Jam Games

Labyrinth
Software Architecture Document

Version 1.7

Revision History

Date	Version	Description	Author
03/mar/15	1.0	Introduction, Architectural Representation, Use-Case View Logical View(Overview)	Sergio, Luis, Tonatiuh.
10/mar/15	1.1	Added Process View and Implementation View.	Sergio, Luis, Tonatiuh.
14/mar/15	1.2	Added Quality section and both 8.1 and 8.2 sub-sections of the Implementation View.	Sergio, Luis, Tonatiuh.
24/mar/15	1.3	Modified Diagrams of the both 8.1 and 8.2 sections.	Sergio, Luis, Tonatiuh.
27/mar/15	1.4	Added the Business Layer content in the 8.1 and 8.2 sections.	Sergio, Luis, Tonatiuh.
18/apr/15	1.5	Added the Data Access Layer content in the 8.1 and 8.2 sections.	Sergio, Luis, Tonatiuh.
20/apr/15	1.6	Updated sections 2, 7 and 9.	Sergio, Luis, Tonatiuh.
04/may/15	1.7	Added multiplayer features.	Sergio, Luis, Tonatiuh.

Table of Contents

Software Architecture Document

1. Introduction

This document provides a high level overview of the architecture for the Labyrinth Project in Unity. It outlines the technologies that the members of the project will use for broad collaboration and participation.

The document provides a high-level description of the goals of the architecture, the use cases supported by the system and the architectural styles and components that have been selected to best achieve the aforementioned use cases.

1.1 Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system, each one explained in detail across the 11 chapters conforming this document.
In summary, this writing is intended to capture and convey the significant architectural decisions which have been made on the system and its functionality.

This document was wrote keeping in mind that the reader is well versed in the area of computer science and software development, so an advanced knowledge of software architecture and design patterns is recommended to fully understand the contents of this paper.

1.2 Scope

This Software Architecture Document provides an architectural view of the Labyrinth Unity Project. The majority of the sections take into consideration the main characteristics of the project according to their relevance to the final user experience.

1.3 Definitions, Acronyms, and Abbreviations

CRUD	**C**reate **R**etrieve **U**pdate **D**elete
Developers	The team responsible for the development of the software system.
Real-time	Functions performed in real time, which should allow the user to interactively perform the function and get feedback about what the function is doing.
Requirement	A condition or capability needed by a user to solve a problem or achieve an objective.
Specification	A document that describes, in a complete, precise, verifiable manner, the requirements, design, behaviors and/or other characteristics of a system or system component.
UML	**U**nified **M**odelling **L**anguage
UI	**U**ser **I**nterface
Use Case	A disciplined method of describing the typical behavior of a component of the system.

User	The person operating and/or using the software system.

1.4 References

A full repository of the versions of this document can be found at: https://github.com/SwordXZ0/DAS.git

1.5 Overview

The rest of this document will address architectural issues in the development of the web-based game with code name 'The Labyrinth Project', according with this subdivisions:

- Chapter 2: Architectural Representation
  This section describes what software architecture is used for the current system, and how it is represented.

- Chapter 3: Architectural Goals and Constraints
  This section describes the software requirements and objectives that have some significant impact on the architecture.

- Chapter 4: Use-Case View
  This section lists use cases or scenarios from the use-case model if they represent some significant or central functionality of the final system, or if they have a large architectural coverage.

- Chapter 5: Logical View
  This section describes the architecturally significant parts of the design model, such as its decomposition into subsystems and packages.

- Chapter 6: Process View
  This section describes the system's decomposition into lightweight processes (single threads of control) and heavyweight processes (groupings of lightweight processes).

- Chapter 7: Deployment View
  This section describes one or more physical network (hardware) configurations on which the software is deployed and run. It is a view of the Deployment Model.

- Chapter 8: Implementation View
  This section describes the overall structure of the implementation model, the decomposition of the software into layers and subsystems in the implementation model, and any architecturally significant components.

- Chapter 9: Data View
  A description of the persistent data storage perspective of the system.

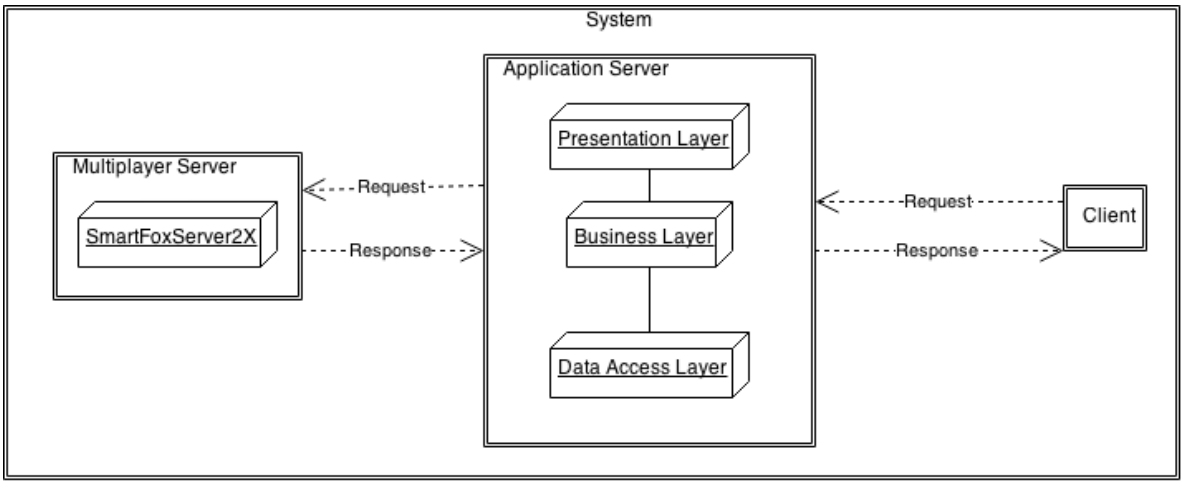
- Chapter 10: Size and Performance
  A description of the major dimensioning characteristics of the software that impact the architecture, as well as the target performance constraints.

- Chapter 11: Quality
  A description of how the software architecture contributes to all capabilities (other than functionality) of the system: extensibility, reliability, portability, and so on.

2. Architectural Representation

This application was designed using a client-server architecture, in combination with a standard three layer architecture, with a Presentation Layer, a Business Layer and a Data Access Layer. The Presentation Layer contains the main visible web page and handles all input from and output to the user. The Business Layer handles all of the Business logic and mechanics for the game, and the Data Access Layer provides an abstraction to the database and a connection to it, which provides the persistence required for the system.

This document presents the aforementioned architecture as a series of 4 views: The Logical View shows a quick overview of all of the basic subsystems in the application and gives a basic overview of the application as a whole. The Deployment view tells how the system is physically configured. The Implementation View gives a more in-depth view into how the system has been implemented. And finally, the Data view shows how the Database is setup and structured.

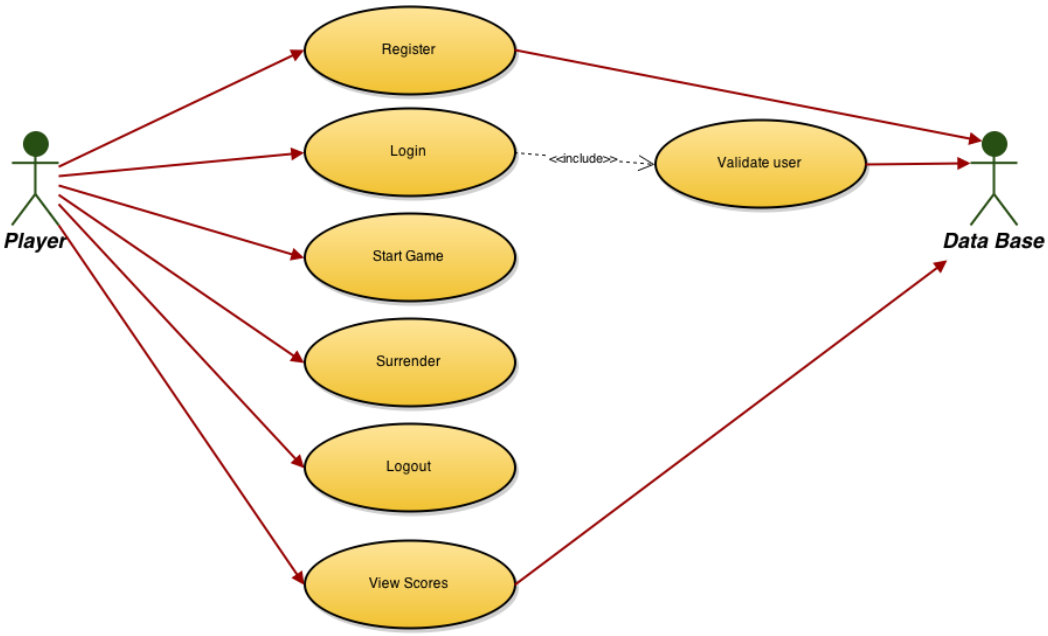


3. Architectural Goals and Constraints

There are some key requirements and system constraints that have a significant bearing on the architecture. They are:

1. The system must be uploaded to a server which must be kept functional at all times.
2. The users must count with connection to the web (internet) in order to access the aforementioned server.
3. Only two users may connect at one session of the game.
4. The user's usernames and scores must be kept in a database.
5. The users must have installed the Unity Web-player.
6. The game should be independent of the operating system in the client machine.
7. The game should work on every web-browser.
8. Users should be able to see the top 6 user's scores.
9. All remote accesses are subject to user identification and password control.
10. The architecture should be flexible and extensible, to ensure reusability for the next phase of the development.
11. Use best practices to assemble a light weight, agile, easy to test, and easy to maintain software system.

