iBOARD
MOBILE APPLICATION

A student and teachers mobile blackboard assistant.

Jesus Contreras

**Instructor's Name : ** Professor, Ph.D Chang-Hyun Jo

Table of Contents

Table of Contents

Planning

Wednesday 7/13	Thursday 7/14	Friday 7/15	Monday 7/18	Tuesday 7/19
Tasks #1-6	Tasks # 7-8	Tasks # 9-10	Tasks #11-12	Miscellaneous/Modifications

Monday 7/25	Tuesday 7/26	Wednesday 7/27	Thursday 7/28	Monday-Tuesday 8/1-8/2
Architectural Design	Component Level Design	User Interface Design	Pattern Based Design	WebApp Design/Code/ Miscellaneous/Modifications

- **<u>Tasks Numbers</u>**

1. 10 Functional Requirements (must focus on 4 only).
2. 5 Non- Functional Requirements (must focus on 2 only).
3. 10 Use cases, one for each functional requirement.
4. 1 UC diagram
5. 1 (Analysis) Class Diagram, it will show all candidate and potential classes.
6. 3-5 Class-Responsibility-Collaborator cards for candidate classes, each must show at least 3 responsibilities and have at least 1 collaborator.
7. 1 Entry Relationship Diagram (ERD) from data objects for data identified from Functional Requirements.
8. 1 Activity Diagram for the most appropriate Functional Requirement/Use Case scenario. An Activity Diagram supplements the use case by providing a graphical representation of the flow of interaction within a specific scenario.
9. 1 Swimlane diagram for most appropriate FR/UC
10. 1 state diagram for the most appropriate FR/UC scenario.
11. 2 Data Flow diagrams for most appropriate FR/UC scenario.
12. 1 Analysis Sequence Diagram for showing interaction between actors and the system as a black box; identify these interactions using FRs/UCs.

Project Communication

Project initiation
At the commencement of this application building experiment I brainstormed about how an application would be able to make the college experience easier on students and teachers. Something that would allow a student and teacher to interact without being physically present in class or in front of a desktop/laptop computer. Introducing Iboard, my inception is a mobile application that will have all the functionality of blackboard but on the go. Smart-phones are becoming such a practical part of students' life that the ability to view grades or open up word documents is almost a thing of second nature therefore providing a platform to do so is of essence.

A short description of the application to build
The application to be developed will be a mobile version of blackboard. Thus it will be a mobile application where a student and teacher of a particular class can interact. The user will be able to see a list of the classes they are enrolled in after logging in the login menu (this requires the user to input an exclusive username and password). The interaction includes but is not limited to the viewing of grades for a certain course, creating threads and commenting on them, and reading documents posted by the instructor (if user is student). The instructor has special permissions such as deleting any inappropriate/irrelevant comments students may post in threads and uploading his/her choice of documents. This application will make the student/teacher experience a much better one by making classroom information accessible to student at all times and creating a more interactive class environment.

Requirements gathering: I began by choosing a definition mechanism to facilitate the collaborative requirements gathering and developing a list of functional requirements, such as login and viewing a course. I then narrowed the list down to just ten of the most essential requirements and prioritized that list from most important to least important to the system. Next I made a list of non-functional requirements like security and concurrent users. Once I had that list, I identified the two most important qualities the system should have and made it a priority that my system design exhibit(s) those qualities.

Project Planning

Estimating: The software is estimated (required) to be done in 5-6 weeks. The software timeline is divided into phases, there are two primary phases, analysis and design.

Scheduling:
Weeks 1-2 : This period of time will be used to analyze what I want from the system and the different elements that I have to put forth to make it possible.

Week 3: Designing and planning the database of the system is going to be done in one week.

Week4-5: Designing and programming the software.

Week6: Testing software, test use cases on actual software.

Functional Requirements:

Prioritized Order	*Functional Requirement*	*TEACHER*	*STUDENT*
1	*LOGIN*	*x*	*x*
2	*VIEWCOURSE*	*x*	*x*
3	*SELECTCOURSE*	*x*	*x*
4	*VIEWGRADES*	*x*	*x*
5	*VIEWANNOUNCEMENTS*	*x*	*x*
6	*POST ANNOUNC.*	*x*	
7	*SEARCHCOURSES*	*x*	*x*
8	*POSTGRADES*	*x*	
9	*VIEWDOCUMENTS*	*x*	*x*
10	*UPLOADOCUMENTS*	*x*	
11	*VIEWBOARD*	*x*	*x*
12	*ADDTHREAD*	*x*	*x*
13	*VIEWTHREAD*	*x*	*x*
14	*POSTCOMMETNT*	*x*	*x*
15	*EDITMYCOMMENT*	*x*	*x*

Entities:

CourseList

Course

Login

grades

course documents

person

DiscussionBoard

Thread

Nonfunctional Requirements:

What are acceptable performance boundaries?

Considering this is a mobile application:

Navigating menus should take less than 20 seconds
Extra leeway acceptable for downloading documents
Pages should “fit all” on mobile device (in other words whole page should fit in mobile screen)

For email:

Display should be intuitive i.e. it should be obvious where subject, recipient and message body go.

For search-bar

Search results should return accurate + relevant results
near-instantaneous cached.

For modify content/layout

User should have complete control of modifiable content, should include different color layouts, layout schemes, ease of navigation.

Nonfunctional Requirements:

1. *secure*
2. *concurrent users*
3. *browser compatibility*
4. *private data*
5. *modularity*
6. *ease of use*

This is a data-viewer, *no data-creating.*
-Usefulness for teachers?
-How does information get created?
-If it isn’t useful for teachers, how is it useful for students?

Chapter Use Case

User Case Scenarios
*Unless otherwise noted, Primary Actor is Student and Teacher
*Unless otherwise noted, Goal is OBVIOUS
*"", prereq is Logged In

Use, Primary, Goal, pre, trigger, scenario, exception, priority, when available, freq use, channel to actor, secondary actor, channel to secondary actor, open issues

Login, SelectCourses, ViewGrades
Todo: Ask teacher about channel to channel-to-actor. Is it device, or how user interacts with the use-case (ie. button, textbox).

1. Use Case: Login
Primary Actor: Student
Goal in context: To access the features that the software provides a registered user
Preconditions: Keyboard interface
Trigger: On the "login" window click "login" button
Scenario:

2. User opens login page
Exceptions: If user is already logged in, in this case the user is directed to the "course list" page.
Priority: High
When Available: First Increment or second increment
Frequency of use: Couple times a day
Channel to actor: Two text boxes for entering "user name"; one button for "login"
Secondary actors: N/A
Channels to secondary actors: N/A
Open Issues:

3.Use-Case: ViewCourse

Primary actor: Student

Goal in context: Student can see list of classes

Preconditions: User is logged in.

Trigger: When user logs in the system

Scenario:

1. User Logs in the system
2. User views the list courses he/she enrolled

Exceptions:

N/A

Priority: Essential

When Available: First increment

Frequency of use: Many times a day

Channel to actor: From mobile Internet communications devices with keyboard interface and browser

Secondary actors: Teacher

Channels to secondary actors: computer

4.Use-Case: SelectCourse

Primary actor: Student

Goal in context: Student can see list of classes and select courses

Preconditions: User is logged in.

Trigger: user opens IBoard app, or leaves a “course screen”

Scenario:

1. View courses in a list

2 (optional). User filters list via SearchCourses

3. User clicks on a class
4. Application goes to the ‘course screen’

Exceptions:

1. User has no courses, therefore, cannot progress
2. User types in a course, and filters so there are no courses

Priority: Essential

When Available: First increment

Frequency of use: Many times per day

Channel to actor: From mobile Internet communications devices with keyboard interface and browser

Secondary actors: Teacher

Channels to secondary actors: computer

5. Use Case: ViewGrades

Goal in context: Allows teacher and student to view grades of selected course.

Primary Actor: Teacher and Student

Trigger: On the "course screen", select "Grades " button

Scenario:

1. User selects Grades from course screen

2 (student): If user is student, he sees his own grade

2 (teacher): If user is teacher, he sees all students' grades

Exceptions: N/A

Priority: Essential

When Available: First increment

Channel to actor: From mobile Internet communications devices with keyboard interface and browser

Secondary actors: N/A

Channels to secondary actors: N/A

Open Issues: N/A

6.Use case: ViewAnnouncements

Primary actor: Student

Goal in context: To view announcements posted by instructor

Preconditions: N/A

Trigger: On the "course screen", select "Announcements"

Scenario:

1. Student: observes announcement
2. Student: finds no announcements

Exceptions:

1. N/A

Priority: Essential, must be implemented

When available: First increment

Frequency of use: Many times per day

Channel to actor: via control panel "Course Announcements" button

Secondary actors: N/A

Channels to secondary actors: N/A

7. Use Case: PostAnnouncement
Goal in context: Allows teacher to post course announcement
Primary Actor: Teacher
Preconditions: Keyboard interface
Trigger: On the announcements window click "send" button
Scenario:
1. Teacher views announcements
2. Teacher presses button for new announcement
3. Teacher types the message inside the tex box and presses "send" button.

Exceptions: N/A

Priority: High
When Available: First Increment
Frequency of use: Couple times a day
Channel to actor: A button saying "new announcement", A text box to type comment, and a button saying "send" to send.
Secondary actors: N/A
Channels to secondary actors: N/A
Open Issues: N/A

8. Use Case : SearchCourses
Primary actor: Student
Goal in context: If the user has a lot of courses, they can filter course list by name
Preconditions: Keyboard interface.
Trigger: Student inputs desired text and "searches", i.e. to click or touch on a search button.
Scenario:
1. Student: observes search interface.
2. Student: enters a course to search in text field.
3. Student: selects "search".
4. Student: observes results of search (list of course(es) that match his text).

Exceptions:
1. The searched course does not exist (course doesn't exist or course name misspelled): In this case student receives a message saying that there is no such course exist.

Priority: Low
When available: First increment.
Frequency of use: A couple of times a week,
Channel to actor: Text field/search button.
Secondary actors: Support technician, Professor.

Channels to secondary actors: N/A

Open issues:

1. Should exception return "suggested results" for faulty/inoperable input?

9 .Use Case: PostGrades

Goal in context: Allows teacher to post grades of students

Primary Actor: Teacher

Trigger: On the "course screen", select "Grades " button and edits the cells, and presses "update" button.

Scenario:

1. Teacher selects "Grades"
2. Teacher enters students' grades
3 Teacher presses "update" button

Exceptions: If the teacher enters a letter to a cell, he receives a warning saying "enter a valid number", and the cell becomes empty.

Priority: Essential

When Available: First increment

Channel to actor: From mobile Internet communications devices with keyboard interface and browser

Secondary actors: N/A

Channels to secondary actors: N/A

Open Issues: N/A

10. Use Case: ViewDocuments

Primary Actor: Student and Teacher

Goal in context: Student wishes to view document for a particular course in which they are enrolled.

Preconditions: That the sought after document has been uploaded by instructor

Trigger: On the "course screen", select "Course Documents"

Scenario:

1. Student: Opens desired document

Exceptions:

1. File type is not supported by device

Priority: Essential, must be implemented

When Available: First increment

Frequency of use: High frequency after teacher uploads document

Channel to actor: Via course documents button

Secondary actors: N/A

Channels to secondary actors: N/A

Open Issues:

1. Which file types are permitted to be uploaded by teacher?
2. What should be file size limit?

11. Use Case: UploadDocuments
Primary Actor: Teacher
Goal in context: Teacher wants to distribute assignments, syllabus, etc
Preconditions: The document exists. File size does not exceed threshold. Only certain files are permitted to be uploaded.

Trigger: “upload document”. Teacher selects a document to upload.
Scenario:
1. On the “course screen”, select “Course Documents”
2. Teacher presses “upload document” button
3. Teacher selects the documents to upload
4. Teacher presses “upload” button

Exceptions:
1.

Priority: Low
When Available: Second increment
Frequency of use: A few times a week
Channel to actor: Via “upload document” button
Secondary actors: N/A
Channels to secondary actors: N/A
Open Issues:
1. Which file types are permitted to be uploaded by teacher?
2. What should be file size limit?

12.Use Case: ViewBoard
Primary Actor: Student and teacher
Goal in context: View list of threads in a course discussion.
Preconditions: N/A
Trigger: In the course screen “discussion board” button
Scenario:
1. In the course screen user clicks on “discussion board” button. Then he/she views the list threads

Exceptions: N/A
Priority: Low
When Available: Second Increment
Frequency of use: Couple times a day
Channel to actor: A button saying “discussion board” to view list of threads
Secondary actors: N/A
Channels to secondary actors: N/A

Open Issues: N/A

13.Use Case: AddThread
Primary Actor: Student and teacher
Goal in context: Add a specific thread in a course discussion.
Preconditions: N/A
Trigger: In the course discussion screen click on “add thread” button.
Scenario:
1. In the course discussion screen user selects the specific thread to view
Exceptions: N/A
Priority: Low
When Available: Second Increment
Frequency of use: Couple times a day
Channel to actor: A button saying “add thread” to add a new thread to the discussion board.
Secondary actors: N/A
Channels to secondary actors: N/A
Open Issues: If there is no keyboard interface, should the user be able post an empty thread?

14.Use Case: ViewThread
Primary Actor: Student and teacher
Goal in context: View a specific thread in a course discussion.
Preconditions: Thread must exist
Trigger: In the course discussion screen click on the title of the desired thread.
Scenario:
1. In the course discussion screen user selects the specific thread to view
Exceptions: N/A
Priority: Low
When Available: Second Increment
Frequency of use: Couple times a day
Channel to actor: A button saying “discussion board” to view list of threads
Secondary actors: N/A
Channels to secondary actors: N/A
Open Issues: N/A

15.Use Case : PostComment

Goal in context: Students and teacher can contribute a specific thread.

Preconditions: The thread must exist.

Primary Actor: Student and Teacher

Trigger: User clicks on “post comment” button/link.

Scenario:

1. The user views a specific thread
2. The user presses “add comment” button
3. The user types comment
4. The user presses “post comment” button

Exceptions: N/A

Priority: Low

When Available: Second Increment

Frequency of use: Couple times a day

Channel to actor: A text box to type comment, and a button saying “post comment”

Secondary actors: N/A

Channels to secondary actors: N/A

Open Issues:

1. Should there be a limit to number of comment

2. Should the comment have limited number of characters?

16.Use Case : EditMyComment

Goal in context: Students and teacher can edit a comment that they sent

Preconditions: The comment must belong to the user.

Primary Actor: Student and Teacher

Trigger: User clicks on “edit comment” button/link.

