Project Part 5: Final Report

Project: 30_FindSportMates
Team: Yi Chou, Yu-Chih Cho, Chih-Wei Lin

1. Features that were implemented

According to our user cases described in project part2, the following table shows which user case we have implemented, and noted that part of description was not implemented, which is labeled with orange colour.

User Case ID:	Use Case Name:	Description:
UC-01	Log in	People of CU, Officers can login to access the system with valid account.
UC-02	Register	People of CU can register for an account using their CU email.
UC-03	Create Events	People of CU can create sport events by filling out the form (sport kind, time, range of number of people, location, message)
UC-04	Cancel Events	People of CU and Officers can cancel events. While people of CU can only cancel their own events, Officers can cancel all the events.
UC-05	Browse Events	People of CU and Officers can browse the events.
UC-06	Search Events	People of CU and Officers can find events with its type and time.
UC-07	Search Users	Officers can find one's profile with one's name or email.
UC-09	Join Events	When browsing events,

		people of CU can join the events which is available.

2. Features that were not implemented

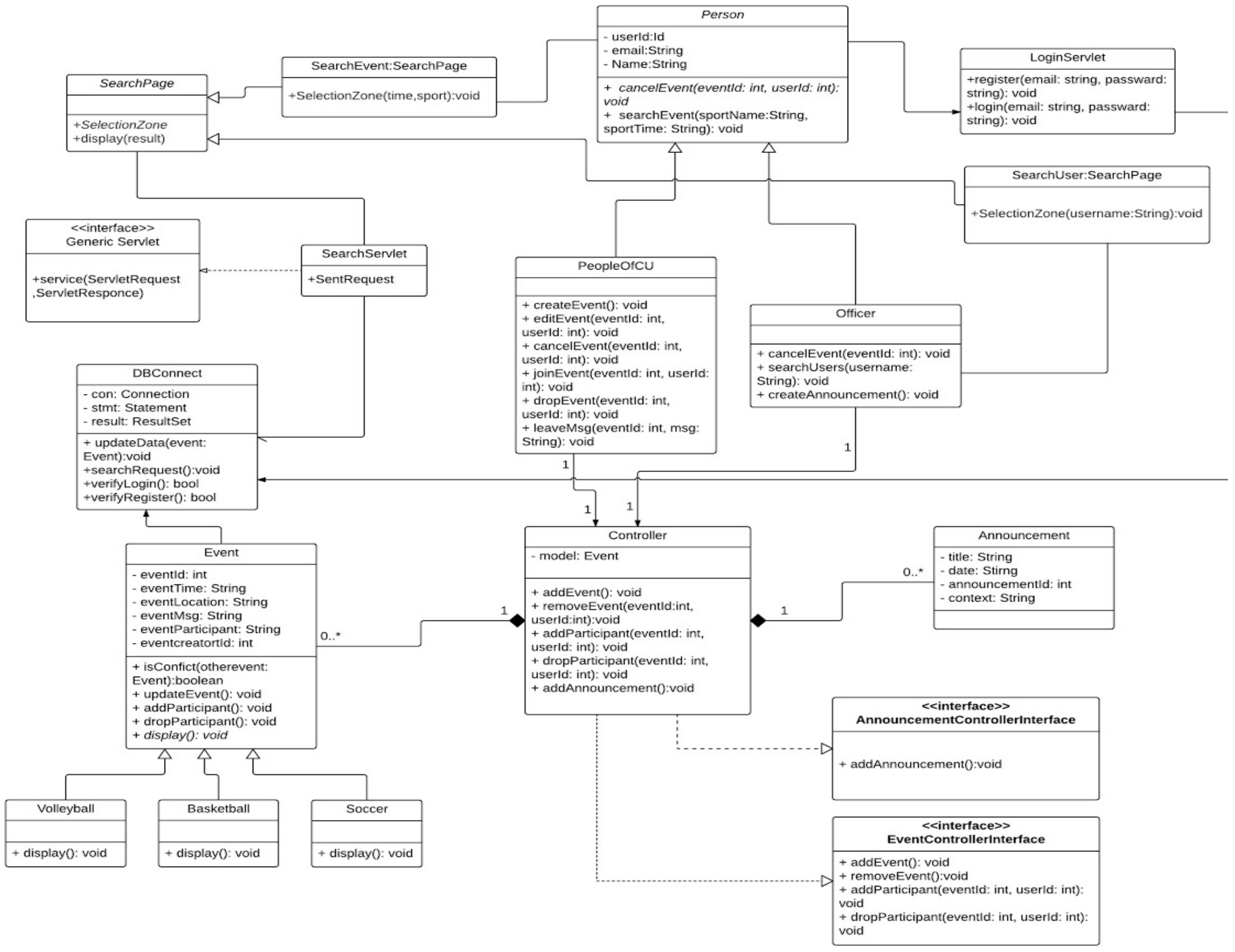
The following table shows which user case were not implemented.

