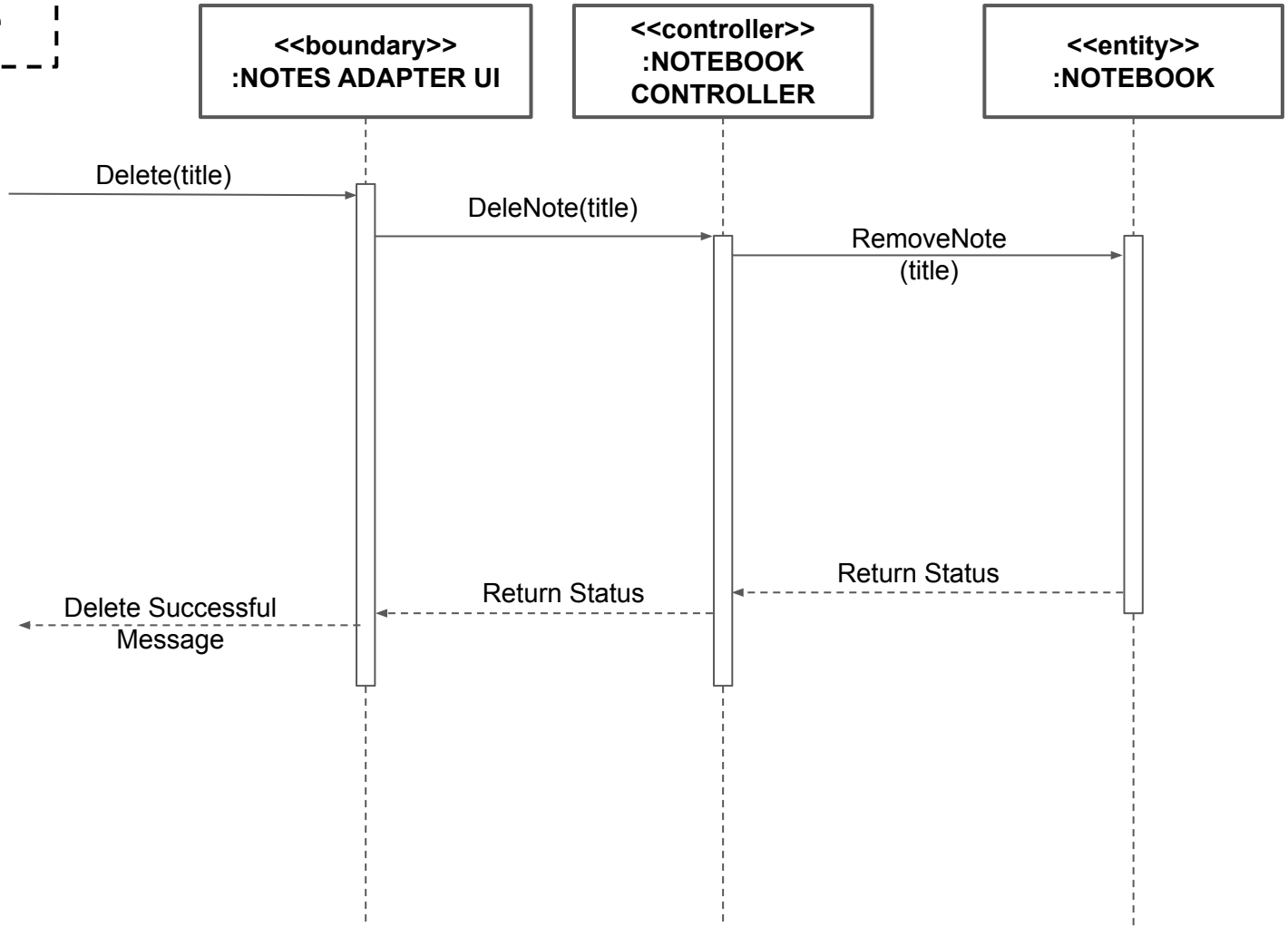
Open Note

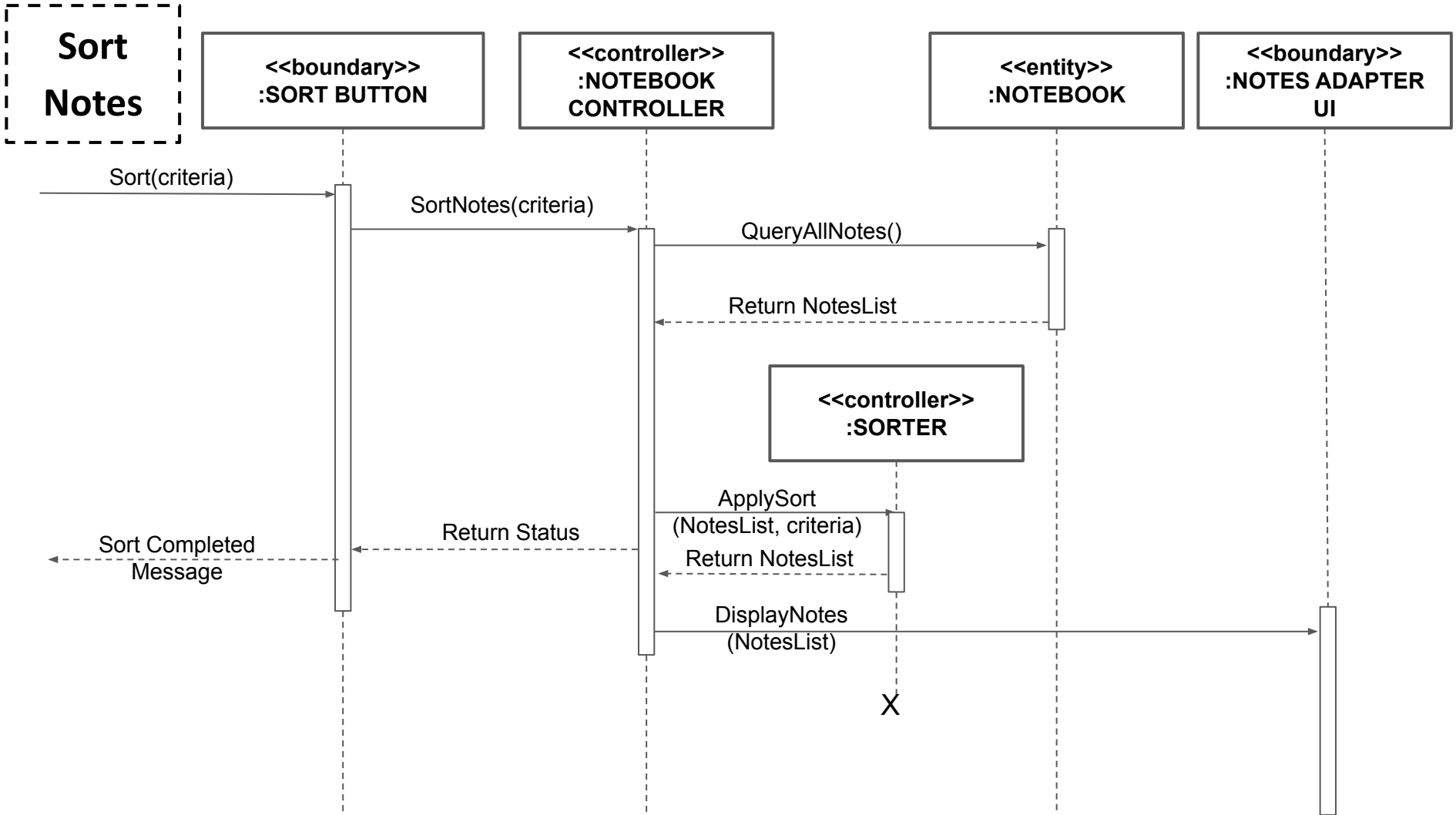

<<boundary>>
:NOTES ADAPTER UI
<<boundary>>
:NOTE EDITOR
<<controller>>
:NOTEBOOK
CONTROLLER
<<entity>>
:NOTEBOOK
SelectNote(NoteTitle)
DisplayNoteEditor(NoteTitle)
OpenNote(NoteTitle)
RetrieveNote(NoteTitle)
Return Note
Return NoteBody
EditNote(text/shape/image)
ChangeNoteBody
(text/shape/image)
SaveNote()
SaveNote
(NoteBody)
SaveNote(Note)
Return Status
Return Status
CloseNoteEditor()
CloseNote
(NoteBody)
SaveNote(Note)
Return Status
Return Status
Closed Successfully
Message
Return
X

