Event Create and Share

Part 5

Team

Shen Huang
Yan Li
Amir Kashipazha

1. List the features that were implemented (table with ID and title).
We implemented the following user requirements:

User Requirements	
ID	**Requirements**
UR-01	As a user, I need to **create an account** so that I can use the system.
UR-02	As a user, I need a username and password to **login to the system** so that I can create, search or join some events.
UR-03	As a user, I need to **logout of my account** when I finish working on the system.
UR-04	As a user, I need to **update my account information** so that I can change my first name, last name, phone number or password.
UR-06	As a user, I need to **create an event** so that I can find some people to join me in activities I am interested in.
UR-07	As a user, I need to **join events** created by other users so that I can participate in some interesting events.
UR-08	As a user, I need to **look for events** so I can find events I am interested in joining.
UR-09	As a user, I need to **close an event** that I have created so I can cancel an event due to problematic circumstances, such as heavy snowfall.
UR-10	As a user, I need to **leave an event** that I previously joined but cannot attend so the creator of the event has a more accurate count of who is attending and so people on the waiting list can join the event.
UR-11	As a user, I need to **make comments** for the event which I created or

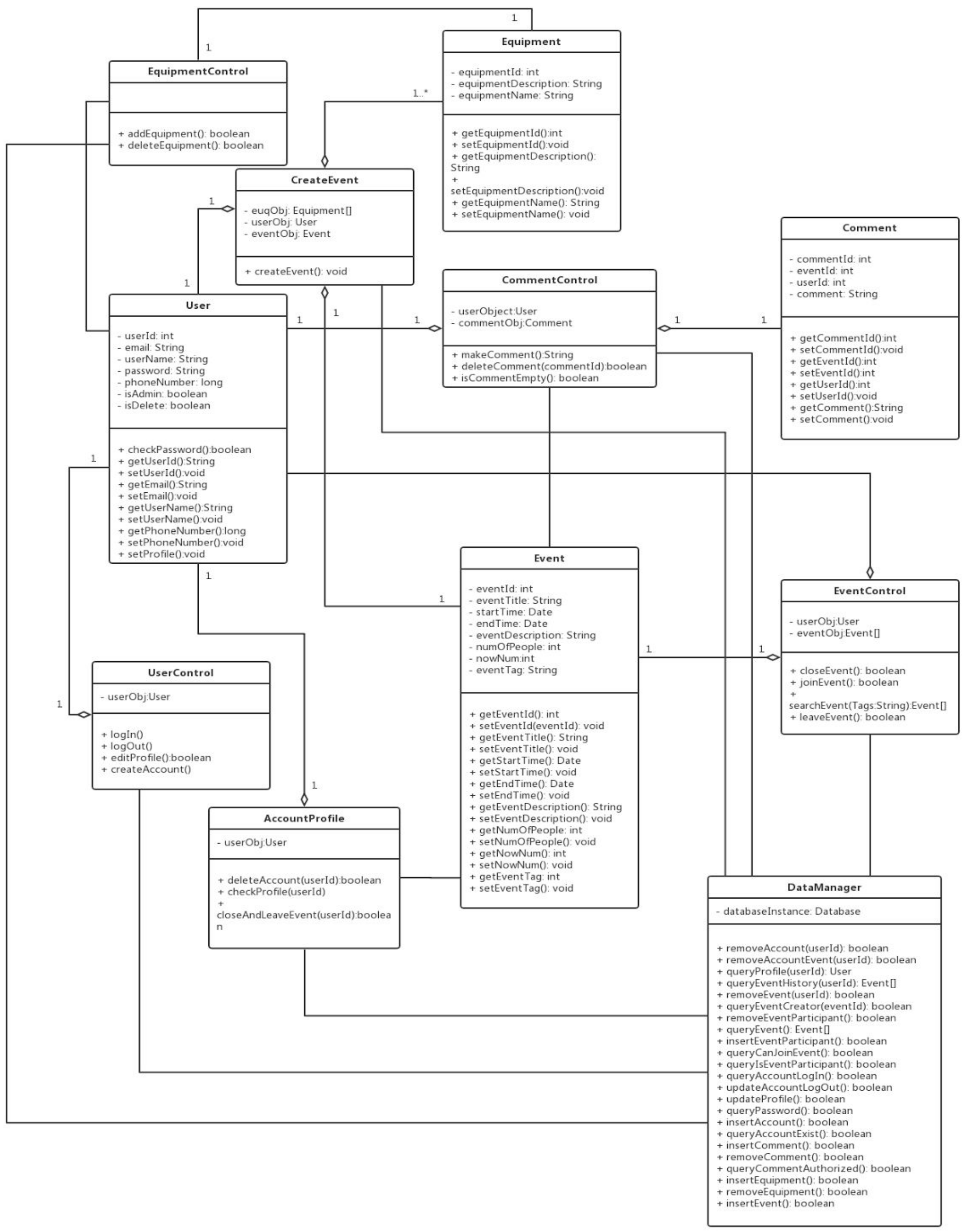
	participated in so that I can give other users some tips about the event or equipment they need.
UR-13	As a user, I need to **check my profile** so that I can get the information regarding myself.
UR-14	As an admin, I need to **add some equipment** so that users know what equipment is needed for a specific event.
UR-15	As an admin, I need to **delete some equipment** which users do not need anymore for a specific event.