User Case ID:	Use Case Name:	Description:
UC-08	Edit Events	People of CU and Officers can manage the events by editing them. While people of CU can only edit their own events, Officers can edit all the events.
UC-10	Drop Events	When People of CU are involved in an event, they could decide to drop the event or not
UC-11	Announce News	Officers could publish some policy or post
UC-12	Leave Messages	People of CU can leave messages and talk to each other under an event

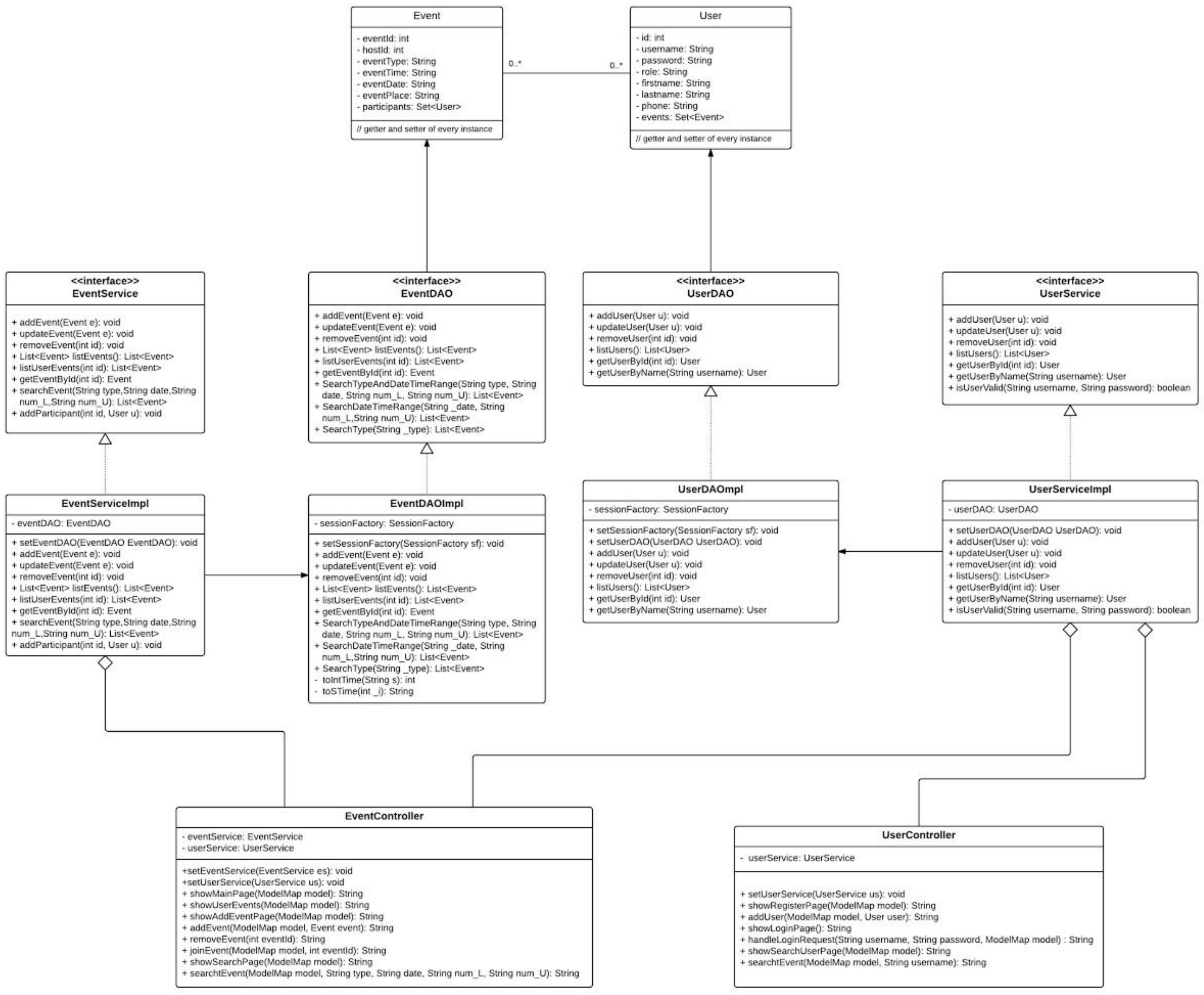
3. Show your Part 2 class diagram and your final class diagram. What changed? Why? If it did not change much, then discuss how doing the design up front helped in the development.

