The Software Development Life Cycle – Group Project

Design specification

Author(s)	James Barnett, Nathan Hand, Daniel Steussy
Configuration Reference	SE_07_DEL_03
Last Updated	07/12/12
Version Number	1
Document Status	First release

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB

Table of Contents

INTRODUCTION

Purpose of this document

This purpose of this document is to describe the format of the design specifications for the application we are creating for our software engineering group project. This document describes the layout and structure of the design.

Scope

This document includes the following sections Decomposition description, dependency description, interface description, detailed design. This document should be read by all group project members.

Objectives

The objective of this document is to help aid the production of a design specification that is a complete and accurate translation of the client's requirements into a description of the design elements necessary for the implementation phase.

DOCUMENT CONTENT

Decomposition Description

Monster Mash Application

The Application to be created will create, manage and maintain an ongoing game called “Monster Mash”. The application will allow for multiple players to have “Monster farms” with various “Monsters” which allow for player to player interaction in the form of fighting, breeding & trading.

The application will run in a web based environment using JSON to send off data and receive data from the application server. The application will also communicate with a select number of other servers running a similar application, the servers should be able to operate with each other as if it was interacting with its self.

Monster Mash Significant Classes

The API Class

The API class is going to be our “hub” for communication between the client to server and server-server interactions. This will include setting up our requests, offers etc. Everything that happens between 2 players will need to go through this class sending and receiving data with the use of JSON.

The Player Class

The player class deals with getting our players friends, monsters, requests, offers etc. All things that our player is going to do inside of the application will be handled by the player class. Other things that occur with the players variables will also take place such as crediting them with money for a transaction such as a fight or debit the player because they accepted a breed offer.

The Monster Class

The monster class deals with our monster attributes, monsters have certain attributes such as strength, evasion, toughness etc. All of these attributes are determined by their age, so rather then storing this in our database we're going to generate our monsters attributes based on their age each time they have an interaction. Interactions will include things such as fighting, breeding etc. These interactions require another player and as such we want the most up to date data when we finally undertake our interaction.

The Request Abstract Class

The request Abstract class will be extended by FriendRequest and FightRequest the class will contain methods for Accepting and rejecting requests and then the extended classes will build upon those for specific functionality.

The Offer Abstract Class

The Offer Abstract Class is to be extended by BreedOffer and TradeOffer the class will contain methods for making the offer, sending the correct player and monster references etc. The extended classes will build upon the methods for specific functionality.

Monster Mash Package Descriptions:

aber.dcs.cs211.group07.data

The Data package includes classes such as Player, Monster, MonsterRefference, PlayerRefference and the interface reference.

This package contains the classes for the data that we're going to be working with, such as monsters having attributes which are useful and need to be known in fight request/ offers. The methods for getting our players monsters, friends, requests and offers etc.

aber.dcs.cs211.group07.data.offers

The Data.Offers package includes classes such as BreedOffer, TradeOffer & the abstract class Offer.

This package will deal with the sending of offers and the various offers that we have available to us, this can be easily expanded from breed and trading offers.

aber.dcs.cs211.group07.data.requests

The Data.Requests package includes classes such as FriendRequest, FightRequest & the abstract class Request.

This package will deal with the receiving/ sending of requests from other players these requests can be easily extended from friends and fight requests.

aber.dcs.cs211.group07.db

The DB package includes classes such as DatabaseConnector, MonsterTableConnector, PlayerTableConnector.

This package deals with the database interaction for the application, we have to create players,

monsters and store their data in the database all database interaction will be handled within this package.

aber.dcs.cs211.group07.tests

The Tests package includes classes for testing out data such as DataTests.

This package deals with the JUnit testing for our application, this will implement as many tests as practical from the test specification document to test our application, this will be especially useful throughout the project for monitoring progress of the application in a working state.

Monster Mash Application Requirements Mapping

The following table indicates the requirements that we meet and which classes meet those requirements.

Requirement	**Class providing requirement**
FR1	APIClass, DatabaseConnectorClass
FR2	PlayerClass, APIClass, DatabaseConnectorClass
FR3	PlayerClass, APIClass, DatabaseConnectorClass
FR4	PlayerClass, MonsterClass
FR5	APIClass, PlayerRefferenceClass, MonsterRefferenceClass, BreedOfferClass, FightRequestClass, FriendRequestClass.
FR6	APIClass, DatabaseConnectorClass, PlayerClass
FR7	PlayerClass, DatabaseConnectorClass, APIClass
FR8	PlayerClass, APIClass, DatabaseConnectorClass
FR9	RequestInterface, FriendRequestClass
FR10	RequestInterface, FightRequestClass
FR11	DatabaseConnectorClass, PlayerClass, APIClass

Dependency Description

Please see component diagram in appendix A located at the bottom of the document.

