Project Part 7: Final Report

1- Project Team Name and #. List of team members. Vision. Project description.
   a- Project Team Name: Schoodle
   b- Team Members: Ahmed Al Hasani, Erik Holbrook
   c- GitHub Accounts: @AhmedAlHasani, @Zephyr1999
   d- Vision: Create an application that requires numerous classes to incorporate Object-Oriented Principles and the course's concepts.
   e- Description: a web-based application for college course-related tasks, like tracking grades, sharing content and submitting assignments.

2- List the features that were implemented (table with ID and title).

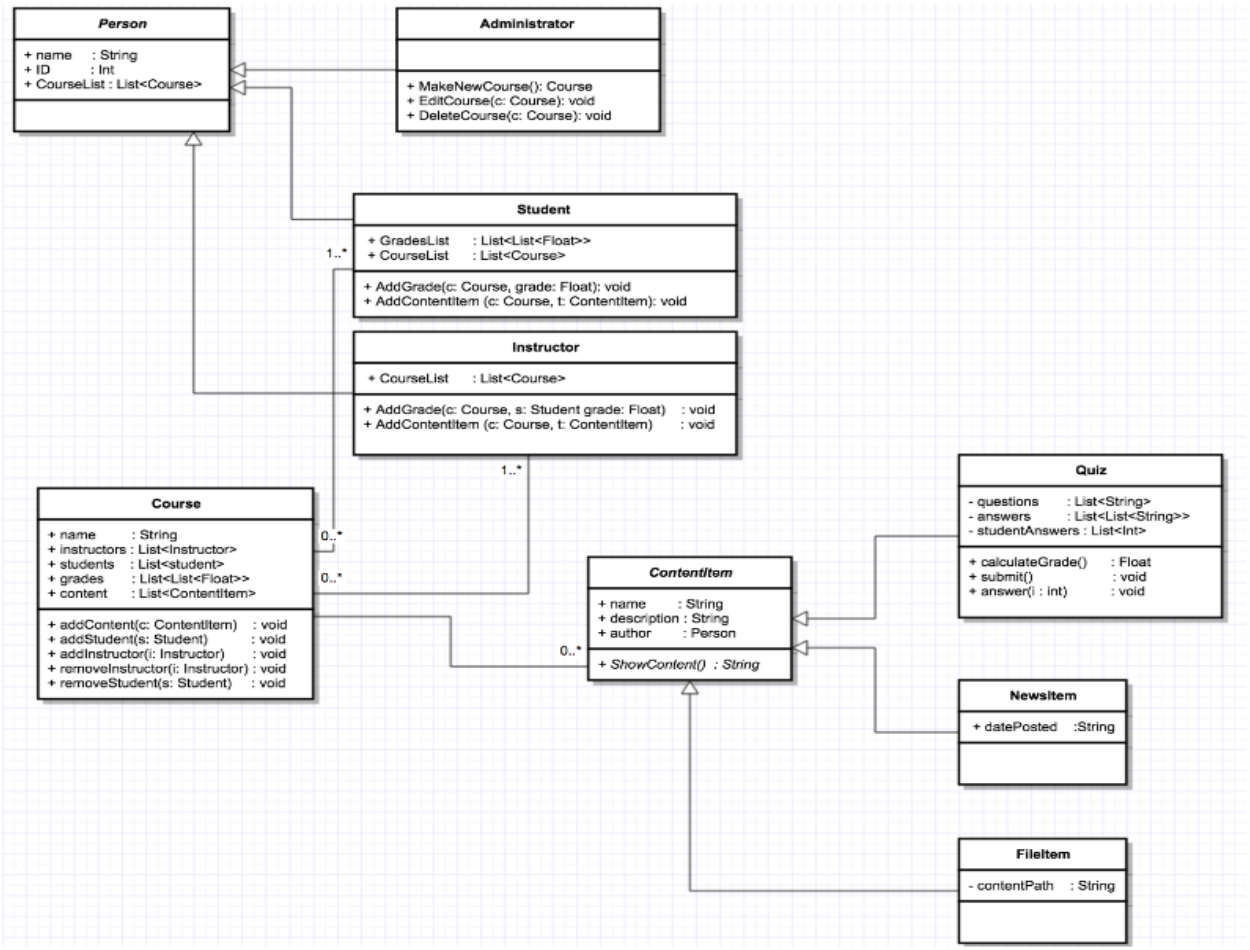
ID	Description
1	An administrator can create a new Course and populates it with at least 1student and 1instructor
2	An administrator can add students and instructor to a course
3	An instructor can upload multiple students' grades
4	An instructor can upload course content
6	A student can view course content
7	A student can view their course grades
9 (Partially)	Students can take automatically-graded quizzes. Our current implementation shows only the view of the quiz but without the questions (model) and the control.