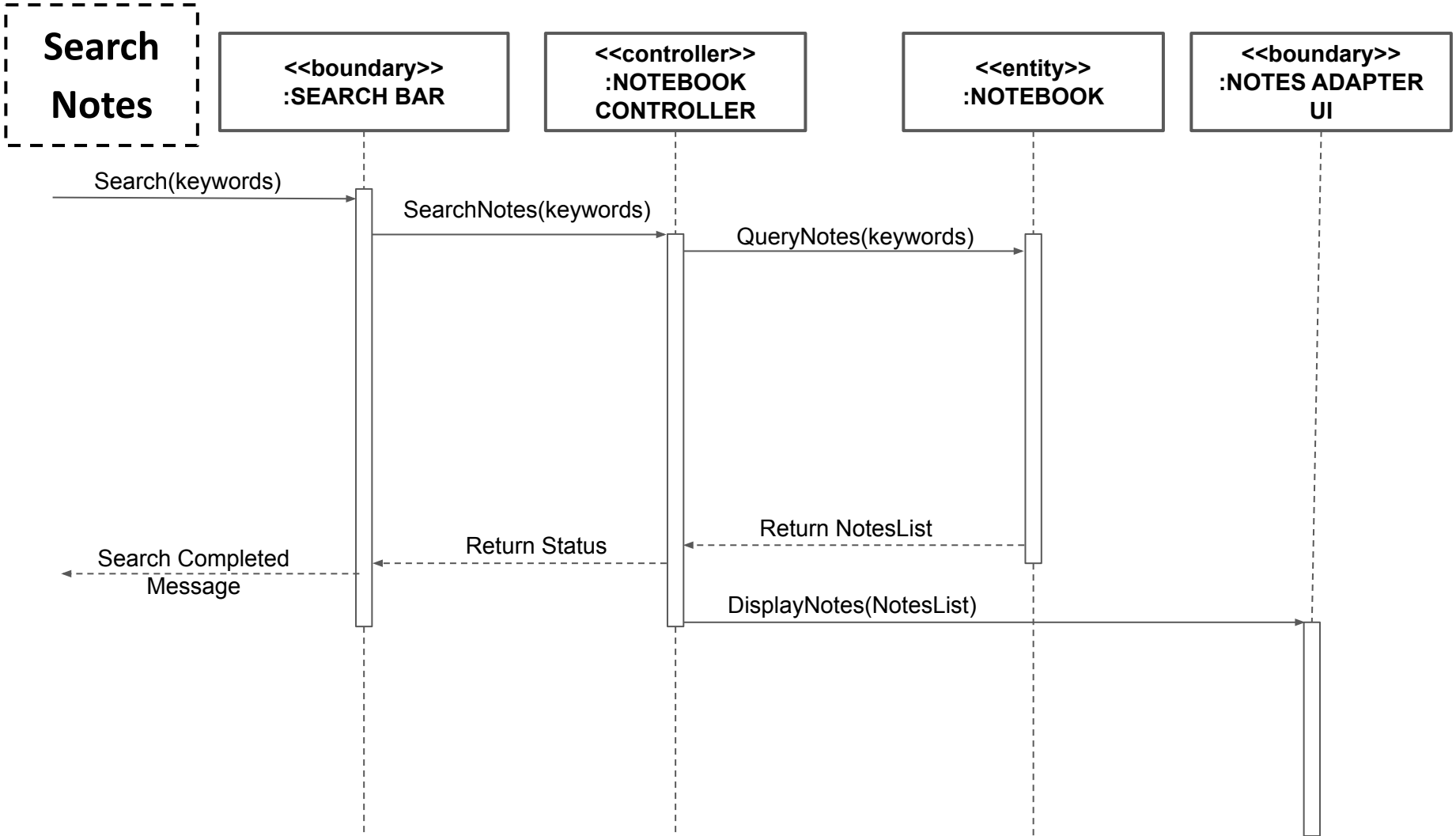
Create Note

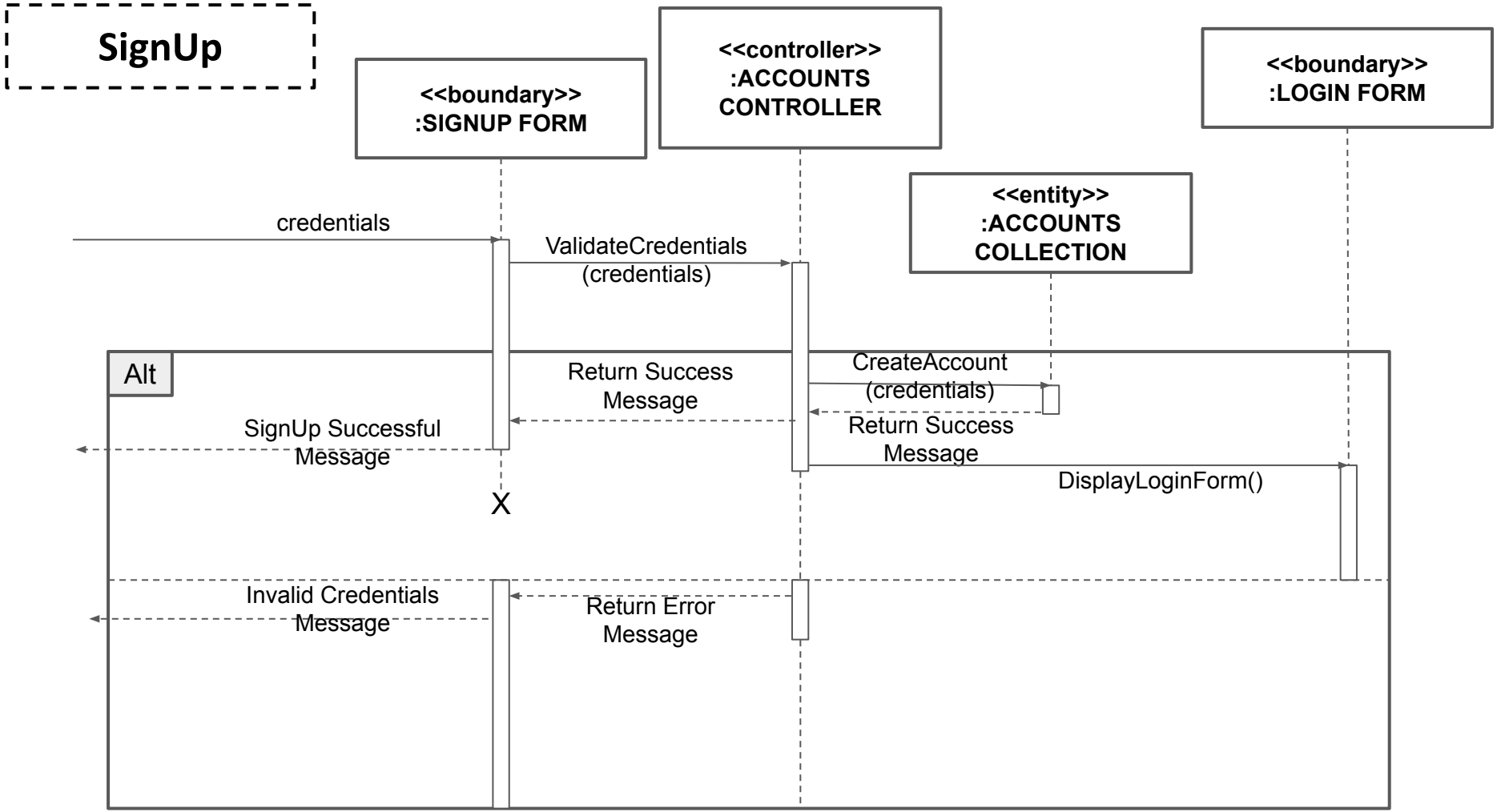

<<boundary>>
:NOTES ADAPTER UI
<<boundary>>
:NOTE EDITOR
<<controller>>
:NOTEBOOK
CONTROLLER
<<entity>>
:NOTEBOOK
CreateNote(NoteTitle)
DisplayNoteEditor(NoteTitle)
CreateNote(NoteTitle)
AddNote(NoteTitle)
Return Note
Return NoteBody
EditNote(text/shape/image)
ChangeNoteBody
(text/shape/image)
SaveNote()
SaveNote
(NoteBody)
SaveNote(Note)
Return Status
Return Status
CloseNoteEditor()
CloseNote
(NoteBody)
SaveNote(Note)
Return Status
Return Status
Closed Successfully
Message
Return
X

