# University of Colorado Boulder

Object Oriented Analysis and Design
CSCI 5448

# Project Report

## NextBox

**Submitted by: Group no. 43**


**Members:**

Anirudh Tiwari
Anirudh.Tiwari@colorado.edu

Saurabh Sood
Saurabh.Sood@colorado.edu

Daniel Milroy
Daniel.Milroy@colorado.edu

Sneha Parmar
Sneha.Parmar@colorado.edu

**Features that were implemented**

| ***Use Case*** | ***Description*** |
|---|---|
| UC 01 | Create an Account |
| UC 02 | Add Credit Card |
| UC 03 | Rename File |
| UC 04 | Delete file |
| UC 07 | Create Directory |
| UC 08 | Delete Directory |
| UC 09 | Move Directory |
| UC 10 | Rename Directory |
| UC 11 | Upload File |
| UC 12 | Download Files |
| UC 14 | View Photo |
| UC 15 | Search for file or directory |
| UC 16 | See account usage |
| UC 17 | Change Plan |
| UC 18 | Share File |
| UC 19 | Check Bill |
| UC 20 | Create User |
| UC 21 | Change activation status |
| UC 22 | Create Plan |
| UC 23 | Modify Plan |

**Features that were not implemented**

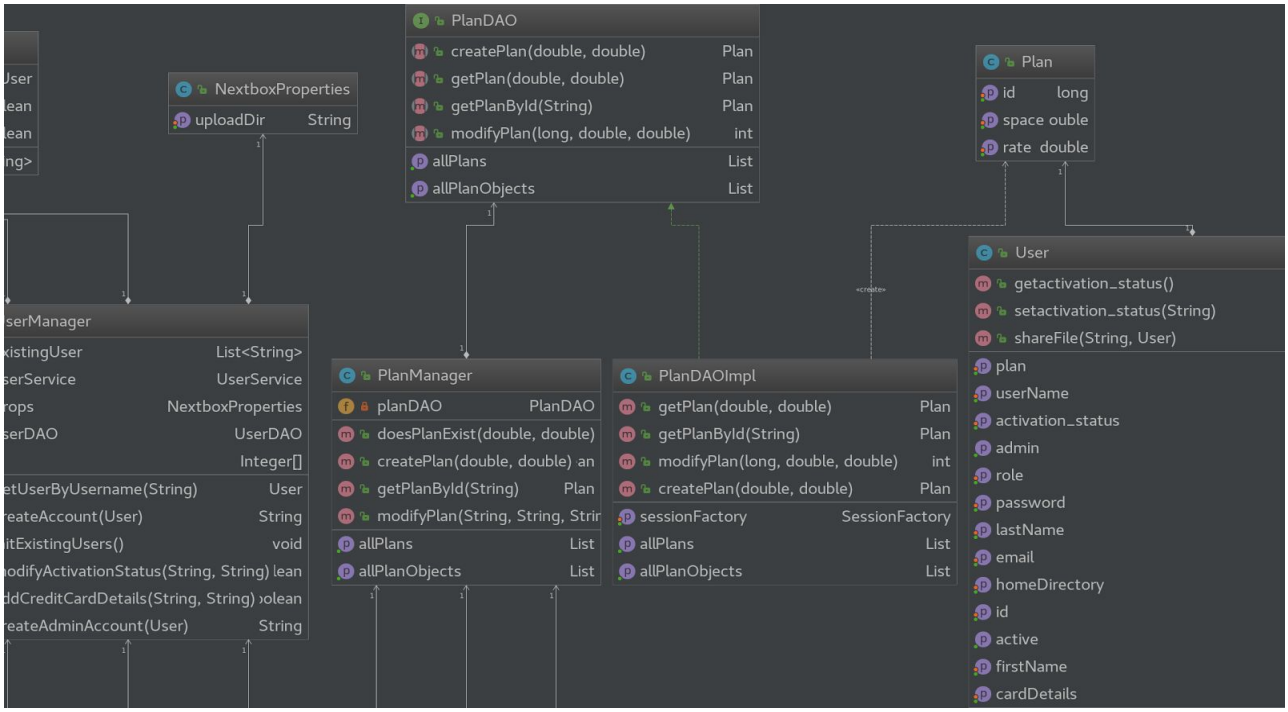
| ***Use Case*** | ***Description*** |
|---|---|
| UC 05 | Copy File |
| UC 06 | Paste File |
| UC 13 | View Text File |
| UC 24 | Edit Text File |

