2020-2021 SPRING SEMESTER

CS319 - Object Oriented Software Engineering

Iteration 1 Peereview Design Report

Group 1B Members

Abdul Razak Daher Khatib	21801340
Muhammad Salman Akhtar Soomro	21701446
Ege Moroğlu	21401240
Utku Gökçen	21703746
Yiğit Dinç	21704275

1. INTRODUCTION

1.1 PURPOSE OF THE SYSTEM

Peereview is a group mates review management system where students can perform several academic activities like enrolling in a course, creating/finding groups, and most importantly evaluate each other. The design is intuitive and simple, it is designed in a minimalistic way where users need the least amount of data to be entered, making it easy for both the instructors and students to familiarize themselves with the user interface while maintaining usability and reliability. With customizable management features, Peerview aims to help instructors and students in achieving the goal evaluating their group mates anonymously throughout the project time.

1.2 DESIGN GOALS

Design is a crucial factor for Peereview as it should be simple and easy to use. Following are the descriptions of the critical design goals.

1.2.1 Criteria

Usability

Peereview does not require any training or instructions before use as the design is very intuitive and minimalist with all the necessary features, thanks to its similarity with other commonly used LMS softwares and its simple user interface.

Performance

Peereview rigorously includes all the necessary features like course enrollment/ creation etc. With some tweaks like group chemistry and evaluation analysis graphs. These features are implemented in a simple way to keep the system fast and easy to load regardless of the device used. The implementation will be simple and minimalist, so the application will take less than 1 second to perform any operation in the system. And will be able to handle a maximum of a 100 online users at the same time, to lower the costs of maintenance. In case the system was used for more courses the number of users can easily be increased by buying more space in the servers to handle more users, since the app will run online.

Extendibility.

In LMS apps like PeeReview, the room for updates and modification is most important as it is designed to maximise user-friendliness and usability. This is why the system was built using the notions of Object Oriented Programming, so that it is easy to extend and modify using various properties of OOP.

Reusability.

Our application is intended for a Bilkent University course, that's said the system would be written in a way which would allow seamless transition to other systems if needed.

1.2.2 Trade-Off:

Memory versus Performance : In order to fully implement the object-oriented system, there will be many objects in the system. Although this increases the memory of the application, the total runtime performance will increase as it will facilitate the reuse of the codes. But, Since the application is written for the web to be used by students, the system needs to be light so that it is easily loaded by all students without worrying about their connection speed. So the memory is preferred over performance.

Understandability versus Functionality : As it is mentioned in the requirement analysis, the whole system must provide the fundamental statistics to review others better. However, adding more numbers and graphs are not the correct answers that increase the understanding. Thus, there exist such challenges that, if too much unrelated information were displayed, the system could become complicated. Hence, the aim to achieve balance in terms of understandability versus functionality stays unsatisfied. To reach the goal of avoiding the functionality of the system being blurred by the excessive use of statistical tools, new ideas are decided to be added after they are compared with each other to test their eligibility and simplicity.

Cost vs. Protabilty

This system is built in a short time, so that it cannot be made completely compatible with mobile phones. However, modern web apps tools allow for a rather smooth experience on phones. Not perfect but the time is tight.

Functionality vs. Robustness

In order to make it light and easy to use we might need to give up on some feature that will result in a less robust product. However, robustness will not be completely neglected, but it will not be preferred over functionality either.

2. SYSTEM ARCHITECTURE

2.1 SUBSYSTEM DECOMPOSITION

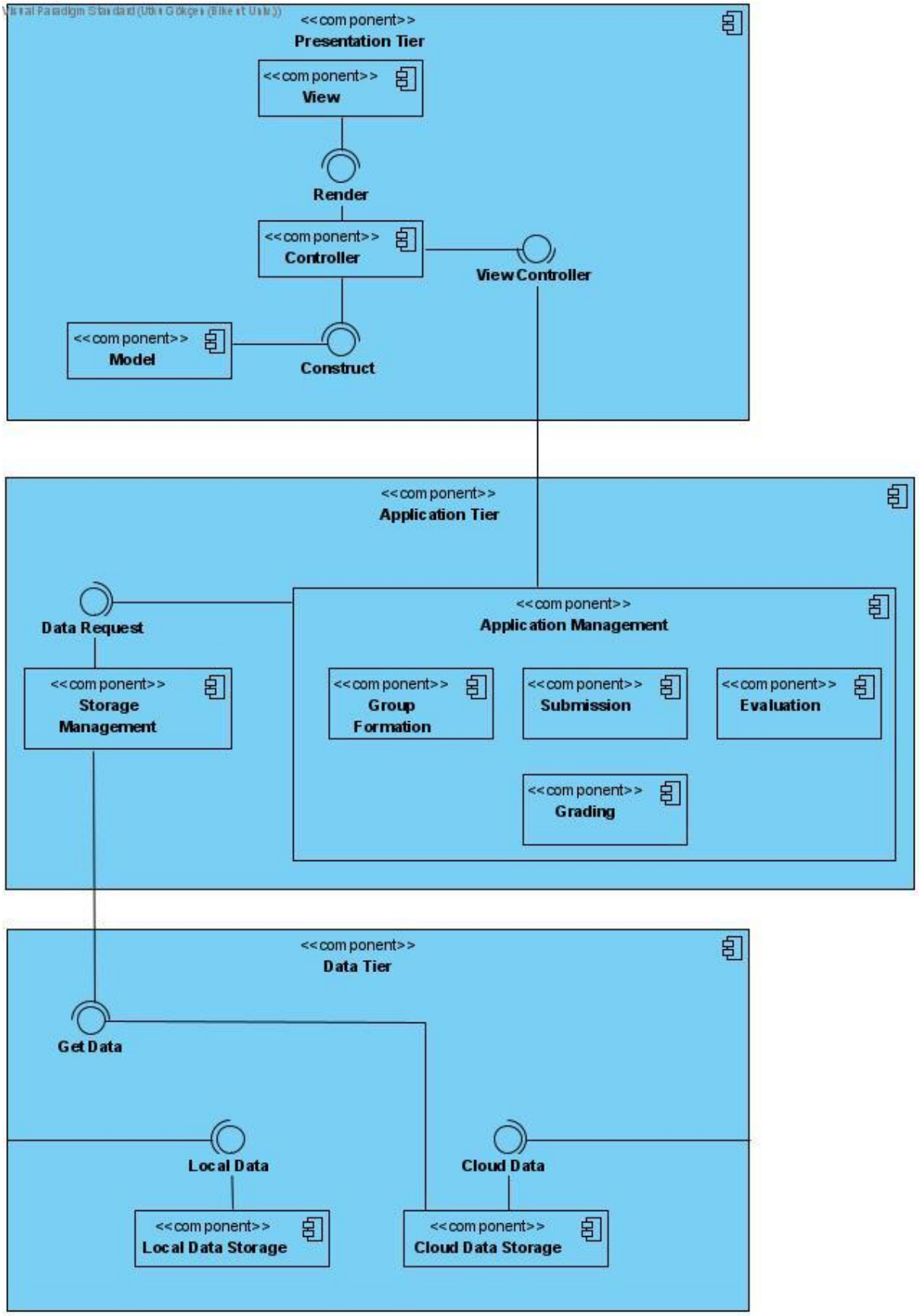


Figure 2.1 Subsystem Decomposition

In accordance with the design goals of our project, we divided the system into 3 different subsystems called Presentation Tier, Application Tier and Data Tier by following the 3-tier architecture logic. By dividing the system into subsystems in this way, we aim to increase the compatibility of the subsystems and minimize the coupling between them. Components of the system in detail are as follows:

● **Presentation Tier:** Presentation tier is the subsystem at the top layer of subsystems where the user interacts with the application. The main function of this subsystem is to transform the processes in the Application Tier into a form that the user can see and interact with.

● **View** is responsible for the interaction of the application with the user.

● **Controller** is responsible for the transfer of data or models to be displayed in the application to the View Component.

● **Model** is responsible for creating the objects to be used in the application.

● **Application Tier:** Application Tier is responsible for coordinating the operations performed within the application. This layer allows data to be retrieved from the Data Tier, processed and then sent to the Presentation Tier.

● **Application Management** is responsible for handling the requests from the user and responding back to the Presentation Tier. It contains the components of the operations that can be performed within the application such as Submission, Evaluation, Group Formation, Grading.

● **Storage management** is responsible for the transfer of data between the Data Tier and the Application Tier.

● **Data Tier:** Data Tier is responsible for storing data both in the user's local and in the cloud. In this layer, the data requested to be processed is sent to the Application Tier. The main function of the layer is to provide a secure access to the database, to protect the privacy of the data and to transfer the correct data to the correct place. We are using Client-Server architecture for data storage and retrieval, meaning that we store the data in the cloud and retrieve whenever asked and then all changes are saved directly in the cloud, even when interactions between users happen, like in reviewals and responses, making the system dependant on the server instead of peer-to-peer.

● **Local Data Storage** enables data to be stored in the user's local, we save only cache data here to enable fast loading in case the same or similar data is requested multiple times.

● **Cloud Data Storage** is responsible for the storage of all data in the application, the data is sent to the cloud to be saved there rather than keeping it in the users’ local devices.

2.2 HARDWARE / SOFTWARE MAPPING

Since the application is web-based, users need a device to connect to the internet and an internet connection. Since the application is web-based a browser is needed to use it. We are using C#9 and ASP.Net framework v4.8. Moreover, ASP.NET needs a server that has Windows 7 or any later versions installed on it to run. However, on the client side the browser and its version are irrelevant as long as the browser supports the main applets, like cookies, HTML and CSS as well as JavaScript. The keyboard is used for many operations in the application like entering username and password, evaluating, grading, and writing reviews and responses. Additionally, a mouse is also required as it is used to click the buttons, images, and other components that are clickable in the application. Since the drag & drop system is used on the assignment submission page for students and the group creation page for instructors, a mouse is also needed there.

2.3 PERSISTENT DATA MANAGEMENT

In the application, all data like the details of users and their submissions and evaluations will be stored 24/7 up to date in an online database, we will use SQL to host and manage the database. Users will be the main actors in the database system. Each user will have certain features that vary depending on what type of user they are. (Student, TA or instructor) These features will be stored in users' sub-tabs in different ways depending on the types of data. For example, the chemistry point of the group can be accessed by following the Courses -> Groups -> Chemistry Score tabs and this data is stored as a double variable and shown in a graph form with its average, comparison to other groups and according to criteria, some of these components will only be seen by members some only by instructor and some by everyone. In addition, when users make any evaluation, leave a note, change their profile, join a new course or group, the data in the storage will be updated instantly.

2.4 ACCESS CONTROL AND SECURITY

When users register to the application, their names, surnames, if they are students, student IDs, email addresses, passwords and department information will be taken and this information will be stored in the online database. Users will not have access to this database, so none of the students, TAs, or instructors will be able to manipulate the application data externally and cause any information leakage. In addition, passwords stored in the database will be stored in encrypted form. Thus, even database administrators will not be able to view users' passwords. Regarding accessibility as seen in the access matrix seen in the figure below we can see the different access levels. For example, instructors can create courses and projects and randomly assign students to groups, students and graders can only join courses and projects and students can create and delete groups and join them while graders can do nothing related to group objects. Additionally, we can see how the grader and instructors can both create assignments and grade them while students can submit them, review others’ submission and invite someone to review their or others’ submissions. For evaluations, an instructor has access to all evaluations of all kinds while grader and students can not see any, they can only see the graphs that analyze them. Students, however, can evaluate both group mates and other groups. For reviews, the

instructors and graders can see all of them, respond to them or endorse them. While students can only respond to them.

Objects Actors	Course	Project	Group	Assignment	Evaluation	Reviews
Instructor	<<create>> createNewCourse()	<<create>> createNewProject()	randomlyAssign()	<<create>> createNewAssignment() grade()	showAll() sendWarnings()	showAll() embrace() respond()
Grader	join()	join()		<<create>> createNewAssignment() grade()		showAll() embrace() respond()
Student	join()	join()	<<create>> createNewGroup() join() delete()	Submit() review() inviteToReview()	evaluateOtherGroups() evaluateGroupMates()	respond()

Figure 2.4 Access Matrix

2.5 BOUNDARY CONDITIONS

The Peerview application will be a web-based application and users will sign in with their Bilkent mails. If the user does not have an account in the application he/she will be able to sign up with their Bilkent mails. If the user tries to sign in or sign up with an email address other than the Bilkent email, the system will not accept the operation. Since the application will work on the web, so that users do not need to install anything.

Once the users sign in the application, they will be able to access the pages provided for them depending on their roles in the system (student, grader, or instructor).They will be able to perform the actions provided for them. If the user is a student, he/she will be able to see their assignments or if the user is an instructor they will be able to assign projects and homeworks.

If the system crashes during any action, data will be lost if the user did not submit their work. Users need to submit their work to avoid data loss. If a student wants to submit a report and system crashes before submission, data will not be stored and the user will need to submit the work again.

If the user wants to log out, they can just close the tab or click on “log out”. Logging out will send the user to log in page.