2. List the features which were not implemented from Part 2 (table with ID and title).

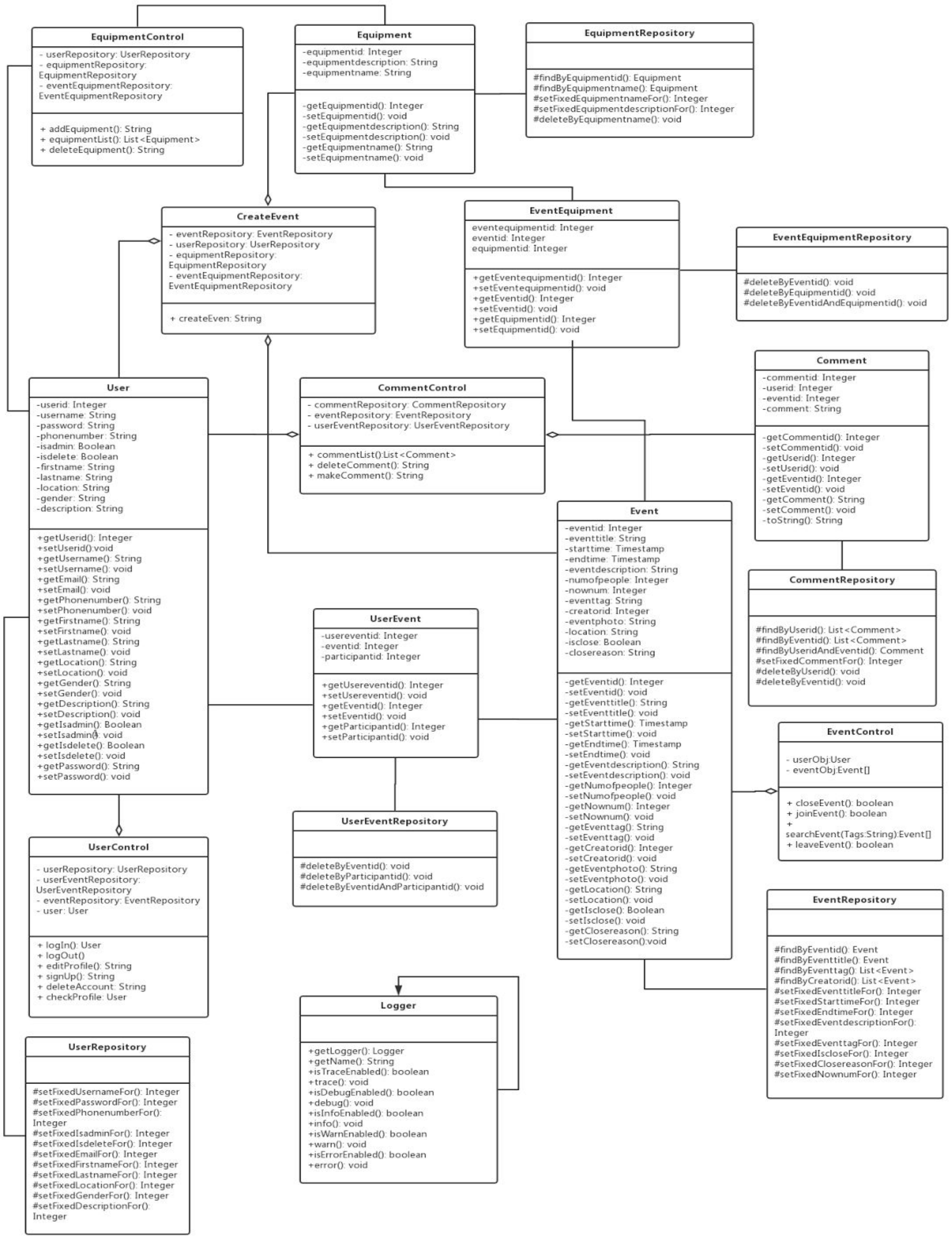
User Requirements	
ID	**Requirements**
UR-05	As a user, I need to **delete my account** when I am not interested in using the system.
UR-12	As a user, I need to **delete my own comments** which I don't want other users to see.