**Class Diagram**

We changed our class diagram a lot from our submission in Part 2. We discovered that files, directories, and filesystem paths could be more efficiently handled by an overarching class (Filepath) that would display and manipulate these objects. We realised that the interaction with the database is a perfect fit for the Proxy design pattern, which our Part 2 design did not accommodate. We added the Proxy classes and the required interfaces during the implementation. Also, we ended up removing a lot of classes for the use cases we could not implement (Such as the Clipboard class, which would have been used for the Copy-Paste file use case). Note that our new class diagram appears to have four dangling classes: FilesystemAPI, FilesystemService, IndexController, and BillingService. These classes have static methods that are called from many other classes rather than instantiated. While it may be advantageous to convert these methods to allow polymorphism, we did not have time for a code refactor. See page 7 for our Part 2 class diagram and our final class diagram on page 8.

## Design Patterns

Our project made heavy use of the Proxy design pattern, which we used to interact with the Database.




In the above class diagram, we have a proxy class called **PlanManager,** an interface called PlanDAO, and a concrete class called **PlanDAOImpl**, which implements **PlanDAO**. The proxy class comprises of a **PlanDAOImpl** object, and can use it to call methods like **getPlan**, which create or delete Plan objects from the database.

## What we learned

The overall design process really helped us plan out the project well. The diagrams helped us to avoid potential problems during the development, and made development easier and faster. The Activity Diagrams really helped us flesh out what the User Interaction will be for a particular

Use Case. Moreover, the Sequence Diagram really helped us in the coding process, as we knew beforehand what classes we need to create, and what methods we need to implement. It really helped speed up the coding process.

We had a great learning curve with using Java and Spring MVC and we learnt how to think in terms of the MVC architecture. It greatly simplified organization of code for a web application like ours.

We also learnt how to think from different perspectives for a software product - starting with thinking like a customer and a solution or program manager and progressing more towards the view of a software developer. It was interesting to experience how they could have differences of opinion in terms of scope in the real world, and to realize how it would make things easier when everyone is on the same page right from the start.

