GetSetGo

Tournament Management System

Project Part 4: Final Report

Team Members:

Kavya Ravikumar

Madhumitha Soundararajan

Upendra Sabnis

[Github Project Link]

1. What features were implemented?

SNo	Req ID	Requirement	User	Priority	Completed
1	BS-02	Only Admin can add a game	Admin	Medium	Yes
2	BS-03	Only Admin can assign a role to a user	Admin	Medium	Yes
3	US-01	Browse all the tournaments by tournament name, game and location.	Host, Participants, Admin	Medium	Yes
4	US-02	I can register for the tournament	Participants	High	Yes
5	US-04	I should be able to add a tournament	Host	High	Yes
6	US-05	I should be able to edit a tournament	Host	High	Yes
7	US-06	I should be able to delete a tournament	Host	Medium	Yes
8	US-07	I should be able to add a game	Admin	Medium	Yes
9	US-08	I should be able to edit a game	Admin	Medium	Yes
10	US-09	I should be able to maintain users/roles	Admin	High	Yes
11	US-10	I should be able to add a match	Host	High	Yes
12	US-11	I should be able to edit a match	Host	Medium	Yes
13	US-12	I should be able to delete a match	Host	Medium	Yes
14	US-15	I should be able to create a team.	Participant	High	Yes
15	US-25	After a participant logs in, they should be able to view the registered and upcoming tournaments	Participant	Medium	Yes

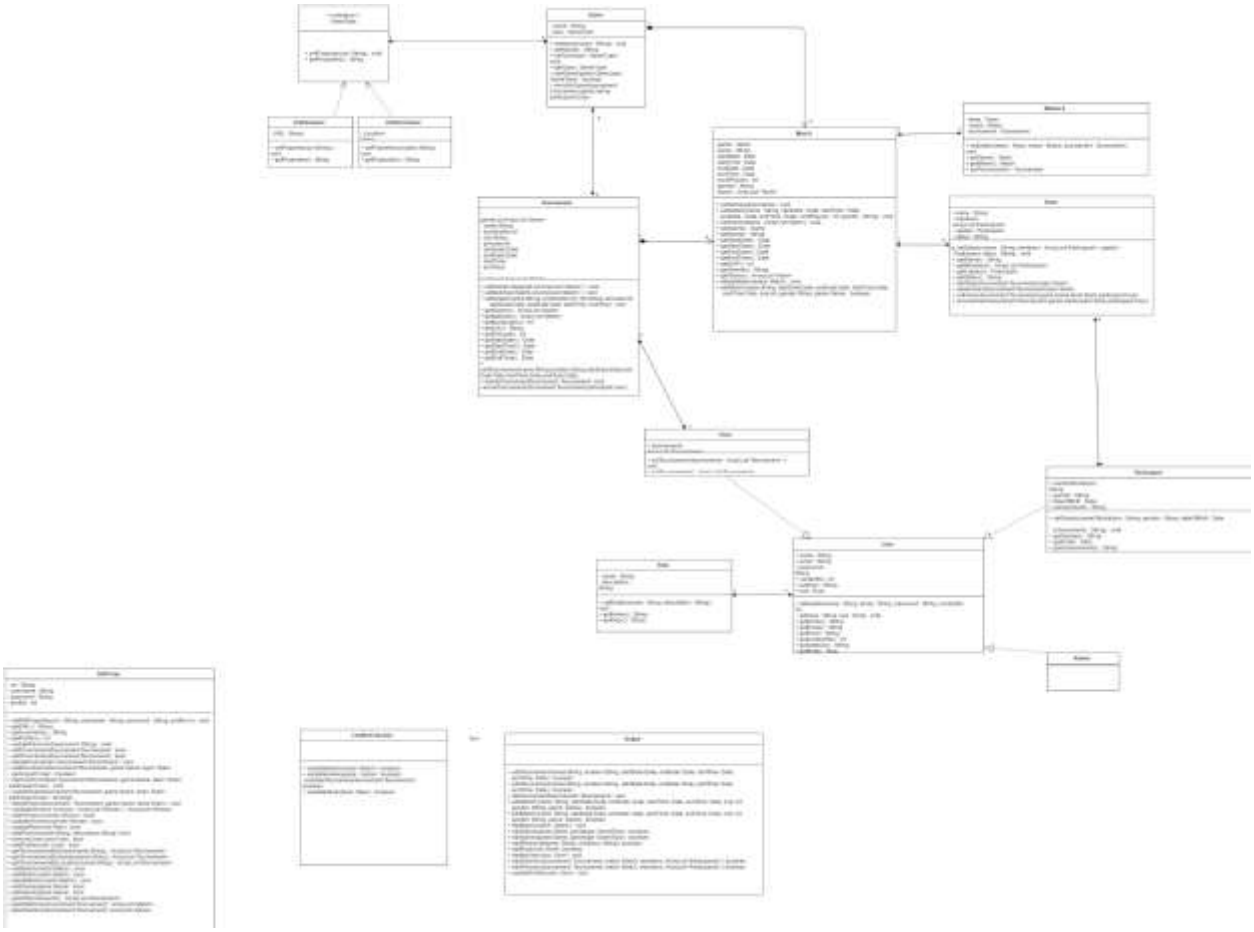
16	US-26	After a admin logs in, they should be able to view the users and the participants on their home page	Admin	Medium	Yes
17	US-27	After a host logs they should be able to view their hosted and upcoming tournaments.	Host	Medium	Yes
18	FR-01	Validation: Validate users (Users with the same email addresses cannot register twice) - for both tournament and an individual game	Host/Admin/Participant	High	Yes
19	FR-02	Database Security: The password is encrypted while storing in the database	Host/Admin/Participant	High	Yes
20	NFR-01	Security: Users with the right username and password only will be allowed to log in.	Host/Admin/Participant	High	Yes
21	NFR-02	Performance: The system should be quick in accessing the database and retrieving details and updating new details.	Host/Admin/Participant	Medium	Yes
22	NFR-03	Database Consistency: Database should be consistent.	System	High	Yes

