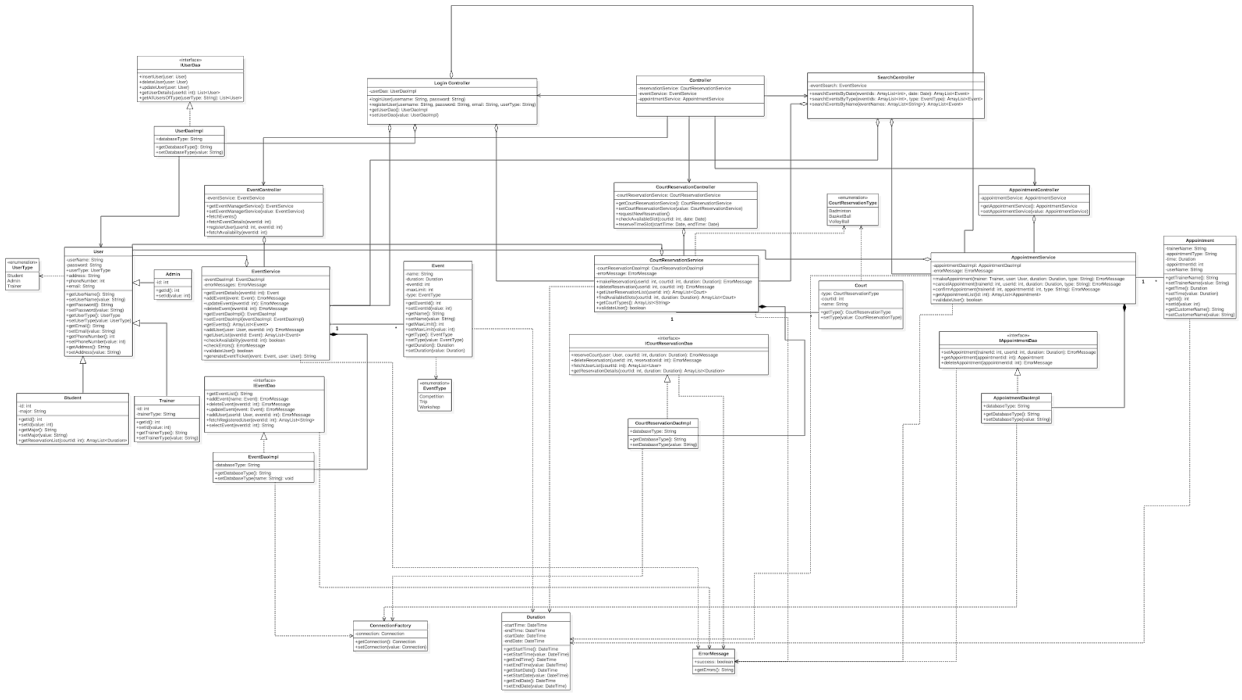
# Project Part 4

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.

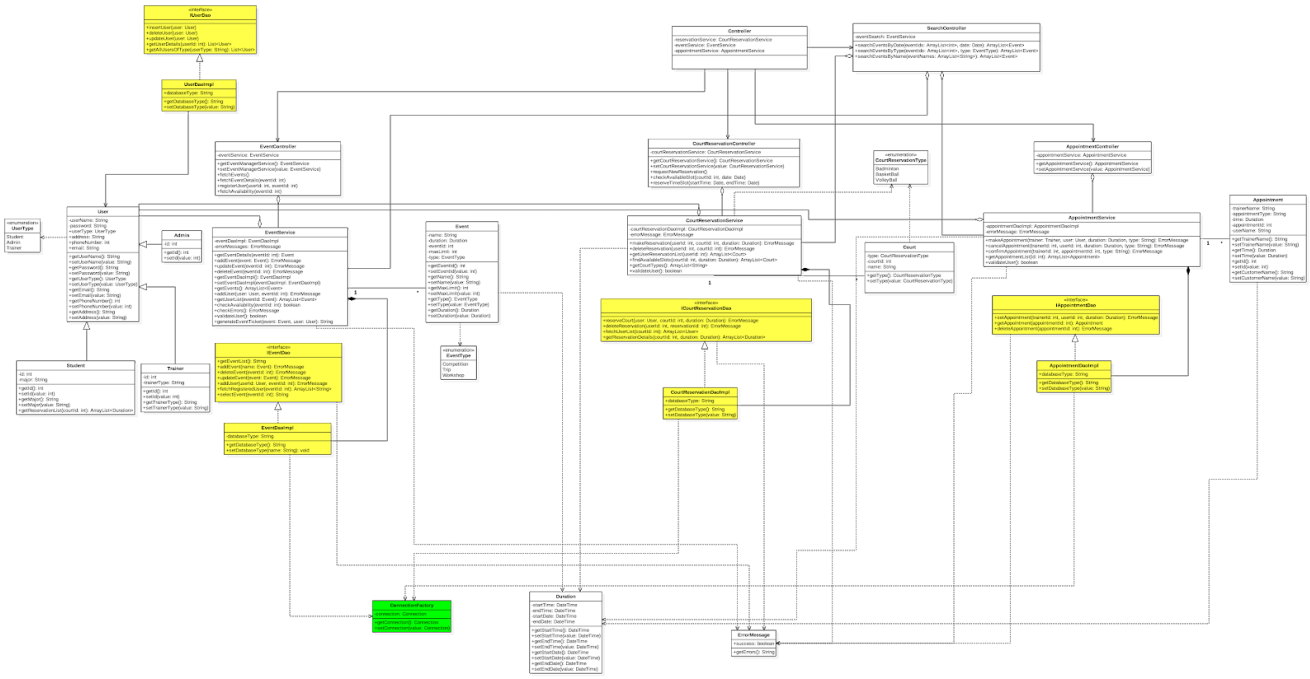
   Apart from registering for the service (includes LogIn/LogOut), it provides basic facilities to the users like reserving courts, equipment and requesting for a coach. It also allows users to check out and participate in the competitions and events that take place in the center. The website will be accessible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

