Monster Mash – Team Awesome-er Design Specification

Author:	Joshua Bird, Phil Wilkinson, Tom Hull, Dave Haenze, Chris Morgan, Kamarus Alimin, Szymon Stec, Lewis Waldron
Config Ref:	SE_02_DS_04
Date:	07-12-2012
Version:	1.3
Status:	Final

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB

CONTENTS

1. INTRODUCTION

1.1 Purpose of this Document

The purpose of this document is to outline the detail and description on various parts of the design for the group project. This document should be read in a manner that takes into consideration the details of the group project assignment, and the group project design specification standards [1].

1.2 Scope

This document is intended to give a breakdown of the group project design via various methods and descriptions, and to also give diagrams showing the interactions between the client and the server.

This document should be especially read by all members of the coding team and by the rest of the group, so they have an understanding of what is happening inside the program.

1.3 Objectives

The objective of this document is to:

- Give a description of the group project components from a high-level point of view
- Illustrate the flow of data between the client and server
- Depict a basic overview of the classes and how they are linked through a UML class diagram
- Give an understanding of the relationships and dependencies between modules in the dependency description
- List and give details of the interface classes, so that it can be used by other coders.

2. ARCHITECHTURAL DESCRIPTION

The monster program consists of three components:

- The DATABASE component
- The CLIENT component
- The SERVLET component

2.1 The DATABASE component

We will be using a Java Database which is a database management system (DBMS) that we use to store and retrieve information to/from a database. It is used to organize the data according to the subject, and it is easy to track and verify the data. It can store information about how different subjects are related, so that it makes it easy to bring related data together. We will create tables such as the user’s profile and their monster’s information, login database and the friend’s database. The database will be implemented in a way that will meet the requirements of (FR1), (FR2), (FR3), (FR6) and (FR11) of the requirement specification documents.

2.2 The CLIENT component

The client is where the user can request a webpage from the server. Our program web-pages will consist of:

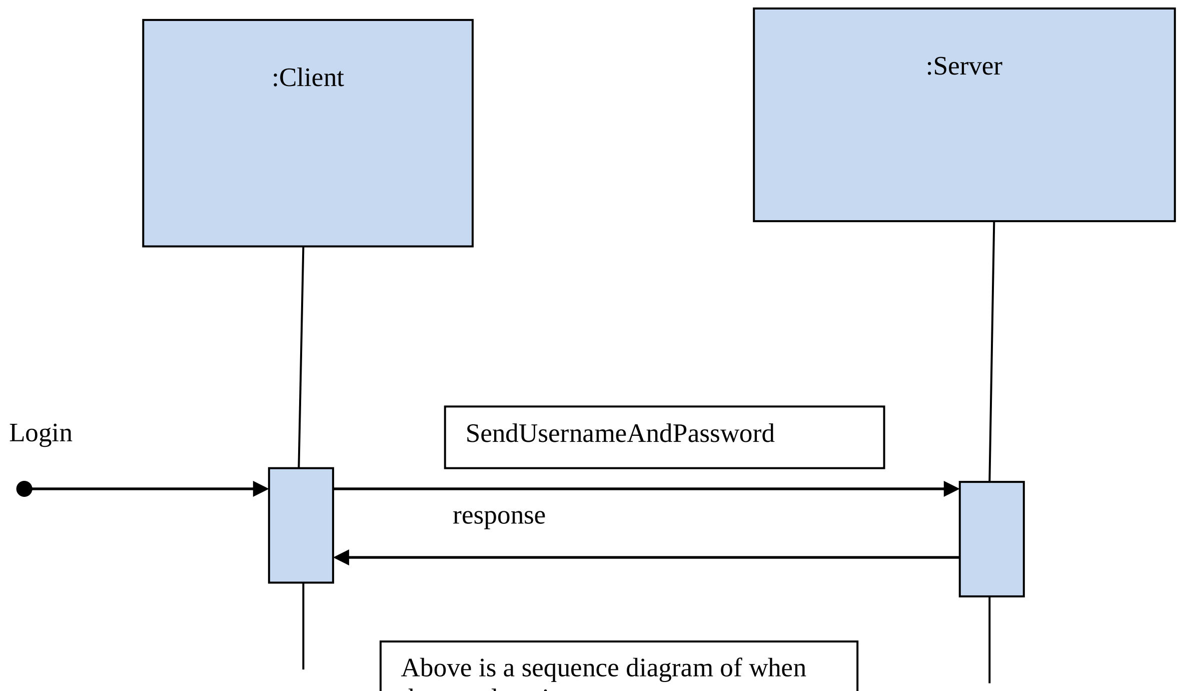
- Login page - this is where the users can login existing account or register a new account.
- Home page - the users will be able to choose various links to navigate.
- Marketplace page - this page allow users to view the monster that are to be bought or rent.
- My monster page - this page allow users to breed the monster they own.
- Monster Fights page - the user will be able to challenge other monsters through this page.
- Friends page - this page allow users to interact with other users.

