Labyrinth

Software Architecture Document

Version 1.1

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. |
|  |  |  |  |
|  |  |  |  |

Table of Contents

1. Introduction 4

1.1 Purpose 4

1.2 Scope 4

1.3 Definitions, Acronyms, and Abbreviations 4

1.4 References 5

1.5 Overview 5

2. Architectural Representation 6

3. Architectural Goals and Constraints 6

4. Use-Case View 6

5. Logical View 8

5.1 Overview 8

5.2 Architecturally Significant Design Packages 8

5.3 Use-Case Realizations 11

6. Process View 12

7. Deployment View 14

8. Implementation View 14

8.1 Overview 15

8.2 Layers 15

9. Data View (optional) 15

10. Size and Performance 15

11. Quality 15

Software Architecture Document

# 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 use cases. Thus, this framework allows the development of the design criteria and documents that define the technical and domain standards in detail of this project.

## 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.

## 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.

## 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, 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. |

## References

NA

## 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 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, 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. This section is optional if there is little or no persistent data, or the translation between the Design Model and the Data Model is trivial.

* 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.

# Architectural Representation

TBD

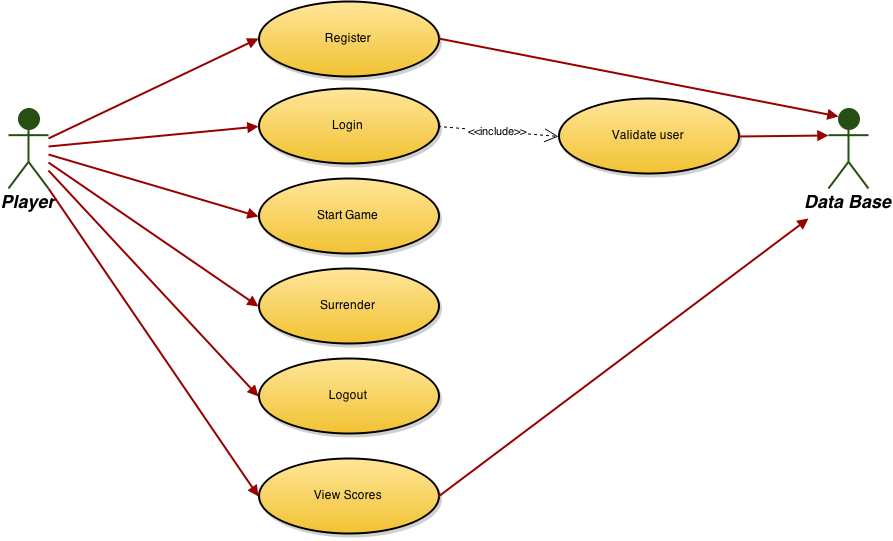
[This section describes what software architecture is for the current system, and how it is represented. Of the **Use-Case**, **Logical**, **Process**, **Deployment**, and **Implementation Views**, it enumerates the views that are necessary, and for each view, explains what types of model elements it contains.]

# 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 one every web-browser.
    8. Users should be able to see their score history.
    9. All remote accesses are subject to user identification and password control.
    10. The architecture should be flexible and extensible, 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

# Use-Case View



* Register:

Brief Description  
This use case allows a Player to sing up into the system if it’s the first time that he enter to the game, or if he just want 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 fist.

* Surrender:

Brief Description  
This use case allows a Player to terminate the game if he don’t want to play anymore and yet there is no winner. In this case the player requesting the surrender will be declared the looser of the game. 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

