Project Part 7

1. **Project**:
   Rec-Center Management System and 37. Dixit Patel, Swathi Upadhyaya, Yuvraj Arora.

   Our vision here is to build a sports center management system for a university sports center that caters to the health and fitness services of customers.

   Apart from registering for the service (includes LogIn/LogOut), it provides basic facilities to the users like reserving courts, equipment and requesting for a coach. It also allows users to check out and participate in the competitions and events that take place in the center. The website will be accessible to the users, Sports administrator, and the trainer.

2. **Features Implemented**:

Business Requirements	
ID	**Title**
BR-02	Customers can request for an appointment with only a single trainer.
BR-03	Customer can only reserve only a single court for a time slot in a day.
BR-04	A unique registration key should be generated when a customer registers for an event. This is accomplished by generating a UUID number.

User Requirements	
ID	**Title**
US-02	Customers can reserve courts in the Rec-Center Management System.

US-03	Customers can cancel the reserved courts.
US-04	Customers can register for events hosted in Rec-Center.
US-05	Customers can view the list of upcoming events in the Rec-Center Management System.
US-06	Customers can view the event details of a particular event.
US-08	Trainer can accept an appointment.
US-09	Trainer can reject an appointment.
US-11	Administrator can create events in the Rec-Center Management System.
US-12	Administrator can make updates to the event.
US-13	Administrator can view list of upcoming events.
US-14	Customers can view the list of courts available.
US-01	Customers can request appointment with trainer.

Functional Requirements	
ID	**Title**
FR-01	Once a customer has requested for an appointment with the trainer for a time slot, that slot is blocked.
FR-02	Once a customer has reserved court for a time slot, that slot is blocked.
FR-03	Customer needs to have valid login for requesting an appointment.
FR-04	Customer needs to have valid login for reserving a court.

FR-05	Customer needs to have valid login for registering to an event.

Non-Functional Requirements	
ID	**Title**
NFR-01	The UML Diagrams conform to Enterprise Architecture Standards. UML 2.0 version is used. (Maintainability)
NFR-02	The user data is written into the relational database during the transaction, so recovery from failure is easy. (Recovery)
NFR-05	The information access levels are different for Customers, Staff and the Administrator. (Security)
NFR-07	User/Event Response Time is Minimum. (Performance)
NFR-08	Relational Database shall be used in the backend. Consistency and Integrity of data is guaranteed. (Integrity)

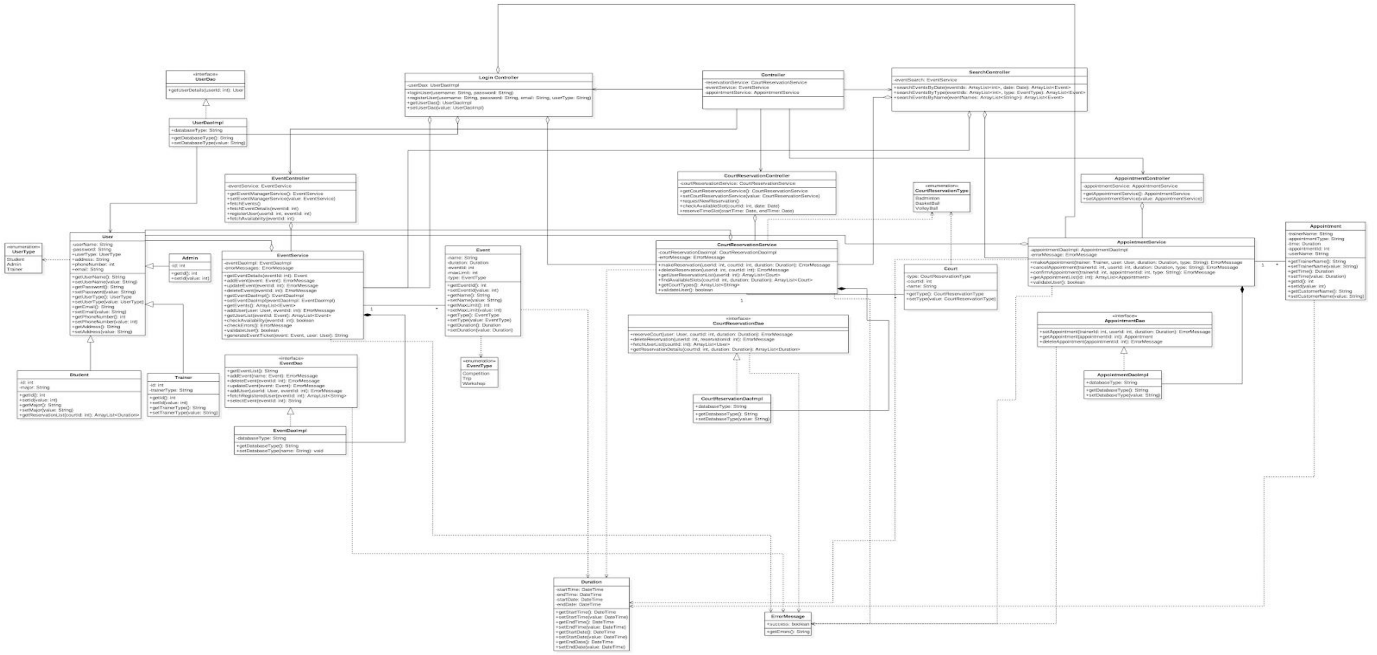
3. **Features Not Implemented**:

ID	**Title**
US-10	Trainers should be able to view appointments requested by the customers.
US-14	Customers should be able to view the appointments requested by them.
US-07	Customers should be able to view the list of

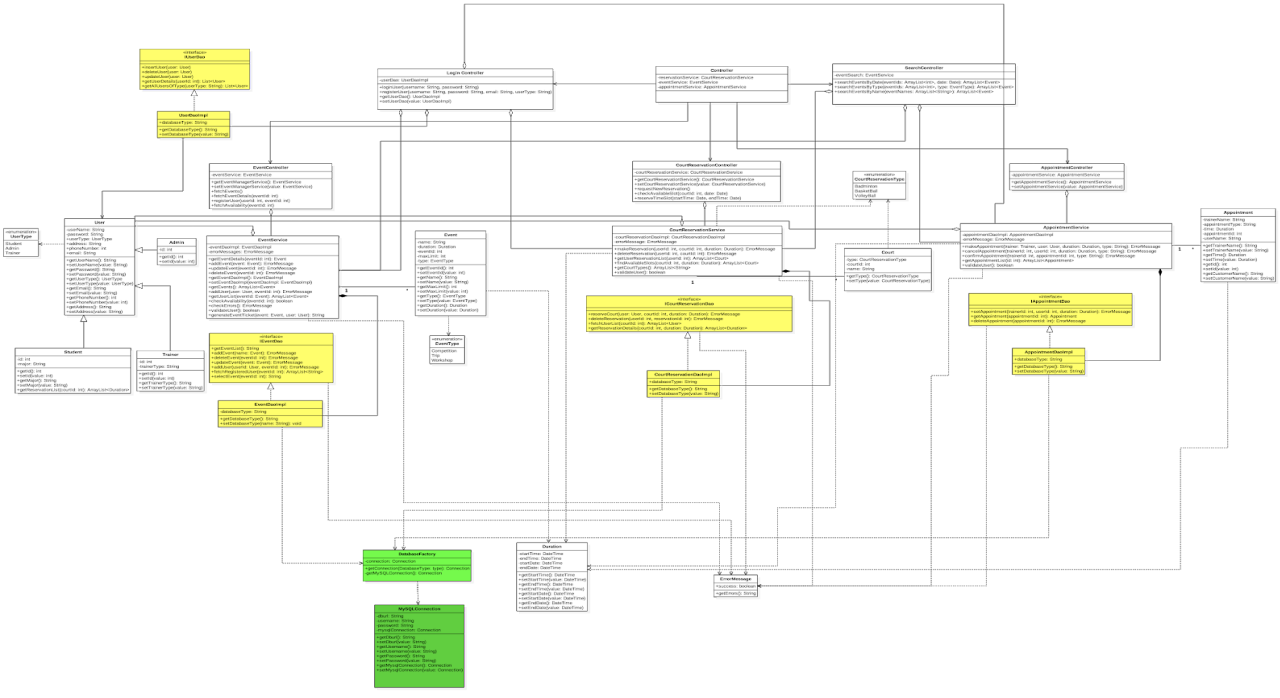
	appointments that they have registered for.