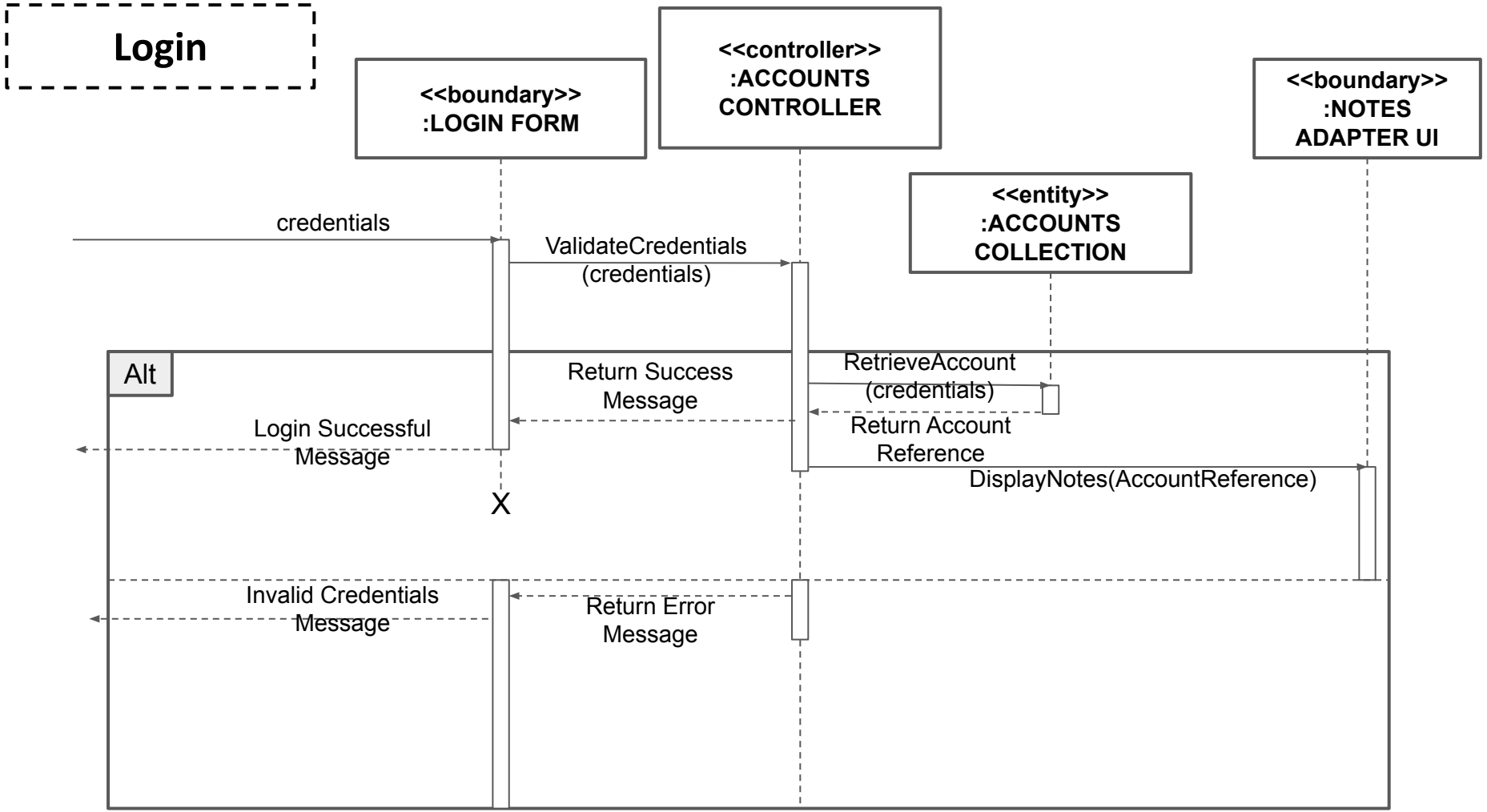
Delete Note

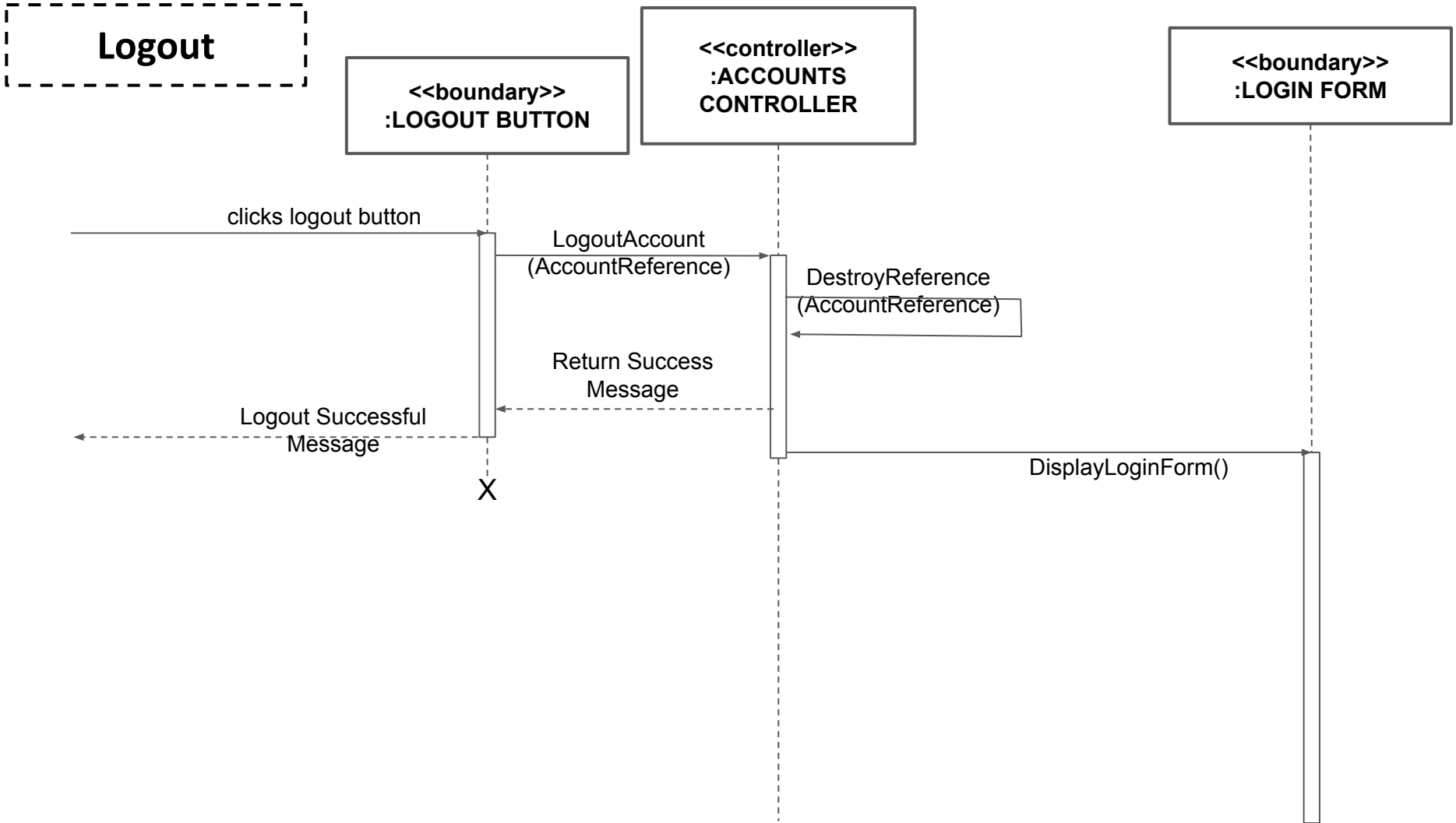

<<boundary>>
:NOTES ADAPTER UI
<<controller>>
:NOTEBOOK
CONTROLLER
<<entity>>
:NOTEBOOK
Delete(title)
DeleNote(title)
RemoveNote
(title)
Return Status
Return Status
Delete Successful
Message

