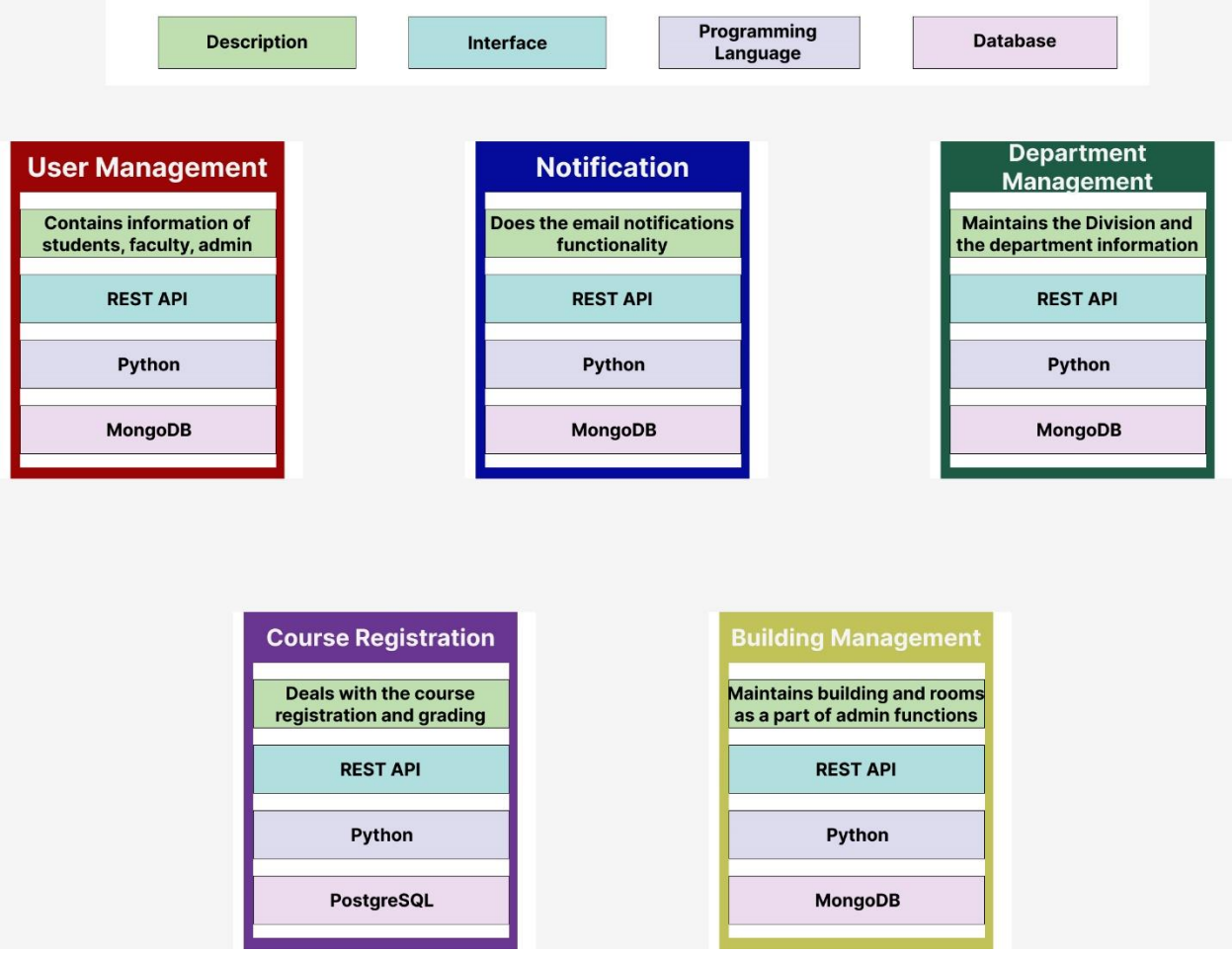
MPCS51410 - Object Oriented Programming

Final Project Deliverable

Thanushri Rajmohan

Bounded Context Diagram

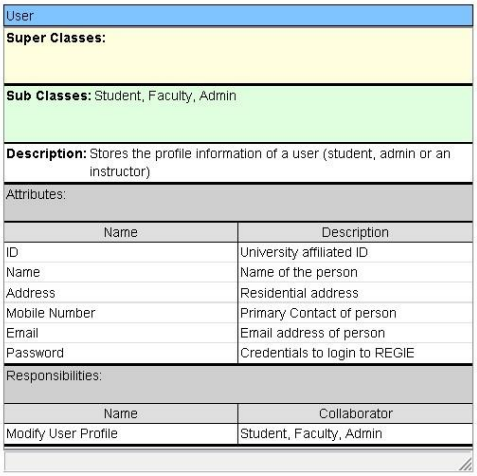
Link to the bounded context diagram:
https://www.figma.com/file/REPgNUSsxwvm0InHykLP6h/Untitled?node-id=0%3A1&t=1kcxbQbdNM6u0qjV-0