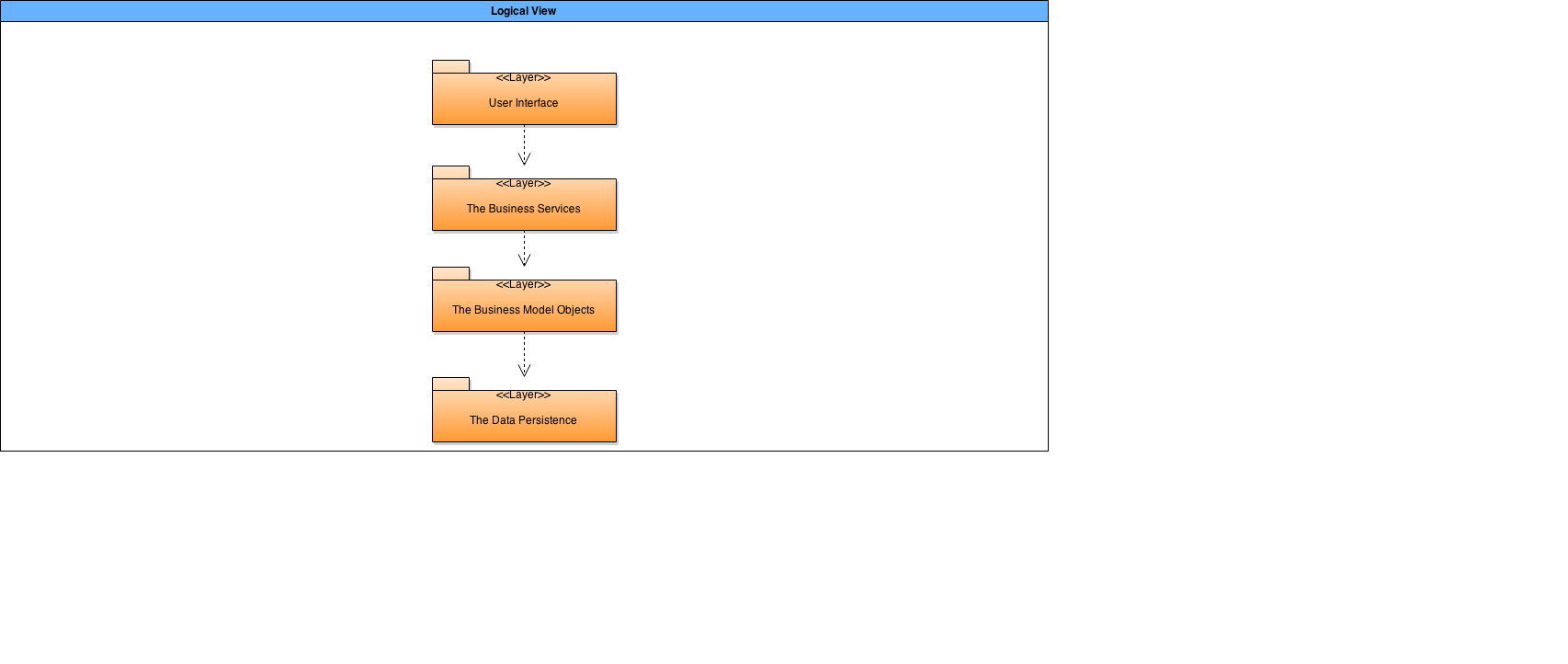
* View Scores:

Brief Description  
This use case allows a Player to view his/her report, or another player report, of previous game parties played. The Player is the actor of this use case. The Data Base System is an actor involved within this use case.