Scenario:

1. The user views his own comment
2. The user presses “edit comment” button
3. The user edits the existed comment
4. The user presses “update comment” button

Exceptions: N/A

Priority: Low

When Available: Second Increment

Frequency of use: Couple times a day

Channel to actor: A text box to type comment, and a button saying “edit comment”
Secondary actors: N/A
Channels to secondary actors: N/A
Open Issues: N/A

CRCs

We download course documents

Class: CourseList *Populated with a list of courses. Useful for conveniently storing what classes a Person is participating in. Can optionally add implementation to search for classes*	
Responsibility:	**Collaborator:**
Viewing a list of courses	Course.GetTitle() Person.MyCourses
Display Courselist Click on a course to view it	

Class: Course *A course. Students gain access to information such as documents. Teachers have special permission to change course.*	
Responsibility:	**Collaborator:**
Goto my Announcements, Grades, Course Documents, Discussion	
TeacherID StudentIDList GetTitle	

Class: Announcement *Teacher can upload announcements. Students can view annoucnements*	
Responsibility:	**Collaborator:**
Create AnnouncementComment	
View AnnoucementComment List of AnnouncementComment strings	

Class: Login *User inputs login information* *If the login/password combination is correct, it assigns the Person's unique characteristics for the session*	
Responsibility:	**Collaborator:**
Login User	Person.LoginDetails
Display Login Screen	

Class: CourseDocuments *Contains important downloadable documents, such as syllabus, lecture material, etc*	
Responsibility:	**Collaborator:**
Lists Downloadable documents	
Display Document List Download Document	
Upload Document	User.uniqueID Course.TeacherID

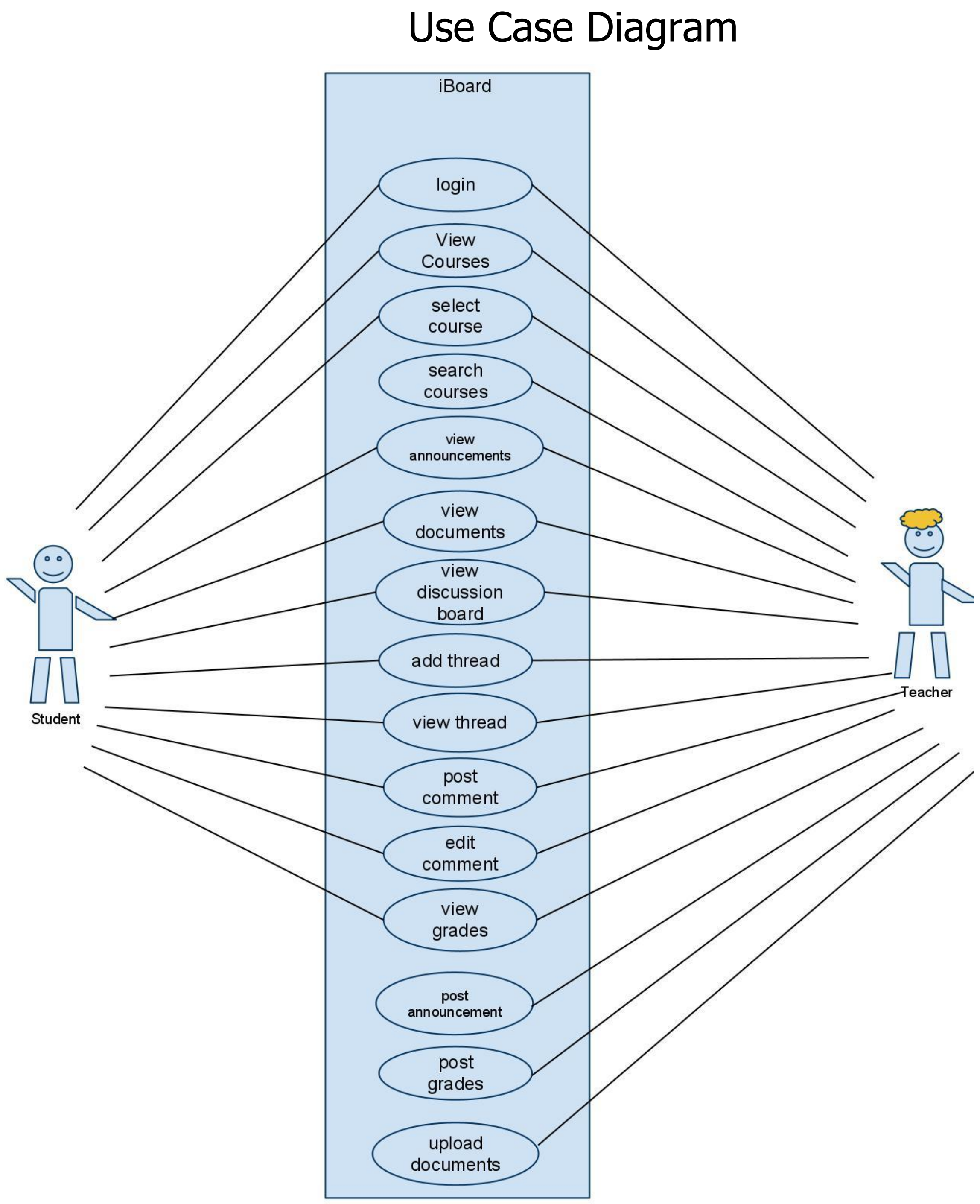
Class: Grades *Contains grades of students. Teachers can change grades. Students can view their own grades*	
Responsibility:	**Collaborator:**
DisplayMyGrades DisplayAllGrades Update Grades	Person.uniqueID Person.uniqueID, course.TeacherID Person.uniqueID, course.TeacherID
Array of grades	

Class: person *Contains login information and access-level stuff. Generic for students and teachers. aka identity*	
Responsibility:	**Collaborator:**
PrimaryID	
Contains list of courses that the specific user is enrolled in. Login Details	

Class: DiscussionBoard *Allows students to view all threads in a course*	
Responsibility:	**Collaborator:**
DisplayDiscussionBoard	Thread.Title
CreateThread GotoThread	Thread Thread
List of Threads	

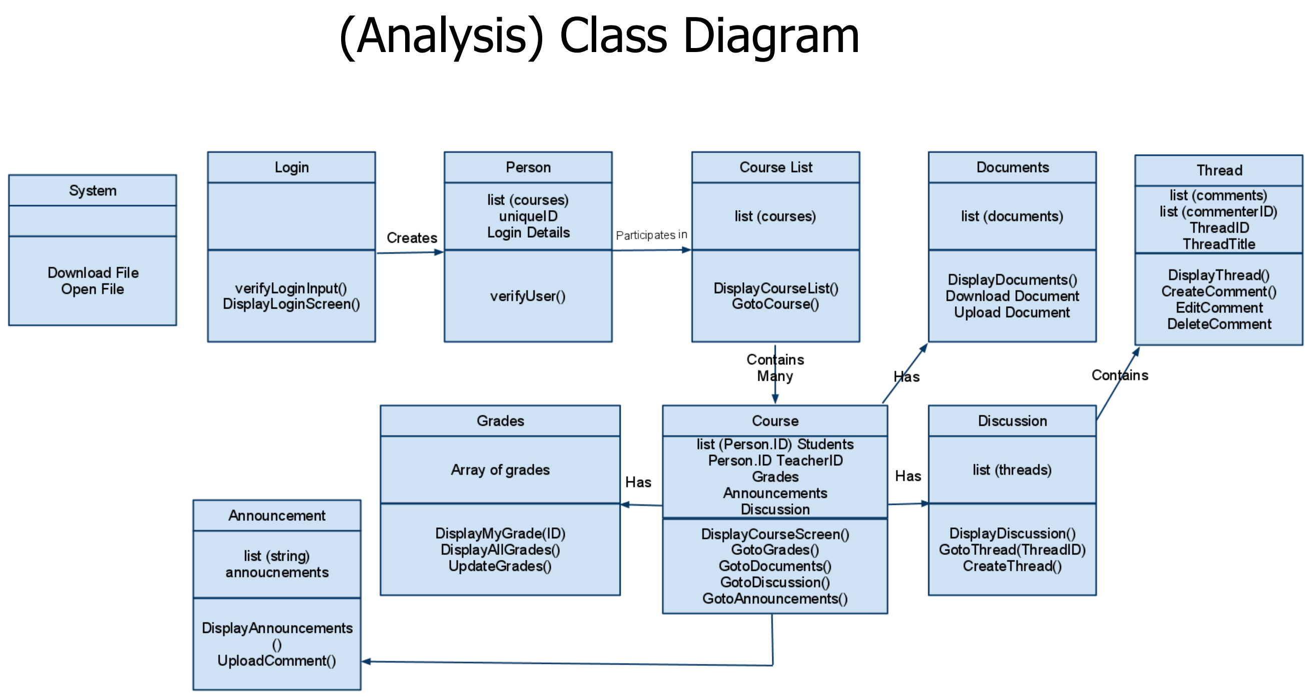
Class: Thread Allow users to view threads in a discussion.*Posters can edit/delete their own comments. Users can post comments. Teacher can obliterate student comments to kingdom come*	
Responsibility:	**Collaborator:**
Postcomment()	Identity.ID
Editcomment()	Identity.ID Course.TeacherID / StudentIDList
Deletecomment() *if the root comment is deleted, the thread is deleted. Teachers can delete everything, and students can delete their own comments*	Identity.ID Course.TeacherID Course.StudentIDList
PosterList- CommentList- *How the thread stores comments*. *Useful for deleting and editing.* PosterList[0]=”Bob” CommentList[0]=”Hi My name is Bob” If John wants to delete CommentList[0], PosterList[0] must equal John, or John has to be teacher. Since he is neither, the implementation ejects him out of the bridge of death.	Identity.ID

Use Case Diagram



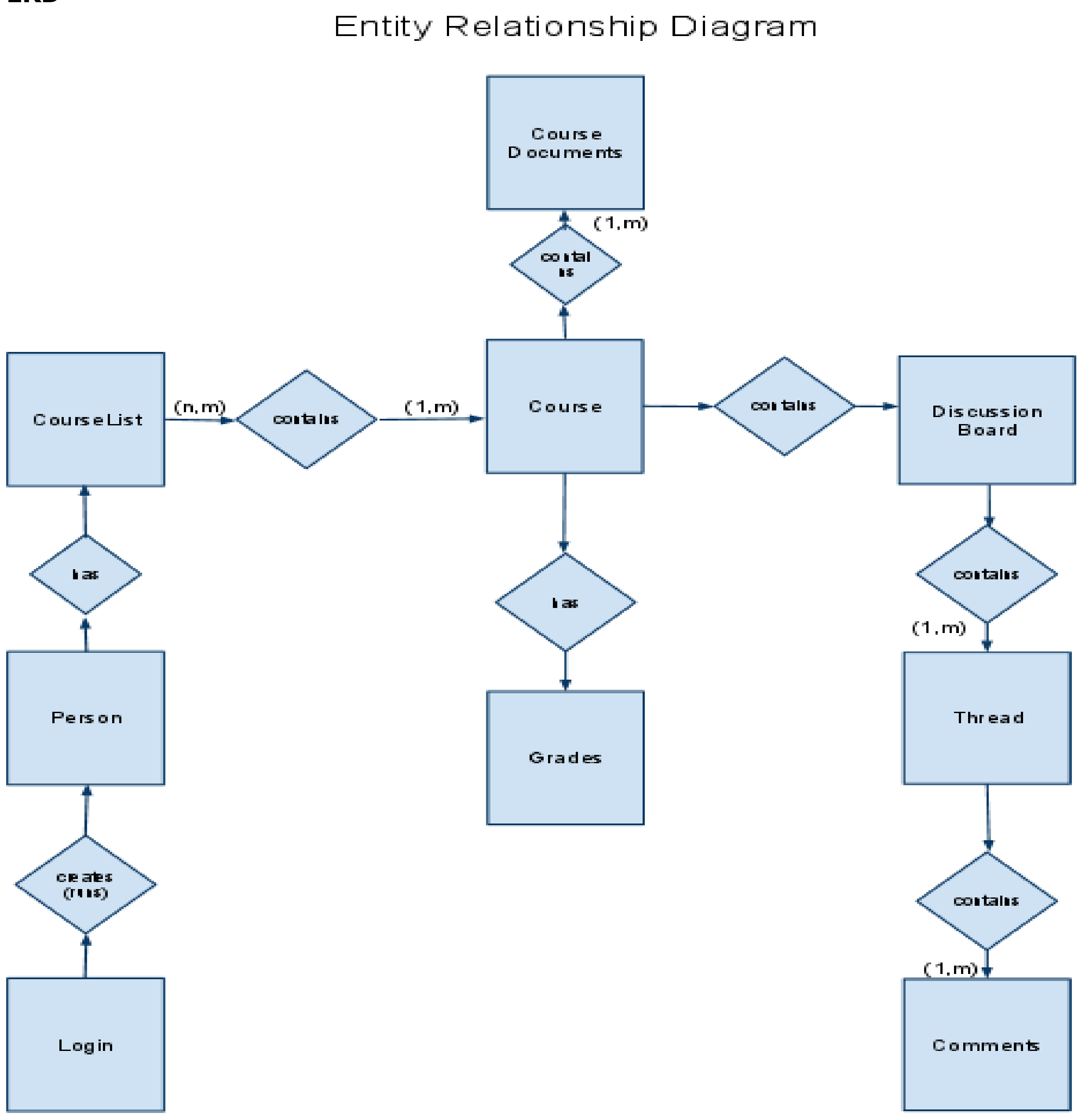
This is a visual diagram of our use cases. Due to the more permissions a teacher (instructor) has he/she has more use cases. This shows what the services are offered from our system to different users.

Class Diagram



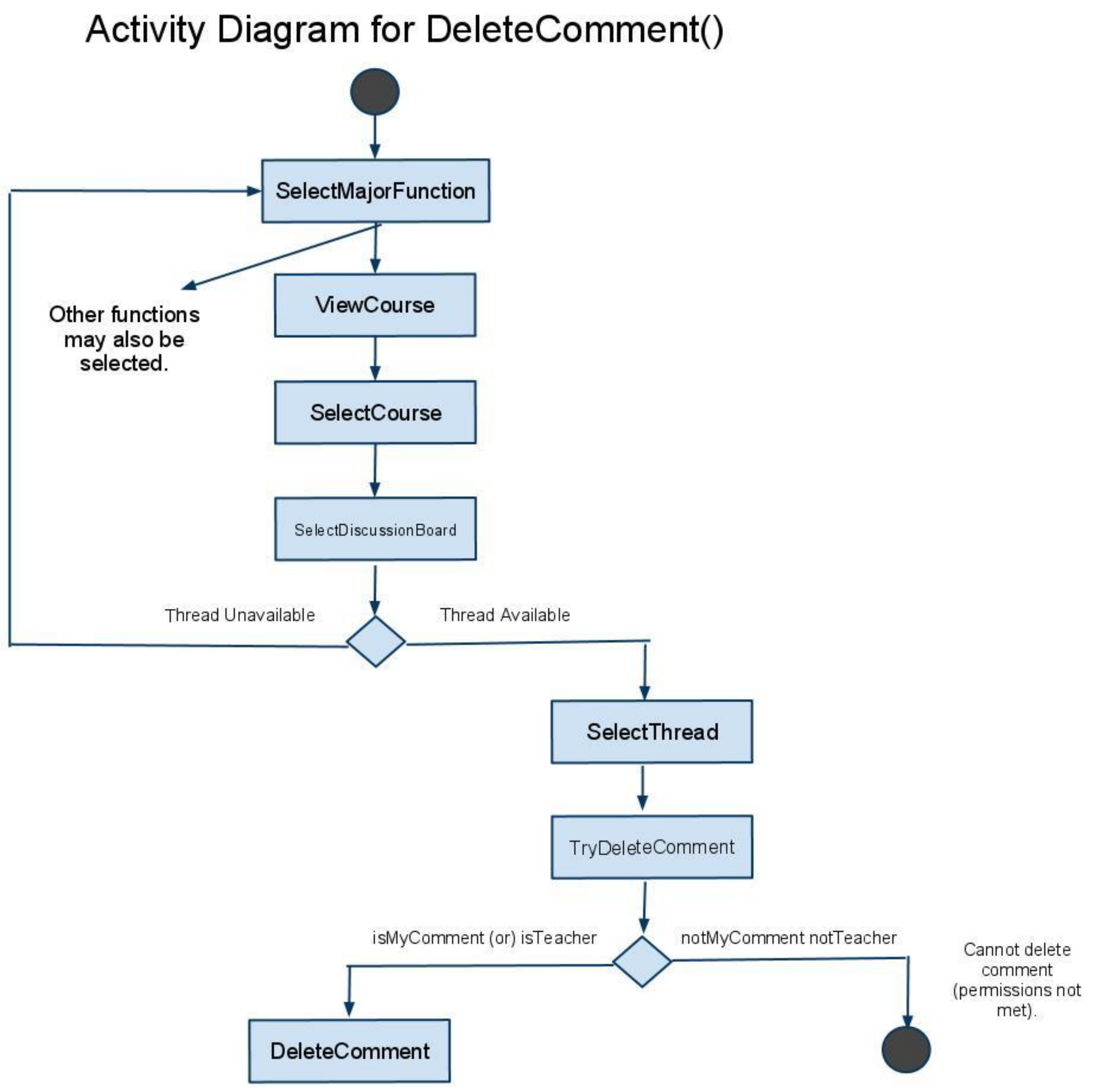
This diagram shows the relationships between different classes. On the "Login" screen, user inputs login/password. If there is a match, it creates a login identity called a "Person" which has unique information such as courselist (a list of courses). A course can be selected - it contains a list of student ids and a teacher id. From the course, users can access grades, announcements, documents, and discussions. Announcements is just a list of strings that the teacher can upload. Course documents is just a list of downloadable files. Grades is a table of grades. Discussion has a list of threads, which are a list of comments. IDs become useful when determining whether users have permission to delete comments, upload documents, view grades, etc.