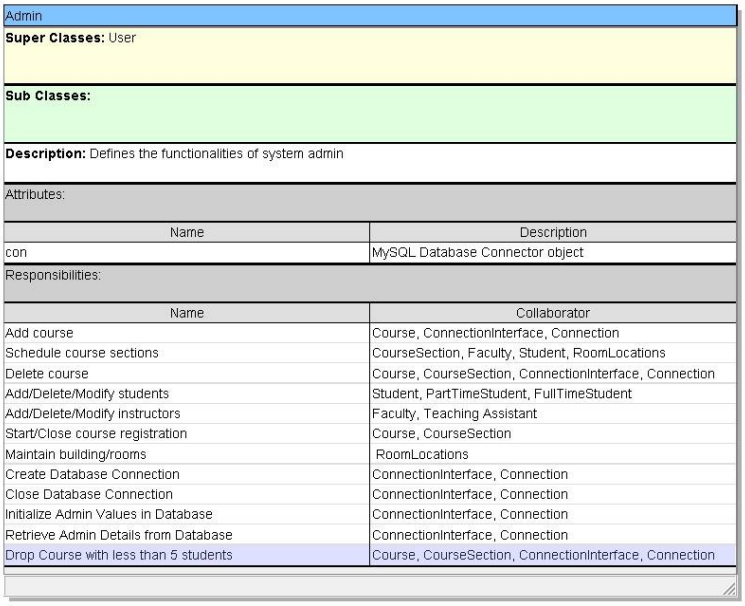
In this final project of the REGIE system I don't see any changes that my bounded context diagram requires. My implementation of the REGIE system focuses on the User Management, Department Management, Course Registration contexts and these bounded contexts are designed perfectly alright and don't require any changes. I have come up with UML CRC diagram designs for the "Course Registration", "User Management" and "Department Management" bounded contexts.

CRC Diagrams

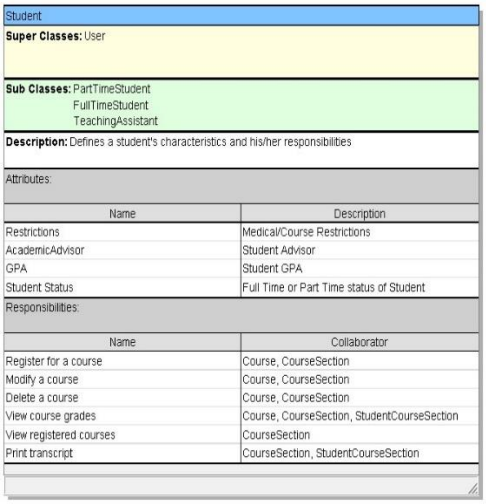
User Management:



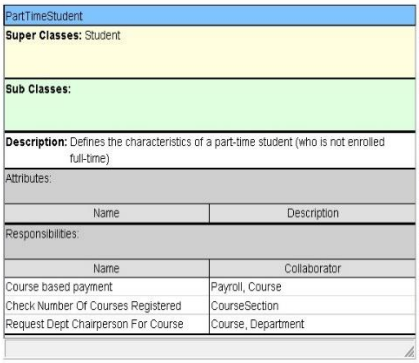
User	
Super Classes:	
Sub Classes: Student, Faculty, Admin	
Description: Stores the profile information of a user (student, admin or an instructor)	
Attributes:	
Name	Description
ID	University affiliated ID
Name	Name of the person
Address	Residential address
Mobile Number	Primary Contact of person
Email	Email address of person
Password	Credentials to login to REGIE
Responsibilities:	
Name	Collaborator
Modify User Profile	Student, Faculty, Admin