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.

- Data Manager class changes to repository class like EquipmentRepository, UserRepository, *etc.* because we separate these insert, select, delete, query operation to each class.
- Added several variables because we need more variables to implement our project. For example, we added a gender variable and a description variable to User class.
- Added several methods to use these added variables.



Part 2 Class Diagram

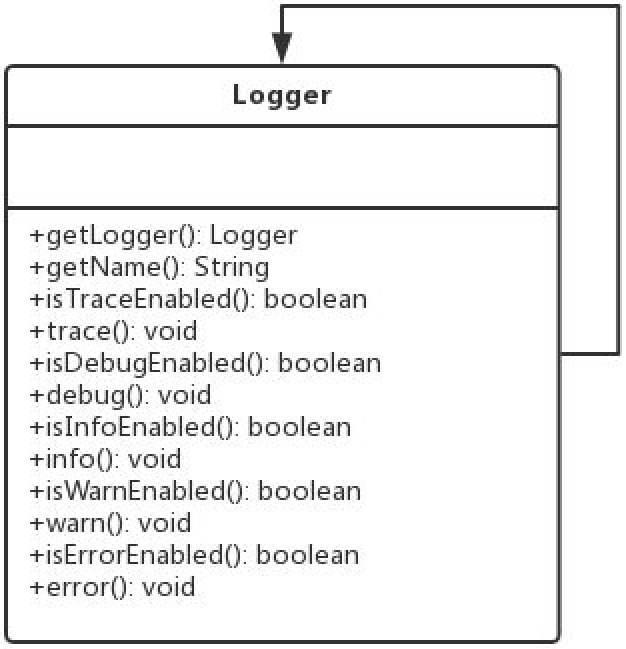


Part 5 Class Diagram

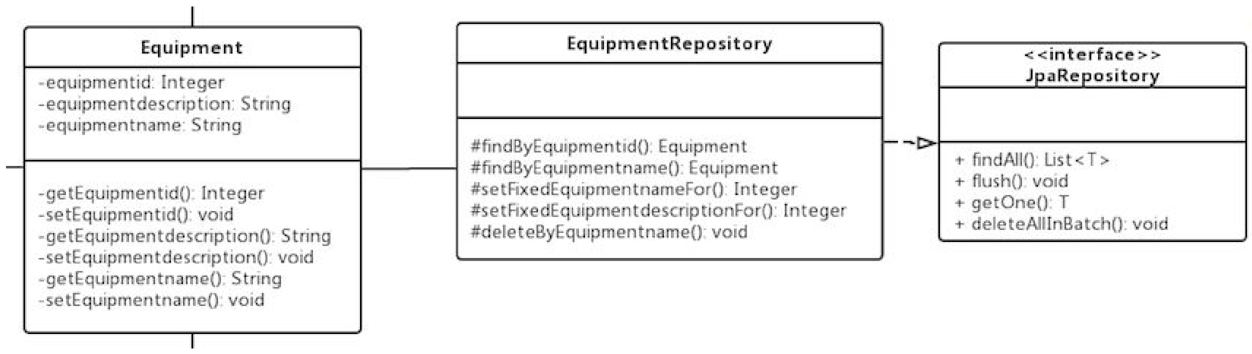
4. Did you make use of any design patterns in the implementation of your final prototype?
If so, how? Show the classes from your class diagram that implement each design pattern (each design pattern as a separate image in the .PDF).
If not, where could you make use of design patterns in your system?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

We used the **singleton**, **strategy**, and **repository** design patterns for this project.

The **Singleton** design pattern was used to implement Logger class which restrict the instantiation of a class to a single object.

