26. FITNESS MANAGER

Project Part 5 – Final Report

Chaitanya Soma
Ramnarayan Krishnamurthy
Jordan Peters

1. *List the features that were implemented (table with ID and title).*

The following Use Cases were implemented:

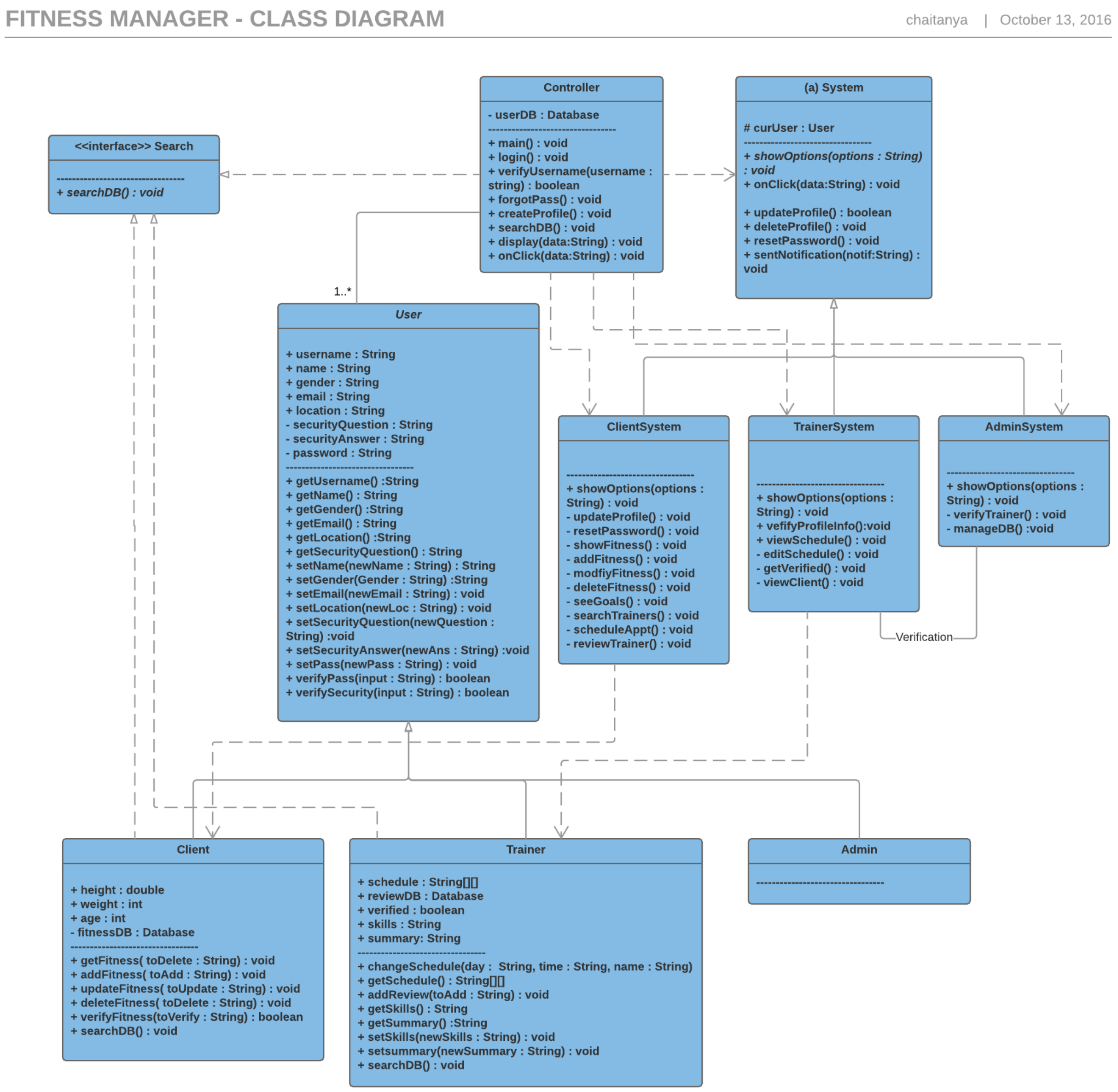
ID	**Use Case**
UC-01	Create Client Profile
UC-02	Create Trainer Profile
UC-03	Create Admin Profile
UC-04	Login User
UC-10	Search Trainer
UC-11	Add Daily Fitness
UC-12	Modify Daily Fitness
UC-13	Delete Daily Fitness
UC-14	Approve Profile
UC-15	Schedule Appointment
UC-16	Create Appointment Slots