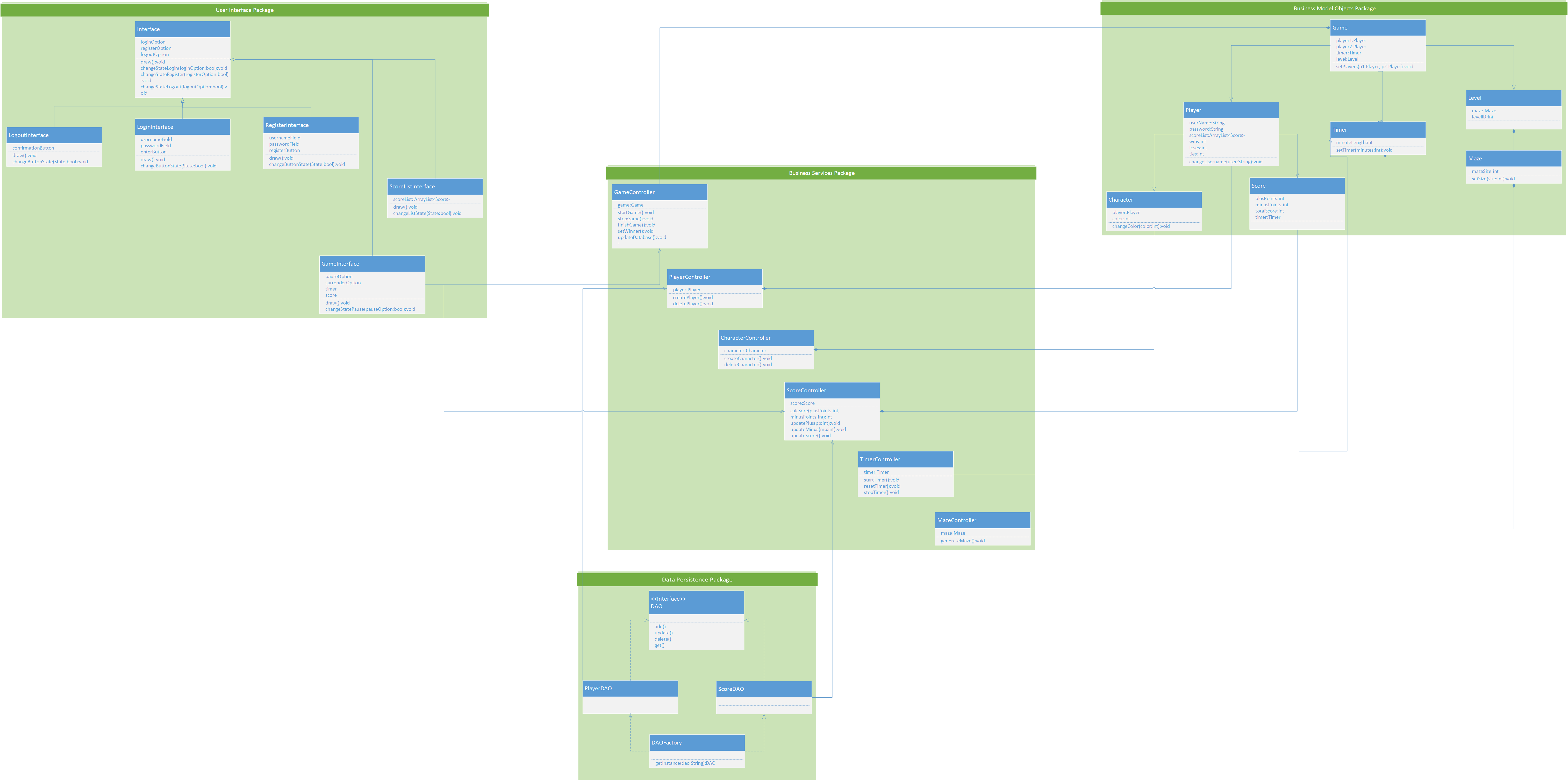
# Logical View

The logical view of the Labyrinth game system is comprised of 4 main packages describing a layered architecture: User Interface, Business Services, Business Model Objects, and Data Persistence.