3. Subsystem Services

3.1 Presentation Tier Subsystem

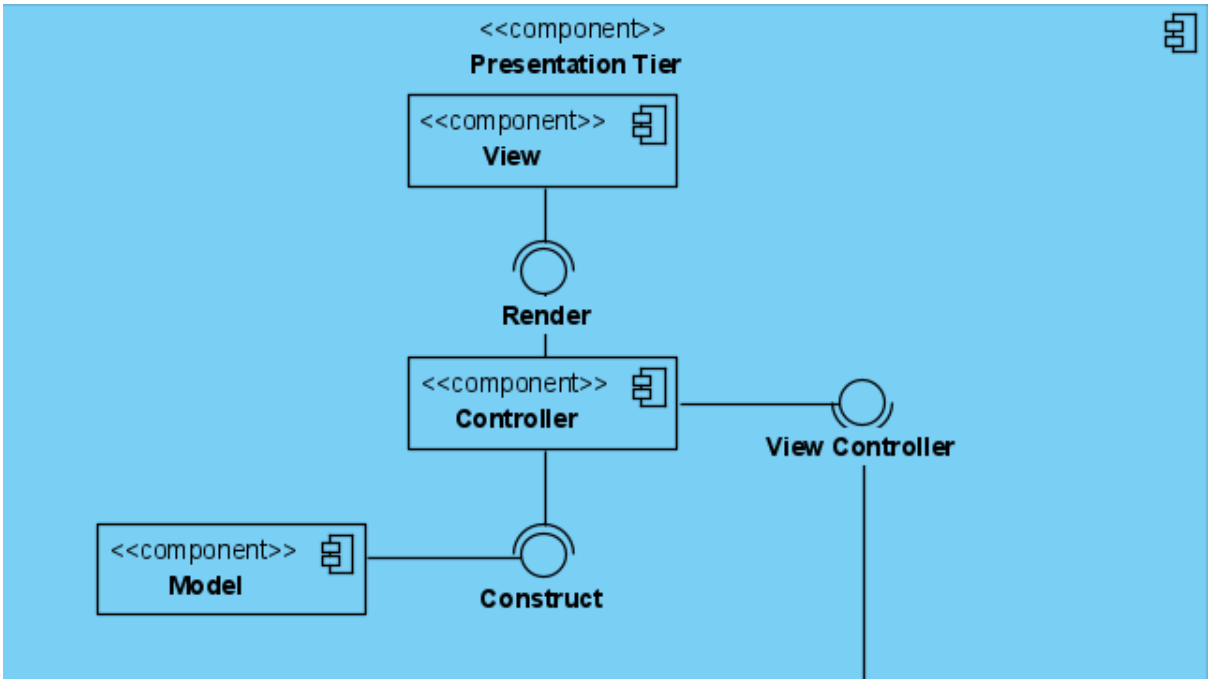


Figure 3.1 Presentation Tier Subsystem

One of the subsystems used by the application is the Presentation Subsystem. This subsystem handles the user interface, namely the front-end part of the application. Since we are using the standard ASP.NET MVC for constructing the Presentation Subsystem, we used Model, View and Controller components in this subsystem.

The user interface of the application is managed by the Presentation Subsystem. This interface contains necessary components such as buttons, labels and views for the user to perform operations within the application. The user interface interacts with the Application Tier, providing end users with the necessary tools for inputting and outputting of content.

Presentation Tier consists of three main components as the following;

3.1.1 Model Component

Model component provides the creation of necessary objects and data for the user interface. These objects and data are transferred to the user interface using the Controller Component when a content will be displayed in the user interface.

3.1.2 View Component

View component is the interface that the end user sees and interacts with. The positions of the buttons, text boxes or labels used in the construction of the interface are handled by the View Component. View Component needs the Controller Component to update the objects or data it contains.

3.1.3 Controller Component

Controller component is responsible for data and object transfer between View and Model components. For example, if an image needs to be displayed in the user interface, the controller calls that image from the model and sends it to the view in order to be displayed. Also, when a button is pressed in the user interface, the listeners in the View Component receive the inputs and transfer them to the Controller Component for the necessary action.

3.2 Application Tier Subsystem

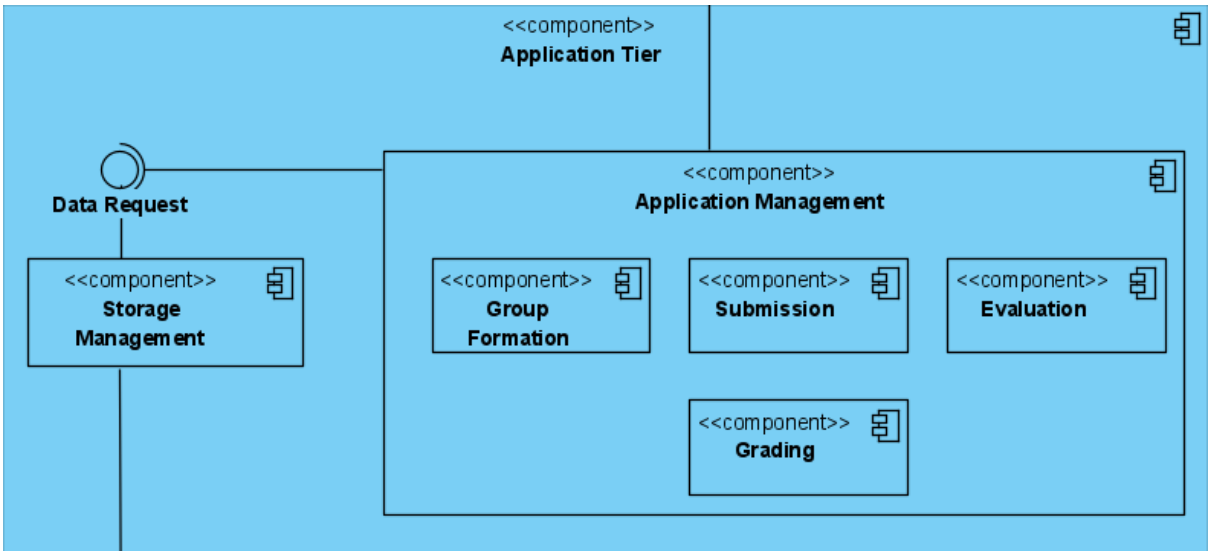


Figure 3.2 Application Tier Subsystem

3.2.1 Storage Management Component

Storage Management component is responsible for storing the necessary data while the application is in use. For example, while the user is viewing another user's profile, data about the user is first transferred from the Data Tier layer to the Storage Management component. This data is then transferred to the Application Management component, where it is processed and then sent to the Presentation Tier for display to the user.

3.2.2 Application Management Component

The Application Management component is responsible for processing the requests made by the user in the Presentation Tier. It takes requests from the Presentation Tier, performs the necessary actions, and responds back to the Presentation Tier. It also requests data from the Storage Management component, processes this data and sends it to the Presentation Tier.

3.2.2.1 Group Formation Component

This component is responsible for the necessary actions that are taken for the user to form a group in the application. Such as creating random groups, sending requests to join groups, creating new groups.

3.2.2.2 Submission Component

This component is responsible for performing tasks such as opening the file browser, uploading the file with the drag and drop method, and editing the submission so that the user can submit assignments.

3.2.2.3 Evaluation Component

This component performs necessary actions in terms of submitting and editing evaluations in the background so that the user can perform the evaluation process within the application.

3.2.2.4 Grading Component

This component is responsible for the necessary actions such as add notes and submit grades so that the user can grade within the application.

3.3 Data Tier Subsystem

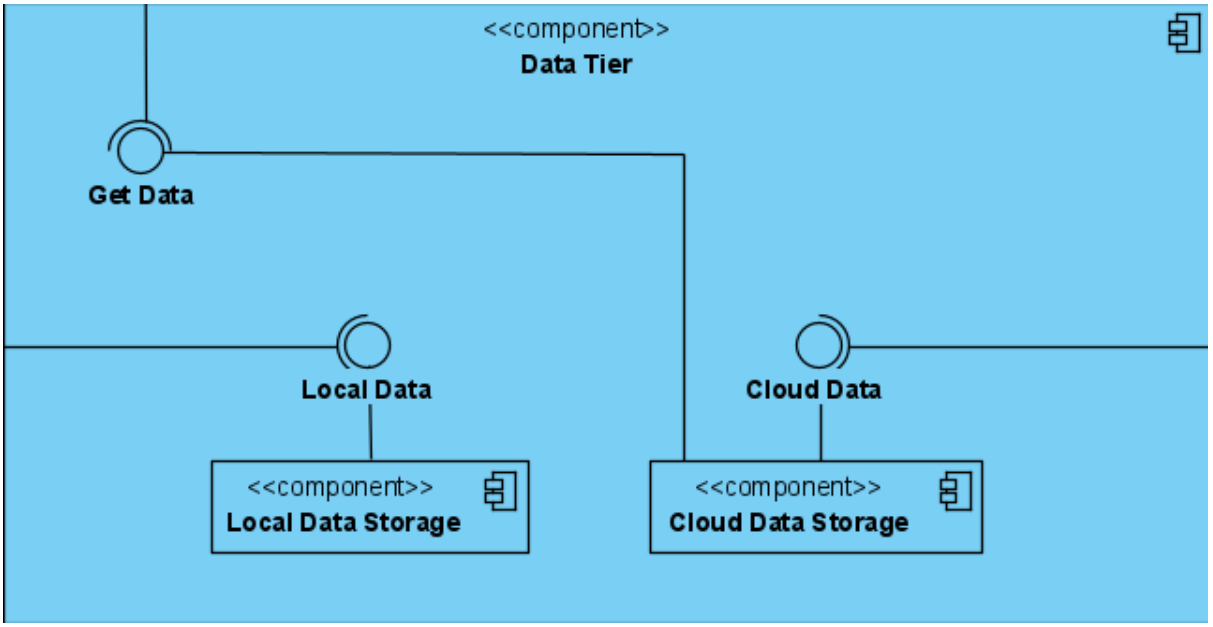


Figure 3.3 Data Tier Subsystem

The Data Tier subsystem is responsible for data management and data communication. This subsystem is secure because it can only be accessed from the Application Tier subsystem. In this way, every change that occurs in the Data Tier subsystem will have a relevant request in the Application Tier subsystem. Also, this subsystem has 2 different components. These components are responsible for storing data in different ways.

3.3.1 Local Data Storage Component

This component is responsible for storing the data in users' local.

3.3.2 Cloud Data Storage Component

This component is responsible for storing all data on the system in the cloud. Data is taken from this component and transmitted to the Application Tier subsystem for processing.

4. LOW LEVEL DESIGN

4.1 Class Diagrams

4.1.1 Evaluation Class Diagram

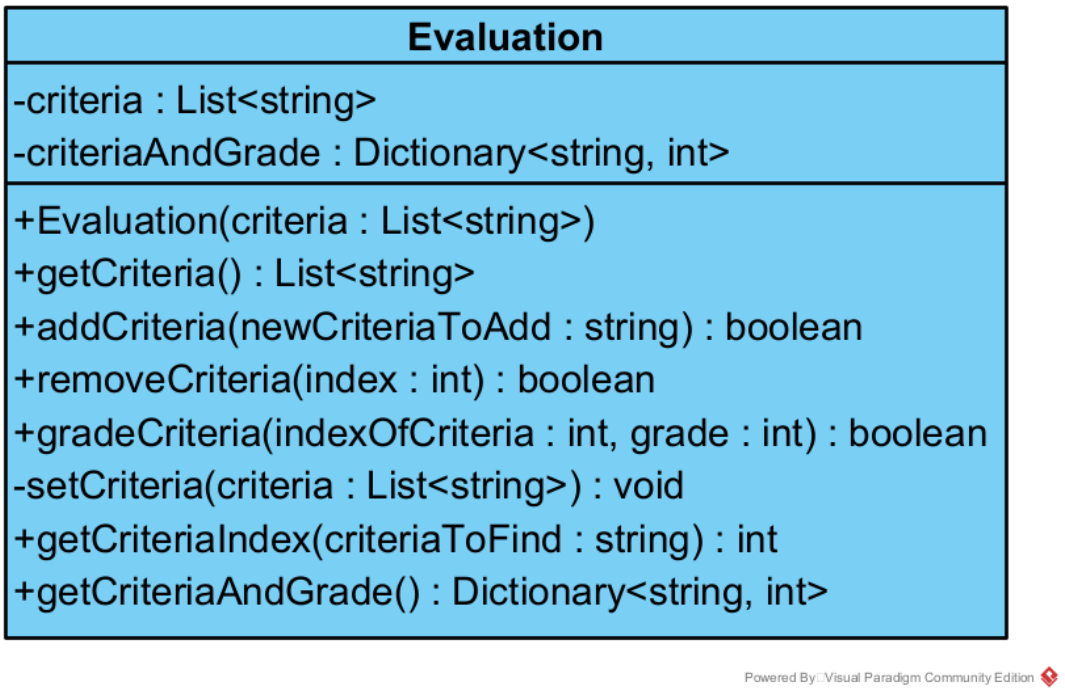


Figure 4.1.1 Evaluation Class Diagram

Attributes
criteria is a List of strings, that is used as questions that are asked to the student when they are asked to evaluate their friends.
criteriaAndGrade is a dictionary that keeps the score of the given criteria.

Methods
Evaluation() is the constructor that takes a list of strings that will be used as criteria on the evaluation form.
getCriteria() returns the questions that are set by the Instructor, and this function is called when the evaluation form is sent to the student.
addCriteria() is called when the Instructor wants to add more evaluation criteria, returns a boolean value that is true if the function call is successful, else false, in that case the program can decide what to prompt the user.
removeCriteria() takes an index that is used to find the criteria that the user wants to remove from the questionnaire. It returns true if the call is successfully ended, else it returns false thus, the program can decide what error message to prompt the Instructor.

gradeCriteria() is used to set a specific grade to a given question. Returns true if the call is successful, else returns false, and thus the program can prompt the correct error message.
setCriteria() is called to set the criteria to a given list of questions.
getCriteriaIndex() returns the index of the given string. If cannot find the given string in the List returns -1.
getCriteriaAndGrade() returns the Dictionary of criteria and its grade.

4.1.2 Response Class Diagram

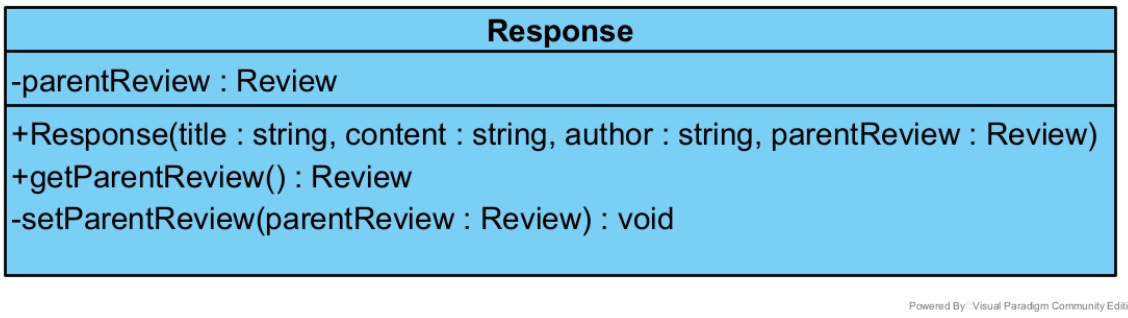


Figure 4.1.2 Response Diagram

Attributes
parentReview is an instance of the review which will be the response.