2. Which features were not implemented from Part 2?

Req ID	Requirement	User	Priority	Completed
BS-01	Users sign up with their gmail credentials	Host, Participants, Admin	Medium	No
US-03	I want to be able to add the winners of a tournament	Host	Medium	No
US-13	I should be able to request the admin to add a new game.	Host	Low	No
US-14	I should be able to edit my profile	Participant, Host, Admin	Low	No
US-16	I want to be able to edit the winners of a tournament	Host	Medium	No
US-17,18,19	I should be able to add,edit,delete a role	Admin	Medium	No
US-20	I should be able to delete a user	Admin	Medium	No
US-21	I should be able to assign a role to user	Admin	Low	No
US-22,23,24	I should be able to edit,delete and withdraw a team.	Participant	Low	No

3. 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.

Class diagram from PART 2: Better version is available online here



- DBProxy class was split into many classes respective to design patterns which have been implemented.
- GameType interface was replaced with MatchStrategy interface based on Part 2 feedback. Online and Outdoor classes implement from GameType interface. MatchStrategy object will be created in Match class.
- Functions which were to be implemented in Helper class was replaced with TournamentController,MatchController classes based on the functionality.
- Requirements with teams and roles were not completed. Hence classes Team and Winners were not implemented.

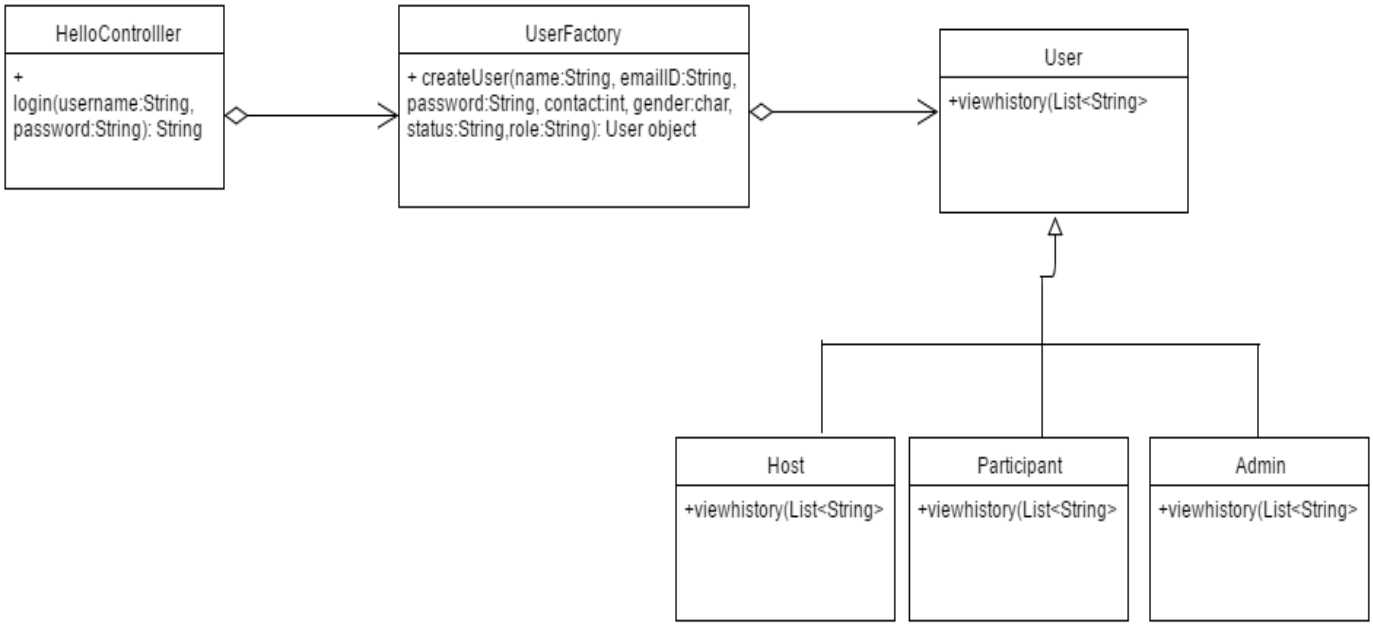
4. Did you make use of any design patterns in the implementation of your final prototype? If so, how? If not, where could you make use of design patterns in your system?

Yes. We used four design patterns in our implementation. Here is the following:

1. **Factory Method:**

   In our system, we have three types of users: Hosts,Participants and Admins. To keep a track of all the users and their roles in the system we have implemented Factory method design pattern. In Factory method design pattern, client class only needs to call the generic function in the factory class. If new roles have to be added in future, we need not change the client class.

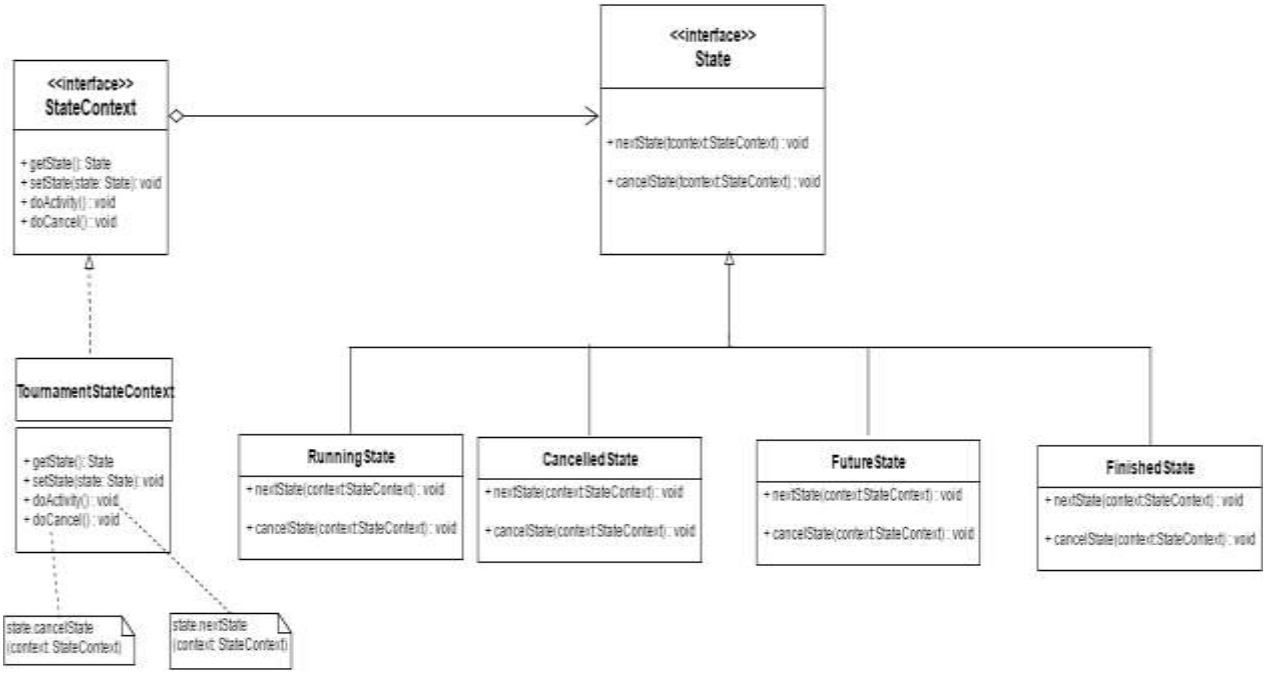
- **Class Diagram for Factory method design pattern**



