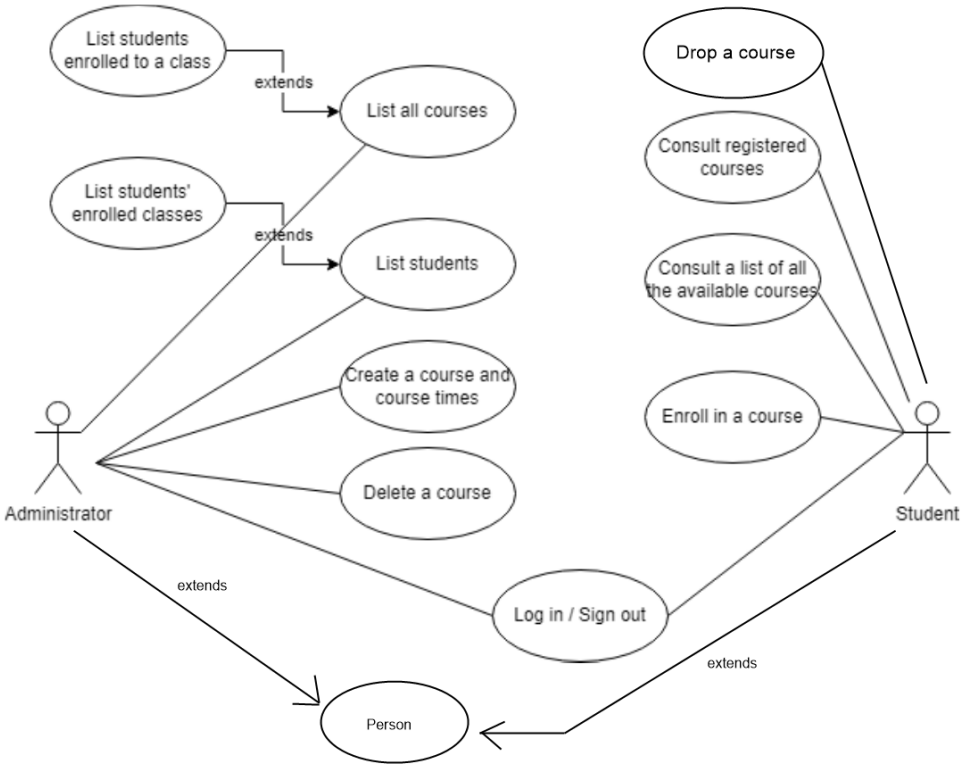
Assignment 3

By:
Joel Lajoie-Corriveau (#40112335)
Hamzah Muhammad (#40156621)
John Paragas (#40064305)
Etienne Pham Do (#40130483)