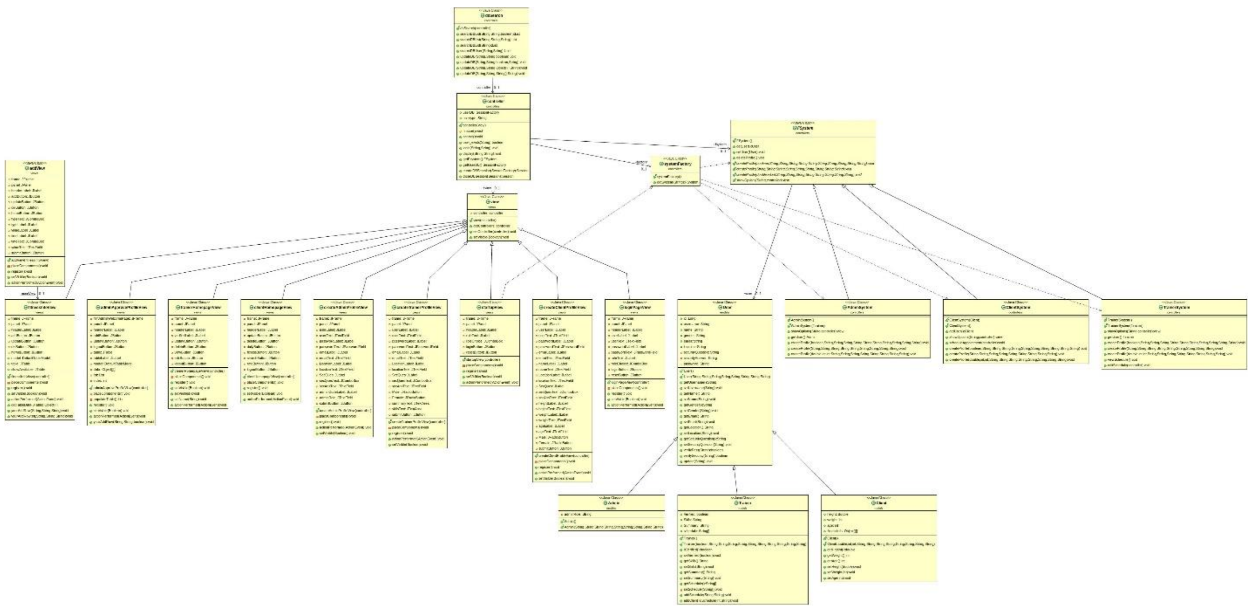
2. List the features that were not implemented from Part 2 (table with ID and title).

The Use cases not implemented were Reset Password, Profile Update/Delete and Fitness Goals.
Fitness Goals were simply a redundant copy of the Daily Fitness Info.

ID	**Requirement**
UC-05	Reset Password via Security Question
UC-06	Reset Password via email
UC-07	Update Profile
UC-08	Delete Profile
UC-09	Review Trainer
UC-17	Add Fitness Goals
UC-18	Modify Fitness Goals
UC-19	Delete Fitness Goals

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.*


FITNESS MANAGER - CLASS DIAGRAM
chaitanya | October 13, 2016
<<interface>> Search
+ searchDB() : void
Controller
- userDB : Database
+ main() : void
+ login() : void
+ verifyUsername(username : string) : boolean
+ forgotPass() : void
+ createProfile() : void
+ searchDB() : void
+ display(data:String) : void
+ onClick(data:String) : void
(a) System
# curUser : User
+ showOptions(options : String) : void
+ onClick(data:String) : void
+ updateProfile() : boolean
+ deleteProfile() : void
+ resetPassword() : void
+ sentNotification(notif:String) : void
1..*
User
+ username : String
+ name : String
+ gender : String
+ email : String
+ location : String
- securityQuestion : String
- securityAnswer : String
- password : String
+ getUsername() :String
+ getName() : String
+ getGender() :String
+ getEmail() : String
+ getLocation() :String
+ getSecurityQuestion() : String
+ setName(newName : String) : String
+ setGender(Gender : String) :String
+ setEmail(newEmail : String) : void
+ setLocation(newLoc : String) : void
+ setSecurityQuestion(newQuestion : String) :void
+ setSecurityAnswer(newAns : String) :void
+ setPass(newPass : String) : void
+ verifyPass(input : String) : boolean
+ verifySecurity(input : String) : boolean
ClientSystem
+ showOptions(options : String) : void
- updateProfile() : void
- resetPassword() : void
- showFitness() : void
- addFitness() : void
- modfiyFitness() : void
- deleteFitness() : void
- seeGoals() : void
- searchTrainers() : void
- scheduleAppt() : void
- reviewTrainer() : void
TrainerSystem
+ showOptions(options : String) : void
+ vefifyProfileInfo():void
+ viewSchedule() : void
- editSchedule() : void
- getVerified() : void
- viewClient() : void
AdminSystem
+ showOptions(options : String) : void
- verifyTrainer() : void
- manageDB() :void
Verification
Client
+ height : double
+ weight : int
+ age : int
- fitnessDB : Database
+ getFitness(toDelete : String) : void
+ addFitness(toAdd : String) : void
+ updateFitness(toUpdate : String) : void
+ deleteFitness(toDelete : String) : void
+ verifyFitness(toVerify : String) : boolean
+ searchDB() : void
Trainer
+ schedule : String[][]
+ reviewDB : Database
+ verified : boolean
+ skills : String
+ summary: String
+ changeSchedule(day : String, time : String, name : String)
+ getSchedule() : String[][]
+ addReview(toAdd : String) : void
+ getSkills() : String
+ getSummary() :String
+ setSkills(newSkills : String) : void
+ setsummary(newSummary : String) : void
+ searchDB() : void
Admin



