Group 06 - Design Specification

3rd December 2012
Amy Rebecca James, James Slater, Dan Mcguckin, Samuel Mills, Felix Farquharson,
Ben Brooks, Christopher Krzysztof Ilkow, Aiman Arafat
Project Coordinator: Nigel Hardy
Version: 1.2
Status: Release
Computer Science Department, University of Wales, Aberystwyth

Contents

1 Introduction

1.1 Purpose of the Document

The main purpose of this document is to identify and describe the information in the design specification document produced by CS22120 Group 06. Each major section of the design will be individually described in detail and the document will explain the design theory behind each individual feature. It will also act as a tool for the group to look back on to keep track of current and future developments.

1.2 Scope

The design specification shows the specific features in relation to the Monster Mash game. This document explains the definitions of each component within the design and explains the details of each feature. It is important that this document is read by all members of the project group, especially the Design team.

1.3 Objectives

The main objective of this document is to reinforce the production of a design specification. It has to be complete, up to date and an accurate interpretation of the users requirements. Another objective is to provide an explanation of the design elements necessary for the implementation and development phases. These will generally include the outline structure, decomposition description, dependency description, interface description and a final detailed design.

2 Outline Structure

1 Homepage (Logged Out)
- 1.1 Login link
- 1.2 Homepage link

2 Homepage (Logged In)
- 2.1 Homepage
- 2.2 Fight (Logged In)
  - 2.1.1 Choose two Monsters
  - 2.2.2 Battle Screen

3 Shop (Logged In)
- 3.1 Buy
  - 3.1.1 Choose a monster and click buy.
- 3.2 Sell
  - 3.2.1 Choose a monster and click sell.

4 Farm (Logged In)
- 4.1 Breed - Choose a monster and click buy.

5 About Page (Logged In)

6 Logout (Logged In)
- 6.1 Homepage (Logged Out)

3 Decomposition Description

3.1 Applications in System

In our Monster Mash design, there will be one single application. This handles everything from serving JSPs that a user can see in their browser, to the database connections, inter-server connections and data processing.

4 Significant classes in each package

4.1 Significant classes in the Model package

4.1.1 User

The User class contains fields to hold the user's data for attributes such as Name, Password, Money, Friends, etc. Each user can have between 1 and 6 monsters which can be purchased and sold to the farm. You can also set the user's default battle monster from the changeBattleMonster method in this class. A user is created using the UserFactory class.

4.1.2 Monster

This class contains all of the monster's stats. For example, height, age, strength, sex, etc. A monster is created using the MonsterFactory class.

4.1.3 PersistManager

PersistManager is the class which can create, update or delete a user or monster. It is also used to search for users or monsters and to start or stop the main application. It is also this class which talks to the User database, Monster database and Shop database.

4.2 Significant classes in the Controller package

4.2.1 Login

Login contains methods that will allow a user to login or register themselves as a user.

4.2.2 Shop

Shop contains methods that let a user buy or sell monsters.

4.2.3 FriendsList

FriendsList contains methods for managing a user's friends. E.g. addFriend, removeFriend, getFriend, etc.

4.2.4 Worth

Calculates monster worth based on attributes and age. Also assesses monster age.

4.2.5 Breed

This breeds two monsters and creates an offspring.

4.2.6 makeAttribute Function

Creates new attributes for the new monster based on parents stats.

4.2.7 generalAssess

Assesses a monster

4.2.8 passToPM

Adds new monster to database.

4.2.9 determineWinner

Selects winning monster.

4.2.10 randomizeProbabilities

Adds some randomness to a battle between two monsters.

4.3 compareAttributesRetM1

Compares monsters attributes and returns a percentage of likelihood for monster one to win. (100-m1 is m2 likelihood)

4.3.1 BuyMonster

Purchases the monster, takes monster from shop and adds it to user's farm and updates user's vMoney.

4.3.2 SellMonster

Sells the monster, takes the monster from the user's Farm, adds it to Shop and updates the vMoney.

4.3.3 Logging Checker

Checks how long it has been since user made an action, after a given amount of time it will log the user off if they are not active.

4.3.4 MonsterListCycle

Ages a users primary monster whilst a user is logged on.

4.3.5 SimulateFight

SimulateFight has methods to calculate a winner of a fight between two monsters and calculates the prize amount of a fight, which is given to the winner.