Methods
Response is the constructor of the Response class. It takes four parameters to initialize the Response which are title, content, author (who writes the response), and the review.
getParentReview is the function which returns the Response in Review type.
setParentReview is the set method of the Response

4.1.3 Review Class Diagram

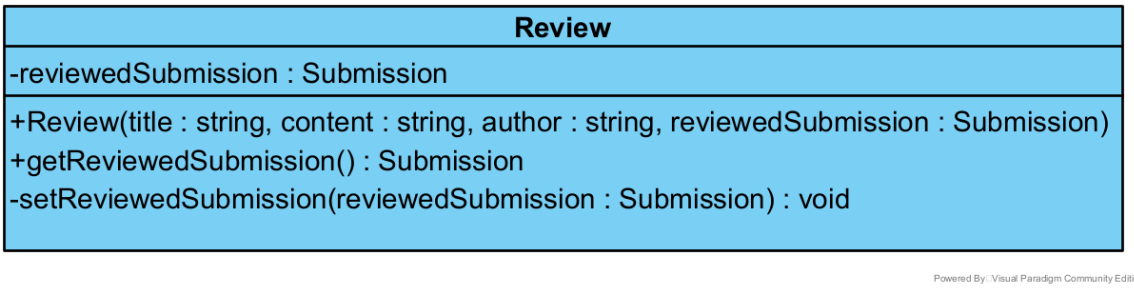


Figure 4.1.3 Review Class Diagram

Attributes
reviewedSubmission is a submission that is required for the Review.
Methods
Review() constructor takes title, content, author, and a Submission.

getReviewedSubmission() returns the submission
setReviewedSubmission() is called to set the reviewedSubmission attribute.

4.1.4 CompletedSubmission Class Diagram

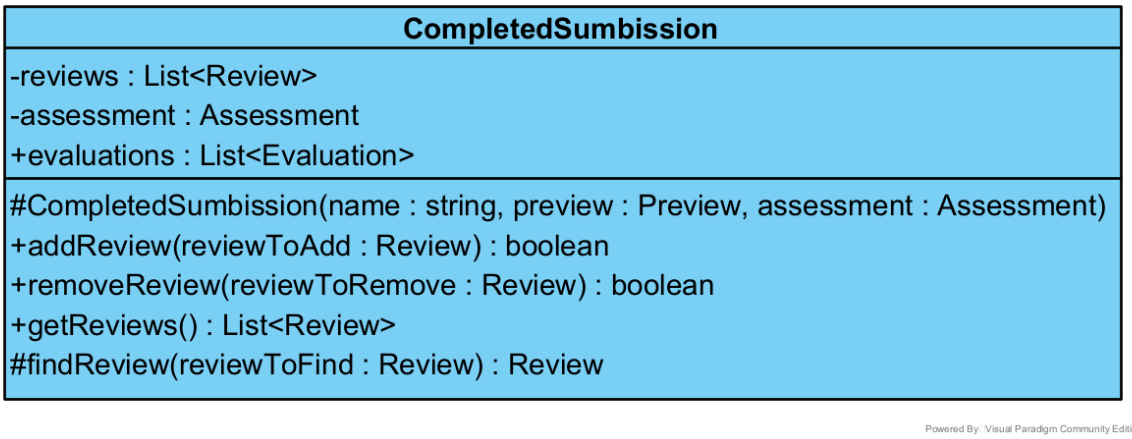


Figure 4.1.4 Completed Submission Class Diagram

Attributes
reviews is a list of reviews for a completed submission.
assessment is the assessment for the given completed submission.
evaluations is a list of evaluations.

Methods
CompletedSubmission() is a constructor that takes name, note, and an assessment.
addReview() is used to add a review to a CompletedSubmission.
removeReview() is used to remove a review to a CompletedSubmission.
getReviews() returns the list of reviews to a given CompletedSubmission.
findReview() is used to search the list of reviews and if the given exists in the review list then returns the review else returns null.

4.1.5 Comment Class Diagram

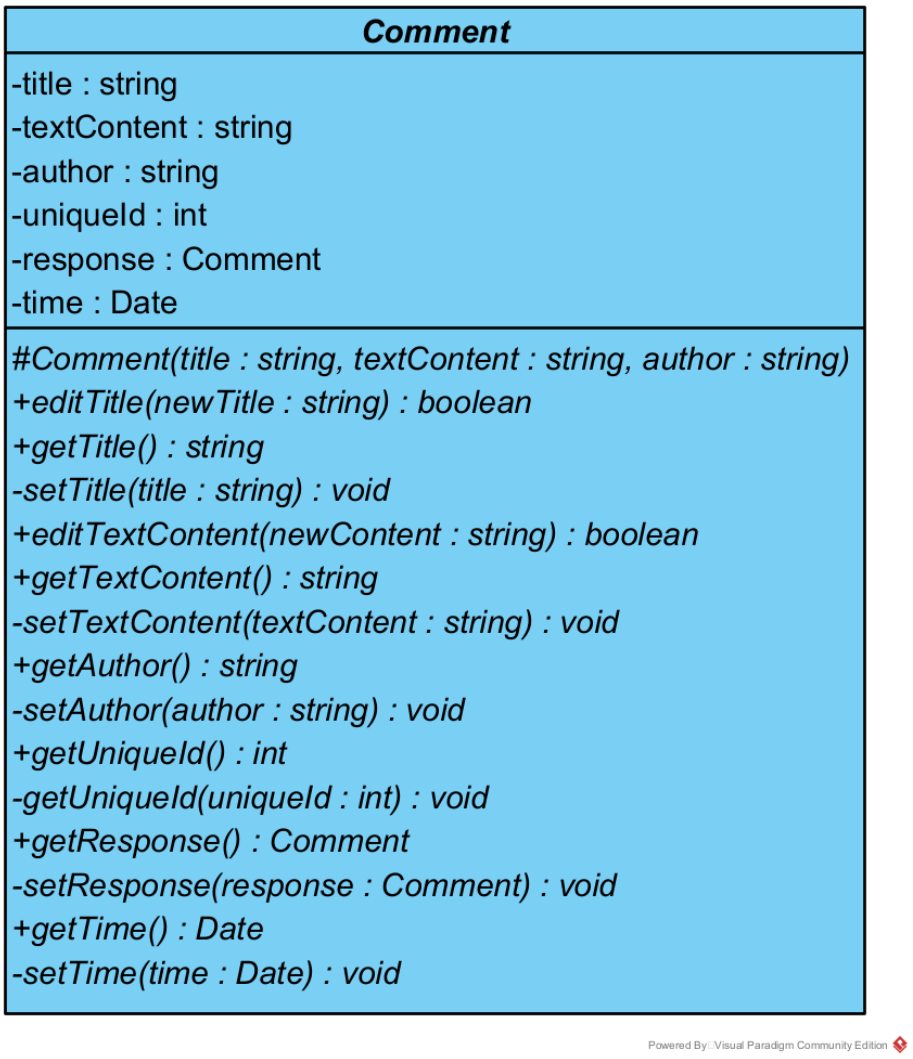


Figure 4.1.5 Comment Class Diagram

Attributes

title is the title of the comment**.**
textContent is given comments content.
author is the name of the commenter.
uniqueId is a unique id.
response is set null if there exists zero or less following up comment, else it points to the next comment.
time is the time that comment added to the program.

Methods

Comment() is the constructor for the Comment
editTitle() is called when a user wants to edit the comment title.
getTitle() returns the title of the comment.

setTitle() is called to set the title of the comment to the given string.
editTextContent() is called when the user wants to edit the comment content.
getTextContent() is called to retrieve the data.
setTextContent() is called to set the content to a given string.
getAuthor() returns the author of the comment.
setAuthor() sets the author of the comment.
getUniqueId() returns the comment's unique id.
getResponse() returns the response to the comment.
setResponse() is called to set the following response to the given comment.
getTime() returns the time when the comment is added to the program.
setTime() sets the time attribute of the comment.

4.1.6 Submission Class Diagram

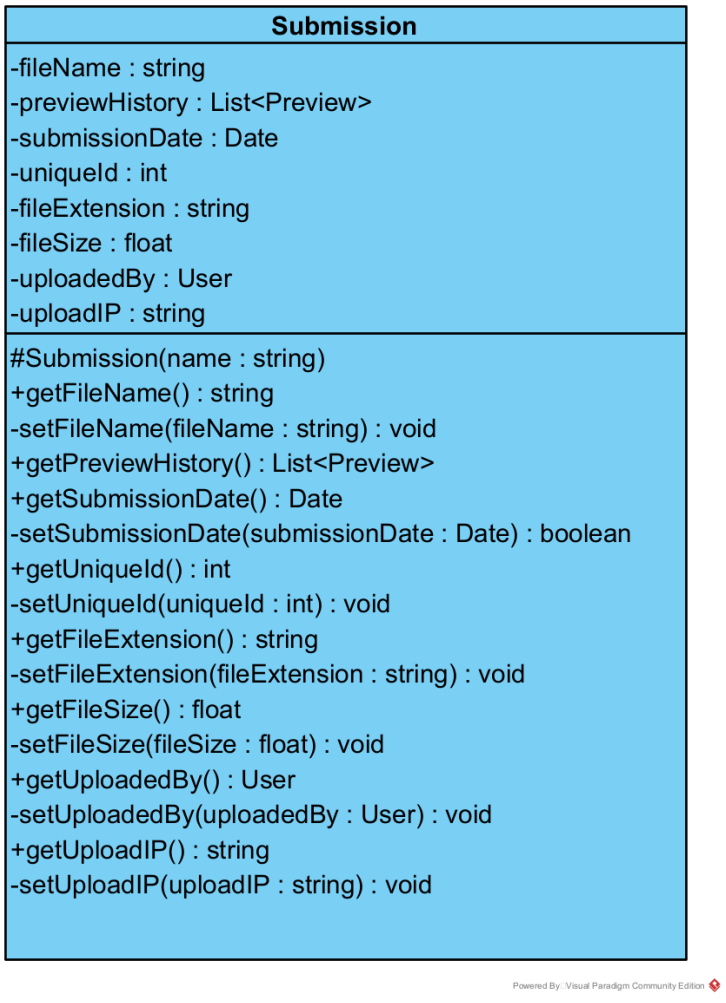


Figure 4.1.6 Submission Class Diagram

Attributes
fileName is the name of the file that is submitted.

previewHistory is a List of Notes that are given for this Submission.
submissionDate is the date of the submission.
uniqueId is the unique id of the submission.
fileExtension is the extension of the type of the submission.
fileSize is the file size of the submission.
uploadedBy is the owner of the submission.
uploadIP is the IP of the owner of the submission.
Methods
Submission() is the constructor for the submission.
getFileName() returns the name of the file that is submitted.
setFileName() sets the name of the file that is submitted.
getPreviewHistory() returns List of Notes that are given for this Submission.
getSubmissionDate() returns the date of the submission.
setSubmissionDate() sets the date of the submission.
getUniqueId() returns the unique id of the submission.
setUniqueId() sets the unique id of the submission.
getFileExtension() returns the extension of the file that is submitted.
setFileExtension() sets the extension of the file that is submitted.
getFileSize() returns the size of the file that is submitted.
setFileSize() sets the size of the file that is submitted.
getUploadedBy() returns the owner of the file that is submitted.
setUploadedBy() sets the owner of the file that is submitted.
getUploadIP() returns the IP of the submission owner.
setUploadIP() sets the IP of the submission owner.

4.1.7 Comment Review Response Submission Class Diagrams with their relationships

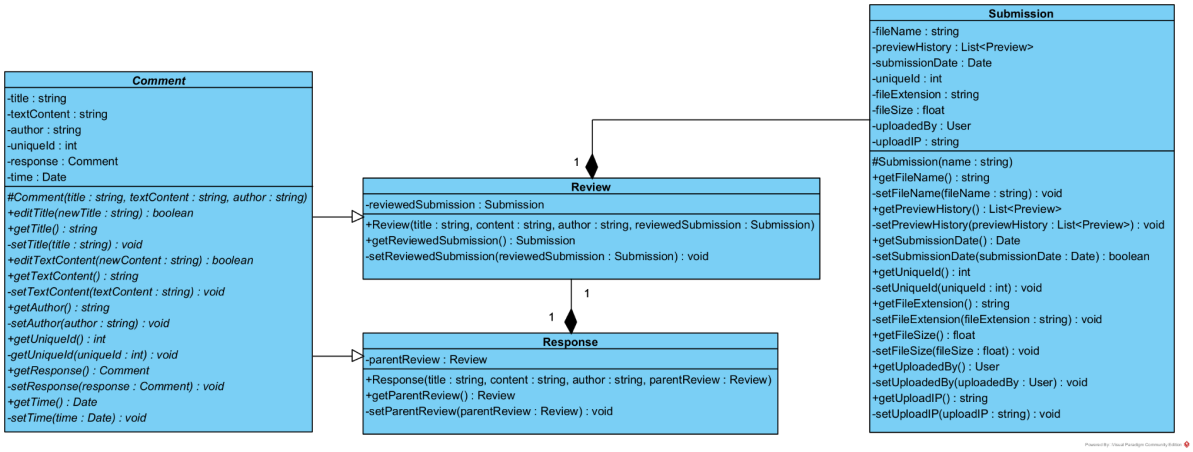


Figure 4.1.7 Comment, Review, Response and submission Class Diagrams

Comment is a blueprint for Review and Response.

