Bilkent University

Department of Computer Engineering

# Senior Design Project

*Project short-name: Peer Review System*

## Design Report

Berke Ceran, Güven Gergerli, Lara Merdol, Akmuhammet Ashyralyyev, Sıla Saraoğlu

Instructor: Eray Tüzün
Teaching Assistant(s): Elgun Jabrayilzade, Erdem Tuna

# 1. Introduction

## 1.1. Purpose of the system:

Peer review system is a helper tool for the class for grouping and reviewing projects. It is a web based application, that provides users to interact with a remote server. The application aims to ensure a convenience to all types of users in class period. Peer review systems provide users a variety of functions that vary according to user type. System allows three types of users which are Instructor, student and teaching assistant (TA). Peer review system is planned to be accessible, user friendly and maintainable. Furthermore, the peer review system web application is designed responsively, so users can access the system from all of their devices (computers, smartphones and tablets) that have internet connection.

## 1.2. Design goals:

Identifying the design goals is important for building  an application from scratch. Therefore this section presents a deeper look into the non-functional requirement that is given in the Analysis Report. The system design is client-server architecture with 3 tier architectural style [1].

### 1.2.2. Criteria

#### 1.2.2.1. End User Criteria

**Usability:**
Peer review systems provide a user-friendly interface that can be easily followed and understood by users who will use the system. The keywords or various feature names to be used in this interface will be chosen from the words or explanations that users frequently use or encounter in daily life. It will be aimed for users to follow them comfortably. Application contains clear cut menus and navigation bars that the user can easily understand and browse while using the application. The system design in a way that reduces complexity and offers functionalities open as possible. The

system provides slightly different interfaces for different user types for the purpose that ease the experience of the user by just showing the functionalities related to them. For instance, the interface that provides forming groups within students is only visible to the students. Furthermore, the interface of starting and ending group formation processes are visible to the instructor. The system provides a FAQ page that users can look at when they have confusion and if they cannot solve their problems, they can easily contact Administers by Contact-Us page.

**Reliability:**

Peer review systems ensure the security of the user’s private data. The system uses an online database to store various data. Private data such as various private information of users, uploads they make into the system, messaging channels, or various notifications will be stored in that online database and these private data will not be shared with anyone and will not be lost for any reason such as power loss, connection down and sign out. System design in a way that ensures users a safe and protected environment.

#### 1.2.2.2. Maintenance Criteria

**Extensibility:**

Object-oriented design concepts will be used to make implementation of the Peer Review System web application so flexible and easily developed.The system can be expanded and can be easily added to new things, so when new elements or features are desired to be added, new features can be added without much trouble and major changes on the main code for future use of the system.

**Maintainability:**

The codes will be coded in a maintainable way for the programmers who want to add to the source code of the student evaluation system or want to understand the source codes. Code block features such as what the code blocks are or what are they associated with will be clearly stated within the comment blocks.

**Reusability:**

Front-end implementation of the project is suitable for using in other systems. Moreover, the application will be suitable for various browsers. The implementation is suitable to reuse with minor changes for other related programs.

**Portability:**

Peer Review System web application is designed responsively, therefore users can  access the system from all of the devices that have an internet connection. The user-interface of the program provides the same elegance and functionality both in computers, smart-phones and tablets.

**Modifiability:**

Peer Review System web application uses client-server architecture design pattern which divides the program into three interconnected parts. By that pattern needed modifications can be handled without damaging all of the parts. Therefore when one of the features or interface parts needs to be changed this change only affects relative areas and not affect other parts of the system.

#### 1.2.2.3. Performance Criteria

**Performance:**

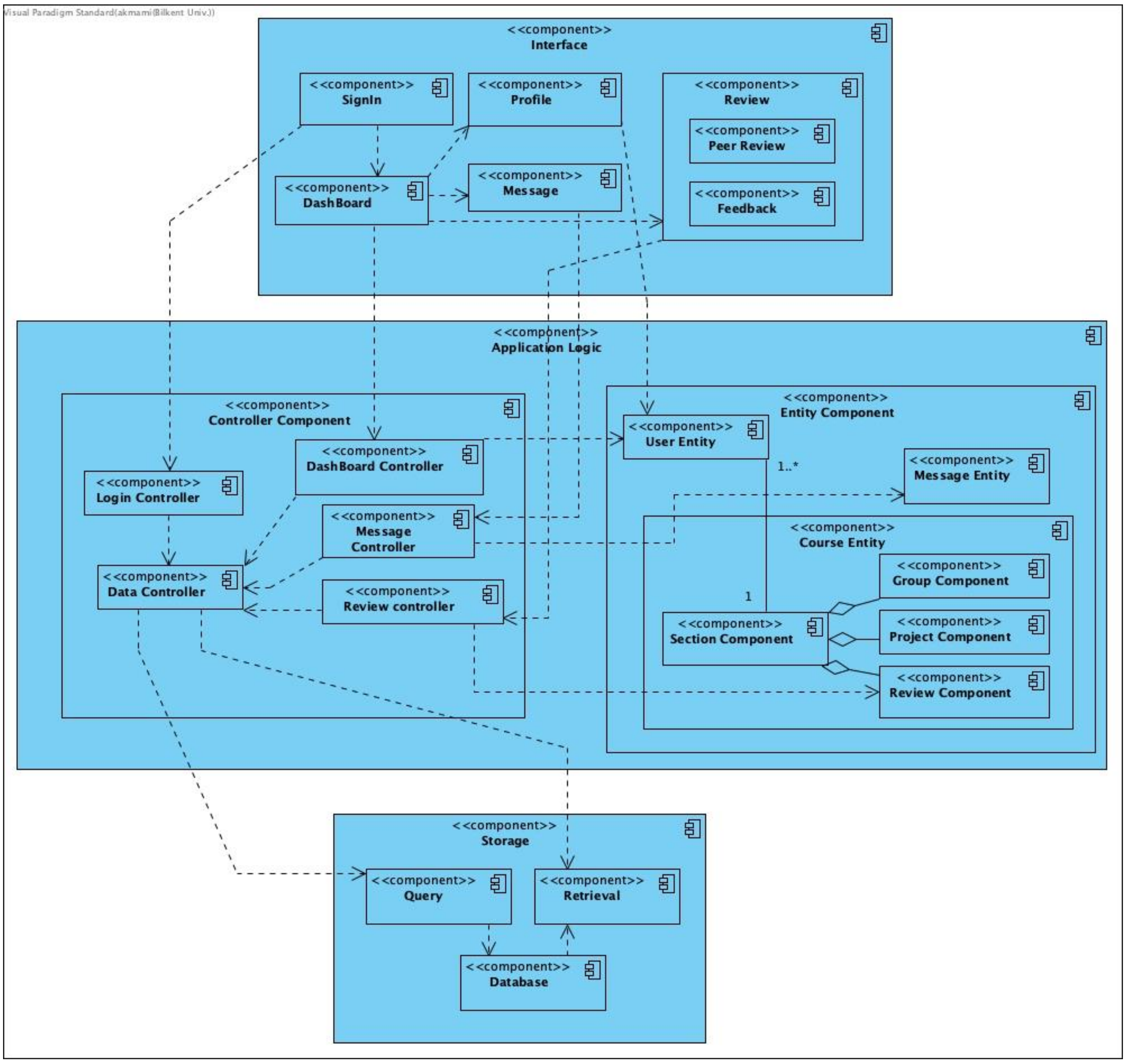
Performance is the other requirement that is crucial for the system. The response time of all operations, such as publishing a post on dashboard and sending a message to a user is aimed to be minimized. Maximum response time for general operations in the system is aimed to be 0.1 seconds. Throughput is aimed to overcome the amount of operations through the system.

## 1.3. Definitions

**Three-tier architectural Style** : Design pattern that is used in the system.
**Object-oriented design**: The approach that is used for solving software problems in the system.
**HTML:** HTML provides page structure for the user interface of the system.
**CSS:** CSS provides style information for formatting, control presentation, and layout.
**JavaScript:** JavaScript is responsible for controlling behaviours of different elements in the user interface elements in the system.
**Java:** Java is the programming language that is used in this project for controller and entity classes. This language has been chosen to create object-oriented design.
**MongoDB:** The database system which will be used in this project [2].

## 2. System Architecture

### 2.1. Subsystem Decomposition



Visual Paradigm Standard(akmami(Bilkent Univ.))
<<component>> Interface
<<component>> SignIn
<<component>> Profile
<<component>> Review
<<component>> Peer Review
<<component>> Feedback
<<component>> DashBoard
<<component>> Message
<<component>> Application Logic
<<component>> Controller Component
<<component>> DashBoard Controller
<<component>> Login Controller
<<component>> Message Controller
<<component>> Data Controller
<<component>> Review controller
<<component>> Entity Component
<<component>> User Entity
1..*
<<component>> Message Entity
<<component>> Course Entity
<<component>> Group Component
1
<<component>> Section Component
<<component>> Project Component
<<component>> Review Component
<<component>> Storage
<<component>> Query
<<component>> Retrieval
<<component>> Database

In this part, the decomposition of our system will be discussed. The purpose of the decomposition into the subsystems is to design the system in a way that will make developers introduce new functionalities and classes to the project easily and divide the subsystems for different teams of the project to develop them separately. Also, by specifying subsystems, it will make much more organized and easy to understand the system.

During the decomposition, there have been used three-tier architecture which is divided into 3 main layers as it is seen in the component diagram above and these 3 layers have only access to one level below layer [1]. These three layers are:

1. **Interface:** Interface, which is responsible for all interfaces of the system, will interact with the user and system. This subsystem takes a role of boundary and does functionalities that have been invoked by the user by clicking reserved buttons/links. Also, this system has been separated from the rest in order to track errors easily and since the technology being used in front-end changes too often (related to browsers, libraries and languages), it will be very efficient to adapt to these changes within the subsystem without changing the others. Moreover, since we are using a whole different platform (HTML/CSS/Javascript) for implementing front-end, it is appropriate that it is divided into a different subsystem[3].
2. **Application Logic:** The application logic consists of entities and controllers. The entity part is the least likely to have changes because general classes (type of users etc.) will not be changed frequently in the future use of the application. Also, introducing new functionalities will be very efficient since it is also separated within the subsystem into two parts as it is shown in the diagram. Controller component is responsible for interaction between the interface and the storage subsystem of the project. Within the Application Logic, coherence is increased while coupling is reduced in order to make the system more adaptable to the modifications.
3. **Storage:** Storage which will store the data as well as retrieving or querying them. This part has been decomposed into another layer in order to increase the security of the system so other subsystems will have less access to the storage. Consequently, all operations relating to the storage will be done within the subsystem when they have been invoked by Application Logic layer’s components.

Three-tier Architectural Style has been chosen as a design pattern due to the following reasons:

1. The system is separated into 3 main layers, which are interface, application logic and storage, which makes it easier to track the errors.