Interface Description

The Player Class, Public, no extensions.

This class contains numerous variables as well as many methods and contains all the important data in regards to the players of the application.

Variables

+ id: int	The id variable is of type int. This will be an auto number that continuously counts up to avoid giving two or more players the same id.
+ serverID: int	The serverID variable will be of type int. This

	will be for the same reasons as above. This will with the correct validation stop servers of the same ID connecting with one another.
+ email: String	The email variable is of type String. This is because it needs to contain letters, numbers and symbols. To make sure that the correct email format is used validation will be implemented.
+ password: String	The password variable is of type String for the same reasons as above. Validation will also be run on this to make sure the password is secure.
+ money: int	The money variable will be of type int as it will be a numerical value.

Methods

+ getFriends(): List <Player>	This method will get friends by showing a list of other players. This will allow the user to view a list of other players in their friends list.
+ getOffers(): List <Offer>	This method will allow users to view a list of the current offers.
+ getRequest(): List <Requests>	This method will allow users to view a list of the current requests.
+ getMonsters(): List <Monster>	This method will allow the user to get monsters from a list of monsters and display them.
+ debit (amount:int)	This method will allow an amount to be debited from the account of the player. As the money variable is an int this variable has to be of the same type.
+ credit (amount: int)	This method will allow an amount to be credited to the account of the player. As the money variable is an int this variable has to be of the same type.

The Monster Class, Public, No extensions.

This class contains numerous variables as well as many methods and contains all the important data in regards to the monsters of the application.

Variables

+ owner: Player	The owner variable allows the owner of a certain monster to be set. The owner will be of type Player.
+ name: String	The name variable allows the name of the monster to be set.
+ birth: final Date	The birth variable allows the date of the

	monsters birth to be set.
+ gender: final bool	The gender variable will be of type boolean. This will mean that the monster can be either male or female only.
+ health_lost: double = 0.0 AGE_RATE: Time	Health lost occurs from fighting as our monster can loose some or all of his health in a fight.

Methods

+ getHealth(): double	This method will allow the health to be returned for the monster.
+ getStrength(): double	This method will allow the strength to be returned for the monster.
+ getToughness(): double	This method will allow the monsters toughness to be returned
+ getEvasion(): double	This method will allow the evasion skill of the monster to be returned.
+ getAge(): Time	This method will allow the age of the monster to be returned based on time in relation to the birth date.

The Offer Class, Public, Abstract.

This class contains numerous variables as well as many methods and contains all the important data in regards to the offers in the application.

Variables

+ player: PlayerReference	This variable is the reference back to our player, so that when we send an offer to our friends we can use this reference to interact with the correct player's stuff.
+ monster: MonsterReference	This variable is the reference back to our monster, so that when we send an offer to our friends we can use this reference to interact with the correct monster from our player.
+ cost: int	This variable gets the cost based on the offer made.

Methods

+ getFromPlayer(): Player	This method will allow a player to get a monster from another player in the application.
+ accept(p:Player)	This method will allow a player to accept an offer from another player in the application.

The PlayerReference Class, Public, implements reference.

This class implements reference because reference is an interface for all references which we would require in an interaction. If we would later need to implement more references back to more information such as a “Team/ Clan” in a later development of the application for extra competitive game play.

This class contains the get for the player as we need to know which player we're dealing with in interactions.

Methods

+ get(): Player	This method will allow a player to be gotten from elsewhere in the application by the server. (i.e. this class is called in both the Offer and Request classes numerous times.

The MonsterReference Class, Public, Implements reference.

This class implements reference because reference is an interface for all references which we would require in an interaction. If we would later need to implement more references back to more information such as a “Team/ Clan” in a later development of the application for extra competitive game play.

This class contains the get for the monster as we need to know which monster we're dealing with in interactions between players and monsters such as fighting.

Methods

+ get(): Monster	This method will allow a monster to be gotten from elsewhere in the application by the server. (i.e. this class is called in both the Offer and Request classes.

The Request Class, Public, Abstract Class.

This class contains numerous variables as well as many methods and contains all the important data in regards to the requests that can be made in the application.

Variables

+ fromPlayer: PlayerReference	This variable holds the information of the “fromPlayer” which will hold the reference for which the request came from.
+ toPlayer: PlayerReference	This variable holds the information of the “toPlayer” which is the player receiving the request from the “fromPlayer”.

Methods

+ accept()	This method will allow a player to accept a request.
+ reject()	This method will allow a player to reject a request.

The BreedOffer Class, Public, Extends Offer.

This class extends offer because it's a type of offer, we could later add multiple offers and as such have an abstract class called offer which includes the outline of every offer.

This class is for the specific offer of a Breeding offer, all breed offers are made to all friends in the friends list and contain the Player and the Monster being offered for breeding.