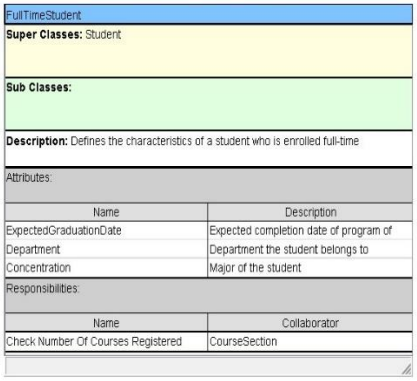
Admin	
Super Classes: User	
Sub Classes:	
Description: Defines the functionalities of system admin	
Attributes:	
Name	Description
con	MySQL Database Connector object
Responsibilities:	
Name	Collaborator
Add course	Course, ConnectionInterface, Connection
Schedule course sections	CourseSection, Faculty, Student, RoomLocations
Delete course	Course, CourseSection, ConnectionInterface, Connection
Add/Delete/Modify students	Student, PartTimeStudent, FullTimeStudent
Add/Delete/Modify instructors	Faculty, Teaching Assistant
Start/Close course registration	Course, CourseSection
Maintain building/rooms	RoomLocations
Create Database Connection	ConnectionInterface, Connection
Close Database Connection	ConnectionInterface, Connection
Initialize Admin Values in Database	ConnectionInterface, Connection
Retrieve Admin Details from Database	ConnectionInterface, Connection
Drop Course with less than 5 students	Course, CourseSection, ConnectionInterface, Connection



Student	
Super Classes: User	
Sub Classes: PartTimeStudent FullTimeStudent TeachingAssistant	
Description: Defines a student's characteristics and his/her responsibilities	
Attributes:	
Name	Description
Restrictions	Medical/Course Restrictions
AcademicAdvisor	Student Advisor
GPA	Student GPA
Student Status	Full Time or Part Time status of Student
Responsibilities:	
Name	Collaborator
Register for a course	Course, CourseSection
Modify a course	Course, CourseSection
Delete a course	Course, CourseSection
View course grades	Course, CourseSection, StudentCourseSection
View registered courses	CourseSection
Print transcript	CourseSection, StudentCourseSection



PartTimeStudent	
Super Classes: Student	
Sub Classes:	
Description: Defines the characteristics of a part-time student (who is not enrolled full-time)	
Attributes:	
Name	Description
Responsibilities:	
Name	Collaborator
Course based payment	Payroll, Course
Check Number Of Courses Registered	CourseSection
Request Dept Chairperson For Course	Course, Department



FullTimeStudent	
Super Classes: Student	
Sub Classes:	
Description: Defines the characteristics of a student who is enrolled full-time	
Attributes:	
Name	Description
ExpectedGraduationDate	Expected completion date of program of
Department	Department the student belongs to
Concentration	Major of the student
Responsibilities:	
Name	Collaborator
Check Number Of Courses Registered	CourseSection

Faculty

Super Classes: User

Sub Classes: TeachingAssistant

Description: Defines the characteristics of an instructor along with his/her rank

Attributes:

Name	Description
Rank	Position/Rank of instructor
Department	Department the instructor belongs to
Status	Full-Time or Part-Time status of the instructor

Responsibilities:

Name	Collaborator
View current/past course schedule	CourseSection
View current/past class rosters of registered	CourseSection
Add/Modify student grades	StudentCourseSection, Student
Approve/Deny registration requests	CourseSection, Student, Notification
View course grade spreadsheet	CourseSection, StudentCourseSection
Sending email to registered students	CourseSection, Notification
Create Database Connection	ConnectionInterface, Connection
Close Database Connection	ConnectionInterface, Connection
Initialize faculty details in Database	ConnectionInterface, Connection
Retrieve faculty details from Database	ConnectionInterface, Connection
Viewing faculty department information	Department, ConnectionInterface, Connection

TeachingAssistant

Super Classes: Student, Faculty

Sub Classes:

Description: Defines the responsibilities (that includes both faculty and student classes) of a Teaching Assistant

Attributes:

Name	Description

Responsibilities:

Name	Collaborator

Course Management:

Course

Super Classes:

Sub Classes:

Description: Defines the attributes and the responsibilities of a course

Attributes:

Name	Description
ID	Unique ID of the course
Name	Name of the course
Description	Brief description of the course
Department	Department that is offering the course
Fee	Course Fee

Responsibilities:

Name	Collaborator
Modify Course Details	Admin
Add/Delete Course Details	Admin

CourseSection

Super Classes:

Sub Classes:

Description: Includes details of the course sections of a course

Attributes:

Name	Description
CourseSectionID	Unique ID for course section
FacultyID list	Unique ID of the instructors
RoomID	Room assignment for the course
CurrentEnrollmentCount	Live count of the students enrolled
Timings	Course Section Timing
Day	Day of course section
StudentID list	Students in the course section
PermissionRequired	Boolean variable
Features	Course Features
QuarterID	The quarter the course section is being offered
CourseID	ID of the course