ERD



The ERD shows how different entities relate to each other. When a user logs in, it creates a unique person. One of the unique attributes is a courselist, which has many courses. Courses create one Grade, Discussion Board, and Documents. Discussions contain many threads, which contain many comments. Unlabelled relationships are implied (1,1).

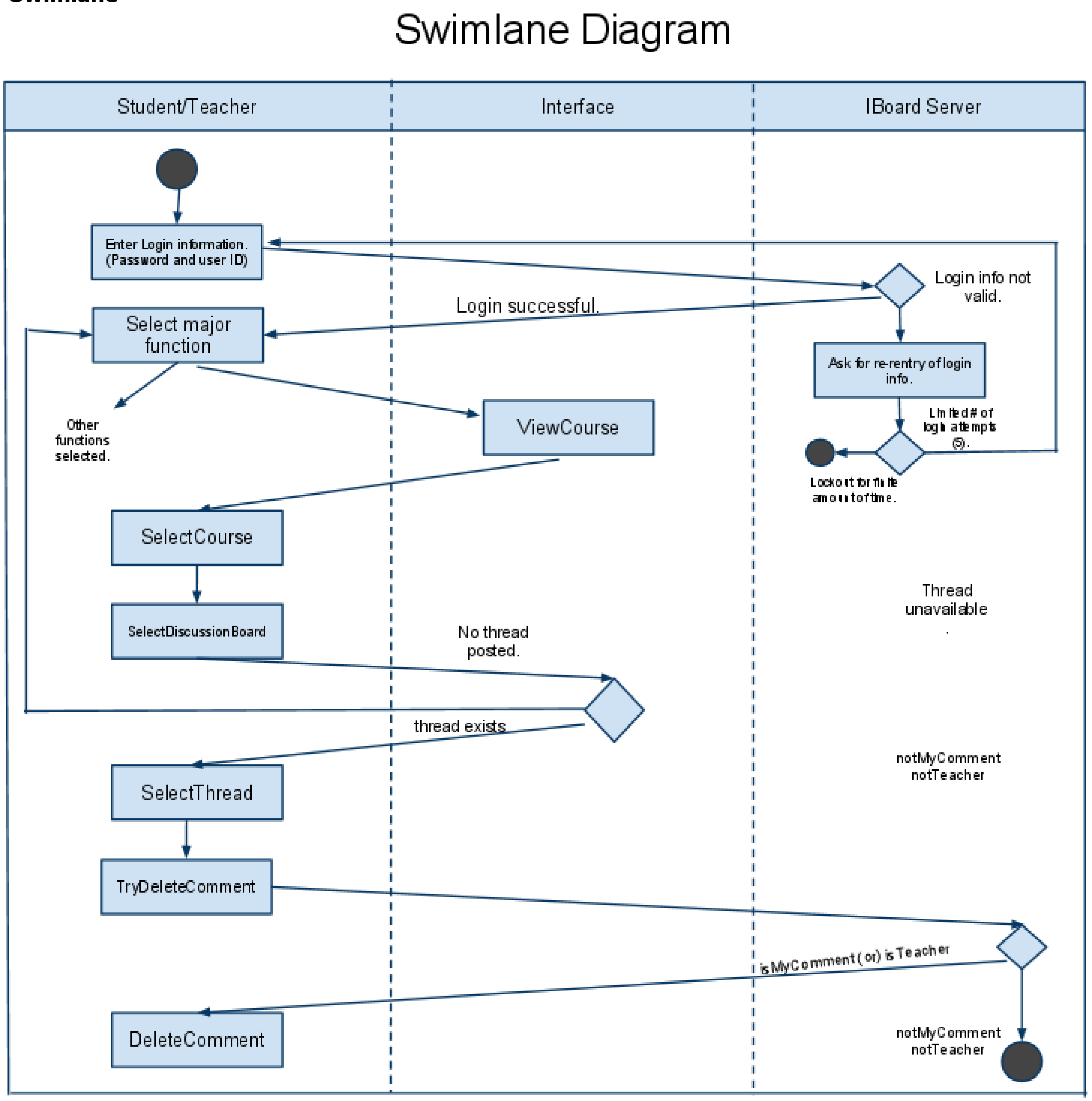
Activity Diagram



This particular activity diagram would be used to design the user interface, and describe the system behavior for a user trying to delete a comment from a course thread.

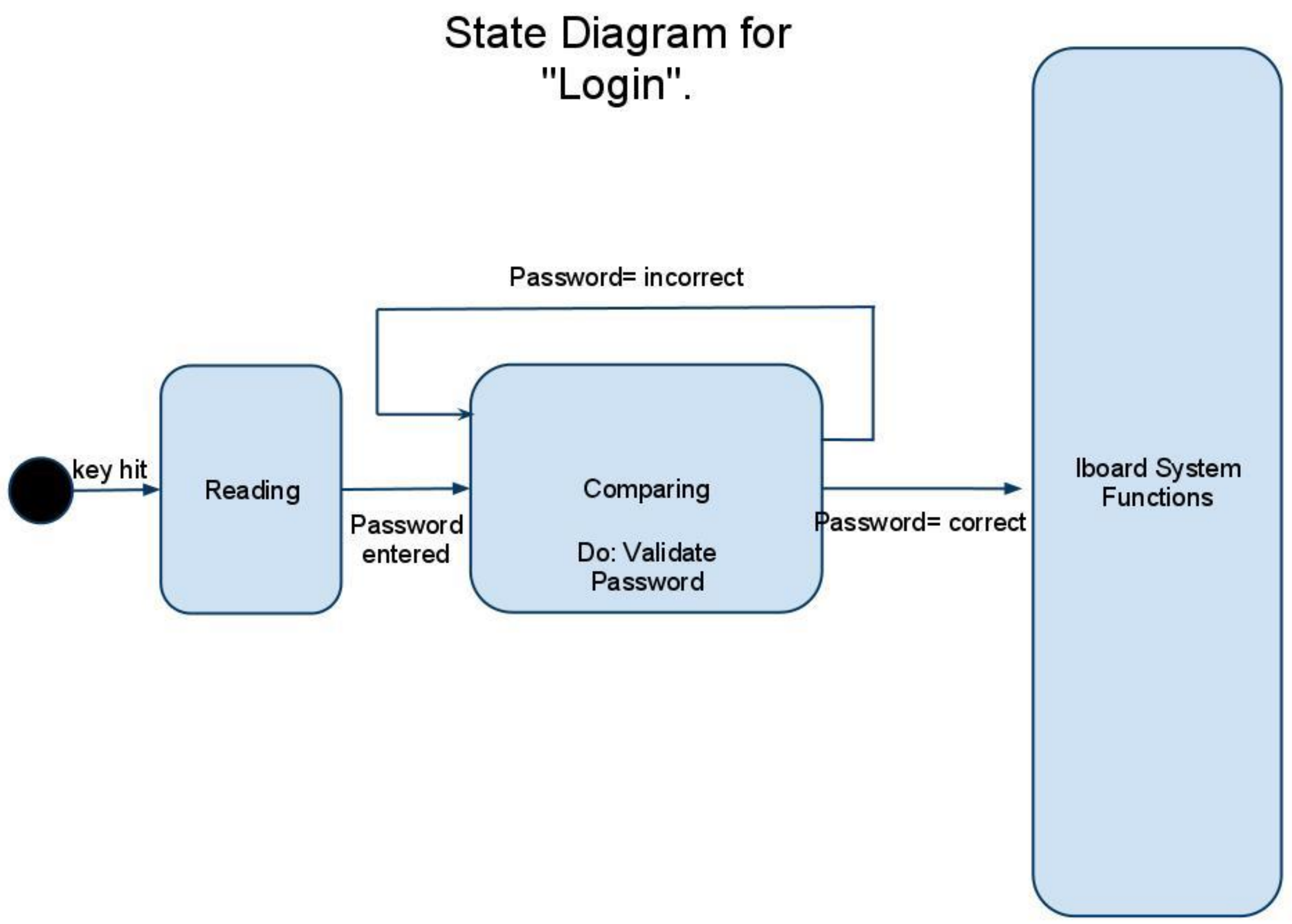
This particular diagram does not include all possible scenarios just a straightforward approach towards deleting a comment.

Swimlane

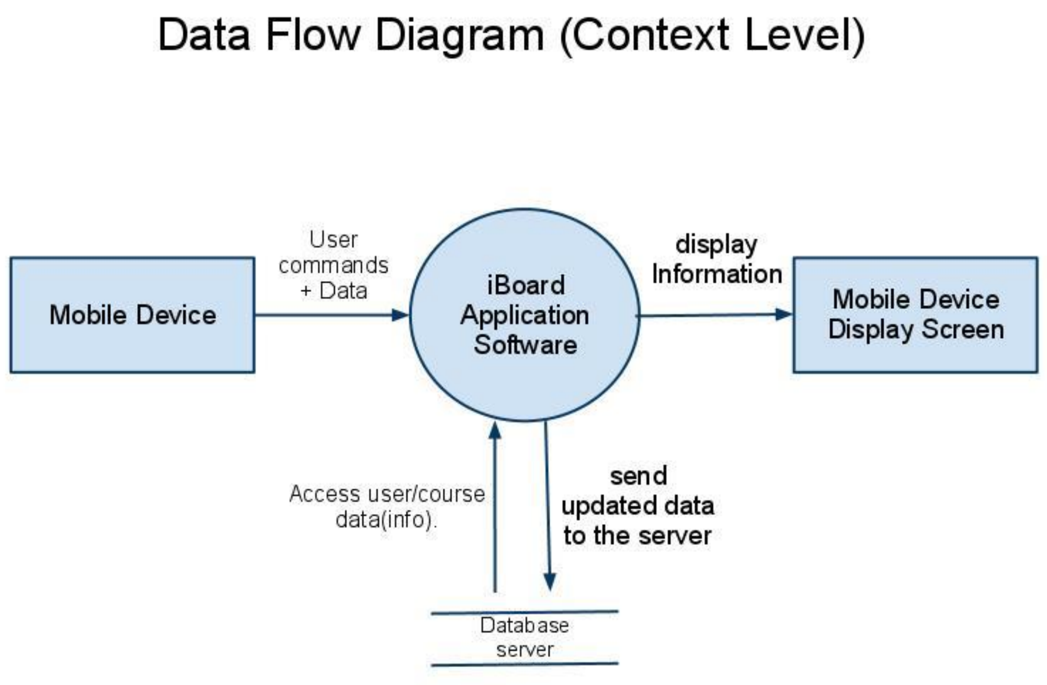


This swimlane diagram demonstrates the deleteComment use case. It indicates which actors or analysis class has responsibility for the action described by an activity rectangle.

Login (ScreenState)

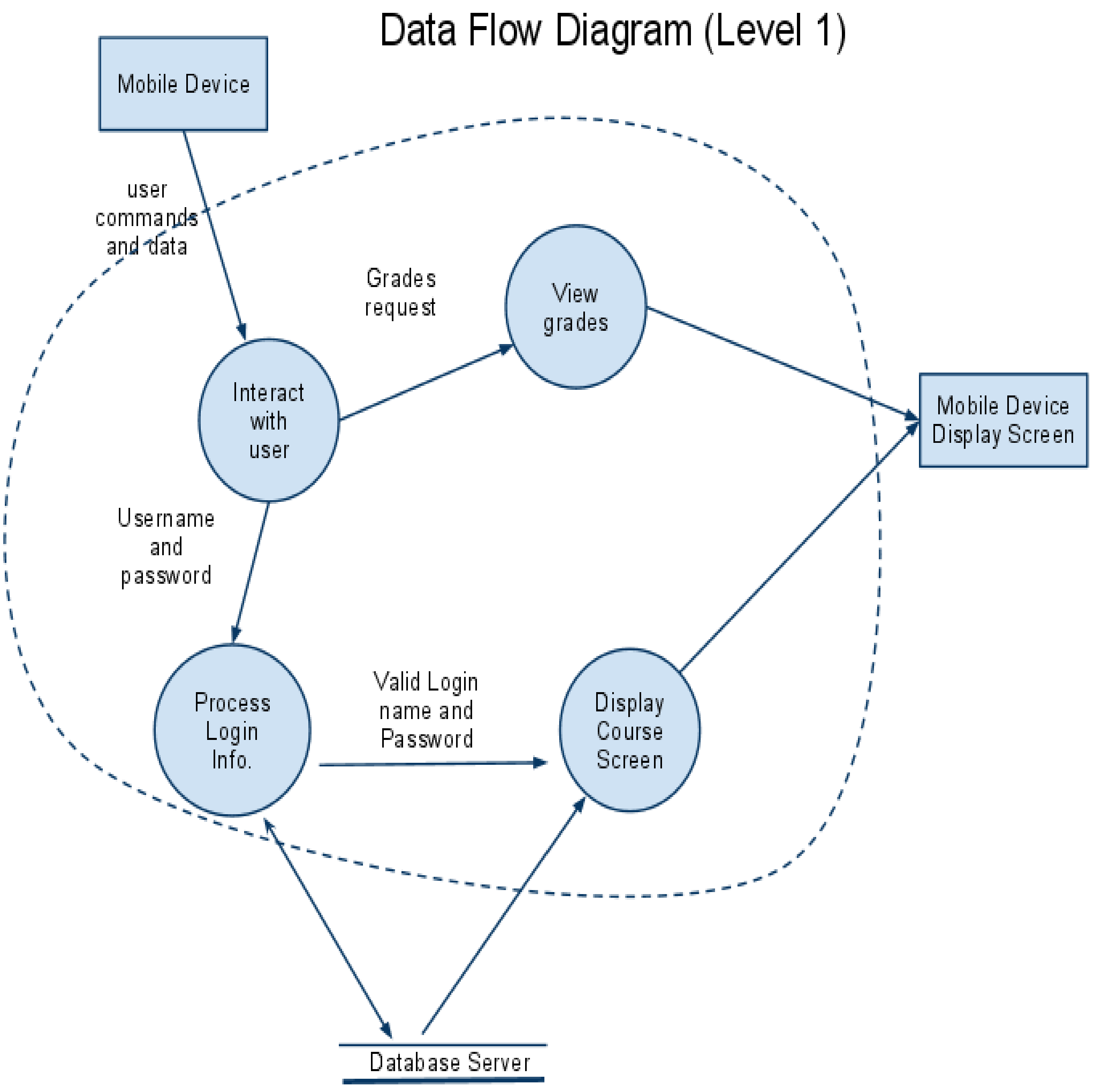


Chapter Data Flow


Data Flow Diagram (Context Level)
Mobile Device
User commands + Data
iBoard Application Software
display Information
Mobile Device Display Screen
Access user/course data(info).
send updated data to the server
Database server

This particular Data Flow Diagram is a model of the information domain and function domain that serves as a semantic bridge between users and systems developers.