The best way to deal with front end and back end is implementing MVC architecture. Therefore, we changed our previous class diagram into MVC pattern because it is more organized. We decoupled data access and data presentation by Controller class, Data Access class, Server class, and Model class. Besides, we used hibernate to access database and spring to communicate between controllers and views. We handled our views by HTML, Javascript and HTTP request. We have not implement inheritance of User yet, so we set up an instance "role" to identify the type of the users.

Previous Class Diagram(Part 2):


Person
- userId:Id
- email:String
- Name:String
+ cancelEvent(eventId: int, userId: int): void
+ searchEvent(sportName:String, sportTime: String): void
LoginServlet
+register(email: string, passward: string): void
+login(email: string, passward: string): void
SearchPage
+SelectionZone
+display(result)
SearchEvent:SearchPage
+SelectionZone(time,sport):void
SearchUser:SearchPage
+SelectionZone(username:String):void
<<interface>>
Generic Servlet
+service(ServletRequest ,ServletResponce)
SearchServlet
+SentRequest
PeopleOfCU
+ createEvent(): void
+ editEvent(eventId: int, userId: int): void
+ cancelEvent(eventId: int, userId: int): void
+ joinEvent(eventId: int, userId: int): void
+ dropEvent(eventId: int, userId: int): void
+ leaveMsg(eventId: int, msg: String): void
Officer
+ cancelEvent(eventId: int): void
+ searchUsers(username: String): void
+ createAnnouncement(): void
DBConnect
- con: Connection
- stmt: Statement
- result: ResultSet
+ updateData(event: Event):void
+searchRequest():void
+verifyLogin(): bool
+verifyRegister(): bool
1
1
1
1
Controller
- model: Event
+ addEvent(): void
+ removeEvent(eventId:int, userId:int):void
+ addParticipant(eventId: int, userId: int): void
+ dropParticipant(eventId: int, userId: int): void
+ addAnnouncement():void
Announcement
- title: String
- date: Stirng
- announcementId: int
- context: String
0..*
1
Event
- eventId: int
- eventTime: String
- eventLocation: String
- eventMsg: String
- eventParticipant: String
- eventcreatortId: int
+ isConfict(otherevent: Event):boolean
+ updateEvent(): void
+ addParticipant(): void
+ dropParticipant(): void
+ display(): void
0..*
1
Volleyball
+ display(): void
Basketball
+ display(): void
Soccer
+ display(): void
<<interface>>
AnnouncementControllerInterface
+ addAnnouncement():void
<<interface>>
EventControllerInterface
+ addEvent(): void
+ removeEvent():void
+ addParticipant(eventId: int, userId: int): void
+ dropParticipant(eventId: int, userId: int): void

Final Class Diagram:


Event
- eventId: int
- hostId: int
- eventType: String
- eventTime: String
- eventDate: String
- eventPlace: String
- participants: Set<User>
// getter and setter of every instance
0..*
0..*
User
- id: int
- username: String
- password: String
- role: String
- firstname: String
- lastname: String
- phone: String
- events: Set<Event>
// getter and setter of every instance
<<interface>>
EventService
+ addEvent(Event e): void
+ updateEvent(Event e): void
+ removeEvent(int id): void
+ List<Event> listEvents(): List<Event>
+ listUserEvents(int id): List<Event>
+ getEventById(int id): Event
+ searchEvent(String type,String date,String num_L,String num_U): List<Event>
+ addParticipant(int id, User u): void
<<interface>>
EventDAO
+ addEvent(Event e): void
+ updateEvent(Event e): void
+ removeEvent(int id): void
+ List<Event> listEvents(): List<Event>
+ listUserEvents(int id): List<Event>
+ getEventById(int id): Event
+ SearchTypeAndDateTimeRange(String type, String date, String num_L, String num_U): List<Event>
+ SearchDateTimeRange(String _date, String num_L,String num_U): List<Event>
+ SearchType(String _type): List<Event>
<<interface>>
UserDAO
+ addUser(User u): void
+ updateUser(User u): void
+ removeUser(int id): void
+ listUsers(): List<User>
+ getUserById(int id): User
+ getUserByName(String username): User
<<interface>>
UserService
+ addUser(User u): void
+ updateUser(User u): void
+ removeUser(int id): void
+ listUsers(): List<User>
+ getUserById(int id): User
+ getUserByName(String username): User
+ isUserValid(String username, String password): boolean
EventServiceImpl
- eventDAO: EventDAO
+ setEventDAO(EventDAO EventDAO): void
+ addEvent(Event e): void
+ updateEvent(Event e): void
+ removeEvent(int id): void
+ List<Event> listEvents(): List<Event>
+ listUserEvents(int id): List<Event>
+ getEventById(int id): Event
+ searchEvent(String type,String date,String num_L,String num_U): List<Event>
+ addParticipant(int id, User u): void
EventDAOImpl
- sessionFactory: SessionFactory
+ setSessionFactory(SessionFactory sf): void
+ addEvent(Event e): void
+ updateEvent(Event e): void
+ removeEvent(int id): void
+ List<Event> listEvents(): List<Event>
+ listUserEvents(int id): List<Event>
+ getEventById(int id): Event
+ SearchTypeAndDateTimeRange(String type, String date, String num_L, String num_U): List<Event>
+ SearchDateTimeRange(String _date, String num_L,String num_U): List<Event>
+ SearchType(String _type): List<Event>
- toIntTime(String s): int
- toSTime(int _i): String
UserDAOmpl
- sessionFactory: SessionFactory
+ setSessionFactory(SessionFactory sf): void
+ setUserDAO(UserDAO UserDAO): void
+ addUser(User u): void
+ updateUser(User u): void
+ removeUser(int id): void
+ listUsers(): List<User>
+ getUserById(int id): User
+ getUserByName(String username): User
UserServiceImpl
- userDAO: UserDAO
+ setUserDAO(UserDAO UserDAO): void
+ addUser(User u): void
+ updateUser(User u): void
+ removeUser(int id): void
+ listUsers(): List<User>
+ getUserById(int id): User
+ getUserByName(String username): User
+ isUserValid(String username, String password): boolean
EventController
- eventService: EventService
- userService: UserService
+setEventService(EventService es): void
+setUserService(UserService us): void
+ showMainPage(ModelMap model): String
+ showUserEvents(ModelMap model): String
+ showAddEventPage(ModelMap model): String
+ addEvent(ModelMap model, Event event): String
+ removeEvent(int eventId): String
+ joinEvent(ModelMap model, int eventId): String
+ showSearchPage(ModelMap model): String
+ searchtEvent(ModelMap model, String type, String date, String num_L, String num_U): String
UserController
- userService: UserService
+ setUserService(UserService us): void
+ showRegisterPage(ModelMap model): String
+ addUser(ModelMap model, User user): String
+ showLoginPage(): String
+ handleLoginRequest(String username, String password, ModelMap model) : String
+ showSearchUserPage(ModelMap model): String
+ searchtEvent(ModelMap model, String username): String

4. Did you make use of any design patterns in the implementation of your final prototype?If so, how? Show the classes from your class diagram that implement each design pattern (each design pattern as a separate image in the .PDF).If not, where could you make use of design patterns in your system? Show a class diagram of how you could implement each design pattern and compare how it would change from your current class diagram.