3- List the features were not implemented from Part 2 (table with ID and title).

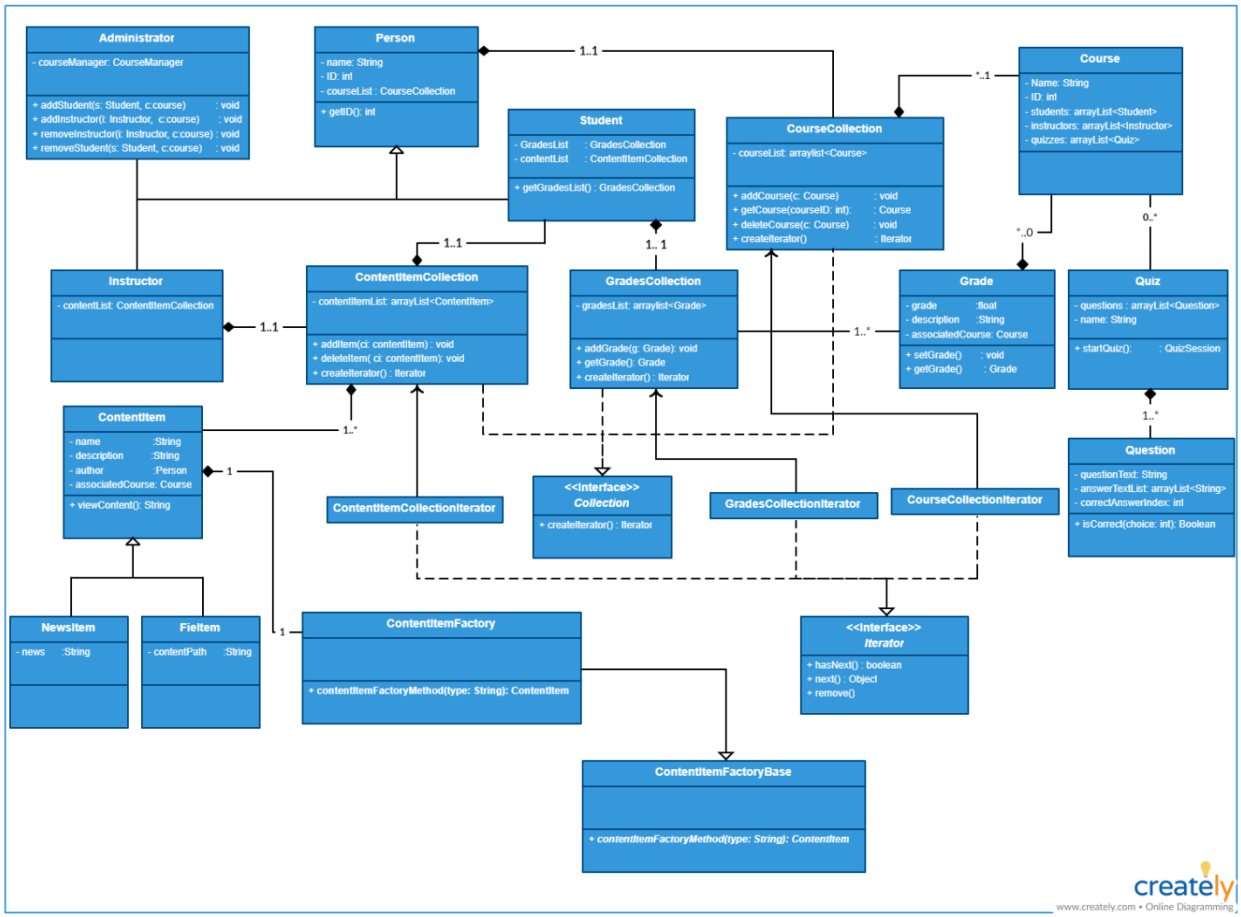
ID	Description
5	An instructor can download students' assignments and grade them
8	Students can submit assignment content (PDFs) for grading

4- Show your Part 2 class diagram and your final class diagram.
What changed? Why? If it did not change much, then discuss how doing the design up front helped in the development.