4. Use-Case View



- Register:

  Brief Description
  This use case allows a Player to sign up into the system if it's the first time that he enters the game, or if he just wants to create another account with another username. A new account must have a unique username and a valid e-mail address. Registrations that do not meet these guidelines will be denied. The main actor of this use case is the Player. The Data Base System is an actor involved within this use case.

- Login:

  Brief Description
  This use case describes how a user logs into the Labyrinth Game System. The actor starting this use case is the Player. This scenario use the Validate User use case.

- Validate User

  Brief Description
  This use case describes how the system validates if the credentials of a user exist on the Database or not. The actor starting this use case is the system itself, after the Login use case is called. The Data Base System is an actor involved within this use case.

- Start Game:

  Brief Description
  This use case allows the Player to start a new game. A new instance of the game will have to wait until a second Player joins the party created by the first player. The actors of this use case are the

Players. Both of the players need to be logged into the system before the game starts, otherwise the game will ask them to Login first.

- Surrender:

Brief Description
This use case allows a Player to terminate the game early, before there is a winner. In this case the player remaining in the game will also be removed from the match. The actor of this use case is the Player.

- Logout:

Brief Description
This use case allows a Player to exit the game system. To be able to access this option, the Player first should have to be logged into the System. The actor of this use case is the Player.

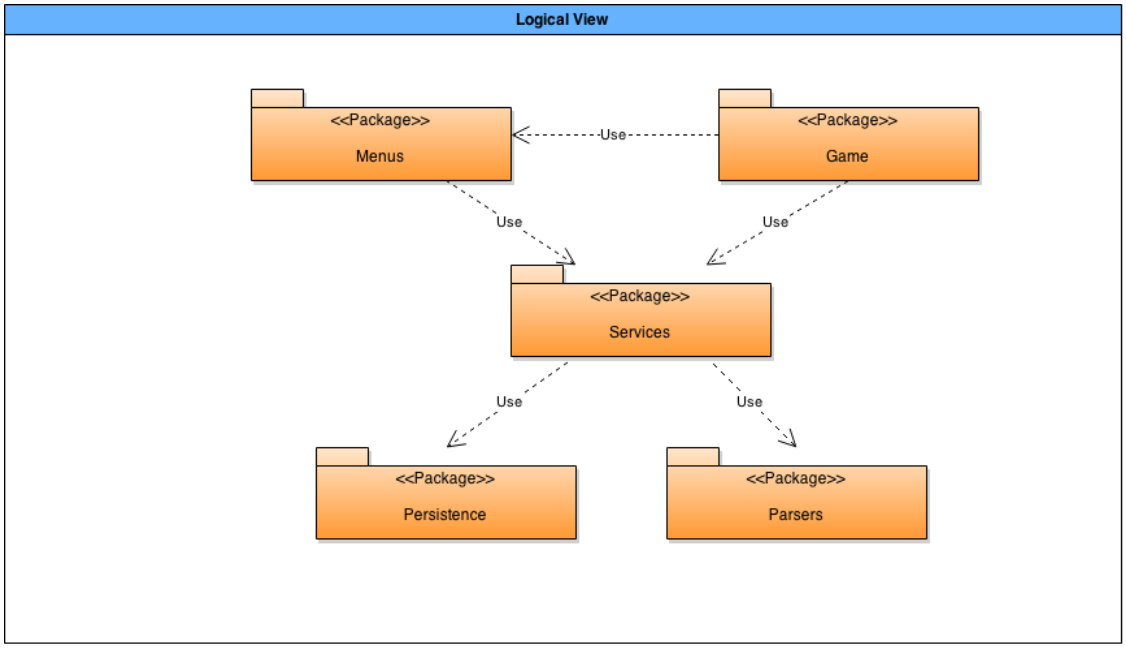
- View Scores:

Brief Description
This use case allows a Player to view the top 6 user's scores. The Player is the actor of this use case. The Data Base System is an actor involved within this use case.

5. Logical View

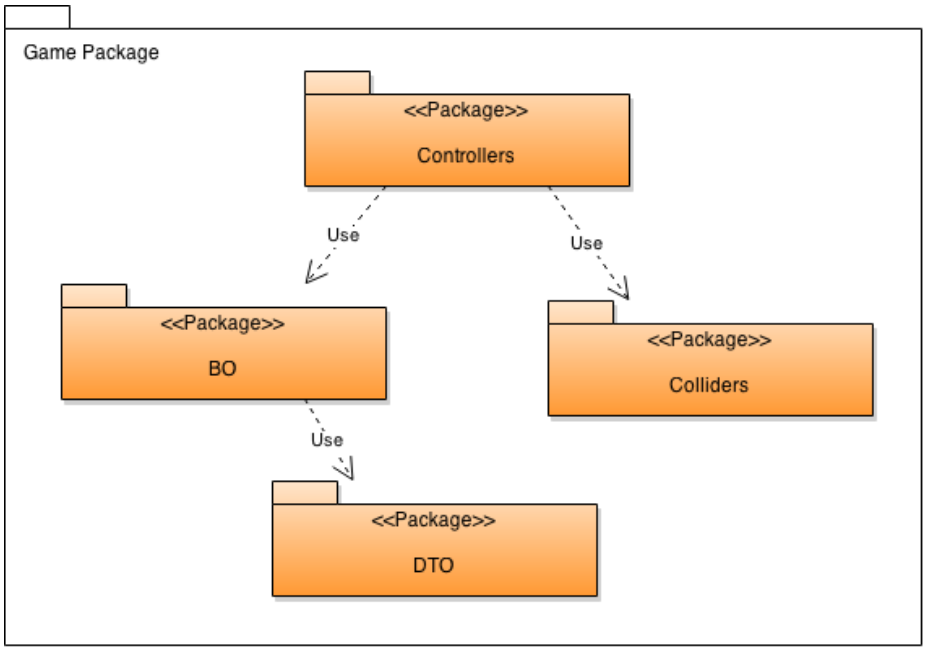
The logical view of the Labyrinth game system is comprised of 5 main packages: Game, Menus, Parsers, Services and Persistence.