2. **Previous class diagram**:



(Included high-res image in github repo - *Class_Diag.png*)

## 3. Completed Class diagram:



(Included high-res image in github repo - *Completed_Class_Diag-Part4*)

## 4. Summary:

We modified the class diagram according to the feedback provided, started with creating our database in MySQL, a connection factory to get the database connection and implemented the DAO interfaces for the user, event and the court repository. Implementation for one of the use case - reserving a court is in successfully done. We implemented two design patterns in our project i.e. Data Access Object Pattern for interacting with the database and the Factory Design Pattern.

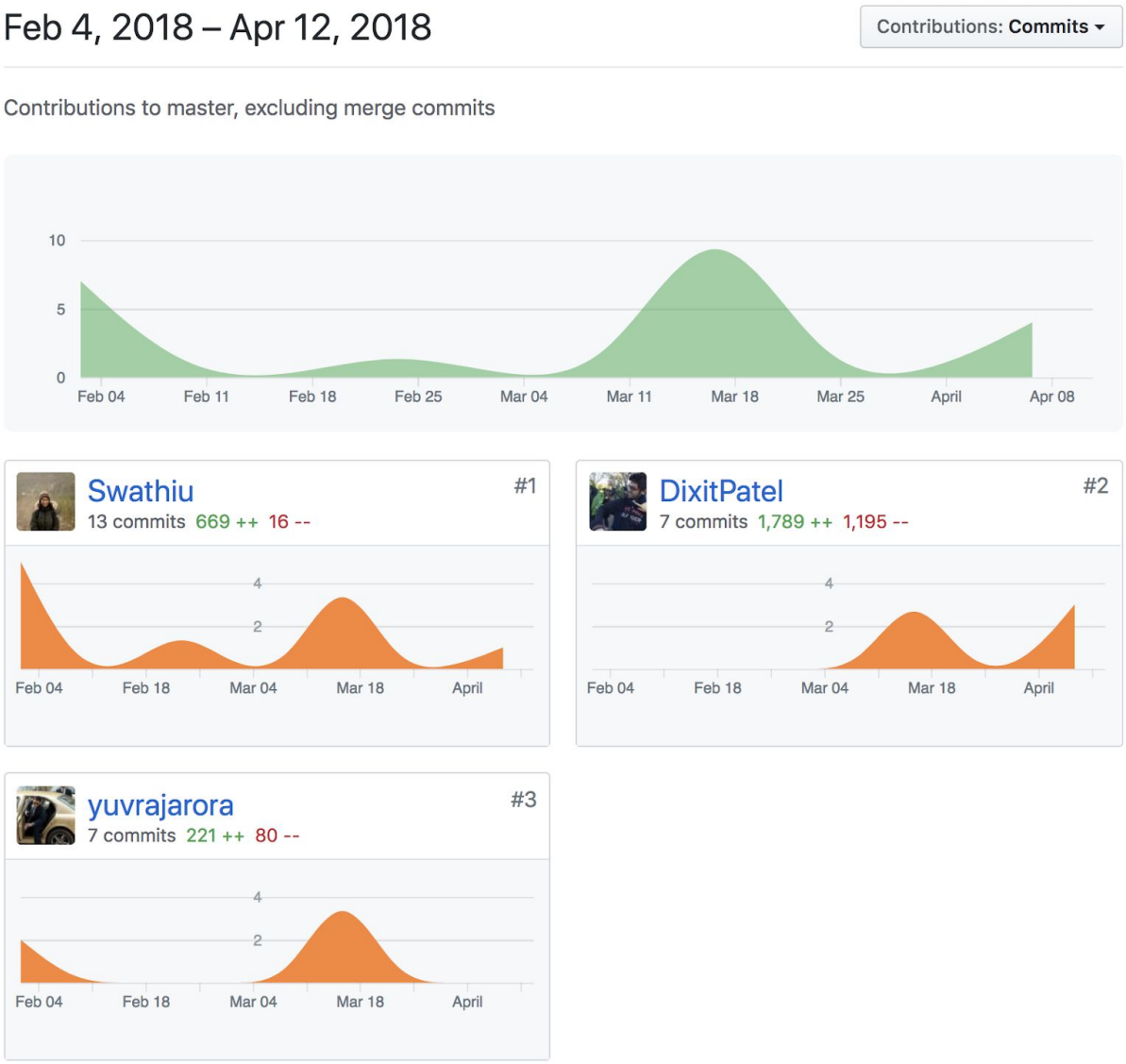
## 5. Breakdown:

Dixit Patel - Restructured the Project file system, refactored enum classes and added controller logic.

Swathi Upadhyaya – Implemented the classes associated with trainer appointment functionality (Service, Controller, Dao Implementation and Appointment class) partly.

Yuvraj Arora – Implemented Model classes - Trainer and Student. Added logic for court controller and court service classes. Implemented Utility classes.

## 6. GitHub Graph:

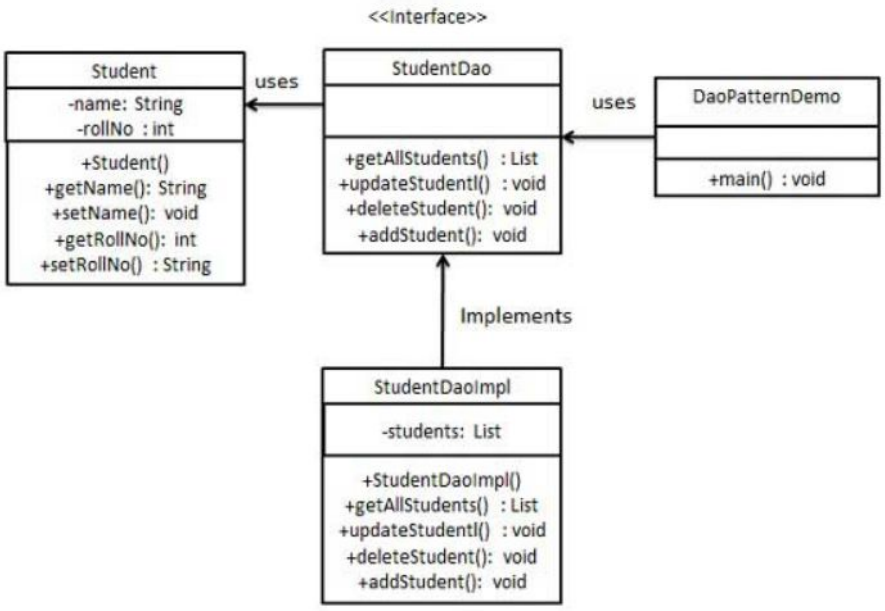



## 7. Estimate Remaining Effort:

Our estimate for successful completion of the project is around 2 weeks.