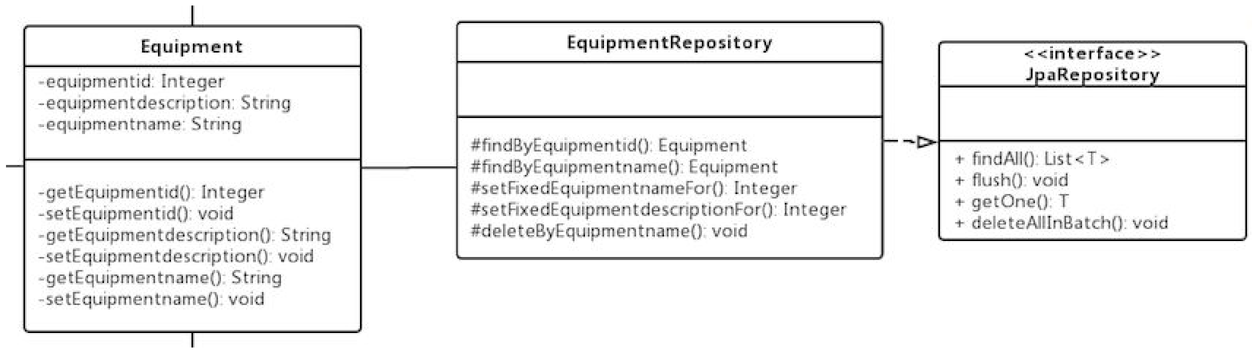

Logger
+getLogger(): Logger
+getName(): String
+isTraceEnabled(): boolean
+trace(): void
+isDebugEnabled(): boolean
+debug(): void
+isInfoEnabled(): boolean
+info(): void
+isWarnEnabled(): boolean
+warn(): void
+isErrorEnabled(): boolean
+error(): void

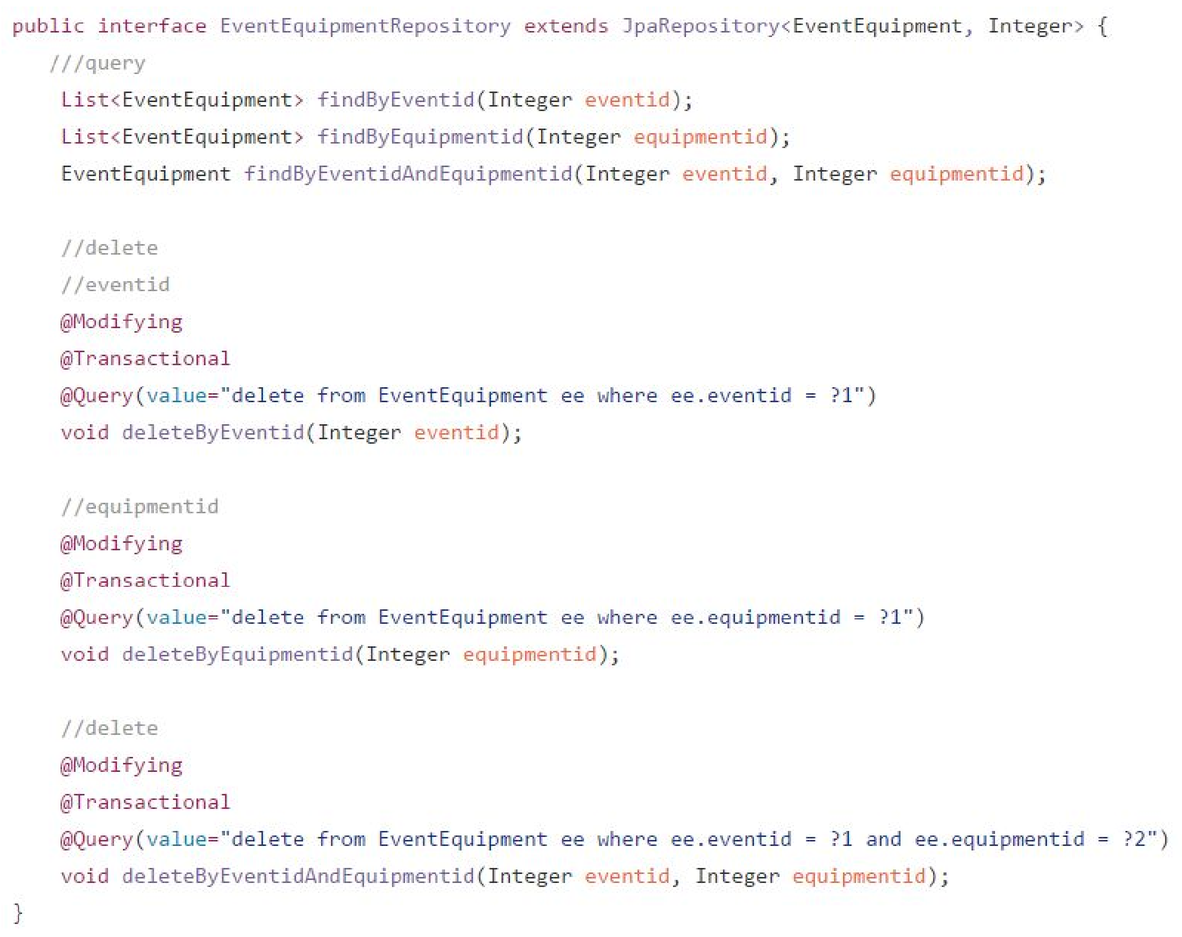
The **repository** design pattern was used to add a layer between the domain and the data mapping layers to isolate domain objects from details of the database access code, to minimize scattering, and to avoid duplication of query code. We use JpaRepository to implement the repository design pattern.


