Specification

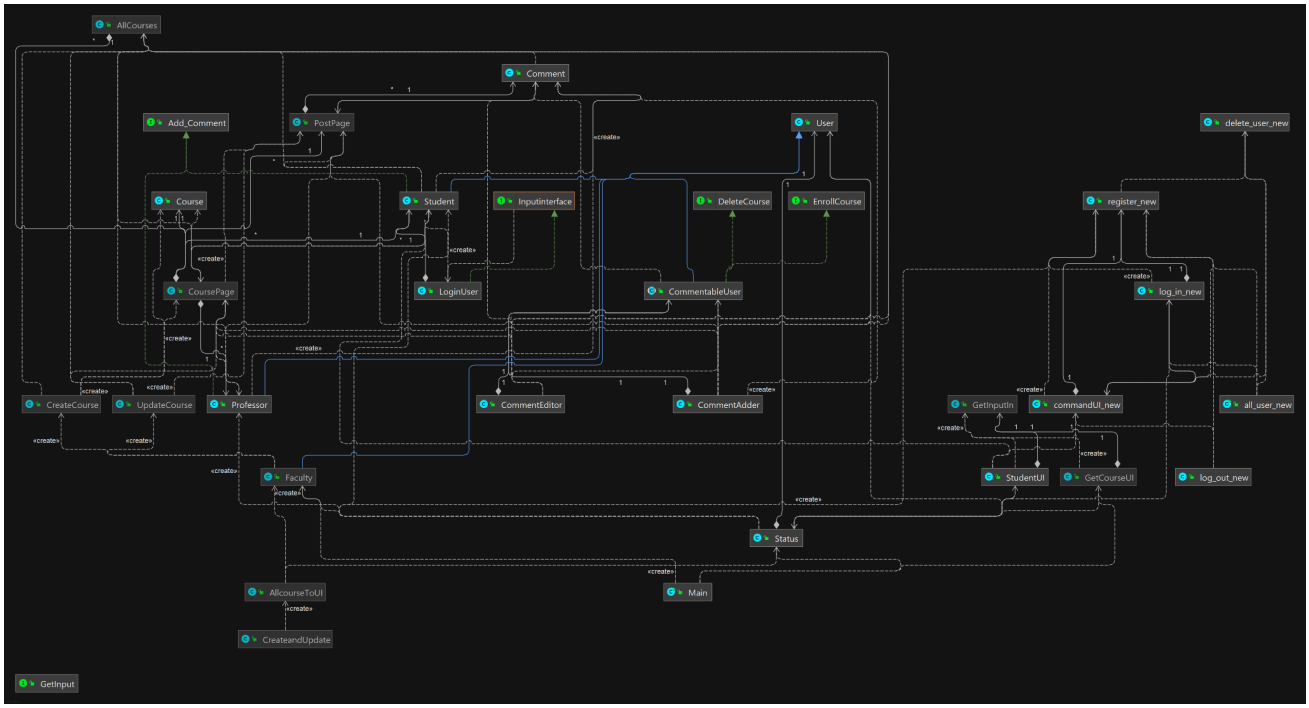
Our program is dedicated to provide a service for students to discuss course content, for faculty to post about course and program information, and for professors to post and delete comments. All three types of users should go through the registration process to access this program. Students will need their student id, and if already known, names of program enrolled and names of courses enrolled. Faculty and professors will need their staff id.

Students who enrolled in the same courses can post to the current course discussion board, reply to others' comments and delete their own comments. They also have access to the past post pages to learn from past students who previously enrolled in the course. They can also see other courses taught by their professors to learn about them.

Inside the course page, students can see course name, course information, professor information, current discussion board and the discussion boards in the past three semesters. Professor and faculty can see the same course page, with faculty being able to update course information. Faculty can create and update course pages. For professors, they have the right to delete comments and reply to the comments.

Clean Architecture

Class Diagram



The Dependency Rule has been followed in the implementation of our code .
Each project member has created classes carefully, making sure that code in inner layers of architecture does not depend on code from outer layers of architecture.