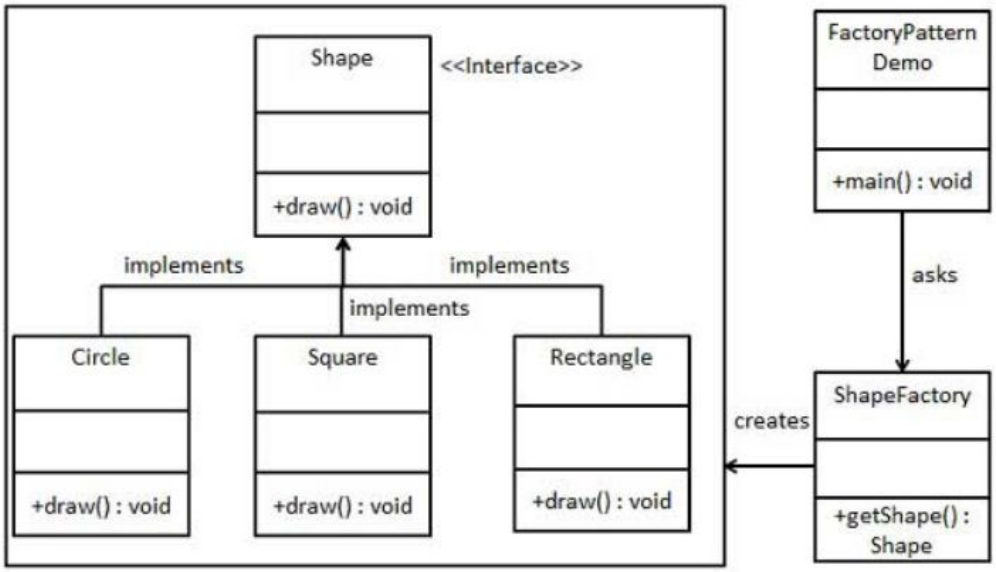
## 8. Design Pattern:

### a. Data Access Object Design Pattern



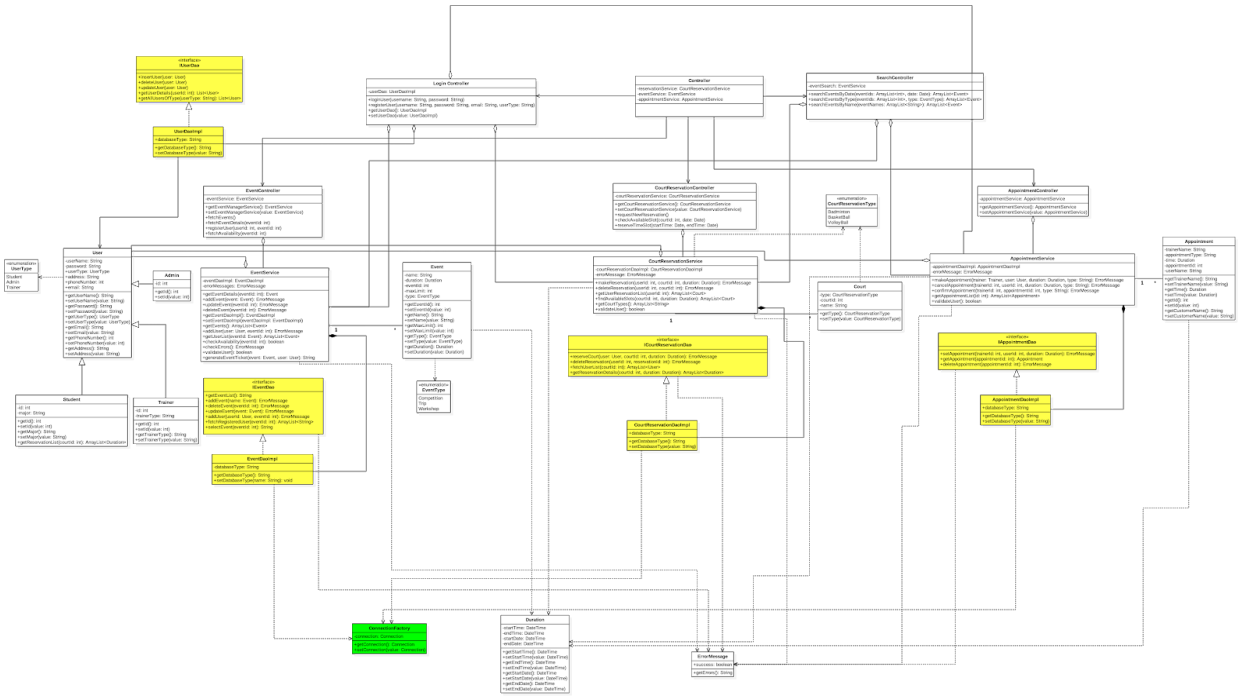

We used this pattern to separate low level data accessing the operations from high level business services. We have an interface that defines the standard operations to be performed on the object, concrete class which implements the interface and gets the data from the database and an object which stores the retrieved data.

### b. 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.

### c. Class Diagram with Design Pattern Implementation



(Included high-res image in github repo - *Class_Diag_Part4.png*)
The above class diagram shows our implementation of the Design Patterns. The **yellow colored** classes represent the Data Access Object Design Pattern and the **green colored** classes represent the implementation of the Factory Design Pattern.

## 9. Next Iteration:

Our next iterations would be as follows:

- Complete the implementation of the Rest APIs in Controller classes.
- Integrate all the modules and develop test cases and test the whole system.