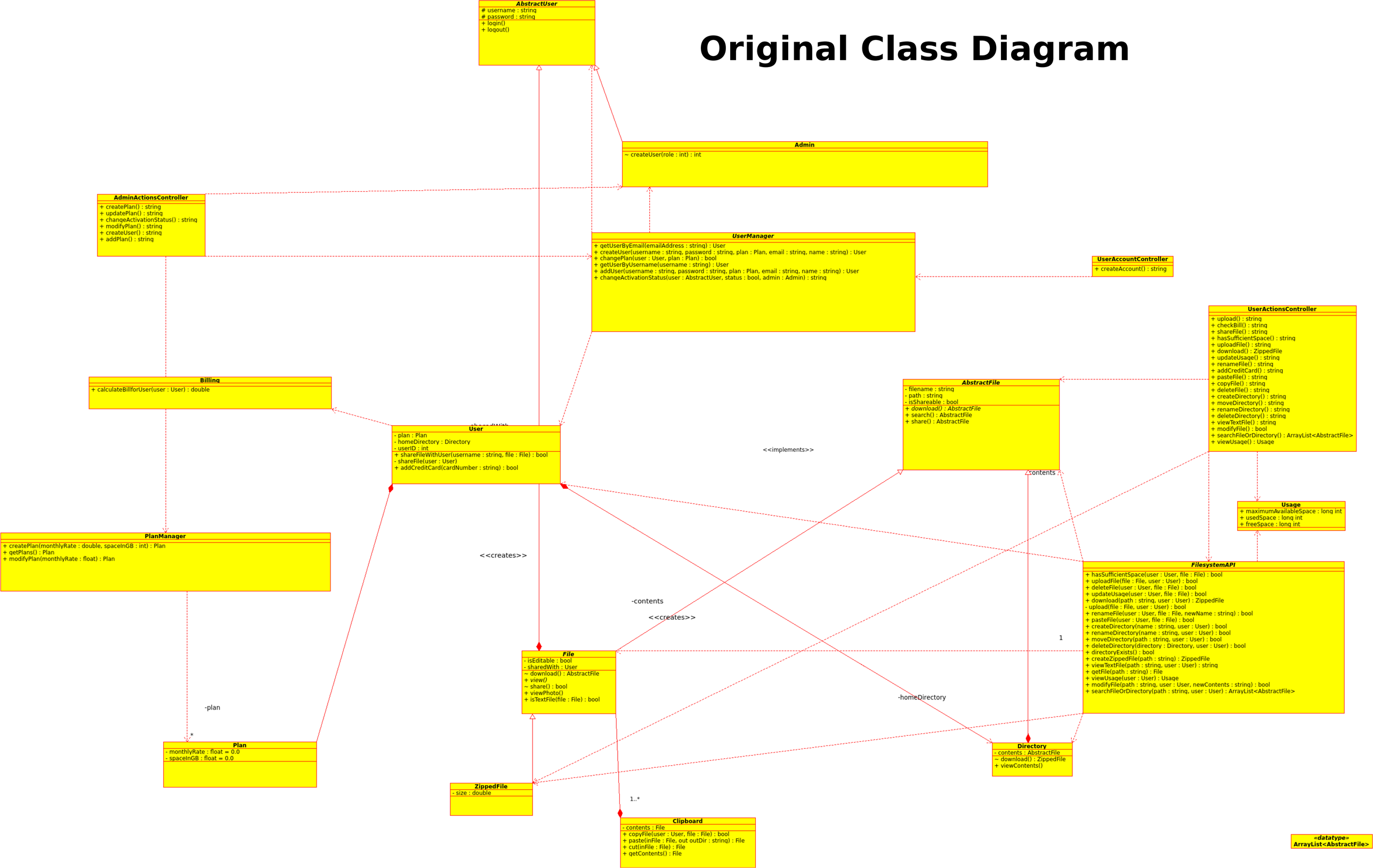
More generally, we learned how to work as a team to build a cohesive software product in GitHub. The organizational logistics necessary to coordinate simultaneous code revisions and changes can be difficult even for a semester project. We learned to collaboratively review pull requests, as well as resolve complex merge conflicts when pulling in new features.