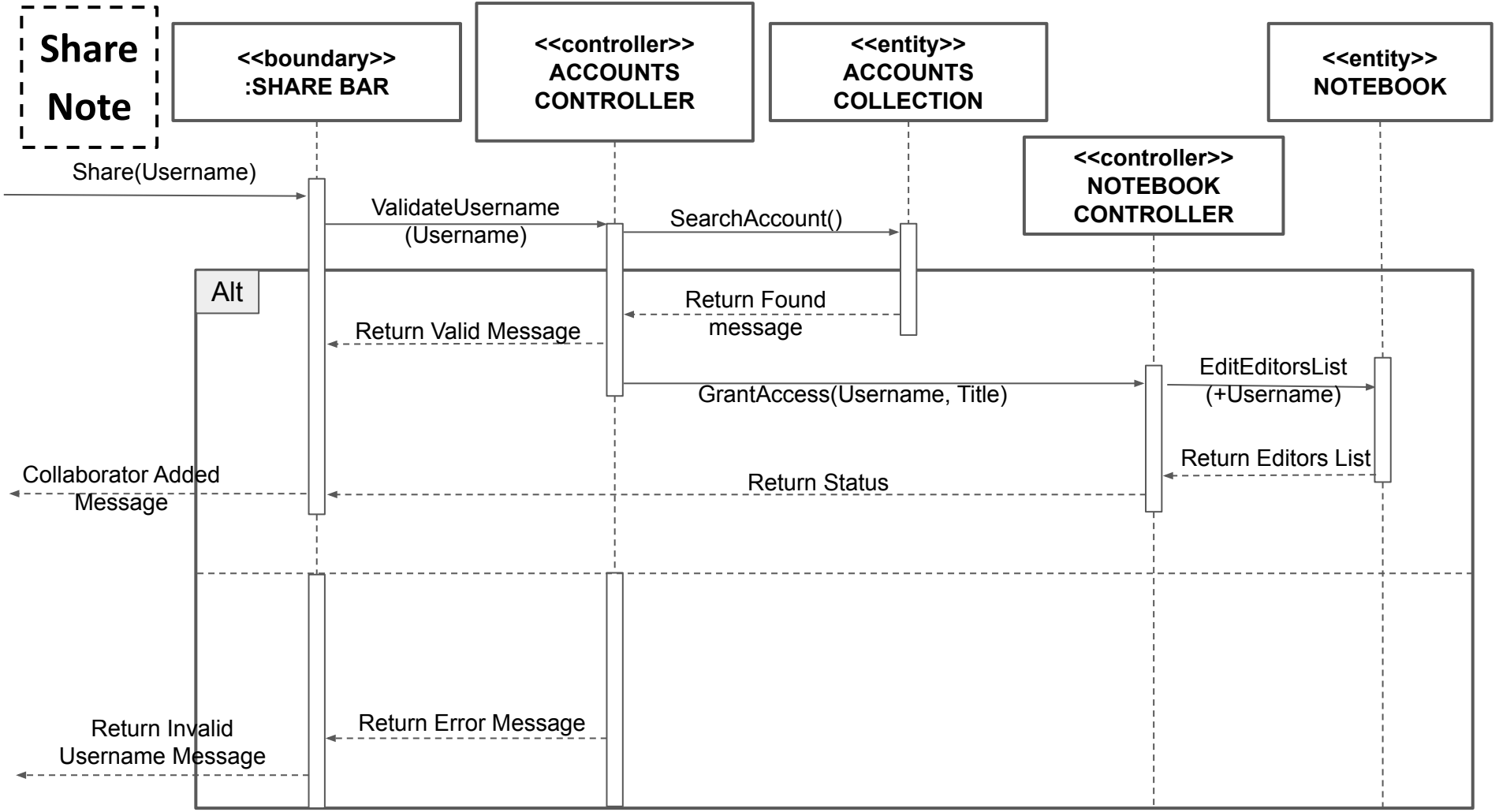

Sort Notes
<<boundary>> :SORT BUTTON
<<controller>> :NOTEBOOK CONTROLLER
<<entity>> :NOTEBOOK
<<boundary>> :NOTES ADAPTER UI
Sort(criteria)
SortNotes(criteria)
QueryAllNotes()
Return NotesList
<<controller>> :SORTER
ApplySort (NotesList, criteria)
Return NotesList
Return Status
Sort Completed Message
DisplayNotes (NotesList)
X