2. State:

When the internal state is changed, State design pattern allows an object to alter its behavior. In our project, each tournament and match in the system can have four different states: Running, Finished, Future, Cancelled. To maintain these different states, we have used State design pattern. The main intention is to separate the attributes of the tournament object from higher level state which alters the object behavior.

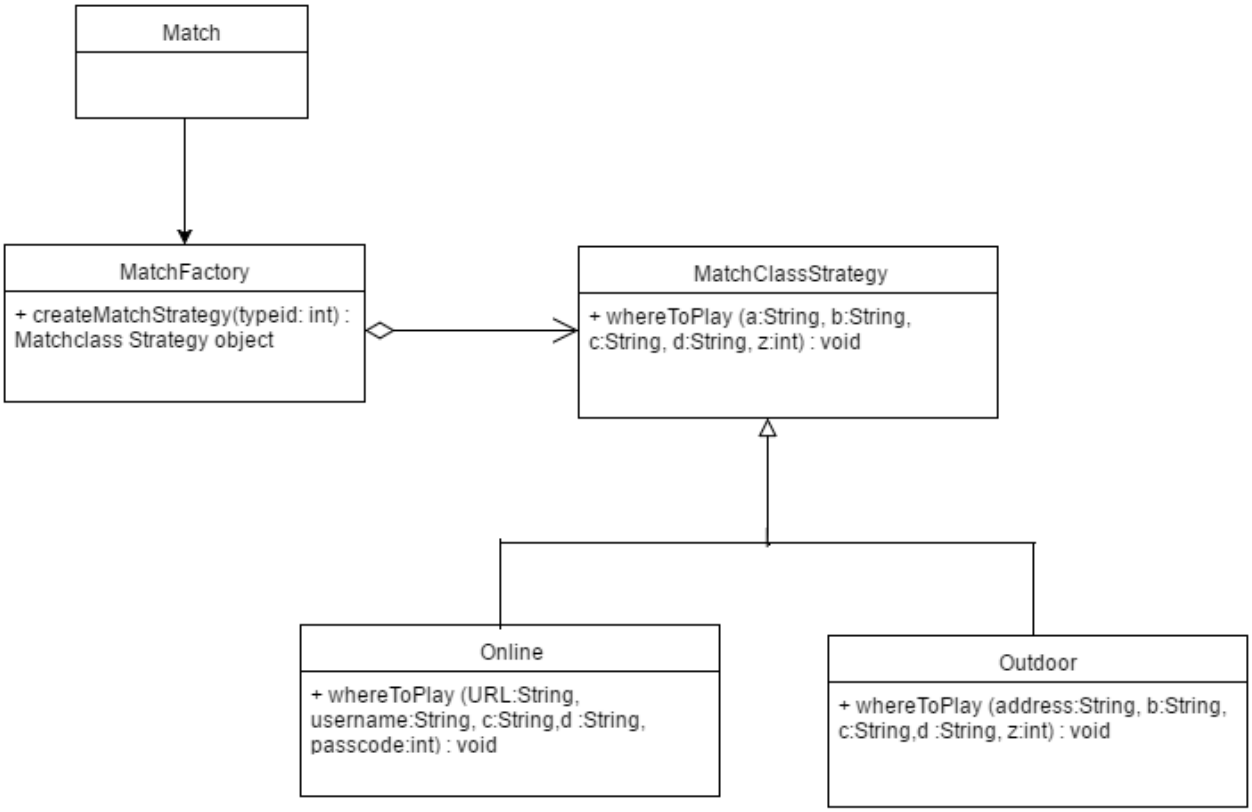
- **Class Diagram for State Design Pattern**



3. Strategy:

We have two types of matches: Online match and outdoor match. To create the type of match object, we made use of Strategy design pattern. Outdoor matches will have address, city, state, country and pincode as parameters where as Online matches will have authorized URL, username and password as parameters to login and play the match. Hence, we captured the abstraction in MatchStrategy interface and buried the implementation in Outdoor and Online classes. Now this implementation is open to be extended for new type of match like indoor.