2. Security is important in web applications, so, separation of storage subsystem is required since the operation related to the database will be done only by this subsystem.
3. Technologies where our application will run will, ex. browsers, and libraries which will be used in front-end development changes so frequently that it would be very difficult to modify the whole project if the interface was not separated from other subsystems.
4. The communication between controller and entity objects will be high, which is coherence while the communication between layers are low, which is coupling. The ideal system should be minimized in coupling and maximized in coherence, which we have done while decomposing the system.
5. The three-tier architecture is simple and it will increase the efficiency in the implementation of the project.

In conclusion, our system is decomposed in order to be as much as developer friendly and modifiable and adjustable.

## 2.2. Hardware/Software Mapping

Peer Review System will be implemented in front-end HTML, CSS, JavaScript, and back-end will be implemented in Java programming language. Although we use 4 kinds of languages as the software language, all transactions will be done mostly by Java. Gradle, another software tool that is necessary for the project and compatible with Java, will also be used in this project. Since Gradle will be used and to avoid problems and run smoother, the Java Development Kit must be in 8 and higher versions.

The people who will use the application Peer Review System do not need very systematic and high-level tools in terms of hardware, and it is sufficient to have a proper mouse and keyboard in terms of hardware requirements. In this application, the mouse will be used to click different buttons and menu selections related to application properties. In addition to this, the keyboard will also be used for various input and output (I/O) operations within the application as a hardware requirement. In addition to the computer, since the application is opened via the browser, users can also enter the application with a smartphone with internet access. As a result, when looking at these requirements, it is seen that the users who will use the application can use this application comfortably with a minimal computer and smartphone and the tools next to it.

On the storage side of the application, the various information of the users using the application or the operations they perform within the application must be stored in a certain place. A database is required to store this various data and the MongoDB database will be used for these operations. In addition, since this application works over the internet, users must have internet connection.

## 2.3. Persistent Data Management

Peer Review System application requires data storage or a database for various tasks. In this application, users' profile information, their chat correspondence with other users or various files that they can upload to the application, the courses, sections, groups in the sections; students, instructors and TAs etc. will be stored in the MongoDB database. In this way, even if there is a malfunction on the end-user side, the data will be restored again as they are stored on the local host of the server.

## 2.4. Access Control and Security

Since various information about users in the application and other operations will be stored in the MongoDB database, their security is also very important. While this information is stored in the local host of the server, the people who can access this storage area are only the application owners, and there will be no problems in terms of security because it is necessary to have various private keys for this access.

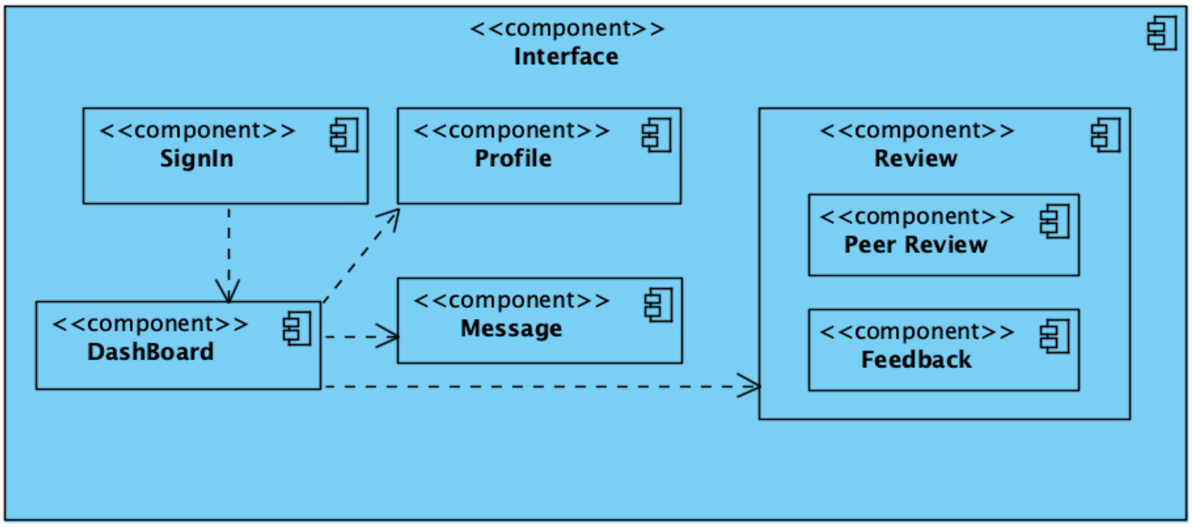
## 2.5. Boundary Conditions

The Peer Review System application does not require operations such as a download or copying from computer to computer, because this application runs on internet browsers, users will be able to access this application by connecting to the internet without downloading any files. This feature will increase the portability of the application and the number of users it can reach.

After users open the application on their browser, a sign-in page will appear. When the users which are the members enter the correct information here, their information will be transferred to the application through the database and this information will be presented to the user. Users will have 2 different options to exit the application after opening the application through their own browser, the first of which is that they can exit the application by using the sign-out feature in the application, and at the same time, they can exit the application by closing the browser directly.

## 3. Subsystem Services

### 3.1. Interface Subsystem



Interface subsystem is responsible for providing UI and establishing the interaction between system and user. It contains 5 major subsystem which are:

1. SignIn
2. DashBoard
3. Profile
4. Message

5. Review
    a. Peer Review
    b. Feedback

SignIn interface is basically responsible for logging and signing up functionalities in the system. It will take inputs off email and password which will be sent to the Login Controller subsystem within Application logic. On the other hand, it will have responsibility for signing up which is taking required information from the user and by sending them to the Login Controller for the account verification (make sure that those informations are valid and account has not been created).

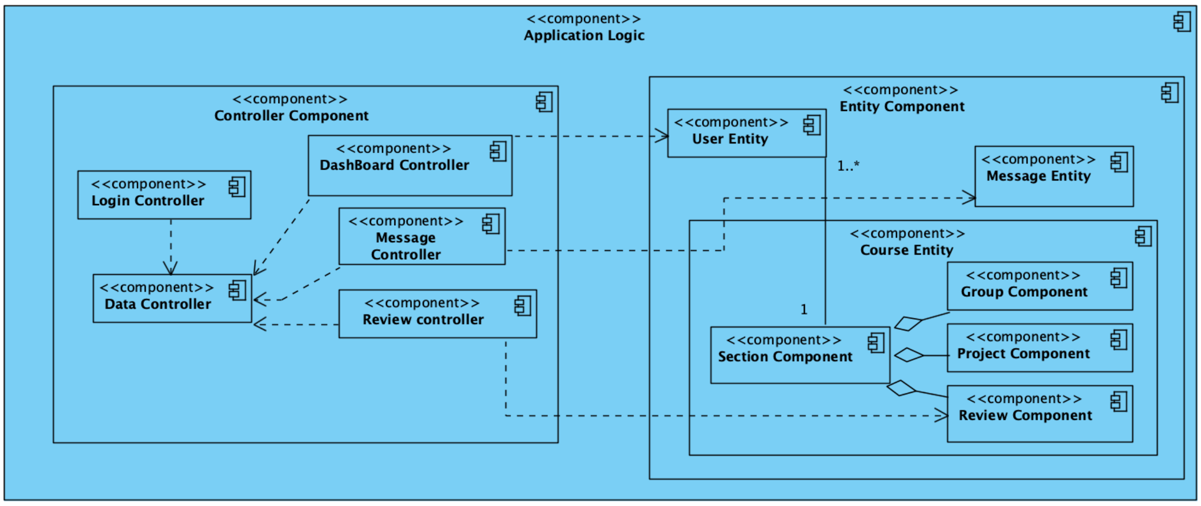
DashBoard interface is responsible for creating a dashboard for the user which varies from student, TA, and instructor. It will communicate with DashBoard Controller for performing those tasks as well as allowing users to send messages in the main timeline. On the other hand, it will be responsible for the navigations to profile pages (Group, section, user), Message page, and Review Pages.

Profile interface is responsible for displaying personal/general information of users/groups which will be extracted from entities.

Message interface is responsible for displaying all conversations as well as messages on this page. Also, this interface will be connected to the Message Controller which will be performing sending and receiving messages.

Lastly, the Review interface is responsible for displaying review interfaces to users and getting their responses. This subsystem is divided into two subsystems since feedback will be done to the class, work or project, while peer review will be performed between users (group members). Consequently, they will have different interfaces, for example in peer review, user will have to choose for who he/she will review. The interface will be connected to a Review Controller which will handle all operations related to review.

## 3.2. Application Logic



Application Logic is responsible for all operational logics in a system such as, controlling objects, checking or validating data, performing function, processing input, and notification as well as entity objects. This layer is divided into 2 main subsystem which are controller and entity components. Controller Components consist of 5 subsystem which are:

1. Login Controller
2. DashBoard Controller
3. Message Controller
4. Review Controller
5. Data Controller

Entity Component consist of 3 main subsystem which are:

1. User Entity
2. Message Entity
3. Course Entity
    a. Section Component
    b. Group Component
    c. Project Component
    d. Review Component

Login Controller is responsible for validating the login inputs by communicating with the data controller. Also, it is responsible for creating new accounts by checking the information and sending them to the data controller for insertion to database. On the other hand, the password reset requests will be handled with this controller.

DashBoard Controller is responsible for all operations, mainly displaying the dashboard according to the users' data and group formations. The data required for displaying and forming groups will be done by communicating with the user entity and data controller.

Message Controller is responsible for handling message sending and receiving and sending notifications to users. This component will communicate with the data controller for inserting or retrieving messages to the database and message entity.

Review Controller is responsible for creating review questions as well as answering them (assigning answers). For storing reviews to the database, this component will communicate with the data controller and review component.