4.1.8 Note Class Diagram

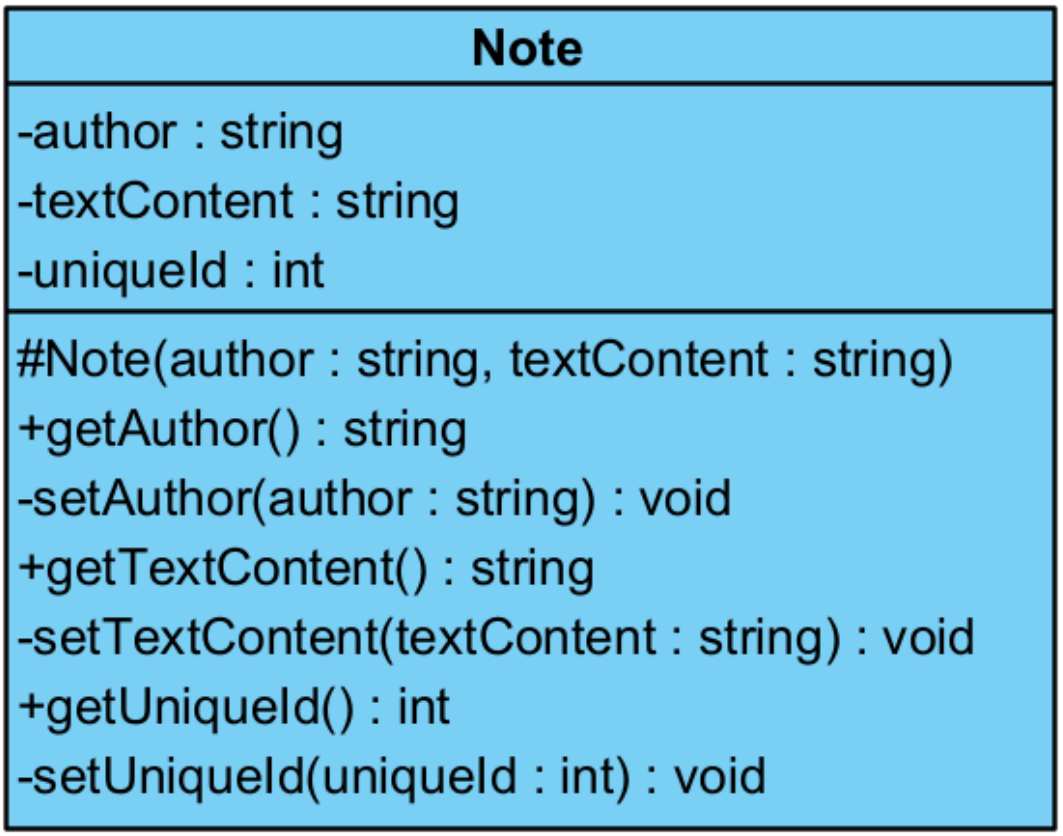


Figure 4.1.8 Note Class Diagram

Attributes

author is the name of the person who writes the Note.
textContent is the content of the Note.
uniqueId Note's unique id to differentiate them between other Notes.

Methods

Note() is the constructor to instantiate a Noteobject.
getAuthor() returns the name of the author.
setAuthor() is called to set the author of the Note.
getTextContent() returns the content of the Note.
setTextContent() is called to set the textual content of the Note.
getUniqueId() returns the unique id of the Note.
setUniqueId() is called to set the unique id of the Note.

4.1.9 Evaluation CompletedSubmission Review Submission Note Diagram

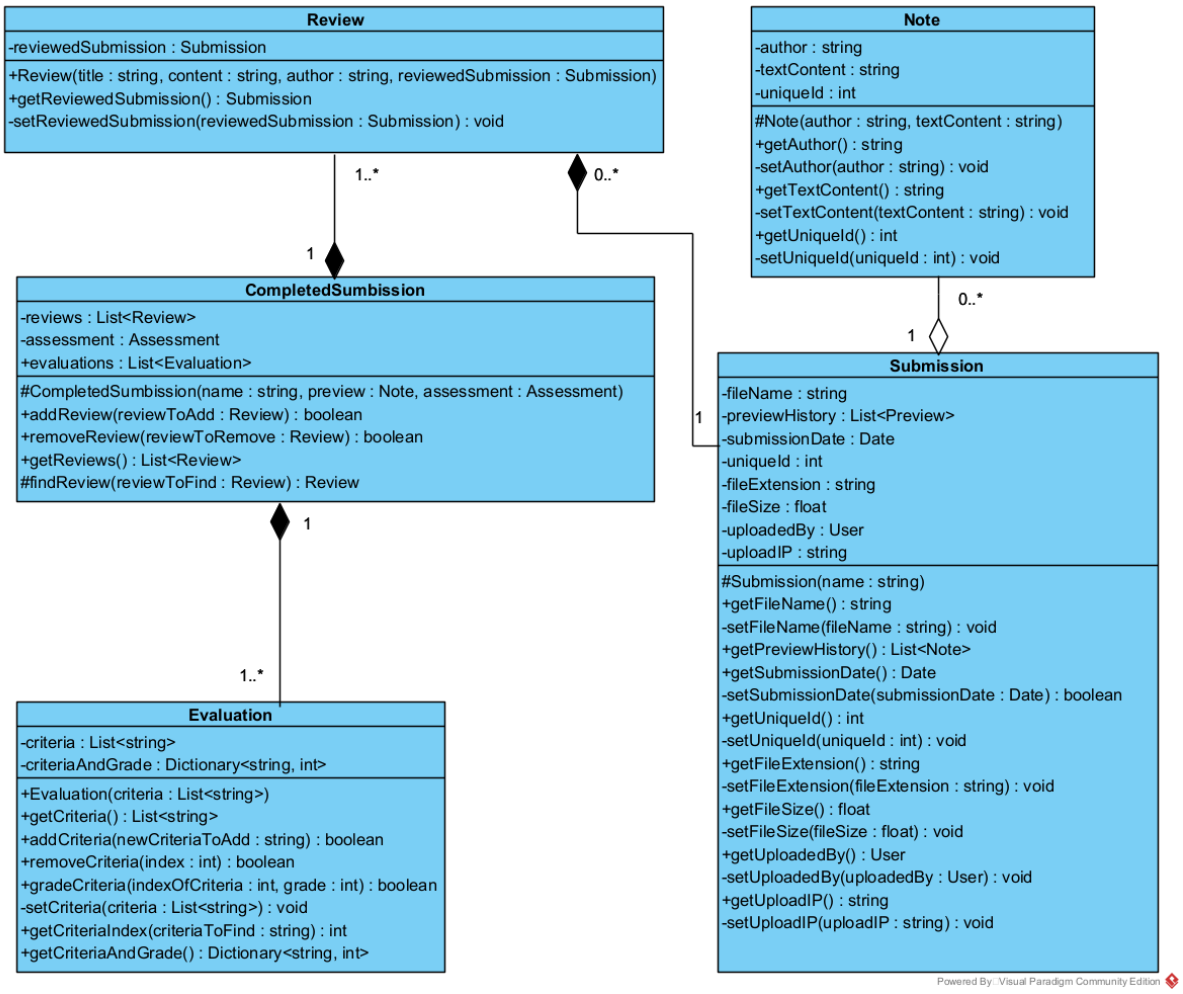
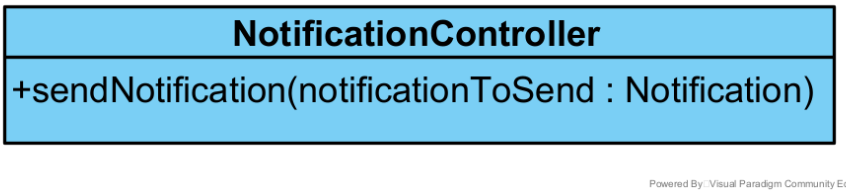


Figure 4.1.9 Diagram depicting relation between classes

Evaluations and Reviews are required for CompletedSubmission thus, a submission can be complete. Notes are gathered at Submissions.

4.1.10 NotificationController Class Diagram



Methods
sendNotification() is responsible for sending notifications to the users.

4.1.11 Notification Class Diagram

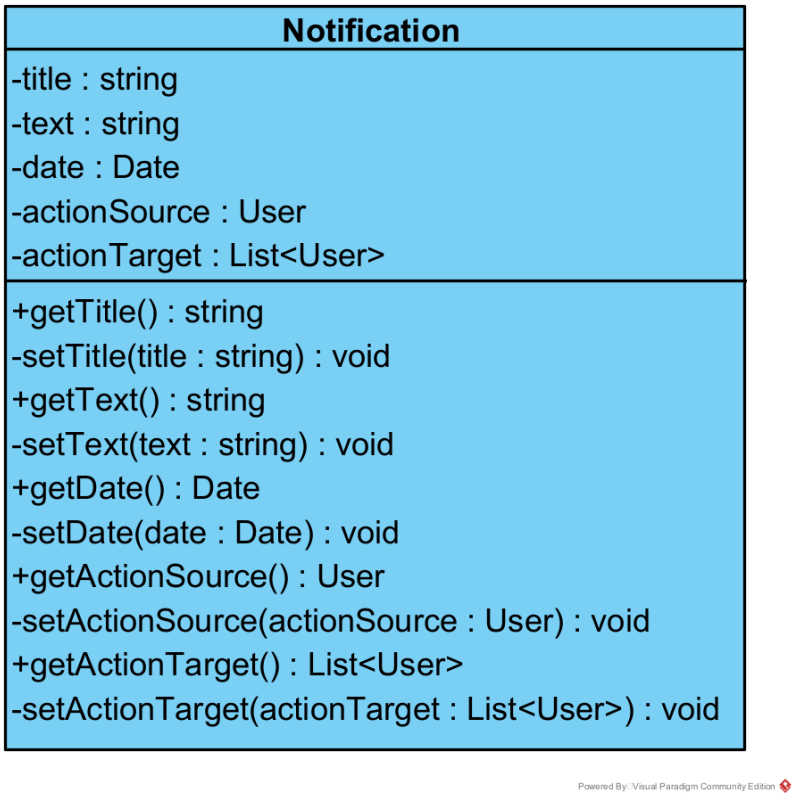


Figure 4.1.11 Notifications Class Diagram

Attributes
title is the header of the notification. It defines what the notification is about
text holds the real content of the notification.
date holds the date when the notification is sent to the user.
actionSource holds the user who is responsible for the notification.
actionTarget holds a list of users who will receive the notification.

Methods
getTitle() returns the title (header) of the notification.
setTitle() is called to set the title of the notification
getText() returns the real content (body) of the notification.
setText() is called to set the content of the notification.
getDate() returns the date of the notification (when the notification is sent).
setDate() is called to set the date of the notification.
getActionSource() returns the user who is responsible for the notification.
setActionSource() is called to set the user who is responsible for the notification.
getActionTarget() returns the list of users who will receive the notification.
setActionTarget() is called to set the list of users that will get the notification.

4.1.12 Notification NotificationController Aggregation

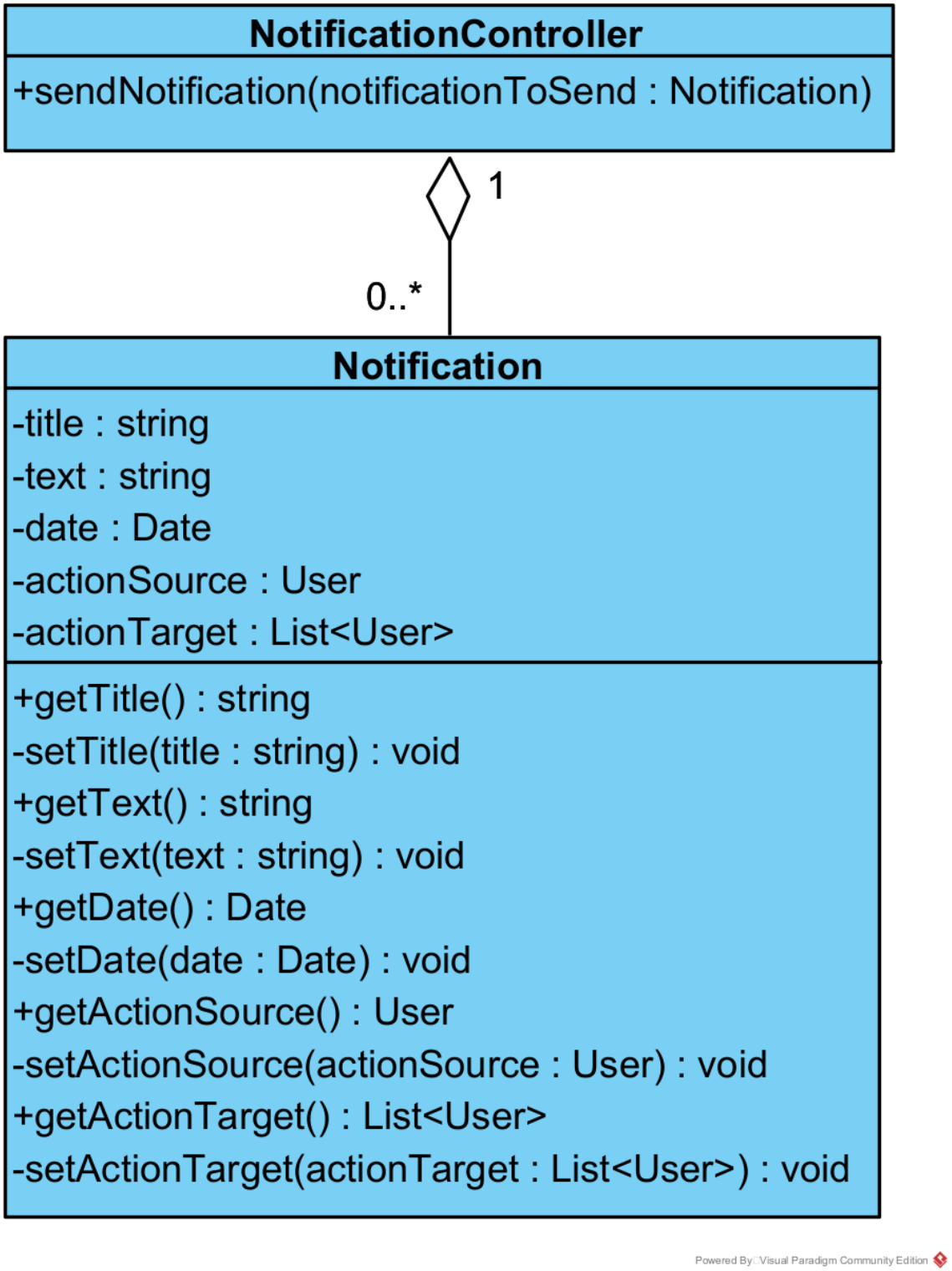


Figure 4.1.12 NotificationController Class Diagram

NotificationController is responsible for creation of the notifications and sending them to the correct users.