SOEN 387
Concordia University
Hassan Hajjdiab
December 2 2022

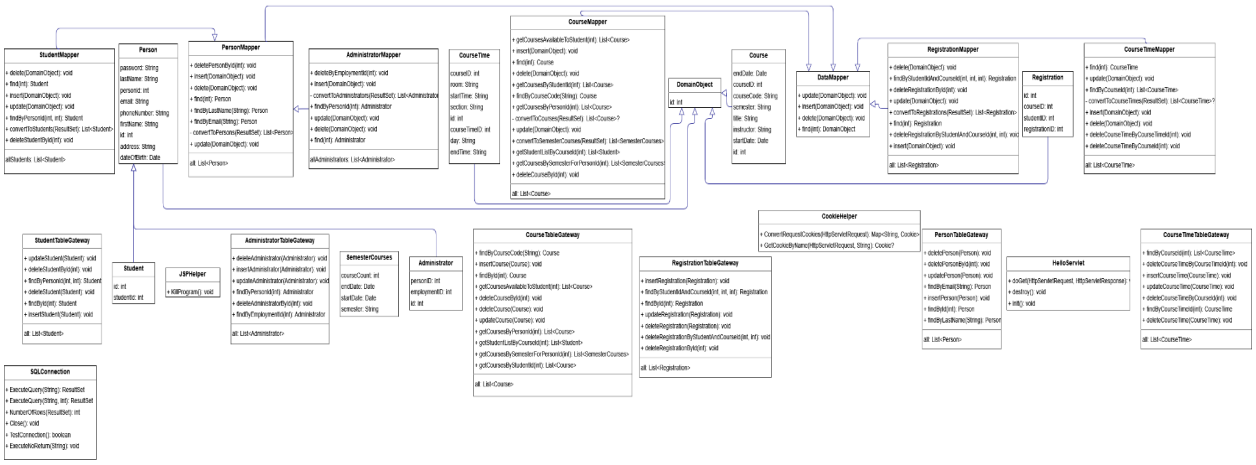
Use-Case Diagram



From our Use-Case Digram, one can see that the only overlap in use cases is the logging in / out feature. Aside from that one functionality, Students and Administrators interact with the website in different ways. For example, administrators can see an overview of students' enrollment activity and can manage the courses available to students. Students, on the other hand, can manage only their own courses, and can enroll into new courses or drop their existing courses. Both have access to all already existing courses.

We can see that in this assignment, the diagram was extended to include the “Person” concept, which is not an Actor per se, but is a component of the system from which both Administrator and Student inherit. Therefore, they “extend” the Person system bubble. Both are seen as a Person when interacting with the system.

Class Diagram of the system

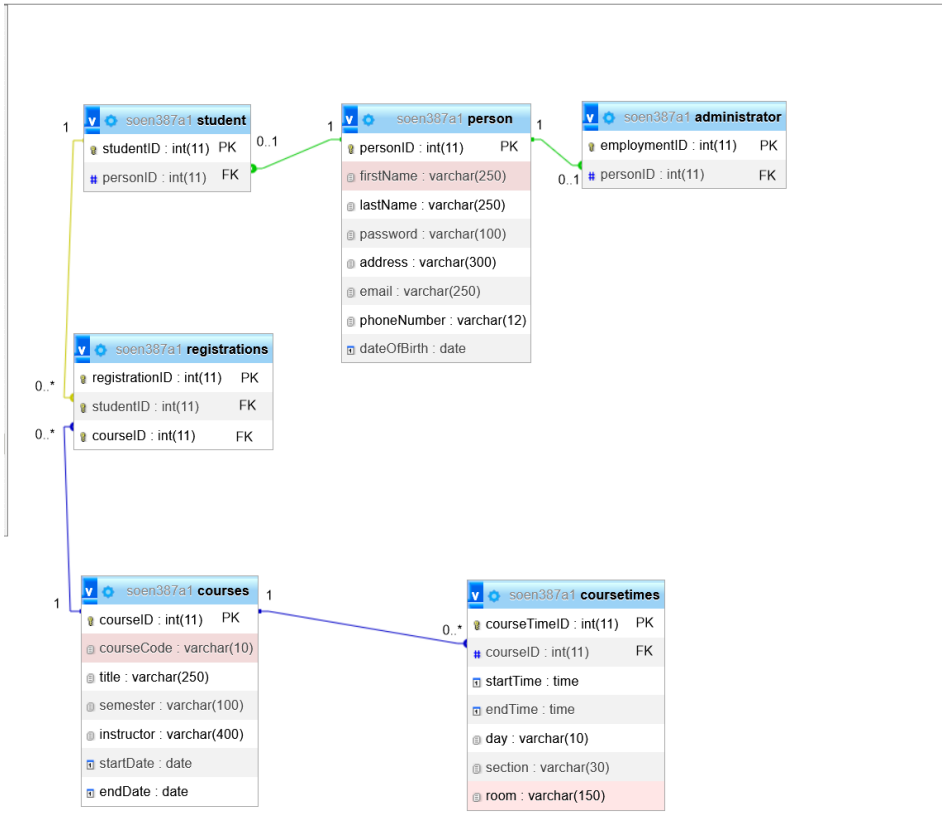


A full sized picture is available in the docs folder of the submission, under the filename “class diagram.png”.

We have changed our model to implement some additional inheritance to simplify the code. All our domain objects inherit from one base class now, and our RowGateways have been renamed to better represent their function as DTOs.
The inheritance mapper is inside the DataMapper for Student and Administrator, which is StudentMapper and AdministratorMapper respectively. Both inherit from DataMapper, an abstract class.
All the required functions are implemented and are visible on the diagram above.