Responsibilities:

Name	Collaborator
View Enrolled Students	Faculty
Add/Delete Student	Faculty, Admin
Add/Delete Instructor	Admin
Schedule courses in rooms	Admin
Delete course sections	Admin
Add/Delete/View Features	Admin, Faculty

StudentCourseSection or Grade Management, Department Management and Building Management:

StudentCourseSection

Super Classes:

Sub Classes:

Description: Includes the details of students in a course section

Attributes:

Name	Description
StudentID	Unique ID of student
CourseSectionID	Unique ID for course section
Grade	Grade of student in course
GPA	Average GPA
QuarterID	Quarter of CourseSection
AssignmentScores	Student scores in Assignments

Responsibilities:

Name	Collaborator
Assign/Modify student course grades	Faculty, TeachingAssistant
View course grade spreadsheet	Faculty, TeachingAssistant, CourseSection
Calculate average GPA	Student
Print Transcript	Student

RoomLocations

Super Classes:

Sub Classes:

Description:

Attributes:

Name	Description
RoomID	ID of the room
RoomName	Name of the room
Maximum Capacity	Capacitiy of the room
Building	Name of the building

Responsibilities:

Name	Collaborator
view Maximum Capacity	Admin, CourseSection
Add/Modify/Delete rooms	Admin
Schedule Rooms	Admin, CourseSection

Department

Super Classes:

Sub Classes:

Description:

Attributes:

Name	Description
Dept ID	Id of the department
Name	Department Name
Division ID	Division ID of the department
Division Name	Division the dept belongs to

Responsibilities:

Name	Collaborator
Viewing Department information	Faculty
Add/Modify/Delete Department	Admin
Add/Modify/Delete Divisions	Admin

The CRC diagrams have minute alterations from the previous iteration regarding the attributes and the operations/responsibilities of each class.

Coding and Testing:

In my previous iterations, I had identified the classes that I would be working on, and I had produced coded my classes appropriately to pass the tests with main focus on the classes of the User Management System such as User, Admin, Faculty, Student, etc. and Course Management system. I have implemented extensive Object-Oriented Design Principles in those such as Abstract Factory method design pattern for User Management, State pattern for Course states, Template design pattern for Users, Composite design pattern for course grading. The source code has a pretty organized structure of my User Management as well as the classes and patterns of the Course Registration System integrated with the MySQL database using python MySQL-database-connector. I also used MongoDB database for the purpose of logging. For a better look at the coding deliverable and the tests, with the database part integrated, take a look at the /src folder of the project.

<u>Implementation of S.O.L.I.D principles in the source code:</u>

- Single Responsibility Principle
  Single – responsibility principle (SRP) brings into attention that each class should have only one responsibility and necessarily only one reason to change. In my REGIE system design, I have the Course Registration and the User Management System bounded contexts. We see that in these bounded contexts, the following classes exist and each of those classes strictly follow the single – responsibility principle where they have only one purpose. This makes the upcoming REGIE system more modular and less prone to errors.

Class	Purpose
User	Responsible for managing user profile information
Admin	Responsible for managing admin-specific operations and responsibilities.
Faculty	Responsible for managing faculty-specific information and operations.
Student	Responsible for student specific operations such as registering/dropping courses.
FullTimeStudent	Responsible for storing information of full-time students like expected graduation date, etc.
PartTimeStudent	Responsible for the part-time student operations.
Course	Responsible for adding/deleting courses to the database and storing course-specific information.
CourseSection	Responsible for managing course sections of each course and contains information of students and instructors of that course section.
StudentCourseSection	Responsible for storing the grading information of students in a course section.

None of the purposes in a class overlap with the responsibilities of another class. Thus, REGIE system design adheres to Single Responsibility Principle.

- Open – Closed principle
  The REGIE system follows Open/Closed principle especially in the User Management system design of Students. Students consist of the common student responsibilities, and they are further extended to Full time students and Part time students to perform student status specific operations. This design that allows entities which are clearly abstracted to easily include additional methods for other entities is called the Open Closed Principle. If you refer to the class diagram, it will be evident that a Full Time Student extends Student and implements Full Time Student Abstract Factory Interface. This is an example of Open/Closed principle where Student class is extended to implement newer functionalities without being modified directly.