4.1.13 Grader Class Diagram

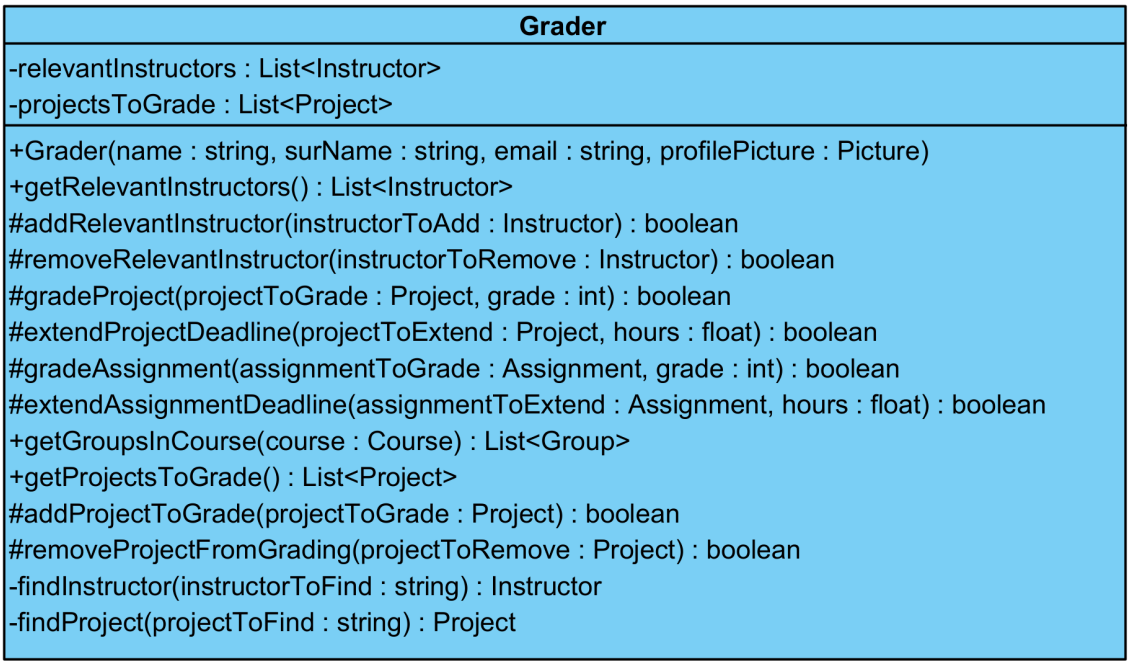


Figure 4.1.13 Grader Class Diagram

Attributes:
relevantInstructor: here have the list of instructors that a grader is added to. The reason for adding this attribute is similar to the reason graders are added to the instructor class, to keep track of things and to limit authority based on the authority of the instructor.
projectsToGrade: This attribute represents the projects that a grader can access and grade.

Methods:
Grader (): A constructor that creates the instance of a grader and assigns its essential attributes.
getRelevantInstructors(): Returns the list of instructors which can grade
addRelevantInstructor(): Adds an instructor to the list
removeRelevantInstructor(): Removes an instructor from the list
gradeProject (): Here a grader grades a project using the parameter passed and then the function returns a boolean to confirm the status of process; success or failure for some reasons.
gradeAssignment (): Here a grader grades an assignment using the parameter passed and then the function returns a boolean to confirm the status of process; success or failure for some reasons.
extendProjectDeadline () : Here a request to the system to extend a deadline of project, then return true in case of success, or false if the user does not have authority or if it is not allowed.
extendAssignmentDeadline () : Here a request to the system to extend a deadline of an assignment, then return true in case of success, or false if the user does not have authority or if it is not allowed.

getGroupsInCourse (): Returns the list of groups in a specific project.
getProjectsToGrade(): Returns the list of projects in the group.
addProjectsToGrade(): Adds a project to the list.
removeProjectsFromGrading(): Removes a project from the list
findInstructor(): Returns the instructor of the group.
findProject(): Returns the assigned project

4.1.14 Instructor Class Diagram

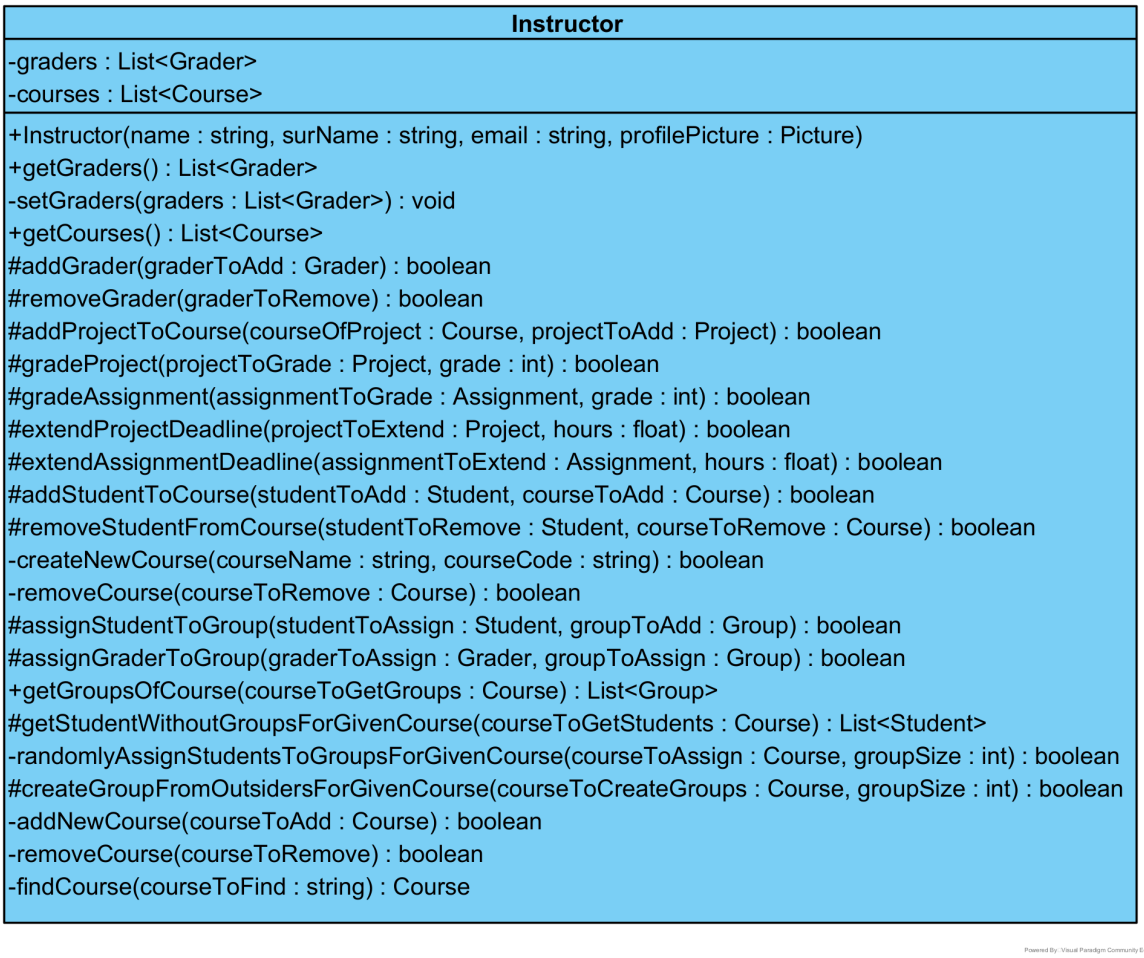


Figure 4.1.14 Instructor Class Diagram

This class represents an instructor, it implements the user class. In this class we represent an instructor with her main info and the methods related to her authority.

Attributes:
graders: This represents the graders that the instructor appointed. Each Grader is assigned to a course but they are also stored according to the instructor so we can limit their access according to the instructor's access and in order to keep track of things.
courses: This represents the courses that the instructor is teaching.

Methods:

Instructor () : This is a constructor where we create an instructor and assign the essential values to it.
getGraders () : Returns a list of all graders that were assigned by the instructor.
setGrader(): Assigns a list of users as graders
addGrader () : In this method we assign a grader to an instructor for reasons explained in the graders part, returns whether succeeded or not in a boolean.
removeGrader() : In this method we remove a grader from the instructor she was assigned to and from the course, then return whether the process was successful or not as a boolean.
addProjectToCourse () : Assigns a project to a course.
gradeProject () : Sends a grade to be assigned in a project instance and then the boolean return variable is used to return whether it was a success to assign the grade or not.
gradeAssignment () : Here the instructor class sends a request to grade an assignment, then if it was completed the return variable is true, otherwise it is false.
extendProjectDeadline () : Here the instructor extends the deadline for the project, then returns a boolean to confirm that the process was completed.
extendAssignmentDeadline () : Here the instructor extends the deadline for the assignment, then returns a boolean to confirm that the process was completed.
removeStudentFromCourse () : Here the instructor can remove a student from a course, the boolean represents the success of the process.
addStudentToCourse () : Here the instructor can add a student to a course, if the authority was given to the instructors to do so, the boolean represents the success of the process.
createNewCourse () : Here we set the name and the code for a course once it is created and then return it.
removeCourse () : Here the instructor's class sends a request to remove a course from the system, then if successful returns a confirmation as a boolean.
assignStudentToGroup () : Here the instructor assigns a student to a group, this option depends on the course, could be rejected if chosen not to be available or if the student is already in a group or if there was a problem. Accordingly the system returns a boolean variable to confirm the success of the process.
assignGraderToGroup () : Here the instructor assigns a grader to a group then a boolean is returned to inform the system regarding success or failure of the process.
getGroupsOfCourse () : returns a list of groups in a course.
getStudentsWithoutGroupsForGivenCourse () : This returns a list of students without a group in a course, this can be used to know how to assign the remaining students to groups.
randomlyAssignStudentsToGroupsForGivenCourse () : Here we have the option to assign all students to random groups of group_size size, then return boolean to show the case of the success of the process.
createGroupsFromOutsidersForGivenCourse () : Here we have the option to assign all remaining students to random groups of group_size size, then return boolean to show the case of the success of the process.
getCourses (): Returns a list of courses that an instructor is a member of.
addNewCourse () : adds course passed in the parameter and then return whether that was done or not
removeCourse (): removes the courses passed as a list in the parameter and then returns whether that was done or not.
findCourse(): finds and returns the specified course.

4.1.15 Student Class Diagram

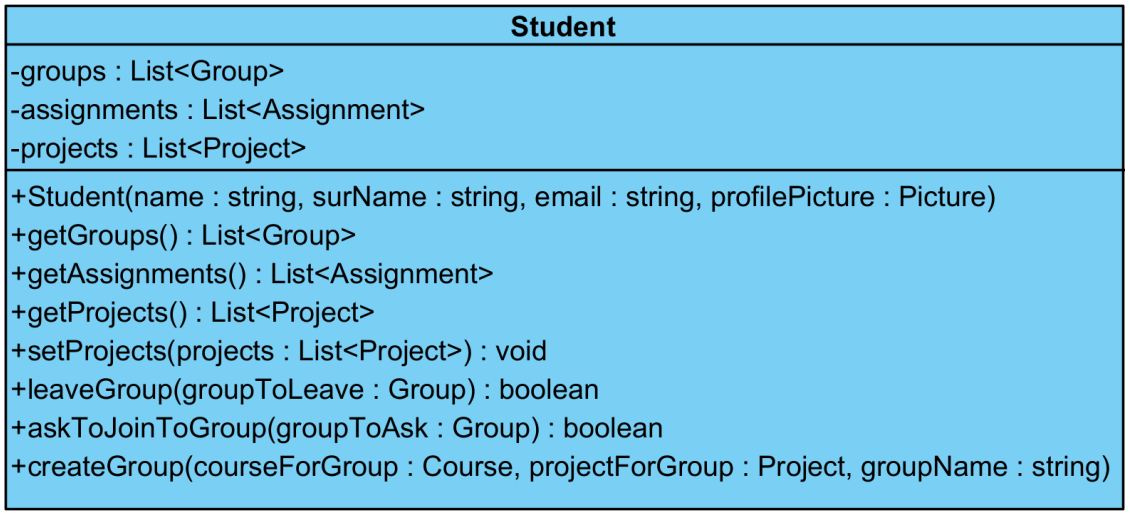


Figure 4.1.15 Student Class Diagram

This class represents the student, it implements the user class, the main user of the system. In this class we have the info of the student, like the name, email, and email, and the courses, groups, and projects that she chose to enrol in.

Attributes:

groups: This attribute represents the array of the groups the student is a member of.

assignments: The attribute represents the array of assignments that the student has to submit.

projects: This attribute represents the projects that the student is enrolled in.

Methods:

Student () : This is a constructor used to initialize the object of a student using the most important information about her.

getGroups() : Returns all groups that the student is a member of.

askToJoinToGroup() : Here we use this when a student sends a request to join a group when there is a place available during the groups' formation stage, returns true in case of success of sending the request.

leaveGroup() : Here we use this when a student sends a request to leave a group returns true in case of success of leaving the group.

getAssignments() : Returns all assignments that a student has to submit.

getProjects() : Return projects that the user is enrolled in.

setProjects(): Gets a list of projects and assigns them to the student.

createGroup() : Here the user sends a request to the system to create a group, if the course policy allows that the group is created, otherwise it is rejected. A boolean is returned accordingly.

removeGroup(): Here the user sends a request to the system to remove her group, if the course policy allows that the group is removed, otherwise it is rejected. A boolean is returned accordingly.