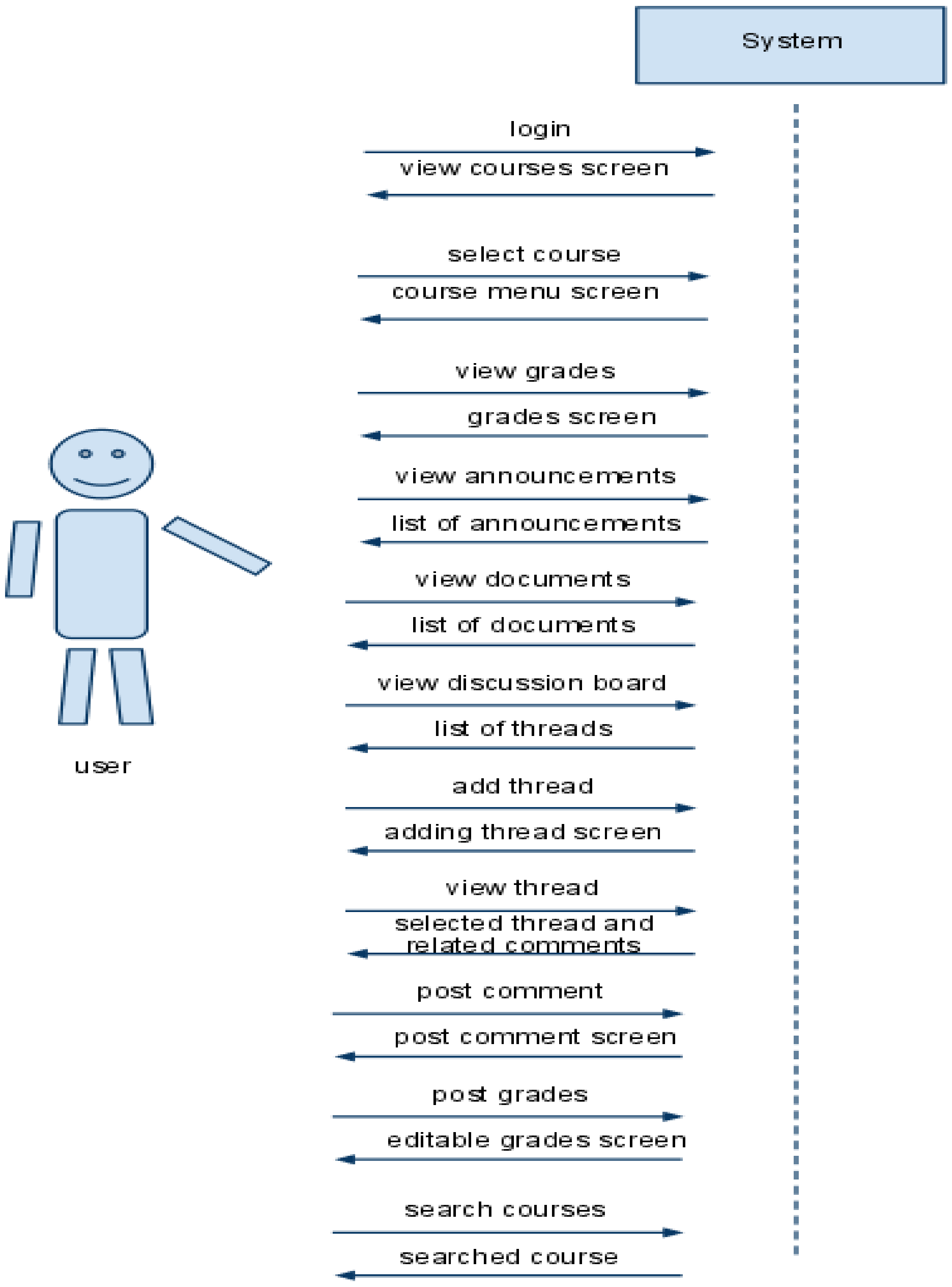
Chapter Data Flow



This particular Data Flow Diagram is an expanded model of the context level information domain and function domain that serves as a semantic bridge between users and systems developers.

Chapter Analysis Sequence

Analysis Sequence Diagram



Analysis sequence diagram shows the interaction between the system and the user. The diagram states what the user expects from a specific function.

Layered Architecture

Pros	Cons
• Re-use and modification of layers accommodates whole layer. • Communicating layers and local dependencies make it easier to detect where bugs occur. • Each layer may hide private information from other layers • Since each layer is set to have related services, architecture provides high cohesion (inside layer).	• Not all systems are structured as layers • Communication from layers must follow strict layer structure (you cannot skip layers). • Processing can only be done layer to layer may lead to performance problems.

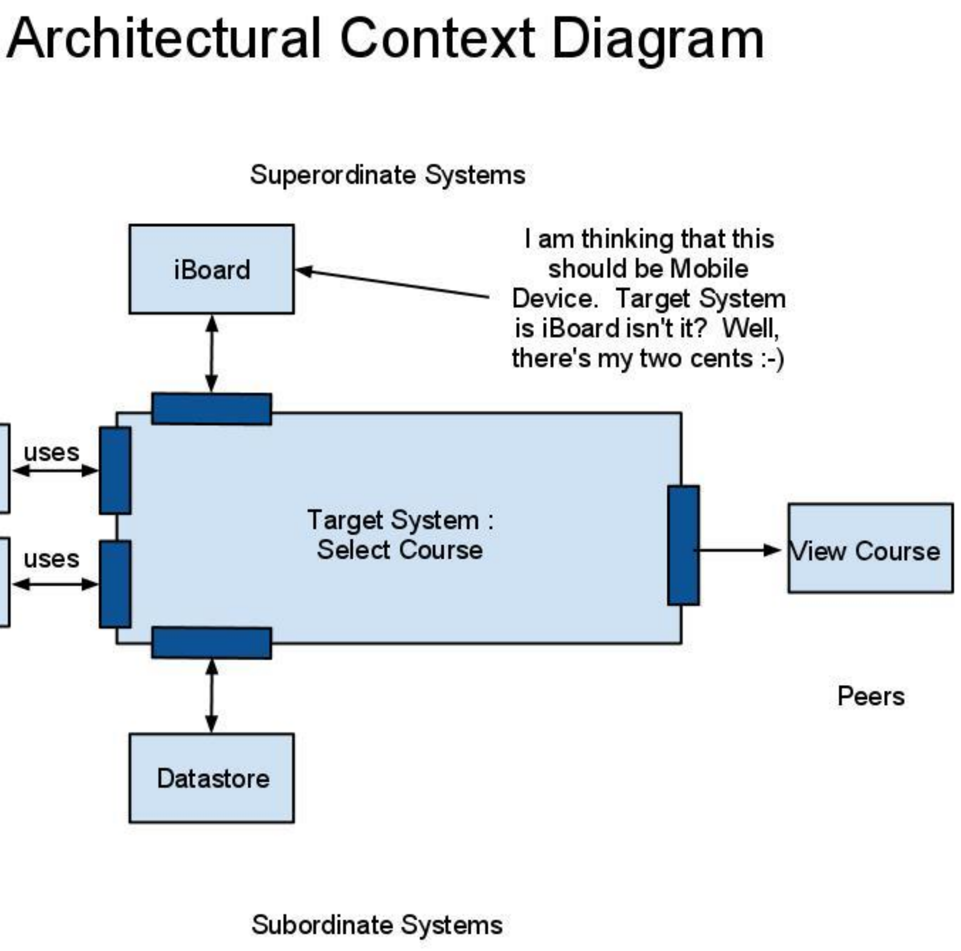
Data Flow (Pipes and Filters)

Pros	Cons
• Parallel processing makes it an efficient style. • Re-use of filters easy. • Modifying filters is easy. • Due to step-wise structure easy to process mathematically. • Easy to understand architectural style. • Filters perform a specific function thus a fairly cohesive style.	• No shared states • Since it has common interface it's efficiency is poor during high traffic. • Fast and simple error processing has to be passed through series of "pipes", thus inefficient error handling. • More on error handling, if an intermediate filter crashes, will ruin flow of data. • Would not be good as interactive application (style).

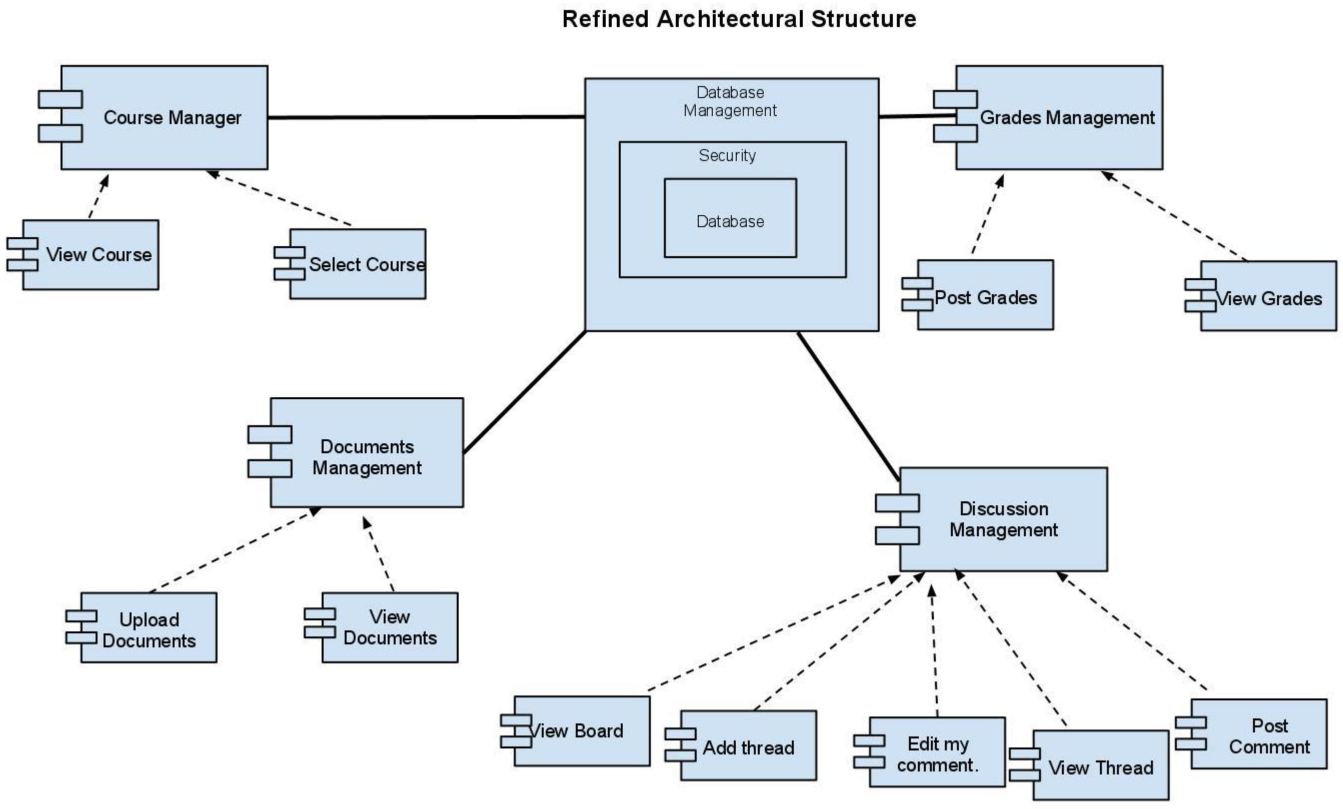
Data Centered

Pros	Cons
• Single data store make the maintenance of data in terms of back up and security easy to manage. • Coupling is restricted to shared data, strong coupling. • Efficient way to share large amount of data. • Data localized in repository module. • Existing components can be modified without concern of other clients.	• If data store fails, all parties are affected and possibly all functions may stop • Will need good back-up and recovery management • Subsystems must agree on a repository data model.

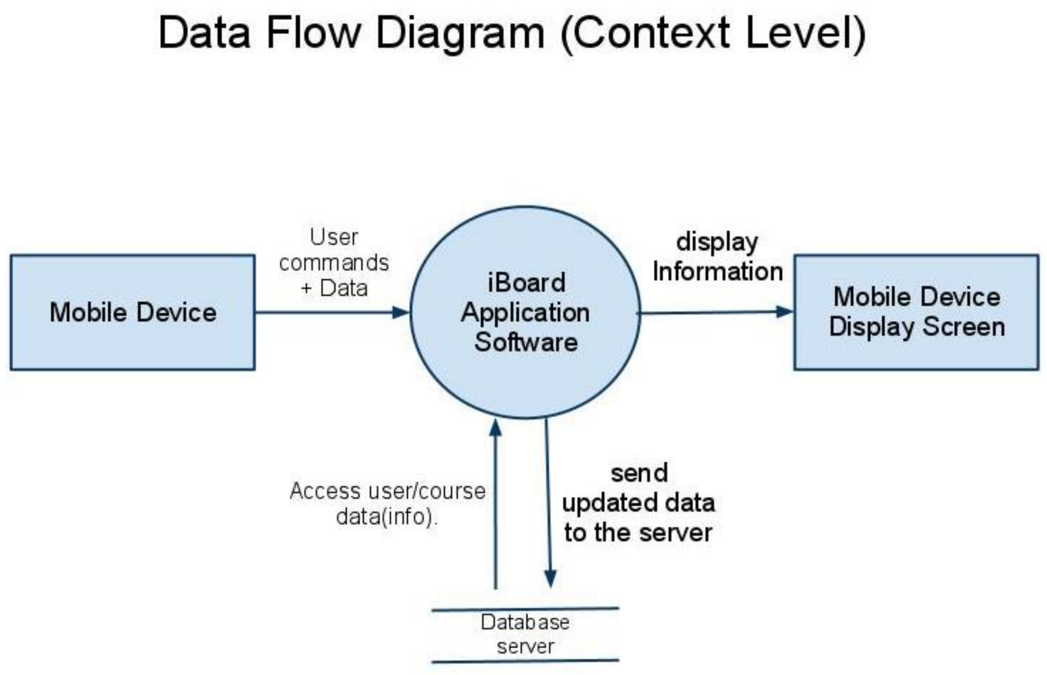
Architectural Context Diagram


Architectural Context Diagram
Superordinate Systems
iBoard
I am thinking that this should be Mobile Device. Target System is iBoard isn't it? Well, there's my two cents :-)
Student
uses
Teacher
uses
Target System : Select Course
View Course
Actors
Peers
Datastore
Subordinate Systems