Note that we do not have Composition or Aggregation in our model. No objects contain other objects. The only relationship we have is inheritance, denoted by the white arrows.

Entity Relationship Model



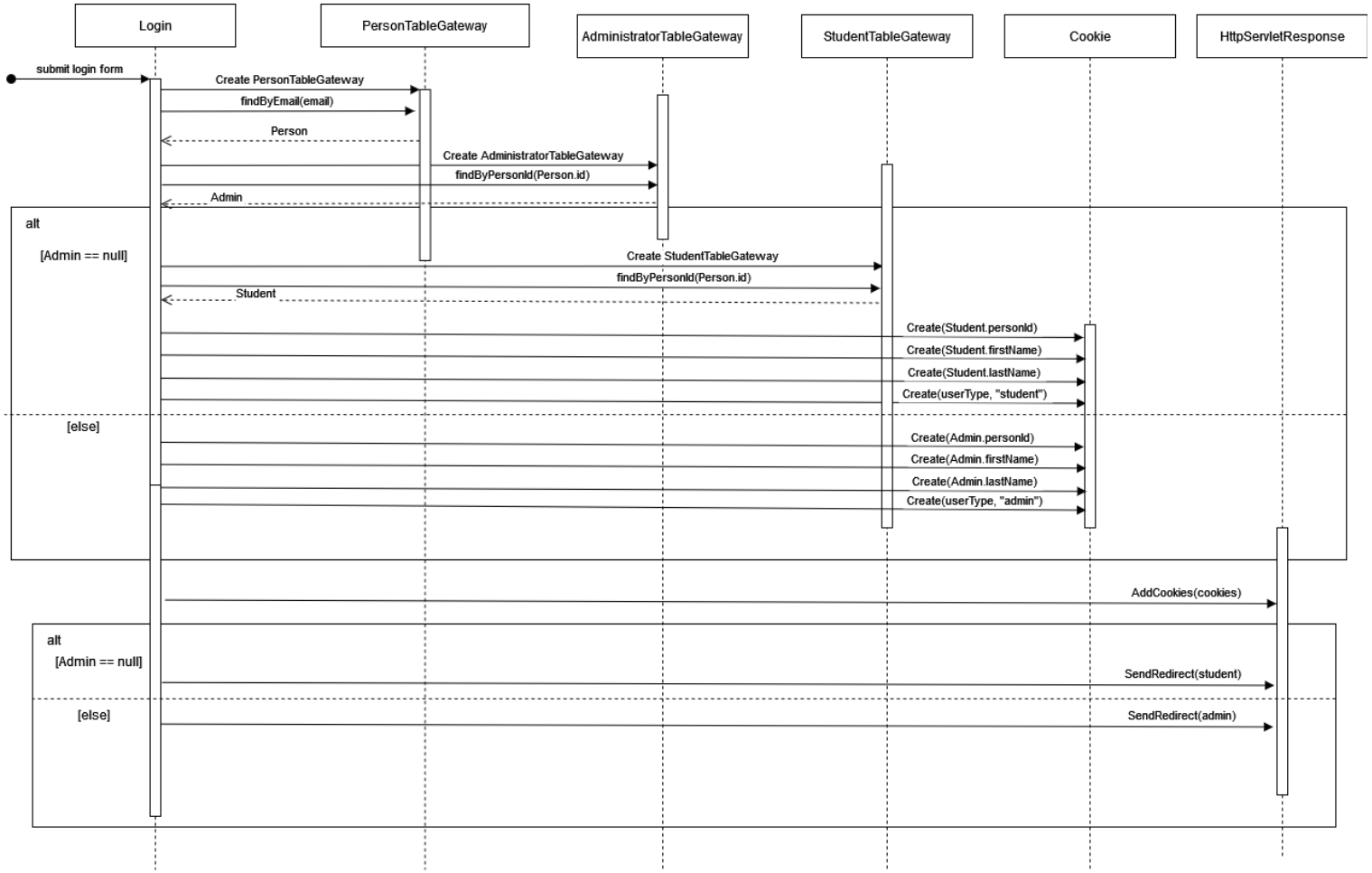
Note that nothing changed in the database between Assignment 2 and Assignment 3. This is because we already had foresighted that a Person could be added to the application that would neither be a student or an admin, and thus made our database expandable and flexible. This had the side-effect of making inheritance implementation in our system very easy, with a one class per table representation.