4.1.16 User Class Diagram

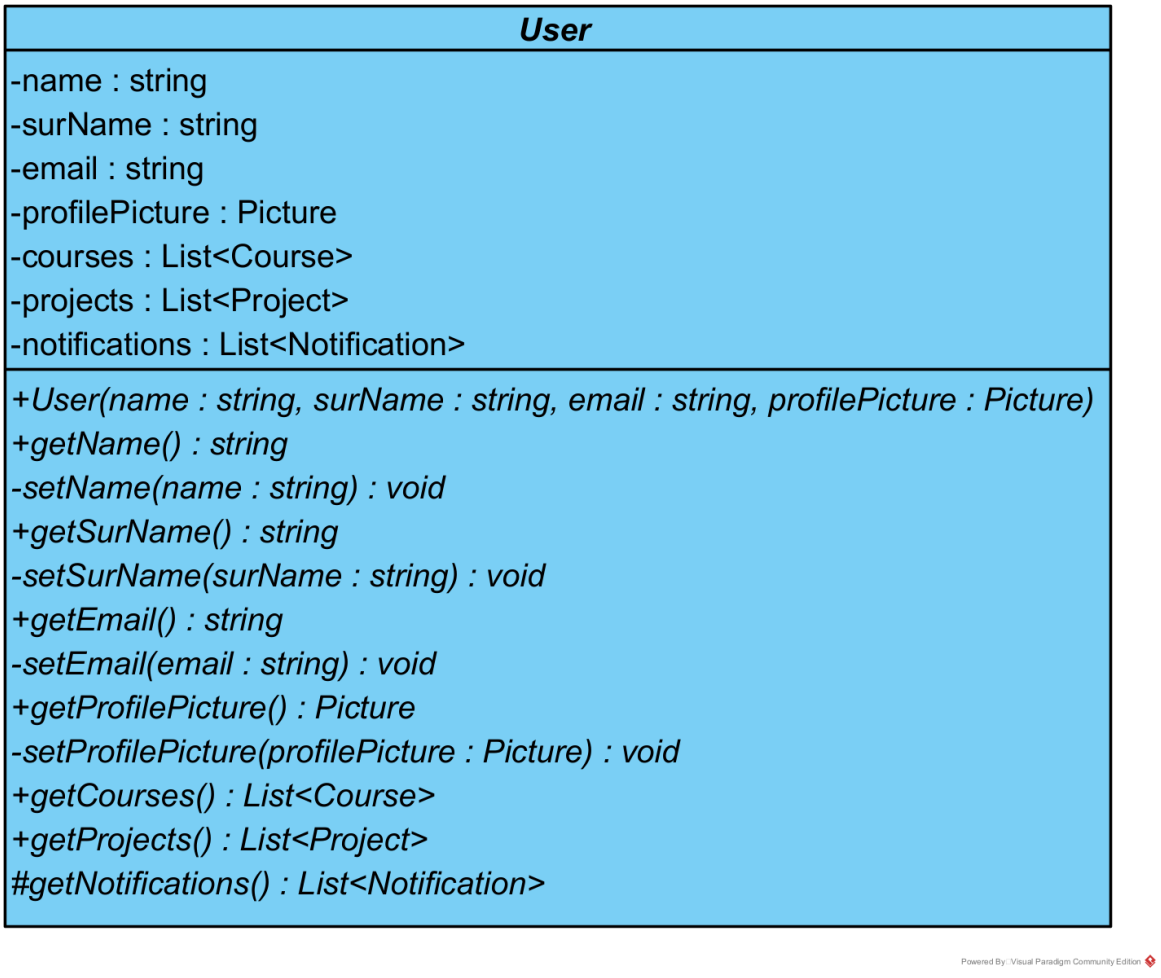


Figure 4.1.16 UserClass Diagram

In this class we represent the user class, which is an abstract class that all user classes inherit. Here we have the most important data that are shared between all users, like name, email and courses. We also deal with notifications and inbox here.

Attributes:

name: Here we save the user's first name.

surName: Here we save the user's last name.
email: Here we save the user's email.
courses: We save the course that a user is a part of. For all users, students, graders, and instructors, we have courses that they are a part of.
profilePicture: A personal picture of the user.
notifications: An array of notifications that a user has.
projects: The list of projects the user is assigned to.

Methods:
User (): Here we instantiate a user using the most important and distinguishable features, like name, surname, and email.
getName (): Returns the name of the user.
getSurName (): Returns the surname of the user.
getEmail (): Return the email of the user.
setName (): Sets the name of the user.
setSurName (): Sets the surname of the user.
setEmail (): Sets the user's email.
getCourses (): return a list of the user's courses.
joinCourse (): Send a request to join a request, if successful return true, otherwise returns false.
leaveCourse (): Sends a request to the system to leave a course if successful returns true, otherwise returns false.
getPic(): Returns the user's picture.
setPic (): Sets the user's personal picture.
getNotifications(): Returns a list of all of the user's notifications.
getInbox(): Returns a list of all messages in the inbox for a user.
sendMessage (): Send a message to a user and then confirm if the other user received it by return true, or false otherwise.

4.1.17 User Class and Instructor Grader Student Generalization

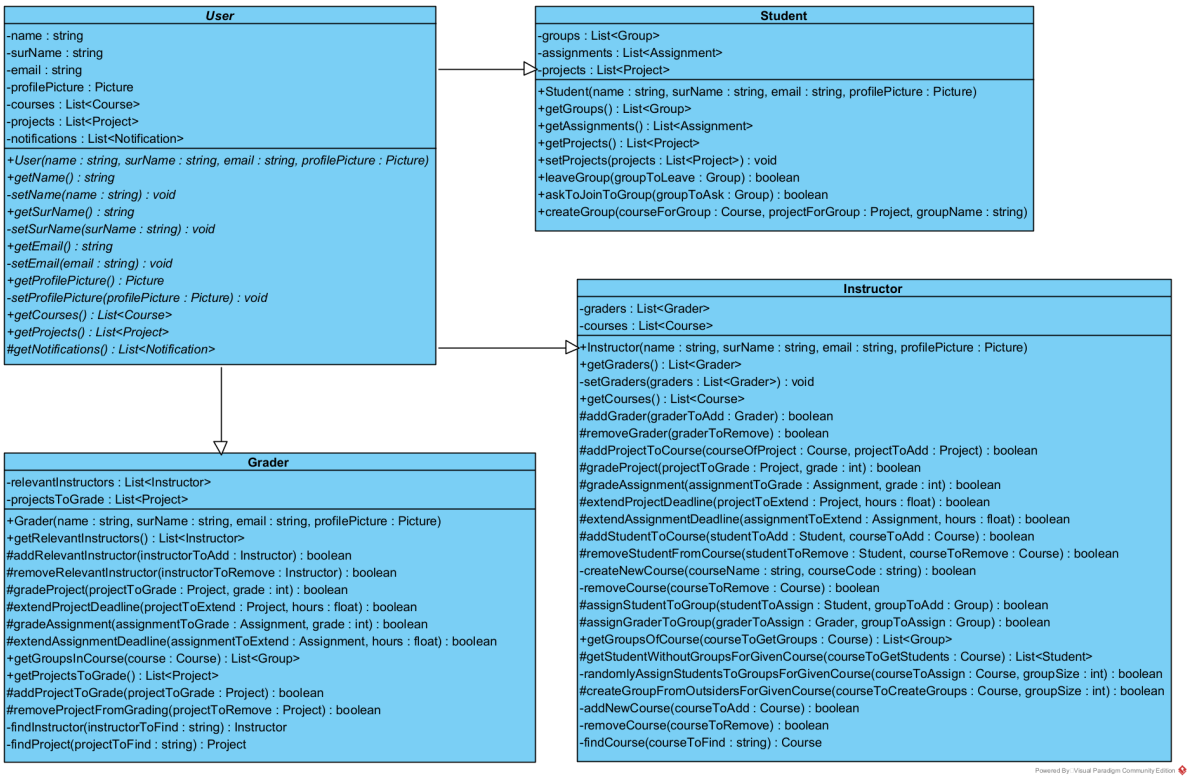


Figure 4.1.17 Grader, User, Student and Instructor Class Diagram depicting relation between classes

User is the abstract class of every type of user that is expected to use the program. Student is for students, Grader is for graders, Instructor is for Instructors, at least we expect that.

4.1.18 Assessment Class Diagram

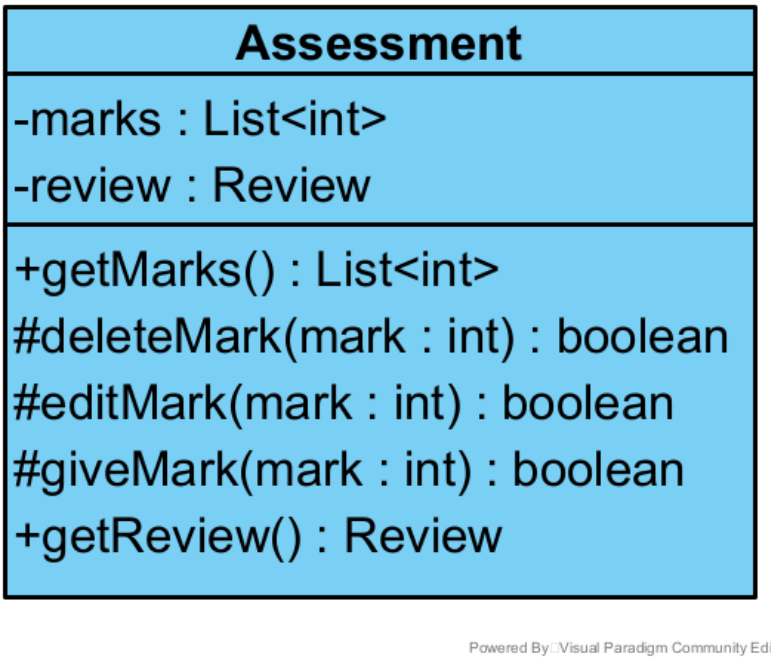


Figure 4.1.18 Assessment Class Diagram

Attributes
marks is the list of integers for the grades.
review is the Review for the given Assessment.
Methods
getMarks() is called to retrieve the mark list.
deleteMark() is called the remove a grade from the list.
editMark() is called to edit the mark.
giveMark() is called to give a mark to the assessment.
getReview() is called to get the Review.

4.1.19 Assignment Class Diagram

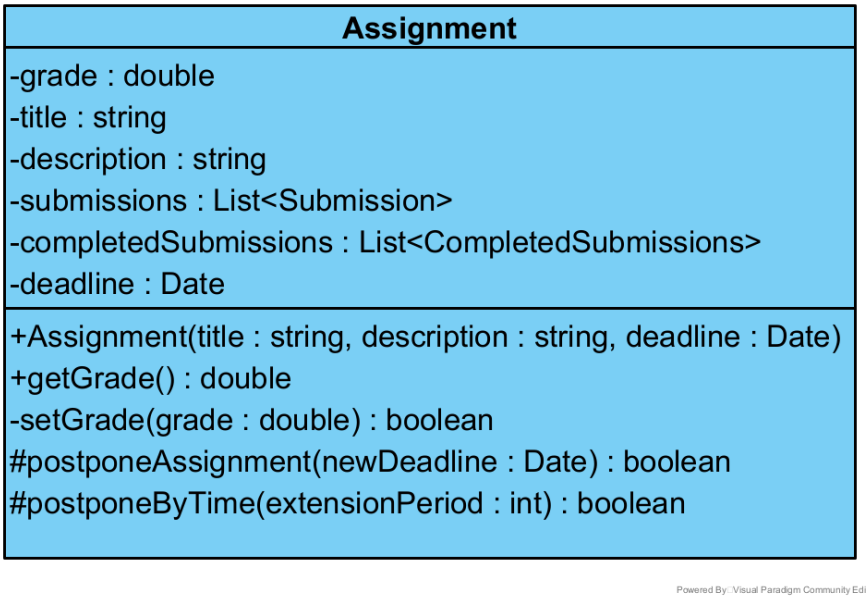


Figure 4.1.19 Assignment Class Diagram

Attributes

grade is a double to store the grade for the assignment.
title is the title of the assignment.
description is the description of the given assignment.
submissions is a list of submissions for this given assignment.
completedSubmissions is a list of submissions that are done.
deadline is the end date of the given assignments submission.

Methods

Assignment() is the constructor for the given assignment.
getGrade() returns the grade.
setGrade() sets the grade of the assignment.
postponeAssignment() takes a date and postpones the submission time to given time.
postponeByTime() takes an int and according to that int it extends the time.

4.1.20 Project Class Diagram

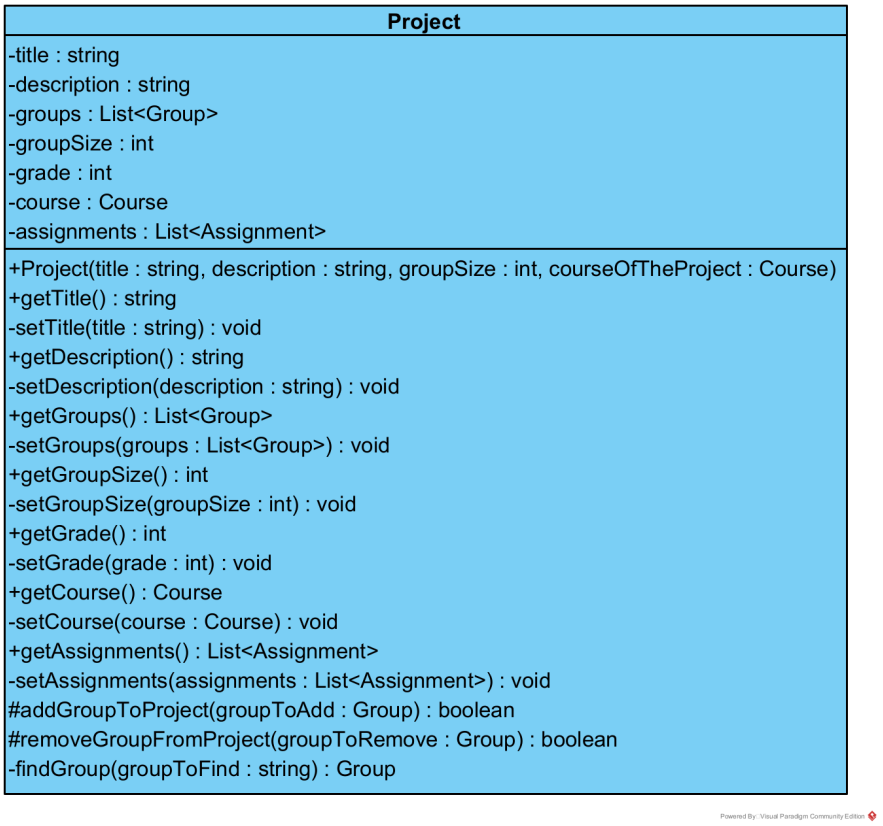


Figure 4.1.20 Project Class Diagram

Attributes
title holds the title of the project
description holds the description (what the project is about) of the project.
groups holds the list of groups which are responsible for completing the project.
groupSize holds the number of students who can be in the group that works on the project
grade holds the grade of the project.
course holds the course which the project belongs to.
assignments holds the assignments that are required to complete the project.