4.4 Mapping from requirements to classes

Requirement	Classes Providing Requirement
FR1 (Server-based authentication)	Login, CreateMonsterAndUser, PersistManager, User and UserFactory
FR2 (Server friends list)	FriendsList and PersistManager
FR3 (Server monster list)	CreateMonsterAndUser, PersistManager, User and Monster
FR4 (Server monster mash management)	SimulateFight, FriendsList, PersistManager, User and Monster
FR5 (Server-server communication)	PersistManager
FR6 (Client options)	Login, Shop, FriendsList, Breed, SimulateFight and PersistManager
FR7 (Startup of software in browser)	Login and PersistManager
FR8 (Game display in browser)	FriendsList, Breed, SimulateFight and PersistManager
FR9 (Friend matching)	FriendsList, PersistManager
FR10 (Fight notifications)	SimulateFight and PersistManager
FR11 (Friends rich list)	FriendsList

5 Dependency Description

5.1 Outline

The following section of the document will explain how the parts of the system integrate together and how they are dependent on each other. All parts of the system will be heavily dependent on the database, as all user details will be stored in it, such as their login details, monsters, V-Money and a list of friends.

5.2 User

In order for the user to play the game and access its internal components, it is a requirement for the user to be logged in. Users will not be able to create a monster, add friends or sell/buy monsters until they are logged in. If this is the case, an error would be returned to the user and the user would be asked to register their username and password to the database, which would then allow access to the application itself.

5.3 Login/Register

Login and register methods will be entirely dependent on the database classes, where the task is to ensure that the details are entered and retrieved correctly allowing user authentication.

5.4 Shop

Buying a monster will not work when the user's V-Money is below the purchase price required. Breed class is dependent on two monsters, when less than two monsters are supplied, the class will return an error to the user and would not proceed further. SimulateFight class also needs two monsters as a minimal requirement for it to be executed, otherwise an error is produced to the user and the user can't proceed further.

5.5 Database

PersistManager is a class that will communicate with the database itself, from which it is going to retrieve the data. The Persist Manager is dependent on the User and Monster classes. The database is set up using Java Persistence API using the open source Hibernate Library. It runs using JavaDB or derby. This changed from HyperSQL due to connection problems. The database tables are created using persistable 'entity' classes through a persistence.xml file. The PersistManager class handles interactions between the database and the server. The database is reliant on the PersistManager and vice versa.

6 Interface Description

6.1 Model Package

The Model package contains the classes that deal with data representation. It will also contain all connections to the database, and methods of using all of the data types the group may need to use in the controller class at a later time.

6.1.1 PersistManager Class

Public class that handles transferring the data currently active in the application to persistent data in a database and it also handles retrieving the data from the database. There are 6 public functions, that are listed below. It uses the Monster class and it works by translating a Monster instance into a set of database friendly fields. It is used by many of the classes in the Controller package, so that they can retain information and process it at a later time.

init Function A public function that when called with initialize the class and instantiates the Persistence class to deal with the database. It takes no arguments and returns no data.

create Function A public function that will create all of the fields in the database for a Monster instance. Takes a Monster instance as an argument.

update Function A public function to update an entry for a Monster or a user in the database, takes a Monster instance or a User instance and returns no data.

search (monster) Function A public function for searching the database to find a list of a user's Monster instances. Takes a User instance and returns an array of Monster instances.

search (user) Function A public function for finding a user in the database. This function takes nothing and returns a User instance and returns nothing.

remove(monster) Function A public function to remove the representations of the Monster instances from the database. Takes a Monster instance and returns nothing.

remove (user) Function A public function to remove the representations of the User instances from the database. Takes a User instance and returns nothing.

isReal Function A public function to check if the user is stored in the database. Takes two Strings, a username and a password. Returns a boolean.

shutdown Function A public function for closing a connection to the database. This function requires nothing and returns nothing.

DropTable Function This function destroys the table i.e. clears it, method is there but is deprecated.

searchGraveYard This method collects dead monsters, this method however is not integrated in the final program.

6.1.2 User Class

A public class to represent a user, it does not inherit from any classes. There are no public variables because everything is dealt with by the PersistManager class.

ChangeBattleMonster Function A public function to change which a user's Monster is set for use in a battle situation. Takes a Monster instance to set as the users new battle Monster.

6.1.3 UserFactory Class

A public class designed to create new User instances when called from somewhere. It has one public function to create the User.

