EZComm LMS  
Software Design Document (SDD)

**Team members:**

Ribe Ninan

Hongquan Yu

Yue Wang

Zeqian Huang

Guangchu Shi

Xiangfeng Wang

[1.Product Overview 4](#_Toc500533559)

[2. Software Architecture 4](#_Toc500533560)

[2.1 Architecture Drivers 4](#_Toc500533561)

[2.1.1 Functional Requirement 4](#_Toc500533562)

[2.1.2 Quality Attribute/Non-functional Requirement 5](#_Toc500533563)

[2.2 Architecture Key Decisions and Rationale(style updated ) 11](#_Toc500533564)

[Architecture Logical View(Hongquan, Zeqian, Ribe,Yue Wang) 13](#_Toc500533565)

[2.4 Architecture Behavior (Yue Wang,Zeqian Huang) 15](#_Toc500533566)

[2.5 Architecture Deployment View 19](#_Toc500533567)

[2.6 Information Design Model 20](#_Toc500533568)

[2.6.1 conceptual data models 20](#_Toc500533569)

[2.6.2 logical data models 21](#_Toc500533570)

[2.6.3 physical data models 22](#_Toc500533571)

[2.8 Requirements Allocation 23](#_Toc500533572)

[2.8.1 Functional requirement 23](#_Toc500533573)

[2.9 Architecture Work Allocation 25](#_Toc500533574)

[3. Software Detail Design 25](#_Toc500533575)

[3.1 Design principles (Yue Wang) 25](#_Toc500533576)

[3.1.1. Modularity----The liskov substitution(LSP) principle 26](#_Toc500533577)

[3.1.1.2. Information Hiding: 27](#_Toc500533578)

[3.2 Common key decision and design patterns in all components(Hongquan Yu) 30](#_Toc500533579)

[3.3 Components Detailed Design Analysis 31](#_Toc500533580)

[3. 3.1: File Management Service (Ribe) 31](#_Toc500533581)

[3.3.1.1 Detailed Design Key Decisions and Rationale 31](#_Toc500533582)

[3.3.1.2 Detailed Design Structure 32](#_Toc500533583)

[3.3.1.3 Detailed Design Behavior 33](#_Toc500533584)

[3.3.1.4 Data Structures 35](#_Toc500533585)

[3.3.1.5 Algorithms 35](#_Toc500533586)

[3.3.1.6 Architecture to Detailed Design Tracing 35](#_Toc500533587)

[3.3.2 Assignment Management (Updated by Hongquan Yu) 35](#_Toc500533588)

[3.3.2.1 Detailed Design Key Decisions and Rationale 35](#_Toc500533589)

[3.3.2.2 Detailed Design Structure <Updated by Yue Wang> 36](#_Toc500533590)

[3.3.2.3 Detailed Design Behavior 36](#_Toc500533591)

[3.3.2.4 Data Structures 39](#_Toc500533592)

[3.3.2.5 Algorithms 39](#_Toc500533593)

[3.3.2.6 Architecture to Detailed Design Tracing 39](#_Toc500533594)

[3.3.3 User account (Updated by Hongquan Yu) 40](#_Toc500533595)

[3.3.3.1 Detailed Design Key Decisions and Rationale 40](#_Toc500533596)

[3.3.3.2 Detailed Design Structure<Updated by Yue Wang> 40](#_Toc500533597)

[3.3.3.3 Detailed Design Behavior 40](#_Toc500533598)

[3.3.3.4 Data Structures 41](#_Toc500533599)

[3.3.3.5 Algorithms 41](#_Toc500533600)

[3.3.6 Architecture to Detailed Design Tracing 41](#_Toc500533601)

[3.3.4 Course Management (Updated by Hongquan Yu) 42](#_Toc500533602)

[3.3.4.2 Detailed Design Structure <Updated by Yue Wang> 43](#_Toc500533603)

[3.3.4.3 Detailed Design Behavior 43](#_Toc500533604)

[3.3.4.4 Data Structures 44](#_Toc500533605)

[3.3.4.5 Algorithms 44](#_Toc500533606)

[3.3.4.6 Architecture to Detailed Design Tracing 45](#_Toc500533607)

[3.3.5 User Interface <Yue Wang> 45](#_Toc500533608)

[3.3.5.1 Detailed Design Key Decisions and Rationale 46](#_Toc500533609)

[3.3.5.2 Detailed Design Structure 47](#_Toc500533610)

[3.3.5.3 User Interface Prototype 48](#_Toc500533611)

[3.3.5.4 Coding: 48](#_Toc500533612)

[3.3.5.6 Algorithms 49](#_Toc500533613)

[3.5.7 Architecture to Detailed Design Tracing 49](#_Toc500533614)

## 1.Product Overview

This Software Architecture Document provides the architectural overview of the Learning Management System, EZComm. This tool is to be used by educational institutions to help facilitate communication, and provide educational courses, and its materials to students. The two primary user roles in this LMS are instructors and students. The architectural design for this Learning Management System was developed to provide the instructors the capabilities such as posting files, grades and other features for students. Similarly, for students, this tool provides the capability to view these files and grades.

## 2. Software Architecture

### 2.1 Architecture Drivers

Although software architecture drivers are diverse which could affect the software architecture, such as requirement, constraints, best practices and organizational, the most important factors could be the functional requirement, quality attribute, and constraints.

For constraints for EZComm project, the available time, an academic semester, for EZComm is not very much. Some group members have full-time job during the day, thus available time for project is limited. As for other members, many are taking multiple courses during the semester, thereby limiting the amount of time they have to spend on the project. Thus time constraint is an import factor that affected the software architecture decisions.

In regards to technical constraints, most members in our team are inexperienced in software development, and do not have knowledge in industry software development. Many technologies used to design the software are new for the team members, therefore technical constraint is also a factor that affected the software architecture decisions.

#### 2.1.1 Functional Requirement

The most important factor affecting the architecture design is the functional requirement, which are the tasks/features that we need to implement for this LMS system. The functional requirements for EZComm are stated as following:

For instructor:

1. instructor shall be able to see the list of all students enrolled in the classes she/he is teaching
2. instructor shall be able to manage (create, edit, delete, and publish/unpublish) course syllabus
3. instructor shall be able to manage (create, edit, delete, and publish/unpublish) course assignments
4. instructor shall be able to see student’s submissions
5. instructor shall be able to grade student’s submissions

For student:

1. student shall able to view syllabus
2. student shall able to view assignments
3. student shall able to submit assignments
4. student shall able to see her/his own submissions
5. student shall able to view her/his own grades

For system:

1. system shall able to automatically creates gradebook for each course
2. system shall able to maintains a grade book for each course

#### 2.1.2 Quality Attribute/Non-functional Requirement

Another architecture driver is quality attribute, we represent it as the utility tree:

Usability：

* Operability.
* Learnability.
* User error protection.
* User interface aesthetics.

|  |  |
| --- | --- |
| Quality Attribute | Usability |
| Scenario | Operability |
| Source | End users |
| Stimulus | Users want to use the system efficiently |
| Artifact | System |
| Environment | At runtime |
| Response | * Reuses already entered data and/or commands * Supports efficient navigation within a screen * Provides a comprehensive searching within the website * Allows multiple simultaneous activities * Anticipate the functions user may need |
| Response Measure | * Less than 5 steps to realize a certain goal (M, M) * 80% of the users should be satisfied about the system’s efficiency (H, H) |

|  |  |
| --- | --- |
| Quality Attribute | Usability |
| Scenario | Learnability |
| Source | Students and instructors |
| Stimulus | Both students and instructors want to adapt to the system and learn system features without difficulties |
| Artifact | System |
| Environment | At runtime |
| Response | * Help topics are rich and sensitive to context * The user interface is similar to other online learning systems that users used before * In an unfamiliar context, the user interface is still easy to use |
| Response Measure | * Time users spend on learning system features is less than 15 minutes (M, M) * 80% of the new users are satisfied about transferring to the new system (M, H) |

|  |  |
| --- | --- |
| Quality Attribute | Usability |
| Scenario | User error protection |
| Source | End users |
| Stimulus | Users want to minimize the impact of their errors |
| Artifact | System |
| Environment | At runtime |
| Response | * Be able to retrieve forgotten usernames and passwords * Be able to undo, cancel, or recover from system failures * Be able to roll back to the initial stage if unsuccessful transactions occur * Be able to recognize and correct user errors * Record the error and report it to system |
| Response Measure | * Less than 5% errors occur during the whole process (H, H) * Password should be retrieved every time if it’s forgotten by user(H,L) |

|  |  |
| --- | --- |
| Quality Attribute | Usability |
| Scenario | User interface aesthetics |
| Source | End users |
| Stimulus | Users want to use a beautifully designed website and feel comfortable navigating through it |
| Artifact | System |
| Environment | At runtime |
| Response | * Layout of the website should be concise and intuitive * The design should be consistent across all subsystems and sections * The system should enable users to work at their own pace, without distractions |
| Response Measure | * 90% of the users feel comfortable and satisfied when operating the system (H, H) |

Modifiability (M, L)：

|  |  |
| --- | --- |
| Quality Attribute | Modifiability |
| Scenario | Some attributes of the system are to be modified/added/deleted |
| Source | Developers and system admins |
| Stimulus | Developers and admins wish to add/delete/modify/vary system functionalities and quality attributes |
| Artifact | System user interface; platform; environment |
| Environment | At design time; compile time; at build time; at run time |
| Response | * System should be able to locate the places that need to be modified * System should be able to allow developers to make the necessary changes, without affecting other functionalities * System should be able to test the modification * System should be able to deploy the modifications |
| Response Measure | * Cost of the modifications, in terms of money and effort, should be less than 10% of the total cost (H, M) * Zero of other attributes may be affected (H, H) |

Performance (H, H)：

|  |  |
| --- | --- |
| Quality Attribute | Performance |
| Scenario | A number of requests asking for a service arrive at the system |
| Source | Web requests |
| Stimulus | Web requests arrive at the system and ask for the response of a certain service |
| Artifact | System |
| Environment | Normal mode; overload mode; |
| Response | * System handles the requests without errors * System be able to protect itself against overwhelming requests |
| Response Measure | * Under normal amount of requests the data loss rate should be less than 1% (H, M) * Under overload mode the data loss should be less than 10% (M, H) * Latency under normal mode should be less than 75ms (H, M) * Latency under overload mode should be less than 150ms (M, H) |

Availability：

* Fault tolerance
* Recoverability
* Excess Data storage

|  |  |
| --- | --- |
| Quality Attribute | Availability |
| Scenario | Fault tolerance |
| Source | Internal to the system |
| Stimulus | Some process fault in the hardware or software |
| Artifact | System’s processor |
| Environment | Degraded mode |
| Response | * Report it； * Notify user/system admin/developer; * Some sources are temporarily unavailable; * Continue degraded mode until cause of the fault is found and fixed |
| Response Measure | * 60% part of the system can be used;(H,L) * Notify user/operator the fault within 3 minutes;(M,M) * The software repaired time will be in 2 days.(H,H) |

|  |  |
| --- | --- |
| Quality Attribute | Availability |
| Scenario | Recoverability |
| Source | External to the system or internal to the system |
| Stimulus | System crashed |
| Artifact | System‘s processor |
| Environment | Degraded mode |
| Response | * Report it; * Notify it to user/system admin/developer; * Developer check the system and fix the problem; * The data will be recovered; * Email user that system can be used. |
| Response Measure | * The time of checking and fixing the bug will be less than 2 days;(H,H) * 99.99% data should not be lost except the data which was editing when the system crashed.(H,H) |

|  |  |
| --- | --- |
| Quality Attribute | Availability |
| Scenario | Excess Data storage |
| Source | Internal to the system |
| Stimulus | User wants to upload the files bigger than the data size limitation |
| Artifact | Persistent storage |
| Environment | Normal operation |
| Response | * Notify user the size limitation of the uploaded files * Continue to operate when use upload files under the limitation. |
| Response Measure | Check the file size and notify user within 60 seconds;(M, L) |

Security (H,M)

|  |  |
| --- | --- |
| Quality Attribute | Security |
| Scenario | Student want to modify their grade. |
| Source | Hacker from inside the organization |
| Stimulus | Students attempt to modify grade data |
| Artifact | Grade data within system |
| Environment | Normal mode |
| Response | * Authenticates user； * blocks access to data and/or services； * records access/modifications or attempts to access/modify data by identity； * informs instructor |
| Response Measure | * The authenticate email will send to instructor after 5 times invalid login;(H,L) * Block access the data within 15 minutes.(H,L) * Block access for at least one hour.(H, L) * Data integrity is unaffected 98% of the time(H,M) |

Testability (H,L)

|  |  |
| --- | --- |
| Quality Attribute | Testability |
| Scenario | System test before delivered |
| Source | System developer and end user |
| Stimulus | Execution of tests due to completion of code increment |
| Artifact | Portion of the system; Full system |
| Environment | Compile time; Integration time; Deployment time; Run time |
| Response | * Controlled to perform the desired tests * Record the response to each test * Capture cause of fault, record it and fix the bug * Monitor the every state of the system |
| Response Measure | * 95% of executable statement executed;(H,L) * 3% of executable statement failed;(H,M) * Less than 2 day was needed to fix the bugs(H,H) |