5.1 Overview

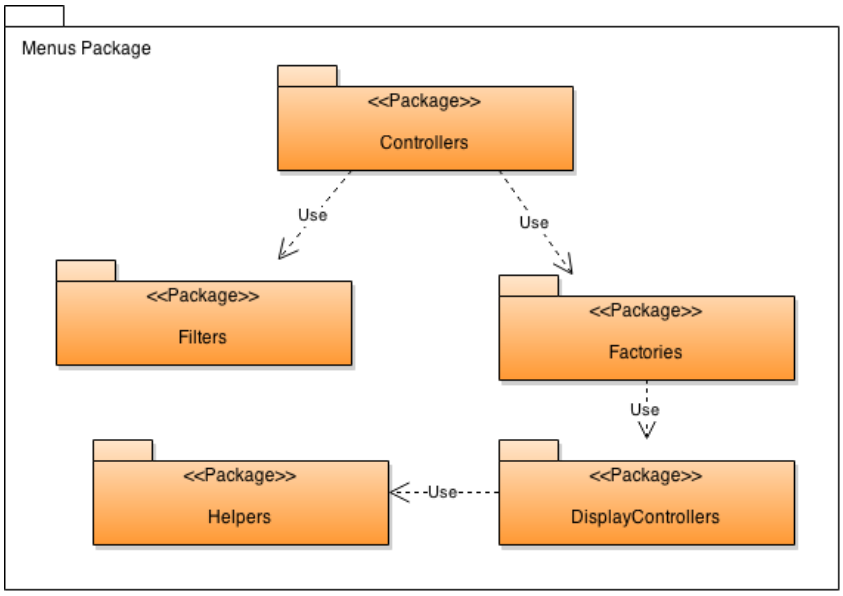


5.2 Architecturally Significant Design Packages

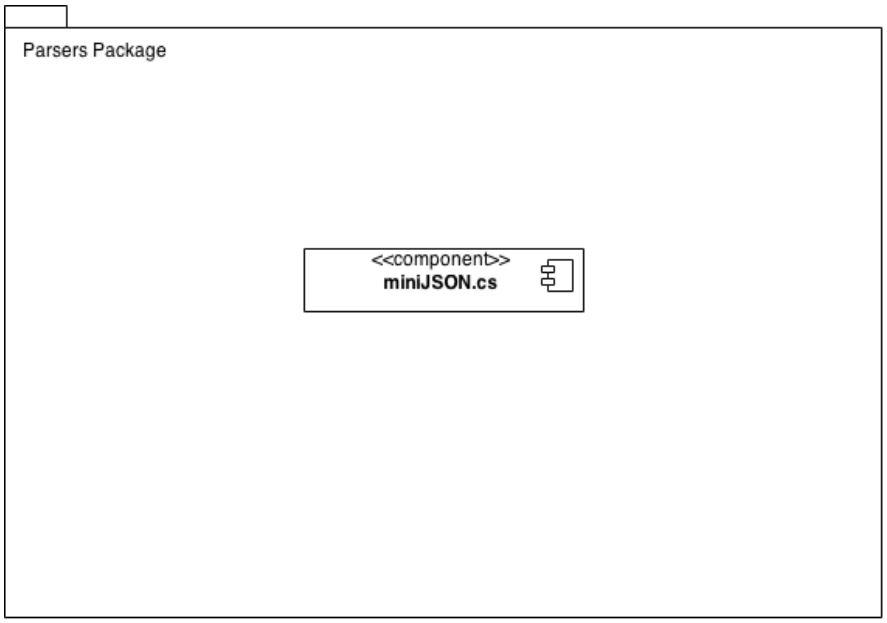
- The Game package is composed of 4 sub-packages:
    - Controllers: Its main component is the GameController.cs file, which is the code that manage the main mechanics of the game and the multiplayer interaction between 2 users.
    - Colliders: It stores the FinishSpot.cs file, which is the component that manage the collision with the goal spot of the game.
    - BO: It stores the Timer.cs and Players.cs files, which are the main objects business of the system.
    - DTO: It stores the time and user data transfer objects for the system.



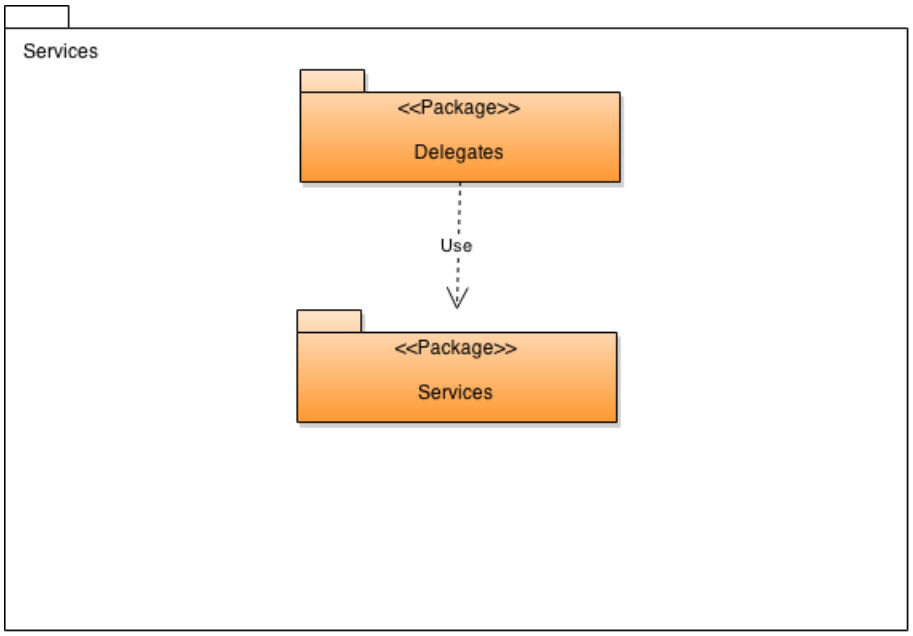
- The Menus package is composed of 5 sub-packages:
    - Controllers: It stores the FrontController.cs, which is the first logic controller that application uses to determine the navigation UI path for the user.
    - Filters: It stores all the filters used to validate the input fields in the menus and the session of the player.
    - Factories: It stores the MenuFactory.cs, which is the file in charge of instantiating the different menu components of the application.
    - DisplayControllers: It stores all the logic functions of the menu components and all the actions that they shall execute.
    - Helpers: It stores the MenuHelper.cs file, which manage all the repetitive operations of the menu views, like validating empty fields.



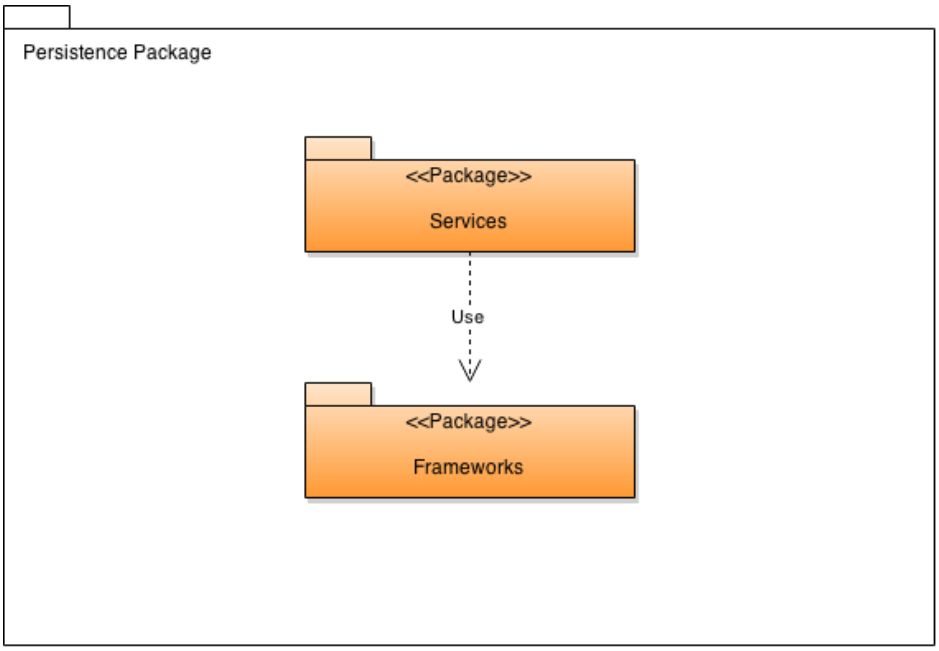
- The Parsers package is only composed by the miniJSON.cs file, which is a simple parser for JSON files.



- The Services package is only composed of 2 sub-packages.
    - Delegates: It stores the BusinessDelegate.cs and the LookUpService.cs files, which are in charge of finding and executing a specific service requested by the application.
    - Services: It stores all the services needed by the application.



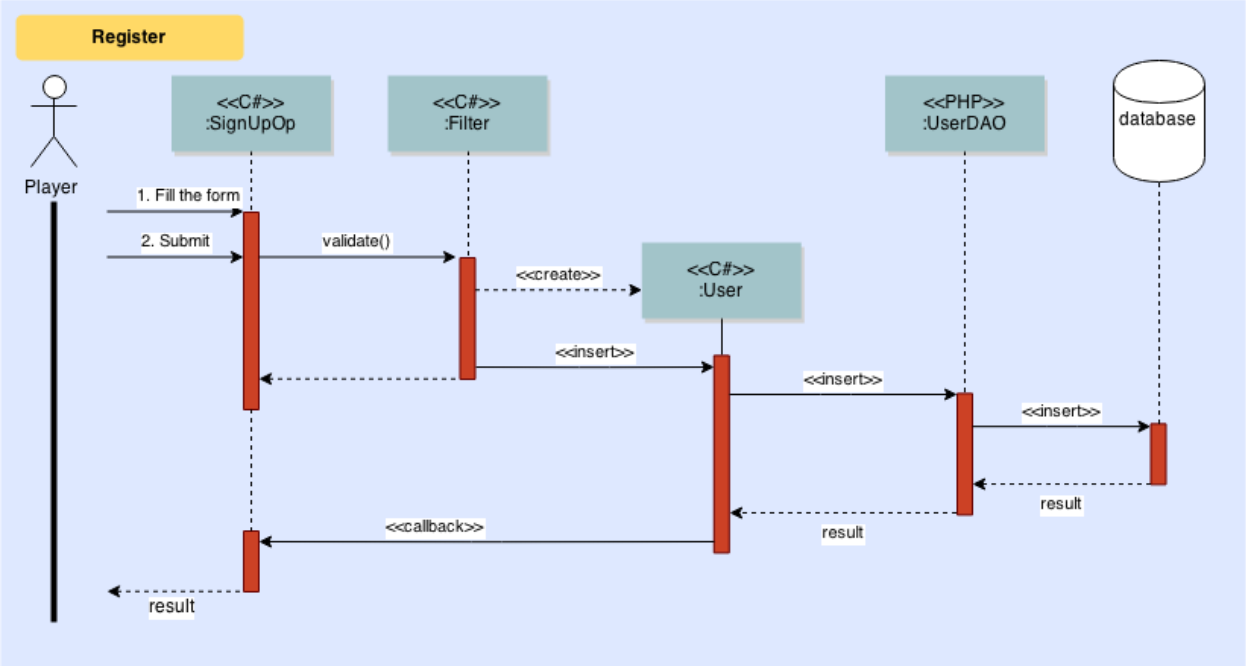
- The Persistence package is only composed of 2 sub-packages:
    - Services: It stores all the database services part of the business logic services.
    - Framework: It stores the data access framework used by the application.