The user will be able to communicate the server through POST and GET method. The POST method is used to catch the user input field which are in the forms. The GET request is used to tell a HTML file which webpage to display. The client will be implemented in a way that will meet the requirements of (FR5), (FR6), (FR7), (FR8), (FR10), (FR11), (PR1) and (PR2) of the requirement specification documents.

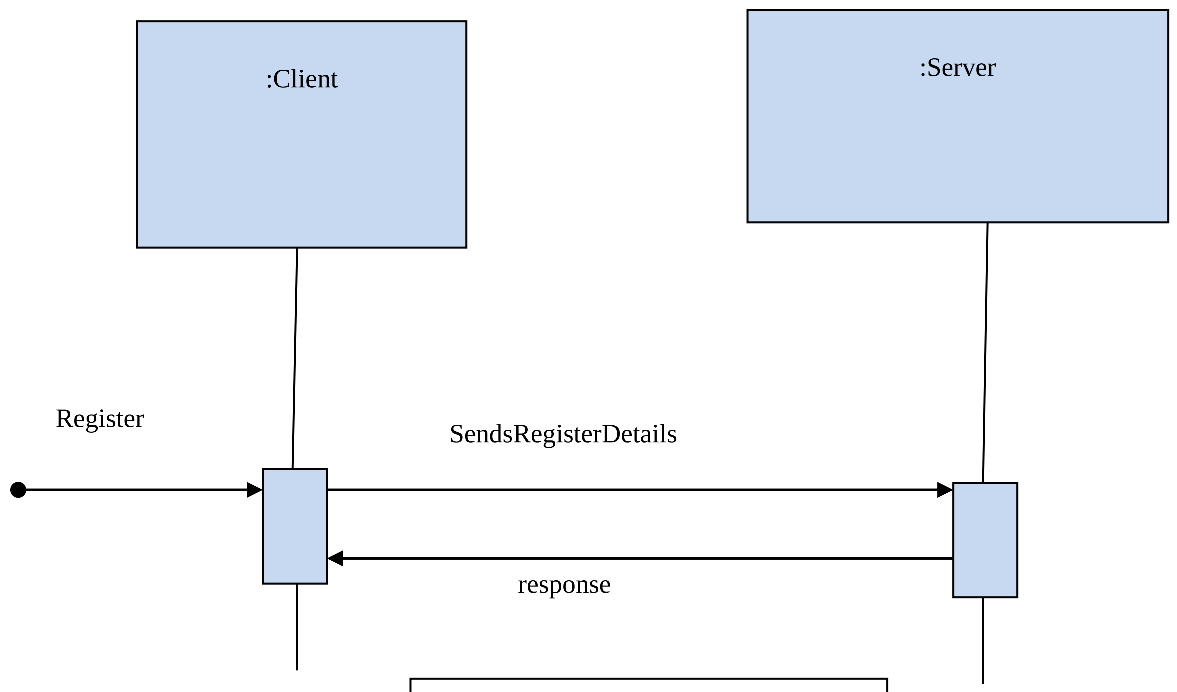
2.3 The SERVLET component

On the servlet side, we are using Glassfish servlets as a development tool which provides storage and communication storage. The servlet is used to produce a response to the request made by the user in the html page and send it back to the requesting browser through the GET and POST requests. The servlet first looks for incoming request data: if it finds none, it presents a blank form. If the servlet finds partial request data, it extracts the partial data, puts it back into the form, and marks the other fields as missing. If the servlet finds the full complement of required data, it processes the request and displays the results. The servlets will be implemented in a way that will meet the requirement of (FR1), (FR2), (FR3), (FR4), (FR5), (PR1), (PR2), (DC1) and (DC2) of the requirement specification documents.