6.1.4 Requests Class

A public class that contains all request attributes to be persistent into the database for request handling.

make it Function This public function will create a User instance, that will later be stored by the PersistManager. Likely called by the Login Class from the Controller package. It requires the following arguments:

- name: String - The name of the new user.
- password: String - The password of the new user.
- vMoney: int - The amount of money the new user has.
- type: Type - The user's type, ie. regular user or superuser.

This function will return a User instance.

6.1.5 Monster Class

The Monster class only has public getters and setters, it represents each monster and contains everything that is necessary to represent a monster. The PersistManager uses the Monster class when it converts each monster into a database friendly form.

6.1.6 MonsterFactory Class

A public class to create Monster instances that will later be stored by the PersistManager as part of a User instance. It has one public function.

make it Function This public function will create a Monster instance. It may be called from the UserFactory class when a new user is created to make their first Monster. It might also be called from anywhere else that needs to create a new Monster instance, such as the Breed (servlet) class. It needs to be called with the following arguments:

- name: String - The name of the new monster.
- height: int - The new monster's height.
- age: int - The new monster's age (in days I expect).
- strength: int - The new monster's strength.
- health: int - The new monster's health.
- sex: Boolean - The new monster's sex as an isMale boolean.
- aggression: int - The aggression of the new monster.
- dob: int - The new monster's Date of Birth.
- battleMonster: Boolean - States if the new monster is the monster used for battle.

This function returns a Monster instance.

6.2 Controller Package

The Controller package contains all the classes that deal with the logic behind the back end of the application. It contains all the servlets and uses the Model to represent data for it.

6.2.1 CreateMonsterAndUser Class

A public class used by the Login (servlet) Class to handle creating a user and the user's first monster. It will rely on the MonsterFactory and UserFactory in the Model. There are two public functions.

CreateMonsterAndUser (Constructor) Function This function will set up the class, by creating a User and with it, its first Monster. It takes no arguments and returns no data.

CreateMonster Function A public function to create a Monster instance, it takes no arguments and returns a Monster instance. Presumably used by the constructor.

6.2.2 Login (Servlet) Class

The public class login handles the authentication of the users. It is a servlet class and is used over http. There are a number of public functions to deal with users.

checkPass Function A public function for checking a password is correct and valid. Takes a String and returns a Boolean value that reflects whether the password is correct.

checkUserName Function A public function for checking if a username is correct. It takes a string and returns a boolean value that reflects if the username is correct.

register Function A public function to register a new user into the system. It needs to be passed by two variables; username and password both as strings. It returns nothing.

Get Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

Post Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

6.2.3 Shop (Servlet) Class

This servlet class handles all of the information required to deal with the shop, it allows users to buy and sell Monsters and to find out prices for other listed items.

calcWorth Function A public function that calculates the value of a monster so that it can be displayed in the Shop. Uses a local variable of Monsters in the Shop to find the monsters and returns an integer that represents the price.

buy Function A public function that allows the user to buy the selected monster from the shop if they can afford it. It takes no arguments and returns nothing.

sell Function Public function that allows the user to sell the selected monster from the user's monster list for the calculated worth of the monster. It takes no arguments and returns nothing.

viewVMoney Function A public function for finding out how much money the user has, because this is needed for display in the shop. There are no arguments and nothing is returned.

addMonster Function A public function that adds a monster to the shop, it takes no arguments and returns nothing.

Get Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

Post Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

6.2.4 FriendsList (Servlet) Class

A public class to deal with the Friends List and all the information required, it also allows users to add, view, and remove from their friends list.

addFriend Function Public function that allows the user to send a friend request. If request is accepted then the friend ID will be added to friends array.

removeFriend Function Public function that asks the user to confirm the removal of the specified friend ID, if confirmed, the specified friend ID will be removed from the friend array.

requestFight Function Public function that allows the user to pick a friend from the friend list and to start a battle against the friend's preset battle monster.

getFriend Function A public function that allows the user to pull out a friends profile. Takes no arguments and returns nothing.

findUserMonsters Function A public function to find all of another user's monsters. This function takes no arguments and returns nothing.

updateList Function A public function to update a user's friend list from the server. Takes no arguments and returns nothing.

Get Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

Post Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

Get Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

Post Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

6.2.5 Breed (Servlet) Class

A public (servlet) class to handle how you might breed monsters in the game. It allows you to perform all relevant checks to the data. There are a few public functions made available. There are no public variables. The only important things are the two parent Monsters, that are passed in as parameters to the createChild function.

calcCost Function This is a public function to work out what it would cost to buy a particular monster, takes a Monster instance and returns an int that is the price to buy the given monster.

createChild Function A public function that allows the user to breed a monster, takes two Monster instances and returns a Monster instance that is a child of the two given monsters.