We kept the design of the database simple, yet expandable. The person table ensures that if any other type of users are added, they can be joined into our database easily.
The CourseTimes and Courses Start and End date are enough to calculate the beginning of a semester and the end of a semester, but if additional clarification on that is required in the future, we can add a specific semester table.

Sequence Diagrams

Please note than in Assignment 3, only the first Sequence Diagram is new. The rest are unchanged from Assignment 2.

Sequence Diagram for the scenario when a user logs in



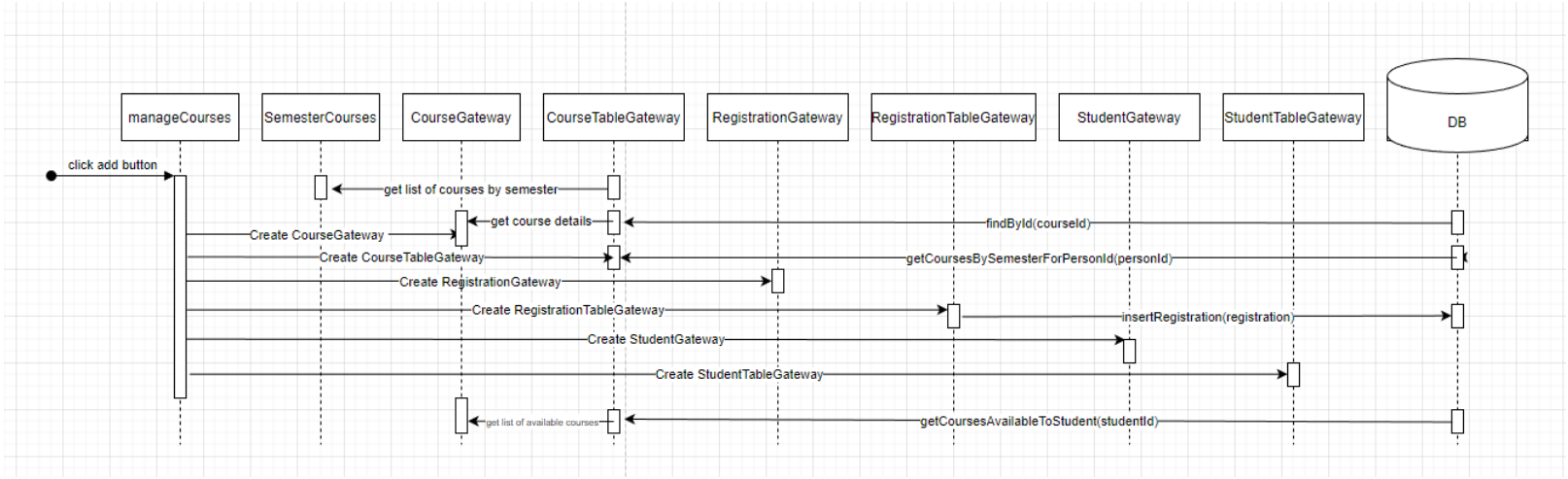
What we can see in this diagram is the whole process that uses inheritance and our gateways to authenticate a user when they try to log in, and redirect them properly. Please note that some of the function calls have been simplified, for readability.

We first query the AdminTableGateway to see if it returns a result. If it does, using our person ID, we know that the person that tried to login is an admin, and we can create cookies with all their information simply by using the Admin object, as it inherits from the person.

However, if Admin is Null, then we know the person is a student, as there are no other user types. So we fill up the cookies with their information, straight from the student class.

This is reflected later in the sequence as well, when we redirect the user to the appropriate page.