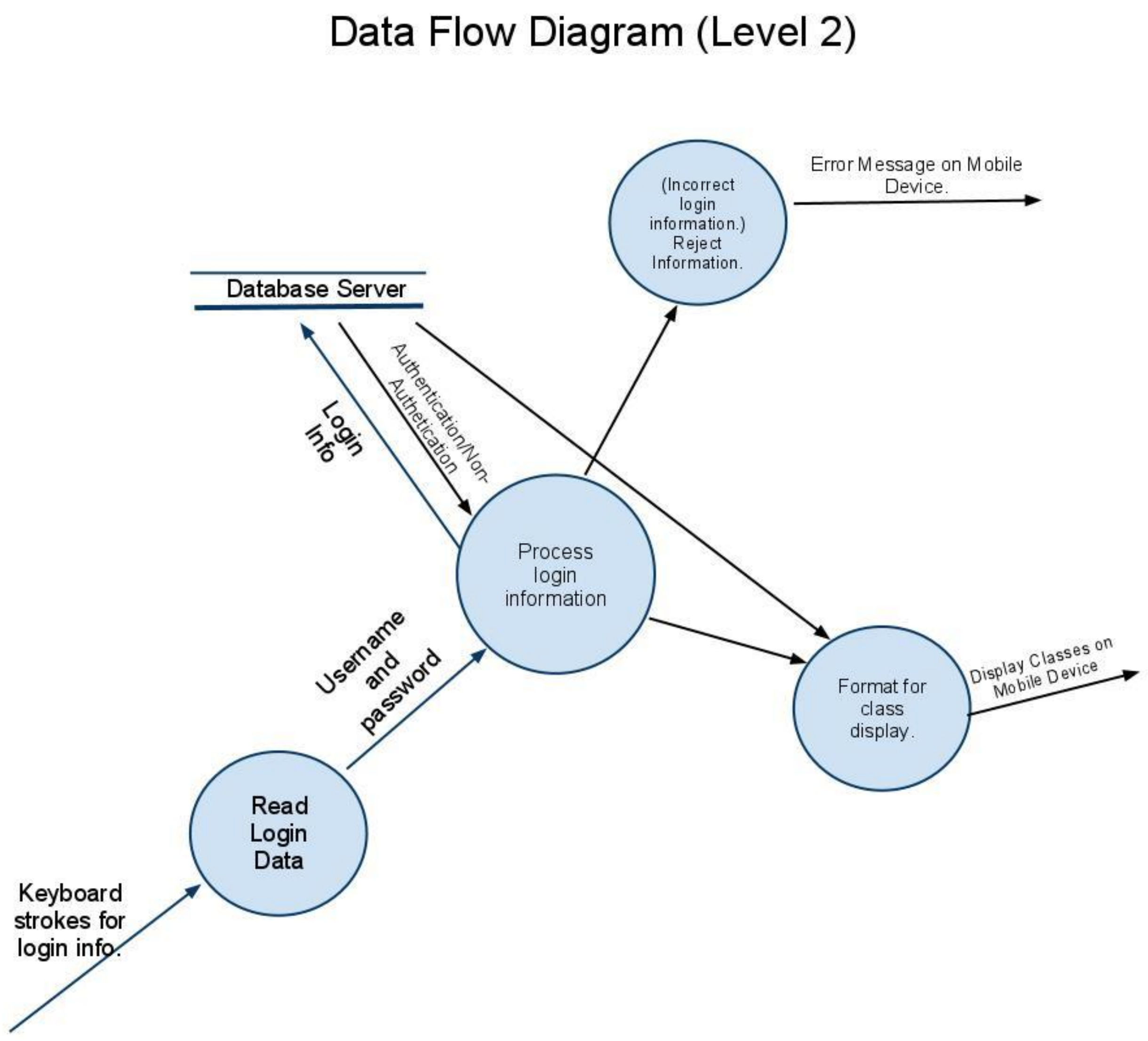
Refined Architectural Structure



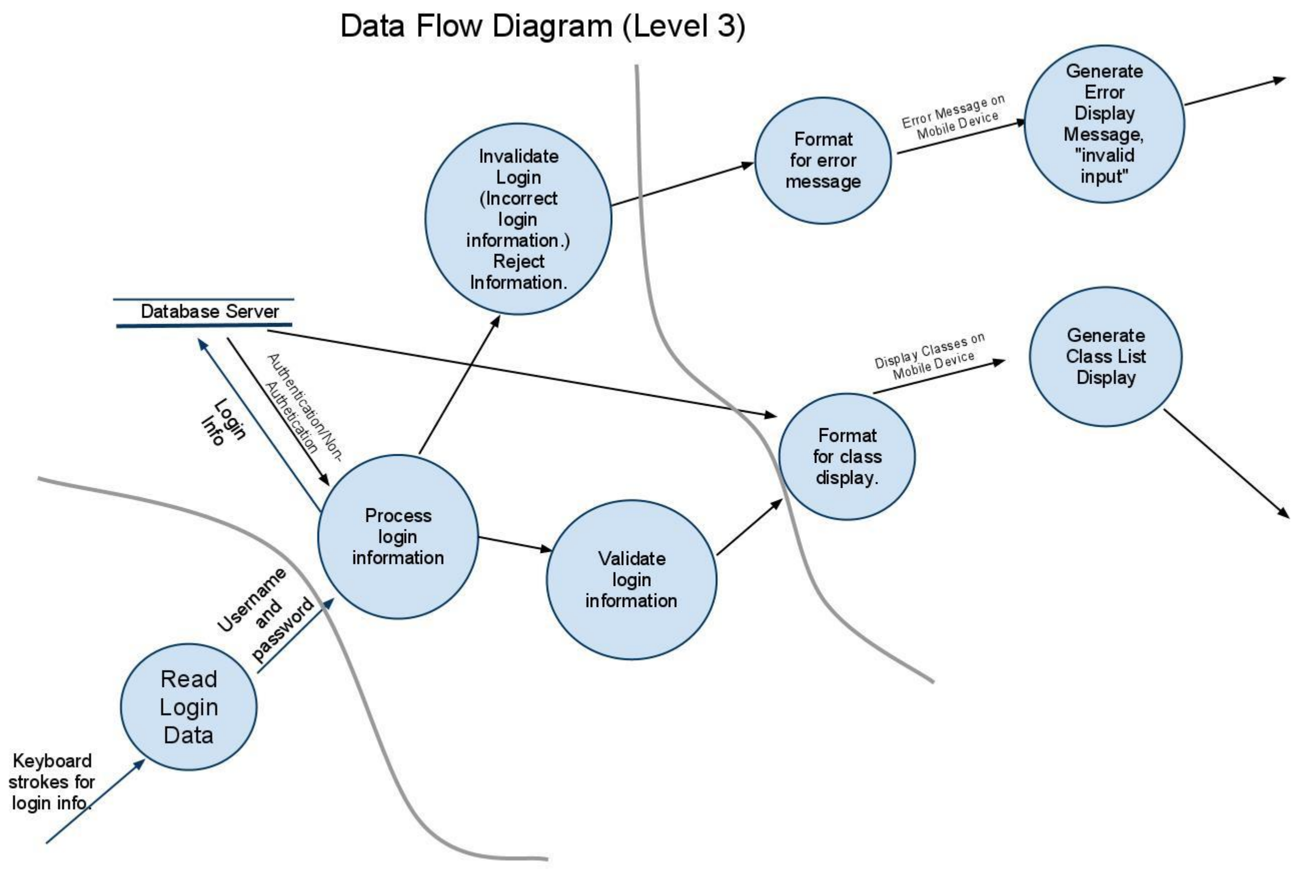
Data Flow Diagram (Context Level)


Data Flow Diagram (Context Level)
Mobile Device
User commands + Data
iBoard Application Software
display Information
Mobile Device Display Screen
Access user/course data(info).
send updated data to the server
Database server

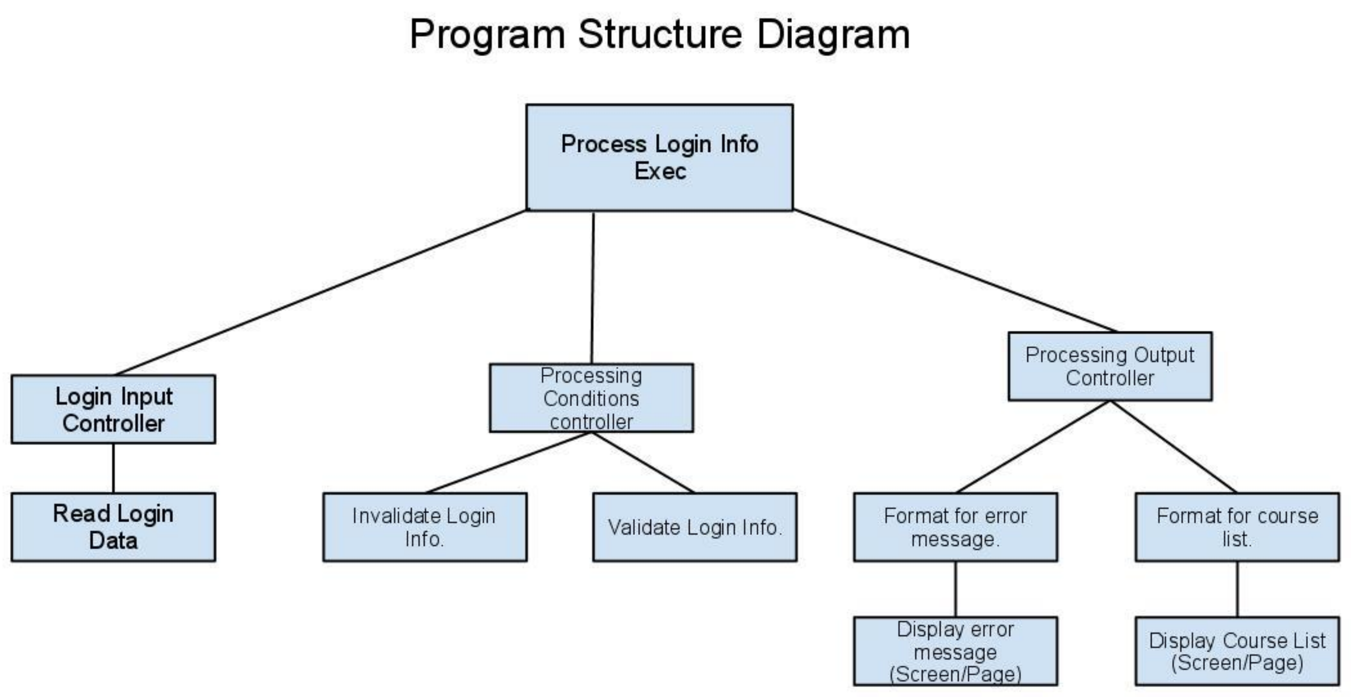
Data Flow Diagram (Level 2)


Data Flow Diagram (Level 2)
(Incorrect login information.) Reject Information.
Error Message on Mobile Device.
Database Server
Login Info
Authentication/Non-Authetication
Process login information
Username and password
Format for class display.
Display Classes on Mobile Device
Read Login Data
Keyboard strokes for login info.

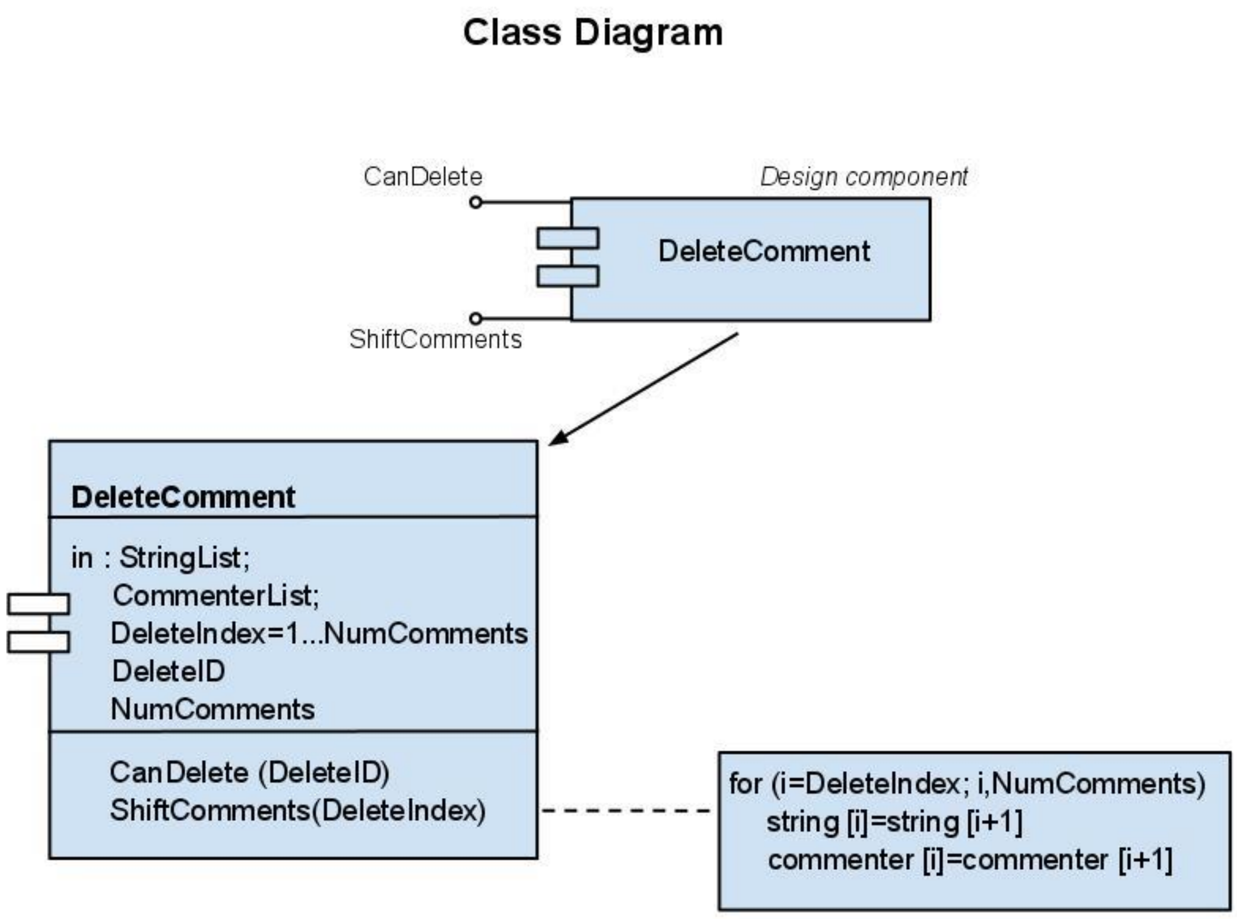
Data Flow Diagram (Level 3)


Data Flow Diagram (Level 3)
Invalidate Login (Incorrect login information.) Reject Information.
Format for error message
Error Message on Mobile Device
Generate Error Display Message, "invalid input"
Database Server
Login Info
Authentication/Non-Authetication
Display Classes on Mobile Device
Generate Class List Display
Format for class display.
Process login information
Validate login information
Username and password
Read Login Data
Keyboard strokes for login info.

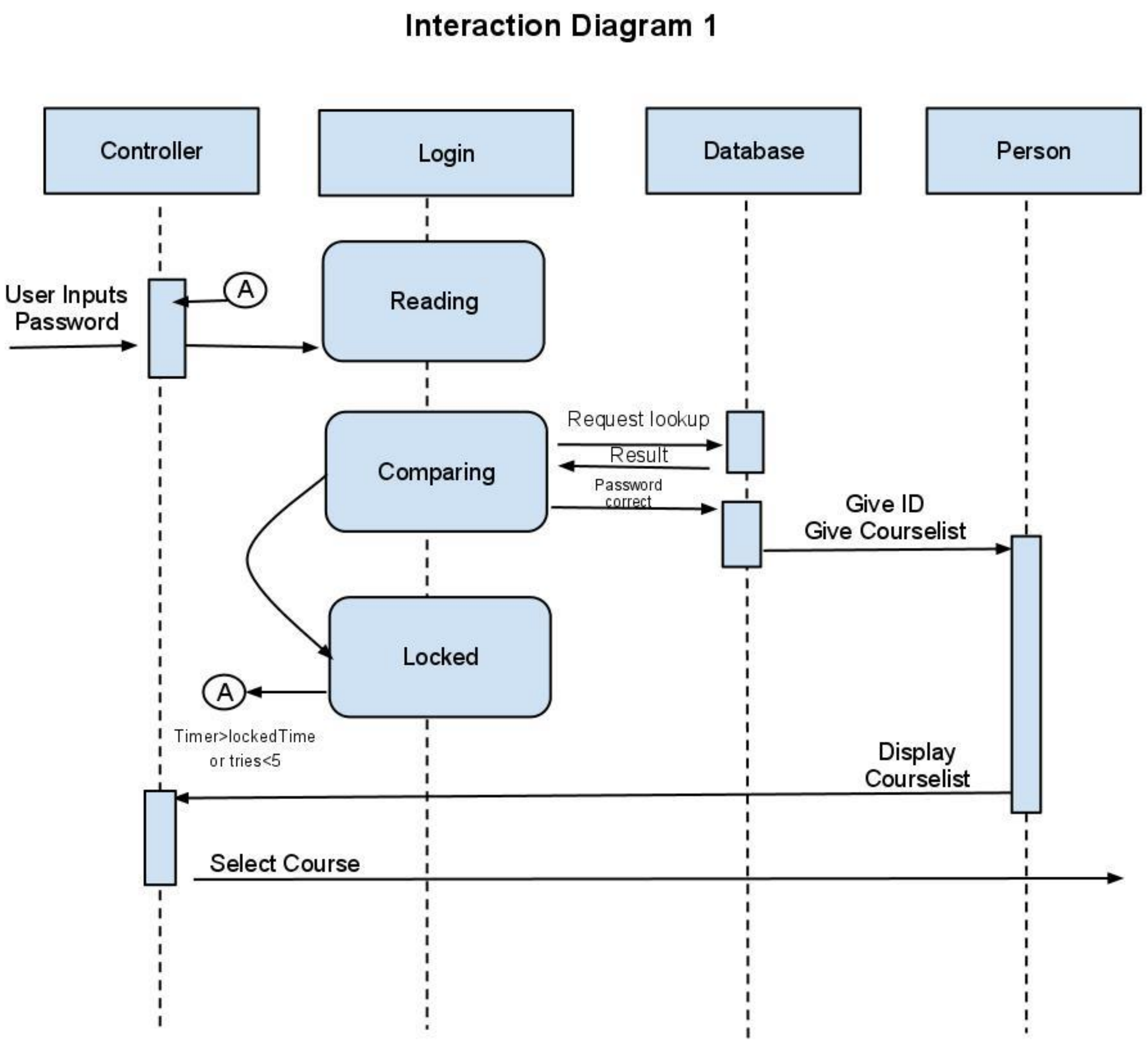
Program Structure Diagram


Program Structure Diagram
Process Login Info Exec
Login Input Controller
Read Login Data
Processing Conditions controller
Invalidate Login Info.
Validate Login Info.
Processing Output Controller
Format for error message.
Display error message (Screen/Page)
Format for course list.
Display Course List (Screen/Page)