Methods

+ accept(p:Player, m:Monster, c:Cost)	This method will allow a player to accept an offer from another player in the application, this includes a player and a monster as they're both arguments. Cost is also going to be taken from the accepting player.

The TradeOffer Class, Public, Extends Offer.

This class extends offer because it's a type of offer, we could later add multiple offers and as such have an abstract class called offer which includes the outline of every offer.

This class is for the specific offer of a Trade offer, all trade offers are made to all friends in the friends list and contain the Player and the Monster being offered for trade with the monster being essentially sold for money.

Methods

+ accept(p:Player, m:Monster, c:Cost)	This method will allow a player to accept an offer from another player in the application, this includes a player and a monster as they're both arguments. Cost is also going to be taken from the accepting player.

The FightRequest Class, Public, Extends Request.

This class extends Request because a fight is a type of request between two players that must be accepted the same as a friend request.

This class contains numerous variables as well as many methods and contains all the important data in regards to the FightRequests that can be made in the application.

Variables

+ fromPlayer: PlayerReference	This variable holds the information of the “fromPlayer” which will hold the reference for which the request came from.
+ toPlayer: PlayerReference	This variable holds the information of the “toPlayer” which is the player receiving the request from the “fromPlayer”.

Methods

+ accept()	This method will allow a player to accept a fight request and then the fight will take place on the server of the player accepting the fight.
+ reject()	This method will allow a player to reject a request.

The FriendRequest Class, Public, Extends Request.

This class extends Request because a fight is a type of request between two players that must be accepted the same as a fight request.

This class contains numerous variables as well as many methods and contains all the important data in regards to the FriendRequests that can be made in the application.

Variables

+ fromPlayer: PlayerReference	This variable holds the information of the “fromPlayer” which will hold the reference for which the request came from.
+ toPlayer: PlayerReference	This variable holds the information of the “toPlayer” which is the player receiving the request from the “fromPlayer”.

Methods

+ accept()	This method will allow a player to accept a fight request and then the fight will take place on the server of the player accepting the fight.
+ reject()	This method will allow a player to reject a request.

The MonsterTableConnector Class, Public, No Extensions.

This class deals with the connecting and editing of database information in the Monster table, this table contains the information of the monsters and who their owners are.

Variables

+Statement: Statement	Statement used to get result sets from the database.

+Results: ResultSet	Results from our database connection and query so that we can process information.
+MonsterTable: String	Table query variable, this can be changed for more specific functionality/ query results.
+Host: String	Variable for the host database address
+UserName: String	The database username variable.
+Password: String	The database password variable.

Methods

+createMonster(Monster mon)	The creation of a monster into the database taking a monster as the argument which will include all the information that a monster needs to be stored in the database
+deleteMonster(Monster mon)	The deletion of a monster from the database, taking a monster as the argument so that we can remove the monster after death or owner account de-activation.
+getMonster(int monID)	Getting the monster out of our database so that we can pass it to the monster class and generate our attributes before we send it off to fight etc.
+editHealthLost(Monster mon, double amount)	When we've taken damage from a fight we need to update our health lost to reflect that fact.
+getSex(Monster mon)	When putting up our monster for trading, breeding and generally purposes where gender is important we've got to find out the sex of our monster before hand.

The PlayerTableConnector Class, Public, No Extensions.

This class deals with the connecting and editing of database information in the Monster table, this table contains the information of the monsters and who their owners are.

Variables

+Statement: Statement	Statement used to get result sets from the database.
+Results: ResultSet	Results from our database connection and query so that we can process information.
+MonsterTable: String	Table query variable, this can be changed for more specific functionality/ query results.
+Host: String	Variable for the host database address
+UserName: String	The database username variable.
+Password: String	The database password variable.

Methods

+createPlayer(Player newPlayer)	The creation of a player into the database taking a player as the argument which will include all the information that a player needs to be stored in the database
+deletePlayer(Player newPlayer)	The deletion of a player from the database, taking a player as the argument so that we can remove the player after deletion of the account.
+getPlayer(int playerID)	Getting the player out of our database so that we can pass it to the player class and get our monsters etc for display back to the user.
+editMoney(Player player, int amount)	When we use the credit or debit methods in player we've got to then update the SQL with the appropriate data.
+login(String Username, String Password)	Server side login, we're logging out player into the application.