### 2.2 Architecture Key Decisions and Rationale(style updated )

|  |  |  |
| --- | --- | --- |
| concerns |  | Technology to use in backend to build EZComm? |
| Alternatives |  | 1. Java  2. PHP  3. C#  4. Python |
| Ranking criteria |  | 1. Feasible to build the service in time;  2. Minimum time for team members to master the technologies;  3. It is free and open source; |
| Architecture decision | description | Almost all of our team members are familiar with Java, We do not have to spend too much time to learn new technology to implement the backend. Java is enough to develop all the required features. |
| Rationale | We will use Java to develop the backend structure and to interact with frontend.  We will use JDBC to connect the MySQL database. |

|  |  |  |
| --- | --- | --- |
| concerns |  | Database to use in backend to build EZComm? |
| Alternatives |  | 1. MySQL  2. SQL Server  3. Oracle  4. MongoDB |
| Ranking criteria |  | 1. It is free  2. Easy to connect to backend |
| Architecture decision | description | MySQL is free for use and portable to Java backend. Also it’s easy to learn for everybody in the group. |
| Rationale | MySQL with mature MySql workbench GUI is suitable for our project. |

|  |  |  |
| --- | --- | --- |
| concerns |  | Technology to use in front-end to build EZComm? |
| Alternatives |  | 1. Bootstrap  2. Foundation  3. Semantic UI |
| Ranking criteria |  | 1. Easy to get solution  2. Dynamic  3. Bootstrap framework offer a great start point for our project and its great responsive design feature. |
| Architecture decision | description | Bootstrap are responsive web design support (can also be disabled if required), it also has extensive documentation |
| Rationale | We will use Bootstrap to build our frontend web service interacting with backend |

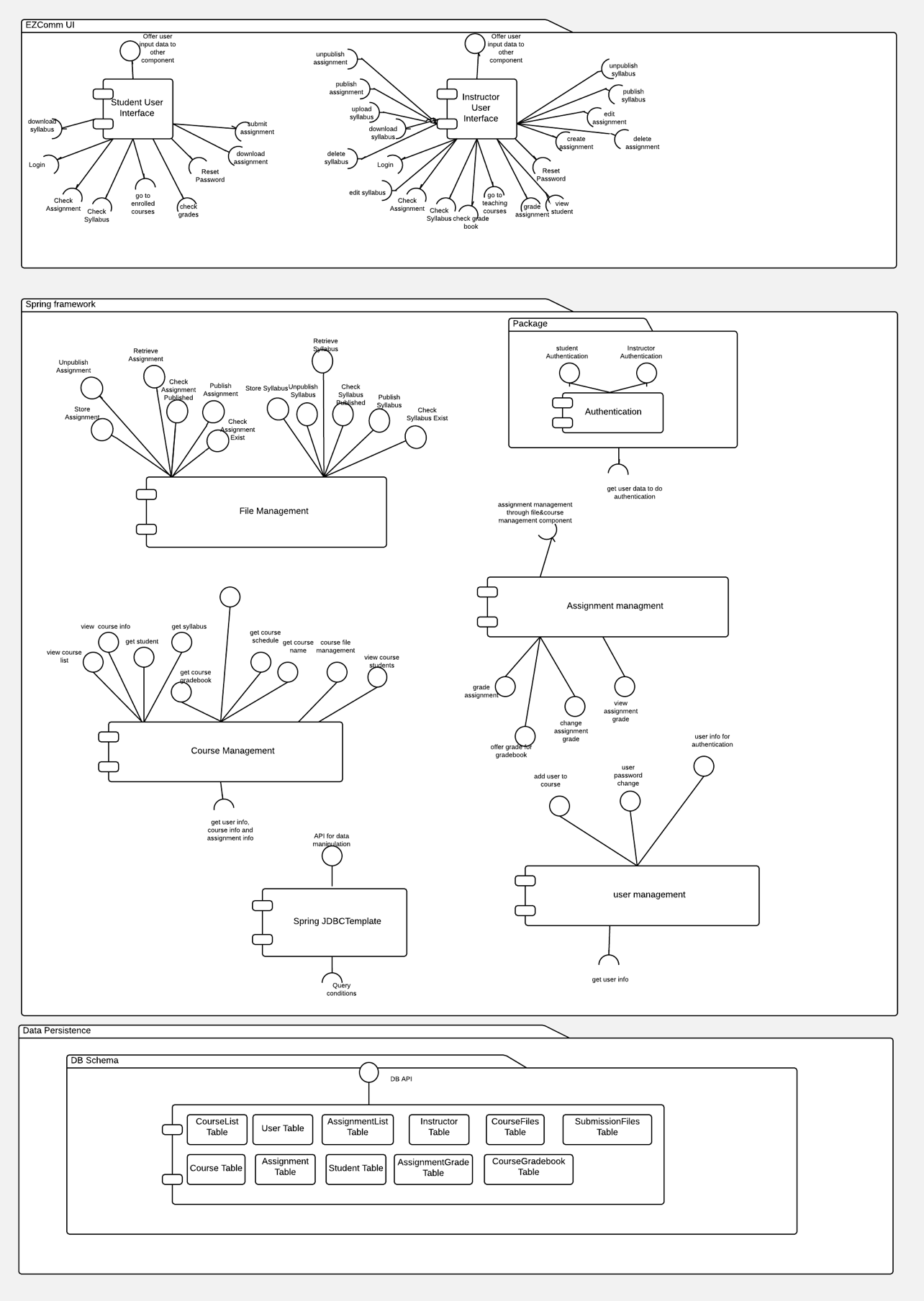
|  |  |  |
| --- | --- | --- |
| concerns |  | Where to deploy EZComm? |
| Alternatives |  | 1. local Server  2.AWS  3. Google cloud |
| Ranking criteria |  | 1. Free and easy to use  2. Accessible from anywhere  3. Every team member can have access for same schema so helpful to development. |
| Architecture decision | description | AWS is free for light use in EC2. it is also the largest cloud provider in this industry. |
| Rationale | We will use AWS to host the web service and database to make it accessible from anywhere |

|  |  |  |
| --- | --- | --- |
| concerns |  | Design pattern for EZComm |
| Alternatives |  | 1.MVC  2.Pipe-filter |
| Ranking criteria |  | 1. Maintainability  2. Reusability  3. Usability |
| Architecture decision | description | MVC has following features:  Easily Maintainable  High Re-usability  Readable because of abstractions |
| Rationale | we use MVC to divide EZComm into three interconnected parts |

## 

## Architecture Logical View(Hongquan, Zeqian, Ribe,Yue Wang)

**Package diagram:**



|  |  |
| --- | --- |
| Components | Description |
| User Interface | Manage all the user interfaces and the interaction with different modules. |
| User Account | Manage all the user account information including changing password information and users personal information : Email address, Job status; |
| Course Management | Manage all the course related tasks, such as course syllabus service, assignment management. As you can see, course component needs support of file management component and assignment management component to complete the relative features. It includes syllabus management, assignment management and gradebook management which offer gradebook for a specific course. |
| File Management | ur project, syllabus and assignment related features are completed rely on the file management. So this file management provides services for another two components, course management and assignment management.  Instructor upload the syllabus and create/ edit/ publish and unpublish the syllabus; Student make the submission and download the files(including download syllabus and assignment). |
| Assignment Management | For student:  student view the syllabus, assignment, make the submission to the assignment, review their submission, download the submission. view the grading information and check the grades for all the assignment inside this course;  For instructor:  Instructor view the assignment and choose to delete /create/ update or publish the assignment.  Instructor grades the assignment. |

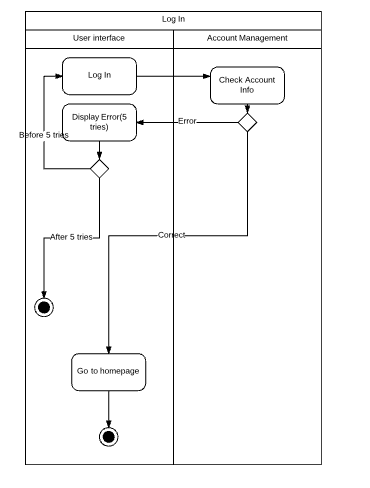
### 

### 

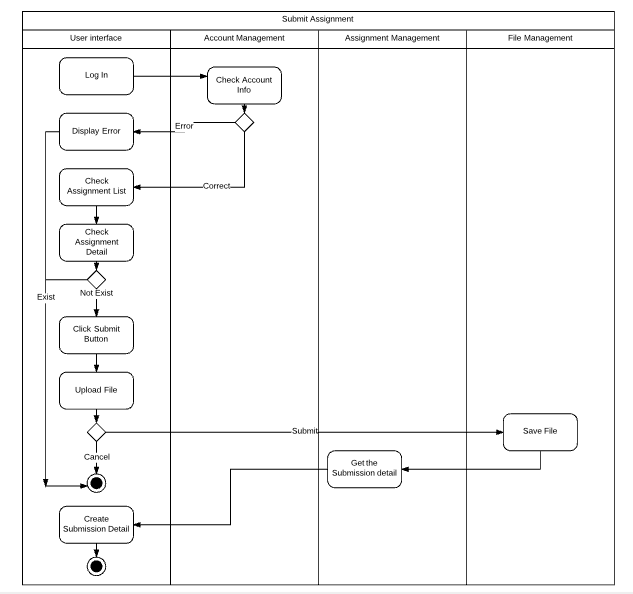
### 2.4 Architecture Behavior (Yue Wang,Zeqian Huang)

**Scenario 1:** High quality scenario - **security**

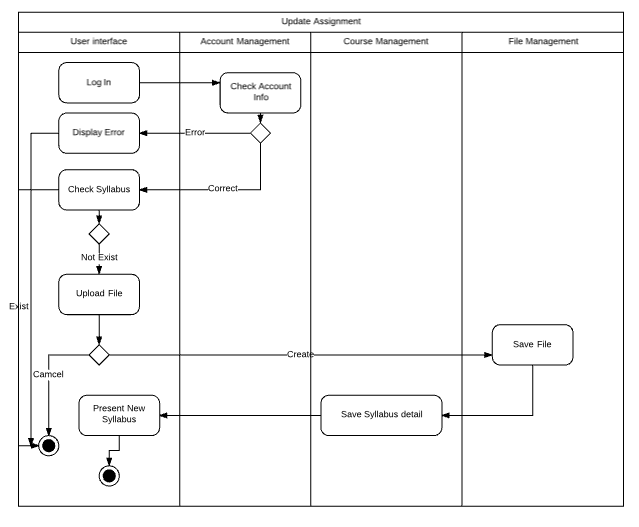
1. user enter username and password
2. check the username and password is match or not
3. if yes, user can enter website. Otherwise, display “invalid username and password” ask user to enter again.
4. if usr enter 5 times incorrect, the account will be blocked and the system will send the email to inform user and admin.

****

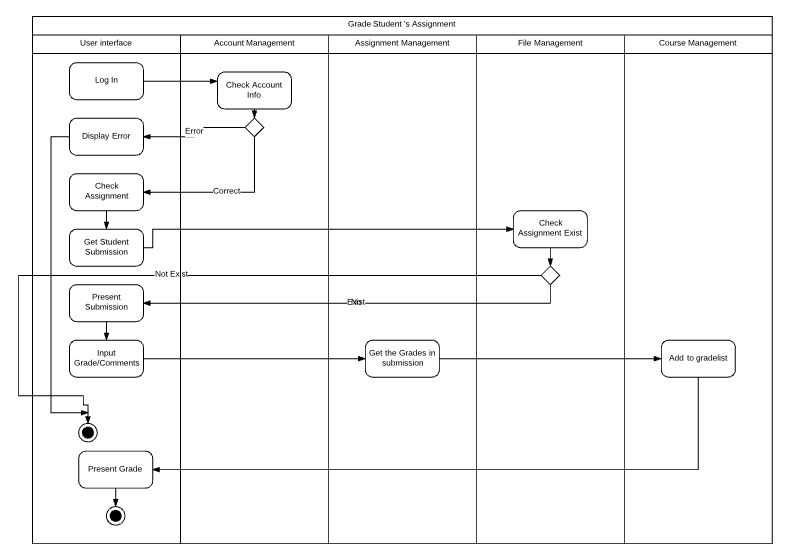
**Scenario 2:** Student Submit assignment



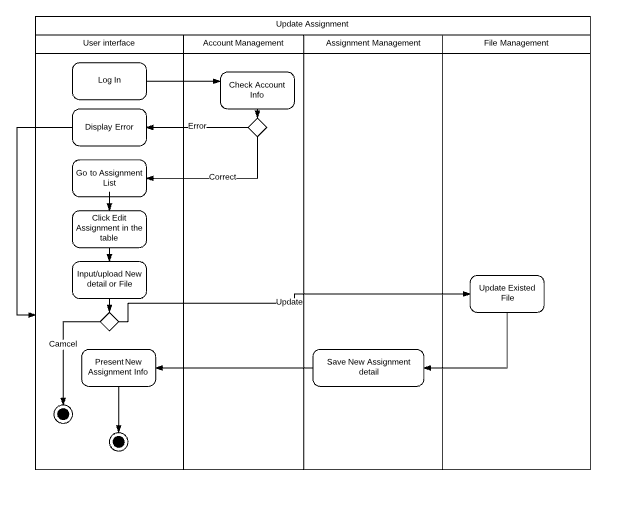
**Scenario 3:** Instructor publish syllabus