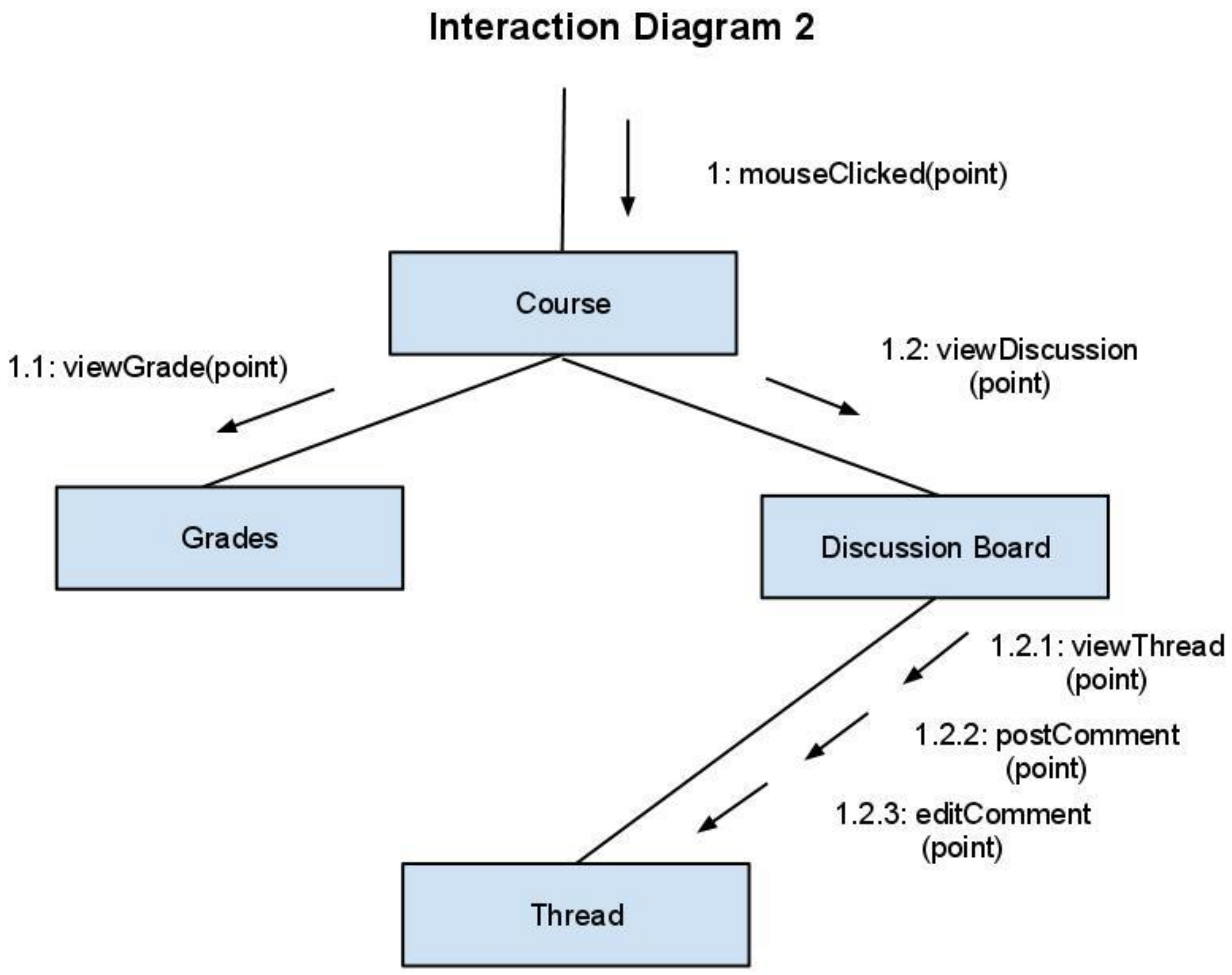
Class Diagram



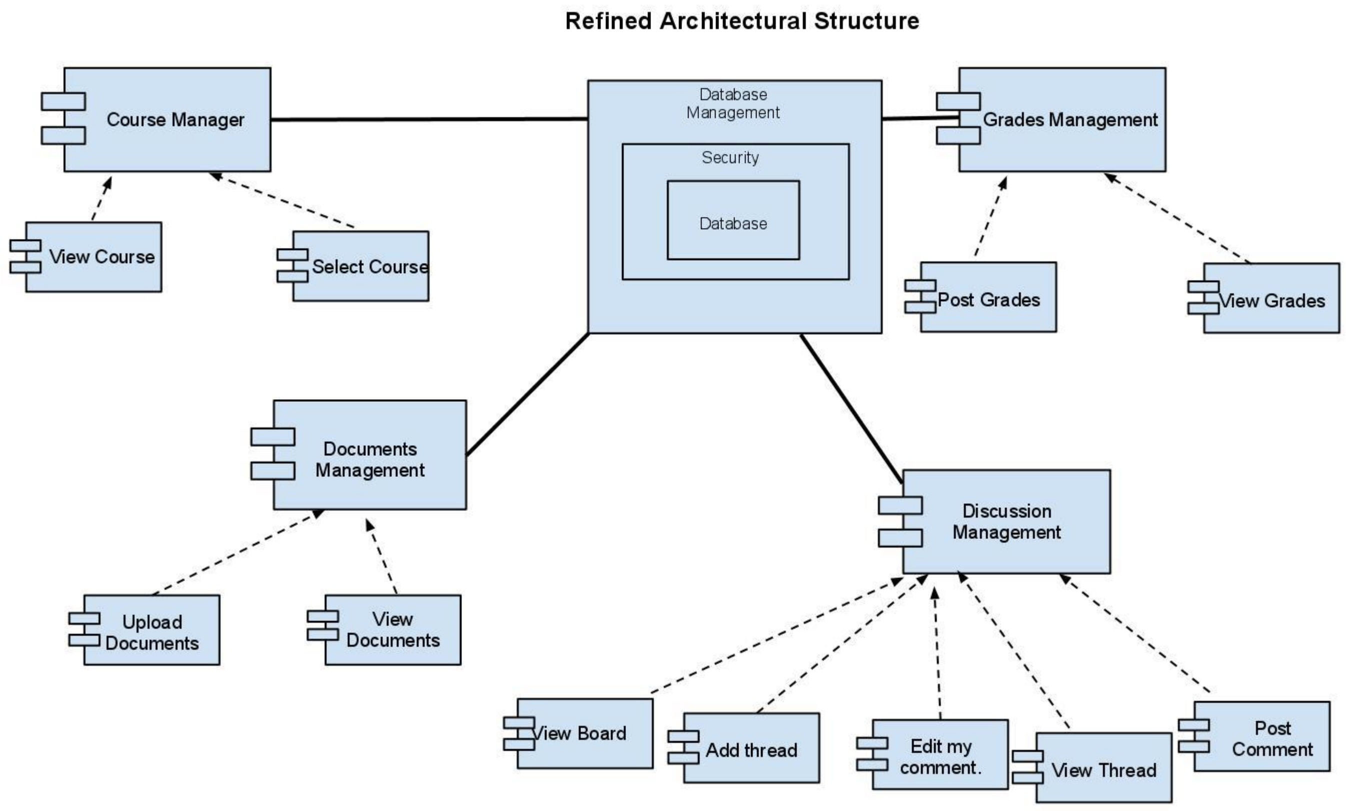
Interaction Diagram - 1



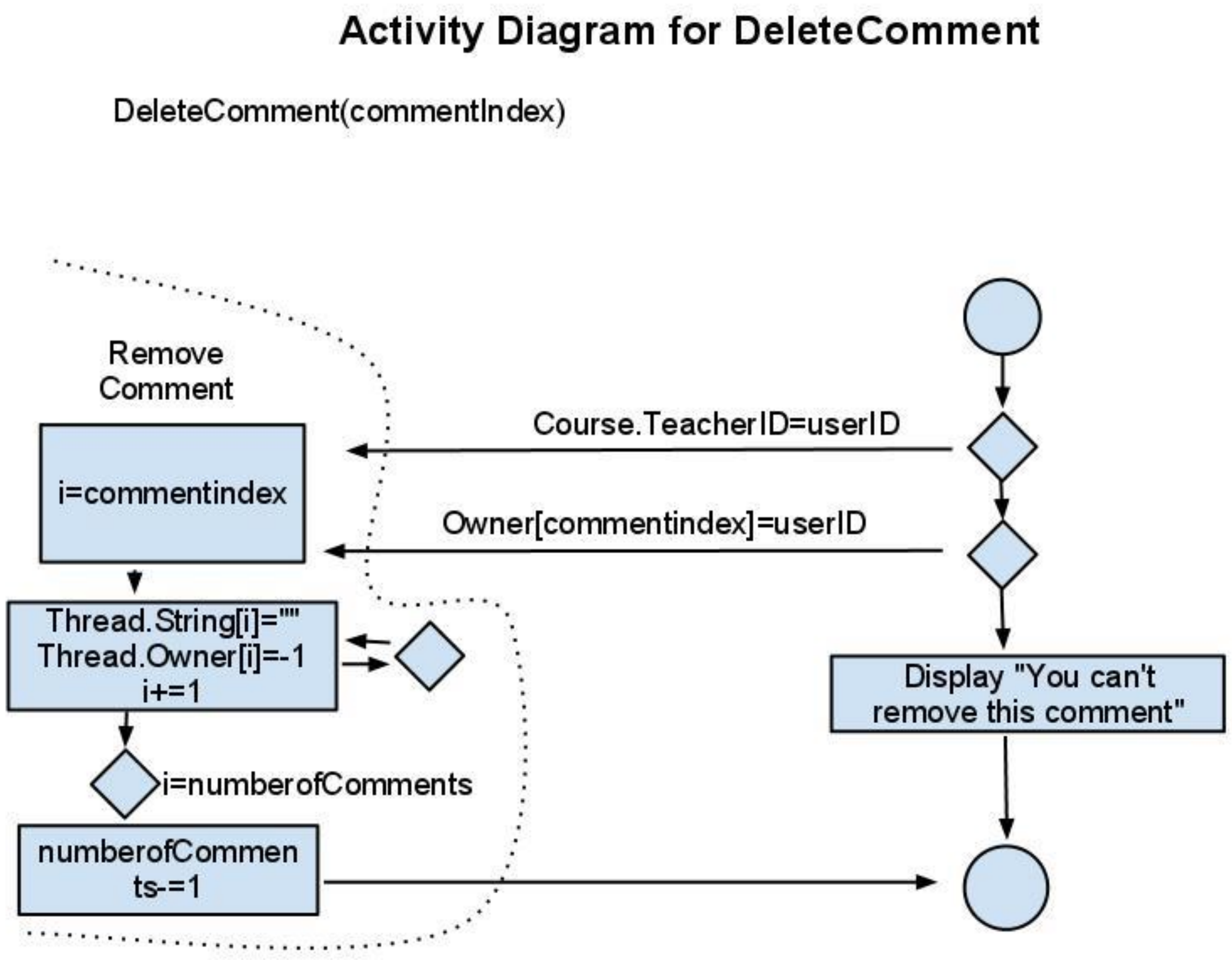
Interaction Diagram - 2


Interaction Diagram 2
1: mouseClicked(point)
Course
1.1: viewGrade(point)
1.2: viewDiscussion (point)
Grades
Discussion Board
1.2.1: viewThread (point)
1.2.2: postComment (point)
1.2.3: editComment (point)
Thread