Detailed Design

REFERENCES

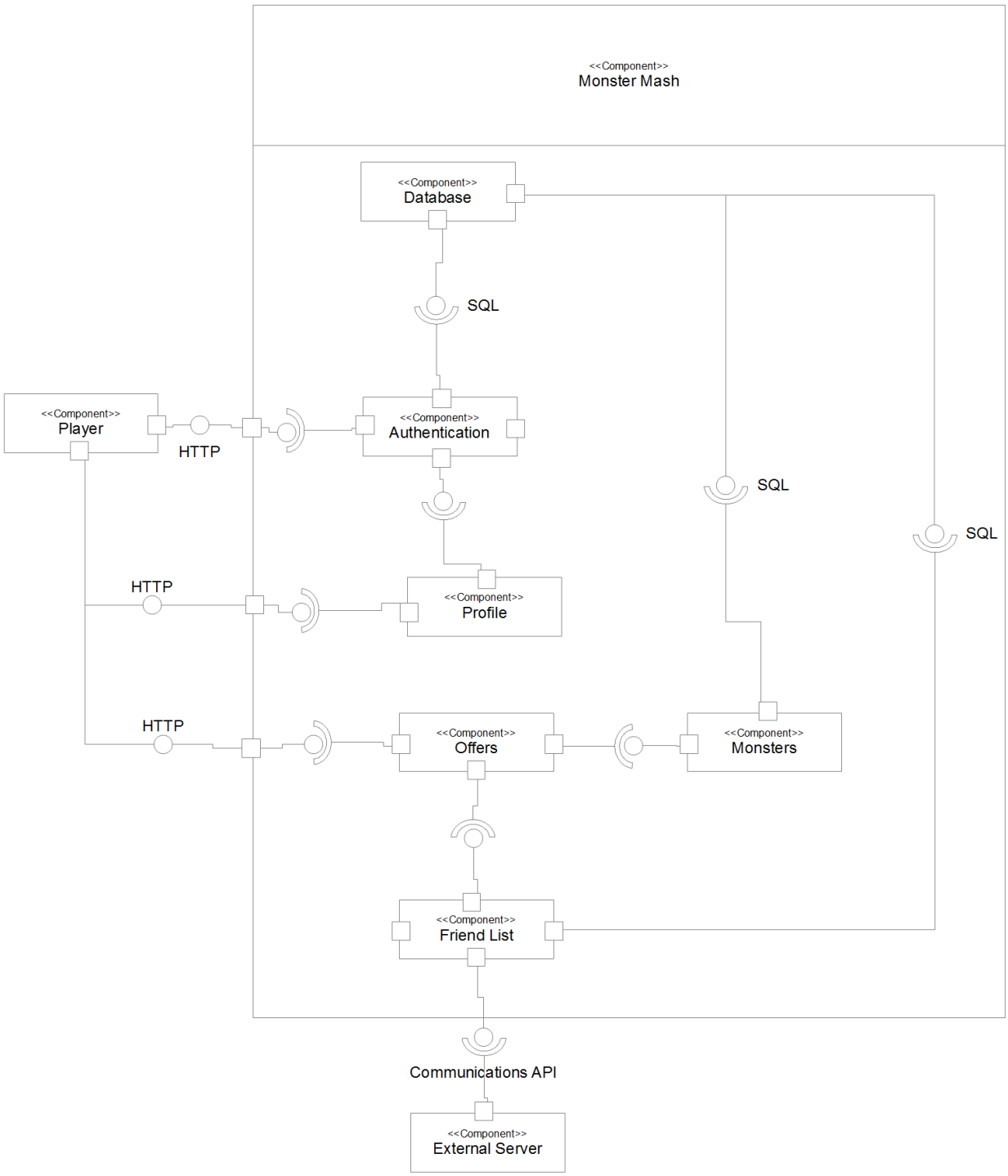
This section is an automatic bibliography.

DOCUMENT CHANGE HISTORY

Version	CCF No.	Date	Sections changed	Changed by
0.1	N/A	19/11/12	Initial file creation and introduction.	James Barnett
0.2	N/A	04/12/12	Initial interface descriptions for some classes.	James Barnett
0.3	N/A	04/12/12	Updated class descriptions and added package descriptions.	Nathan Hand
0.4	N/A	05/12/12	Update class and package descriptions, added requirements table.	Nathan Hand
0.5	N/A	06/12/12	Updated all the content and Merged content with template document.	Nathan Hand
0.6	N/A	07/12/12	Doing final checks/ re-arranging of content before upload.	Nathan Hand

Version	CCF No.	Date	Sections changed	Changed by
0.7	N/A	07/12/12	Added component diagram to appendix A.	Daniel Stuessy
1	N/A	07/12/12	First version to upload to client.	Nathan Hand

APPENDIX A


<<Component>>
Monster Mash
<<Component>>
Database
SQL
<<Component>>
Player
HTTP
<<Component>>
Authentication
SQL
SQL
HTTP
<<Component>>
Profile
HTTP
<<Component>>
Offers
<<Component>>
Monsters
<<Component>>
Friend List
Communications API
<<Component>>
External Server