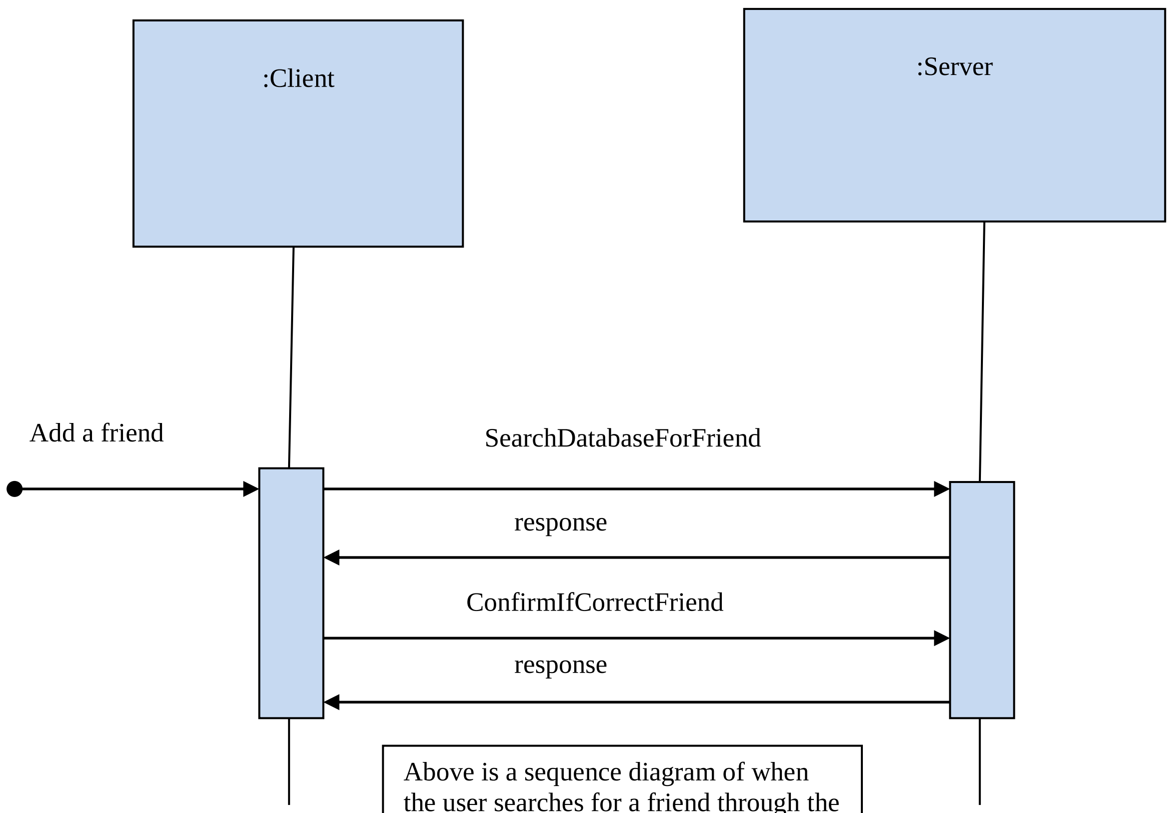
3. SEQUENCE DIAGRAMS



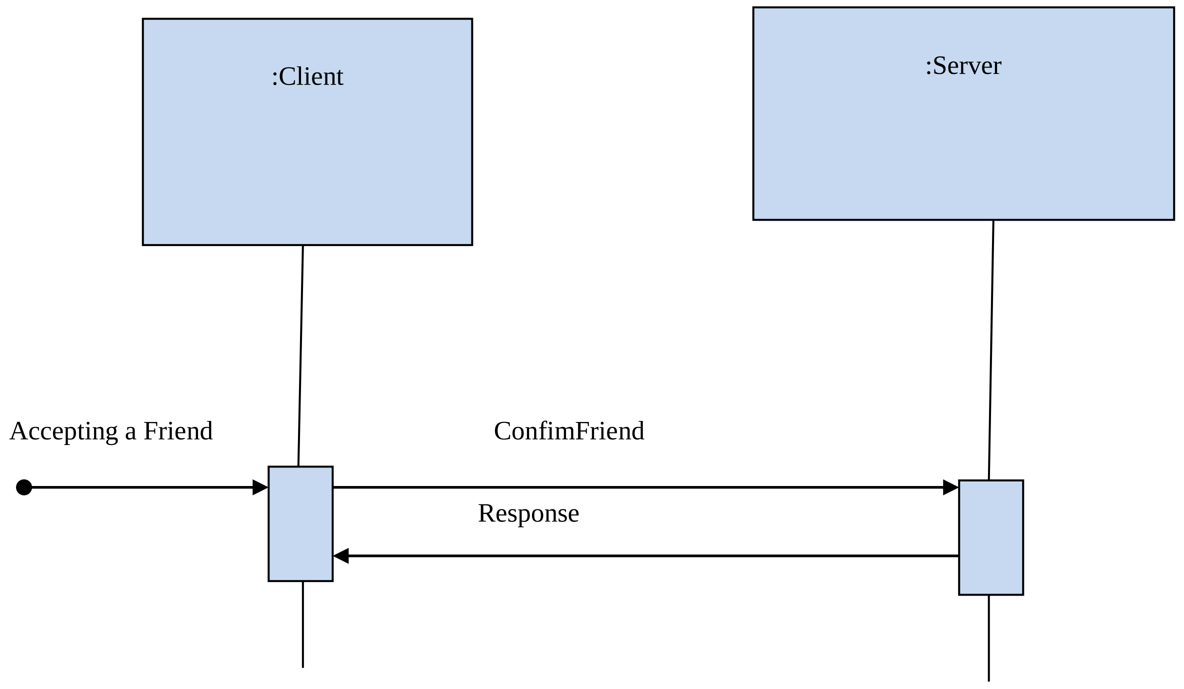
Above is a sequence diagram of when the user logs in.