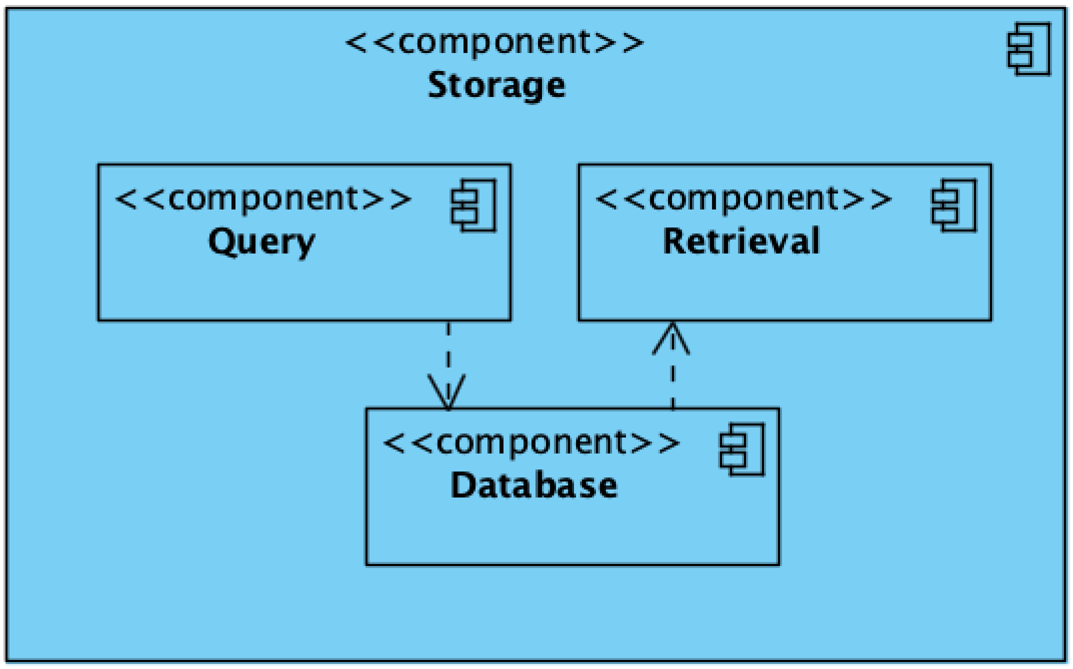
Data Controller is responsible for all operations related to the database. This controller will communicate with the Storage layer and by preventing all controllers connecting to the Storage layer directly, cohesion will be decreased and coherence will be increased as it is desired.

User entity is a component of the user class. This component will be created whenever a user will login to the system. All required information ( ID, name, surname, group, class, etc.) will be provided by the database during login.

Message entity is a component of the message class. Also, this component will serve for notification purposes. The related information such as receiver, sender, message will be stored to the database by message controller.

Course component contains all components related to the course such as sections, projects, reviews.

## 3.3. Storage



Storage subsystem is responsible for all operations related to databases. This subsystem consists of 3 subsystem which are:

1. Query
2. Retrieval
3. Database

Query component is responsible for inserting or updating data in the database. This data can be user, group, project, course, message, review or some part of data for updating functions such as group members, new messages etc. This subsystem is connected to Database which is the main storage in the system.

Retrieval component is responsible for searching and retrieving the data in the database. This data can be name, surname, ID, group, section, review etc. and operations will be done according to them. This subsystem will have connection with Database and retrieval of data will be done from there.

Database components will establish the connection between system and database. The connection will be done whenever the system will require insertion or retrieval of data from the database [4].

# 4. Low Level Design

## 4.1. Trade-Offs:

### 4.1.1 Space vs Speed:

Speed of the web based application is one of the most crucial trades for it, however our program needs to store large amounts of data such as user names, uploading files, reviews. Furthermore Peer Review System web application uses a database to store user information and other crucial data. Therefore, speed is a trade off with memory in implementation.

### 4.1.2 Development Time vs Performance:

For the front-end part, Peer Review System web application uses HTML, CSS, and Javascript because it is easy to learn, has a rich library and simply to navigate. However, there are more high performance open sources like ReactJS. React is a more flexible library that mixes HTML with JavaScript. But react is more complex and hard to understand, thus due to having a limited time for implementation, we choose to use HTML, CSS, and Javascript instead of more efficient libraries.

### 4.1.3. Functionality vs Usability:

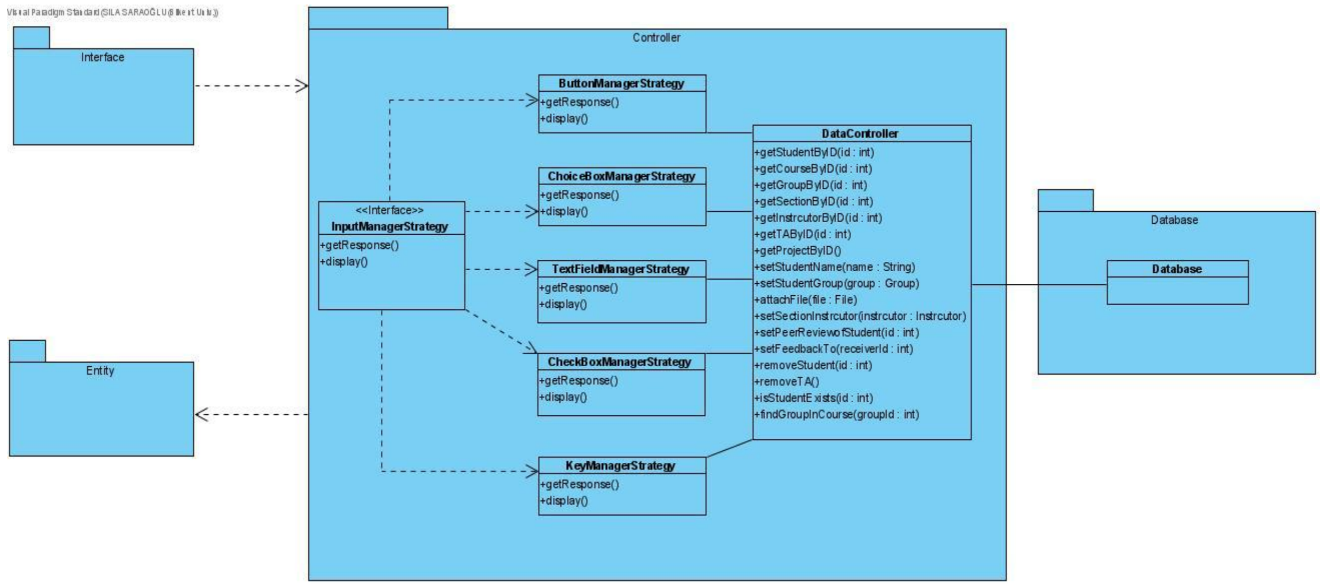
The system prefers usability over functionality unless the system satisfies main functions. Therefore, during the period of analysis the project we focus on main tasks instead of unnecessary additional tasks. Although we prefer simplicity, the Peer Review System web application offers many functions to the user that are detailed in the Analysis Report. Majority of those functions are easy to understand by the help of the user-friendly interfaces that the application has and they are functional.

### 4.1.4. Memory vs Maintainability:

The system prefers maintainability over memory because the classes in the project will have many common attributes and operations which will cause unnecessary memory allocation. Since our aim is to maintain main functionalities of the application, we will try to implement interface classes or try to connect the classes with inheritance to reduce the memory usage to minimum. On the other hand, since we grouped the common operations and attributes together, it would be easy to maintain and update the system.

## 4.2. Final Object Design

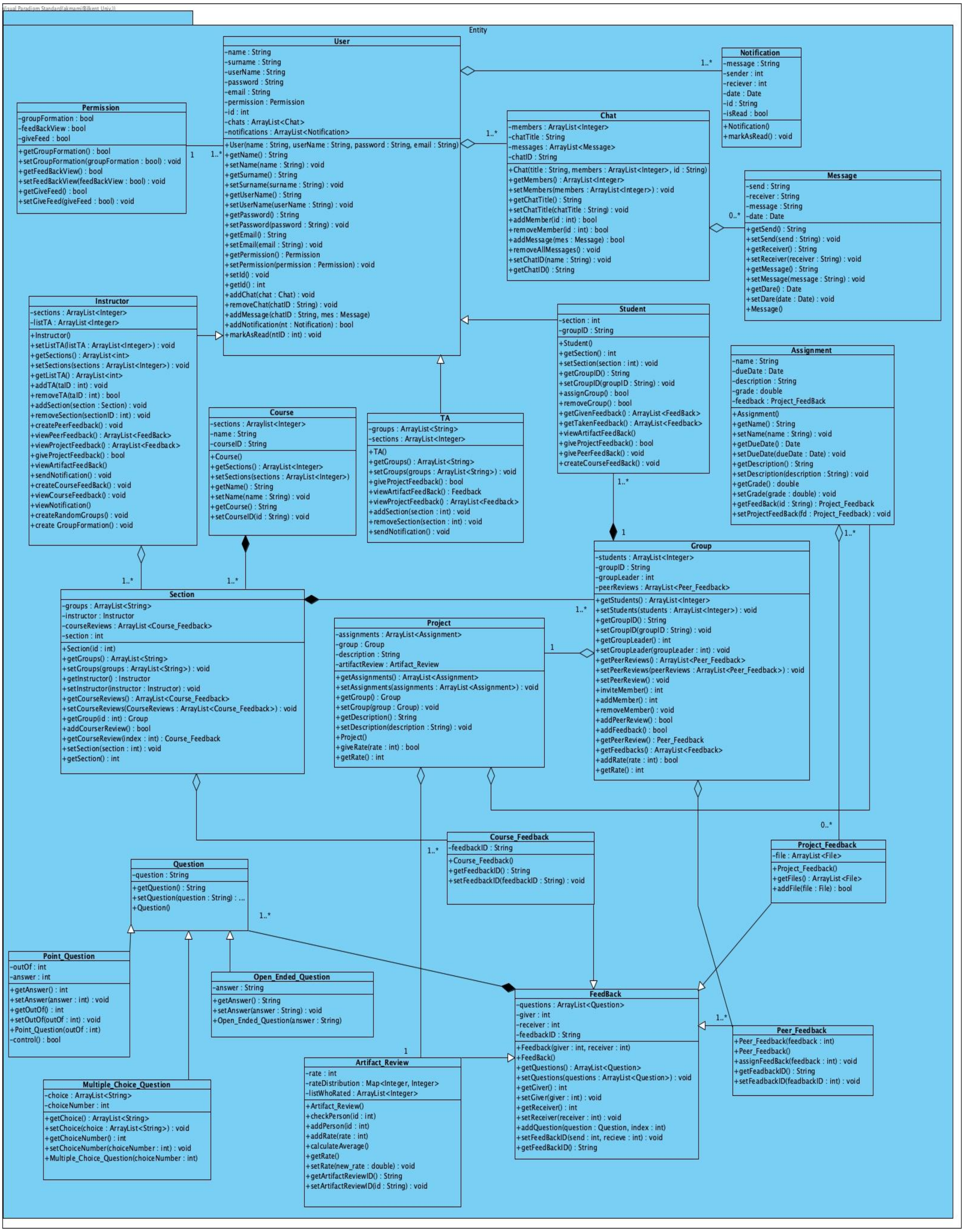
The diagram of final object design is divided into two pieces to give a more clean structure .



When the low level design of the project is designed we divide the classes of the same purposes and put them into the same packages. Since we are using HTML/CSS and Javascript for our interface of the project, we divide them into one package called interface. As is the nature of the HTML, CSS, and JavaScript; we did not have any classes for our user interface. The package is drawn for showing that they do interact with the controller package. As we indicate in the subsystem decomposition, we are using three tier layer architecture and each layer has only access to one layer below. Our first layer is an interface package. The interface package only interacts with our InputManagerStrategy interface class in our controller package. We think about our controller and entity package as our second layer of our architecture called Application logic in our subsystem decomposition. This second layer has only access to the database package as the last layer of the project.