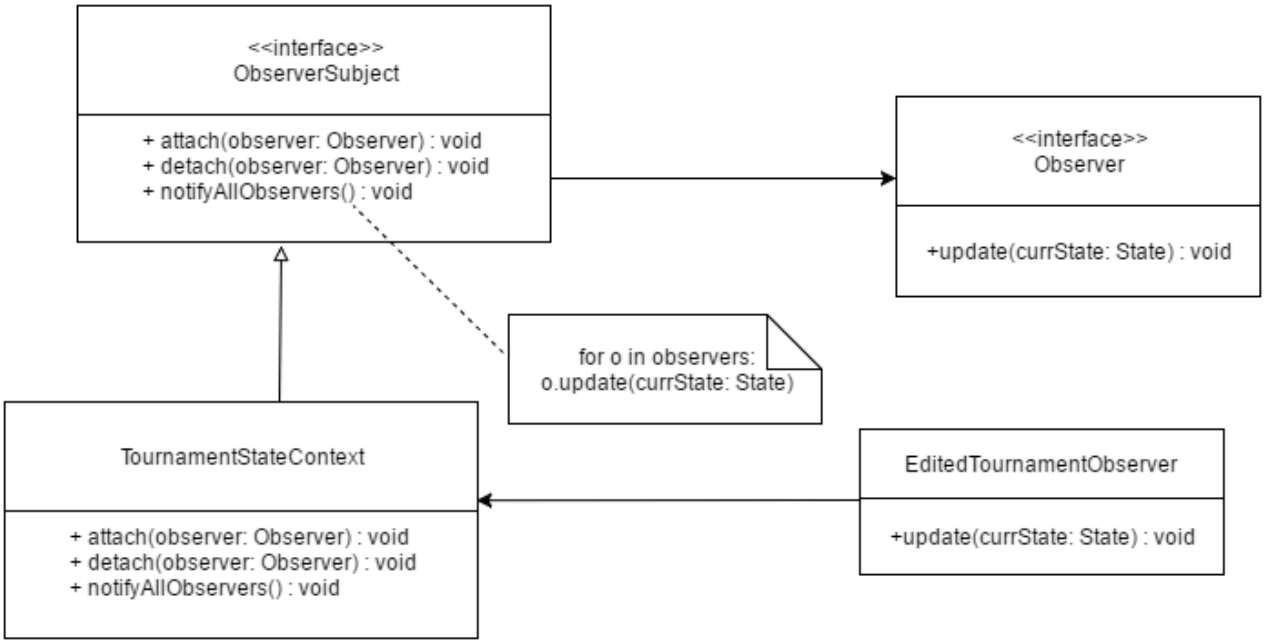
- **Class Diagram for Strategy Design pattern**



4. **Observer:**

The main purpose of the Observer pattern is to allow one object to inform others about a change of state. In our implementation, when the host adds a tournament, the newly added tournament should show up for the participants. Also, when host deletes a particular tournament, the user is notified about it. Hence, to implement these requirements, we made use of Observer design pattern.

- **Class Diagram for Observer Design Pattern**



5. 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?

- Communication between various modules was identified in design phase.
- Since analysis was done before implementation, it was easy to implement since we did not spend more time in analyzing.
- Functionality of the system was identified.
- During analysis phase, we tried figuring out all the users of the system.
- Requirements were properly gathered from each user's perspective.

- In design phase, we identified the user interface through UI mockup
- Interaction between various components of MVC and database specifications were clarified.
- UML diagrams give a visual representation to view the system. Since our system is quite large and complex with 3 users and their roles, UML was very convenient to figure out all the relationships between them.
- Figuring out problems was easy with the help of UML diagrams.
- As developers, we were able to clearly understand the system before implementation.
- The use case flow for each requirement also provided a clear knowledge of the flow of the system.
- Modeling the system using UML diagrams provided better communication for the team discussions and meetings.
- Since we have few more requirements to be implemented, UML diagrams will act as a clear documentation in future to enhance or fix bugs in the system.