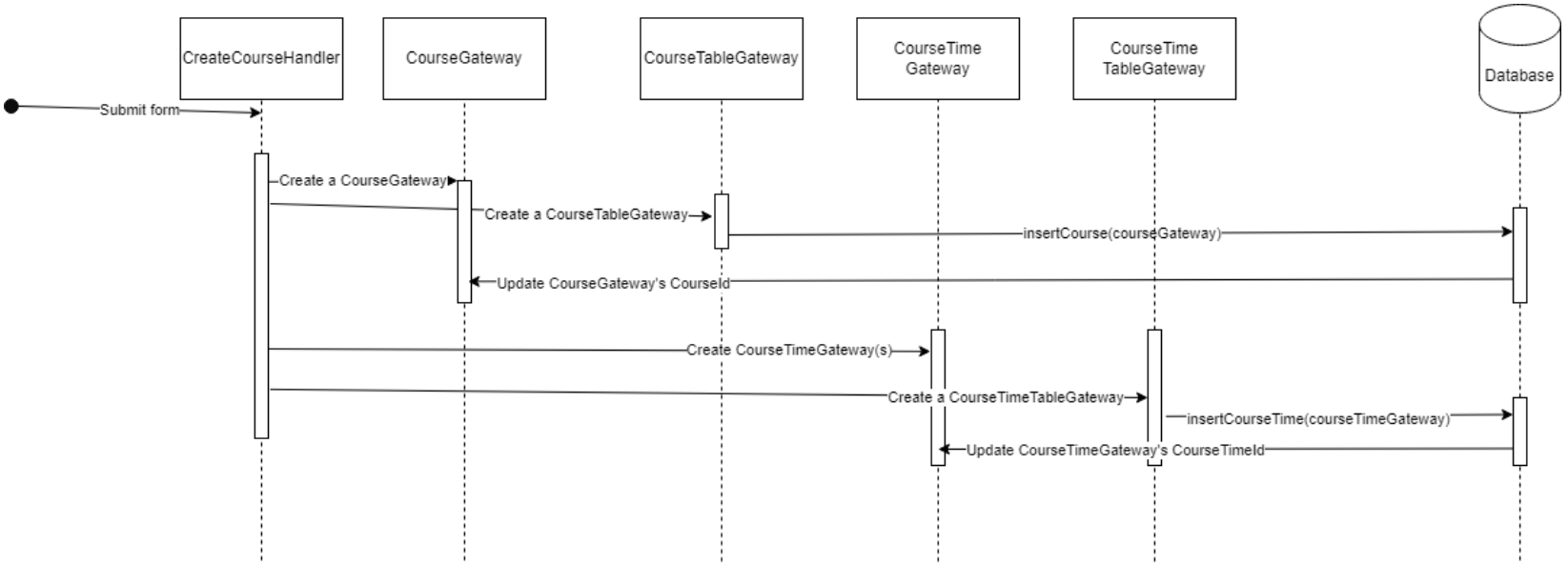
Sequence Diagram for the scenario when a student adds a course



Upon loading the manageCourses.jsp page, it creates 3 table gateways: an instance of the CourseTableGateway class, the StudentTableGateway class and the RegistrationTableGateway class. A list of CourseGateway objects that denotes the available courses is retrieved from the DB via the CourseTableGateway object, using the function getCoursesAvailableToStudent(studentId). A student user can then add a course from that list, whose registration is stored in the DB via the RegistrationTableGateway object. However, a list of SemesterCourses objects is obtained from the DB using the CourseTableGateway object to verify whether a student has not exceeded the max amount of courses for a given semester. Also, a CourseGateway object is obtained from the CourseTableGateway object to verify whether a course was added before the first week of the

semester.

Sequence Diagram for the scenario when an administrator creates a course



The sequence diagram for creating a course is pretty simple. From the Administrator's "Create Course" page, we submit a form with the different parameters necessary for a course. We then pack that data into a CourseGateway. We also create a CourseTableGateway to allow us access to the database. Using this Table Gateway, we then insert the CourseGateway to the database. On insert, we also update the CourseGateway in memory to reflect the auto-incremented value of CourseId.

The same logic applies for CourseTimes. However, on submission of the form, there can be various amounts of CourseTimes that have to be added to the database.