Above is a sequence diagram of when the user register to play the game.

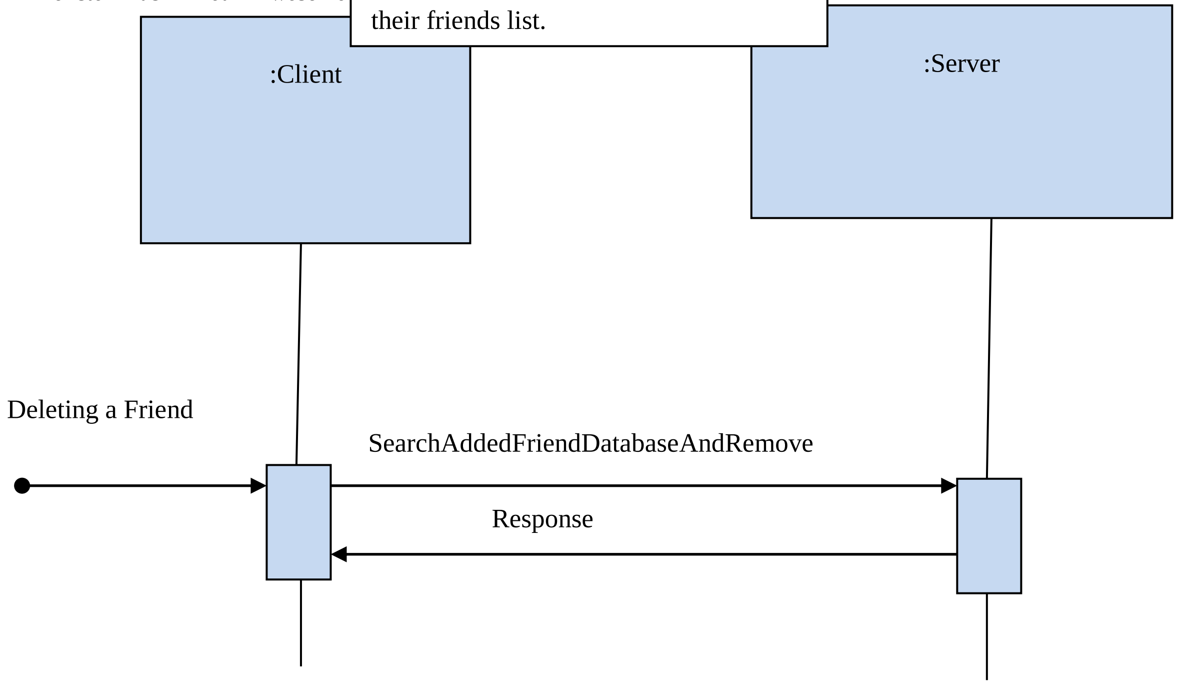


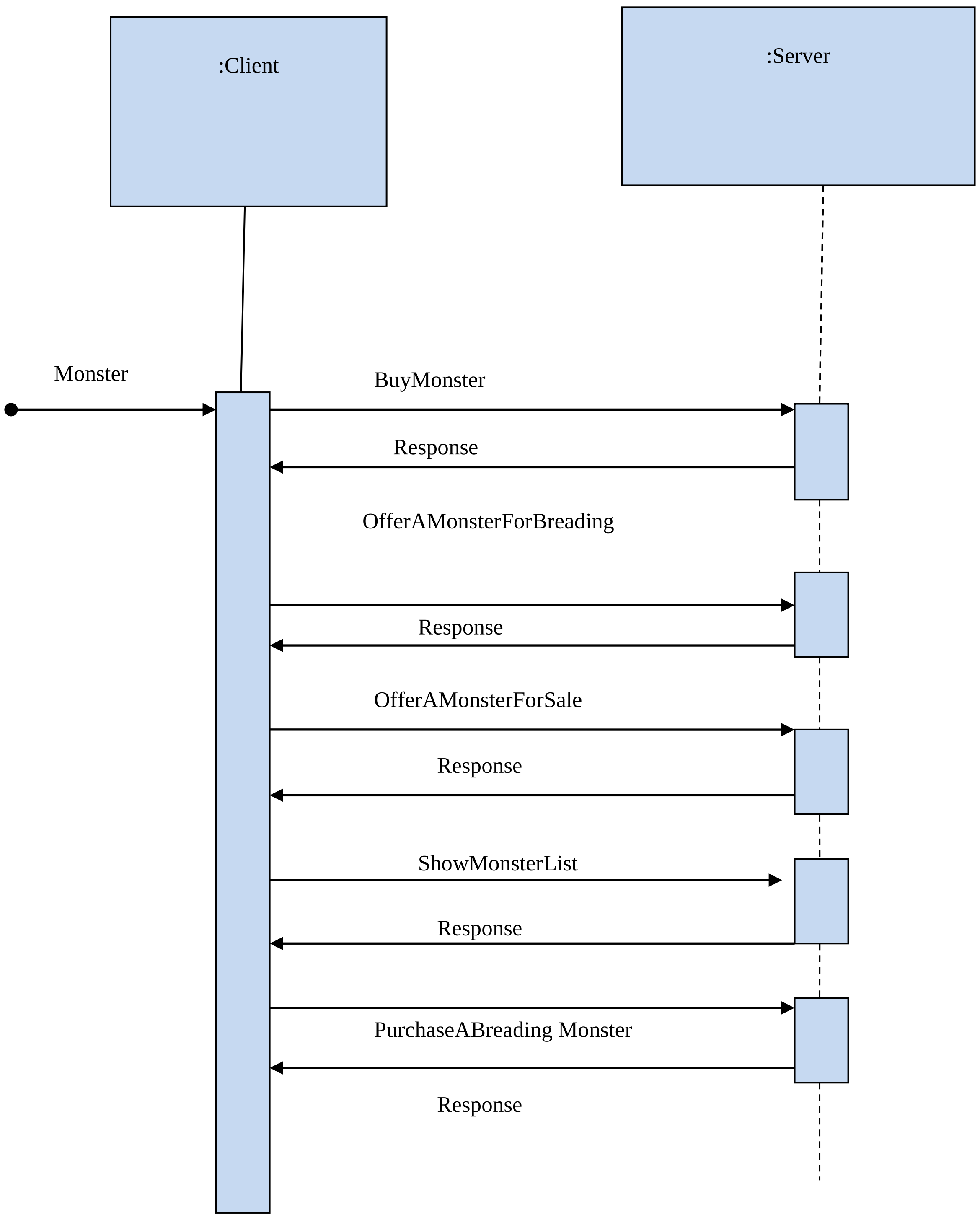
Above is a sequence diagram of when the user searches for a friend through the database and then adds that friend.