The class diagrams of part 2 and final project look similar for the most parts. In the class diagram in part 2, we have not accounted for the classes responsible for view part of the MVC and have updated the diagram to reflect that. The new class diagram also implements the factory design pattern and SystemFactory class has been added appropriately.

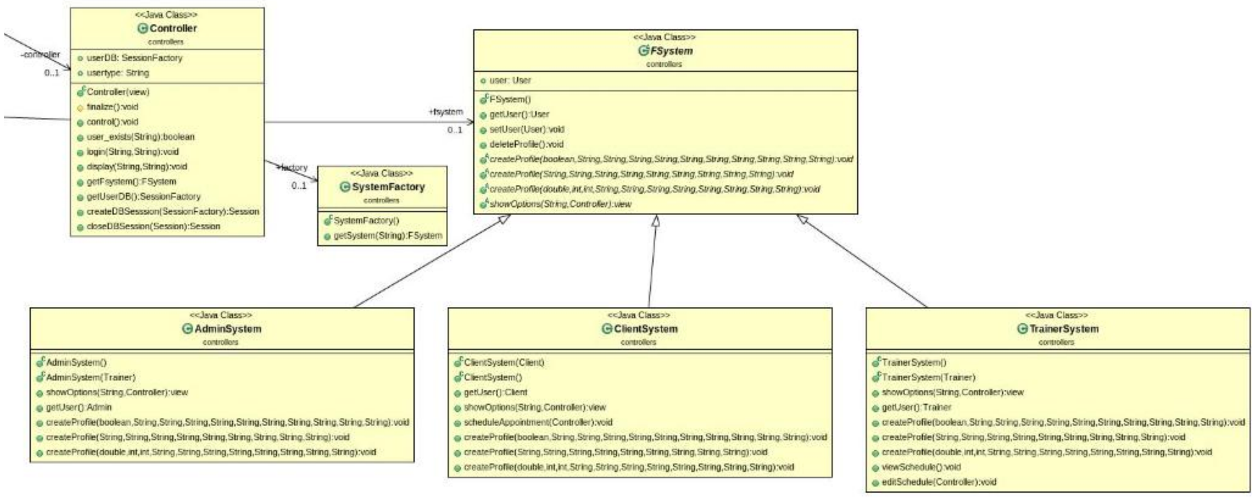
In the older class diagram/design the database search was implemented as an interface to potentially implement the iterator design pattern and create separate databases for Client and Trainers. This was redesigned after learning hibernate and the new class diagram uses the DbSearch class to implement all database transactions.

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 made use of the strategy and factory design patterns in the implementation of our final prototype.

<u>Model View Controller</u> architectural pattern has been used to implement the project. In addition to MVC we used the Factory Design Pattern. For the future we would also incorporate the Strategy Design Pattern as explained below.