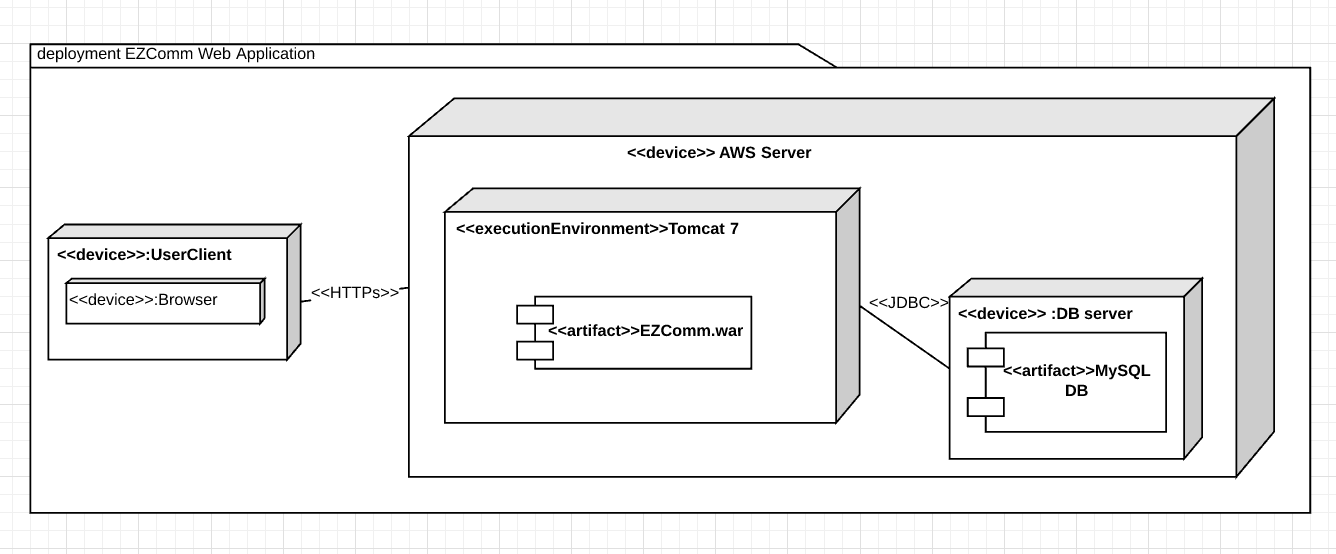
**Scenario 4:** Instructor grade the assignment wrote by student



Scenario 5 : Instructor Edit Assignment

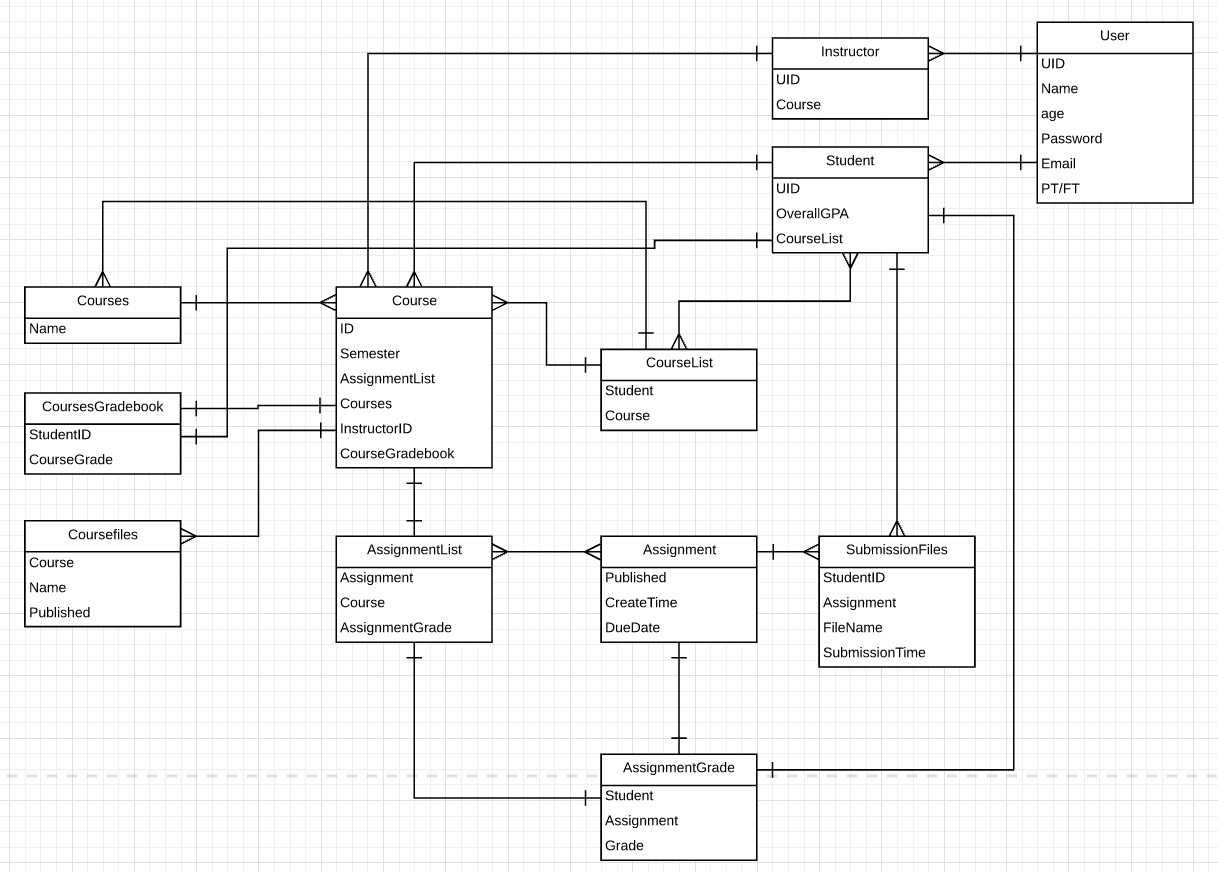


## 2.5 Architecture Deployment View

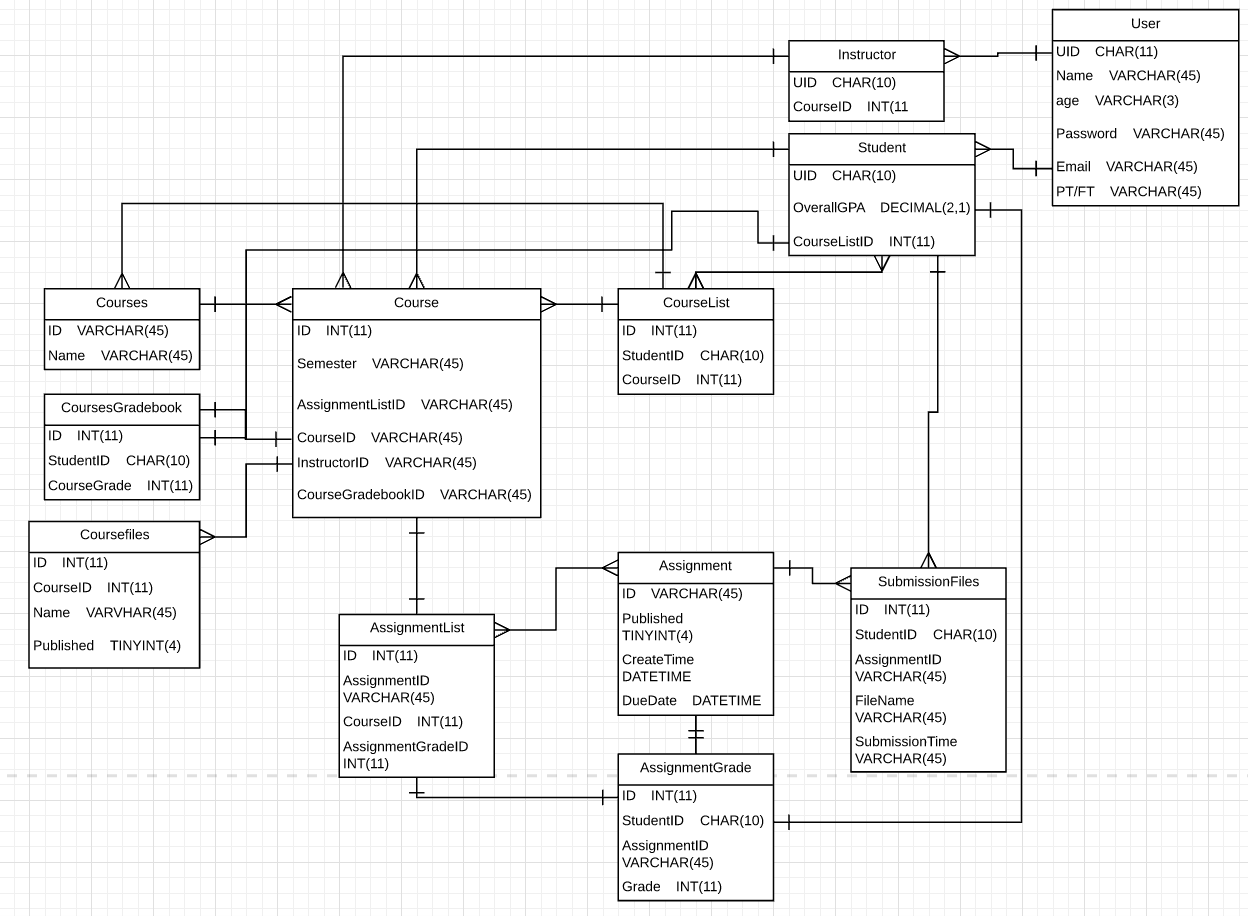
**

### 2.6 Information Design Model

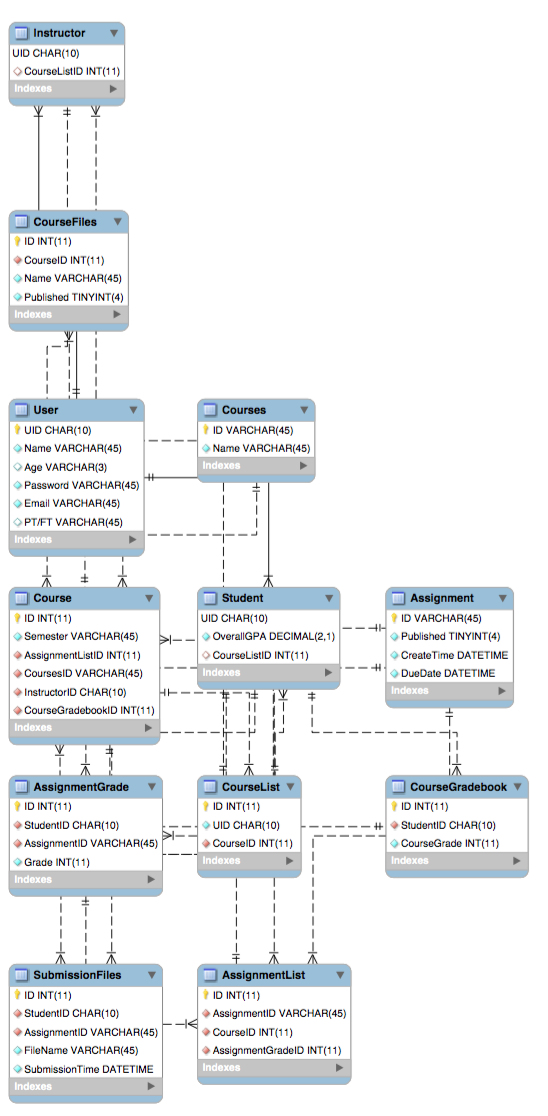
#### 2.6.1 conceptual data models



#### 2.6.2 logical data models



#### 2.6.3 physical data models



### 2.8 Requirements Allocation

#### 2.8.1 Functional requirement

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Use Cases/ Architecture Components | Component 1 File Management | Component 2  Assignment Management | Component 3 User Accounts | Component 4 Course Management | Component 5 User Interface |
| Students view syllabus | X |  | X |  | X |
| Students view assignment |  | X | X |  | X |
| Student submit assignment and view submission |  | X | X |  | X |
| View her/his own grades |  | X | X | X | X |
| Instructor see enrolled student list |  |  | X | X | X |
| Instructor create/delete/publish/unpublish syllabus | X |  | X |  | X |
| Instructor create/delete/publish/unpublish assignment |  | X | X |  | X |
| Instructor see and grade student assignment |  | X | X |  | X |
| Student view grade book for Certain course |  |  | X | X | X |
| Instructor view grades for all student in his class |  |  | X | X | X |

**2.8.2 Abuse Use Case**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Abuse Use Cases/ Architecture Components | Component 1 File Management | Component 2  Assignment management | Component 3 User Accounts | Component 4 Course Management | Component 5 User Interface |
| Students delete syllabus | X |  | X |  |  |
| Students Change course schedule | X |  | X |  |  |
| Student upload malicious documents | X |  |  |  |  |
| Student modify grades |  | X | X | X |  |
| Plagiarize from other students | X |  | X |  |  |
| Instructor modify/delete student assignment |  | X |  |  |  |
| instructor modify student grade arbitrary |  | X |  | X |  |

## 

## 2.9 Architecture Work Allocation

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Architecture component name/ Team member** | **Ribe Ninan** | **Zeqian Huang** | **Xiangfeng Wang** | **Hongquan Yu** | **Yue Wang** | **Guangchu Shi** |
| **Course Management** | Development | Testing | Testing | Development | Development | Testing |
| **User Accounts** | Testing | Development | Development | Testing | Testing | Development |
| **File Management** | Development | Development | Testing | Testing | Testing | Development |
| **Assignment Management** | Testing | Testing | Development | Development | Testing | Testing |
| **User Interface** | Testing | Development | Testing | Testing | Development | Testing |

## 3. Software Detail Design

In this section, the primary design pattern we use is broker pattern. because it serves as a mediator between the instructor/student and the course management system.