Part 2 Class Diagram



Final Class Diagram



As you can see, the first-class diagram is significantly different from our last class diagram. There are many changes and they include:

a- Two design patterns were included in the final class diagram, the iterator and the factory design patterns. We implemented the iterator design patterns because it fit our need in retrieving a list, or in our case, a collection of items that include grades, courses, and content items that students and instructors' objects might need. On the other hand, the factory design pattern was implemented to allow creating content items dynamically by the instructors. If we did not implement design patterns, we believe we would have been required to write more code than what we implemented. The iterator design pattern allowed us to iterate through different objects as well.

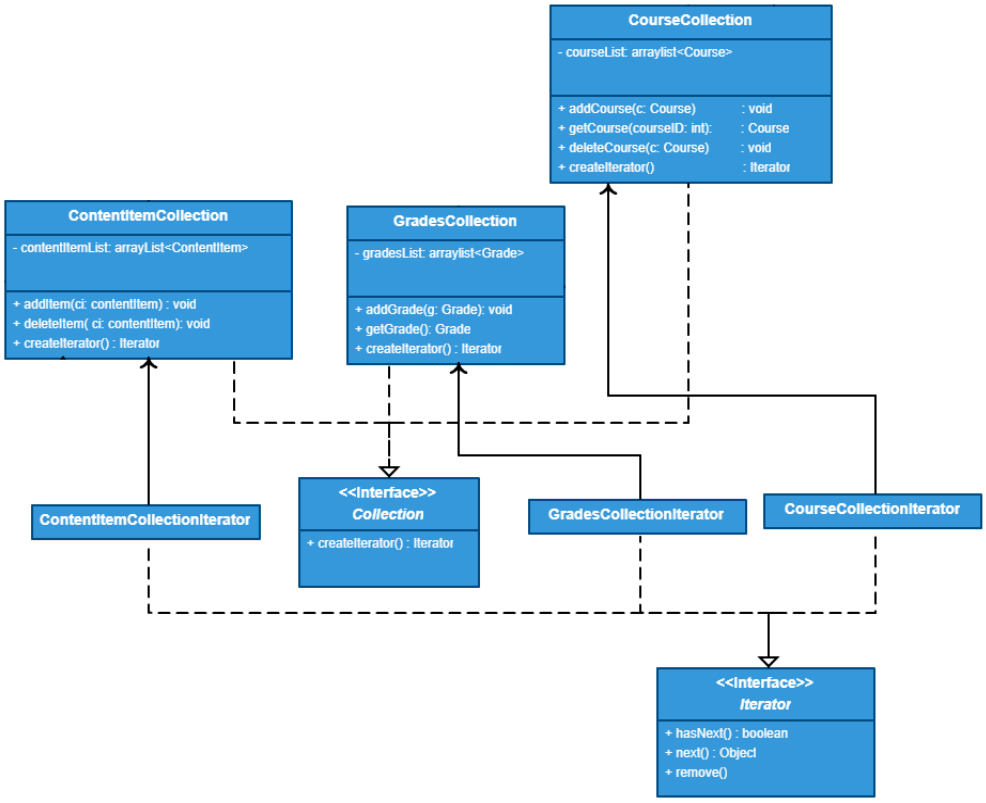
b- Blob classes in the first design were changed to classes that follow the single responsibility principle. For instance, we extracted grades related tasks from the Course class in the first class diagram and created a separate Grade class. This allowed us to modify and extend classes easily instead of having to refactor and redesign our system when we implemented the classes and the design patterns. We further decoupled responsibilities that were placed inside the Student and Instructor class and placed them in more appropriate classes.

c- We did not reflect relationships between classes accurately because we did not understand relationships initially. Once we separated responsibilities from blob classes to single classes, we understood how the smaller classes depended on one another and the type of relationship they required. As a result, we reflected that in the final class diagram, and our final implementation in Django went smoothly as compared to the implementation reflected from the initial class diagram.