5.3 Use-Case Realizations

There are two main use cases that are critical for the operation of the system: The Register use case and the star game use case, both discussed in detail below.

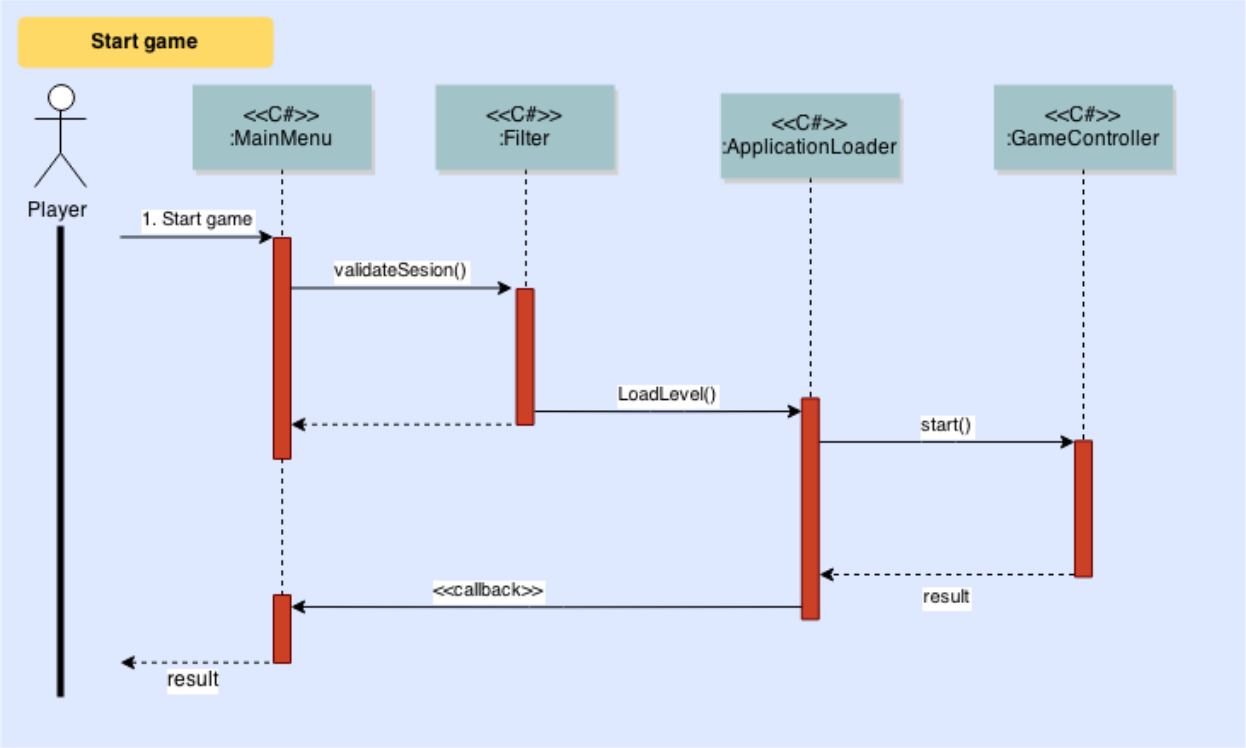
- Register use case.



Participating objects

Object	Class	Description
:SignUpOp	UI	Manage all GUI elements and user interactions.
:Filter	Filter	Acts as a liaison between the GUI components and the User Entity Class.
:User	User	Represents an entity model class in the system for a Player.
:**UserDao**	UserDAO	Serves as an aggregation of all the CRUD functions for a User Entity Model class for data persistency.

- Start game use case.



Participating objects

Object	Class	Description
:MainMenu	UI	Manage all GUI elements and user interactions.
:Filter	Filter	Acts as a liaison between the GUI components and the Level Entity Class.
:**ApplicationLoader**	Application	Manages all the resources and assets of the game and charges into memory.
:**GameController**	GameController	Starts all the mechanics of the game.

6. Process View

The Labyrinth game described in this document will have two main kinds of processes:

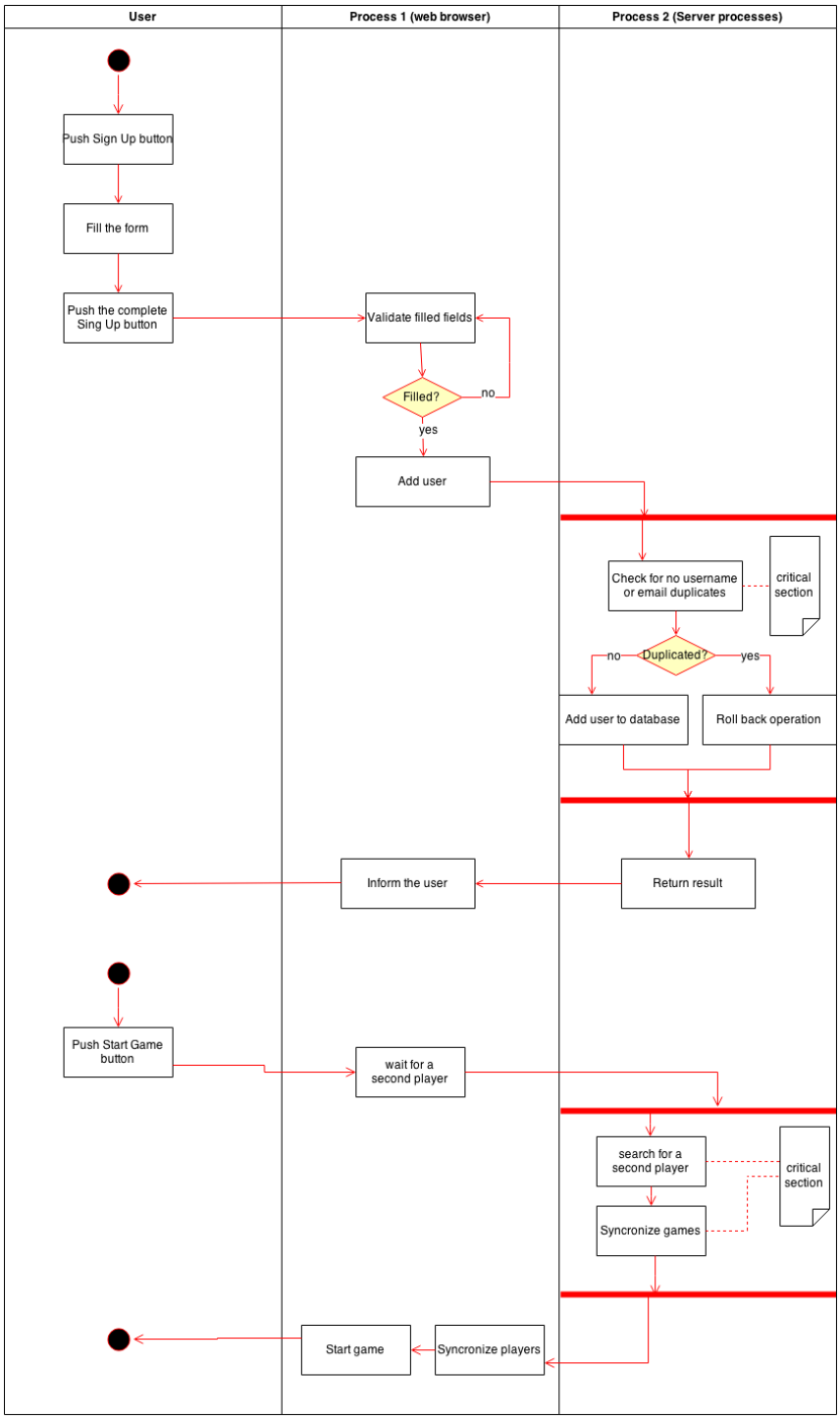
- Client side (lightweight processes): Client will have only one thread of the game execution which is supported by any web browser.