In the controller package, we tried to implement a strategy design pattern to project as implementing an interface class called InputManagerStrategy and concrete strategy classes which implements this interface. Then this manager strategy classes are associated with the DataController class which interacts with our database system. The reason why we implement a single class that interacts with our database is to meet the requirements of security purposes. The database package in the diagram represents the storage subsystem in the subsystem decomposition.

Above, we did not give details of the entity package to give a more precise picture of the interaction between packages. Following is the content of our entity package which interacts with our controller package in the application logic subsystem [6].


Visual Paradigm Standard(akmami(Bilkent Univ.))
Entity
User
-name : String
-surname : String
-userName : String
-password : String
-email : String
-permission : Permission
-id : int
-chats : ArrayList<Chat>
-notifications : ArrayList<Notification>
+User(name : String, userName : String, password : String, email : String)
+getName() : String
+setName(name : String) : void
+getSurname() : String
+setSurname(surname : String) : void
+getUserName() : String
+setUserName(userName : String) : void
+getPassword() : String
+setPassword(password : String) : void
+getEmail() : String
+setEmail(email : String) : void
+getPermission() : Permission
+setPermission(permission : Permission) : void
+setId() : void
+getId() : int
+addChat(chat : Chat) : void
+removeChat(chatID : String) : void
+addMessage(chatID : String, mes : Message)
+addNotification(nt : Notification) : bool
+markAsRead(ntID : int) : void
Permission
-groupFormation : bool
-feedBackView : bool
-giveFeed : bool
+getGroupFormation() : bool
+setGroupFormation(groupFormation : bool) : void
+getFeedBackView() : bool
+setFeedBackView(feedBackView : bool) : void
+getGiveFeed() : bool
+setGiveFeed(giveFeed : bool) : void
1
1..*
Notification
-message : String
-sender : int
-reciever : int
-date : Date
-id : String
-isRead : bool
+Notification()
+markAsRead() : void
1..*
Chat
-members : ArrayList<Integer>
-chatTitle : String
-messages : ArrayList<Message>
-chatID : String
+Chat(title : String, members : ArrayList<Integer>, id : String)
+getMembers() : ArrayList<Integer>
+setMembers(members : ArrayList<Integer>) : void
+getChatTitle() : String
+setChatTitle(chatTitle : String) : void
+addMember(id : int) : bool
+removeMember(id : int) : bool
+addMessage(mes : Message) : bool
+removeAllMessages() : void
+setChatID(name : String) : void
+getChatID() : String
1..*
Message
-send : String
-receiver : String
-message : String
-date : Date
+getSend() : String
+setSend(send : String) : void
+getReceiver() : String
+setReceiver(receiver : String) : void
+getMessage() : String
+setMessage(message : String) : void
+getDare() : Date
+setDare(date : Date) : void
+Message()
0..*
Instructor
-sections : ArrayList<Integer>
-listTA : ArrayList<Integer>
+Instructor()
+setListTA(listTA : ArrayList<Integer>) : void
+getSections() : ArrayList<int>
+setSections(sections : ArrayList<Integer>) : void
+getListTA() : ArrayList<int>
+addTA(taID : int) : void
+removeTA(taID : int) : bool
+addSection(section : Section) : void
+removeSection(sectionID : int) : void
+createPeerFeedback() : void
+viewPeerFeedback() : ArrayList<FeedBack>
+viewProjectFeedback() : ArrayList<Feedback>
+giveProjectFeedback() : bool
+viewArtifactFeedBack()
+sendNotification() : void
+createCourseFeedBack() : void
+viewCourseFeedback() : void
+viewNotification()
+createRandomGroups() : void
+create GroupFormation() : void
Student
-section : int
-groupID : String
+Student()
+getSection() : int
+setSection(section : int) : void
+getGroupID() : String
+setGroupID(groupID : String) : void
+assignGroup() : bool
+removeGroup() : bool
+getGivenFeedback() : ArrayList<FeedBack>
+getTakenFeedback() : ArrayList<Feedback>
+viewArtifactFeedBack()
+giveProjectFeedback() : bool
+givePeerFeedBack() : void
+createCourseFeedBack() : void
Assignment
-name : String
-dueDate : Date
-description : String
-grade : double
-feedback : Project_FeedBack
+Assignment()
+getName() : String
+setName(name : String) : void
+getDueDate() : Date
+setDueDate(dueDate : Date) : void
+getDescription() : String
+setDescription(description : String) : void
+getGrade() : double
+setGrade(grade : double) : void
+getFeedBack(id : String) : Project_Feedback
+setProjectFeedBack(fd : Project_Feedback) : void
Course
-sections : Arraylist<Integer>
-name : String
-courseID : String
+Course()
+getSections() : ArrayList<Integer>
+setSections(sections : ArrayList<Integer>)
+getName() : String
+setName(name : String) : void
+getCourse() : String
+setCourseID(id : String) : void
TA
-groups : ArrayList<String>
-sections : ArrayList<Integer>
+TA()
+getGroups() : ArrayList<String>
+setGroups(groups : ArrayList<String>) : void
+giveProjectFeedback() : bool
+viewArtifactFeedBack() : Feedback
+viewProjectFeedback() : ArrayList<Feedback>
+addSection(section : int) : void
+removeSection(section : int) : void
+sendNotification() : void
1..*
1
1..*
Group
-students : ArrayList<Integer>
-groupID : String
-groupLeader : int
-peerReviews : ArrayList<Peer_Feedback>
+getStudents() : ArrayList<Integer>
+setStudents(students : ArrayList<Integer>) : void
+getGroupID() : String
+setGroupID(groupID : String) : void
+getGroupLeader() : int
+setGroupLeader(groupLeader : int) : void
+getPeerReviews() : ArrayList<Peer_Feedback>
+setPeerReviews(peerReviews : ArrayList<Peer_Feedback>) : void
+setPeerReview() : void
+inviteMember() : int
+addMember() : int
+removeMember() : void
+addPeerReview() : bool
+addFeedback() : bool
+getPeerReview() : Peer_Feedback
+getFeedbacks() : ArrayList<Feedback>
+addRate(rate : int) : bool
+getRate() : int
1..*
1..*
Section
-groups : ArrayList<String>
-instructor : Instructor
-courseReviews : ArrayList<Course_Feedback>
-section : int
+Section(id : int)
+getGroups() : ArrayList<String>
+setGroups(groups : ArrayList<String>) : void
+getInstructor() : Instructor
+setInstructor(instructor : Instructor) : void
+getCourseReviews() : ArrayList<Course_Feedback>
+setCourseReviews(CourseReviews : ArrayList<Course_Feedback>) : void
+getGroup(id : int) : Group
+addCourserReview() : bool
+getCourseReview(index : int) : Course_Feedback
+setSection(section : int) : void
+getSection() : int
1..*
Project
-assignments : ArrayList<Assignment>
-group : Group
-description : String
-artifactReview : Artifact_Review
+getAssignments() : ArrayList<Assignment>
+setAssignments(assignments : ArrayList<Assignment>) : void
+getGroup() : Group
+setGroup(group : Group) : void
+getDescription() : String
+setDescription(description : String) : void
+Project()
+giveRate(rate : int) : bool
+getRate() : int
1
0..*
Course_Feedback
-feedbackID : String
+Course_Feedback()
+getFeedbackID() : String
+setFeedbackID(feedbackID : String) : void
1..*
Project_Feedback
-file : ArrayList<File>
+Project_Feedback()
+getFiles() : ArrayList<File>
+addFile(file : File) : bool
Question
-question : String
+getQuestion() : String
+setQuestion(question : String) : ...
+Question()
1..*
Point_Question
-outOf : int
-answer : int
+getAnswer() : int
+setAnswer(answer : int) : void
+getOutOf() : int
+setOutOf(outOf : int) : void
+Point_Question(outOf : int)
-control() : bool
Open_Ended_Question
-answer : String
+getAnswer() : String
+setAnswer(answer : String) : void
+Open_Ended_Question(answer : String)
Multiple_Choice_Question
-choice : ArrayList<String>
-choiceNumber : int
+getChoice() : ArrayList<String>
+setChoice(choice : ArrayList<String>) : void
+getChoiceNumber() : int
+setChoiceNumber(choiceNumber : int) : void
+Multiple_Choice_Question(choiceNumber : int)
1
Artifact_Review
-rate : int
-rateDistribution : Map<Integer, Integer>
-listWhoRated : ArrayList<Integer>
+Artifact_Review()
+checkPerson(id : int)
+addPerson(id : int)
+addRate(rate : int)
+calculateAverage()
+getRate()
+setRate(new_rate : double) : void
+getArtifactReviewID() : String
+setArtifactReviewID(id : String) : void
FeedBack
-questions : ArrayList<Question>
-giver : int
-receiver : int
-feedbackID : String
+Feedback(giver : int, receiver : int)
+FeedBack()
+getQuestions() : ArrayList<Question>
+setQuestions(questions : ArrayList<Question>) : void
+getGiver() : int
+setGiver(giver : int) : void
+getReceiver() : int
+setReceiver(receiver : int) : void
+addQuestion(question : Question, index : int)
+setFeedBackID(send : int, recieve : int) : void
+getFeedBackID() : String
1..*
Peer_Feedback
+Peer_Feedback(feedback : int)
+Peer_Feedback()
+assignFeedBack(feedback : int) : void
+getFeadbackID() : String
+setFeadbackID(feadbackID : int) : void

## 4.3. Packages

We will use two kinds of packages in our implementation. First packages that are defined by us which content can be seen in above section 4.1 and some external packages that will be used for various implementations.

### 4.3.1. Packages created by Developers

#### 4.3.1.1. Controller Package

This package includes manager classes and controllers, this package interacts with all other packages. For instance the Input Manager which interacts with interface and DataController interact with Database are included in this package.

#### 4.3.1.2. Entity Package

This package contains all of the entity objects that can be found in our system such as Student Class or Assignment class . This package only interacts with the controller package's classes.

#### 4.3.1.3. Database Package

The Database package holds a database class which contains a list of students, peer reviews, assignment files, messages  and all other data that must be stored. This package also interacts with just the controller.

#### 4.3.1.4. Interface Package

The Interface package interacts with the controller. Due to the fact that we use CSS, HTML and JavaScript for the user interface of our project this package is initially empty.

### 4.3.2 External Packages

#### 4.3.2.1. java.util

Java.util packages contain the list, date or calendar classes which will be used in the project. For example, this package will be used for creating ArrayList which will hold users of the program. This package contains a scanner class which we can use for testing our code (giving input and testing the code). Furthermore, this package contains the Date class which will be used in our program for specifying chat messages dates etc [5].

#### 4.3.2.2 java.io

Java.io packages contain a file class which will be used in the project. This class will be used to implement the attach file feature in the project [5].