### 3.1 Design principles (Yue Wang)

This part we not only analyze the basic principles for the development of the software but also take the OO SOLID design principle into consideration to make it robust to the future maintenance and reusability.

Followed Design principle:

1. Modularity
2. Information Hiding
3. Implementability (reusability)
4. The liskov substitution(LSP) principle
5. The open closed principle(OCP)
6. Single Responsibility Principle(SRP)
7. Interface SegRegation Principle(ISP)
8. The dependency inversion principle(DIP)

### 3.1.1. Modularity----The liskov substitution(LSP) principle

**Low Coupling**

Spring MVC convention conforms with this principle.

**3.1.1.1.Clear distinction between interface and implementation(key design principle)**

(1)by categorizing the service and service implementation in two different packages. as can be found in class diagram in every module;

implementation:

src.main.java.com.umd.ezcomm.model.dao.service.impl

Services:

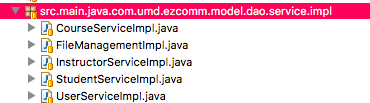
src.main.java.com.umd.ezcomm.model.dao.service

(2) by defining different services for different objects:



here we also followed the Interface SegRegation Principle(ISP) by giving instrcutor and student different interfaces (services)

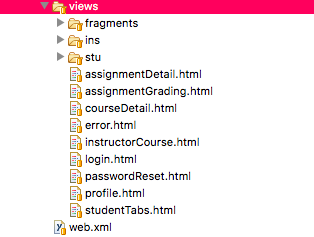
(3) by creating the corresponding services implementation for different objects:



(4)by creating all the data inside the DAO’s domain:



(5)by categorizing the html inside different users:



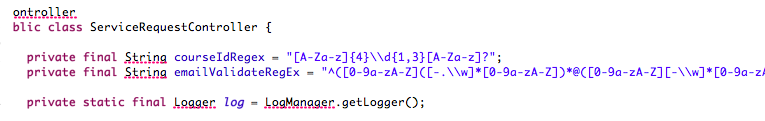
### 3.1.1.2. Information Hiding:

1.Use scope and visibility markers to restrict visibility

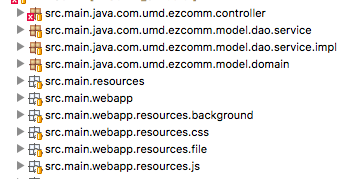
we use Object Oriented visibility to achieve this goal by defining different object and methods with different visibility:

(1). Private—Visible only within the class where it is defined

all the local variable are defined within the controller’s method:

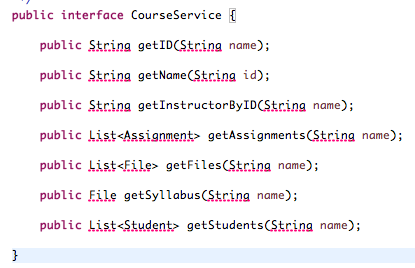


(2)Package—Visible in the class where it is defined as well as classes in the same package or namespace



(3)Public—Visible anywhere the class is visible

all interfaces and the services under the interfaces are defined to be public:



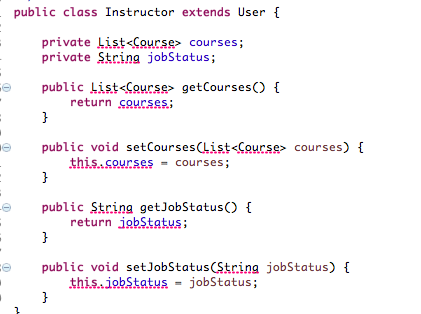
**3.1.2 Implementability**

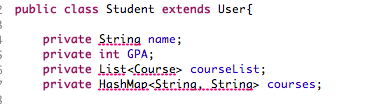
Design for reuse:

**Inheritance**:----**the open closed principle(OCP)**

One of the Object oriented programming language’s basic characteristics, one can extend class behavior by adding more methods in its subclasses. This mechanism enables developers design their system in a good practice which program is extendable. We also follow this mechanism because we use Java as our programming language.

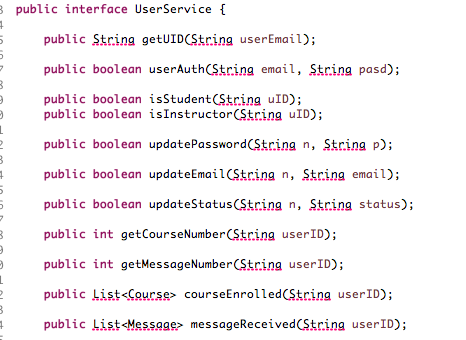
instructor and student inherit the attribute from user. This allows reuse of attributes and operation from User class in instructor and student classes. this allow the extend user class behavior without modifying user class.





**3.1.3 Single Responsibility Principle(SRP):**

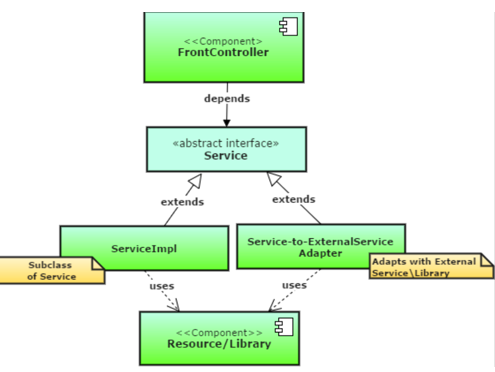
Everything in the class are designed be related to the single purpose. like userservice we put all the user related method and properties into this class:

  
The domain class should not be changed only if database tables change. we manage all our objects class in domain package. Services implementation classes use them to store database-retrieved data, multiple classes can with relative data structure can satisfy the services’ requirements.

**3.1.4. The dependency inversion principle(DIP).**

Dependency Injection is one of most essential principle of SpringMVC framwork, compared with factory pattern, we need need to care what type of service we are going to choose. We can just send request to Spring of what services we want to get, then the spring framework will provide the requested services.

We use Java SpringFramework to make the system to be layered. and the change in lower level wouldn’t influence the higher layer module;



### 3.2 Common key decision and design patterns in all components(Hongquan Yu)

While it is clear that we use Spring MVC as our project infrastructure, how does Spring follow the design patterns? This section we will present the major design patterns Spring has followed:

1. **Dependency Injection**(inversion of control) - Whenever you want to be some logic/implementation exchangeable you typically find an interface and an appropriate setter method on the host class to wire your custom implementation of that interface.
2. **Factory method** - In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface.
3. **Singleton** - In Spring, beans defined in spring config file (xml) are only created once. No matter how many calls were made using getBean() method, it will always have only one bean. This is because, by default all beans in spring are singletons. This can be overridden by using Prototype bean scope.Then spring will create a new bean object for every request
4. **MVC** - The advantage with Spring MVC is that your controllers are POJOs as opposed to being servlets. This makes for easier testing of controllers. One thing to note is that the controller is only required to return a logical view name, and the view selection is left to a separate ViewResolver. This makes it easier to reuse controllers for different view technologies.
5. **Front Controller** – Spring provides DispatcherServlet to ensure an incoming request gets dispatched to your controllers.
6. **Template method** – used extensively to deal with boilerplate repeated code (such as closing connections cleanly, etc..). For example JdbcTemplate, JmsTemplate, JpaTemplate.

### 3.3 Components Detailed Design Analysis

#### 3. 3.1: File Management Service (Ribe)

##### 3.3.1.1 Detailed Design Key Decisions and Rationale

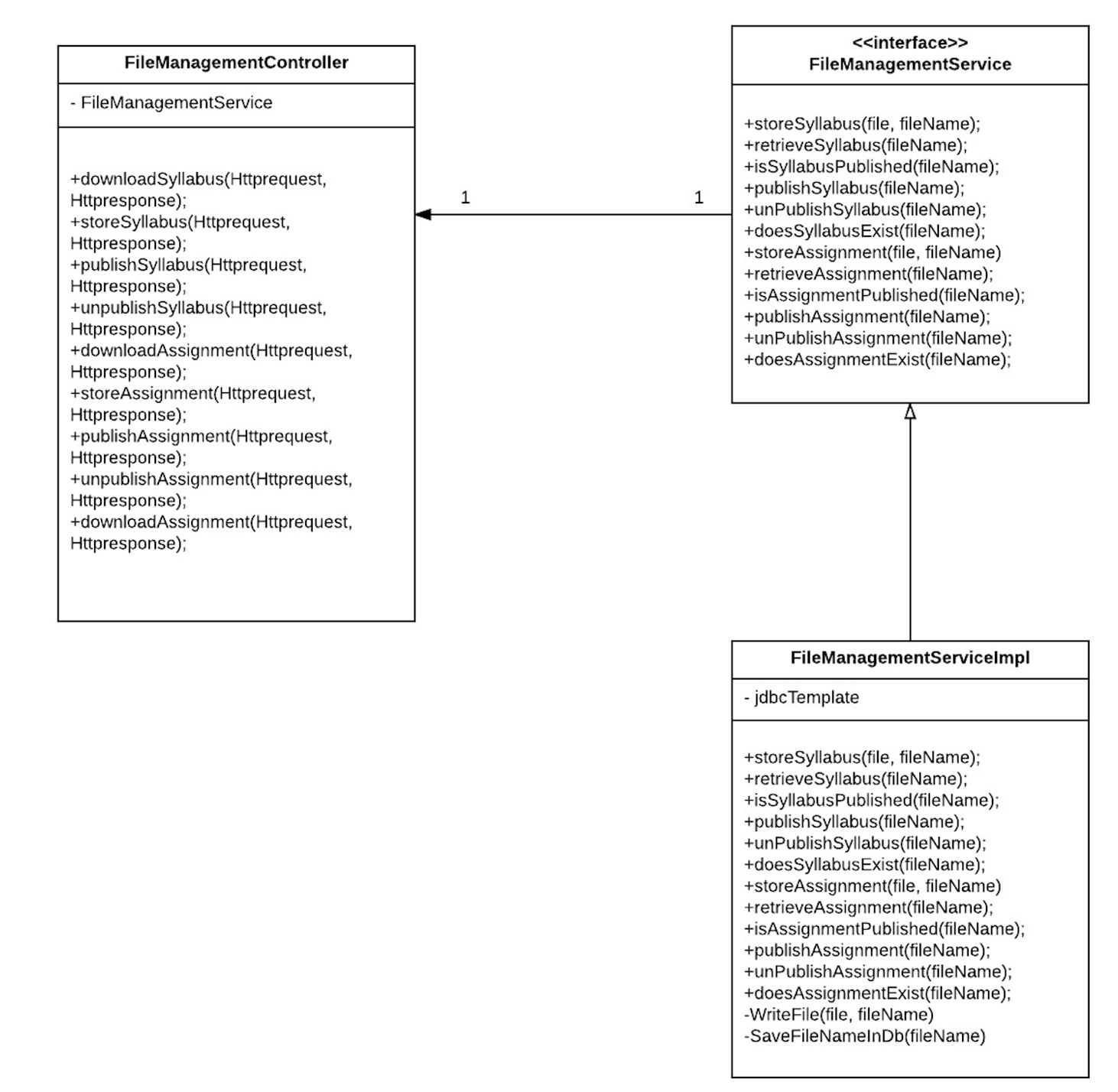
The File Management component is the only component that has access to the files stored by the application, and it is used to store/retrieve/update files for the system.