- Liskov Substitution Principle
  This states that the subclass should be used instead of the base class and still there shouldn't be anything that causes an issue in the REGIE system. This is being followed in the Student and the Full time Student class of the User Management module. All the functionalities of

the student can be replaced and used by a Full-time student object, and they would still work without changing the behavior of the system. It can also be seen that the assumptions of the Full-time student class does not vary with the assumptions of the Student class (for example, the input and the output format, etc.). Both the classes are consistent and thus follow the Liskov Substitution Principle for Object Oriented Design.

- Interface Segregation Principle
  This principle mentions that interfaces should be designed to meet the specific needs of clients rather than providing only a broad range of method operations that might not be relevant to all clients. In the REGIE system, it is seen that (from the class diagram), there is a UserFactory interface that has only the create_user() method in common for all users and then there is a separate FacultyFactory and StudentFactory that creates Faculty and Student (Part-time and Full time students). Thus, specific interfaces are defined for specific purposes and the general interface UserFactory has only the common methods. Thus, REGIE system follows the Interface Segregation Principle.

- Dependency Inversion Principle
  The database connector code has a high level of abstraction in its design with a class that implements the Connection Interface. (Refer to class diagrams in Deliverable 2 for the implementation structure). This class is designed separately from all the other bounded context classes so make sure the connection is an independent class altogether. The connection class's object is passed to the other classes for performing database related operations. The other classes do not need to know anything about the design of the Connection class thus adhering to abstraction and independency. These classes don't necessarily know the type of the connection (if it's a MySQL or NoSQL) connector. So, it is possible, sometime later in the future, to modify and use a different database than MySQL and still nothing would change in the system design. This is called the Dependency inversion principle which is strictly followed in the REGIE system design.

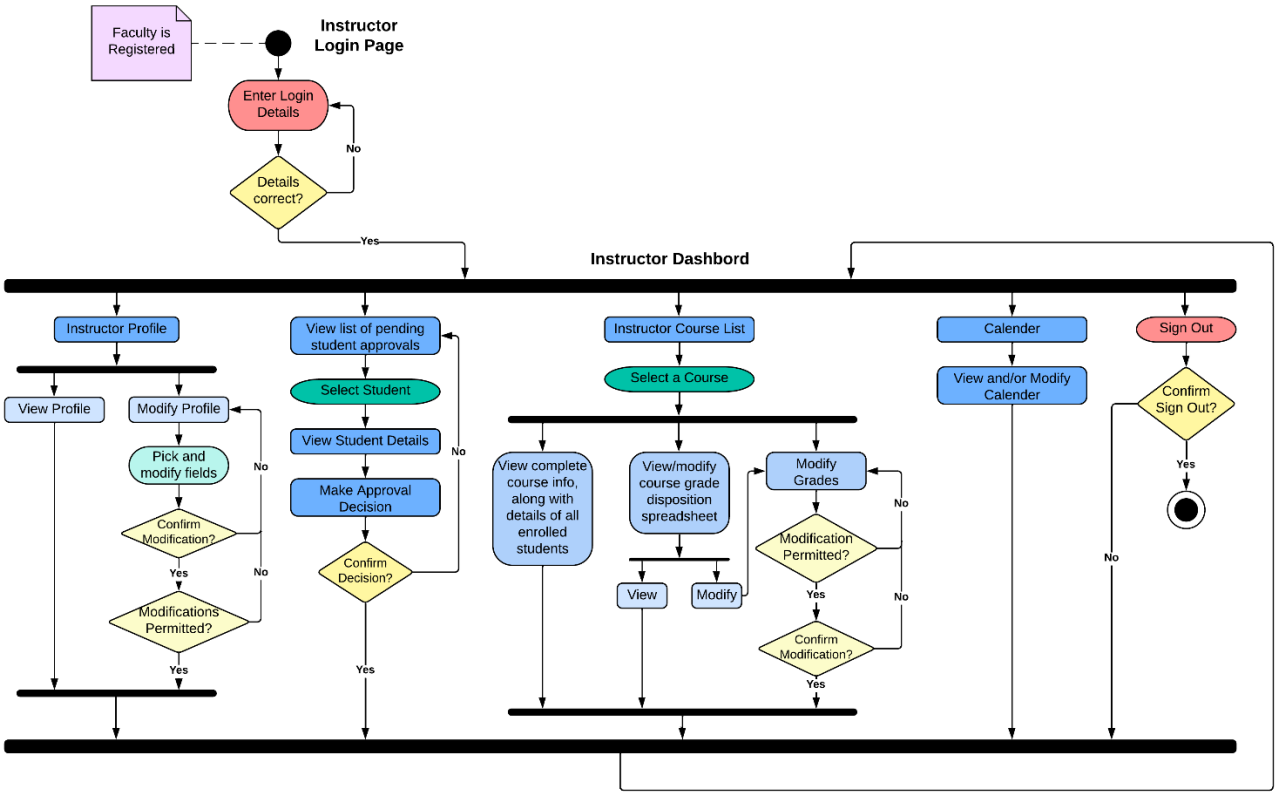
REFER TO THE CODE AND TESTS IN SRC FOLDER THAT COVERS ALL OF THESE DESIGN PRINCIPLES INTEGRATED WITH THE SQL AND NOSQL DATABASES.