## 4.4. Class Interfaces

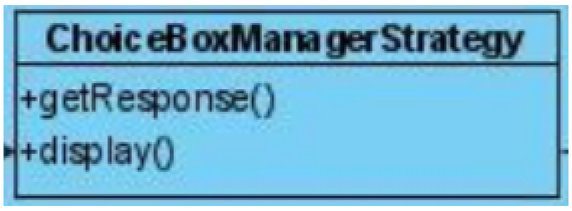
### 4.4.1. Controller Package Classes

#### 4.4.1.1. ButtonManagerStrategy

ButtonManagerStrategy
+getResponse()
+display()

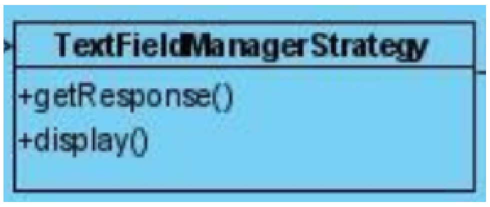
**Operations:**
**public bool getResponse ( ) :** Check if there is a response on button or not.
**public void display ():** Display the response of the button depending on the button's function.

#### 4.4.1.2. ChoiceBoxManagerStrategy



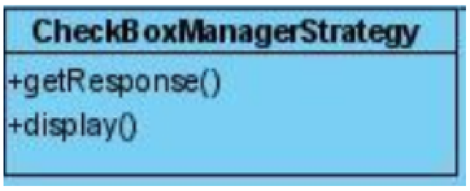
**Operations:**
public int getResponse ( ) : This method for getting the response choice from the question.
public void display (): Display the response of the choicebox. For example, displaying a message to the user that the choice is successfully chosen.

### 4.4.1.3. TextField Manager Strategy



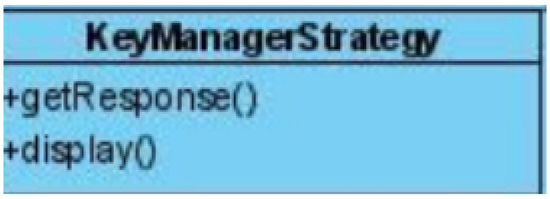
Operations:
public String getResponse ( ) : This method gets the response text in the textfield.
public void display (): Display the response text.

### 4.4.1.4. CheckBoxManagerStrategy



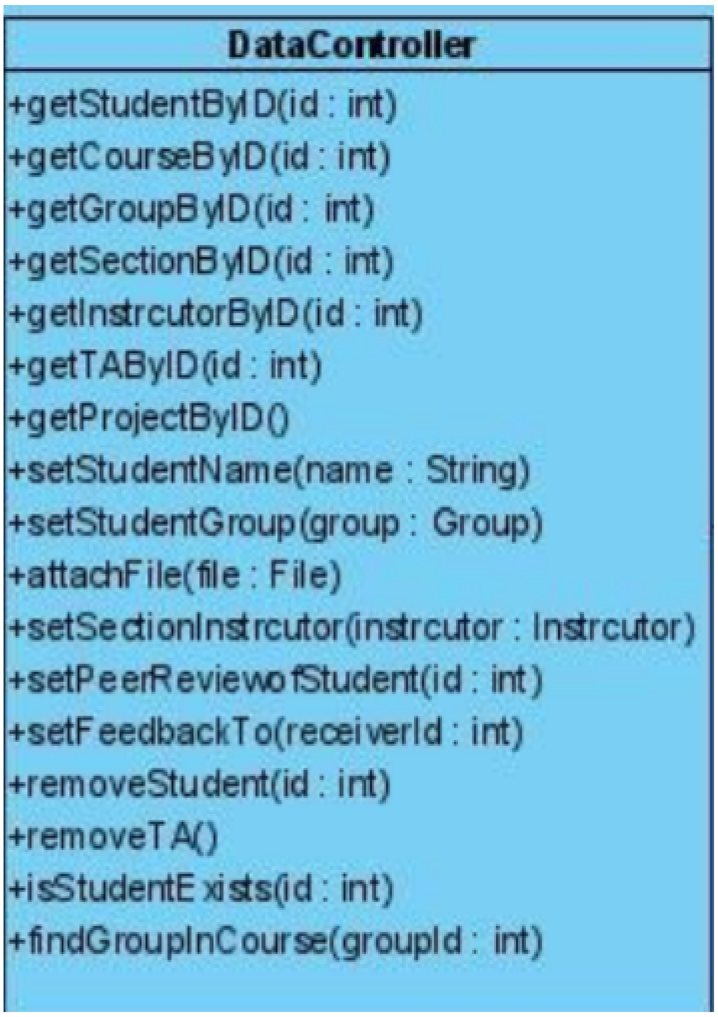
**Operations:**
public bool getResponse ( ) : Check if there is a response on box or not.
public void display (): Display the response.

### 4.4.1.5. Key Manager Strategy



**Operations:**
public bool getResponse ( ) : Check if there is a response on key or not.
public void display (): Display the response.

### 4.4.1.6. DataController


DataController
+getStudentByID(id : int)
+getCourseByID(id : int)
+getGroupByID(id : int)
+getSectionByID(id : int)
+getInstrcutorByID(id : int)
+getTAByID(id : int)
+getProjectByID()
+setStudentName(name : String)
+setStudentGroup(group : Group)
+attachFile(file : File)
+setSectionInstrcutor(instrcutor : Instrcutor)
+setPeerReviewofStudent(id : int)
+setFeedbackTo(receiverId : int)
+removeStudent(id : int)
+removeTA()
+isStudentExists(id : int)
+findGroupInCourse(groupId : int)

**Operations:**
**public Student getStudentByID( int id) :** Request the student data from the database by it’s ID.
**public Course getCourseByID(int id):** Request the course data from the database by it’s ID.
**public Group getGroupByID(int id):** Request the group data from the database by it’s ID.
**public Section getSectionByID(int id):** Request the section data from the database by it’s ID.
**public Instructor getInstructorByID(int id):** Request the instructor data from the database by it’s ID.
**public TA getTAByID(int id):** Request the teaching assistant data from the database by it’s ID.

**public Project getProjectByID(int id):** Request the project data from the database by it's ID.
**public void setStudentName (Student student,String name):** Set the student name in the database.
**public void setStudentGroup (Group group):** This method use for setting the new group into database
**public void attachFile( int groupId, File file):** This method is used for adding attachment files into database under specified groups.
**public void setSectionInstructor( instructor instructor):** This method is used for setting section's instructor.
**public void setPeerReview of Student (int id):** This method is used for setting peer review of specific students .
**public void setFeedbackTo (int receiveID) :**This method is used for setting feedback of specific students .
**public void removeStudent (int id):** This method for removing a student from the system. For example, if the student withdraws from the course.
**public void removeTA (int id):**This method for removing teaching assistants, if some issue happens.
**public void isStudetExist (int id):** This method for checking if a specific student is found in the system.
**public Group findGroupInCourse(int groupId):** This method for finding a group in the system. For example, to reach the group's project.

## 4.4.2 Database Package Classes

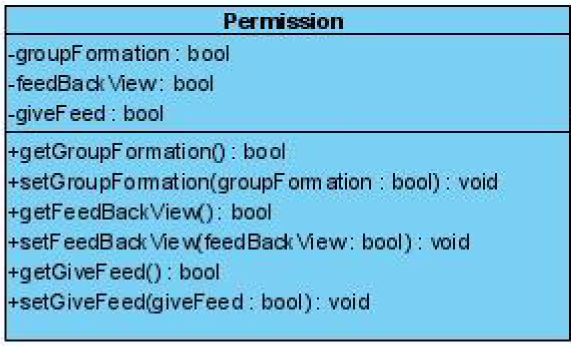
### 4.4.2.1. Database Class

No attribute of operation defined yet. This is the database of our system that contains all of the information about students, groups, instructors etc.

## 4.4.3 Entity Package Classes

### 4.4.3.1. Permission:

The Permission Class is used to determine which tasks 3 types of users registered in the application will be assigned (Student, Instructor, or TA).


Permission
-groupFormation : bool
-feedBackView: bool
-giveFeed : bool
+getGroupFormation() : bool
+setGroupFormation(groupFormation : bool) : void
+getFeedBackView() : bool
+setFeedBackView(feedBackView: bool) : void
+getGiveFeed() : bool
+setGiveFeed(giveFeed : bool) : void

**Attributes:**

**boolean groupFormation:** Permission for activate groupFormation period

**boolean feedBackView:** Permission for view feedback

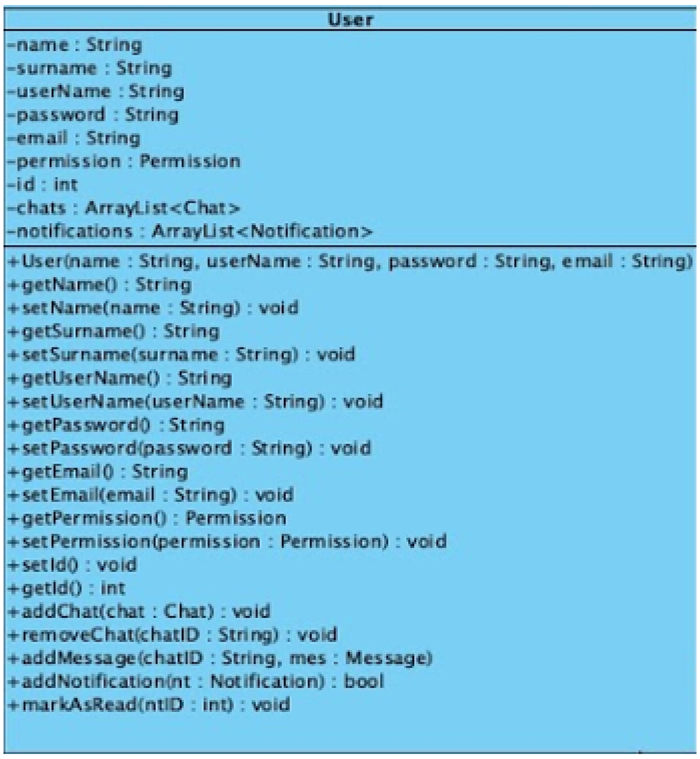
**boolean giveFeedback:** Permission for give feedback

**Operations:**

**set and get methods of the attributes:** Methods for interact private attributes

### 4.4.3.2. User:

In the application, the User Class makes users register or access their own pages after registration. According to the permission class, users can be of 3 types, student, TA, or instructor.