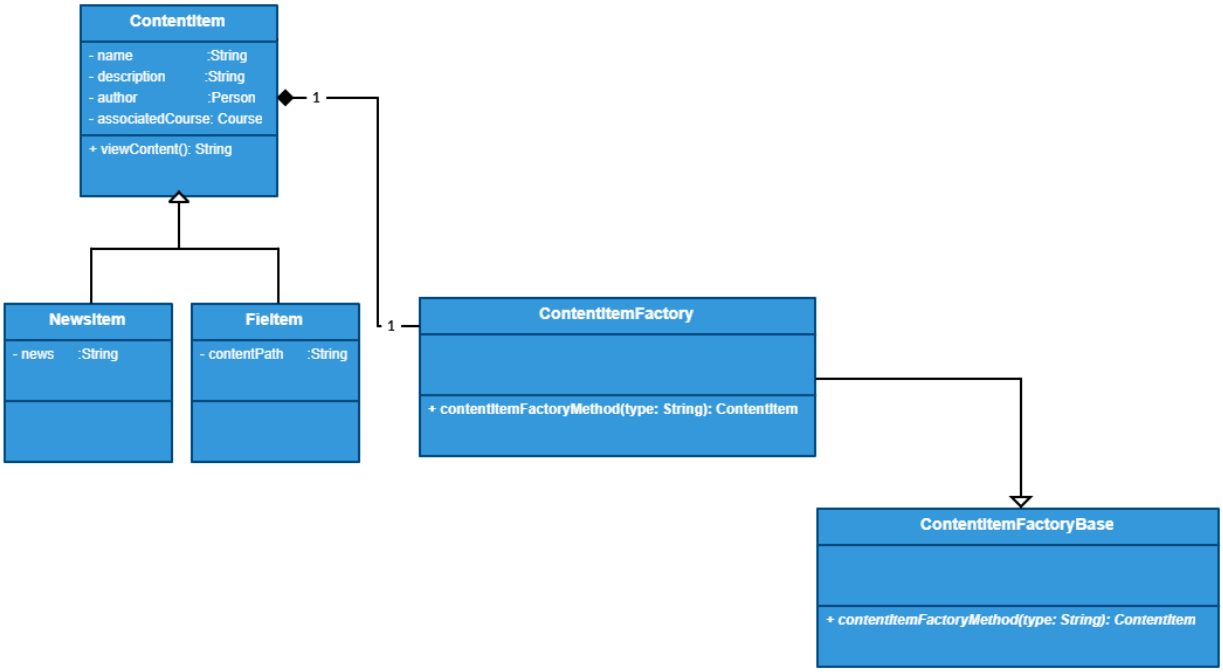
d- Our initial class diagram did not reflect our proposed use cases. For example, the Course class initially included removeInstructor and removeStudent. While it is obvious we want to remove students as we want to add them, our use case tables did not include removing students, therefore, we design a class diagram that strictly followed our proposed use cases.

5- Show the classes from your class diagram that implement each design pattern (each design pattern as a separate image in the .PDF).

a. Iterator



b. Factory



6- What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system.

It is easy to include many methods in a single class and initially think they are a suitable fit together. Therefore, we created blob classes unintentionally. When we learned about SOLID principles and through careful analysis, we realized the classes violated the single responsibility principle. We accidentally created a class diagram that reflected low cohesion and high coupling. Additionally, when we understood the disadvantages from our initial design which included long hours of refactoring code, we modified our diagram to implement classes that have high cohesion and low coupling which will aid us greatly in the implementation stage. We extended our system without having to greatly modify our classes.

Once we designed a more appropriate class diagram, it was easy to implement design patterns and understand how they can fit our project. Our classes were small, and followed the single-responsibility principle, therefore, we were able to modify our project easily then to fit the design patterns implemented.

Additionally, incorporating design patterns reduced implementing unnecessary classes and code. This reduced the unnecessary work load in implementing numerous incorrect and useless classes. It also allowed for more code reuse, for example, the iterator design pattern allowed iterating through multiple objects. We were able to specify the design patterns needed because we understood their intents. Consequently, we combined them with each use case's purpose and outcome and how the project was connected internally.

Lastly, when we were planning to incorporate polymorphism, it allowed us to easily approach restricting access late in the project. For instance, everyone has access to their courses, but only instructors can add grades (special methods) and only admins can add students/ instructors to a course.

We were able to implement the classes in Django and connect them together in the models python file easily because the classes were modified through extensive class diagram redesigns. This emphasizes how a good plan and design reduces execution significantly. It taught us the importance of how meticulous planning eases implementation.