Methods
Project is the constructor of the Project class. It takes four parameters which are title, description,groupSize and courseOftheProject.
getTitle() returns the title of the project.
setTitle() is called to set the title of the project.
getDescription() returns the description (what the project is about) of the project.
setDescription() is called to set the description of the project.
getGroups() returns the list of groups that are working on the project.
setGroup() is called to set the groups that will work on the project.
getGrade() return the grade of the project.
setGrade() is called to set the grade of the project.
getCourse() returns the course which the project belongs to.

setCourse() is called to set the project to its belonging course.
getAssignments() returns the list of assignments which are required to complete the project
setAssignments() is called to set the required assignments to complete the project.
addGroupToProject() is called to add a group to the groups list that will work on the project
removeGroupFromProject() is called to remove a group that will not work on the project
findGroup() is called to find the specific group that works on the project.

4.1.21 Review Assessment Aggregation

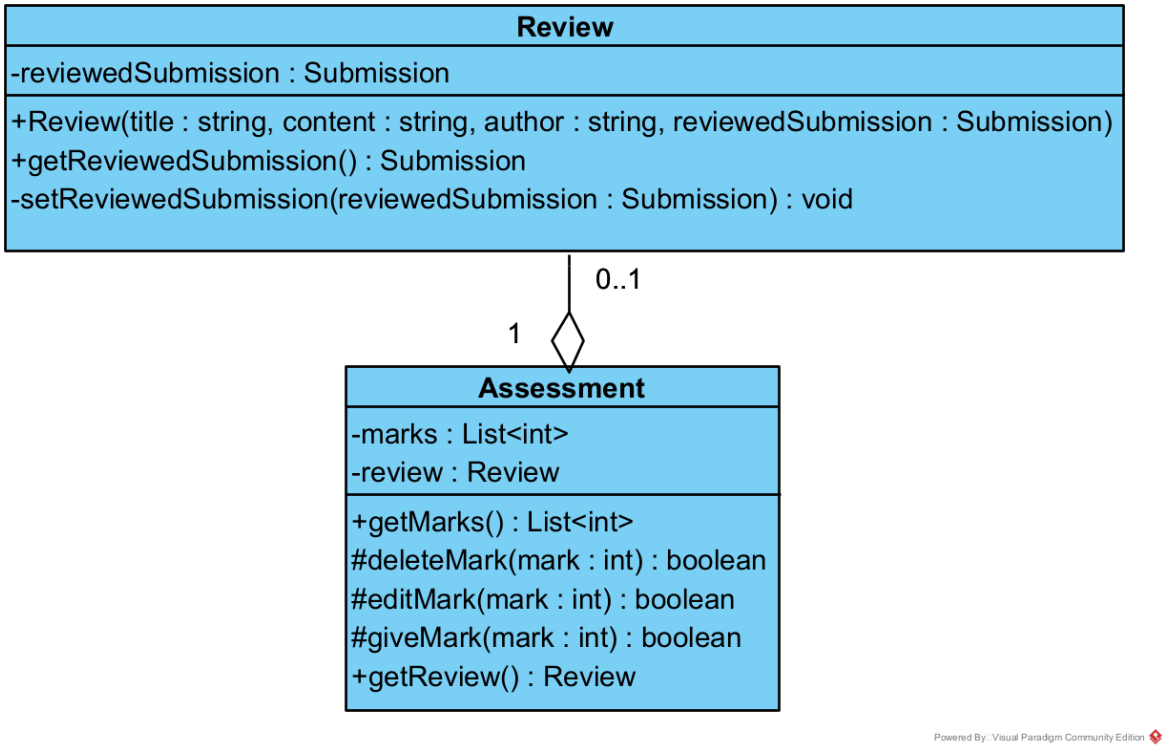


Figure 4.1.21 Review and Assessment Diagram

Assessments consist of Reviews. Thus, for an Assessment there may exist more Reviews.

4.1.22 Project Assignment CompletedSubmission Submission Relationship

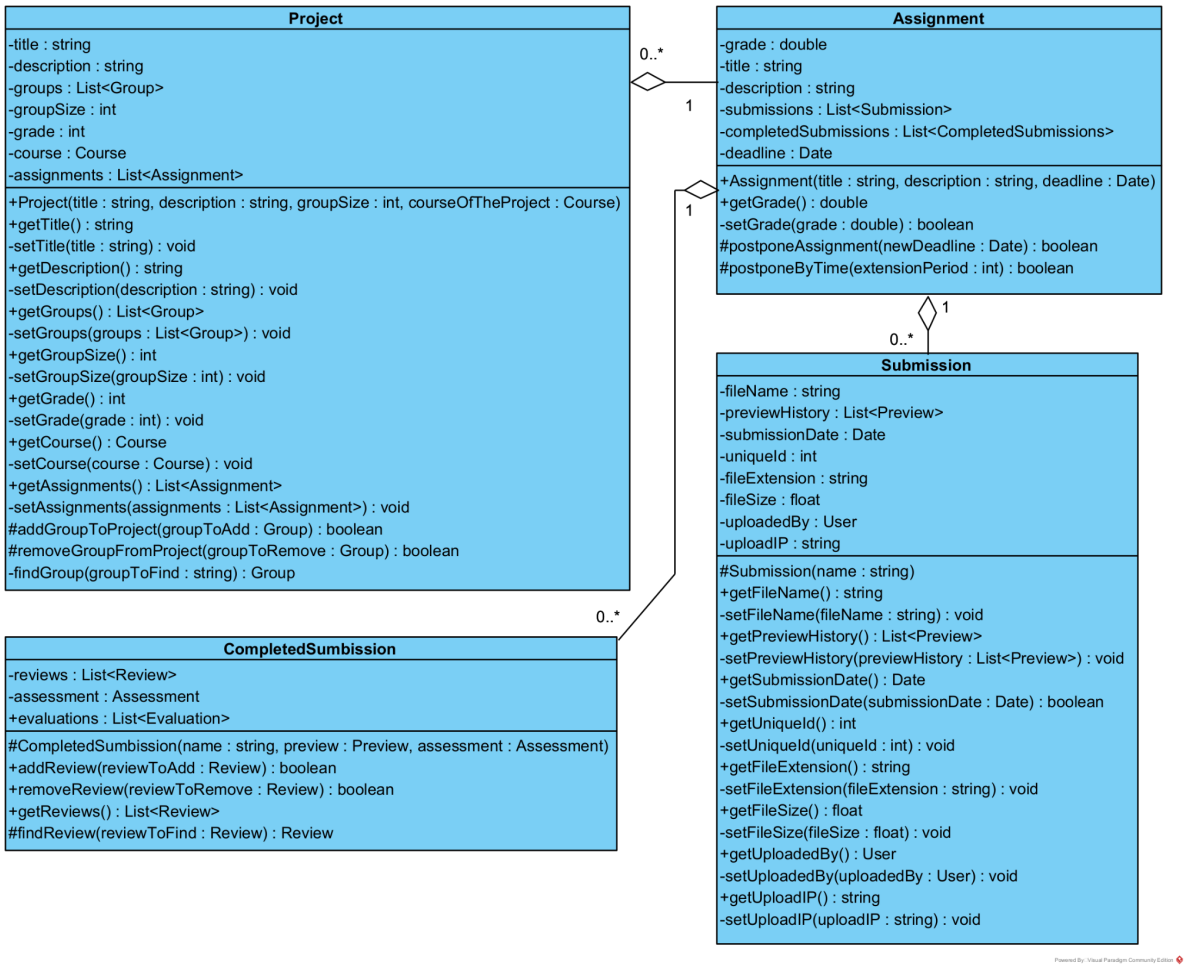


Figure 4.1.22 Diagram Depicting Relation Between Classes

Project contains Assignment that can be a submission or a CompletedSubmission which is an already end assignment.

4.1.23 Course Class Diagram

Figure 4.1.23 Course Class Diagram

Attributes

projects are the course's projects.
groups are the groups that are in the course.
students are the students taking the course.
courseName is the name of the course.
courseCode is the course code.
courseGroupPolicy is the limitations policy.
instructors is a list of instructors that prepares the course.
graders are the graders who work for the course.
submittedAssignments assignments that end.
assignments all the assignments.
groupController is the group controller for group related calls.

Methods

Course() is the constructor.
addProject() is called to add a project.
getProjects() returns the projects.
getGroups() returns the groups.
addStudent() adds student.
removeStudent() removes student.
getStudents() returns all the students.
findStudent() finds the student.
getCourseName() returns course name.
setCourseName() sets course name.
getCourseCode() returns course code.
setCourseCode() gets course code.
getCourseGroupPolicy() returns course group policy.
setCourseGroupPolicy() sets course group policy.
addInstructor() adds instructor.
removeInstructor() removes instructor.
getInstructors() gets all the instructors that give this lecture.
findInstructor() finds the instructor.
addGrader() adds grader.
removeGrader() removes grader.
getGraders() returns all the graders.
findGrader() finds the grader.
getSubmittedAssignments() returns submitted assignments.

4.1.24 Group Class Diagram

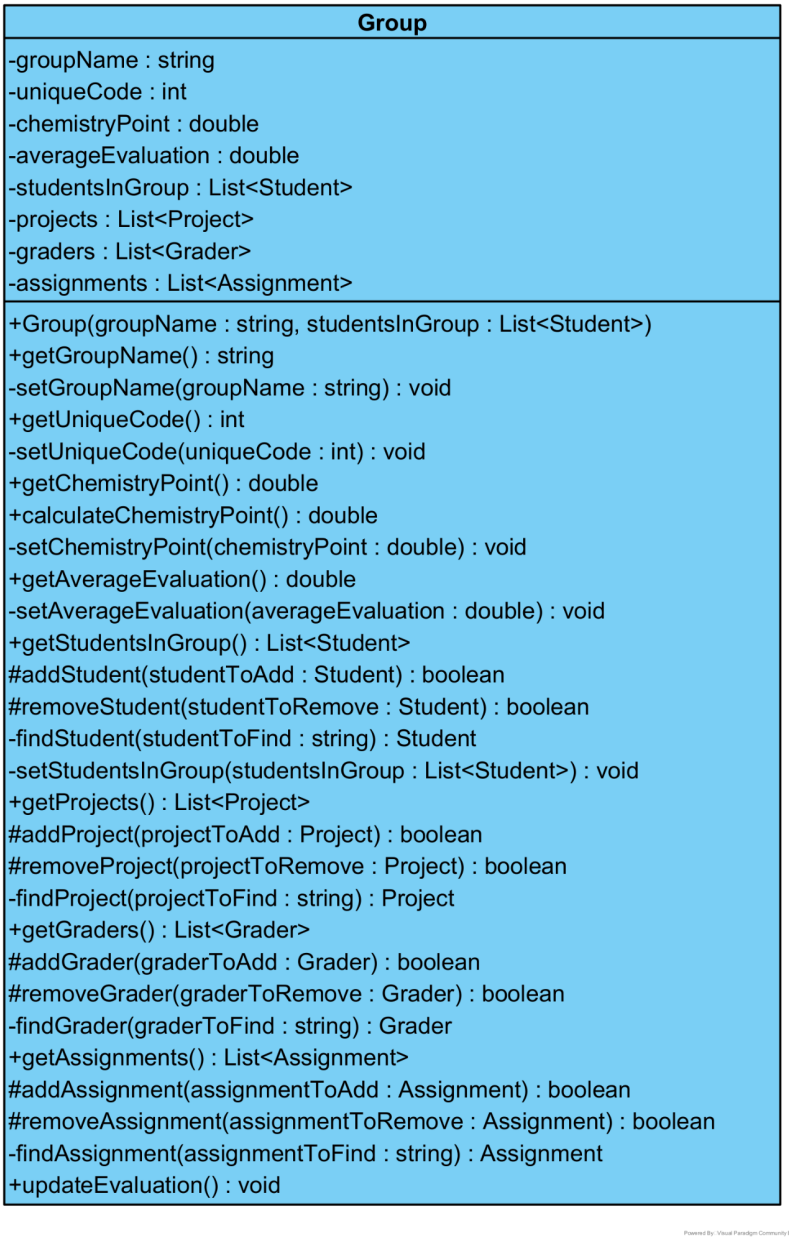


Figure 4.1.24 Group Class Diagram

Attributes
groupName is the name of the group.
uniqueCode is the group's unique code.
chemistryPoint is the group's chemistry point.
averageEvaluation is the group's average evaluation score.
studentsInGroup is the list of all the students in the group.
projects is the list of the all projects that are assigned to this group.
graders is the list of the graders that are assigned for this group.
assignments is the list of assignments that are assigned to this group.

Methods

Group() is the constructor for Group.
getGroupName() returns the group name.
setGroupName() takes a name and sets it to the group's name.
getUniqueCode() returns the group's unique code.
setUniqueCode() sets the group code.
getChemistryPoint() returns the chemistry grade.
calculateChemistryPoint() calculates the group's chemistry point.
setChemistryPoint() sets the group's chemistry.
getAverageEvaluation() returns the average evaluation of the group.
setAverageEvaluation() sets the average evaluation of the group.
getStudentsInGroup() returns the student list that is in the group.
addStudent() adds a student to a given group.
removeStudent() removes the given student from the group.
findStudent() searches for the given student name.
setsStudentsInGroup() is used to copy a group.
getProjects() returns the projects that are assigned to this group.
addProject() adds the project to the group's project list.
removeProject() removes the given project from the group's project list.
findProject() searches group's project list.
getGraders() returns the graders that are assigned to the group.
addGrader() adds a grader to the group.
removeGrader() removes a grader from the grader list.
findGrader() finds the grader in the graders list.
getAssignments() returns all the assignments assigned for this group.
addAssignment() adds a new assignment to the group.
removeAssignment() removes assignment from the given group.
findAssignment() finds the assignment in the assignment list.
updateEvaluation() updates the evaluation grade.