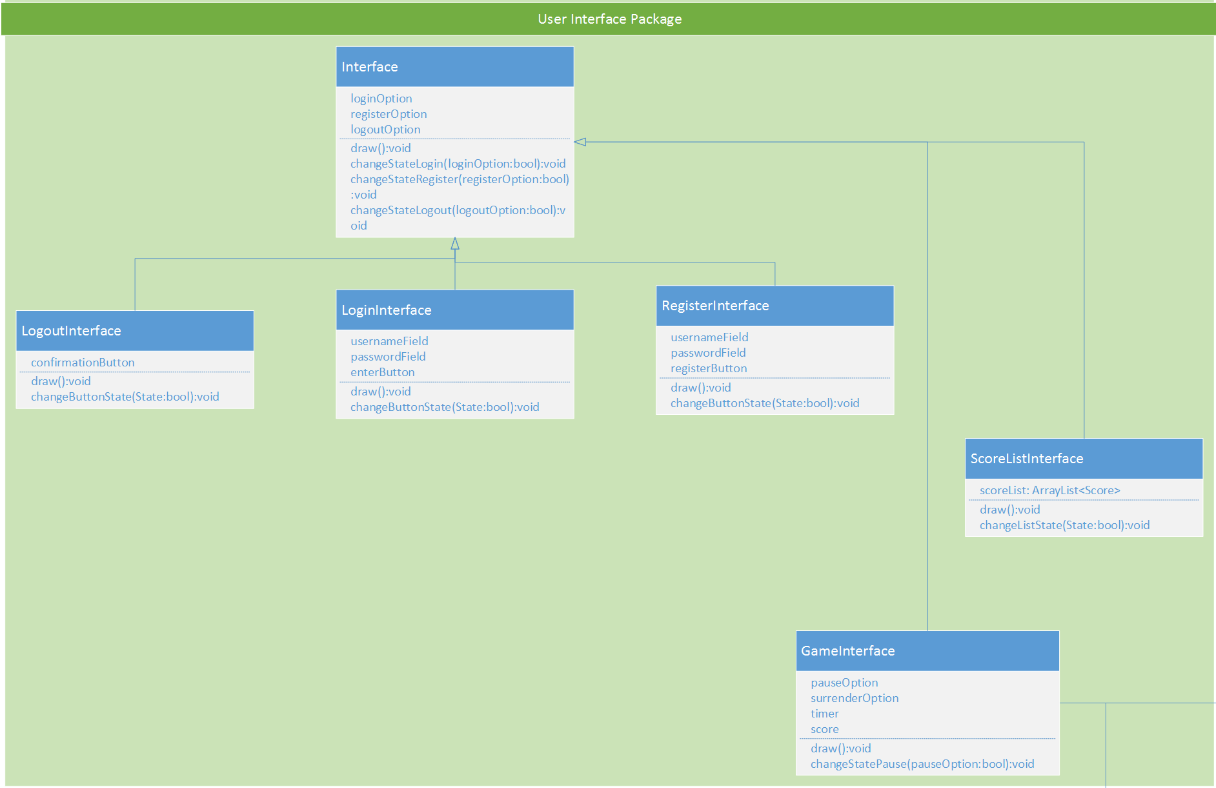
## Overview



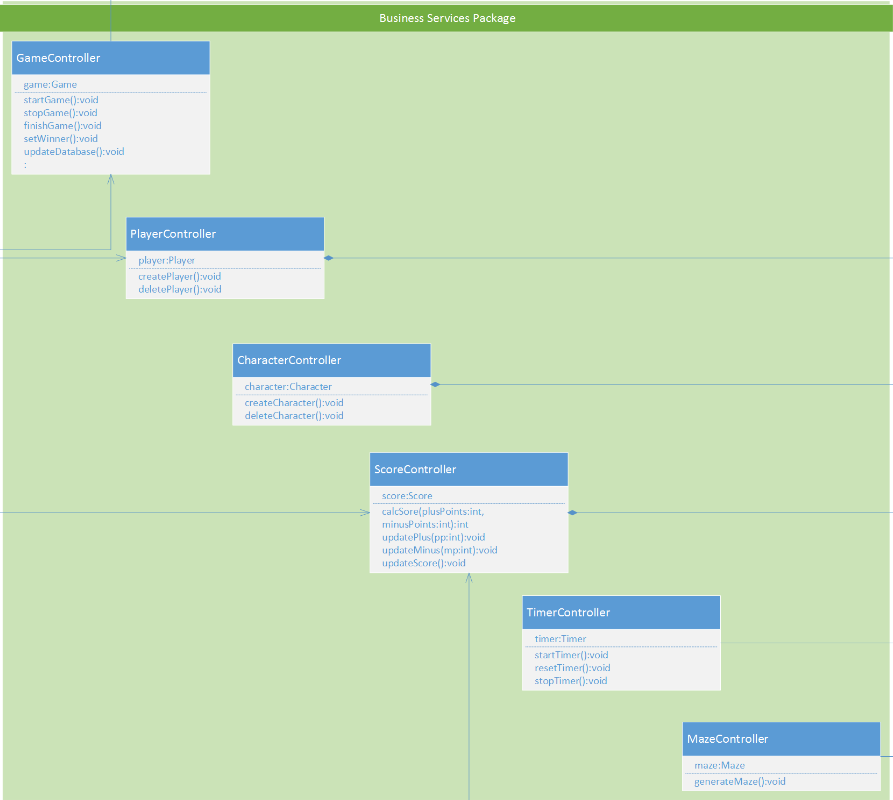
## Architecturally Significant Design Packages