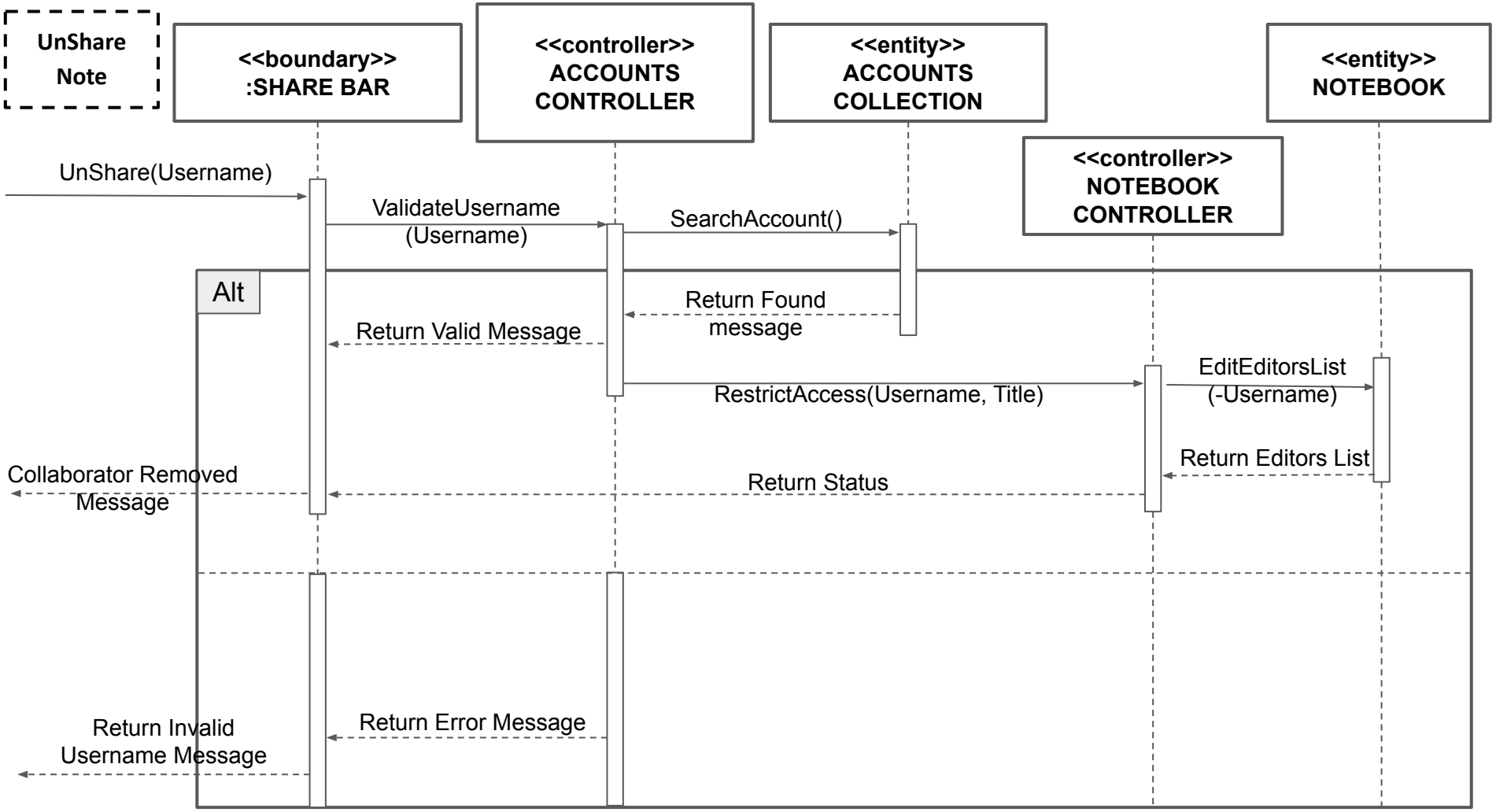

Search Notes
<<boundary>> :SEARCH BAR
<<controller>> :NOTEBOOK CONTROLLER
<<entity>> :NOTEBOOK
<<boundary>> :NOTES ADAPTER UI
Search(keywords)
SearchNotes(keywords)
QueryNotes(keywords)
Return NotesList
Return Status
Search Completed Message
DisplayNotes(NotesList)


SignUp
<<boundary>>
:SIGNUP FORM
<<controller>>
:ACCOUNTS
CONTROLLER
<<boundary>>
:LOGIN FORM
<<entity>>
:ACCOUNTS
COLLECTION
credentials
ValidateCredentials
(credentials)
Alt
Return Success
Message
CreateAccount
(credentials)
Return Success
Message
SignUp Successful
Message
DisplayLoginForm()
X
Invalid Credentials
Message
Return Error
Message


Login
<<boundary>>
:LOGIN FORM
<<controller>>
:ACCOUNTS
CONTROLLER
<<boundary>>
:NOTES
ADAPTER UI
<<entity>>
:ACCOUNTS
COLLECTION
credentials
ValidateCredentials
(credentials)
Alt
Return Success
Message
RetrieveAccount
(credentials)
Return Account
Reference
Login Successful
Message
DisplayNotes(AccountReference)
X
Invalid Credentials
Message
Return Error
Message

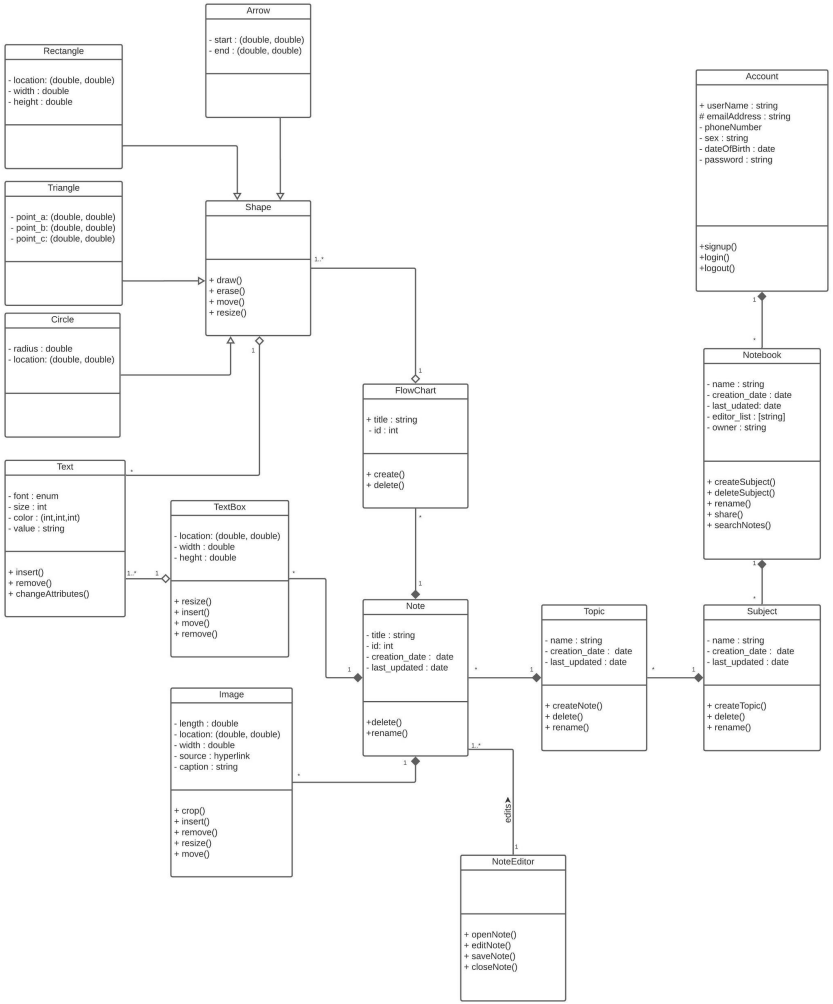

Logout
<<boundary>>
:LOGOUT BUTTON