- Server side (heavyweight processes): Server may have as much threads as number of online users. Concurrency control in server side for providing appropriate information for users and affecting the system state is a significant problem here. Concurrency problem may occur in different states

of the system mainly in those listed below:

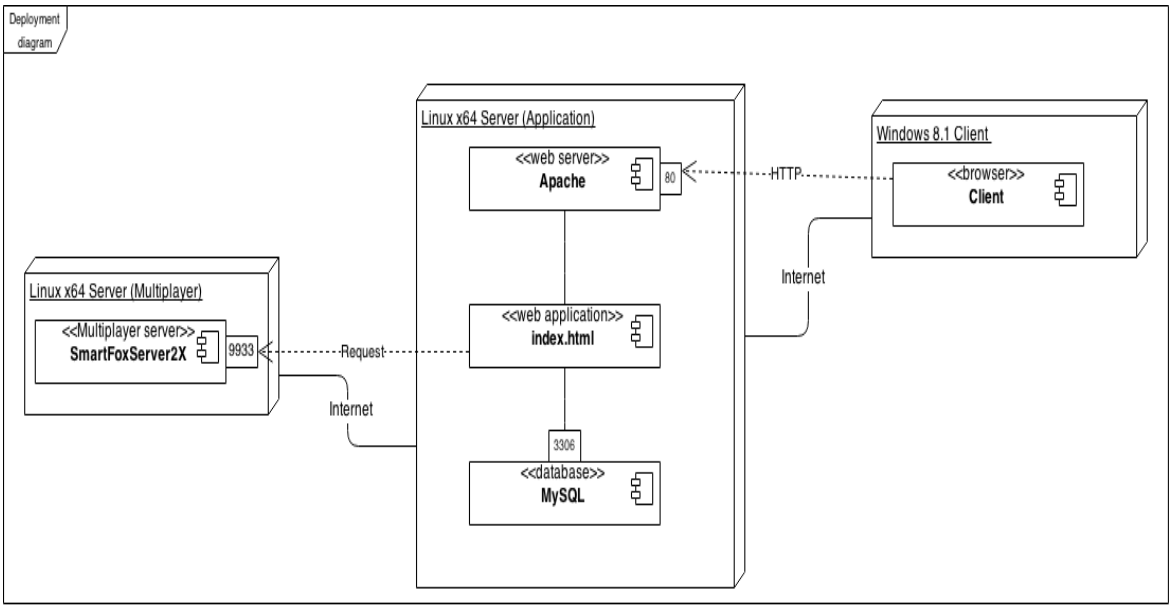
- Sign Up into the system.
- Wait for the second player.

Thus, the interaction of that processes is presented on the following activity diagram:

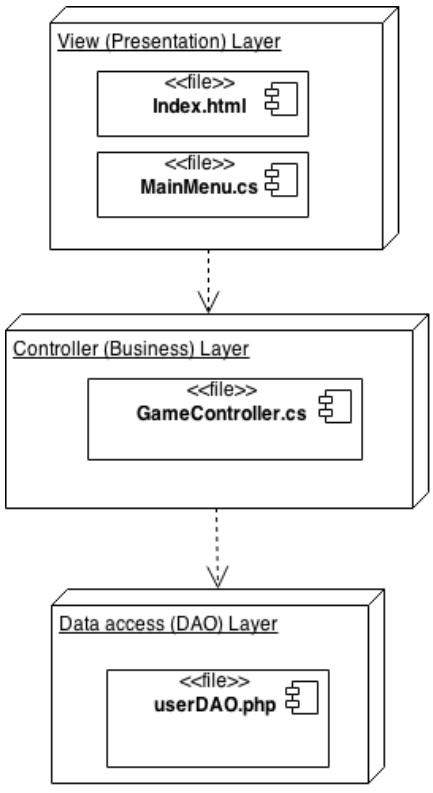


7. Deployment View

This diagram illustrates the system deployed on two 64-bit Linux servers and a PC client with Windows 8.1. However, the application shall work on any pc client regardless of the operating system that it has.



8. Implementation View



The overall system will be divided in 3 main logical layers: The Presentation Layer, The Business Layer and the Data Access Layer. Each one is described briefly bellow:

- Presentation Layer: This layer will group all the components that will interact directly with the user, and in this case the most important files of this layer will be the index.html file, which is the first GUI that user will see and use to interact with the system once that he has made an http request with a web browser, and the MainMenu.cs file, which will contain all the critical behavior methods for the user interface of the game.

- Business Layer: This layer will group all the components that will handle all the mechanics and control of the game, as well as the multiplayer features. The main file of this layer will be the GameController.cs file, which will group all the core behavior functions of the game system.

- Data access Layer: This layer will group all the components that will handle the interaction between the system and the database. The main file of this layer will be the userDAO.php file, which will be used to retrieve and storage information about the users of the system.

8.1 Overview

I. Presentation Layer:
The Presentation Layer will be composed of the c# files that manage the visual assets and aesthetics of the game, and also will contain the main HTML page that the users will see when they enter to the system.

The Presentation layer was designed taking into consideration the following process and structure:

1. Client: Web browser.

2. Technology: HTML and C#.

3. Modular GUI and navigation: The GUI navigation of the system will consists mainly of 4 different menus: the main menu, the scores menu, the login menu and the pause menu, each one with its own controller class and with a hybrid navigation between them. This GUI components shall be designed to match the usability heuristics described by Jacob Nielsen, and the Gestalt psychology, ensuring continuity and proximity between relevant GUI components, and keeping its meaning clear to the user with a minimalist design. For a more detailed description, review the navigation diagram and the component diagram of the Presentation Layer in the sub-section 8.2 of this document.

4. Entity business object components in the Presentation Layer: There will be a main component that will represent the user entity of the Business Layer in the Presentation Layer. For a more detailed description, review the component diagram of the Presentation Layer in the sub-section 8.2 of this document.

5. For this particular layer, the next architectural framework elements were considered:

    - Caching: Only not sensible data that could be retrieved again easily if the user erase this data should be kept in the client cache, like username or scores, as well as game visual assets. Session will be kept on the client side with the username of the player.

    - Validations: Field validations for the sign up and login processes shall be done in the client portion of the application, in order to decrease the overhead in the server side. Only the

validation of the session for a certain user should be done on the server side.

- Restriction and access control: Access control should only be determined and assigned by the manager of the system with the implementation of privilege roles. All the users shall have by default the lowest privileges to see data on the presentation layer, which only should permit the visualization of the username of other players and its scores.

- Communication with the Business Layer: The game system will use a "Chatty" communication pattern, with multiple short requests to the business layer on a needed basis. The presentation layer also should avoid duplicated transactions with the implementation of button and key locks once that they have been pressed for a certain transaction.

- Navigation: The navigation in the graphical user interface of the system should be lineal between the main components, and with a tree hierarchy between sub-components.

- Session Management: The user session shall be independent of the navigation state, and it should be kept and tracked by the client side of the application.

- Performance: All of the user interface should run smoothly and with responsiveness, updating the interface on time on both of the player's PCs in a same party of the game.

6. Design patterns: The design patterns used to design this layer were the Intercepting Filter pattern, the Front Controller pattern and the View Helper pattern, in order to contribute to a greater modularization. For a more detailed description, review the class diagram of the Presentation Layer in the sub-section 8.2 of this document.

II. Business Layer:

The Business Layer will be composed of all the c# files that manage the behavior and mechanics of the game, as well as the sign up and log in operations of the system. This layer will provide 3 main services which will be the managing of the user session, the registration of a new user, and the control of the game mechanics.

Thus, for this particular layer and the services mentioned above, the next architectural framework elements were considered:

- Authentication: The access to the user authentication operations shall only be accessible through the Presentation Layer, with the implementation of a business delegate class to encapsulate and modularize the functionality and data transition between the layers.

- Authorization: The authorization will be handled and implemented with the use of privilege roles, assigning by default the lowest level of privileges to the players of the game. Only one higher level of privileges should be permitted, corresponding to the administrators and managers of the system.

- Validation: The data coming from the Presentation Layer shall be validated with the implementation of a SQL parser, in order to avoid malicious string queries that try to attempt an SQL injection attack on the system.

- Maintenance: All the code of the Business Layer shall be designed according with the GRASP principles (object-oriented design General Responsibility Assignment Software Patterns) and the GoF (Gang of Four) patterns, to ensure high cohesion, low coupling and high modularization between the components of system, providing reusable code that shall make the system easy to maintain and to extend. Also a Business Delegate pattern will be a key factor for this objectives.

- Caching: Singleton classes shall be implemented to prevent duplicated processes per instance of the game. Fixed parameters like the connection to the database should be stored whenever possible in the business cache to enhance the performance of the server.

- Instrumentation: All the tracking, monitoring and managing of the system resources and business events shall be done through the web-based tools of a SmartFoxServer service.

- Exceptions Handling: Exceptions shall not handle the flow of the business logic and shall only be thrown to deal with unexpected situations and unnatural behavior of the system. In that case, the exception should be reported to higher layers in a clear, relevant, concise and user-friendly format, without compromising the security of the system.

- Logging and Audit: Only the access time and changes made by the game masters and administrators of the system shall be tracked, in order to maintain a record of the main changes made to the system. The actions of the general users in the system only will be recorded to generate performance statists about the system.

For a more detailed description of this layer, please review the component and class diagrams in the 8.2 section.

III. Data Access Layer:

The Data Access Layer will be composed of all the PHP files that manage the connection between the database and the logic of the application, and will be in charge of all the necessary CRUD transactions to operate the data.

The Data Access Layer was designed taking into consideration the following process and structure:

1. Stage: In this system, the database will be generated from scratch alongside the application. Thus, the development process will be on a “greenfield” stage, because the developers will have full control over the decisions made about the design of the database.

2. Entities format and types: All the entities of the Data Access Layer will be generated from PHP classes and its output will be in a JSON format, with the objective of an easy integration of future modules of the game that will probably use this persistence layer, like a mobile version of the game.