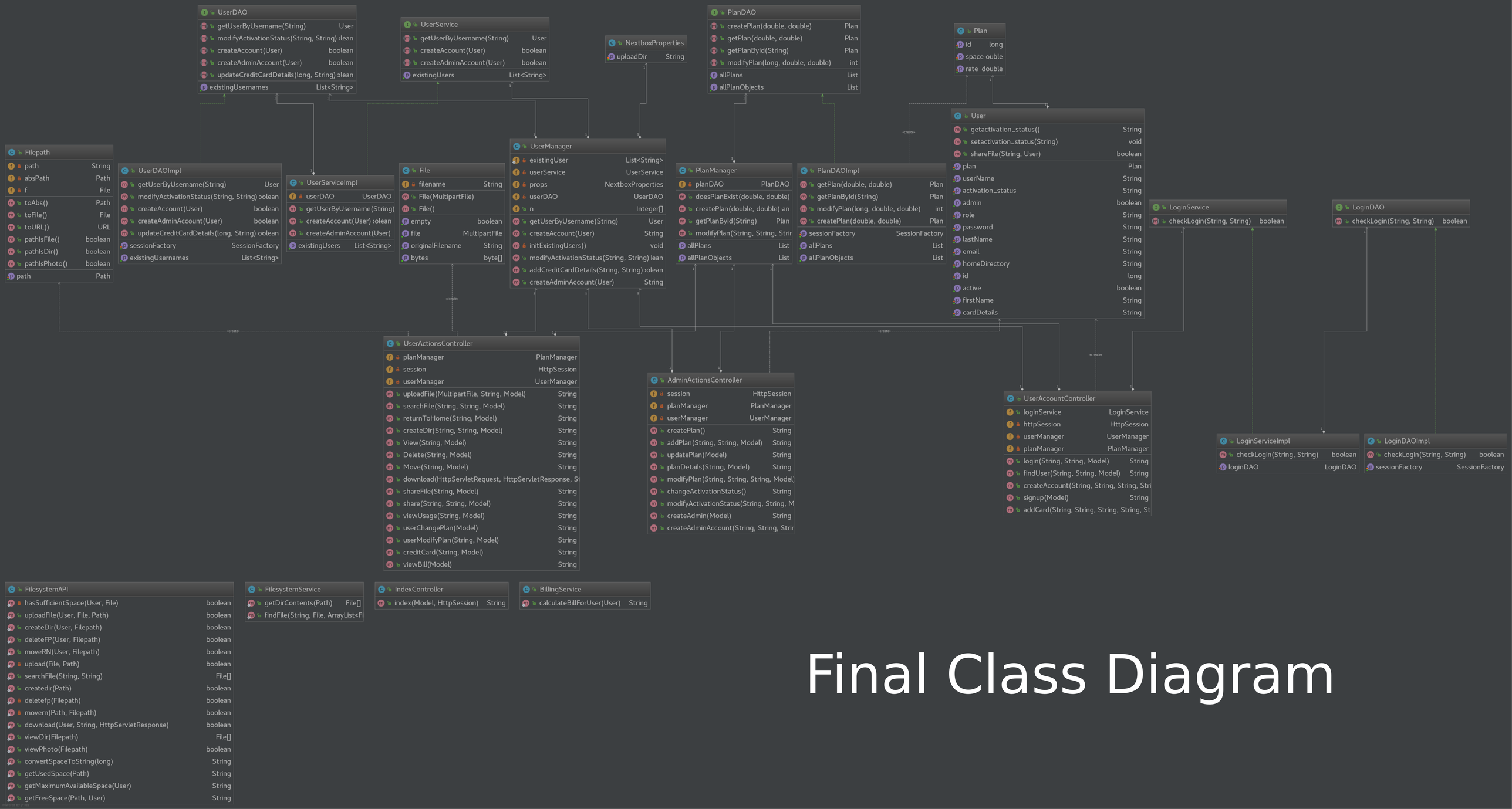
**Code Usage References**

1. Hibernate:
   http://www.roseindia.net/spring/spring4/login-form-using-spring-mvc-and-hibernate.shtml

2. Search file: (line 20-32 in FilesystemService.java)
   http://stackoverflow.com/questions/15624226/java-search-for-files-in-a-directory
3. Download file (lines 165-182 in FilesystemAPI.java):
   http://howtodoinjava.com/spring/spring-mvc/spring-mvc-download-file-controller-example/
4. Exception handling for delete/move/rename files and directories (see corresponding methods in FilesystemAPI.java):
   https://docs.oracle.com/javase/tutorial/essential/io/delete.html
5. Checking MIMEtype for photo (lines 54,56 in Filepath.java):
   http://stackoverflow.com/questions/9643228/test-if-file-is-an-image
6. Displaying photo in JFrame (lines 193-207 in FilesystemAPI.java):
   http://stackoverflow.com/questions/14353302/displaying-image-in-java