The "src" directory in coding deliverable of this Practicum has the developed code with the user management and the course management system implemented thoroughly. Course registration system and database integration is well versed with the implementation of several design patterns.

The tests can be run by going to "/src/python" of Deliverable 1. Run "python main.py" first to load the database followed by the "pytest" command in the terminal and you can see that all the tests pass and the class functionalities are implemented.

Activity Diagram for the Faculty Class

This helped me understand how the Faculty class communicates and sends messages involving other classes. I got a better understanding of which all classes Faculty interacts with based on the message flow. Faculty interacts with CourseSection class mainly. (See the class diagram below for more information on the CourseSection class).

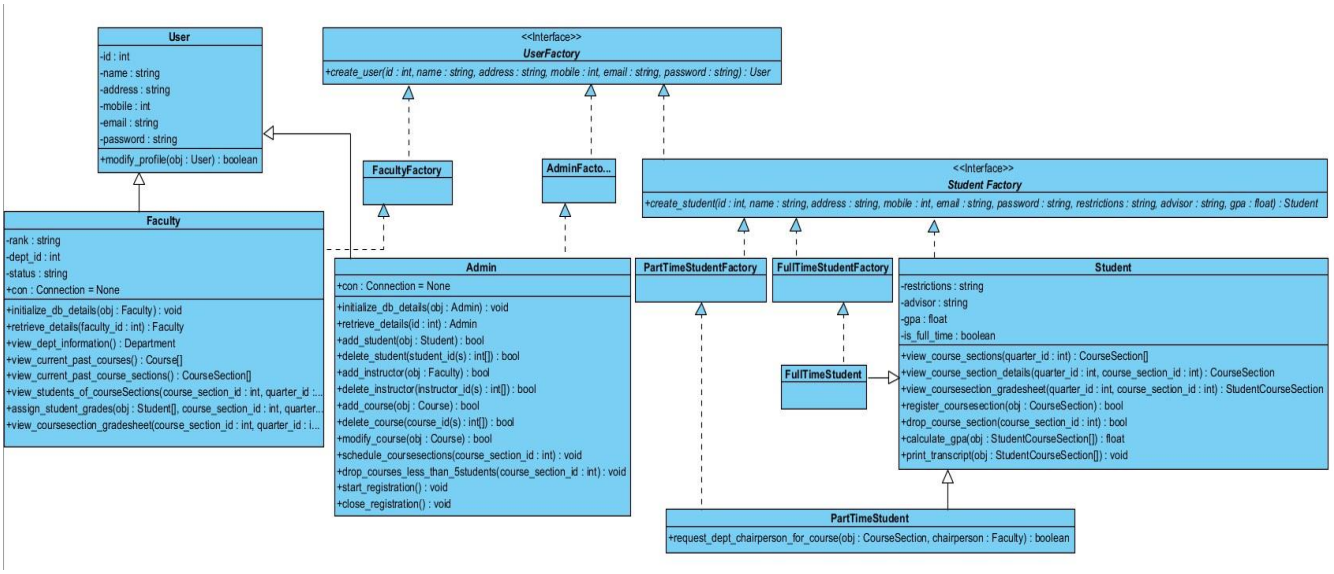


Sequence Diagram for the Admin Class

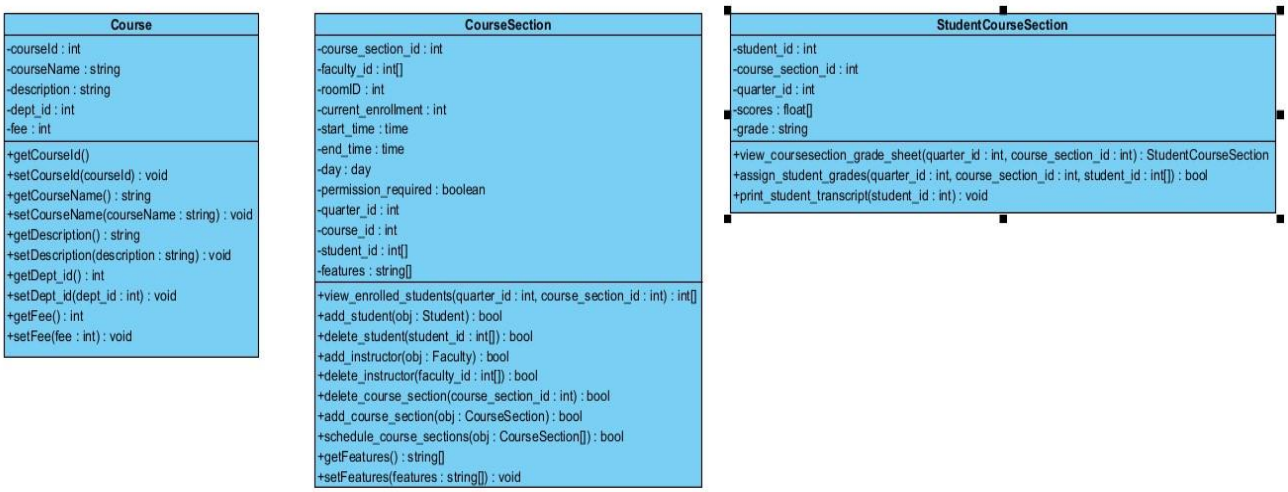
Refer to the sequence diagram pdf in the Models folder for a Sequence diagram involving Admin Interactions with other Classes, Users and Databases.

Class Diagrams

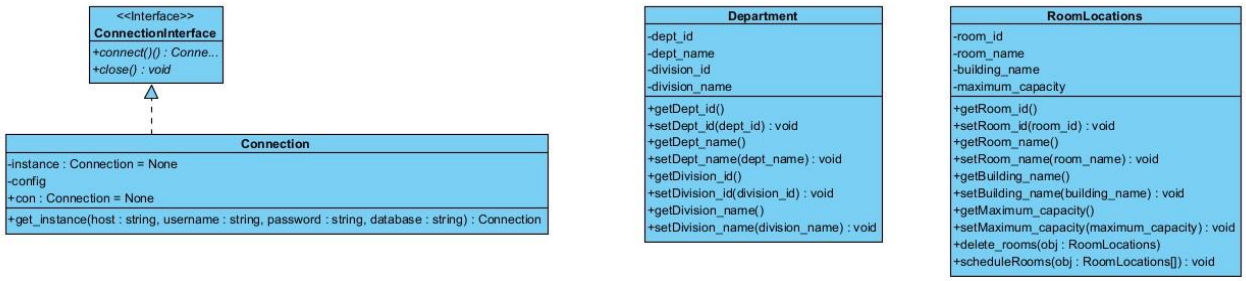
User Management:



Course and Grade Management:



Database Connection Classes, Department and Building Management Classes:



Design Patterns

The following design patterns are implemented in the coding deliverables which can be found in the "src" folder with working tests and .sql files for creating and populating the database.