User
-name : String
-surname : String
-userName : String
-password : String
-email : String
-permission : Permission
-id : int
-chats : ArrayList<Chat>
-notifications : ArrayList<Notification>
+User(name : String, userName : String, password : String, email : String)
+getName() : String
+setName(name : String) : void
+getSurname() : String
+setSurname(surname : String) : void
+getUserName() : String
+setUserName(userName : String) : void
+getPassword() : String
+setPassword(password : String) : void
+getEmail() : String
+setEmail(email : String) : void
+getPermission() : Permission
+setPermission(permission : Permission) : void
+setId() : void
+getId() : int
+addChat(chat : Chat) : void
+removeChat(chatID : String) : void
+addMessage(chatID : String, mes : Message)
+addNotification(nt : Notification) : bool
+markAsRead(ntID : int) : void

**Attributes:**

**String name:** Name of the user.

**String surname:**Surname of the user.

**String userName:**UserName of the user.

**String password:**Password  of the user.

**String email:**Email of the user.

**Permission: permission:** Permission object that determines the permissions of the user.

**int id:**ID of the user.

**ArrayList<Chat> chats:** Chats that user is part of.

**ArrayList<Notifications> notifications:** Notifications that concern the user<.

**Operations:**

**set and get methods of the attributes:** Methods for interact private attributes

**User( String userName, String password, String email):** Constructor of the user.

**void addChat(Chat chat):** Method for adding a new chat object to the chat list of the user.

**removeMessage(Chat chat)**: Removing specific messages from the chat object of the user.

**addMessage(String chatID, Message mes):** Method for adding new messages to the chat object of the user.

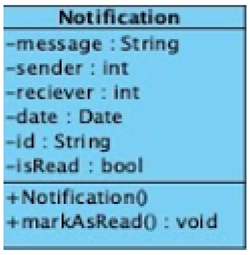
**void removeChat( String chatID)**: Removing specific chat from the chat list of the user.

**bool addNotification(Notification nt)**: Adding new notification on different issues.

**void markAsRead(int ntID)**: Marking different messages as read.

**4.4.3.3 Notification:**

The Notification Class enables various notifications to be created and sent. It will also store the message.



**Attributes:**

**String message:** Notification message text.

**int sender:**User Id of the sender user.

**int receiver:**User Id of the receiver user.

**Date date:**Date of the notification sent.

**String id :** Id of notification sent.

**Bool isRead:** Notice of notification is read or not.

**Operations:**

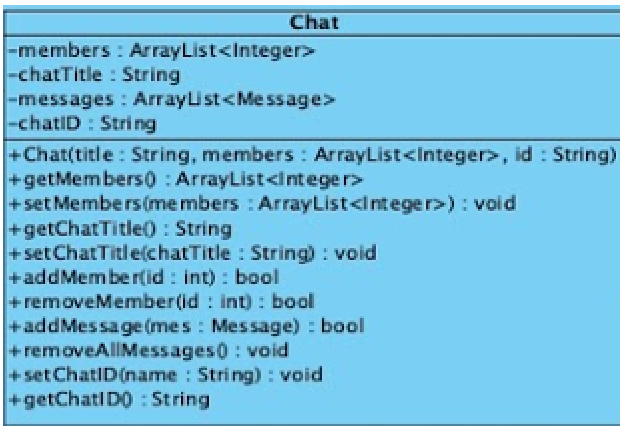
**Notification():** Constructor of the Notification class.

**void markAsRead():** Marking a specific notification as read.

**Notification():**Constructor of the notification object.

**4.4.3.4 Chat:**

Chat Class allows users in the application to message with each other or to send and receive files with each other.



```
Chat
-members : ArrayList<Integer>
-chatTitle : String
-messages : ArrayList<Message>
-chatID : String
+Chat(title : String, members : ArrayList<Integer>, id : String)
+getMembers() : ArrayList<Integer>
+setMembers(members : ArrayList<Integer>) : void
+getChatTitle() : String
+setChatTitle(chatTitle : String) : void
+addMember(id : int) : bool
+removeMember(id : int) : bool
+addMessage(mes : Message) : bool
+removeAllMessages() : void
+setChatID(name : String) : void
+getChatID() : String
```

**Attributes:**

**ArrayList<int> members:**Users id  that participate in the chat

**String chatTitle:** Chat title.

**ArrayList<Messages> messages:** User messages in the chat.

**String chatID:** Id of specific chat.

**Operations:**

**Chat(String title, ArrayList<Integer> members, String Id):**

Constructor of the Chat class.

**set and get methods of the attributes:** Methods for interact private attributes

**bool addMember(int id):** Method for adding new participants.

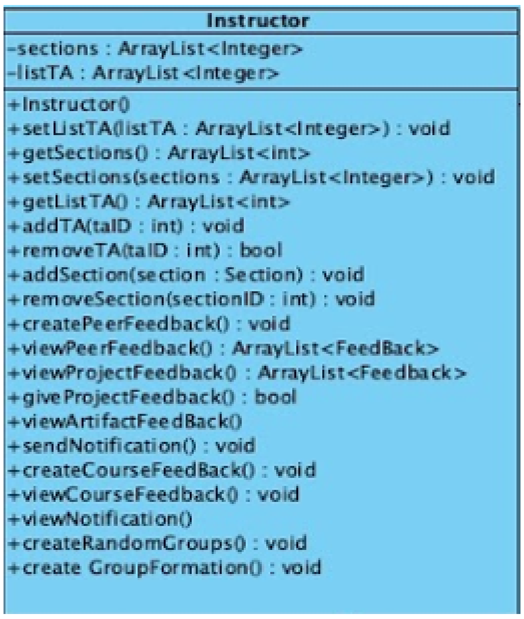
**bool removeMember(int id):**Method for removing participants.

**bool addMessage(Message mes):** Method for adding a new message to the chat.

**void removeAllMessages():** Removing all messages in the chat screen.

### 4.4.3.5 Instructor:

The Instructor Class is specific to the instructor and allows the instructor-related work to be done.



```
Instructor
-sections : ArrayList<Integer>
-listTA : ArrayList <Integer>
+Instructor()
+setListTA(listTA : ArrayList<Integer>) : void
+getSections() : ArrayList<int>
+setSections(sections : ArrayList<Integer>) : void
+getListTA() : ArrayList<int>
+addTA(taID : int) : void
+removeTA(taID : int) : bool
+addSection(section : Section) : void
+removeSection(sectionID : int) : void
+createPeerFeedback() : void
+viewPeerFeedback() : ArrayList<FeedBack>
+viewProjectFeedback() : ArrayList<Feedback>
+giveProjectFeedback() : bool
+viewArtifactFeedBack()
+sendNotification() : void
+createCourseFeedBack() : void
+viewCourseFeedback() : void
+viewNotification()
+createRandomGroups() : void
+create GroupFormation() : void
```

**Attributes:**

**ArrayList<Integer> sections:** the sections of the instructor

**ArrayList<Integer> listTA:** the list of the TAs of the section

**Operations:**

**set and get methods of the attributes:** Methods for interact private attributes

**Instructor():** the constructor of the class

**void addTA(int taId):** Method for assigning TA to an instructor

**boolean removeTA(int taId):** Method for removing a TA from an instructor

**void addSection(Section section):** Method for adding a section to an instructor

**void removeSection(int sectionId):** Method for removing a section from an instructor

**void createPeerFeedback():** Method for creating a peer feedback form for the student which can be done by an instructor

**ArrayList<Feedback> viewPeerFeedback():** Method for viewing a peer feedback

**ArrayList<Feedback> viewProjectFeedback():** Method for viewing a project feedback

**boolean giveProjectFeedback():** Method for giving a project feedback

**viewArtifactFeedback():** Method for viewing a artifact feedback

**void sendNotification():** Method for sending notifications

**void createCourseFeedback():** Method for creating a course feedback form

**void viewCourseFeedback():** Method for viewing a course feedback

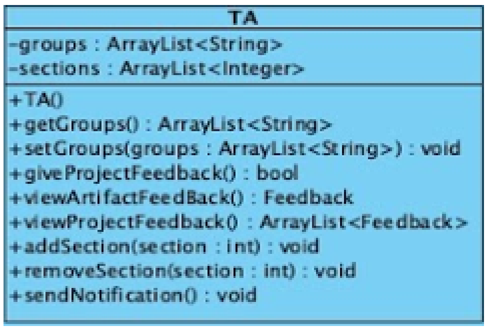
**void viewNotification():** Method for viewing a notification

**void createRandomGroups():** Method for creating groups with random distribution

**void createGroupFormation():** Method for starting group formation process

### 4.4.3.6 TA:

The TA Class is specific to the instructor and allows the instructor-related work to be done.


TA
-groups : ArrayList<String>
-sections : ArrayList<Integer>
+TA()
+getGroups() : ArrayList<String>
+setGroups(groups : ArrayList<String>) : void
+giveProjectFeedback() : bool
+viewArtifactFeedBack() : Feedback
+viewProjectFeedback() : ArrayList<Feedback>
+addSection(section : int) : void
+removeSection(section : int) : void
+sendNotification() : void

**Attributes:**

**ArrayList<String> groups:** Group names that teaching assistant is responsible for.

**ArrayList<Integer> sections:** Sections names that teaching assistant is participating.

**Operations:**

**set and get methods of the attributes:** Methods for interact private attributes

**bool giveProjectFeedBack():** Method for TA to give feedback to the projects

**Feedback viewArtifactFeedBack ():** Method for TA to view artifact feedback of the project.

**ArrayList<Feedbacks> void viewProjectFeedBack():** Method for viewing a project feedback

**void addSection(int section):** Method for adding a section to TA

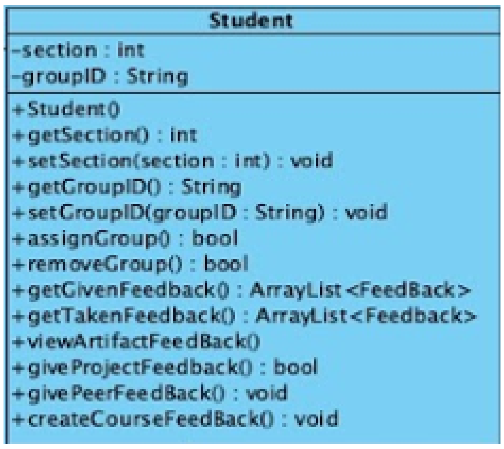
**void removeSection(int section):** Method for removing a section from a TA

**TA():** Constructor of TA class

**void sendNotifications():** Method for sending notifications

### 4.4.3.7 Student:

The Student Class is specific to the instructor and allows the instructor-related work to be done.