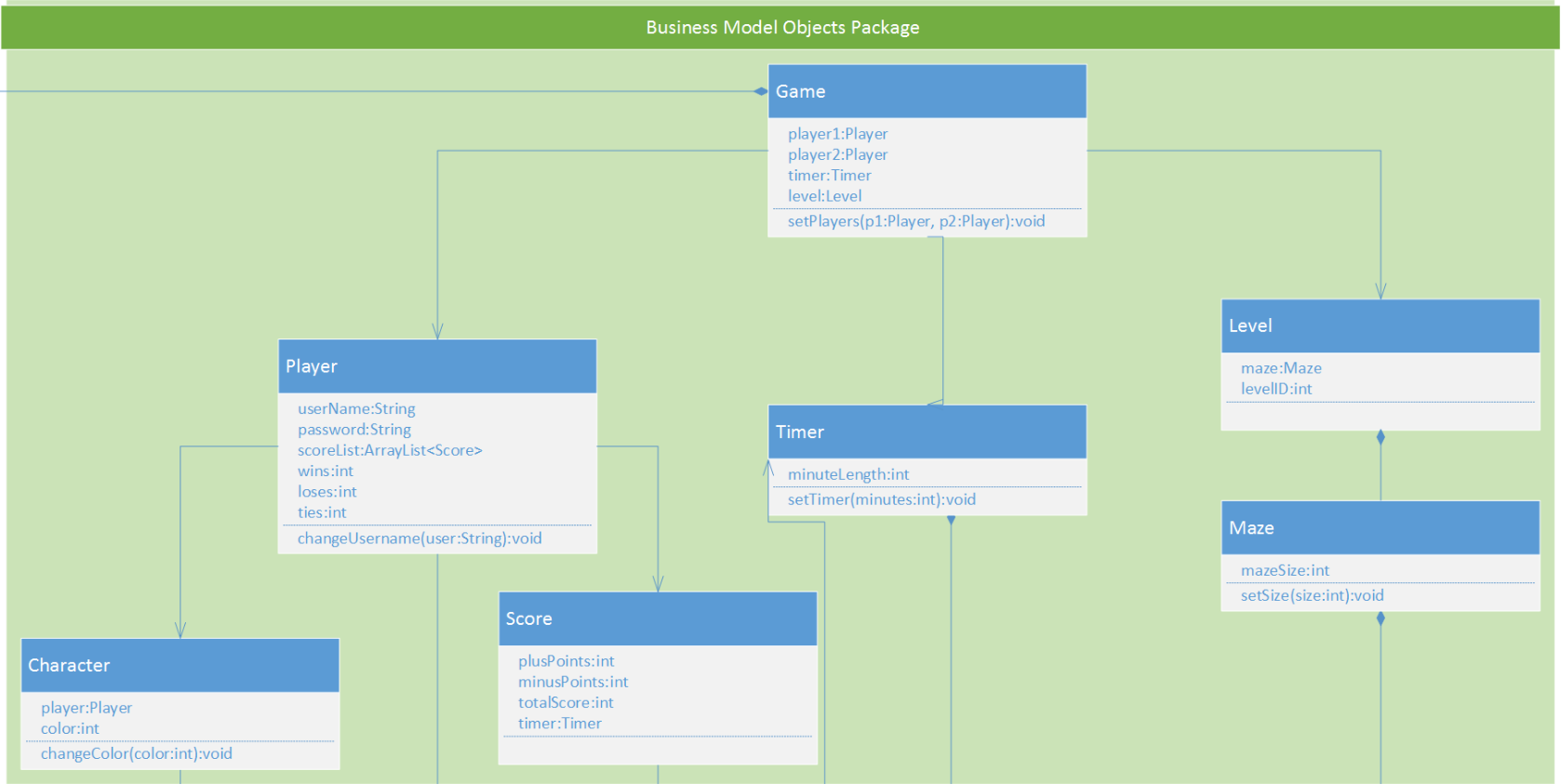
* The User Interface Package contains classes for each of the forms that the actors use to communicate with the System. Boundary classes exist to support login, logout, registration, gameplay, and review of the scores.