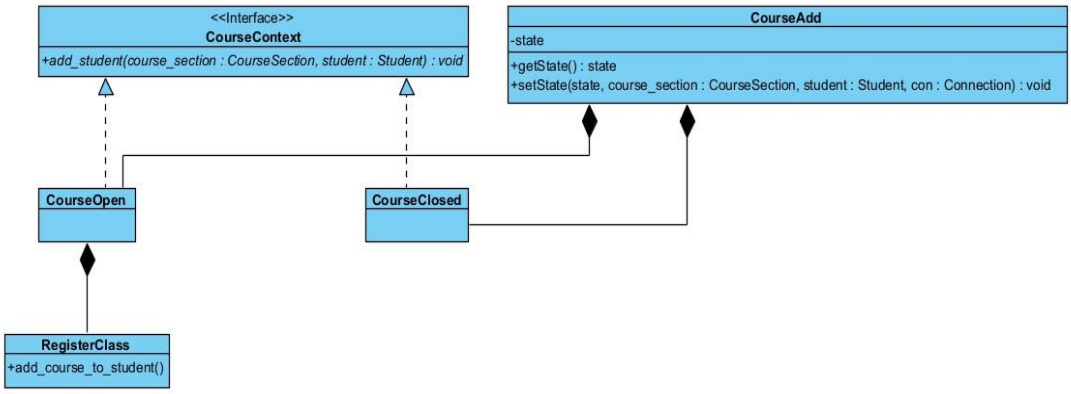
- Abstract Factory Design Pattern
  The REGIE system design uses the Abstract Factory Design Pattern mainly for Object Creation. I understood that this is a creational pattern that is used to create families of related classes and objects and when the code has to be abstracted for creating those objects. I saw that this is similar to what I would require for my REGIE User Management System where Admin, Faculty and Students have to be created but the implementation of this creation should be created. This pattern solves my requirement and I used this pattern to create different types of Users such as Admin, Faculty, Student, Full time and Part time students using Abstract and Concrete Factory classes.
  UserFactory (Abstract Factory) – FacultyFactory (Concrete Factory), AdminFactory (Concrete Factory), Student Factory (Abstract Factory)
  Student Factory – FullTimeStudentFactory (Concrete Factory), PartTimeStudentFactory (Concrete Factory).
  Each of these factories call the respective classes for object creation but this implementation is hidden to the outside world. The code for this implementation can be found in the "abstract_factory.py" in the "src" folder of Deliverable 3 and working tests can be found in "test_abstract_factory.py".

- Template Design Pattern
  Template design pattern allows a class to have the skeleton of the process that contain the common methods and the subclasses implement specific functionalities. This came into use in my User Management system again where User is an abstract class containing just the user profile information and has a modify_profile() method that is general to all the users. This User is being extended by Faculty and Admin to implement the Faculty and Admin specific functions. In addition to this, the User management system is supported by inheritance where Student class is inherited by Full time Student and Part time Student classes for their respective functionalities. Each subclass (Admin, Faculty, etc.) thus supplies its own default for adding courses (either to course database, to course sections or to their rosters or timetables) but inherits the common default functions from Class User for the getter, setter functions and modify_profile. The codes can be found in "user.py", "admin.py", "faculty.py", "student.py", etc.

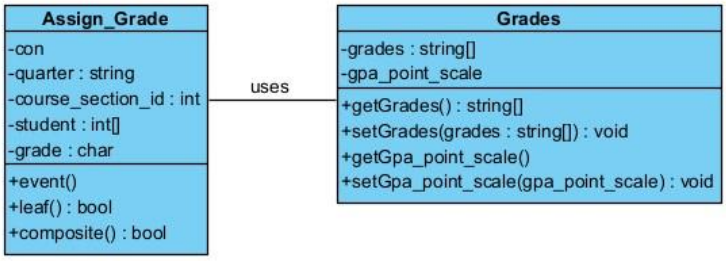
- Singleton Pattern
  The database connection has been implemented with a singleton pattern. This is brought into use because at any point, only one database connection should occur and only one connection object is created and used throughout the application. So, a singleton design pattern is produced with a ConnectionInterface and a Connection. The interface serves the purpose of abstraction. The Connection Class has a private instance variable instance and whenever the class method get_instance() is being called, if an instance doesn't exist, it

creates the new instance and returns the connection object. If it exists, then it just returns the existing connection object saved in the private instance variable. Apart from that, the connect() and close() methods of interface are being implemented in the class which returns MYSQL connection objects to be used throughout the application. This way only one class or module can modify the database at a given execution preventing deadlocks and consistency issues. The code can be found in "database_connect.py" and the tests can be found in "test_database_connect.py".

- State pattern
  Courses can basically have two states – open when the enrollment count is less than the room capacity and closed when the enrollment count exceeds the room capacity. State pattern can be used in this context and the implementation (code and tests) can be viewed in the "course_states.py" and the "test_course_states.py" folder.



- Composite pattern
  "assign_grade.py" has the Assign Grade class that implements the composite design pattern where to assign a student grade, the class takes as parameters the student_id, course_section_id, grade and sets the grade if the student is registered in the course. A faculty can assign grade to a group of students or a single student by calling the same event function that demonstrates the concept of function overloading or polymorphism.



- Observer pattern
  The requirement of REGIE system that lets faculties add the feature of “Instructor Approval Required” is based on the Observer design pattern. When the feature is added, system shouldn’t add students to the course directly and should let the students request Instructor approval for the courses via email to be approved or denied by the instructor. Similarly, when the feature is deleted, students should be added to the courses by the system. Observer design pattern seems to be the best fit for this case, and the implementation can be found in “modify_features.py” and “observer.py” files.