Get Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

Post Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

6.2.6 SimulateFight (Servlet) Class

A public (servlet) class that represents two monsters battling in the application. There is only one public function available and no public variables.

calcWinner Function A public function to work out which Monster is going to perform better in a battle. Takes two Monster instances, returns the winning Monster instance.

isPrized Function A public function to work out if a match is prized, takes nothing as an argument, but returns a boolean.

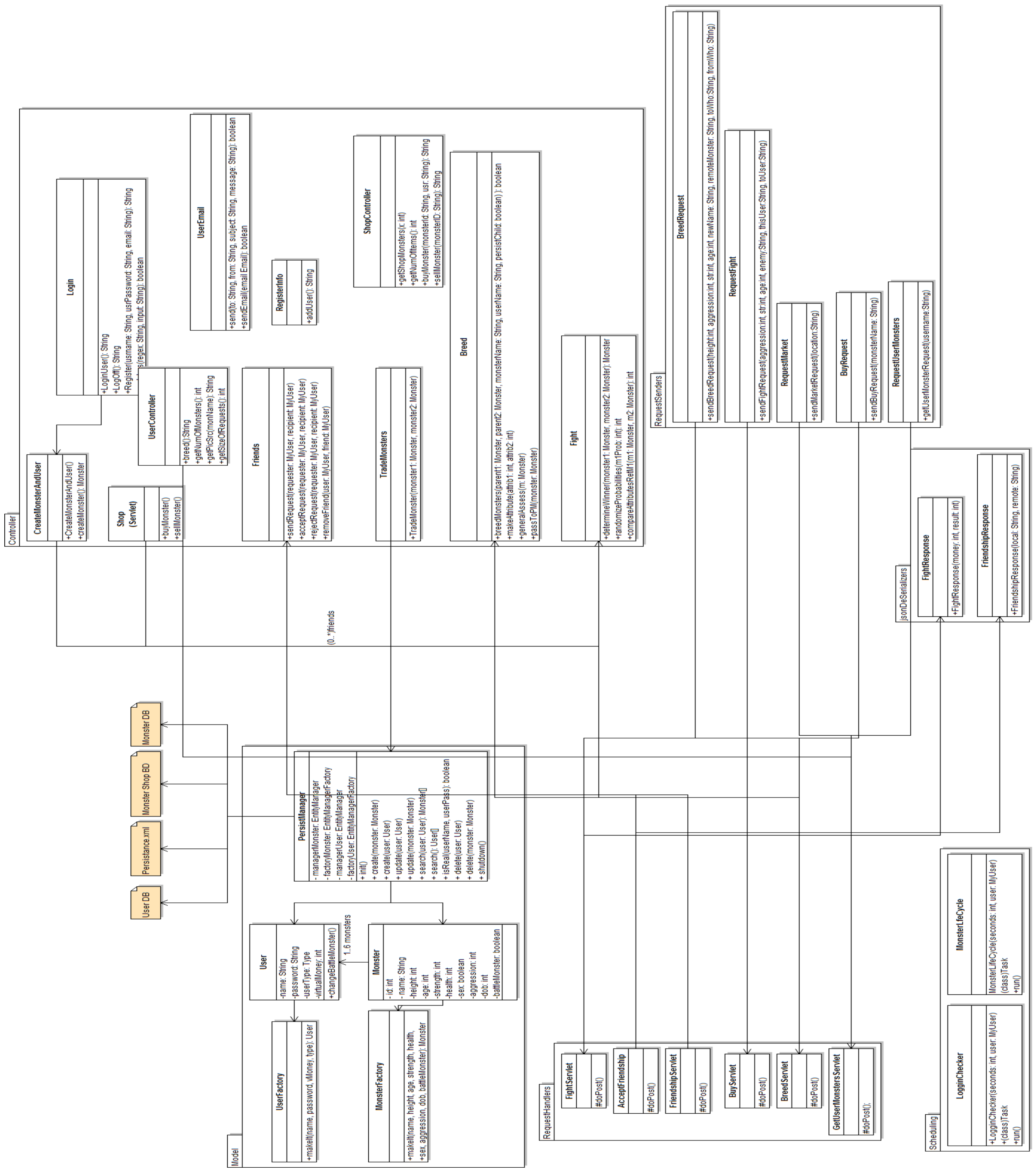
prizeAmount Function A public function to return the amount that is given as a prize, takes nothing as an argument and returns an int that represents the prize money.