<<controller>>
:ACCOUNTS
CONTROLLER
<<boundary>>
:LOGIN FORM
clicks logout button
LogoutAccount
(AccountReference)
DestroyReference
(AccountReference)
Return Success
Message
Logout Successful
Message
X
DisplayLoginForm()


Share Note
<<boundary>> :SHARE BAR
<<controller>> ACCOUNTS CONTROLLER
<<entity>> ACCOUNTS COLLECTION
<<controller>> NOTEBOOK CONTROLLER
<<entity>> NOTEBOOK
Share(Username)
ValidateUsername (Username)
SearchAccount()
Alt
Return Found message
Return Valid Message
EditEditorsList (+Username)
GrantAccess(Username, Title)
Return Editors List
Collaborator Added Message
Return Status
Return Error Message
Return Invalid Username Message


UnShare Note
<<boundary>> :SHARE BAR
<<controller>> ACCOUNTS CONTROLLER
<<entity>> ACCOUNTS COLLECTION
<<controller>> NOTEBOOK CONTROLLER
<<entity>> NOTEBOOK
UnShare(Username)
ValidateUsername (Username)
SearchAccount()
Alt
Return Found message
Return Valid Message
EditEditorsList (-Username)
RestrictAccess(Username, Title)
Return Editors List
Collaborator Removed Message
Return Status
Return Error Message
Return Invalid Username Message