Above is a sequence diagram of when the user adding a friend.

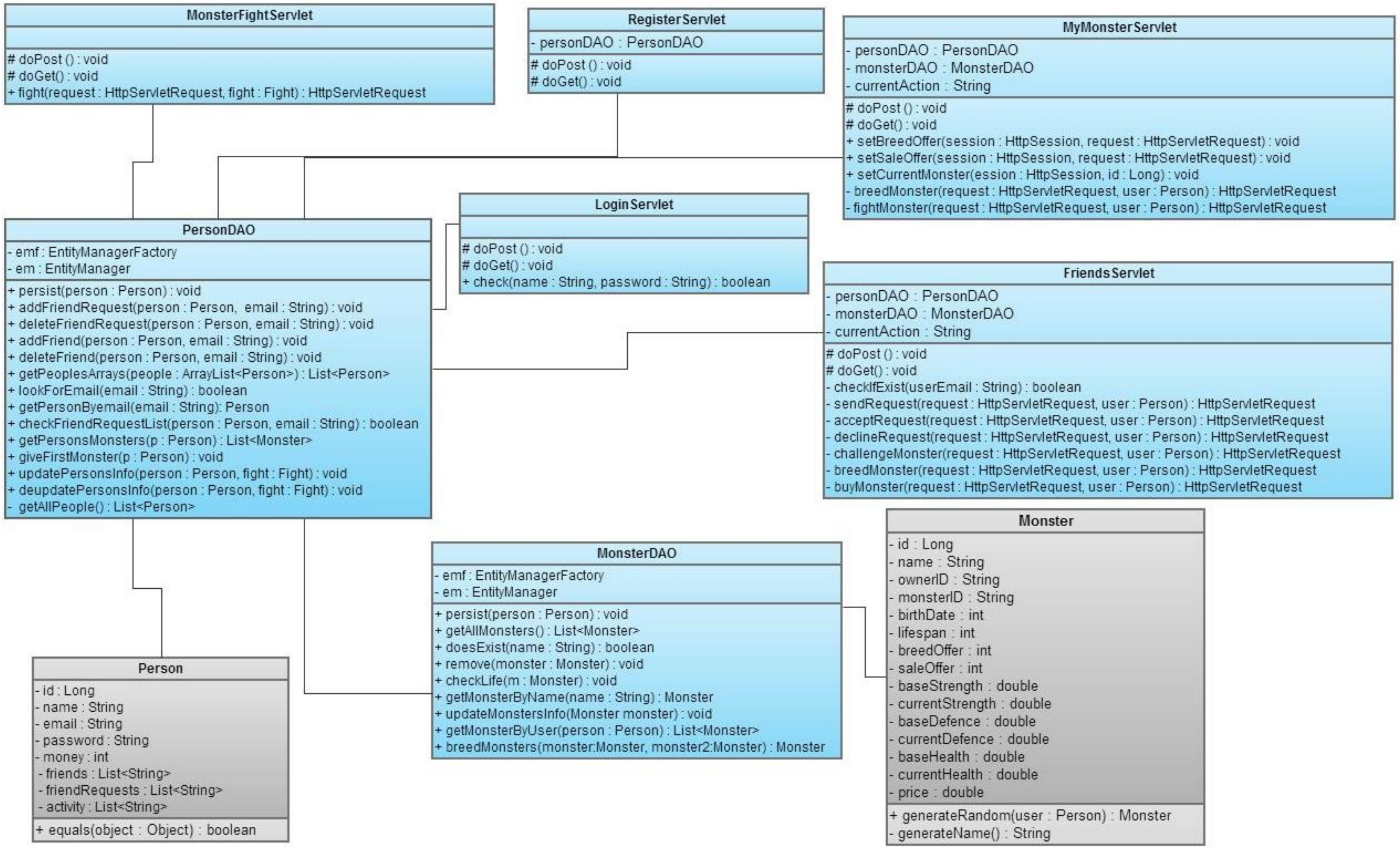
Above is a sequence diagram of when the user wants to delete a friend from their friends list.