SOLID Design Principles

Single Responsibility Principle

- CreateCourse and UpdateCourse only create Courses and update Courses respectively
- Students and Professors are able to add, edit and delete Comments.
- These respective methods were extracted from these two classes and added to a new class CommentableUser.
- This CommentableUser class has two classes CommentAdder and CommentEditor which are responsible for adding and editing comments respectively.

Open Closed Principle

- Initially when trying to obey the Single Responsibility Principle, a use case class CommentDeleter, responsible for deleting classes, was created.
- However, as the implementation of the deleteComment method contained if statements that check whether the stored CommentableUser class was an instance of Student or Professor, this violated the Open Closed Principle as wanting to implement other CommentableUsers would require more if blocks being added.
- The solution was to remove this use case class entirely and move the deleteComment method back into each subclass of CommentableUser.
- Faculty has delegated responsibilities to other classes, easily allowing for extension

Liskov Substitution Principle

- The User class is the superclass of all Users created when a user runs the program.
- The User has minimal methods apart from getters and setters so that subclasses that do not require these methods do not have to implement them
- All subclasses of User can substitute for an instance of User

Design Patterns

Template Method Design Pattern

- Our algorithm has the classes Student and Professor which share the same functionality of adding, deleting and editing comments from PostPages which are both children of User.
- The implementation is creating a new subclass called CommentableUser that extends the User superclass that has Comment functionality.
- This design pattern choice does not impact other functionality as Student and Professor separate instance variables that could be used in other methods.

Facade Design Pattern

- Created a facade class called Faculty which is responsible for both creating courses and updating course information
- Delegate the createCourse method to a new CreateCourse class
- Delegate the updateCourseInfo method to a new UpdateCourse class
- The instance variables of type CreateCourse and type UpdateCourse will be contained in the Faculty class and will be used to call the associated methods.
- This helps to keep our code in the Faculty Class simpler and encapsulates more of the implementations.

Progress Report

Hanqi Zhang:

- Faculty methods (create a new course, create a new post semester, edit course info)
- Implemented the sub UI for faculty and wrote the Status class for linking to different UI(i.e Faculty, Student, Professor)
- Student UI implementation

Chuanyang Qiao:

- Responsible for all materials related to users.
- Implemented main menu UI.
- Implemented a register function that can save a new user's information to a csv file according to their type-in username, password, and type(student, professor, faculty).
- Implemented a log in function that can check the csv file according to the user's input and decide whether to log them in or not.
- Implemented a quit function that can save all information and quit the program.
- Implemented administrator privileges functions(need a valid developer password) that can check all users currently registered or delete the whole database.

Junhao Saw:

- Design Document
- Implemented CommentableUser class and Template Design Pattern
- Created CommentAdder and CommentEditor classes

Kuang Jiang:

- Implemented the Student method enrollCourse and deleteCourse
- Wrote the StudentUI class
- Implemented the Professor methods enrollCourse and deleteCourse

Zeyuan Li:

- Student and Professor methods for adding, editing and deleting comments

Zhijun Wang:

- Added presenter function into Student and Professor classes
- Implemented UI

Areas for Improvement and Future Implementation

Clean Architecture

- Make more Use Cases inputs and outputs to stop violating the dependency inversion principle
- Create a GUI

SOLID Design Principles

- Make sure all classes obey the single responsibility principle
- Lots of UI classes have large and complex if statement blocks violating the open closed principle
- There is an attempt at creating a Status class so that the User class can be accessed directly but the dependency inversion principle could be followed more rigorously
- Separate class methods into different interfaces/classes to follow single responsibility principle.

Design Patterns

- Implement appropriate design patterns

Testing

- Implement all tests rigorously
- Use IntelliJ features to see which lies of code have been tested

Code Style and Documentation

- Use correct code style
- Write more comments and document when coding
- Stop using Python naming convention

Functionality

- Implement recursive adding/deleting of comments
- Use UUID to designate unique User/Comments

Refactoring

- Long method code smell in several UI classes
- Switch statement code smell in several UI classes
- Speculative Generality code smell for GetInput interface
- Long class code smell for resgister_new (also change naming convention)

Use of Github Features

- Use more Github features such as issues

Code Organization

- Implement a packaging structure