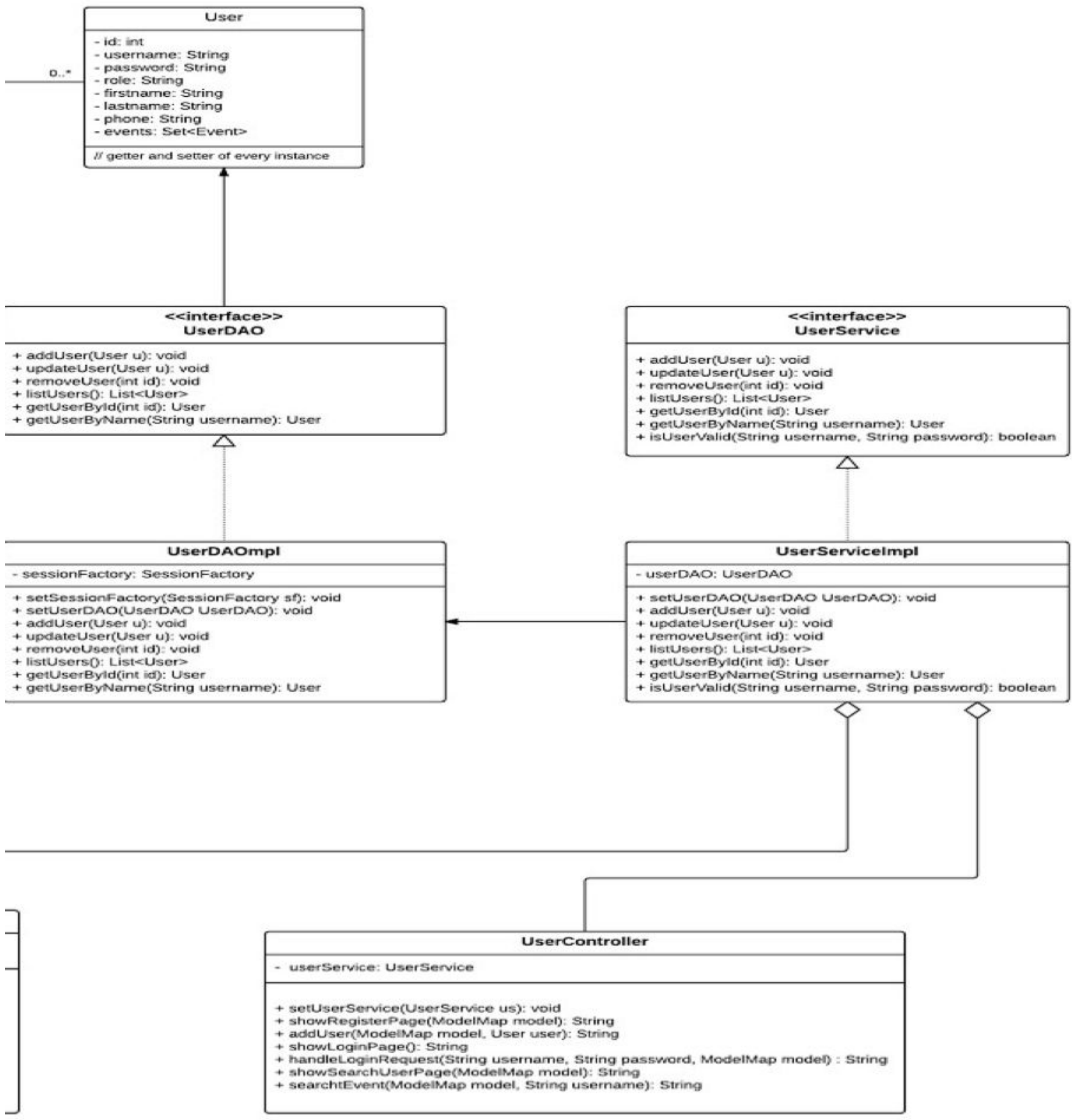
Yes. In our class diagram which is provide in previous page, we used MVC architecture based on Observer, Strategy and Composite design patterns to implement user interfaces. In our architecture, the model directly manages the data, logic, and rules of the application. The view is our output representation of information and the controller accepts input and converts it to commands for the model or view.

Here are the steps of our cycle for the request using MVC:

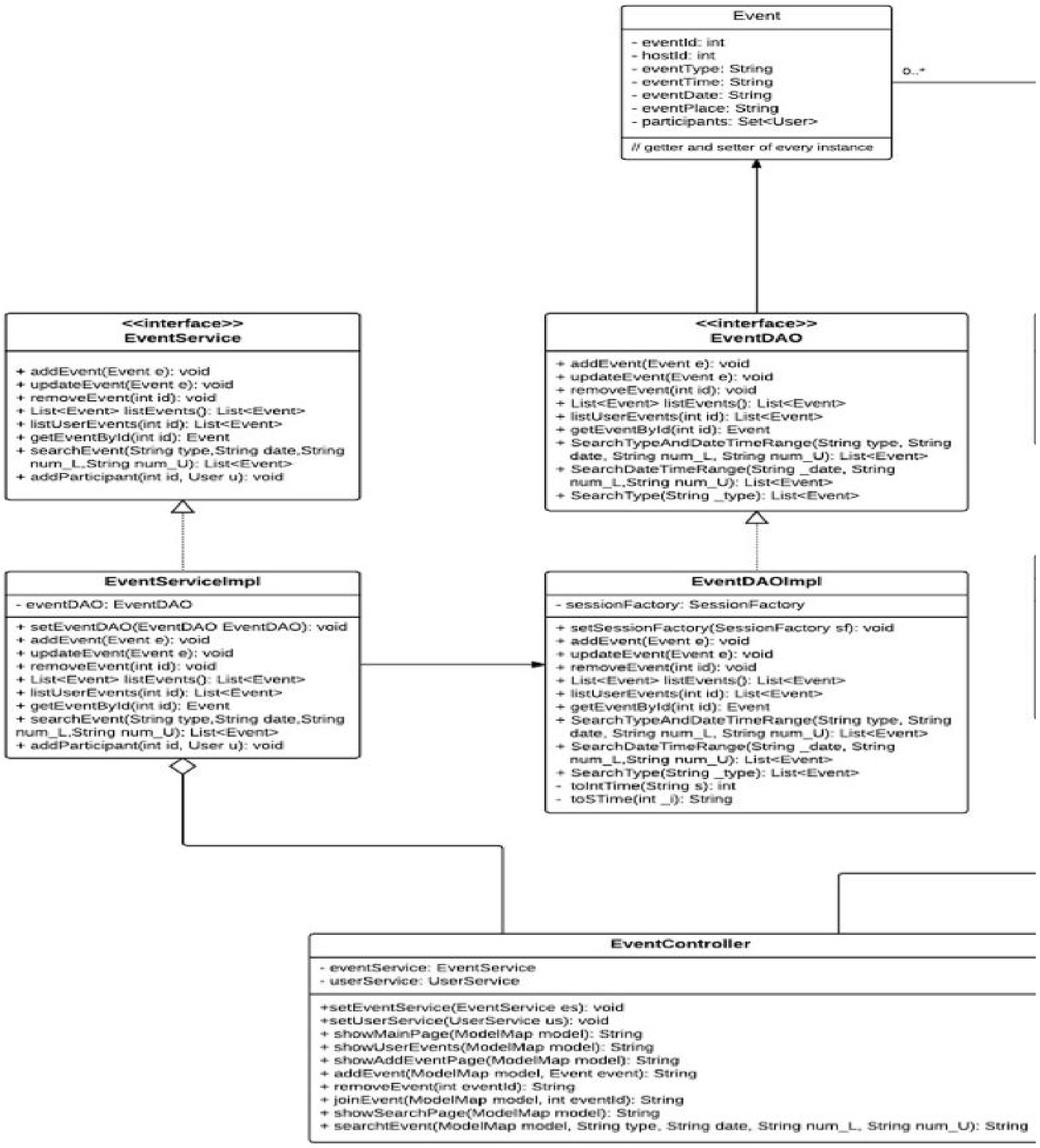
1. The user sends the HTTP request;
2. The controller intercepts it;
3. The controller calls the service;
4. The service calls the dao, which returns some data;
5. The service treats the data, and returns data to the controller;
6. The controller stores the data in the appropriate model and calls the appropriate view;
7. The view get *instantiated* with the model's data, and get returned as the HTTP response

Below is the detailed part (classes) for the class diagram.

User MVC:


0..*
User
- id: int
- username: String
- password: String
- role: String
- firstname: String
- lastname: String
- phone: String
- events: Set<Event>
// getter and setter of every instance
<<interface>>
UserDAO
+ addUser(User u): void
+ updateUser(User u): void
+ removeUser(int id): void
+ listUsers(): List<User>
+ getUserById(int id): User
+ getUserByName(String username): User
<<interface>>
UserService
+ addUser(User u): void
+ updateUser(User u): void
+ removeUser(int id): void
+ listUsers(): List<User>
+ getUserById(int id): User
+ getUserByName(String username): User
+ isUserValid(String username, String password): boolean
UserDAOmpl
- sessionFactory: SessionFactory
+ setSessionFactory(SessionFactory sf): void
+ setUserDAO(UserDAO UserDAO): void
+ addUser(User u): void
+ updateUser(User u): void
+ removeUser(int id): void
+ listUsers(): List<User>
+ getUserById(int id): User
+ getUserByName(String username): User
UserServiceImpl
- userDAO: UserDAO
+ setUserDAO(UserDAO UserDAO): void
+ addUser(User u): void
+ updateUser(User u): void
+ removeUser(int id): void
+ listUsers(): List<User>
+ getUserById(int id): User
+ getUserByName(String username): User
+ isUserValid(String username, String password): boolean
UserController
- userService: UserService
+ setUserService(UserService us): void
+ showRegisterPage(ModelMap model): String
+ addUser(ModelMap model, User user): String
+ showLoginPage(): String
+ handleLoginRequest(String username, String password, ModelMap model) : String
+ showSearchUserPage(ModelMap model): String
+ searchtEvent(ModelMap model, String username): String