Student
-section : int
-groupID : String
+Student()
+getSection() : int
+setSection(section : int) : void
+getGroupID() : String
+setGroupID(groupID : String) : void
+assignGroup() : bool
+removeGroup() : bool
+getGivenFeedback() : ArrayList<FeedBack>
+getTakenFeedback() : ArrayList<Feedback>
+viewArtifactFeedBack()
+giveProjectFeedback() : bool
+givePeerFeedBack() : void
+createCourseFeedBack() : void

**Attributes:**

**int section:** Section id of the student

**String groupID:** Group id of the student

**Operations:**

**Student():** Constructor of the Student class

**set and get methods of the attributes:** Methods for interact private attributes

**boolean assignGroup():** Method for assigning a group to a student

**boolean removeGroup():** Method for removing a group from a student

**ArrayList<Feedback> getGivenFeedback():** Method for getting the feedback of this student as its' receiver

**ArrayList<Feedback> getTakenFeedback():** Method for getting the feedback of this student as its' sender

**void viewArtifactFeedback():** Method for viewing artifact feedback whose owner is the student

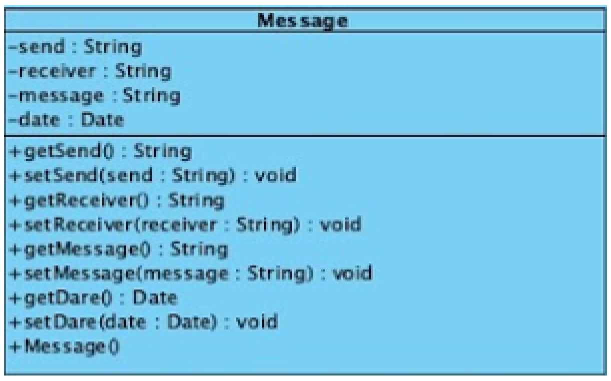
**boolean giveProjectFeedback():** Method for giving project feedback to a student

**void givePeerFeedback():** Method for giving peer feedback

**void createCourseFeedback():** Method for creating course feedback

**4.4.3.8 Message:**

Message Class shows the properties of the messaging channel in the Chat class.



**Attributes:**

**String send:** The name of the sender of the message.

**String receiver:** The name of the receiver of the message.

**String message:** The message’s text prompt.

**Date date:** The sending date of the message.

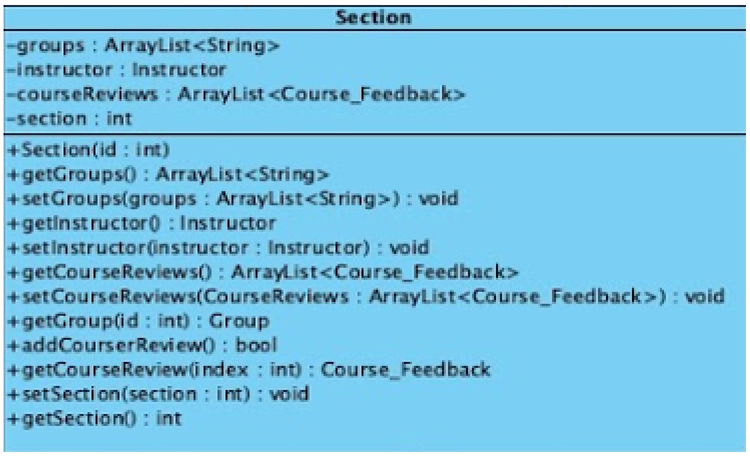
**Operations:**

**set and get methods of the attributes:** Methods for interact private attributes

**Message():** Constructor of message class.

**4.4.3.9 Section:**

Section Class is used to determine the various sections determined by the instructor and their various features.



```
Section
-groups : ArrayList<String>
-instructor : Instructor
-courseReviews : ArrayList <Course_Feedback>
-section : int
+Section(id : int)
+getGroups() : ArrayList<String>
+setGroups(groups : ArrayList<String>) : void
+getInstructor() : Instructor
+setInstructor(instructor : Instructor) : void
+getCourseReviews() : ArrayList<Course_Feedback>
+setCourseReviews(CourseReviews : ArrayList<Course_Feedback>) : void
+getGroup(id : int) : Group
+addCourserReview() : bool
+getCourseReview(index : int) : Course_Feedback
+setSection(section : int) : void
+getSection() : int
```

**Attributes:**

**ArrayList<String> groups:** The list that holds every group in the section.

**Instructor instructor:** The instructor of the section.

**ArrayList<Course_Feedback> CourseReviews:** The list that holds every course reviews that is filled by the students of the section.

**int section:** Id of the specific section.

**Operations:**

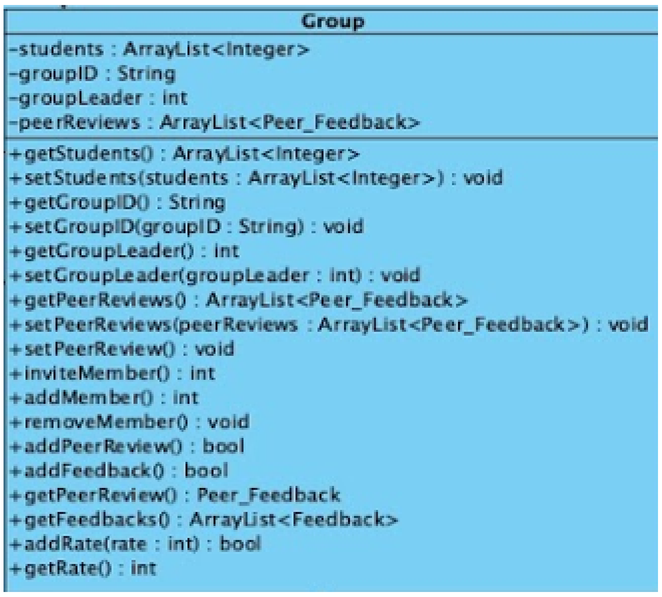
**set and get methods of the attributes:** Methods for interact private attributes

**Section(int id):** Constructor of section class.

**bool addCourserReview()**: The method for adding the course review to the section and returning true if successful.

### 4.4.3.10 Group:

Group Class allows the various students to perform tasks related to the groups they have and the characteristics of these groups.



```
Group
-students : ArrayList<Integer>
-groupID : String
-groupLeader : int
-peerReviews : ArrayList<Peer_Feedback>
+getStudents() : ArrayList<Integer>
+setStudents(students : ArrayList<Integer>) : void
+getGroupID() : String
+setGroupID(groupID : String) : void
+getGroupLeader() : int
+setGroupLeader(groupLeader : int) : void
+getPeerReviews() : ArrayList<Peer_Feedback>
+setPeerReviews(peerReviews : ArrayList<Peer_Feedback>) : void
+setPeerReview() : void
+inviteMember() : int
+addMember() : int
+removeMember() : void
+addPeerReview() : bool
+addFeedback() : bool
+getPeerReview() : Peer_Feedback
+getFeedbacks() : ArrayList<Feedback>
+addRate(rate : int) : bool
+getRate() : int
```

**Attributes:**

**ArrayList<Integer> students:** The list of the students id.

**String groupId:** The id of the group.

**int groupLeader:** The student id of the groupLeader.

**ArrayList<Peer_Feedback> peerReviews:** The list of the peer reviews that are filled by the students.

**Operations:**

**set and get methods of the attributes:** Methods for interact private attributes

**int inviteMember():** Invites a specified student to the group

**int addMember():** Adds a member to the group.

**void removeMember():** Removes the member from the group.

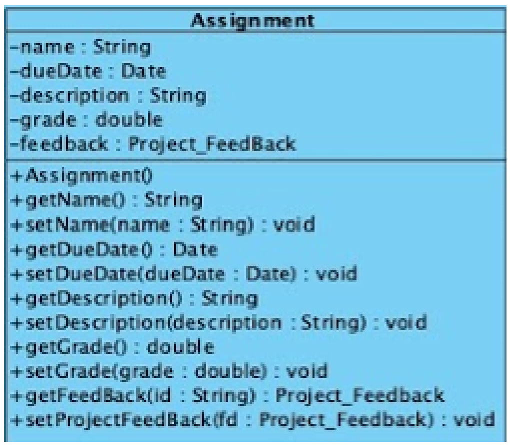
**boolean addPeerReview():** Adds a peer review to the group.

**boolean addFeedback():** Adds a feedback to the group.

**boolean addRate(int rate):** Adds a rate to the group.

### 4.4.3.11 Assignment:

The Assignment Class helps the instructor to set tasks or project-related responsibilities to students or various groups.



```
Assignment
-name : String
-dueDate : Date
-description : String
-grade : double
-feedback : Project_FeedBack
+Assignment()
+getName() : String
+setName(name : String) : void
+getDueDate() : Date
+setDueDate(dueDate : Date) : void
+getDescription() : String
+setDescription(description : String) : void
+getGrade() : double
+setGrade(grade : double) : void
+getFeedBack(id : String) : Project_Feedback
+setProjectFeedBack(fd : Project_Feedback) : void
```

**Attributes:**

**String name:** Name of the assignment

**Date dueDate:** Due date of the assignment

**String description:** Description of the assignment

**double grade:** Grade that assignment take

**Project_FeedBack feedback :** Feedback that assignment take

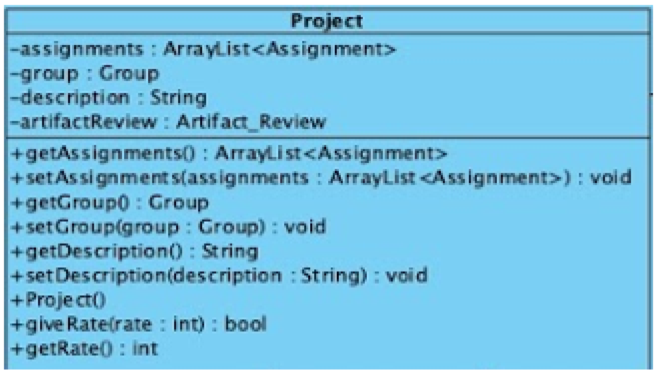
**Operations:**

**Assignment():** Constructor of the assignment object.

**set and get methods of the attributes:** Methods for interact private attributes

### 4.4.3.12 Project:

Project class determines the properties of projects owned by various groups.


Project
-assignments : ArrayList<Assignment>
-group : Group
-description : String
-artifactReview : Artifact_Review
+getAssignments() : ArrayList<Assignment>
+setAssignments(assignments : ArrayList<Assignment>) : void
+getGroup() : Group
+setGroup(group : Group) : void
+getDescription() : String
+setDescription(description : String) : void
+Project()
+giveRate(rate : int) : bool
+getRate() : int

**Attributes:**

**ArrayList<Assignment> assignments:** Assignment list that one project holds.

**Group group:** Group that contains participants that participate in the project

**Artifact_Review artifactReview:** Artifact review that project takes

**String description:** the description of the project

**Operations:**

**set and get methods of the attributes:** Methods for interact private attributes

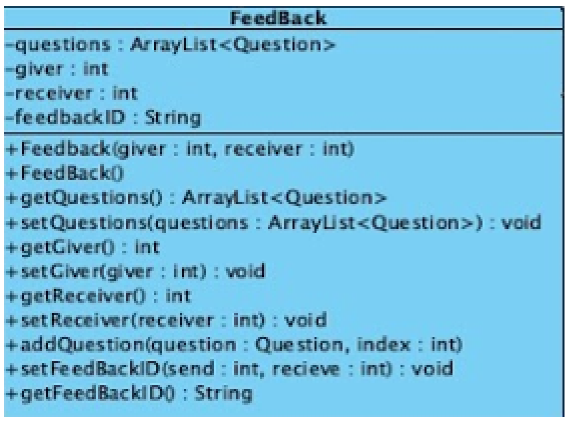
**boolean giveRate(int rate):** Method for rating the project

**Project():** the constructor of the Project class

**int getRate():** Method for getting rate of the group class

### 4.4.3.13 FeedBack:

FeedBack Class enables the instructor to give feedback to various groups about their work and tasks. This Class is divided into 3 separate private classes.