3. Service agents: There will be no service agents required, as there will be only one data source for this system.

4. Data access technology: In this system, a MySQL database will be implemented due to being the world's most popular open source database, having a huge documentation and a large community support. Added to this, as our business model classes and our relational model tables in the database are equal between them, and since we do not need detailed control over the SQL queries made to the database, but we need a good simplicity of implementation and a good adaptability to data model changes, the RedBeanPHP framework will be used for the CRUD operations and object-relational mapping.

5. Data access components: The main data access components in this layer will be the user and time data access objects. However, this classes will be auto-generated dynamically on runtime by the RedBeanPHP persistence framework mentioned in the previous point, according to the tables in the database system.

6. For this particular layer, the next architectural framework elements were considered:

    - Stage: As mentioned before, the development process will be on a “greenfield” stage, because the developers will have full control over the decisions made about the design of the database.

    - Batch: All processes only shall be executed when the Business Layer send an explicit request for a query to the database.

    - Cache: The data resulted from a query will only be stored until it no longer be needed by the application and all the required data has been send to the Business Layer.

    - BLOBs (Binary Large Objects): For no reason BLOBs should be stored on the database. Only references to the files shall be stored.

    - Exceptions: No database exceptions shall be passed to the Business Layer. User only should be notified to press the button again for retry the transaction that was attempting.

    - Data format: As stipulated on the previous section, all the output data of this layer will be in a JSON format, with the objective of an easy integration of future modules of the game that will probably use this persistence layer, like a mobile version of the game.

    - ORM (Object-Relational mapping): In this system, the RedBeanPHP framework will be use for the encapsulation of the object oriented classes that represent their respective table on the database. All the data types of the parameters in the database tables will be kept the same through the mapping. For a more detailed specification of the structure of this tables, please review the enhanced entity–relationship model and the component diagram of the 8.2 section of this document.

    - Transactions: Every transaction shall be independent and should not affect other transactions. The database itself will manage concurrency problems to avoid data corruption, so the application will operate on an optimist pattern of concurrency (not blocking data).

    - Connections: A connection per transaction will be used, and it shall be opened only when the application need it and closed as soon as the transaction has ended. In other words, open it later and close it soon.

    - Queries: Every query to the database shall be constructed using the parametrized queries tool that the ReadBeanPHP framework provides, with the objective of improve the security of the system.

    - Store Procedures: Under no circumstance store procedures should be use into the database, unless they have been implemented for the purpose of keeping track of transactions and changes made to the database, with no business logic operations within them.

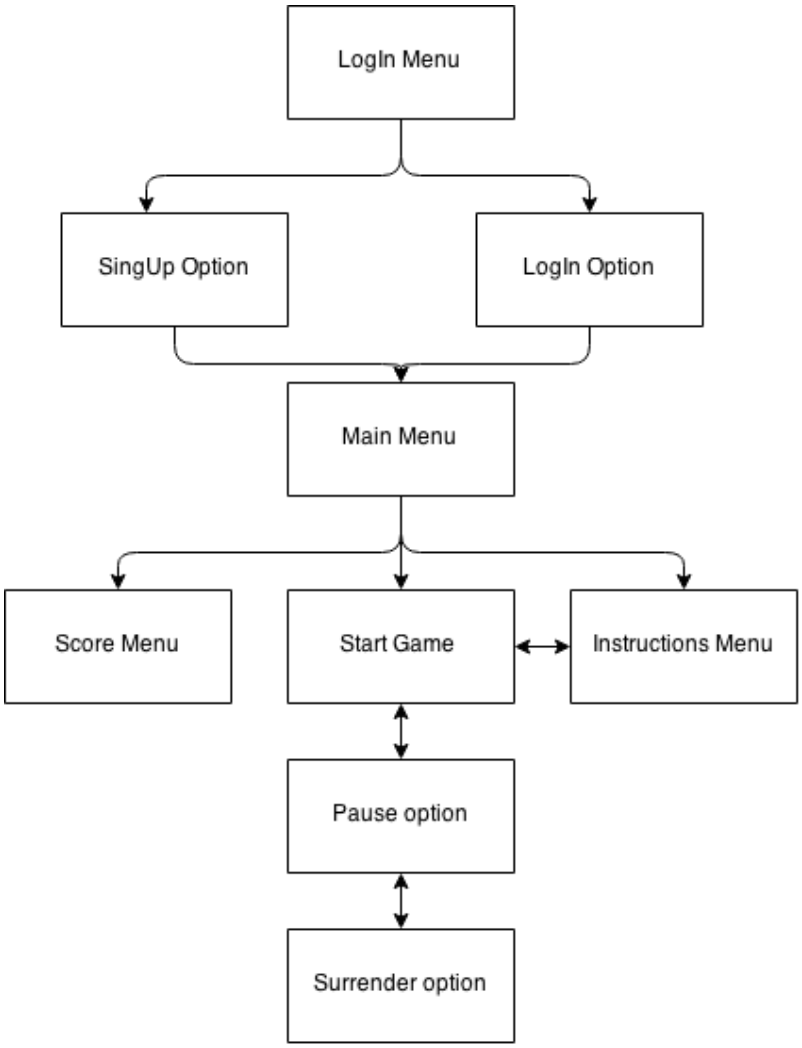
    - Validations: Every transaction should check that the data send by the Business Layer is complete and does not have null fields.

For a more detailed description of this layer, please review the entity-relationship diagram in the 8.2 section.