Event MVC:


Event
- eventId: int
- hostId: int
- eventType: String
- eventTime: String
- eventDate: String
- eventPlace: String
- participants: Set<User>
// getter and setter of every instance
0..*
<<interface>>
EventService
+ addEvent(Event e): void
+ updateEvent(Event e): void
+ removeEvent(int id): void
+ List<Event> listEvents(): List<Event>
+ listUserEvents(int id): List<Event>
+ getEventById(int id): Event
+ searchEvent(String type,String date,String num_L,String num_U): List<Event>
+ addParticipant(int id, User u): void
<<interface>>
EventDAO
+ addEvent(Event e): void
+ updateEvent(Event e): void
+ removeEvent(int id): void
+ List<Event> listEvents(): List<Event>
+ listUserEvents(int id): List<Event>
+ getEventById(int id): Event
+ SearchTypeAndDateTimeRange(String type, String date, String num_L, String num_U): List<Event>
+ SearchDateTimeRange(String _date, String num_L,String num_U): List<Event>
+ SearchType(String _type): List<Event>
EventServiceImpl
- eventDAO: EventDAO
+ setEventDAO(EventDAO EventDAO): void
+ addEvent(Event e): void
+ updateEvent(Event e): void
+ removeEvent(int id): void
+ List<Event> listEvents(): List<Event>
+ listUserEvents(int id): List<Event>
+ getEventById(int id): Event
+ searchEvent(String type,String date,String num_L,String num_U): List<Event>
+ addParticipant(int id, User u): void
EventDAOImpl
- sessionFactory: SessionFactory
+ setSessionFactory(SessionFactory sf): void
+ addEvent(Event e): void
+ updateEvent(Event e): void
+ removeEvent(int id): void
+ List<Event> listEvents(): List<Event>
+ listUserEvents(int id): List<Event>
+ getEventById(int id): Event
+ SearchTypeAndDateTimeRange(String type, String date, String num_L, String num_U): List<Event>
+ SearchDateTimeRange(String _date, String num_L,String num_U): List<Event>
+ SearchType(String _type): List<Event>
- toIntTime(String s): int
- toSTime(int _i): String
EventController
- eventService: EventService
- userService: UserService
+setEventService(EventService es): void
+setUserService(UserService us): void
+ showMainPage(ModelMap model): String
+ showUserEvents(ModelMap model): String
+ showAddEventPage(ModelMap model): String
+ addEvent(ModelMap model, Event event): String
+ removeEvent(int eventId): String
+ joinEvent(ModelMap model, int eventId): String
+ showSearchPage(ModelMap model): String
+ searchtEvent(ModelMap model, String type, String date, String num_L, String num_U): String

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?

In the beginning of the course, we learned UML which is useful in providing a standard way to visualize the design of a system. The software architecture is the blueprint for the system; it allows reasoning about performance, availability, security, and other attributes, allows planning for incremental development, and guides work assignments and tracking. Now we could use UML to explain why should we document the architecture which is important to communicate with people who need to use, design or cowork in a system. Besides, we have learned some principles of Object Oriented Design during this class. They are good guidelines when we're lost in creating and designing a system. Also, design patterns can speed up the development process by providing tested, proven development paradigms. We've understood to communicate using well-known, well understood names for software interactions by using design patterns and applied them to improve the systems and make systems more robust. Moreover, the Spring Framework supports us the integration with Hibernate, Java Persistence API and Java Data Objects for resource management, data access object implementations, and transaction strategies. Generally, now we could address the use of object-oriented techniques in a system creation, design and implementation by using the tutorials taught in this course.