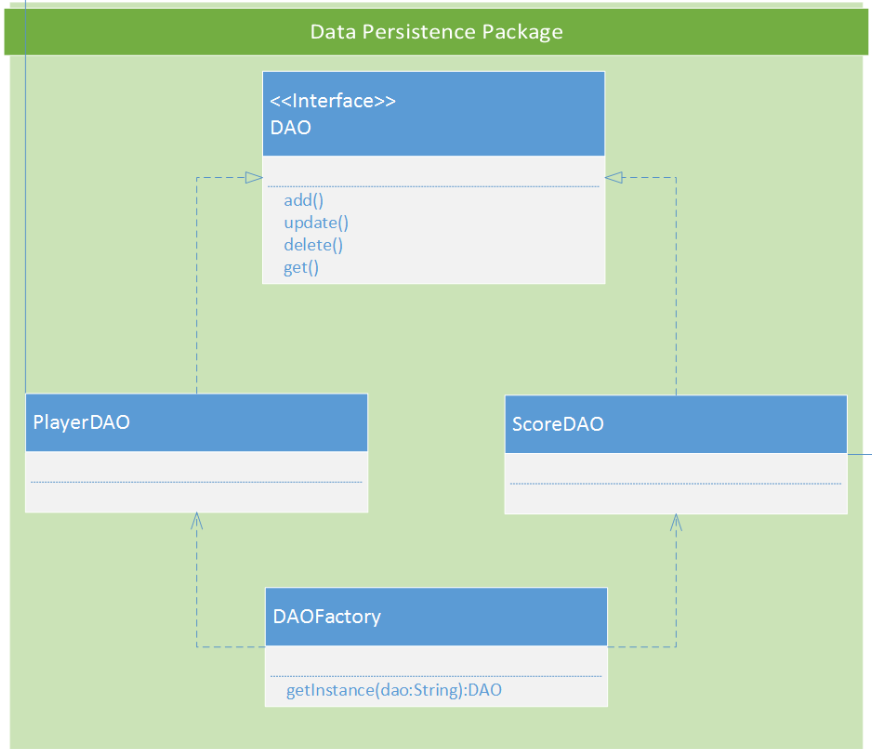
* The Business Services Package contains control classes for interfacing with the Business Model Objects, controlling player registration, and managing the game control like game mechanics, game behavior, results and scores.



* The Business Model Objects Package includes entity classes for the Labyrinth game (i.e. player, character, timer, score, level, maze, etc.) and Boundary classes also exist to support interaction with the database system.



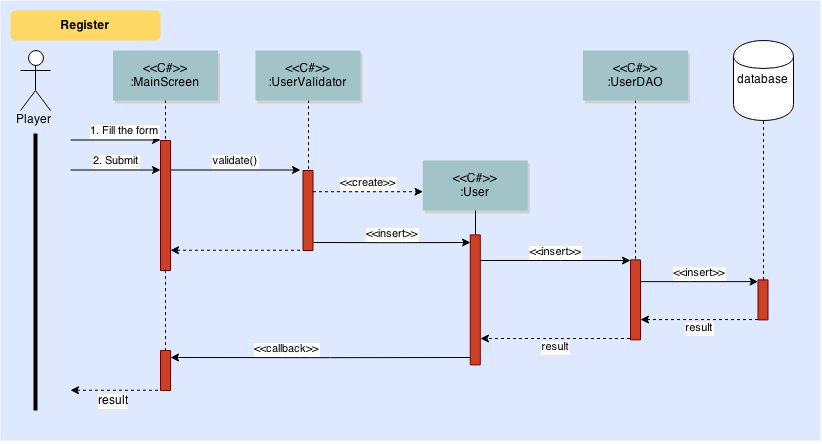
* The Data Persistence Package includes classes for persistence access implementation: including object – relational mapping, data caching, query, schema generation, and transactional object life cycle support.