Above is a sequence diagram of what options are available to the user to do in the game. The user has the option of buying a monster. Offering a monster for breading, or offer a monster for sale. The user can also see their monsters. They can also purchase a monster for breading.

4. CLASS DIAGRAM


MonsterFightServlet
# doPost () : void
# doGet() : void
+ fight(request : HttpServletRequest, fight : Fight) : HttpServletRequest
RegisterServlet
- personDAO : PersonDAO
# doPost () : void
# doGet() : void
MyMonsterServlet
- personDAO : PersonDAO
- monsterDAO : MonsterDAO
- currentAction : String
# doPost () : void
# doGet() : void
+ setBreedOffer(session : HttpSession, request : HttpServletRequest) : void
+ setSaleOffer(session : HttpSession, request : HttpServletRequest) : void
+ setCurrentMonster(ession : HttpSession, id : Long) : void
- breedMonster(request : HttpServletRequest, user : Person) : HttpServletRequest
- fightMonster(request : HttpServletRequest, user : Person) : HttpServletRequest
LoginServlet
# doPost () : void
# doGet() : void
+ check(name : String, password : String) : boolean
PersonDAO
- emf : EntityManagerFactory
- em : EntityManager
+ persist(person : Person) : void
+ addFriendRequest(person : Person, email : String) : void
+ deleteFriendRequest(person : Person, email : String) : void
+ addFriend(person : Person, email : String) : void
+ deleteFriend(person : Person, email : String) : void
+ getPeoplesArrays(people : ArrayList<Person>) : List<Person>
+ lookForEmail(email : String) : boolean
+ getPersonByemail(email : String): Person
+ checkFriendRequestList(person : Person, email : String) : boolean
+ getPersonsMonsters(p : Person) : List<Monster>
+ giveFirstMonster(p : Person) : void
+ updatePersonsInfo(person : Person, fight : Fight) : void
+ deupdatePersonsInfo(person : Person, fight : Fight) : void
- getAllPeople() : List<Person>
FriendsServlet
- personDAO : PersonDAO
- monsterDAO : MonsterDAO
- currentAction : String
# doPost () : void
# doGet() : void
- checkIfExist(userEmail : String) : boolean
- sendRequest(request : HttpServletRequest, user : Person) : HttpServletRequest
- acceptRequest(request : HttpServletRequest, user : Person) : HttpServletRequest
- declineRequest(request : HttpServletRequest, user : Person) : HttpServletRequest
- challengeMonster(request : HttpServletRequest, user : Person) : HttpServletRequest
- breedMonster(request : HttpServletRequest, user : Person) : HttpServletRequest
- buyMonster(request : HttpServletRequest, user : Person) : HttpServletRequest
MonsterDAO
- emf : EntityManagerFactory
- em : EntityManager
+ persist(person : Person) : void
+ getAllMonsters() : List<Monster>
+ doesExist(name : String) : boolean
+ remove(monster : Monster) : void
+ checkLife(m : Monster) : void
+ getMonsterByName(name : String) : Monster
+ updateMonstersInfo(Monster monster) : void
+ getMonsterByUser(person : Person) : List<Monster>
+ breedMonsters(monster:Monster, monster2:Monster) : Monster
Monster
- id : Long
- name : String
- ownerID : String
- monsterID : String
- birthDate : int
- lifespan : int
- breedOffer : int
- saleOffer : int
- baseStrength : double
- currentStrength : double
- baseDefence : double
- currentDefence : double
- baseHealth : double
- currentHealth : double
- price : double
+ generateRandom(user : Person) : Monster
- generateName() : String
Person
- id : Long
- name : String
- email : String
- password : String
- money : int
- friends : List<String>
- friendRequests : List<String>
- activity : List<String>
+ equals(object : Object) : boolean

5. DEPENDENCY DESCRIPTION

5.1 Introduction

Each of the web pages depends on its servlet in order to function. Each of the servlets then depends on the DAO session beans in order to access the database. These access classes then rely on the database server for them to function.