FeedBack
-questions : ArrayList<Question>
-giver : int
-receiver : int
-feedbackID : String
+Feedback(giver : int, receiver : int)
+FeedBack()
+getQuestions() : ArrayList<Question>
+setQuestions(questions : ArrayList<Question>) : void
+getGiver() : int
+setGiver(giver : int) : void
+getReceiver() : int
+setReceiver(receiver : int) : void
+addQuestion(question : Question, index : int)
+setFeedBackID(send : int, recieve : int) : void
+getFeedBackID() : String

**Attributes:**

**ArrayList<Question> questions :** Questions that include in the feedback form.

**int giver:** Id of the user that gives the feedback.

**int receiver:** Id of the receiver.

**Operations:**

**Feedback( int giver, int receiver):** Constructor for the feedback class.

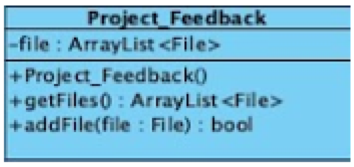
**Feedback()**: Default Constructor for the feedback class

**set and get methods of the attributes:** Methods for interact private attributes

**addQuestion(Question question, int index):** Method for adding a question to the feedback

### 4.4.3.14 Project_Feedback:

Project_Feedback Class contains feedback related to the projects owned by the groups.



**Attributes:**

**ArrayList<File> file:** the files in the project feedback

**Operations:**

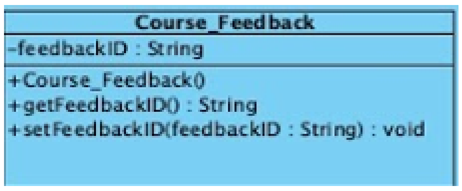
**Project_Feedback():** the constructor of the class

**ArrayList<File> getFiles():** Method for getting the files of the feedback

**boolean addFile(File file):** Method for adding a file to the feedback

**4.4.3.15 Course_Feedback:**

Course_Feedback Class contains feedback related to the given courses from the instructor.


Course_Feedback
-feedbackID : String
+Course_Feedback()
+getFeedbackID() : String
+setFeedbackID(feedbackID : String) : void

**Attributes:**

**String feedbackID:** the id of the feedback

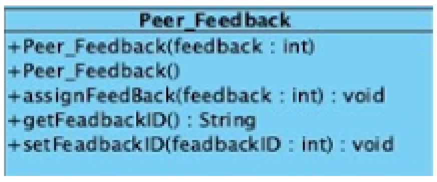
**Operations:**

**Course_Feedback():** the constructor of the class

**set and get methods of the attributes:** Methods for interact private attributes

**4.4.3.16 Peer_Feedback:**

Peer_Feedback Class contains feedback related to between group members in a particular group.


Peer_Feedback
+Peer_Feedback(feedback : int)
+Peer_Feedback()
+assignFeedBack(feedback : int) : void
+getFeadbackID() : String
+setFeadbackID(feadbackID : int) : void

**Attributes:**

**String feedbackID: the id of the feedback**

**Operations:**

**Peer_Feedback(int feedback):** the constructor of the class

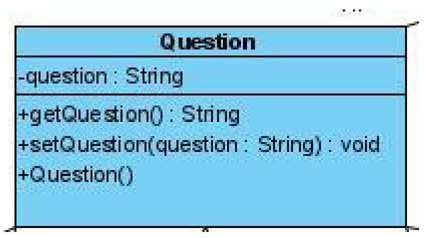
**Peer_Feedback():** the default constructor of the class

**set and get methods of the attributes:** Methods for interact private attributes

**void assignFeedback(int feedback):** Method for assigning peer feedback

**4.4.3.17 Question:**

The Question Class allows setting up the questions to be asked while giving feedback. This Class is divided into 3 separate private classes.



**Attributes:**

**String question:** the string form of the question

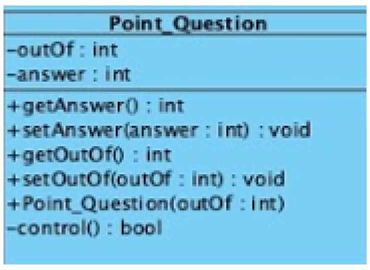
**Operations:**

**set and get methods of the attributes:** Methods for interact private attributes

**Question():** Constructor of the Question class

### 4.4.3.18 Point_Question:

The Point_Question Class allows determining the point question among the questions to be created while giving feedback.



**Attributes:**

**int outOf:** the maximum grade of the answer for example 50 score out of 100 - 100 is the maximum grade-.

**int answer:** the given grade

**Operations:**

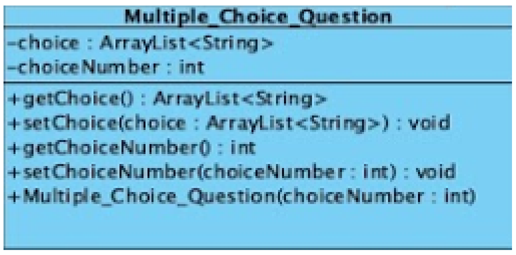
**set and get methods of the attributes:** Methods for interact private attributes

**Point_Question(int outOf):** Constructor of class.

**bool control():** Controlling the point questions.

### 4.4.3.19 Multiple_Choice_Question:

Multiple_Choice_Question Class allows determining multiple-choice questions among the questions to be created while giving feedback.


Multiple_Choice_Question
-choice : ArrayList<String>
-choiceNumber : int
+getChoice() : ArrayList<String>
+setChoice(choice : ArrayList<String>) : void
+getChoiceNumber() : int
+setChoiceNumber(choiceNumber : int) : void
+Multiple_Choice_Question(choiceNumber : int)

**Attributes:**

**ArrayList<String> choice:** the choices of the question

**int choiceNumber:** the number of choices

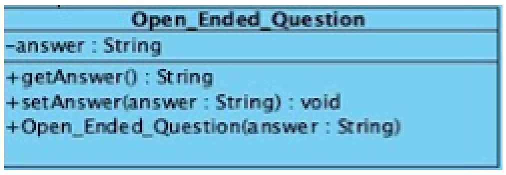
**Operations:**

**set and get methods of the attributes:** Methods for interact private attributes

**voidMultiple_Choice_Question(int choiceNumber):** initializing number of multiple choice questions.

### 4.4.3.20 Open_Ended_Question:

Open_Ended_Question Class allows determining open-ended questions among the questions to be created while giving feedback.


Open_Ended_Question
-answer : String
+getAnswer() : String
+setAnswer(answer : String) : void
+Open_Ended_Question(answer : String)

**Attributes:**

**String answer:** the answer of the question

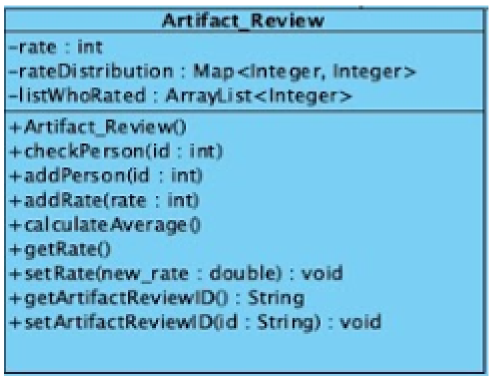
**Operations:**

**set and get methods of the attributes:** Methods for interact private attributes

**Open_Ended_Question(String answer):** the constructor of the class

### 4.4.3.21 Artifact_Review:

The Artifact_Review Class allows various groups to give feedback and various scoring to other groups' projects at the end of the term.


Artifact_Review
-rate : int
-rateDistribution : Map<Integer, Integer>
-listWhoRated : ArrayList<Integer>
+Artifact_Review()
+checkPerson(id : int)
+addPerson(id : int)
+addRate(rate : int)
+calculateAverage()
+getRate()
+setRate(new_rate : double) : void
+getArtifactReviewID() : String
+setArtifactReviewID(id : String) : void

**Attributes:**

**int rate:** the rate(score) of the artifact review

**Map<integer,integer> rateDistribution:** the score distribution of the Artifact Review

**ArrayList<Integer> listWhoRated:** the list of the people who rated the Artifact

**Operations:**

**set and get methods of the attributes:** Methods for interact private attributes

**Artifact_Review():** the constructor of the class

**void checkPerson(int id):** Method for checking the user if she/he rated the artifact

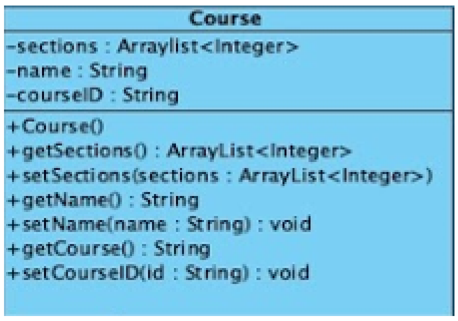
**void addPerson(int id):** Method for adding user as a rater

**void addRate(int rate):** Method for adding rate to an artifact

**void CalculateAvarage():** Calculating average point of artifact review.

### 4.4.3.22 Course:

The Course class is used to determine all of the sections that belong to the course.


Course
-sections : Arraylist<Integer>
-name : String
-courseID : String
+Course()
+getSections() : ArrayList<Integer>
+setSections(sections : ArrayList<Integer>)
+getName() : String
+setName(name : String) : void
+getCourse() : String
+setCourseID(id : String) : void

**Attributes:**

**ArrayList<Integer> sections:** sections of the course

**String name:** the name of the course

**String courseId:** the id of the course

**Operations:**

**set and get methods of the attributes:** Methods for interact private attributes

**Course():** The constructor of the Course class

## 5. References

[1] A. Singh, “Client-Server Architecture: Types Of Client Server Architecture,” *Tech Blog*, 26-Jul-2020. [Online]. Available: https://msatechnosoft.in/blog/types-of-client-server-architecture/. [Accessed: 11-Apr-2021].

[2] The most popular database for modern apps. (n.d.). Retrieved April 11, 2021, from https://www.mongodb.com/

[3] Cox, L. (2020, October 26). Web design 101: How HTML, CSS, and Javascriptwork.RetrievedApril11,2021,from https://blog.hubspot.com/marketing/web-design-html-css-javascript

[4] Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). *Database system concepts*. New York, NY: McGraw-Hill.

[5] Java package - JAVATPOINT. (n.d.). Retrieved April 11, 2021, from https://www.javatpoint.com/package#:~:text=A%20java%20package%20is%20a,io%2C%20util%2C%20sql%20etc.

[6] Brügge, B., & Dutoit, A. H. (2004). *Object oriented software engineering using UML, patterns, and Java*. Upper Saddle River, NJ: Pearson Education.