4. Class Diagram:

- Part 2 Class Diagram (https://bit.ly/2KuWQ3U)



- Final Class Diagram (https://bit.ly/2HJpksL)

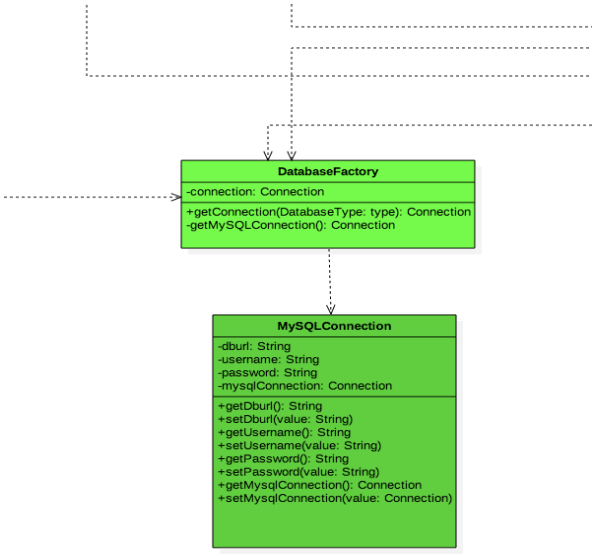


What changed? Why? If it did not change much, then discuss how doing the design up front helped in the development.

We chose Data Object Design (DAO) pattern for our project. We built our class diagram structured on this design pattern. As a result, most of our model classes - Event, Reservation, Court, Appointment, the interfaces and the DAO Implementation classes remained unchanged. The classes that we had to modify a little were the controller classes and the service classes where the logic to display persistent data fetched from the database and the logic to convert the input parameters to a form that is stored in database resides. Choosing this design made the development of project easy and simple.
The associations, the aggregation, inheritance, and composition structure of class diagram remained intact. The only modifications were for error checking, invalid data, function parameters and return types. A few functions were added and redundant functions were deleted. The enums used were refactored. And to implement the database connection factory design pattern, few modifications were made as suggested in the feedback at various stages of the project.

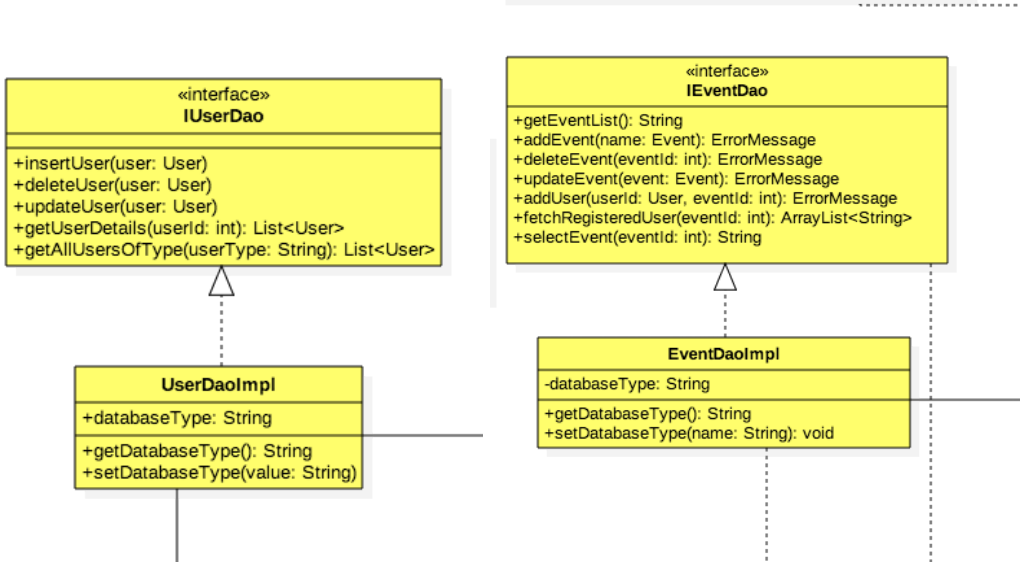
5. **Design Patterns:**
    - **Factory Design Pattern**