5.2 Class descriptions

5.2.1 Servlet Classes

Each servlet class serves the function of a single webpage.

5.2.2 Login Servlet

This takes the log in credentials from the user with a post and tries to find them in the database using the check method. If the credentials are correct, the user will be forwarded to a home page, otherwise the page is reloaded with an error message.

5.2.3 Register Servlet

This takes the users input with a post, checks to database to insure the email address is unique. If the email address is unique then a new person is added to the database and the user is forwarded to the log in page. Otherwise the page is reloaded with an error message.

5.2.4 Friends Servlet

The friends servlet will allow the user to display a list of their friends, add new friends and confirm friend requests. All of this will be done by accessing methods in the PersonDAO session bean.

5.2.5 My Monsters Servlet

This will allow the user to get a list of his monsters and their stats. There is a method to breed monsters which will take two monsters and produce a new monster based on their genetics.

5.2.6 Market Place Servlet

This gives access to monsters that have been flagged for sale and allows the user to flag their on monsters. Sell Monster will flag a monster for sale, Buy Monster will transfer a monster from the user selling it to the user buying it, Buy Monster To Breed will give the user access to another user's monster's genetics.

5.2.7 Fight Servlet

The methods in this class facilitate actual fights. Send Fight Request will add the users name to a list in his friend’s record flagging that he/she wants to have a fight. Accept fight will take one of these requests and initiate the fight using the fight method which takes the two monsters fighting, decides the outcome and updates the users/monsters profiles to reflect the outcome. Get monster list will return the list of a user’s monsters and in order to choose one to fight.

5.2.8 Session Beans

Session Beans allow access to the database using entity managers. Each of the servlets has an injected instance of the PersonDao Bean and through this is able to access the users and since monsters will most likely be accessed in reference to a user, the Monsters Bean will be accessed through this class.

5.2.9 PersonDAO

Has an entity manager to access the database. Persist will add a new entity to the database, Check email will take an email address and return true if this email address is found in the database, get person by email, will take an email and return the person that has that email.

5.2.10 MonsterDAO

This will provide much the same functionality as the person one only with reference to the monster table.

5.2.11 Entity Classes.

These are the form that entity is saved in the database, they contain all the attributes needed for each type and gets and sets for these attributes. They will be created in their respective session bean and added to the database files.

6. CLASS INTERFACE DESCRIPTION

6.1 Class Interface for Login Servlet

```
/* Outline for LoginServlet Class */


/*
This class handles the logging in for users.
It provides the ability to check the provided password and username
combinations against the database. Also allows performance of POST and GET
HTTP request methods.
*/
package [package name]
import [whatever]
public class LoginServlet {
      protected void doPost(){
      }
      protected void doGet(){
      }
      public boolean check(String name, String password){
      /* Takes name and password from form as parameters */
      /* Compares against database */
      /*
      Returns true if match, false if either String doesn't match
      */
      }
}
```

6.2 Class Interface for RegisterServlet

```
/* Outline for RegisterServlet Class */


/*
This class handles the registration of new users.
It provides the ability to check if the provided username exists in the
database. Also allows performance of POST and GET HTTP request methods.
*/
package [package name]
import [whatever]
public class RegisterServlet {
      protected void doPost(){
      }
      public boolean checkIfAlreadyExists(String email){
      /* Takes email provided by user as parameter */
      /* Compares against database */
      /*
      Returns true if match, false if String not found in database
      */
      }
}
```