## 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.

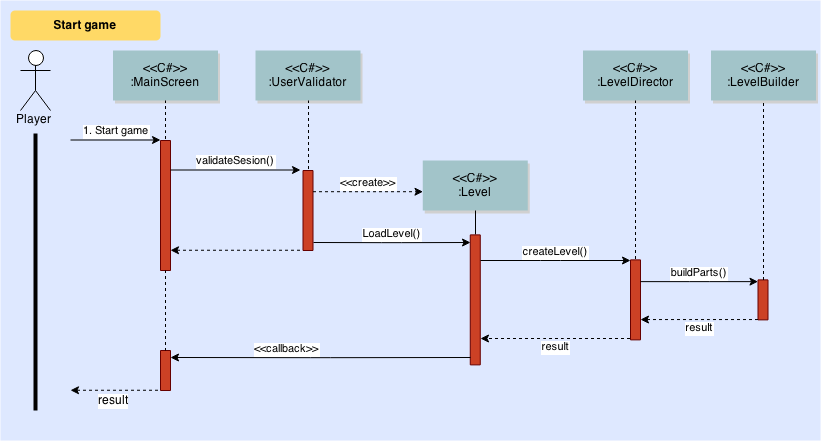
* Register use case.



### Participating objects

|  |  |  |
| --- | --- | --- |
| **Object** | **Class** | **Description** |
| **:MainScreen** | Display | Manage all GUI elements and user interactions. |
| **:UserValidator** | ValidatorController | 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