Follows the Dependency Inversion Principle (ISP) to ensure that the high-level modules do not depend on the implementation of the file management component. The high-level modules will use the interfaces defined by the file management service component to save and retrieve file.

|  |  |  |
| --- | --- | --- |
| Key decisions | Rationale | Use of design patterns |
| Use a facade design pattern for the File Management Service. Provide an interface for the File Management Service. | The File Management Service provides the interface for the application to store and retrieve a file. The way this is done/modifications to the logic of how this is done should not in any way affect the users of this class. | Facade Pattern |

## 

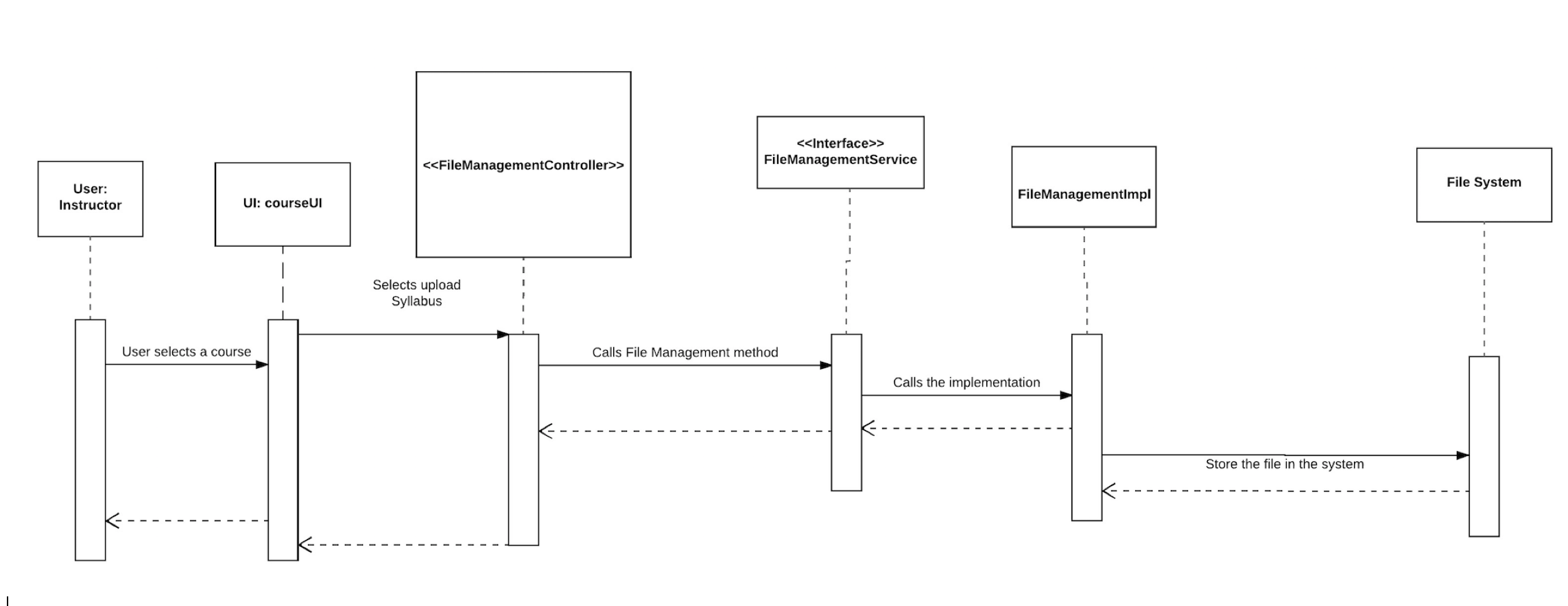
## 3.3.1.2 Detailed Design Structure

****

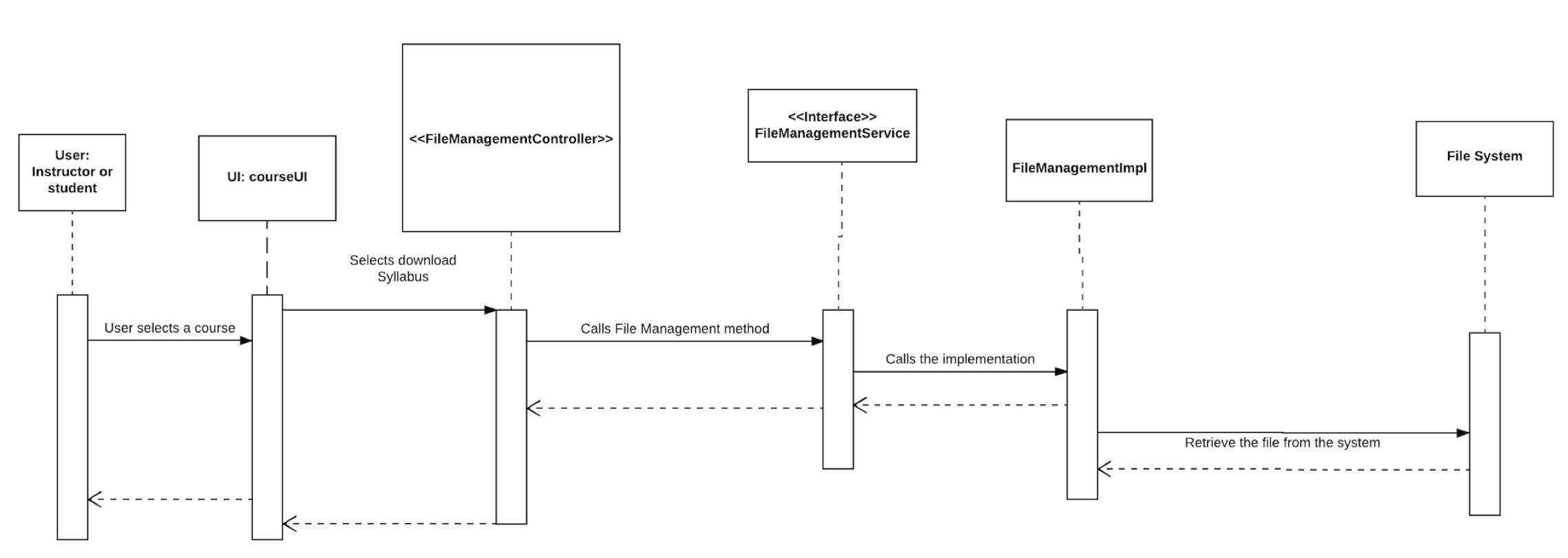
## 

## 3.3.1.3 Detailed Design Behavior

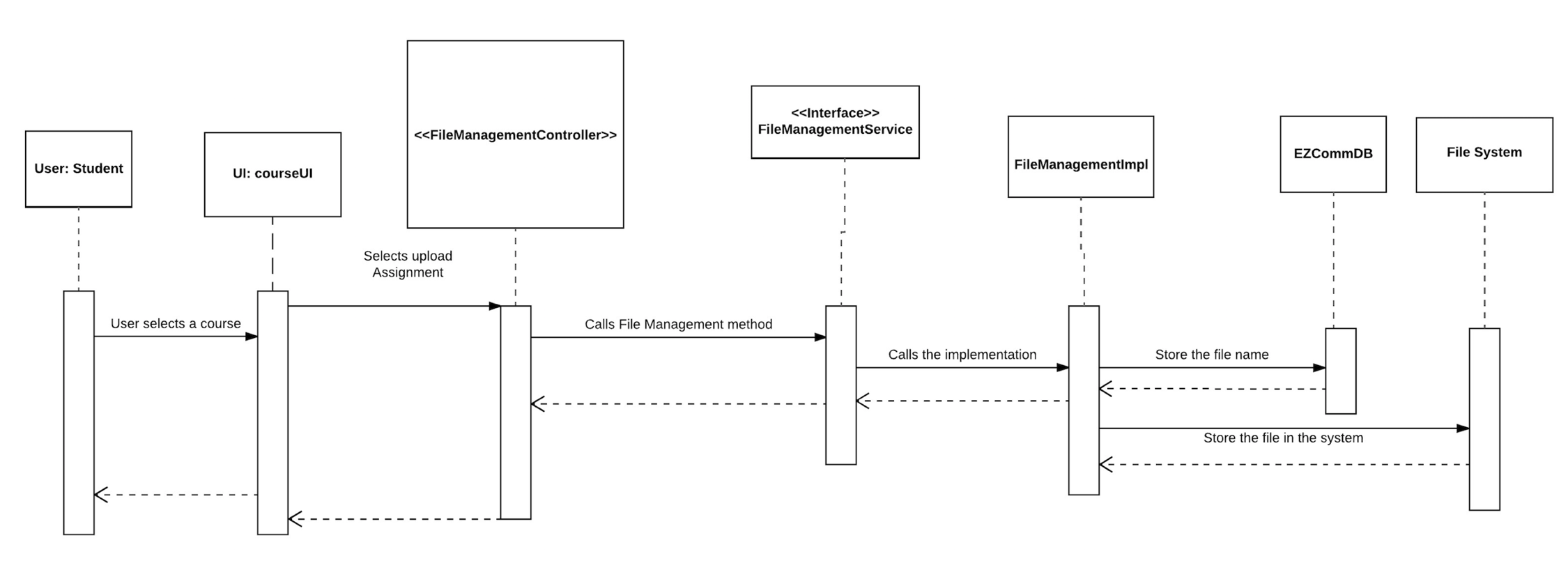
Store Syllabus



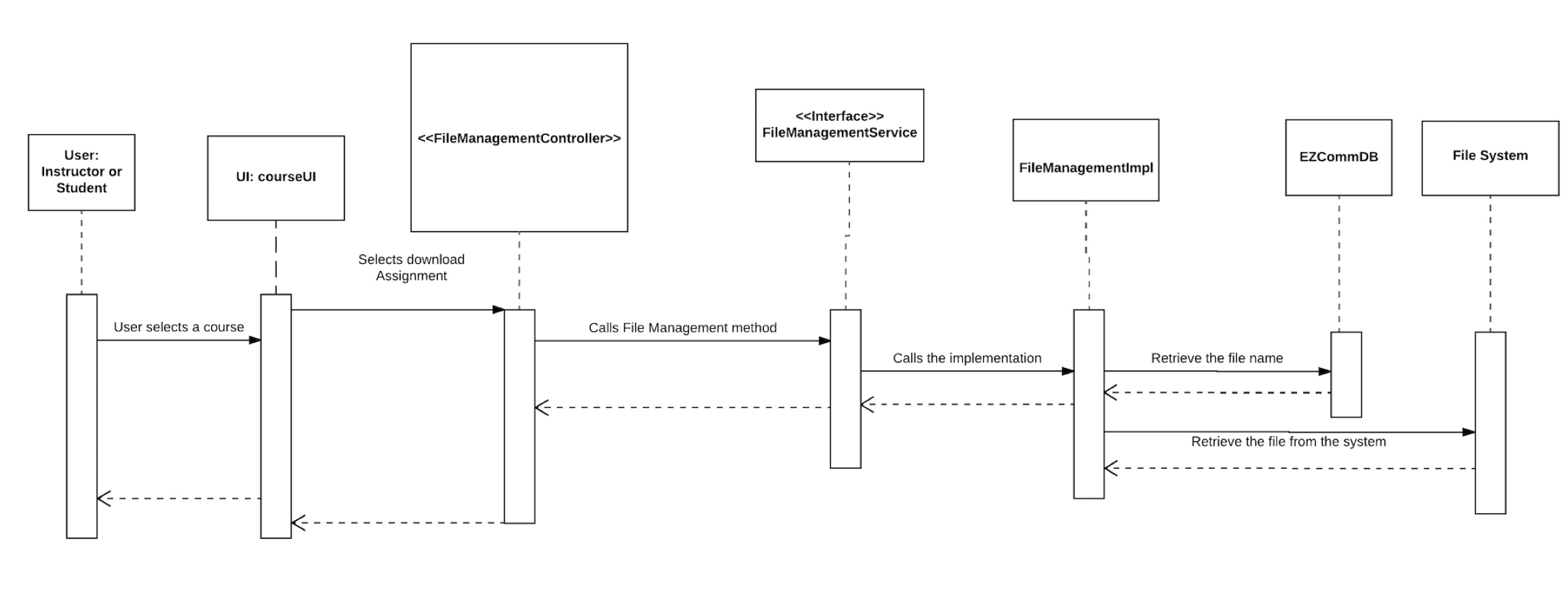
Download Syllabus



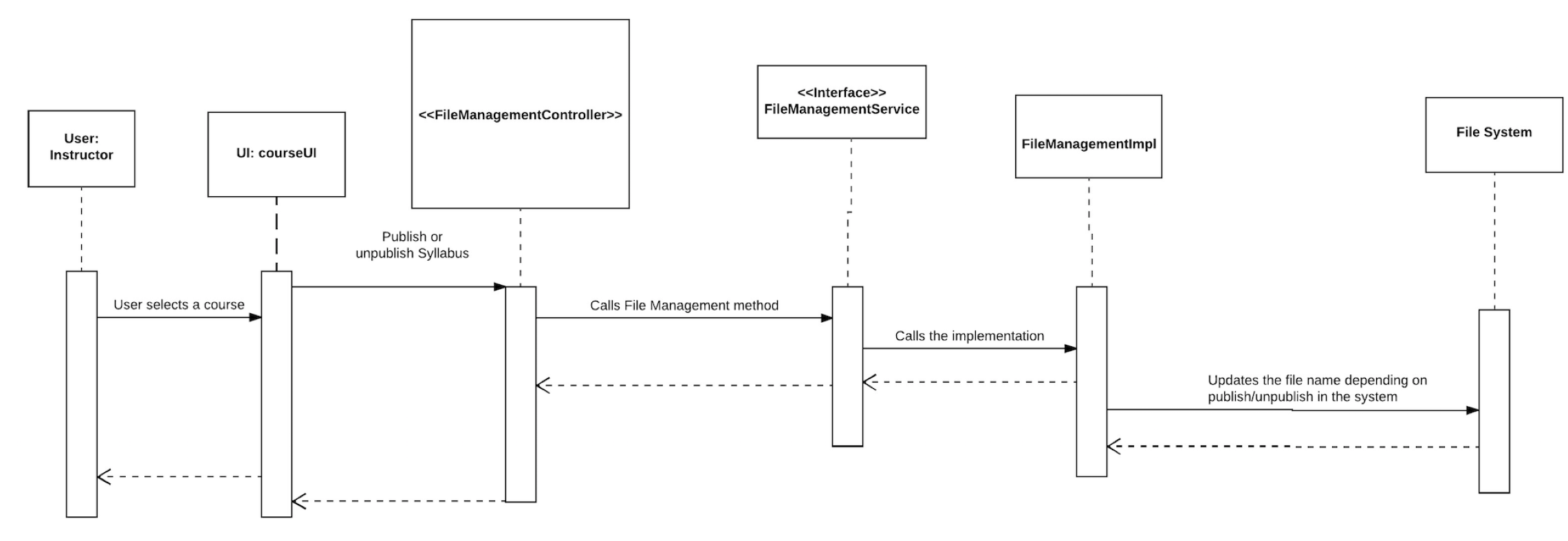
Store Assignment



Download Assignment



Publish/Unpublish Syllabus



## 3.3.1.4 Data Structures

List is used to store values/objects when a collection is required.