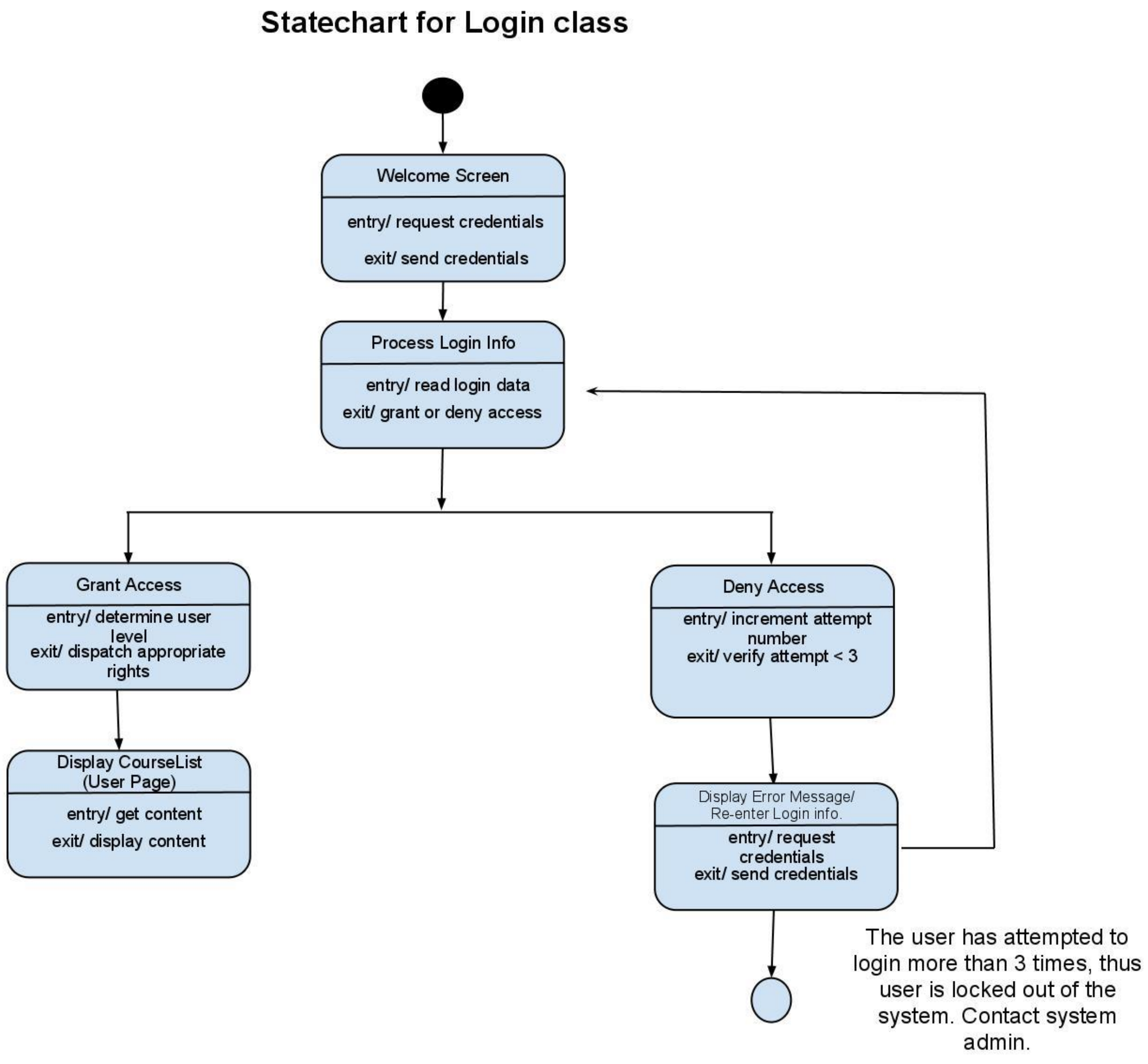
Refined Architectural Structure Diagram



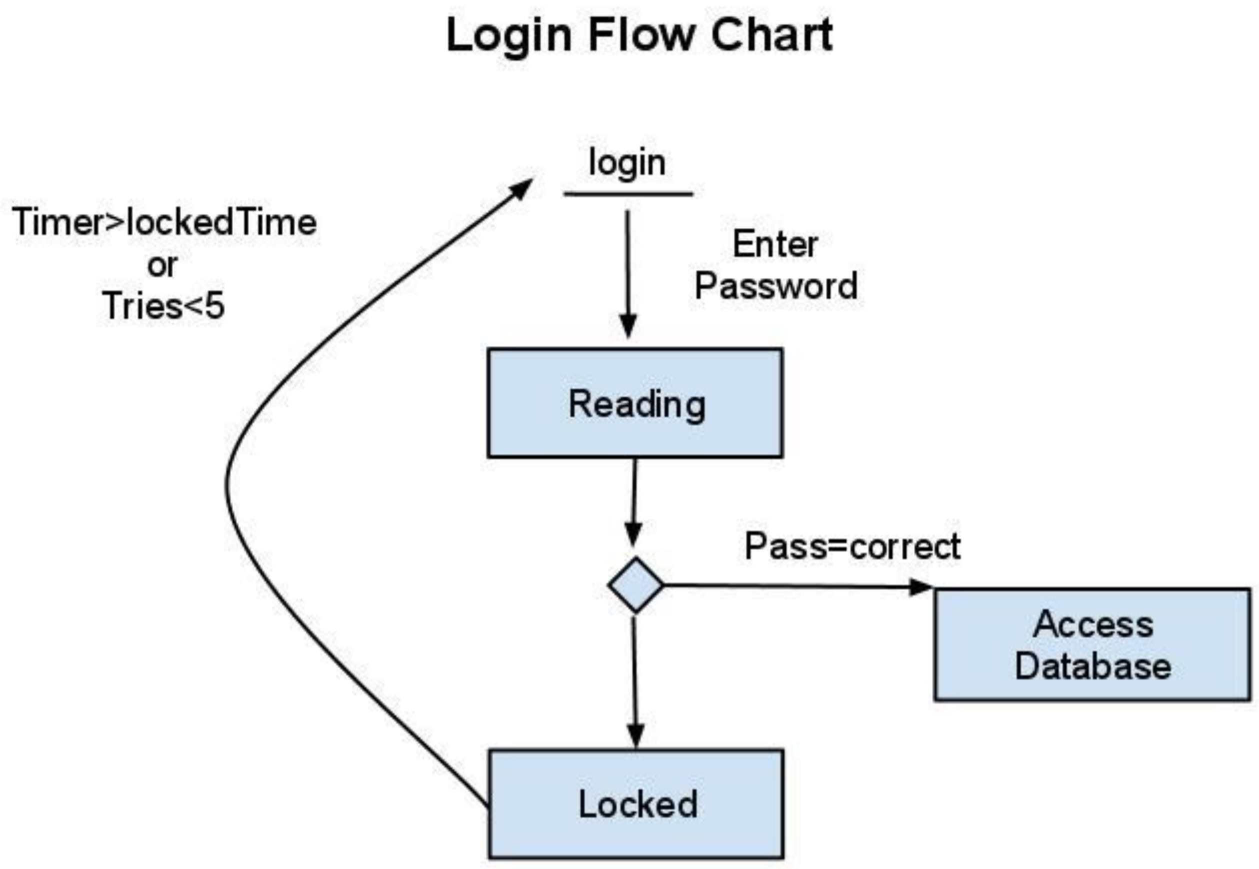
Activity Diagram for Delete Comment



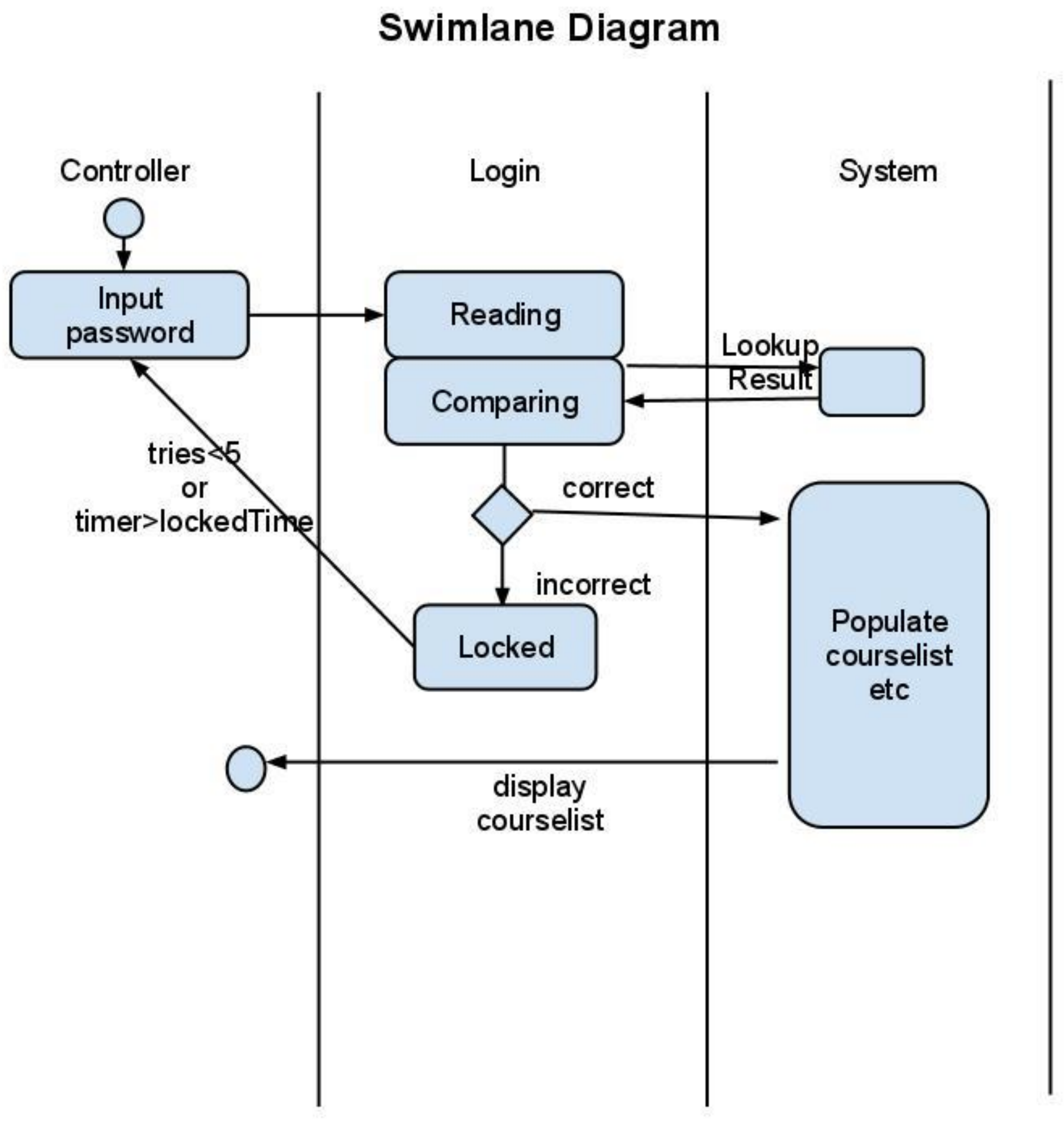
State Transition Diagram


Statechart for Login class
Welcome Screen
entry/ request credentials
exit/ send credentials
Process Login Info
entry/ read login data
exit/ grant or deny access
Grant Access
entry/ determine user level
exit/ dispatch appropriate rights
Display CourseList (User Page)
entry/ get content
exit/ display content
Deny Access
entry/ increment attempt number
exit/ verify attempt < 3
Display Error Message/ Re-enter Login info.
entry/ request credentials
exit/ send credentials
The user has attempted to login more than 3 times, thus user is locked out of the system. Contact system admin.

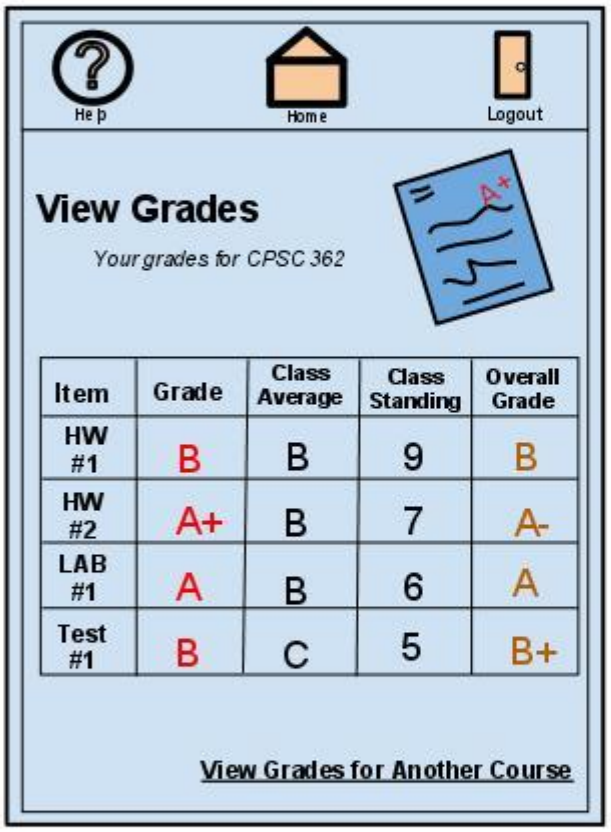
Login Flow Chart



Swimlane Diagram



Preliminary Screen Layout



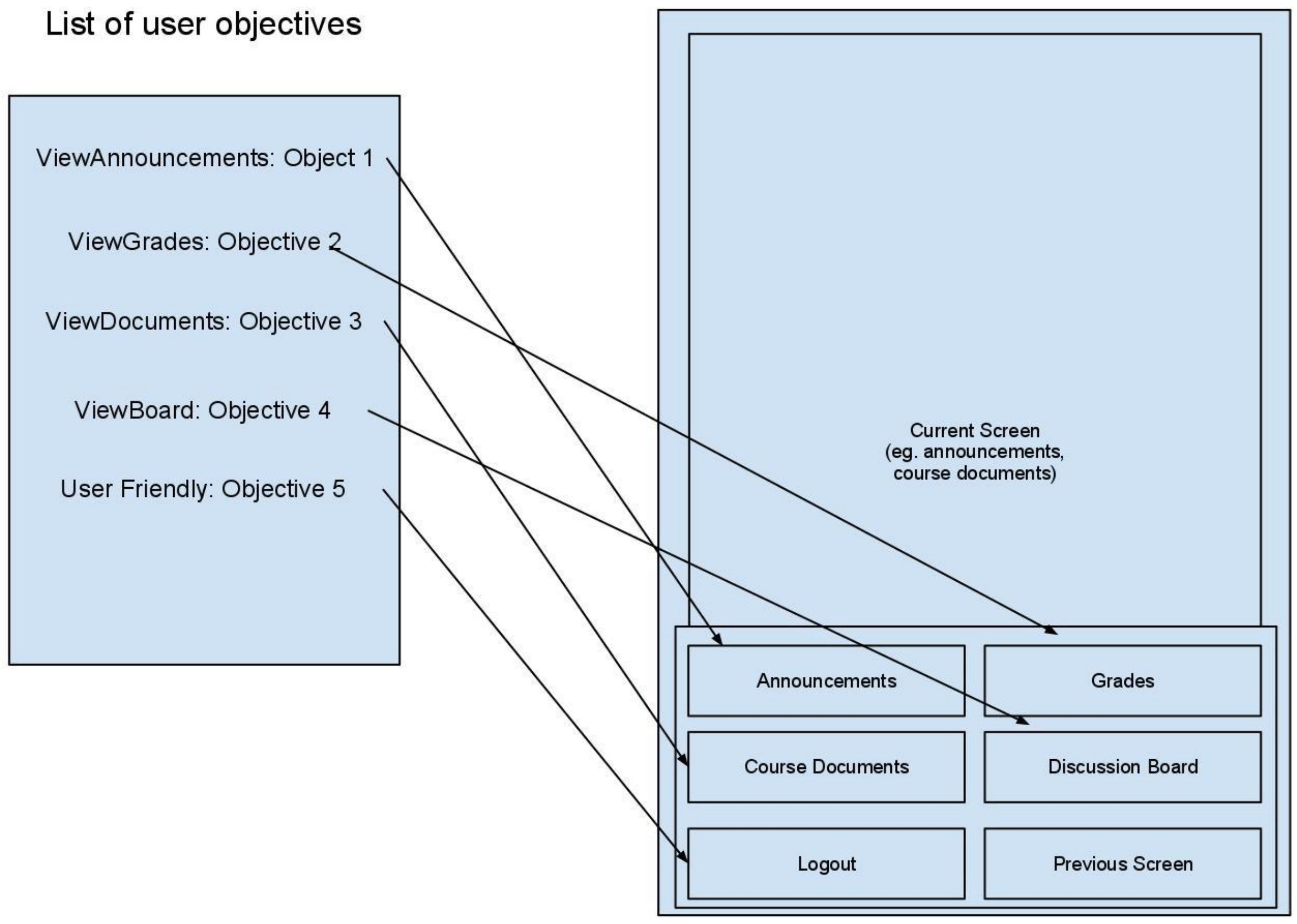
View Grades

Your grades for CPSC 362

Item	Grade	Class Average	Class Standing	Overall Grade
HW #1	B	B	9	B
HW #2	A+	B	7	A-
LAB #1	A	B	6	A
Test #1	B	C	5	B+

View Grades for Another Course

Mapping User Objectives Into Interface Actions



Design Patterns

Pattern Based Design

Design Problem- iboard includes all the classes that the user is currently taking, the user wants to avoid manually looking for the class in CourseList. It would be convenient for the user to somehow search for a specific class without "looking for it" his-self/her-self.

Candidate Patterns:

SchoolSearchEngine- The school has a search engine already, link this search engine to a search option and return relevant results to user.

FindSearchedText- User inputs texts in a "find box" and the system returns results with searched text in them.

CourseSearchBox- User inputs text (name of course) and the results are returned. The results will only have access to courses the user is taking/teaching.// thus results will also be limited to courses exclusive to user.

For iBoard the number of classes that are displayed in a users CourseList is strictly dependant on the number of classes the user is enrolled in, so SchoolSearchEngine will not be necessary since it will likely return an exaggerated amount of results. Similarly FindSearchedText is unnecessary since it most likely will yield irrelevant results. On the other hand CourseSearchBox suggests an efficient and seamless way to look for courses. It's Template is as follows.

CourseSearchBox

Problem: The users need to find a specific course from a list of courses.

Motivation: A scenario where the iBoard user needs to search for a specific course via a search bar, is applied across a collection of course objects.

Context: Rather than manually using the navigation to access a certain course, the user would like to search for a course straightforwardly.

Forces: The website has primary navigation. A user may like to search for an item specifically and have the search results return the desired course that was searched for. There is a limitation to the courses a user may search for, the user is limited to search for courses the he/she is enrolled in/teaching.

Solution: Offer search functionality that requires a keyword, a course filter a
course filter and a “search” button. Clicking on “search” or pressing the return key will activate the search. The course filter will filter the results to return only courses that are exclusive to user, thus eliminating any results that are undesirable to user. The search results are presented on a new page with a label containing “Search Results”.

Intent: This pattern will take a string of non-null characters, search through a list of courses, match the string of characters with courses in the list (of courses) and return results with relevant value.

Collaborations: This will be used along with Navigation and UserInterface patterns so that CourseSearchBar blends in the navigation and interface of the application.

Consequences: May make user less dependent on navigation and more dependent on search. If search is used more than navigation, time may be spent for usefully by making search more accurate. Inaccurate search results may arise due to user input.

Implementation: There should be the special consideration that user will not be able to find other classes other than those exclusive to him/her.

Known uses: The CourseSearchBar is used by many academic institution applications for it’s ease of use and convenience.

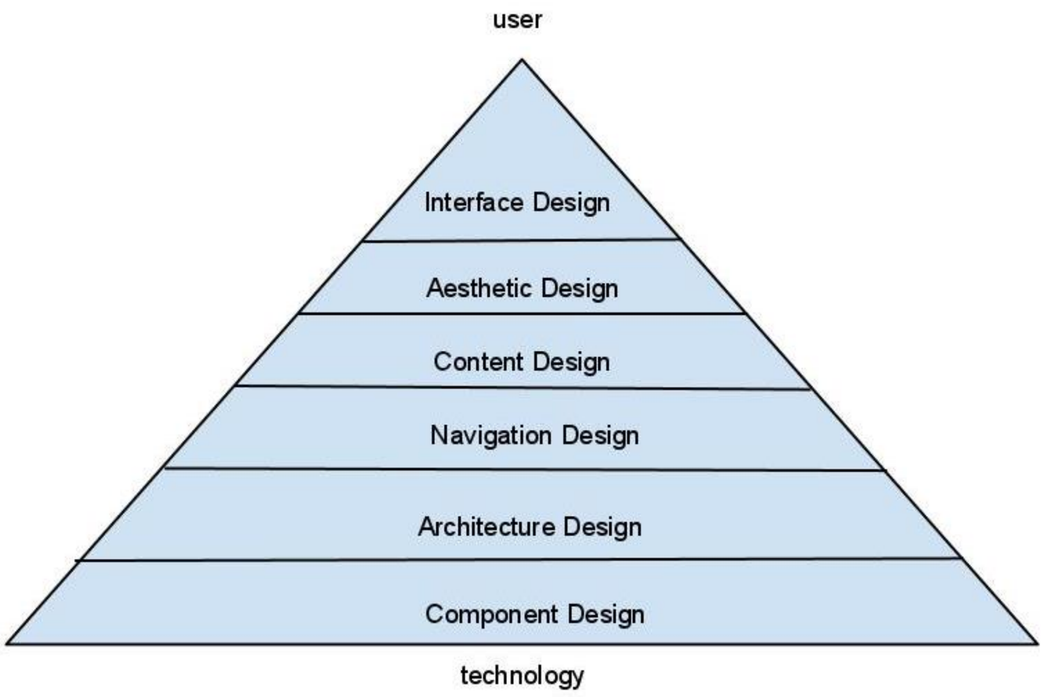
Related Patterns: SearchBar and other variations of “search bars” are highly related to CourseSearchBar.