CLASS HIERARCHY

Class Diagrams and Definitions


Arrow
- start : (double, double)
- end : (double, double)
Rectangle
- location: (double, double)
- width : double
- height : double
Triangle
- point_a: (double, double)
- point_b: (double, double)
- point_c: (double, double)
Shape
+ draw()
+ erase()
+ move()
+ resize()
Circle
- radius : double
- location: (double, double)
Account
+ userName : string
# emailAddress : string
- phoneNumber
- sex : string
- dateOfBirth : date
- password : string
+signup()
+login()
+logout()
Notebook
- name : string
- creation_date : date
- last_udated: date
- editor_list : [string]
- owner : string
+ createSubject()
+ deleteSubject()
+ rename()
+ share()
+ searchNotes()
FlowChart
+ title : string
- id : int
+ create()
+ delete()
Text
- font : enum
- size : int
- color : (int,int,int)
- value : string
+ insert()
+ remove()
+ changeAttributes()
TextBox
- location: (double, double)
- width : double
- heght : double
+ resize()
+ insert()
+ move()
+ remove()
Note
- title : string
- id: int
- creation_date : date
- last_updated : date
+delete()
+rename()
Topic
- name : string
- creation_date : date
- last_updated : date
+ createNote()
+ delete()
+ rename()
Subject
- name : string
- creation_date : date
- last_updated : date
+ createTopic()
+ delete()
+ rename()
Image
- length : double
- location: (double, double)
- width : double
- source : hyperlink
- caption : string
+ crop()
+ insert()
+ remove()
+ resize()
+ move()
edits
NoteEditor
+ openNote()
+ editNote()
+ saveNote()
+ closeNote()

Account

Methods:

1. **signup():**
   - Parameters: username,email,password
   - Return value: success/error message (string)
   - Description: this method is used to create a new account
2. **login():**
   - Parameters: email,password
   - Return value: success/error message (string)
   - Description: this method is used to login to an existing account
3. **logout():**
   - Parameters: NA
   - Return value: success/error message (string)
   - Description: this method is used to log out an account
4. **createNotebook():**
   - Parameters: Title
   - Return value: Returns object of Notebook class
   - Description: this method is used to create a new notebook under the account

Notebook

Methods:

1. **createSubject():**
   - Parameters: subject name
   - Return value: Returns object of Subject class
   - Description: this method is used to create a new subject under notebook
2. **delete():**
   - Parameters: NA
   - Return value: success/error message (string)
   - Description: this method is used to delete the notebook object
3. **rename():**
   - Parameters: Name
   - Return value: success/error message (string)
   - Description: this method is used to change the name of the notebook object
4. **share():**
   - Parameters: user emails
   - Return value: success/error message (string)
   - Description: this method is used to share a notebook with other users
5. **search_notes():**
   - Parameters: keywords(list of strings)
   - Return value: list of notes
   - Description: this method is used to search the notebook for keywords

Subject

Methods:

1. **createTopic():**
    - Parameters: Title
    - Return value: Returns object of Subject class
    - Description: this method is used to create a new topic under the subject
2. **delete():**
    - Parameters: NA
    - Return value: success/error message (string)
    - Description: this method is used to delete the subject object
3. **rename():**
    - Parameters: new subject name
    - Return value: NA
    - Description: this method is used to update the subject name

Topic

Methods:

1. **createNote():**
   - Parameters: Title
   - Return value: Returns object of Note class
   - Description: this method is used to create a new note under the topic
2. **delete():**
   - Parameters: NA
   - Return value: success/error message (string)
   - Description: this method is used to delete the topic object
3. **rename():**
   - Parameters: new topic name
   - Return value: NA
   - Description: this method is used to update the topic name

Note

Methods:

1. **delete():**
    - Parameters: NA
    - Return value: success/error message (string)
    - Description: this method is used to delete the noteobject
2. **rename():**
    - Parameters: new subject name
    - Return value: success/error message (string)
    - Description: this method is used to update the note name

Note Editor

Methods:

1. **openNote():**
   - Parameters: note object
   - Return value: success/error message (string)
   - Description: this method is used to display the note object
2. **editNote():**
   - Parameters: note object
   - Return value: success/error message (string)
   - Description: this method is used to change the note object's contents
   - Calls: insertImage(), insertFlowChart(), insertTextBox()
3. **saveNote():**
   - Parameters: note object
   - Return value: success/error message (string)
   - Description: this method is used to save the note object's contents
4. **closeNote():**
   - Parameters: note object
   - Return value: success/error message (string)
   - Description: this method is used to remove the note object's from display

Image

Methods:

1. **crop():**
   - Parameters: length,width
   - Return value: success/error message (string)
   - Description: this method is used crop an image in a note.
2. **insert():**
   - Parameters: note object
   - Return value: success/error message (string)
   - Description: this method is used to insert the image in a note.
3. **remove():**
   - Parameters: note object
   - Return value: success/error message (string)
   - Description: this method is used remove the image from a note
4. **resize():**
   - Parameters: length,width
   - Return value: success/error message (string)
   - Description: this method is used resize an image.
5. **move():**
   - Parameters: note , location
   - Return value: success/error message (string)
   - Description: this method is used to move the image to new location in a note .

TextBox

Methods:

1. **resize():**
   - Parameters: length,width
   - Return value: NA
   - Description: this method is used to resize the dimensions of the text-box.
2. **insert():**
   - Parameters: note Object
   - Return value: success/error message (string)
   - Description: this method is used to insert the text-box into note
3. **move():**
   - Parameters: new position coordinates
   - Return value: NA
   - Description: this method is used to update the location of the text-box

Shape

Methods:

1. **draw():**
   - Parameters: note
   - Return value: success/error message (string)
   - Description: this method is used to draw the shape inside the note.
2. **erase():**
   - Parameters: note
   - Return value: success/error message (string)
   - Description: this method is used to insert the text-box into note
3. **move():**
   - Parameters: new position coordinates
   - Return value: NA
   - Description: this method is used to update the location of the shape
4. **resize():**
   - Parameters: dimensions(Integers)
   - Return value: NA
   - Description: this method is used to resize the shape

Text

Methods:

1. **insert():**
    - Parameters: TextBox object
    - Return value: success/error message (string)
    - Description: this method is used to insert the text into text-box.
2. **remove():**
    - Parameters: NA
    - Return value: success/error message (string)
    - Description: this method is used to remove the text
3. **changeAttributes():**
    - Parameters: attributes
    - Return value: success/error message (string)
    - Description: this method is used to text attributes

USABILITY DOCUMENT

Eight golden rules of Shneiderman

Strive For Consistency

1. **External consistency:** We are planning to follow the material UI, which is an interface design used by a lot of applications created by Google. This way we hope that users will find the interface more familiar and fluid. Our note taking application will have an interface which will look like a real notebook with pages. This will make the users' experience consistent with their experience taking notes on physical notebooks. The buttons on the interface will have icons that convey what they do. For example, a dustbin icon for a button that deletes the page, a pencil icon on a button that allows you to scribble on the page and so on.