## 3.3.1.5 Algorithms

No algorithm is used in this component.

## 3.3.1.6 Architecture to Detailed Design Tracing

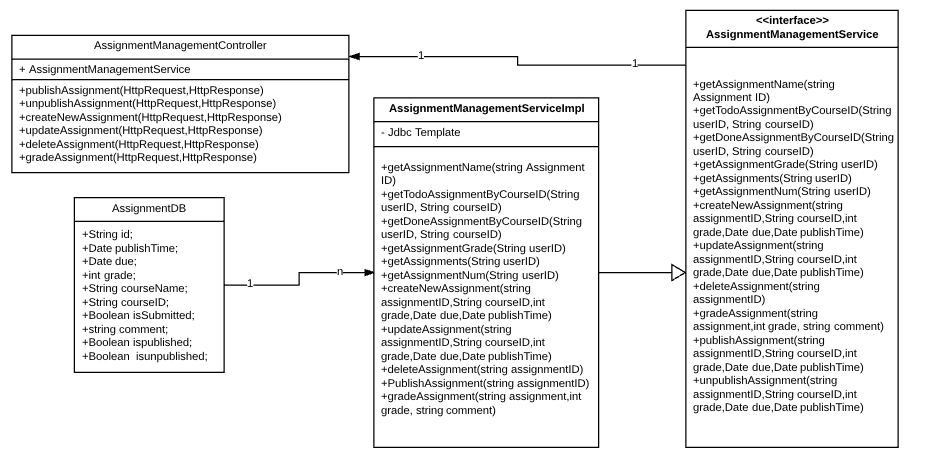
This component does not use any other components in the system. It provides an interface that other components can use.

### 3.3.2 Assignment Management (Updated by Hongquan Yu)

## 3.3.2.1 Detailed Design Key Decisions and Rationale

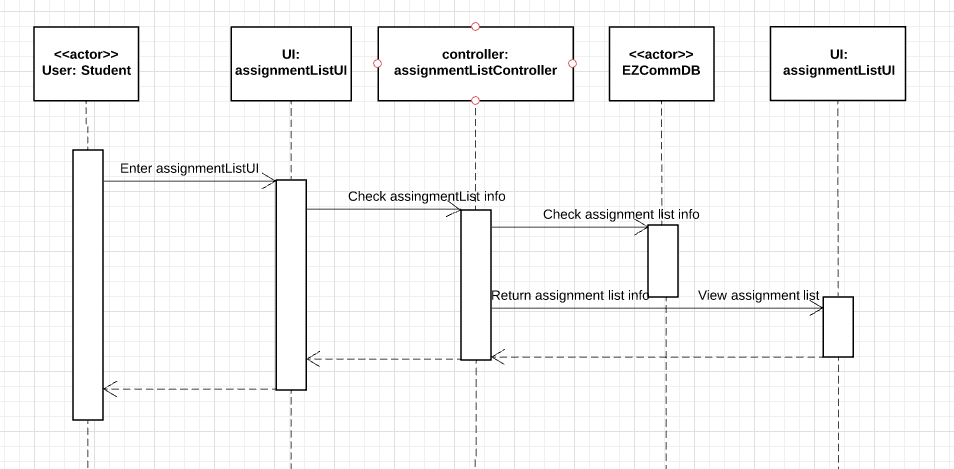
|  |  |  |
| --- | --- | --- |
| **Key decisions** | **Rationale** | **Use of design patterns** |
| According to our design, the assignment management is largely based on file management component, which means when user request assignment services, the assignment component will request file upload and download service from file management component. | The design pattern conforms to object oriented design which requires us to encapsulation the data and operation. | facade and broker pattern |
| Only course registered student and instructor can   1. View assignments for this course 2. submit and download   Instructor create, delete and grade assignment | Dependency injection of Spring framework provides services for user request. The service implemented object does a database query to check if requester is legal to do operation(create, view, submit) on the assignment. | Broker Patterns  Why: A broker mediates the interaction between client and supplier. The framework in the system provides a service for user request and database query. |

## 3.3.2.2 Detailed Design Structure <Updated by Yue Wang>

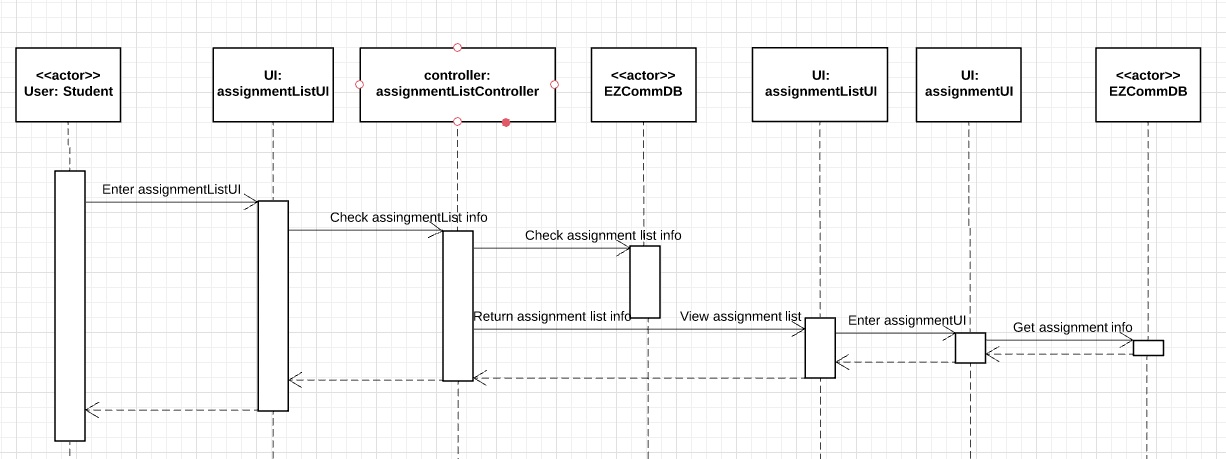
**

## 3.3.2.3 Detailed Design Behavior

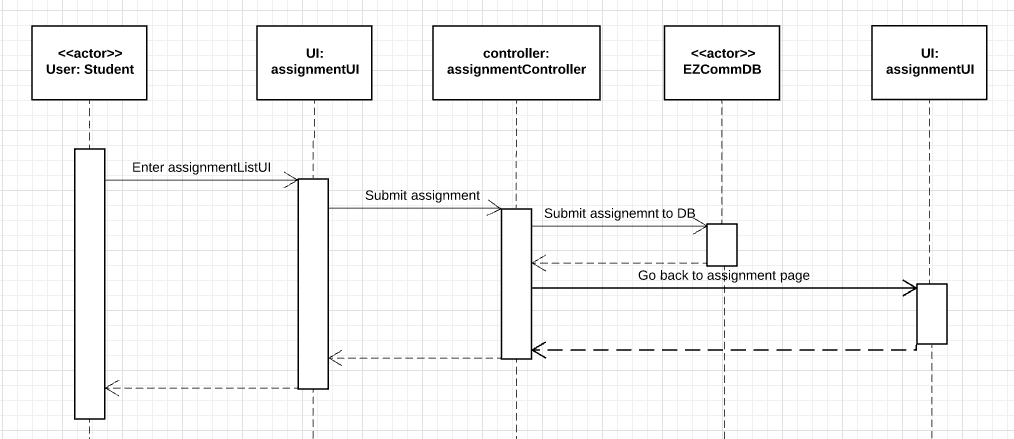
**View assignment list**:



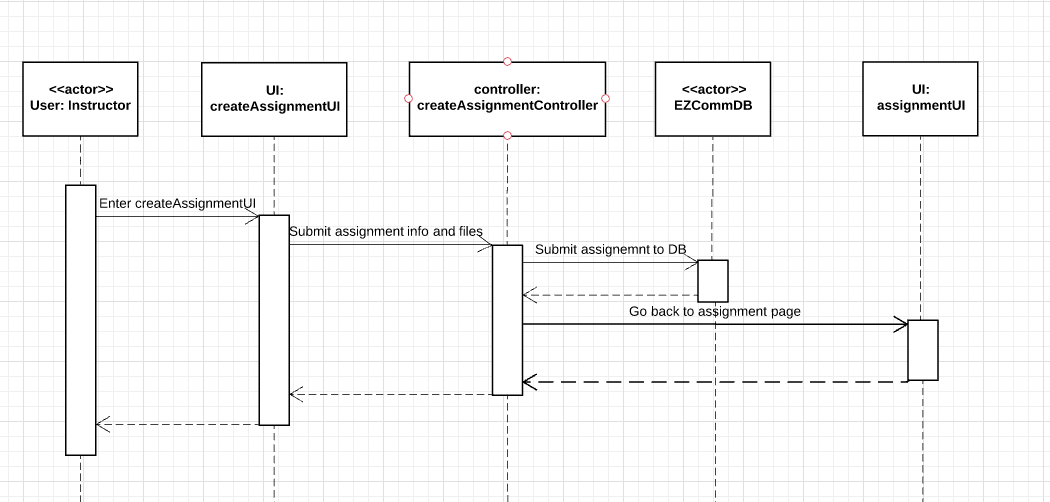
**View assignment:**



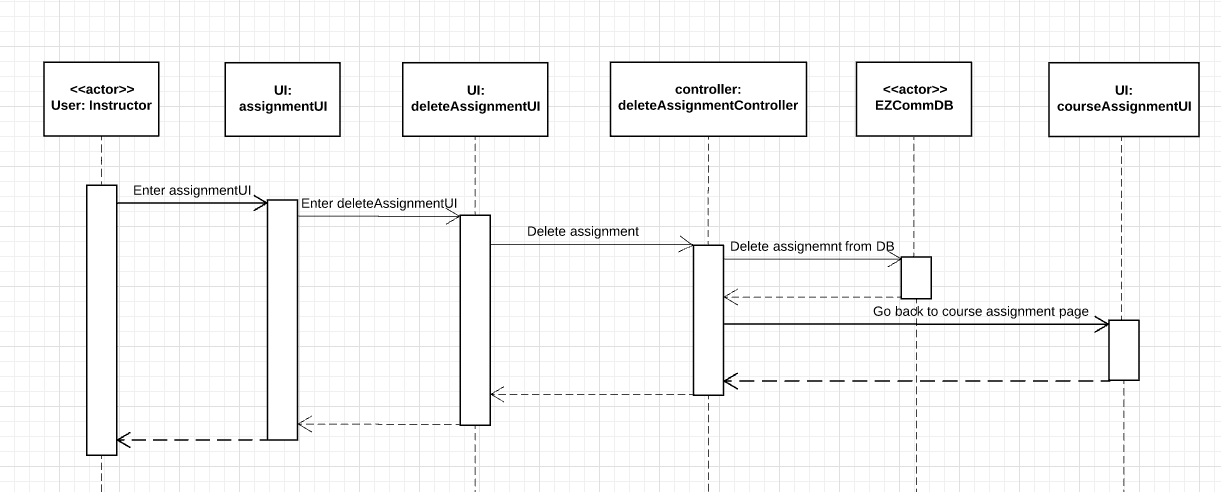
**Submit assignment:**



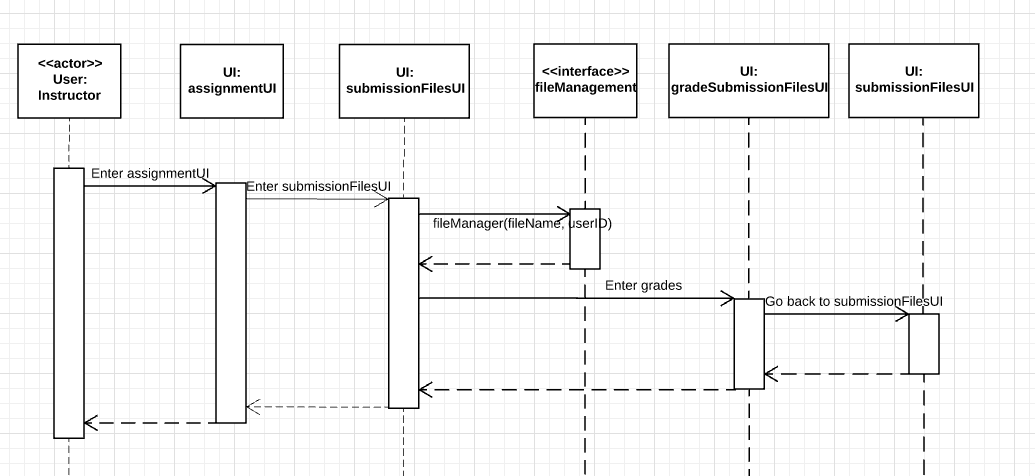
**Create new assignment:**



**Delete assignment:**



**Grade assignment:**



## 3.3.2.4 Data Structures

Java LinkedList is used to store the assignments for a particular course .

## 3.3.2.5 Algorithms

No algorithm is used in this component.