Based off of this template iBoard can design the components required to implement the Course Search Bar.

2.16) Pattern-organizing Table

	Database	**Application**	**Implementation**	**Infrastructure**
Data/Content				
Architecture				
Component-Level				
Problem Statement	Search for items in web.	School Search Engine		
Problem Statement	Search for a specific text.	Find Searched Text		
Problem Statement	Search for academic courses.	Search Course Box		
User Interface				

A Design Pyramid for Mobile Apps



Interface Design: The interface was designed to be as simple and intuitive as possible.

Aesthetic Design: Although the app is aesthetically pleasing to the eye, this is a moblie app and there is a limited amount of screen real estate so very limited attention was paid to aesthetics compared to function and ease of use.

Content Design: This app is all about content, so it had to be quick and easy to find. This was done through large easy to see icons on the home screen for the all the content a user might want to access e.g. to view grades, click the Grades button etc...

Navigation Design: Again, this is a mobile app, so all desired information has to be accessible within two or three screens, with intuitive button names.

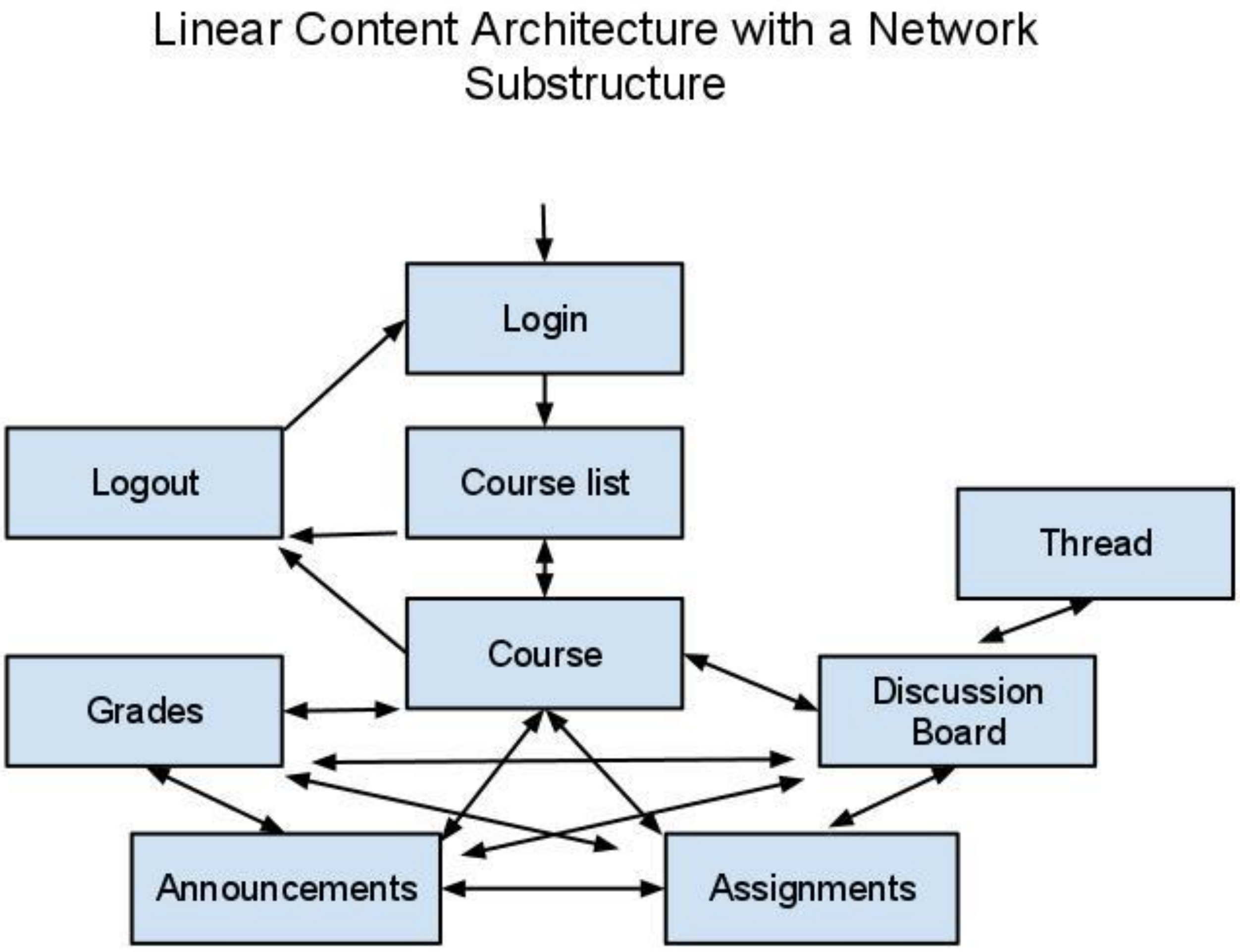
Architecture Design: We chose a combination of layered and data centered architectures. This provides us plenty of both security for confidential student data, and speed for quick access from many users concurrently.

Component Design: The component design was made through the style design process we evaluated several styles for each of our components and chose the ones that were most relevant to the iboard system and those which were compatible with our system architecture. Via the user interface you'll be able to see some of our components such as the search course bar.

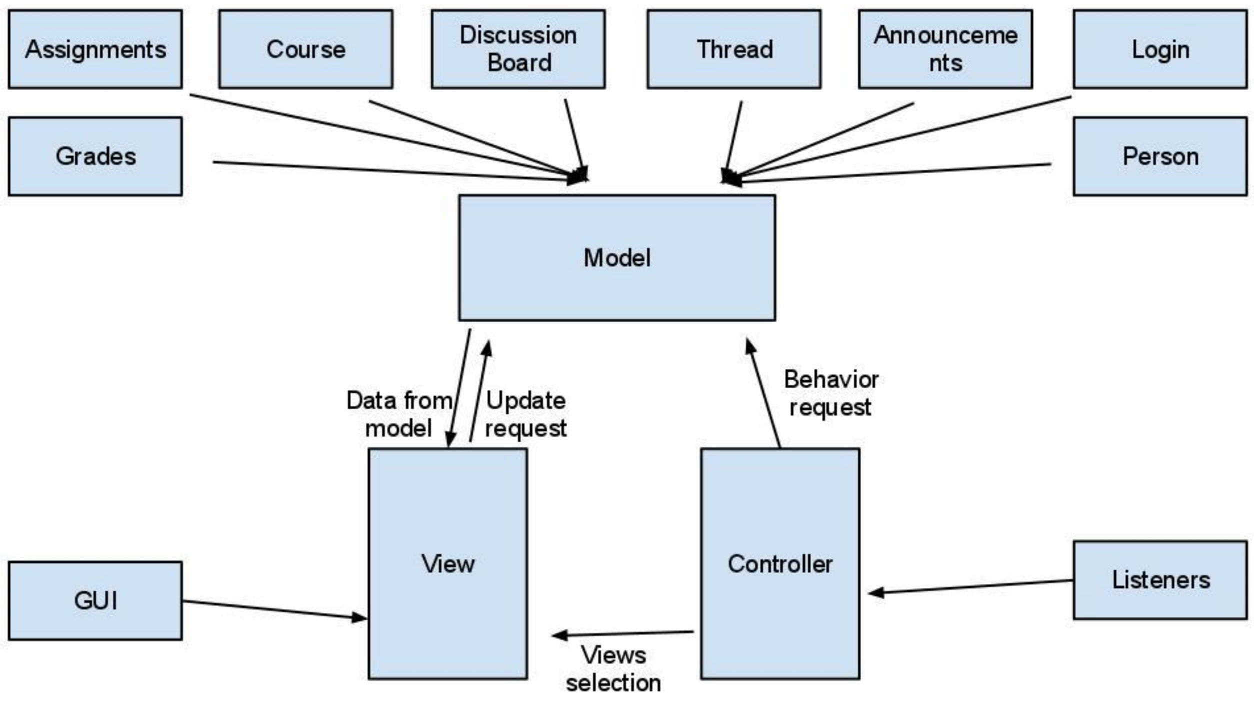
Content Architecture

I used a Linear Content Architecture, which connects to a Network structure.
The flow goes from "Login", to a "Course List" screen, at which point the user selects a "course". There is optional flow, where a user may "Logout".

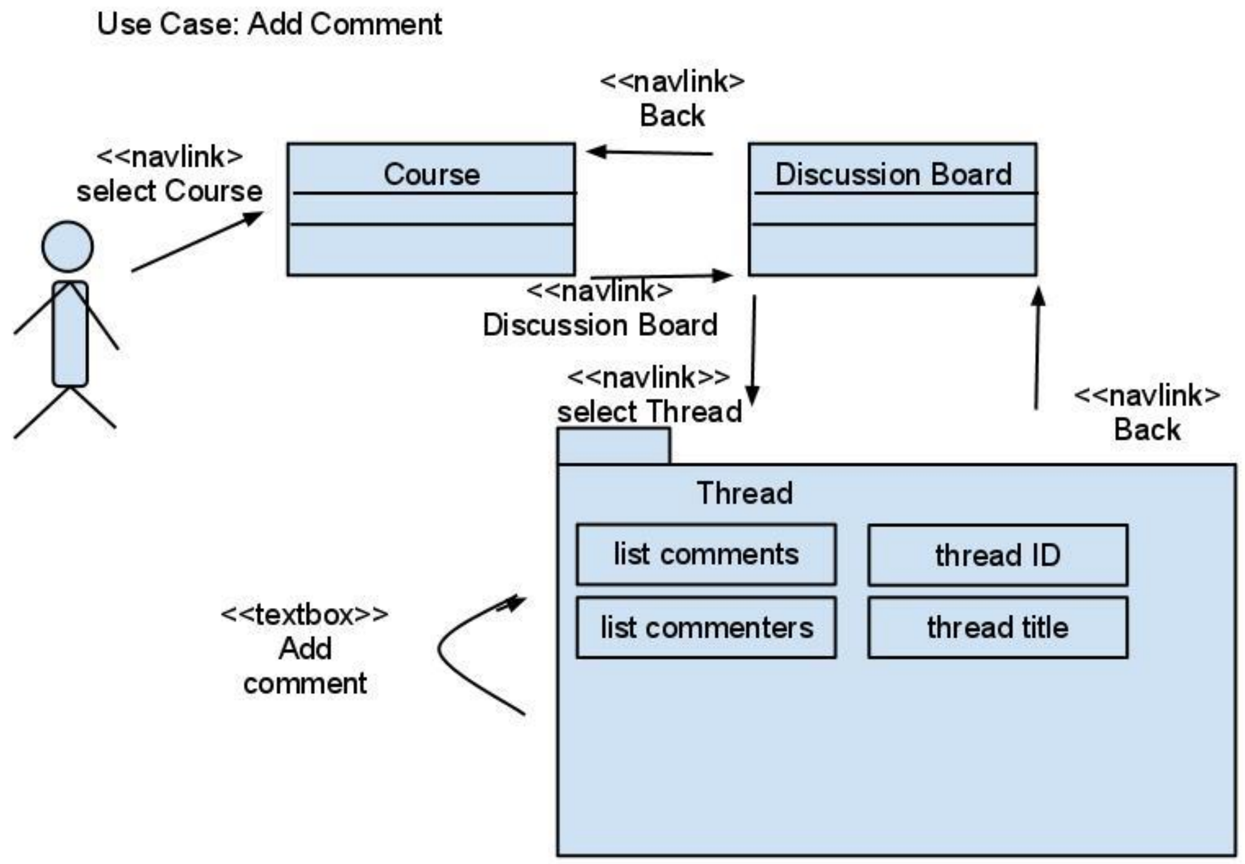
The course is Networked. User will begin at a course screen, where they can select any tab - "Grades", "Announcements", "Assignments" etc. If they view "Announcements", they can choose the tab for "Grades" to quickly view grades.



2.19) MVC Architecture



Navigation Semantic Unit



Criticizing Nonfunctional Requirements

Were the functional non-requirements necessary?

After a long and careful review I have learned that for the most part the non-functional requirements are indeed needed to deliver a ***quality*** iBoard application. Security is an essential part of iBoard there should be limitations to the access a user has, consider basic login requirements a user should have a user name and should know his/her password in order to access information that is considered private. The rest of the non-functional requirements are also essential to the success of the design. Some things that should be considered however are the priority of the non-functional requirements, if the hierarchy of the non-functional requirements drive resource allocation then security and private data should be the top 2 priorities. I can then shift my attention and resources to browser compatibility, modularity, ease of use and concurrency, respectively. Although it is essential to have concurrent users on our application, this particular non-functional will be more of a network issue than design/implementation.

Prototype Coding of Classes and Methods

```
class Login
{
        public DisplayLoginScreen(){} //contains String user, int pass
        public VerifyLoginInformation(String user, int pass){}
}

class Person
{
        private int uniqueID;
        private CourseList myCourseList;
        private String userName;
        private int password;
        public bool VerifyLogin(String User,int Pass){}
        public int getUniqueID(){}
}

class CourseList
{
        list <Course> CourseList;
        public DisplayCourseList(){}
        public GotoCourse(courseID){} //course ID from DisplayCourseList
}

class Course
{
        list <int> StudentIDs;
        private int TeacherID;
        private Grades myGrades;
        private Announcement

        public DisplayCourseScreen(){}
        public GotoGrades(){}
        public GotoAnnouncements(){}
        public GotoCourse(){}
        public GotoDocuments(){}
}

class Grades
{
        private int **GradeTable;
        public DisplayMyGrade(int ID){}
        public DisplayAllGrades(int ID){} //only teacher can view all
        public UpdateGrades(int ID){}; //uploads grades from predefined “Grades.xml”.
}
```

```
class Announcement
{
	private char ** announcement;
	public DisplayAnnouncements(){}
	public UploadComment(String comment){} //called from display announcemnt
}

class Documents
{
	private list<String> DownloadLinks;
	private list<String> DownloadNames; //these two combined make a document

	public DisplayDocuments(){}
	public DownloadDocument(int DownloadID){}
	public UploadDocument(String filename,String displayname){}
}

class Discussion
{
	private list<Thread> myThreads;
	public void DisplayDiscussion(){}
	public void GotoThread(Thread aThread){}
	public Thread CreateThread(String name){}
}

class Thread
{
	private list<String> comments;
	private list<int> IDs;
	private int ThreadID;
	private String ThreadTitle;
	public void DisplayThread(){}
	public void CreateComment(){}
	public void EditComment(int commentIndex){}
	public void DeleteComment(int commentIndex){}
}
```

References

1. Pressman, Roger. "Software Engineering: A Practitioner's Approach ."
   New York: McGraw-Hill Science, 2009. Print.

2. Neil B. Harrison and Paris Avgeriou (2011) *Pattern-Based Architecture Reviews.*
   IEEE Software, 2011.

3. Beyer, Dirk, Arindam Chakrabarti, and Thomas Hezinger. "Web Services Interfaces."
   Sosy-lab (2005): 12. Web.
   Aug 2011.<http://www.sosy-lab.org/~dbeyer/Publications/2005-WWW.Web_Service_Interfaces.pdf>.

4. Stotts, David. "State Transition Diagrams."
   http://www.cs.unc.edu/~stotts. University of North Carolina at Chapel Hill, 24/09/2008.
   Web. 28 Jul 2011. <http://www.cs.unc.edu/~stotts/145/CRC/state.html>.