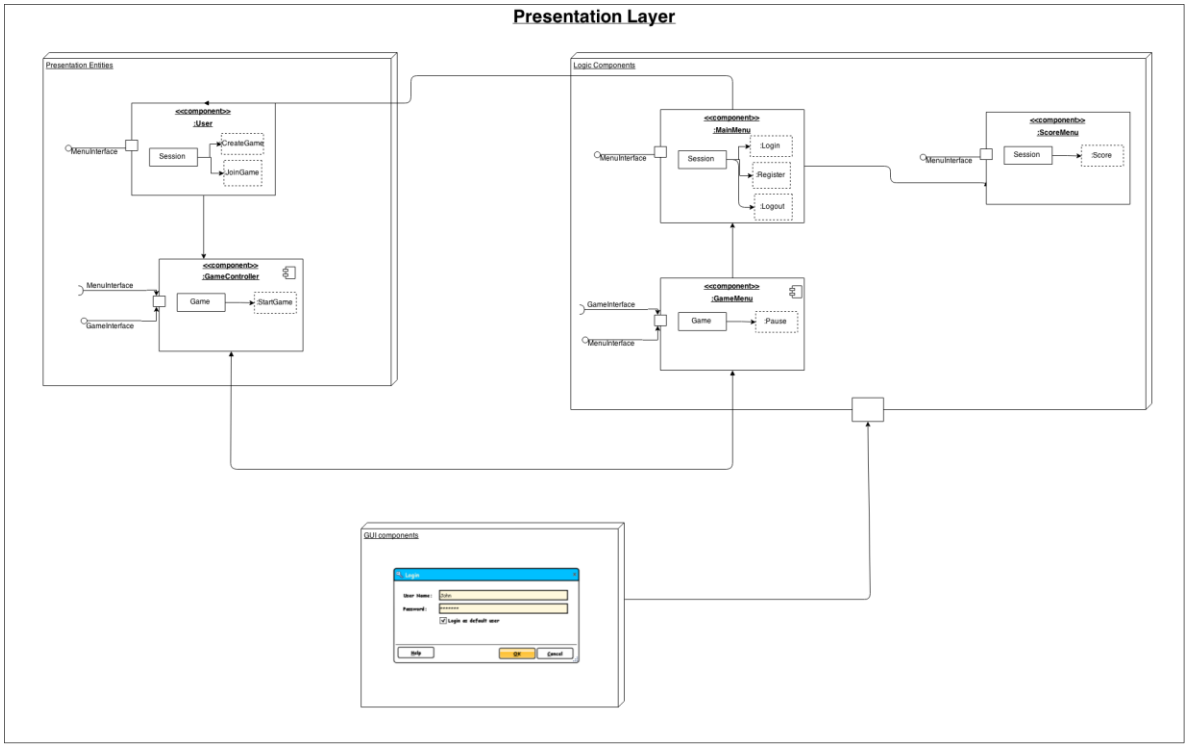
8.2 Layers

I. Presentation Layer:

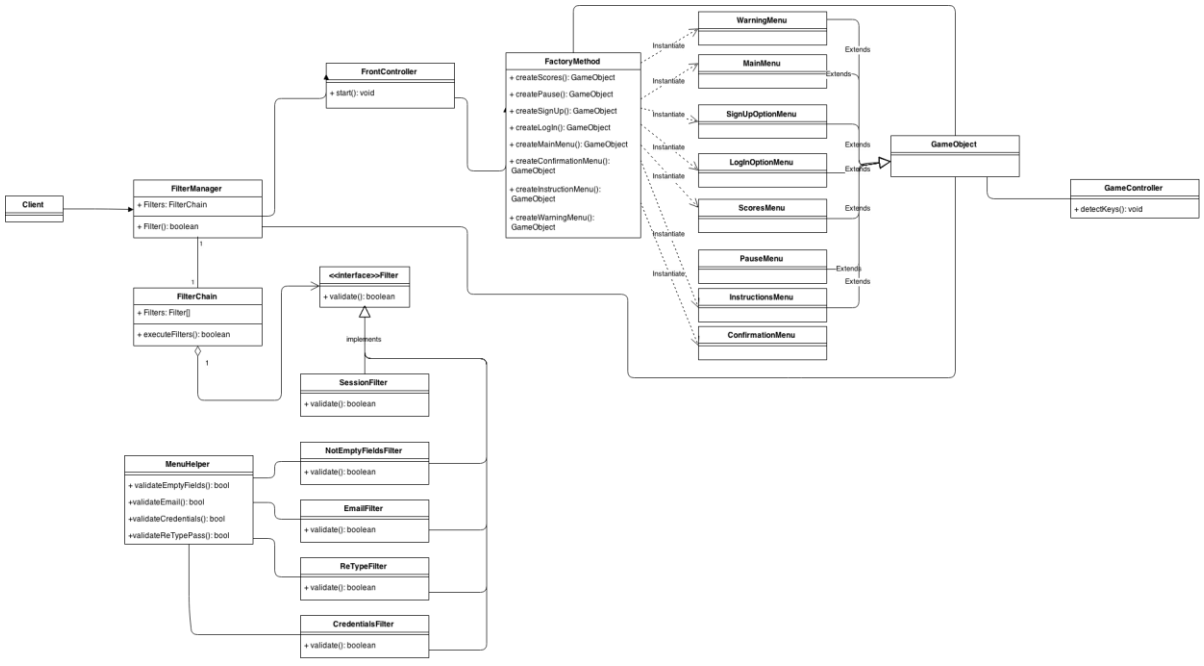
- Navigation Diagram:



- Component Diagram:

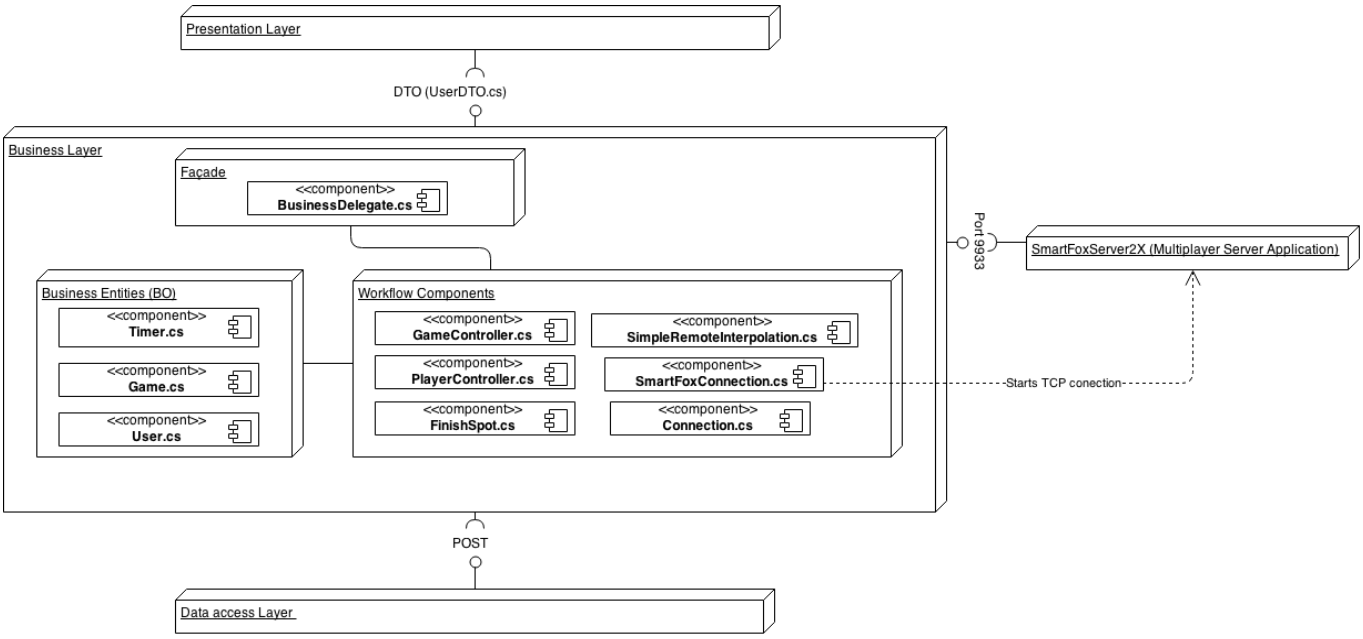


- Class Diagram:

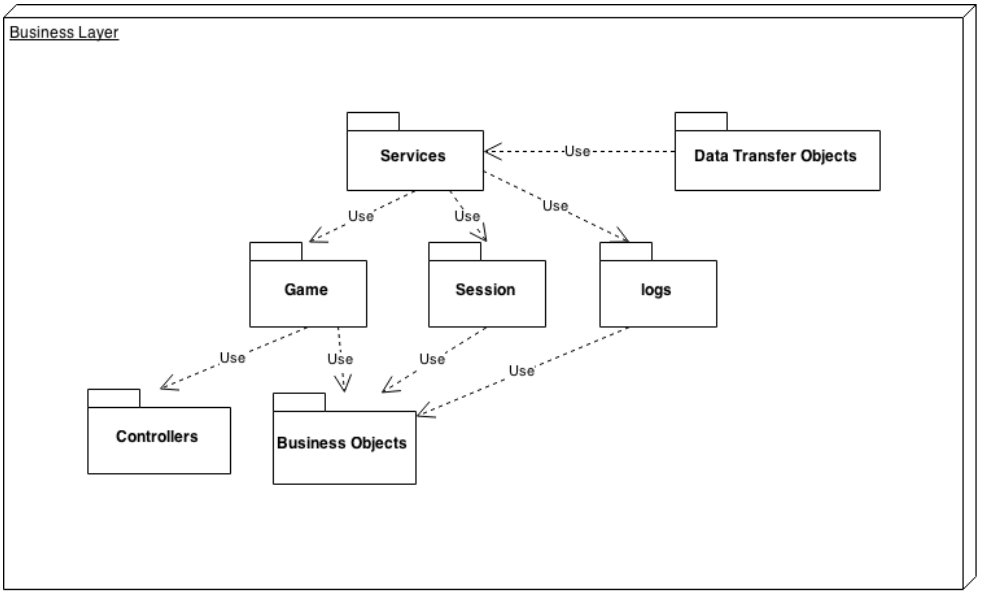


II. Business Layer:

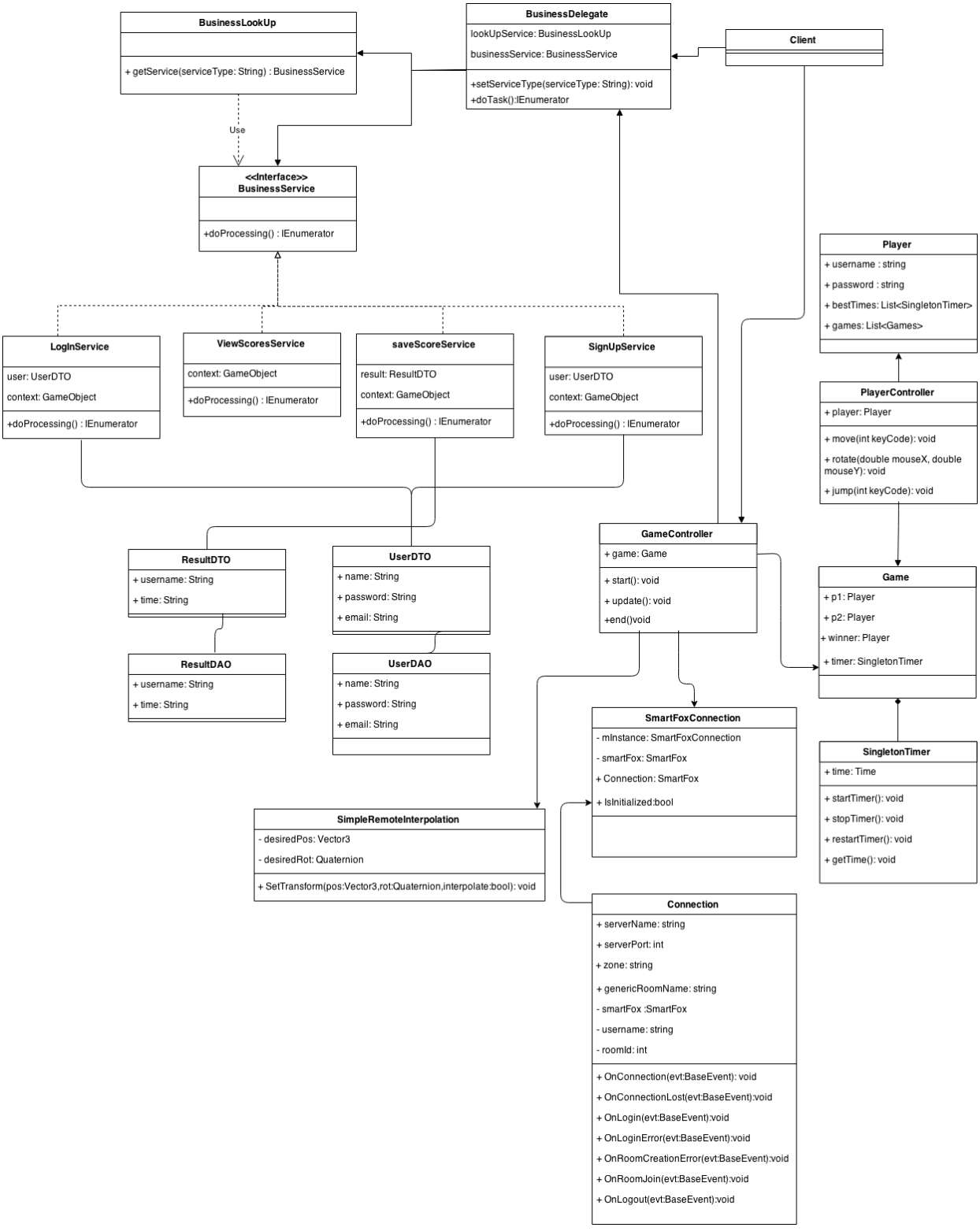
- Component Diagram (main components):