## 3.3.2.6 Architecture to Detailed Design Tracing

|  |  |  |  |
| --- | --- | --- | --- |
| **Architecture Components/ Detailed Design components** | **Assignment List Viewer** | **Assignment Viewer** | **Submission Files Viewer** |
| **User Account** | *x* | *x* | *x* |
| **Assignment Management** | *x* | *x* | *x* |
| **File Management** |  |  | *x* |
| **Course Management** |  |  | *x* |
| **User interface Management** | *x* | *x* | *x* |
| **Arch Interface** |  |  | *x* |

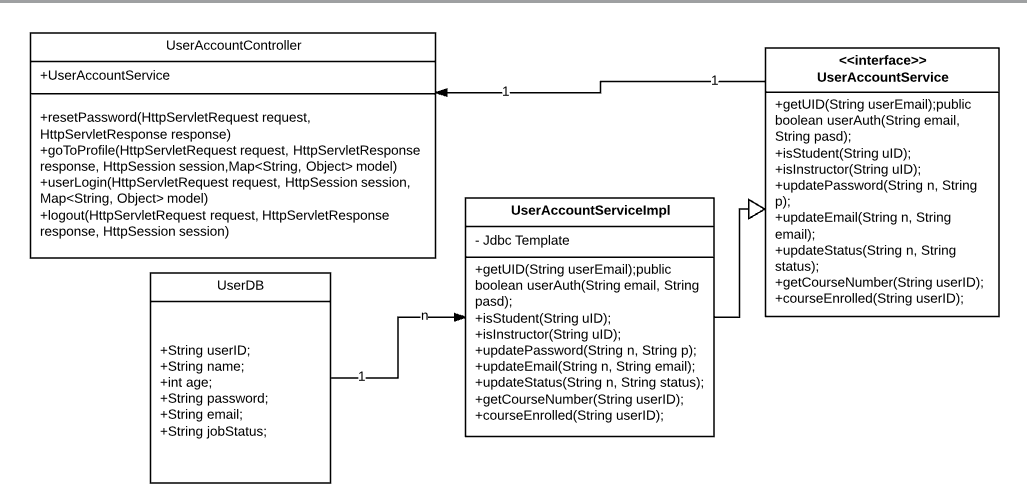
### 3.3.3 User account (Updated by Hongquan Yu)

## 3.3.3.1 Detailed Design Key Decisions and Rationale

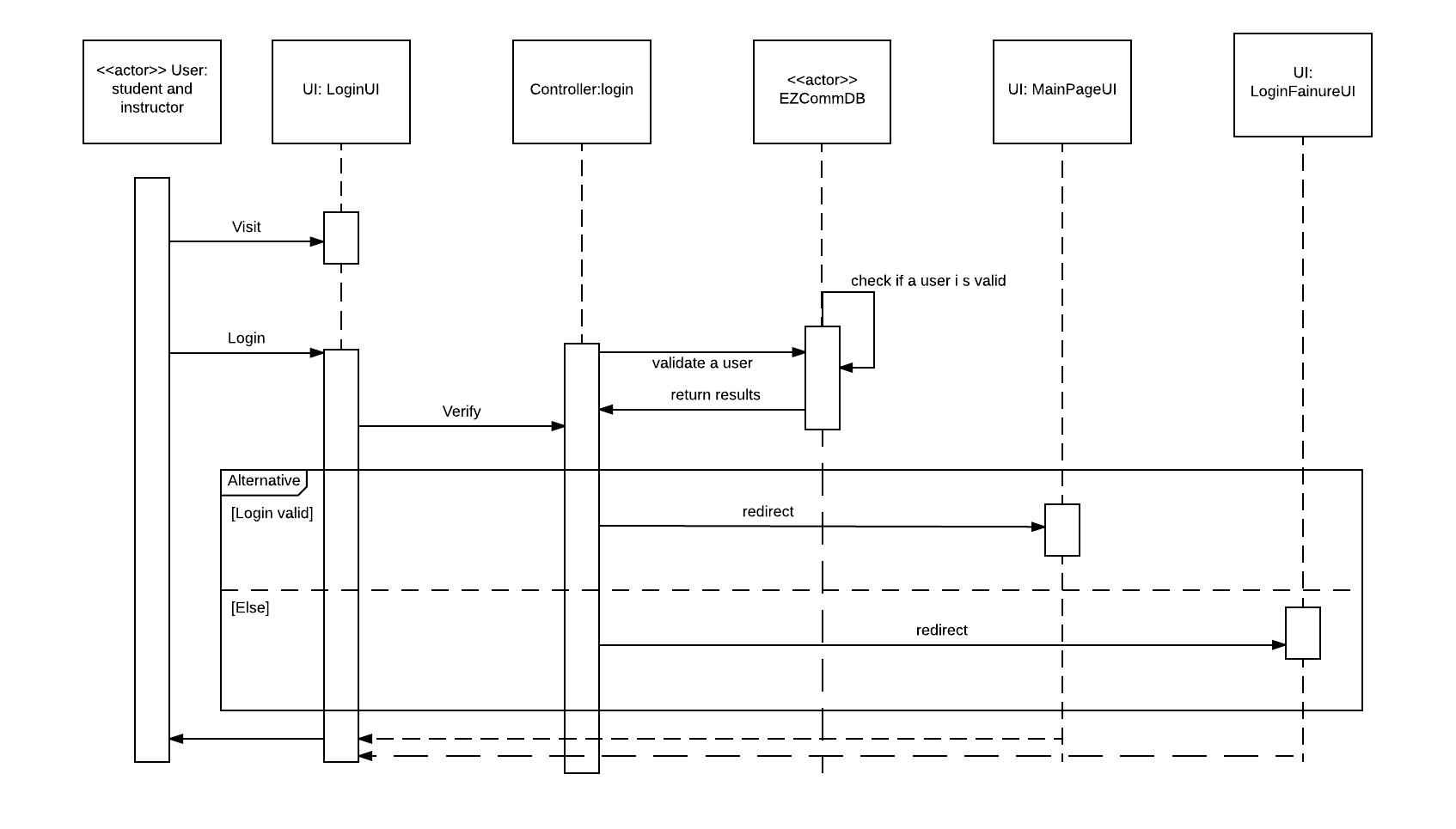
|  |  |
| --- | --- |
| Key decisions | Rationale |
| User(student and instructor) Login authentication | Easy and valid way to do user authentication that’s secure our system |
| Password updation | Only valid user can update his password |

## 

## 3.3.3.2 Detailed Design Structure<Updated by Yue Wang>

**

## 3.3.3.3 Detailed Design Behavior



## 3.3.3.4 Data Structures

no data structure is used.

## 3.3.3.5 Algorithms

Hash encryption is used to increase the security of password.

## 3.3.6 Architecture to Detailed Design Tracing

|  |  |
| --- | --- |
| **Architecture Components/ Detailed Design components** | **Login** |
| **User Account** | X |
| **Assignment Management** | X |
| **File Management** | X |
| **Course Management** | X |
| **User interface Management** | X |

### 3.3.4 Course Management (Updated by Hongquan Yu)

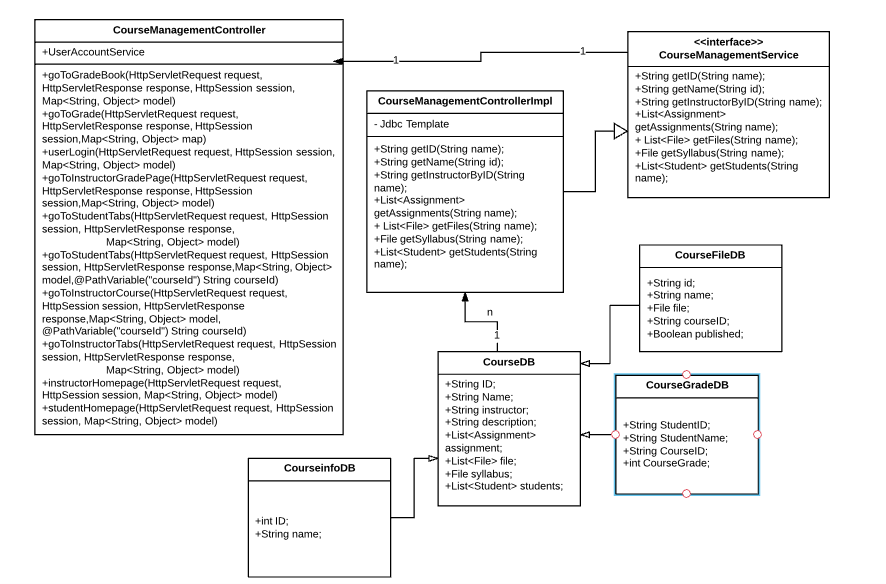
3.3.4.1 Detailed Design Key Decisions and Rationale

The course management is used for manage all the course related task.

There are several key decisions here and rationale, also the use of design patterns are corresponding to it:

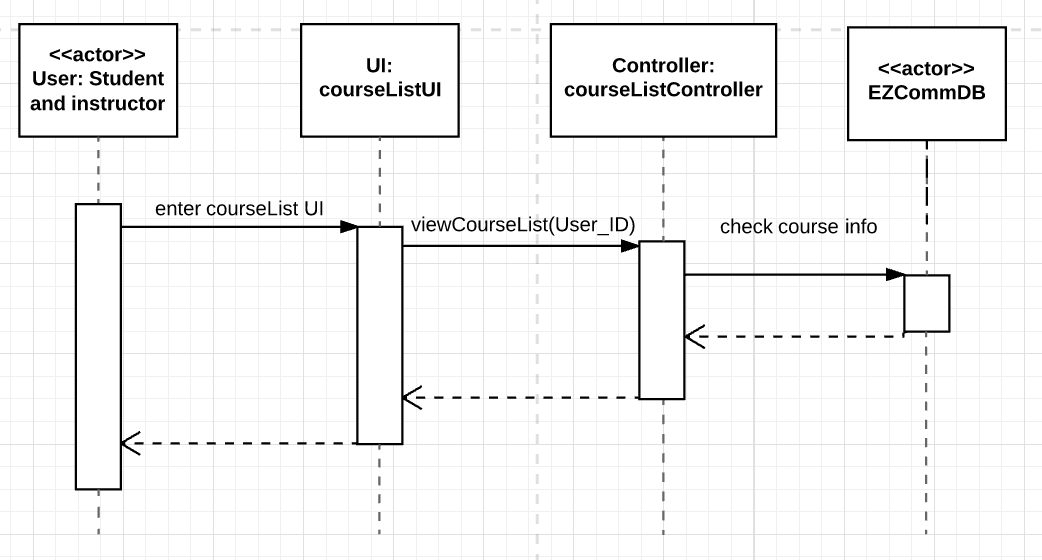
|  |  |  |
| --- | --- | --- |
| **Key decisions** | **Rationale** | **Use of design patterns** |
| According to our design, the assignment management is largely based on file management component, which means when user request assignment services, the assignment component will request file upload and download service from file management component. | The design pattern conforms to object oriented design which requires us to encapsulation the data and operation. | facade and broker patterns |
| User authentication for course access. We do the user authentication in Model part of MVC framework.  When a valid user login, we create a valid session for him/her. User must check a course within the valid session and also model did another user authentication. | Controller authentication:    In model part, we did another verification: |  |