6.3 Class Interface for FriendsServlet

```
/* Outline for FriendsServlet Class */


/*

This class handles all the functionality to do with managing the player's
friends list.

It provides the ability to send, accept and decline friend requests.
Additionally, it allows the user to pull up their list of friends or the
list of a friends monsters. Thirdly, allows the deletion of friends. Also
allows perfomance of POST and GET HTTP request methods.

*/

package [package name]

import [whatever]

public class FriendsServlet {

      protected void doPost(){

      }

      protected void doGet(){

      }

      public void sendFriendRequest(int friendID){

      /* Takes friendID (int) as a parameter */

      /* Searches database for selected friend ID */

      /* Sends request */

      }

      public void acceptFriendRequest(Friend friend){

      /* Takes specified person from list of requests */

      /* Adds to database of friends */

      }

      public void declineFriendRequest(Friend friend){

      /* Takes specified person from list of requests */

      /* Removes from list */

      }

      public List<Person> getFriendList(){

      /* Retrieves list of friends from database

      }
public List<Monster> getFriendMonsterList(Person friend){
      /* Searches database for friend specified in parameter */
      /* Pulls friend's list of monsters from database */
      /* Returns List<Monster> of friend */
      }
      public void deleteFriend(int friendID){
      /* Takes friendID parameter and searches database */
      /* Upon finding it, removes user from Friend list */
      }
      public boolean checkIfExists(int friendID){
      /* Takes parameter friendID as query for database */
      /* Searches for friend in the db */
      /* If found, return true, else return false */
      }
}
```

6.4 Class Interface for MyMonsterServlet

```
/* Outline for MyMonsterServlet Class */


/*
This class handles some of the basic functions to do with the player's
monsters.
It provides the ability to breed monsters, as well as check their status
and stats. Also allows perfomance of POST and GET HTTP request methods.
*/
package [package name]
import [whatever]
public class MyMonsterServlet {
	protected void doPost(){
	}
	protected void doGet(){
	}
	public Monster breed(Monster monster){
	}
	public Monster checkMonsterStatus(Monster monster){
	/* Takes parameter monster and accesses database */
	/* Returns monster and its stats (health, strength, etc */
	}
}
```

6.5 Class Interface for MarketPlaceServlet

```
/* Outline for MarketPlaceServlet Class */


/*
This class handles the various processes of the online marketplace for
trading monsters, etc.
It provides the ability to buy, sell and list monsters. Also allows
perfomance of POST and GET HTTP request methods.
*/
package [package name]
import [whatever]
public class MarketPlaceServlet {
      protected void doPost(){
      }
      protected void doGet(){
      }
      public void sellMonster(Monster myMonster, int money){
      /* Takes monster and amount of money (price) from params */
      /* Removes monster from player list and adds it to friend's */
      /* Decreases friend's money and increases player money */
      }
      public void buyMonster(Monster monster, int money){
      /* Takes monster and amount of money (price) from params */
      /* Adds monster to player list and removes it from friend's */
      /* Increases friend's money and decreases player money */
      }
      public void buyMonsterToBreed(Monster monster, int money){
      }
      public List<Monster> getMonsterList(){
      /* Returns list of Monsters */
      }
}
```

6.6 Class Interface for MonsterFightServlet

```
/* Outline for MonsterFightServlet Class */

/*

This class handles the functionality of the monster fights.

It provides the ability to accept, decline and send fight requests to users
on the player's friends' list. Additionally, manages the actual fights
between users' monsters. Also allows perfomance of POST and GET HTTP
request methods.

*/

package [package name]

import [whatever]

public class MonsterFightServlet {

      protected void doPost(){

      }

      protected void doGet(){

      }

      public void sendFightRequest(Person friend){

      /*

      Takes friend parameter and sends fight request to that person

      */

      }

      public void acceptFight(Person friend){

      /* accepts fight from friend parameter */

      }

      public void declineFight(Person friend){

      /* declines fight from friend parameter */

      }

      public void fight(int myMonsterID, int friendMonsterID){

      /*

      Begin fight between monsters, pulling stats from database by
      using monsterID

      */

      }

      public List<Monster> getMonsterList(Person friend){

      /* Retrieves list of monsters owned by friend */

      }

}
```

6.7 Class Interface for PersonDAO

```
package [package name]
import [whatever]
public class PersonDAO {
      private EntityManagerFactory emf;
      private EntityManager em;

      public void persist(Person person){
      }
      public boolean checkEmail(String email){
      }
      public Person getPersonByEmail(String email){
      }
      public List<Person> getAllPeople(){
      }
}
```

6.8 Class Interface for MonsterDAO

```
package [package name]
import [whatever]
public class MonsterDAO {
      private EntityManagerFactory emf;
      private EntityManager em;

      public void persist(Monster monster){
      }
      public List<Person> getAllMonsters(){
      }
      public Monster remove(Monster monster){
      }
}
```

REFERENCES

[1] Software Engineering Group Projects: Design Specification Standards. C. J. Price, N. W. Hardy. SE.QA.05A. 1.6 Relea\se

DOCUMENT HISTORY

Version	*CCF No.*	*Date*	*Changes made to document*	*Changed by*
1.0	N/A	06/12/12	N/A - original version	PW
1.1	N/A	07/12/12	Added architectural description, sequence diagram and class diagram	PW
1.2	N/A	07/12/12	Added class interface and dependency descriptions, fixed spelling/grammar, added reference.	PW
1.3	N/A	15/02/2013	Put in updated class diagram, now final version	PW