4.1.25 Singleton Class Diagram

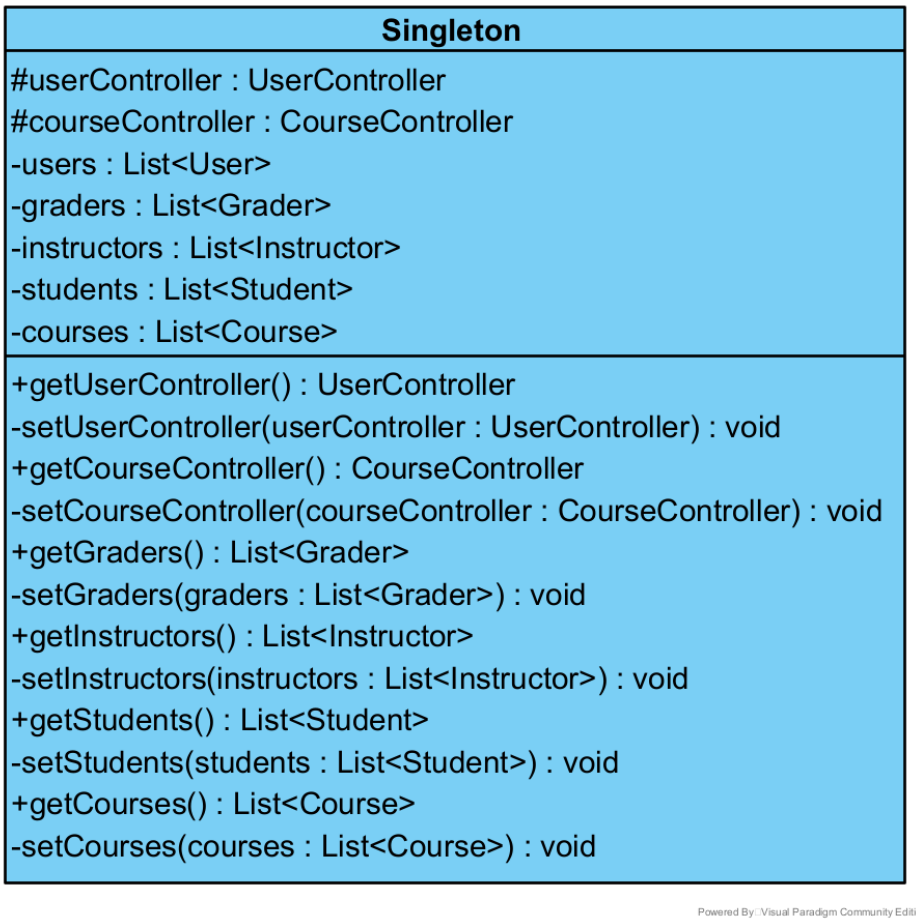


Figure 4.1.25 SingletonClass Diagram

Attributes
userController is the userController.
courseController is the courseController.
users is the list of all the users in the program.
graders is the list of all the graders in the program.
instructor is the list of all the instructors in the program.
students is the list of all the students in the program.
courses is the list of all the courses in the program.

Methods
getUserController() returns the userController, to control the user related calls.
setUserController() is expected to be used only in the singleton creation.
getCourseController() returns the courseController, to control the course related calls.
setCourseController() is expected to be called only in the singleton instantiation.
getGraders() returns the graders in the program.
setGraders() is a test function.
getInstructors() returns the instructors in the program.
setInstructors() is a test function.

getStudents() returns the students in the program.
setStudents() is a test function.
getCourses() returns the courses of all in the system.
setCourses() is a test function.

4.1.26 Singleton UserController CourseController Composition

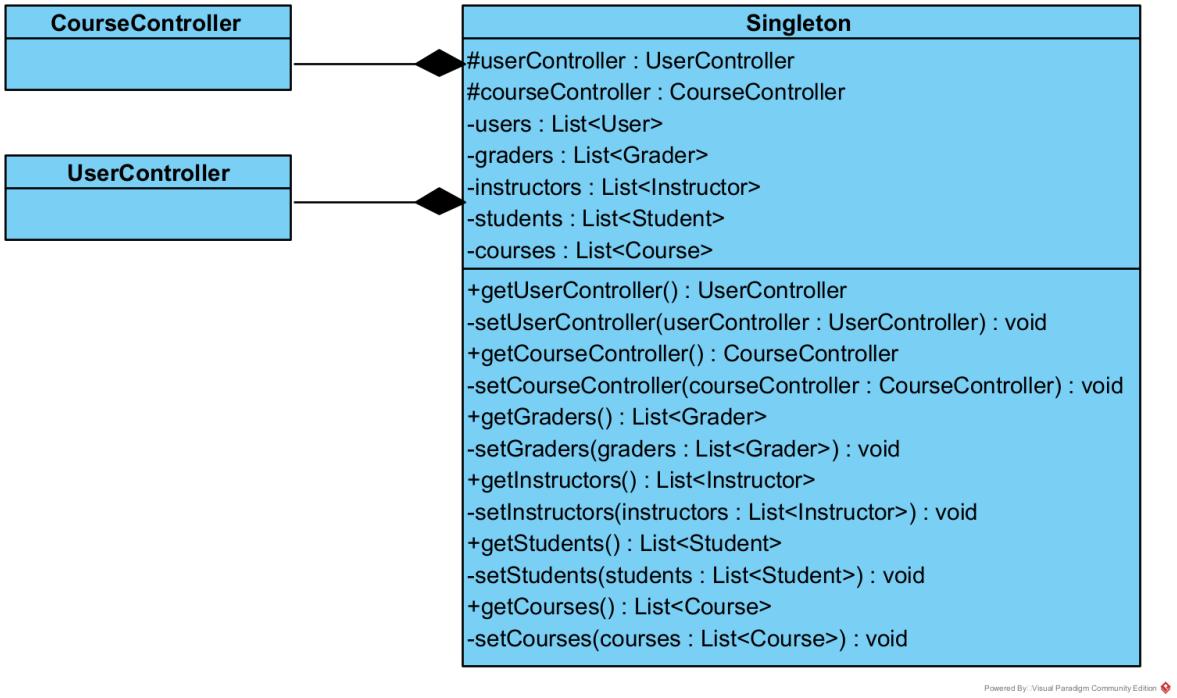


Figure 4.1.26 Singleton, UserController, CourseController Composition

CourseController and UserController are the working parts of the Singleton.

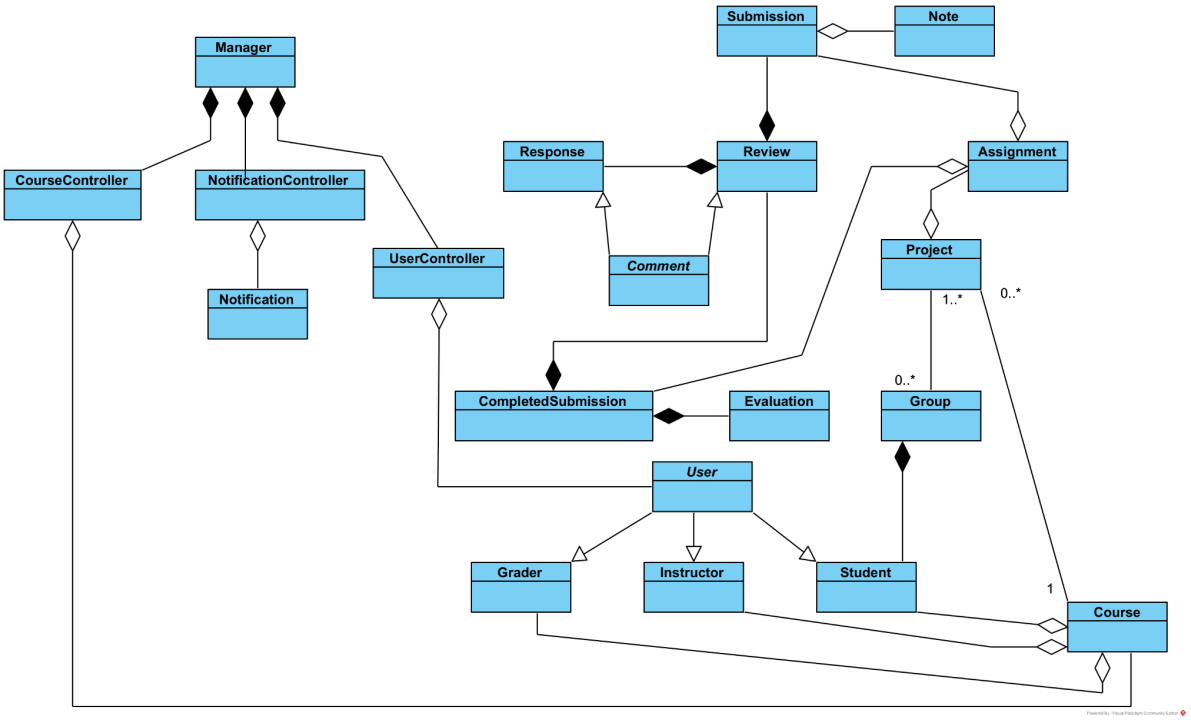


Figure 4.1.27 Overview of all classes and their relationships

4.1.27 Controller

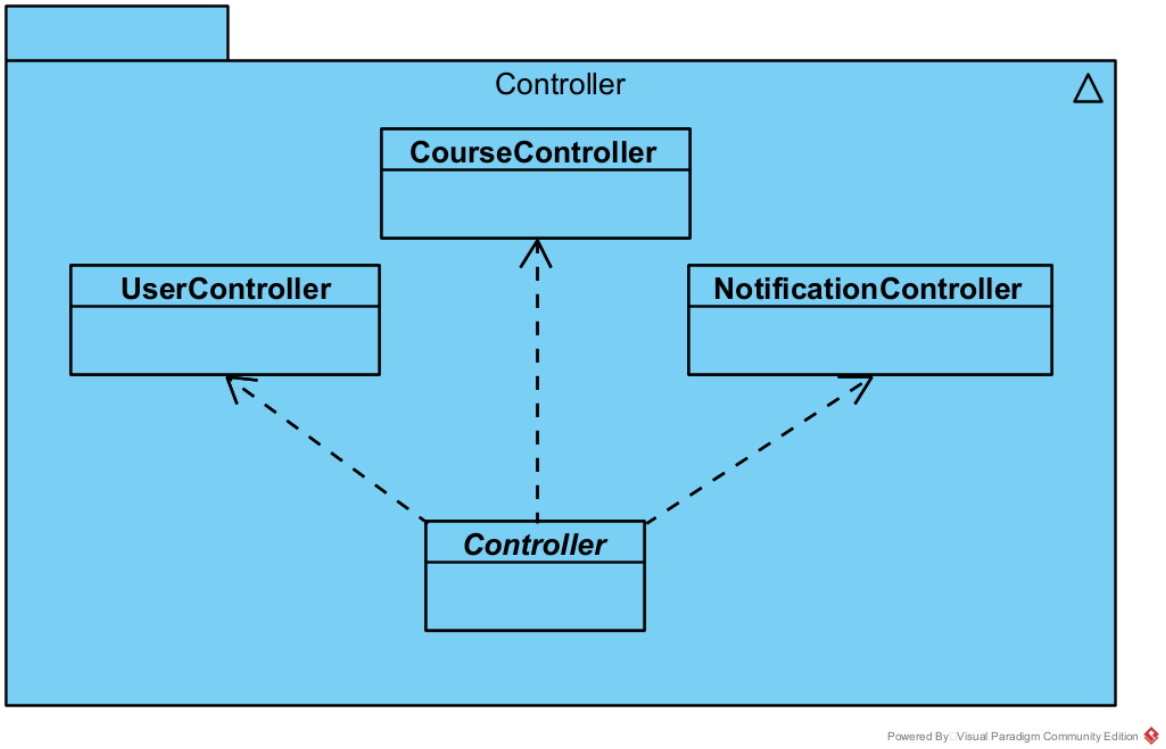


Figure 4.1.26 Controller Diagram

Controller design is based on a Controller abstract. According to our needs CourseController, UserController, NotificationController are prepared.

4.2 PACKAGES

4.2.1 NodaTime

NodaTime is a date and time API acting as an alternative to the built-in DateTime/DateTimeOffset etc. types in .NET.

4.2.2 NodaTime.Testing

Provides extra types which may be useful when testing code which uses Noda Time, such as a fake programmable implementation of IClock and a time zone which has a fixed transition. These types are also used to test Noda Time itself.

4.2.3 NodaTime.Serialization.JsonNet

Provides serialization support between Noda Time and Json.NET.

4.2.4 System.Data.SqlClient

Provides the data provider for SQL Server. These classes provide access to versions of SQL Server and encapsulate database-specific protocols, including tabular data stream.

5. Summary and Improvements

In our web application, PeeReview, which is used to evaluate group mates as well as in projects we prioritized some design goals, like usability, performance, extendability, and reusability. These goals were chosen to fit the main goal of this program, being used by students in future courses to evaluate each other in an easy and simple manner. Using C#9, ASP.NET framework, the architecture of the system is 3-tier, with presentation, application, and data each representing a tier. We also are using a client-server style. We are using a SQL database to store all data and retrieve it from there. Additionally, since the data is sensitive we kept the security and privacy into consideration, so each user type has different access to data with the instructor being able to view most of the data. The low level design reflects these values and gives a look at how we are planning to structure the code to provide the best service and keep it secure. The improvements can be seen with the sophistication and details added to the program. With more Object Oriented design patterns being applied to the structure of the system, like polymorphism and encapsulation stressed more to boost the reusability of the code, and make it easier and to understand. Additionally, the framework and the language versions were modified to fit the best and latest practices implemented. Finally, more dependencies were realized in sub-system decomposition and added to the architecture of the application.

6. Glossary and References

1. https://www.nuget.org/packages/NodaTime
2. https://www.nuget.org/packages/NodaTime.Testing
3. https://www.nuget.org/packages/NodaTime.Serialization.JsonNet
4. https://www.nuget.org/packages/Microsoft.Data.SqlClient/