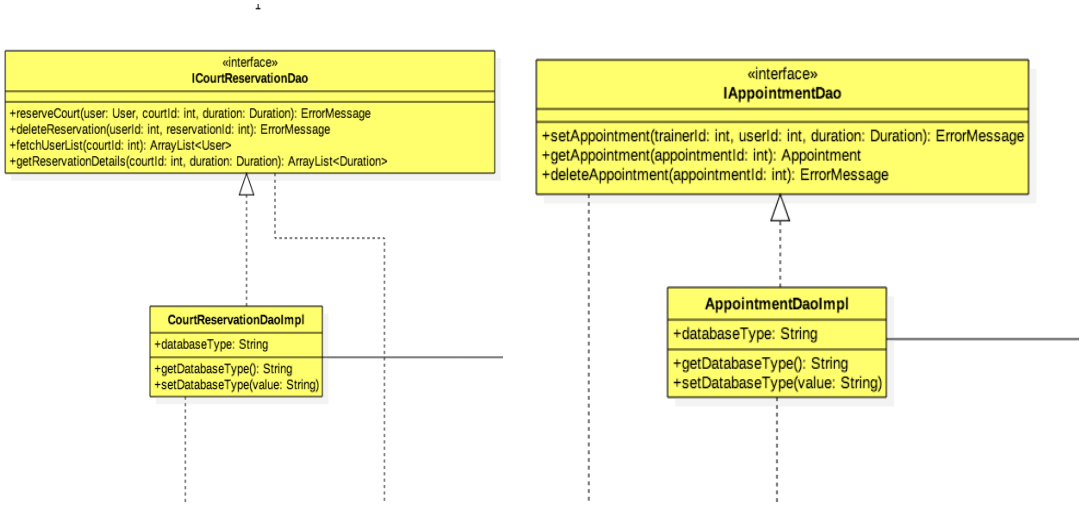
      We implemented the Factory design pattern to create a connection object without exposing the creation logic and to refer that object using a common interface. Below we can see two classes (Database Factory & MySQLConnection) which implements the Factory design pattern. There is one connection class as we have used only one kind of repository to store all the data, which is MySql relational database.



- **Data Access Object Pattern**

  We used this pattern to separate low level data accessing the operations from high level business services. We have an interface that defines the standard operations to be performed on the object, concrete class which implements the interface and gets the data from the database and an object which stores the retrieved data. For all the use cases we have Dao and Dao Implementation classes which have been configured.





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?

This entire process of creating the system from the scratch has given us a complete picture of the Software Development Life Cycle. The first step of deciding on a project, followed by understanding the various types of requirements - user, business, functional and non-functional, helped us formulate the abstract idea of the system into a system with set of features. Further analyzing the interaction between the system and the user using UML diagrams - swimlanes, sequence diagrams helped us understand the behavior of the system and the user. The next step of designing the class diagrams and defining the classes gave us more clarity on how each behavior of the system and user can be incorporated. The design patterns used made our implementation more robust, flexible and reusable for extension. The continuous evaluations and feedback helped us in refining the system further. It was a practical hands-on approach towards efficient designing and implementation of a software system in a parallel development environment.