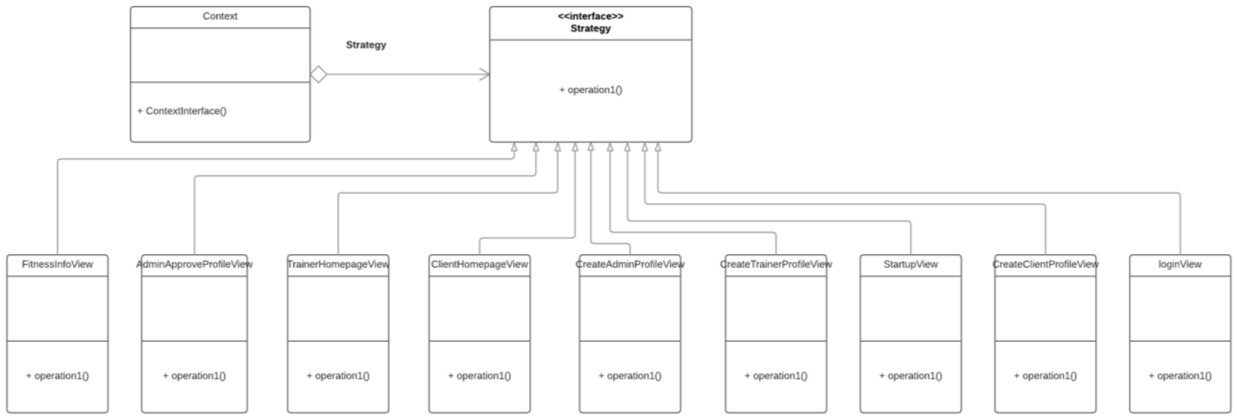
Factory Design Pattern:
Our ‘System’ class group makes use of the factory design pattern. After what type of user if logging in is determined we create an instance of either ClientSystem, TrainerSystem or AdminSystem (all subclasses of the abstract superclass System) at runtime using the SystemFactory class.



<<Java Class>>
Controller
controllers
userDB: SessionFactory
usertype: String
Controller(view)
finalize():void
control():void
user_exists(String):boolean
login(String,String):void
display(String,String):void
getFsystem():FSystem
getUserDB():SessionFactory
createDBSesssion(SessionFactory):Session
closeDBSession(Session):Session
-controller
0..1
+fsystem
0..1
+factory
0..1
<<Java Class>>
SystemFactory
controllers
SystemFactory()
getSystem(String):FSystem
<<Java Class>>
FSystem
controllers
user: User
FSystem()
getUser():User
setUser(User):void
deleteProfile():void
createProfile(boolean,String,String,String,String,String,String,String,String,String,String):void
createProfile(String,String,String,String,String,String,String,String,String):void
createProfile(double,int,int,String,String,String,String,String,String,String,String):void
showOptions(String,Controller):view
<<Java Class>>
AdminSystem
controllers
AdminSystem()
AdminSystem(Trainer)
showOptions(String,Controller):view
getUser():Admin
createProfile(boolean,String,String,String,String,String,String,String,String,String,String):void
createProfile(String,String,String,String,String,String,String,String,String):void
createProfile(double,int,int,String,String,String,String,String,String,String,String):void
<<Java Class>>
ClientSystem
controllers
ClientSystem(Client)
ClientSystem()
getUser():Client
showOptions(String,Controller):view
scheduleAppointment(Controller):void
createProfile(boolean,String,String,String,String,String,String,String,String,String,String):void
createProfile(String,String,String,String,String,String,String,String,String):void
createProfile(double,int,int,String,String,String,String,String,String,String,String):void
<<Java Class>>
TrainerSystem
controllers
TrainerSystem()
TrainerSystem(Trainer)
showOptions(String,Controller):view
getUser():Trainer
createProfile(boolean,String,String,String,String,String,String,String,String,String,String):void
createProfile(String,String,String,String,String,String,String,String,String):void
createProfile(double,int,int,String,String,String,String,String,String,String,String):void
viewSchedule():void
editSchedule(Controller):void

Strategy Design Pattern:

The view class and all the corresponding subclasses where not added as a part of the previous class diagram. The View class implements some standard functions which all the subclasses implement and override. This provides the controller a unified medium to work with all the views. In the future, we would like to implement the view using strategy design pattern instead of using inheritance. This can be done by replacing the View superclass with an interface and have all view subclasses implemented the interface. This helps us decouple behavior with the class that is using the behavior. Strategy pattern can be implemented as shown in the diagram below.



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?*

After completing the process of creating, designing and implementing a system, we have learned that spending a significant portion of time on designing is extremely helpful in implementing it, and ends up saving a ton of time. However, due to our inexperience, it we've also learned that actually going through this process was in itself extremely important. This was because there were a lot of things that needed to be added, removed or changed because we were unable to fully conceptualize the final system. This, to us, seemed like something that would come with time and experience, and it may also help to spend more time thinking about things such as how things fit together and what input parameters and returns functions would need. In addition, we noticed that object oriented approach improved the maintainability of the code and also made integrating additional functionality simpler.