# Original Class Diagram




UserDAO
getUserByUsername(String) User
modifyActivationStatus(String, String) boolean
createAccount(User) boolean
createAdminAccount(User) boolean
updateCreditCardDetails(long, String) boolean
existingUsernames List<String>
UserService
getUserByUsername(String) User
createAccount(User) boolean
createAdminAccount(User) boolean
existingUsers List<String>
NextboxProperties
uploadDir String
PlanDAO
createPlan(double, double) Plan
getPlan(double, double) Plan
getPlanById(String) Plan
modifyPlan(long, double, double) int
allPlans List
allPlanObjects List
Plan
id long
space ouble
rate double
Filepath
path String
absPath Path
f File
toAbs() Path
toFile() File
toURL() URL
pathIsFile() boolean
pathIsDir() boolean
pathIsPhoto() boolean
path Path
UserDAOImpl
getUserByUsername(String) User
modifyActivationStatus(String, String) boolean
createAccount(User) boolean
createAdminAccount(User) boolean
updateCreditCardDetails(long, String) oolean
sessionFactory SessionFactory
existingUsernames List<String>
UserServiceImpl
userDAO UserDAO
getUserByUsername(String)
createAccount(User) oolean
createAdminAccount(User)
existingUsers List<String>
File
filename String
File(MultipartFile)
File()
empty boolean
file MultipartFile
originalFilename String
bytes byte[]
UserManager
existingUser List<String>
userService UserService
props NextboxProperties
userDAO UserDAO
n Integer[]
getUserByUsername(String) User
createAccount(User) String
initExistingUsers() void
modifyActivationStatus(String, String) lean
addCreditCardDetails(String, String) oolean
createAdminAccount(User) String
PlanManager
planDAO PlanDAO
doesPlanExist(double, double)
createPlan(double, double) an
getPlanById(String) Plan
modifyPlan(String, String, Strir
allPlans List
allPlanObjects List
PlanDAOImpl
getPlan(double, double) Plan
getPlanById(String) Plan
modifyPlan(long, double, double) int
createPlan(double, double) Plan
sessionFactory SessionFactory
allPlans List
allPlanObjects List
User
getactivation_status() String
setactivation_status(String) void
shareFile(String, User) boolean
plan Plan
userName String
activation_status String
admin boolean
role String
password String
lastName String
email String
homeDirectory String
id long
active boolean
firstName String
cardDetails String
LoginService
checkLogin(String, String) boolean
LoginDAO
checkLogin(String, String) boolean
UserActionsController
planManager PlanManager
session HttpSession
userManager UserManager
uploadFile(MultipartFile, String, Model) String
searchFile(String, String, Model) String
returnToHome(String, Model) String
createDir(String, String, Model) String
View(String, Model) String
Delete(String, Model) String
Move(String, Model) String
download(HttpServletRequest, HttpServletResponse, St
shareFile(String, Model) String
share(String, String, Model) String
viewUsage(String, Model) String
userChangePlan(Model) String
userModifyPlan(String, Model) String
creditCard(String, Model) String
viewBill(Model) String
AdminActionsController
session HttpSession
planManager PlanManager
userManager UserManager
createPlan() String
addPlan(String, String, Model) String
updatePlan(Model) String
planDetails(String, Model) String
modifyPlan(String, String, String, Model) String
changeActivationStatus() String
modifyActivationStatus(String, String, M
createAdmin(Model) String
createAdminAccount(String, String, Strir
UserAccountController
loginService LoginService
httpSession HttpSession
userManager UserManager
planManager PlanManager
login(String, String, Model) String
findUser(String, String, Model) String
createAccount(String, String, String, Stri
signup(Model) String
addCard(String, String, String, String, St
LoginServiceImpl
checkLogin(String, String) boolean
loginDAO LoginDAO
LoginDAOImpl
checkLogin(String, String) boolean
sessionFactory SessionFactory
FilesystemAPI
hasSufficientSpace(User, File) boolean
uploadFile(User, File, Path) boolean
createDir(User, Filepath) boolean
deleteFP(User, Filepath) boolean
moveRN(User, Filepath) boolean
upload(File, Path) boolean
searchFile(String, String) File[]
createdir(Path) boolean
deletefp(Filepath) boolean
movern(Path, Filepath) boolean
download(User, String, HttpServletResponse) boolean
viewDir(Filepath) File[]
viewPhoto(Filepath) boolean
convertSpaceToString(long) String
getUsedSpace(Path) String
getMaximumAvailableSpace(User) String
getFreeSpace(Path, User) String
FilesystemService
getDirContents(Path) File[]
findFile(String, File, ArrayList<Fi
IndexController
index(Model, HttpSession) String
BillingService
calculateBillForUser(User) String
«create»
«create»
«create»
«create»
«create»
Final Class Diagram