Get Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

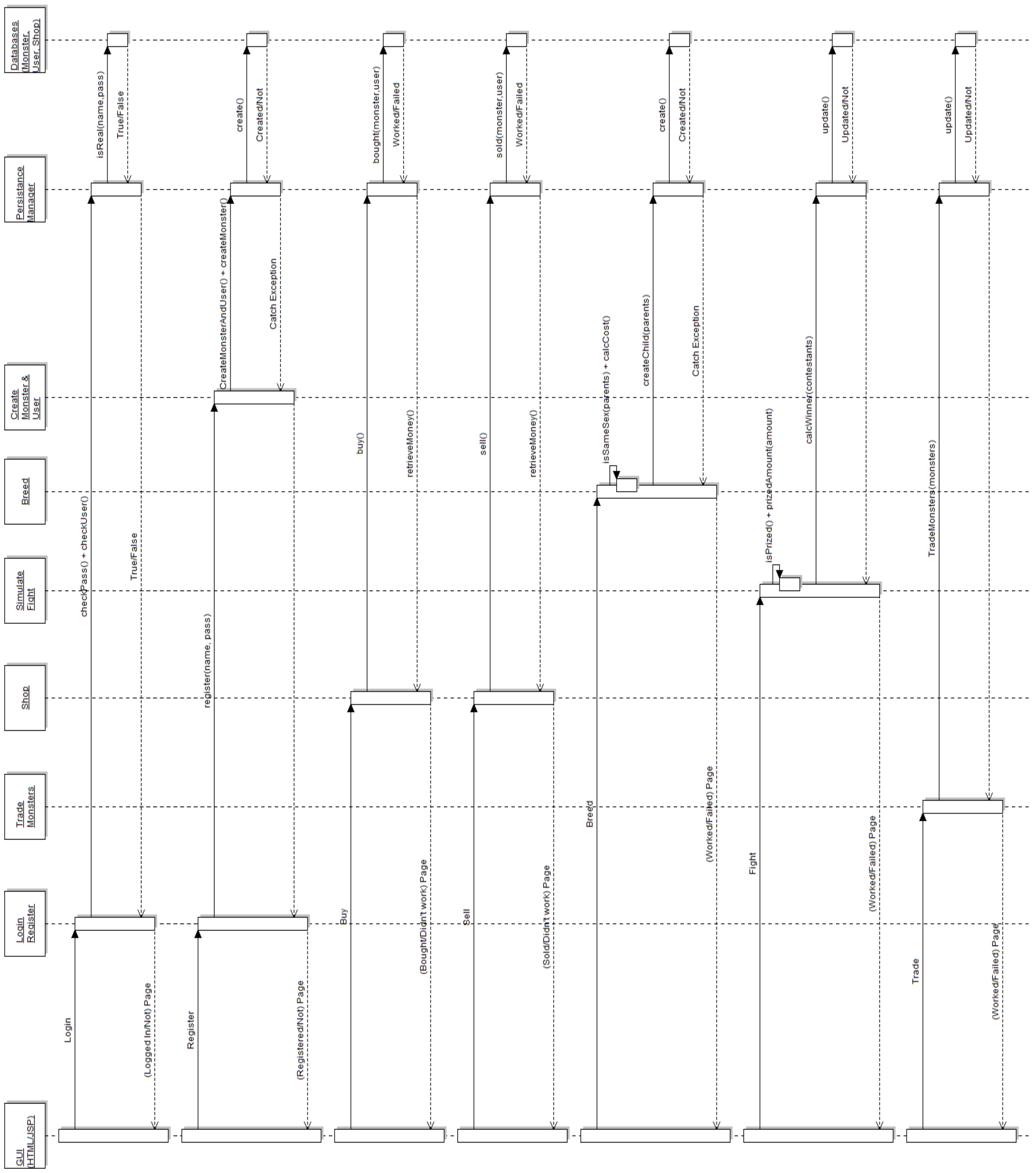
Post Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