2. **Internal Consistency:** The material UI that we will use will bring in a lot of consistency as well. To elaborate, the font face, colors and typography will remain more or less uniform across the windows that the user interacts with. A button with a particular icon on it will perform the same action across all windows. The application will adapt to the platform requirements making it just as pleasant in Linux, macOS or Windows. For example, the shortcut key to copy in Linux and Windows would be Ctrl + c but in macOS would be cmd + c.

Design For Universal Usability

1. **Novice:** The user will find the material UI very pleasant. The interface will be very suggestive. The interface will be organized very cleanly so that the user can find tools that they want to use easily. The buttons on icons will indicate the action they perform. Hovering on different elements of the interface will reveal how the user can interact with them. A help guide with examples will also be made available to the user.

2. **Expert:** Users that have already tried other applications should find the transition pleasant because the design includes useful features from existing applications. The organization of the interface will ensure that the expert user can work efficiently without perceiving any serious difficulty in making the switch to our application.

Offer Informative Feedback

The application is intended to be very interactive and give back useful feedback. The following features would ensure that this is the case.

1. Hovering over UI elements that are clickable will change their color indicating that they are clickable. Buttons that are clicked will change their colour.
2. Hyperlinks would be underlined and colored blue.
3. Proper loading bars and progress bars will be used while loading notes and logging in and out of the account. This will keep the user informed about what is going on.
4. Important actions would prompt for confirmation before execution. Example: Before deleting a note the system will prompt the user for confirmation.
5. The color of the floppy icon will change after the save operation is done indicating the work is saved.
6. Hovering on different elements of the interface will reveal how the user can interact with them.
7. The user would literally be able to see the things he/she drags or drops across the screen.

These features would give the user a sense of security and comfort.

Design Dialogues to Yield Closure

The users will receive confirmation when the actions that they try to perform succeed. If they fail, users will see a message telling them that the action was not performed, the reason why it was not performed and what they can do about it. This way the users will not be kept guessing. For example, when a user saves a page successfully a pop-up message will inform the user that the page was saved successfully.

Offer Error Prevention And Simple Error Handling

1. **Error prevention:** As far as error prevention is concerned the user will be asked for confirmation by the System before any major actions like deletion of a subject with note, making them aware of the consequences of their actions. Automatic checks will be made for entered data (like password, email) and alerts or reminders will be provided for inappropriate data entry formats. We will try to associate red with error prone actions as much as possible. The elements on the screen will be organized clearly to reduce errors.
2. **Error handling:** Flag the text fields where the users forgot to provide input in an online login/signup form. The error messages displayed will be easy for the user to parse and will also tell the user what they should do to resolve the error. The interface will be robust to prevent the user from making irreversible changes that result in loss of data. Recovery mechanisms will be made available to help the user recover from unintended actions.

Permit Easy Reversal Of Actions

The following options are provided to reverse actions:

1. The undo option can reverse the last change made to a page.
2. The redo option can reverse the last use of the undo option.
3. The backspace key can be used to delete the last inserted text.
4. The eraser tool can be used to erase lines that were previously drawn.
5. The unshare button can be used to unshare a document shared with another user.

Keep Users In Control

The external consistency provides the user with a familiar comfort of doing the tasks as they would do with pen and notebooks. The fact that actions are easily reversible means that the user will be able to quickly backtrack from whatever they are doing allowing them to perform tasks without the constant fear of failure. The informative feedback provided by the system will provide a sense of security and comfort to the user by allowing them to see the tasks progress. Proper error handling and tooltips will allow users to easily learn the functionality of the elements. The users will be given enough choices to make them feel in control but not so much so that they feel lost.

Reduce Short-Term Memory Load

The amount of learning required will be minimal as the interface will have cues to guide the user through each step. The icons on buttons, the information on hovering over elements of the window, tips that guide the user on how to do something will mean that the user does not have to remember a lot of things. The interface will make the experience very fluid.

DESIGN JUSTIFICATION

Cohesion And Coupling

Cohesion

Any UML design should strive to achieve high cohesion and low coupling among its classes. That is what we have done too . The following are the occurrences of different types of cohesion in our design followed by the justification of how few classes in our design are tightly coupled and mostly loose coupling is seen.

Cohesion: Encapsulation enables a class to be highly cohesive i.e. their purpose is very clear. Put another way, the class does one thing, only one thing, and it does that only one thing well. Most of the classes we made have a single purpose.

1. **Logical cohesion:** A class exhibits logical cohesion if the tasks its methods perform are conceptually related which can be seen in the account and shape class, .In account, login(), signup() and logout() are logically related to authentication and in shape interface also draw(), move(), erase() are logically related to shape formatting.

Cohesion

1. **Temporal cohesion:** Temporal cohesion is present in a class if some of its methods are executed in the same time span . This can be seen in the note editor class , where both opennote() and editnote() and performed together for editing a note , and also in savenote() and closenote() when the user closes the note , saving all his work.

2. **Communicational cohesion:** A communicationally cohesive module is one which performs several functions on the same input or output data. The classes, flowchart, textbox and image along with its classes are working on the same input object which is a note.

3. **Functional cohesion:** Functional cohesion can be seen in the authentication module, notebook module as well as the note module. The goal of the notebook module is to provide for functions that allow the user to organize his/her notes according to their liking . For example making new notebooks,new subjects, new topics. . Similarly, All the functions related to creation and modification of notes(create note,insert image,insert flowchart to name a few) are grouped under the note module and all the functions related to authentication of users, logging them in and out of the system are grouped under the authentication module.

Coupling

Coupling: Most Classes are lightly coupled. But tight coupling can be seen between notebook and subject , subject and topic, topic and note, where the creation of latter's object is dependent on the initial one. For example createtopic() in subject class is the method that is going to create a topic object.

1. **Data Coupling:** There is minimal data coupling between modules. The data is only passed while creation of an object or while passing messages to perform a subroutine. One of the few examples of this that can be seen is between note and note editor, where the note class is going to provide the data to note editor class for displaying contents on the screen.

2. **Control Coupling:** Modules do not pass control information to each other. Therefore, control coupling is almost non-existent.

3. **Content Coupling:** This type of coupling can be seen across image , textbox, shape classes as all of them contain the move() method , the code of which is going to be the same for all of them. Similar is the renaming method for the subject , notebook, topic or note class.