## 3.3.4.2 Detailed Design Structure <Updated by Yue Wang>

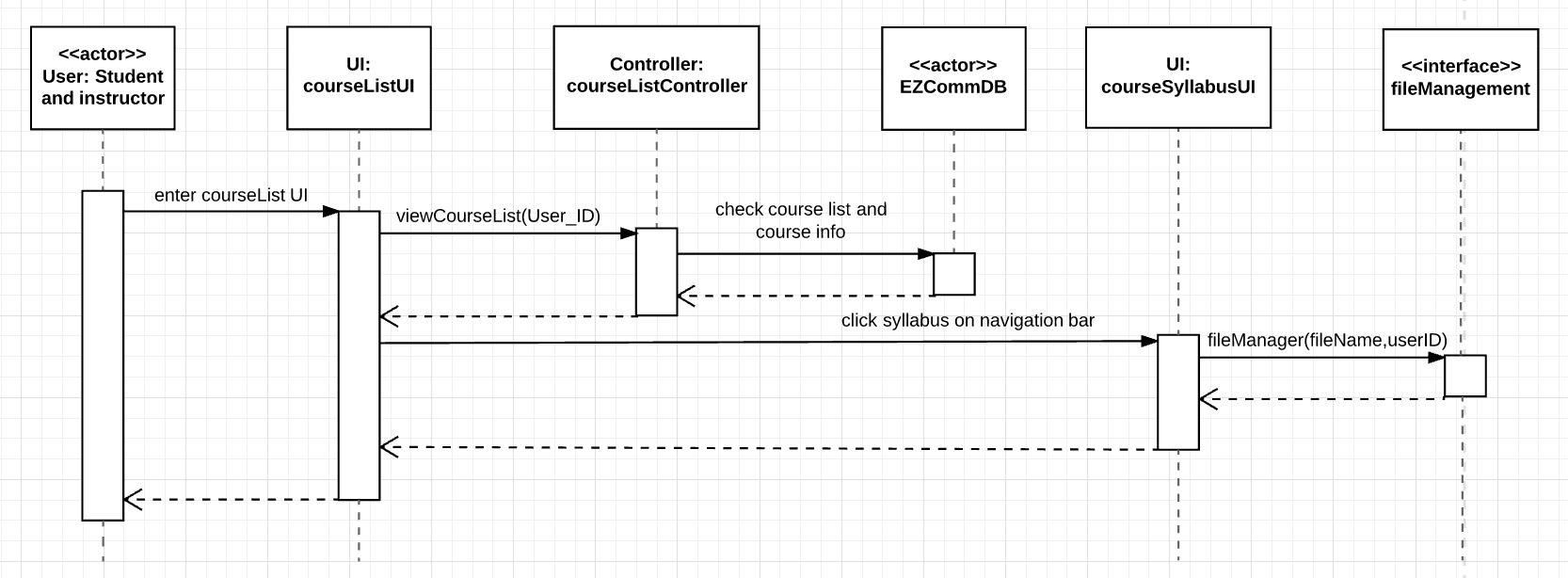
**

## 3.3.4.3 Detailed Design Behavior

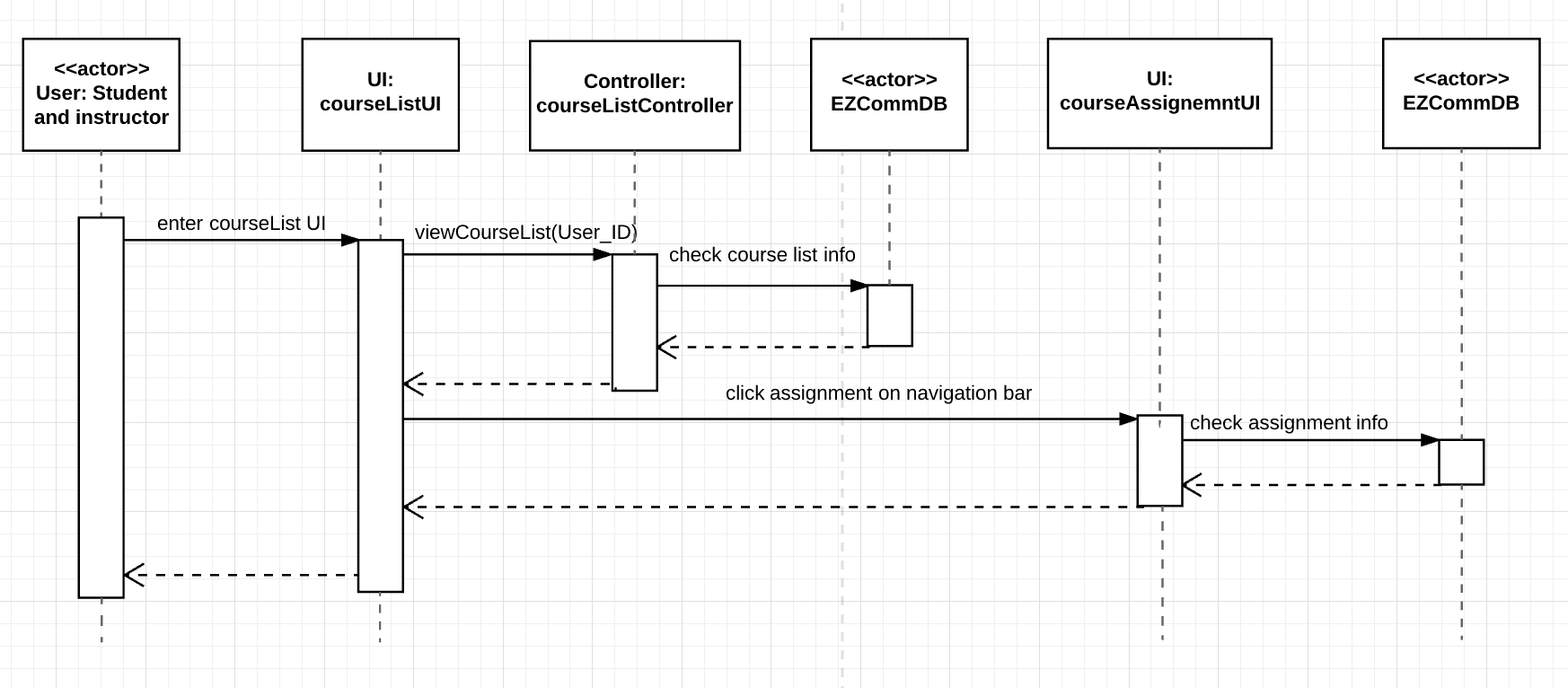
View course list



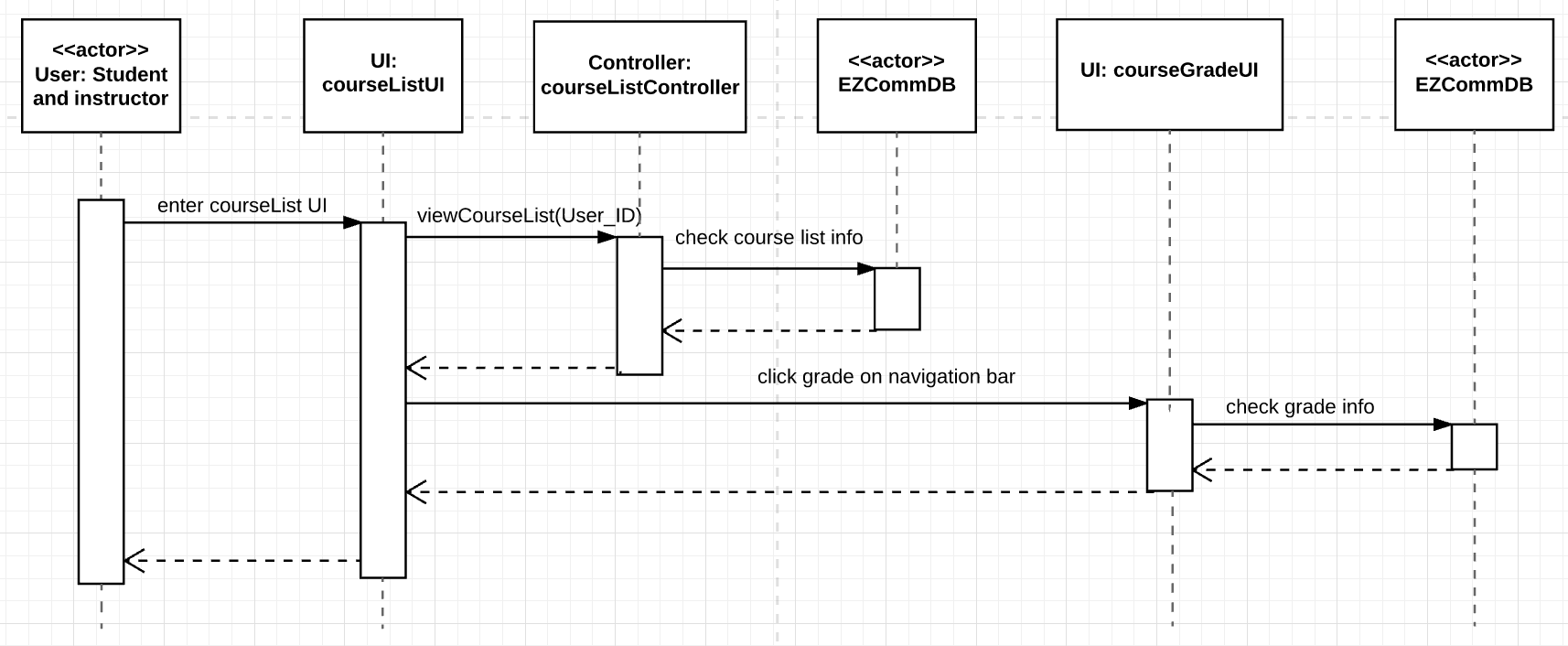
View course syllabus



View course assignment



view course grades



## 3.3.4.4 Data Structures

List is used to contain the courses in the course management component.

## 3.3.4.5 Algorithms

No algorithm is used in this component.

## 3.3.4.6 Architecture to Detailed Design Tracing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Architecture Components/ Detailed Design components** | **Course Syllabus Viewer** | **Course Assignment viewer** | **Course Grade Viewer** | **Course List Viewer** |
| **User Account** | *x* | *x* | *x* | *x* |
| **Assignment Management** |  | *x* |  |  |
| **File Management** | *x* |  |  |  |
| **Course Management** | *x* | *x* | *x* | *x* |
| **User interface Management** | *x* | *x* | *x* | *x* |

### 3.3.5 User Interface <Yue Wang>

Since former part already presents each interface class for each functional component,this part is mainly for front end detailed design analysis.

This part user interface mainly help to provide a user friendly interface to help user to easily explore the web page and get the information they want or do the expected operations.

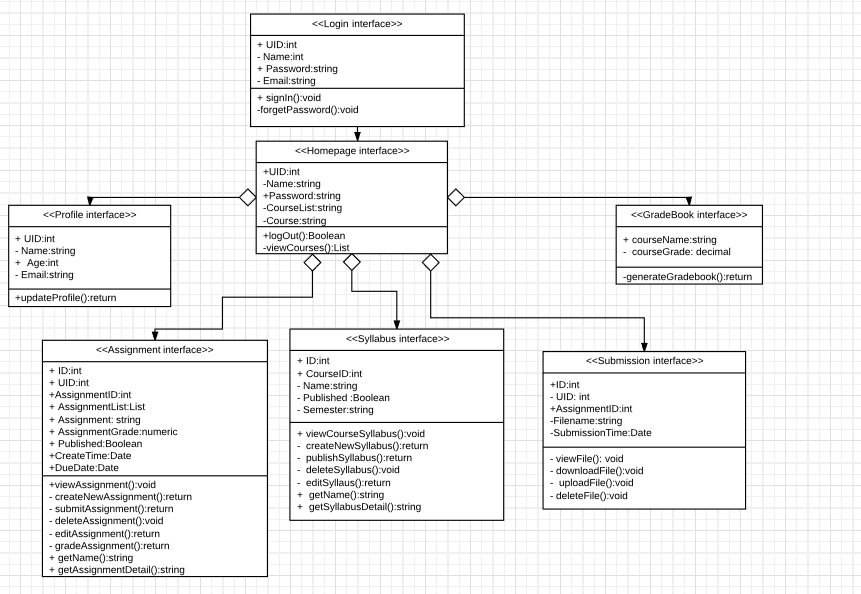
There are several user interface design patterns followed when designing the user interface:

|  |  |
| --- | --- |
| **Purposes** | Patterns |
| **Getting input from user** | Form: Fill in the blanks.  button: for upload files to the system. |
| **Navigation** | Navigation bar and navigation tab.  Dropdown menus  Content:Cards, Toggling. |
| **Dealing with data** | Tables: sort by keyword. delete. edit.  Formatting data  Images  Search:Search filter. |
| **Social:** | Student list |

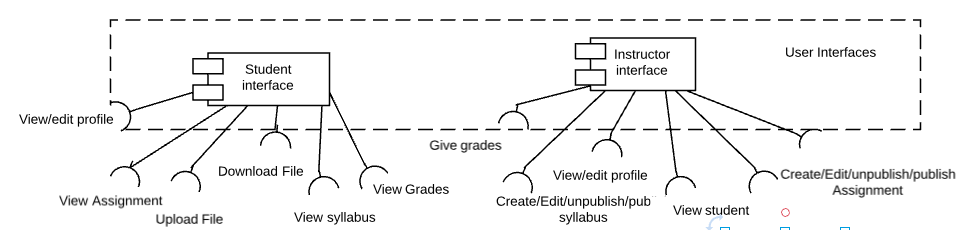
### 3.3.5.1 Detailed Design Key Decisions and Rationale

|  |  |  |
| --- | --- | --- |
| **Key decisions** | **Rationale** | **Use of design patterns** |
| Using a single login page to deal with security problem | The login page not only manages the legal accessibility of user to the system but also prevent the illegal access to the information related to courses since no course related information is presented without legal username and password. | Broker pattern  user interface serves as mediator between the instructor/student and course management system. |
| The navigation bar on the top for navigation through different sections | Following the logic of the system. Courses are the main objects ,then there are several courses we use dropdown to provide several courses options, followed with profile and gradebook for student for information review which is not directly under the course. | broker |
| Use tab as a secondary navigation to manage the course related | objects like syllabus and assignment and grades or student list are listed under each course, so use this helps make it clear that all these objects are under certain courses. | broker |
| Model is used for the further actions under certain actions | For example, the submission button for assignment submission will guide you to a model where student will be able to upload documents which reduce the webpage number and more process more consistent. | broker |
| Table are commonly used for information display | Table helps to display different kinds of information in different columns which button and other can't achieve. It also makes the actions like inserting a new row or deleting or editing this row possible. | broker |

## 3.3.5.2 Detailed Design Structure

**

## 3.3.5.3 User Interface Prototype



## 

## 3.3.5.4 Coding:

Html and CSS and bootstrap and javascript are used in this part.

Currently we have the following user interfaces pages:

|  |  |
| --- | --- |
| **Module** | **Coding File** |
| **Authentication** | login.html  profile.html  error.html  forgetPassword.html |
| **Instructor** | Instructor\_main.html  Syllabus\_create.html  Assignment\_edit.html  Assignment\_create.html  grading.html  grading.css  Assignment\_create.css  Instructor.css  syllabus\_create.css |
| **Student** | home.html  Student\_main.html  Student.css  Assignment\_detail.html |

using the tab helps saved alot of pages:  


3.3.5.5Data Structures

Basically for all the assignment and course list. linked list is used.

## 3.3.5.6 Algorithms

no algorithm is used.

## 3.5.7 Architecture to Detailed Design Tracing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Architecture Components/ Detailed Design components** | **Course Syllabus Viewer** | **Course Assignment viewer** | **Course Grade Viewer** | **Course List Viewer** |
| **Architecture interface** | x | x | x | x |