Equipment
-equipmentid: Integer
-equipmentdescription: String
-equipmentname: String
-getEquipmentid(): Integer
-setEquipmentid(): void
-getEquipmentdescription(): String
-setEquipmentdescription(): void
-getEquipmentname(): String
-setEquipmentname(): void
EquipmentRepository
#findByEquipmentid(): Equipment
#findByEquipmentname(): Equipment
#setFixedEquipmentnameFor(): Integer
#setFixedEquipmentdescriptionFor(): Integer
#deleteByEquipmentname(): void
<<interface>>
JpaRepository
+ findAll(): List<T>
+ flush(): void
+ getOne(): T
+ deleteAllInBatch(): void

A controller object implements the **Strategy** design pattern for one or more view objects. It delegates to the controller all decisions about the application-specific meaning of the interface behavior.

The Repositories interfaces and classes in our project uses the Strategy design pattern. The JpaRepository is in the framework.


Equipment
-equipmentid: Integer
-equipmentdescription: String
-equipmentname: String
-getEquipmentid(): Integer
-setEquipmentid(): void
-getEquipmentdescription(): String
-setEquipmentdescription(): void
-getEquipmentname(): String
-setEquipmentname(): void
EquipmentRepository
#findByEquipmentid(): Equipment
#findByEquipmentname(): Equipment
#setFixedEquipmentnameFor(): Integer
#setFixedEquipmentdescriptionFor(): Integer
#deleteByEquipmentname(): void
<<interface>>
JpaRepository
+ findAll(): List<T>
+ flush(): void
+ getOne(): T
+ deleteAllInBatch(): void



```
public interface EventEquipmentRepository extends JpaRepository<EventEquipment, Integer> {
   ///query
    List<EventEquipment> findByEventid(Integer eventid);
    List<EventEquipment> findByEquipmentid(Integer equipmentid);
    EventEquipment findByEventidAndEquipmentid(Integer eventid, Integer equipmentid);

    //delete
    //eventid
    @Modifying
    @Transactional
    @Query(value="delete from EventEquipment ee where ee.eventid = ?1")
    void deleteByEventid(Integer eventid);

    //equipmentid
    @Modifying
    @Transactional
    @Query(value="delete from EventEquipment ee where ee.equipmentid = ?1")
    void deleteByEquipmentid(Integer equipmentid);

    //delete
    @Modifying
    @Transactional
    @Query(value="delete from EventEquipment ee where ee.eventid = ?1 and ee.equipmentid = ?2")
    void deleteByEventidAndEquipmentid(Integer eventid, Integer equipmentid);
}
```

The class diagram is in the ProcessOn studio.

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?

We have learned several important things:

- Diagrams makes it easier to visualize how components interact with each other.
- Design patterns are a really helpful guide to implementing good practices.
- Analysis and design is important. Project follows use cases, class diagram and design patterns.
- Requirements should be concise depending on the user’s perspective.
- We learned how to use different design patterns and frameworks.
- We learned HTML, CSS, and Bootstrap for making webpages.