7 Detailed Design

7.1 Monster Mash Class Diagram


Controller
CreateMonsterAndUser
+CreateMonsterAndUser()
+createMonster(): Monster
Login
+LoginUser(): String
+LogOff(): String
+Register(usrname: String, usrPassword: String, email: String): String
+s(regex: String, input: String): boolean
UserController
+breed():String
+getNumOfMonsters(): int
+getPicSrc(monName): String
+getSizeOfRequests(): int
Shop (Servlet)
+buyMonster()
+sellMonster()
UserEmail
+send(to: String, from: String, subject: String, message: String): boolean
+sendEmail(email Email): boolean
RegisterInfo
+addUser(): String
Friends
+sendRequest(requester: MyUser, recipient: MyUser)
+acceptRequest(requester: MyUser, recipient: MyUser)
+rejectRequest(requester: MyUser, recipient: MyUser)
+removeFriend(user: MyUser, friend: MyUser)
ShopController
+getShopMonsters(x: int)
+getNumOfItems(): int
+buyMonster(monsterId: String, usr: String): String
+sellMonster(monsterID: String): String
TradeMonsters
+TradeMonster(monster1: Monster, monster2: Monster)
Breed
+breedMonsters(parent1: Monster, parent2: Monster, monsterName: String, userName: String, persistChild: boolean): boolean
+makeAttribute(attrib1: int, attrib2: int)
+generalAssess(m: Monster)
+passToPM(monster: Monster)
Fight
+determineWinner(monster1: Monster, monster2: Monster): Monster
+randomizeProbabilities(m1Prob: int): int
+compareAttributesRetM1(m1: Monster, m2: Monster): int
RequestSenders
BreedRequest
+sendBreedRequest(height:int, aggression:int, str:int, age:int, newName: String, remoteMonster: String, toWho:String, fromWho: String)
RequestFight
+sendFightRequest(aggression:int, str:int, age:int, enemy:String, thisUser:String, toUser:String)
RequestMarket
+sendMarketRequest(location:String)
BuyRequest
+sendBuyRequest(monsterName: String)
RequestUserMonsters
+getUserMonsterRequest(username:String)
jsonDeSerializers
FightResponse
+FightResponse(money: int, result: int)
FriendshipResponse
+FriendshipResponse(local: String, remote: String)
(0..*)friends
Monster DB
Monster Shop BD
Persistance.xml
User DB
Model
PersistManager
- managerMonster: EntityManager
- factoryMonster: EntityManagerFactory
- managerUser: EntityManager
- factoryUser: EntityManagerFactory
+ init()
+ create(monster: Monster)
+ create(user: User)
+ update(user: User)
+ update(monster: Monster)
+ search(user: User): Monster[]
+ search(): User[]
+ isReal(userName, userPass): boolean
+ delete(user: User)
+ delete(monster: Monster)
+ shutdown()
User
-name: String
-password: String
-userType: Type
-virtualMoney: int
+changeBattleMonster()
1..6 monsters
Monster
- id: int
-name: String
-height: int
-age: int
-strength: int
-health: int
-sex: boolean
-aggression: int
-dob: int
-battleMonster: boolean
UserFactory
+makeIt(name, password, vMoney, type): User
MonsterFactory
+makeIt(name, height, age, strength, health,
+sex, aggression, dob, battleMonster): Monster
RequestHandlers
FightServlet
#doPost()
AcceptFriendship
#doPost()
FriendshipServlet
#doPost()
BuyServlet
#doPost()
BreedServlet
#doPost()
GetUserMonstersServlet
#doPost();
Scheduling
LogginChecker
+LogginChecker(seconds: int, user: MyUser)
+(class)Task
+run()
MonsterLifeCycle
MonsterLifeCycle(seconds: int, user: MyUser)
(class)Task
+run()

7.2 Monster Mash Sequence Diagram



8 References

1] Software Engineering Group Project Plan Design Specification Standards. C. J. Price, N. W. Hardy. SE.QA.05a. 1.6
2] Project Plan - CS221 Project Group 6. Version 1.0
3] Test specification - CS221 Project Group 6. Version 1.0
4] Software Engineering Group Projects General Documentation Standards. C. J. Price, N. W. Hardy. SE.QA.03. Version 1.5
5] Software Engineering Group Projects Monster Mash Game Requirements Specification. B. P . Tiddeman. SE.CS.RS. Version 1.2

9 Document History

Table 1: Document History

Version	CCF No.	Date	Changes made to document	Changed by:
1.0	N/A	06/12/2012	Created Design Specification	sam39
1.1	N/A	12/02/2013	Made required changes	sam39
1.2	N/A	13/02/2013	Few sentence changes on some functions	sam39