- Package Diagram:

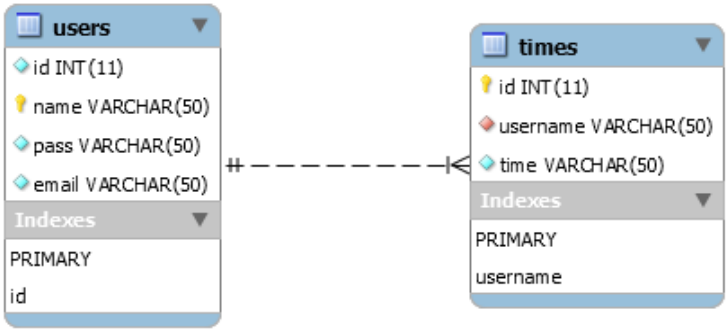


- Class Diagram:
  The main business patterns used to implement this layer were The Business Delegate pattern, The Service Locator pattern, and the Transfer Object pattern.

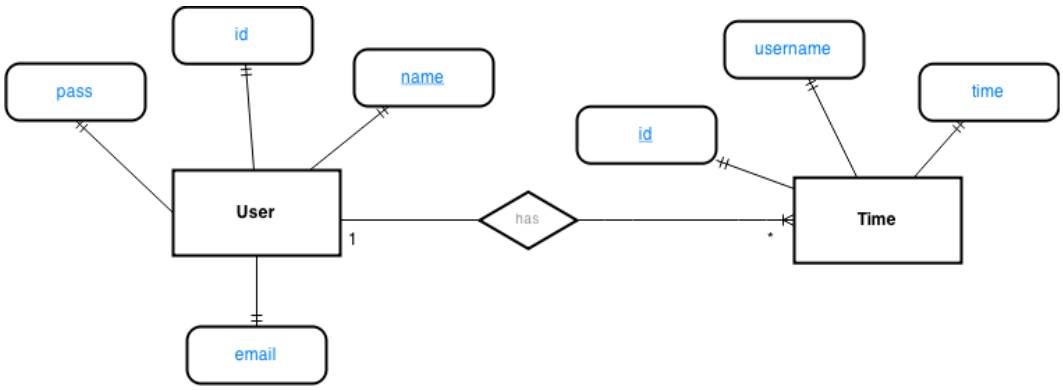


III. Data Access Layer:

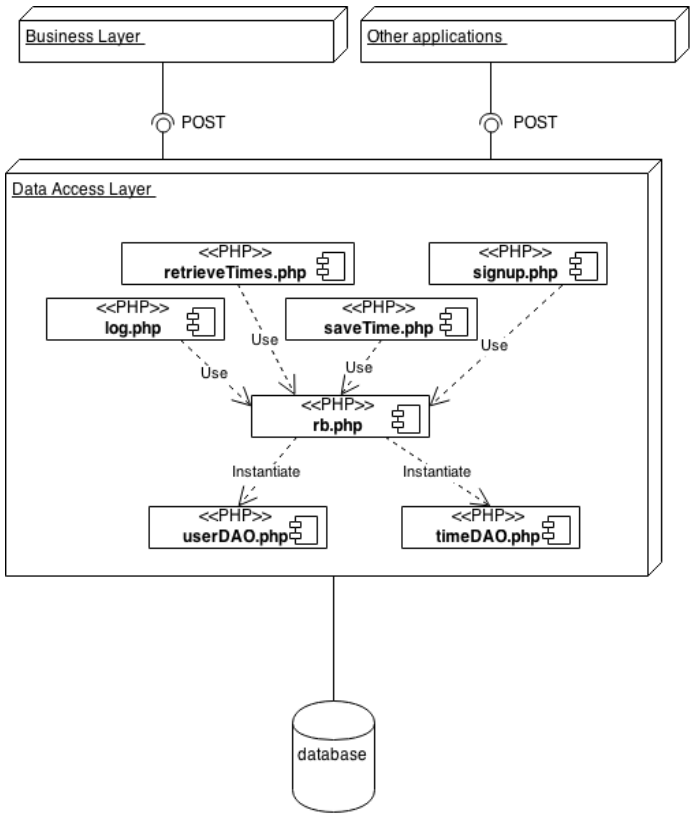
- Enhanced entity–relationship (EER) model diagram:



- Entity–relationship (ER) model diagram:



- Component diagram:



9. Data View

This view is highly detailed on the Data Access Layer of the previous section, so please review the 8.1 and 8.2 Data Access Layer sections of this document for a detailed description and explanation of the implementation of this view in this system.

10. Size and Performance

The chosen software architecture supports the key sizing and timing requirements, as stipulated below:

1. The system shall support up to 2 simultaneous users per instance of the game at given time.
2. Internet speed connection should be of at least 1 Mbit per second to achieve the optimal results and avoid network lag.
3. Client PC's graphics card should have DX9 (shader model 2.0) capabilities.
4. Client PC's CPU should have SSE2 instruction set support.
5. The game should run smoothly on every web browser, like IE, Chrome, Safari, and others.
6. The client portion shall require less than 20 MB disk space and 512 MB of RAM.

The selected architecture supports the sizing and timing requirements through the implementation of a client-server approach. The client portion is implemented on local user PCs. The components have been designed to ensure that minimal disk and memory requirements that are needed on the PC client portion.

11. Quality

The software architecture will support the quality requirements as stipulated bellow:

A. Compatibility:

I. The game system shall work in any of the versions of the following web browsers, or above: Internet Explorer v11, Safari v5.1.7, Mozilla Firefox v32.0 and Google Chrome v41.

B. Performance:

I. Under no circumstance the game system should have more than 170 milliseconds of network latency.

C. Availability:

I. The game system shall be available 24 hours a day, 7 days a week. There should be no more than 4% down time.

D. Interoperability:

I. The game system shall be able to interact with a SQL database system in order to store player's information, and also it must be able to interact with a SmartFoxServer service to manage and track the multi-user instances of the game.

E. Manageable:

I. The game system shall be able to provide an easy GUI to manage the session of the users and keep track of the system's resources being used at a given moment, mainly through the online utilities of a SmartFoxServer service.

F. Testability:

I. It is always critical to ensure that new development does not break other components of the software. To ensure this, the game system build scripts optionally should execute a set of unitary test cases.

G. Reliability:

I. Mean time between failures shall exceed 300 hours.

H. Scalability:

I. Upgrades to the PC client portion of the game system shall be downloadable from the servers over the internet. This feature enables users to have easy access to system upgrades.
II. The server performance shall be capable to grow or decrease accordingly to the users load and on demand.

I. Security:

I. The user password of the players should be encrypted into the database with an AES algorithm with a 256 bit key.
II. Logs should be recorded and maintained to keep track of user activities inside the game system.

J. Conceptual integrity:

I. The game mechanics should be consistent with the Dungeon Crawler game genre.

K. Flexibility:

I. The game system must permit an easy modification of the game mechanics by the game masters if it is needed.

L. Maintenance:

I. The entity and business classes of the core code of the system should be mainly closed to modification, but modularized enough to permit easy addition of new features in the system by extending that existing code.

M. Reusability:

I. All generalized classes and methods should be encapsulated enough to permit its reusability in another game of the same genre if it's needed in the future.

N. Portability:

I. The game must operate on full capacity regardless of the operating system of the client host, which could be Windows 7, OS X Yosemite, Ubuntu 14.04.2, or above.

O. Usability/User experience:

I. The game user-interface must be designed with minimalist patterns.
II. The user interface of the game must be designed for ease-of-use and should be appropriate for a computer-literate user community with no additional training on the System.
III. Additionally, each feature of the gameplay should have built-in online help for the user. Online Help must include step by step instructions on using the System.