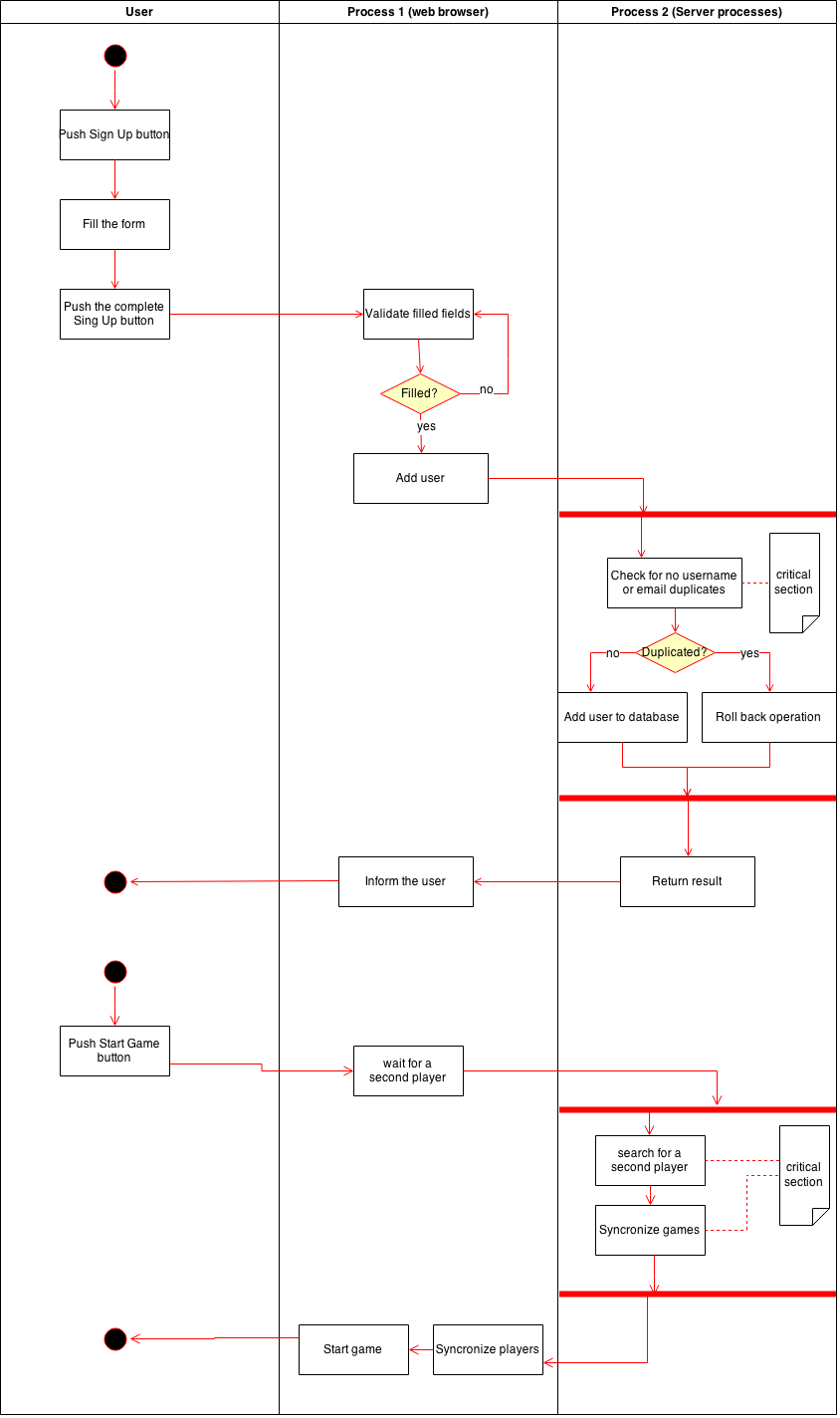
|  |  |  |
| --- | --- | --- |
| **Object** | **Class** | **Description** |
| **:MainScreen** | Display | Manage all GUI elements and user interactions. |
| **:UserValidator** | ValidatorController | Acts as a liaison between the GUI components and the Level Entity Class. |
| **:User** | Level | Represents an entity model class in the system for a Level. |
| :**LevelDirector** | LevelDirector | Serves as a momentary storage of the information of a new level. |
| :**LevelBuilder** | LevelBuilder | In charge of the creation of a new level and all of its components |

# 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 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:

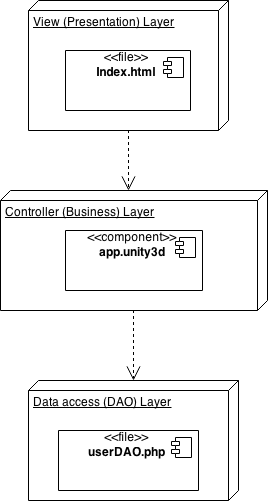


# Deployment View

TBD

[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. At a minimum for each configuration it should indicate the physical nodes (computers, CPUs) that execute the software and their interconnections (bus, LAN, point-to-point, and so on.) Also include a mapping of the processes of the **Process View** onto the physical nodes.]

# 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 file 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 petition with a web browser.
* Business Layer: This layer will group all the components that will handle all the mechanics and control of the game. The main file of this layer will be the app.unity3d executable file, which will group all the core c# classes 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.

## Overview

TBD

[This subsection names and defines the various layers and their contents, the rules that govern the inclusion to a given layer, and the boundaries between layers. Include a component diagram that shows the relations between layers. ]

## Layers

TBD

[For each layer, include a subsection with its name, an enumeration of the subsystems located in the layer, and a component diagram.]

# Data View (optional)

TBD

[A description of the persistent data storage perspective of the system. This section is optional if there is little or no persistent data, or the translation between the Design Model and the Data Model is trivial.]

# 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.

# Quality

TBD

[A description of how the software architecture contributes to all capabilities (other than functionality) of the system: extensibility, reliability, portability, and so on. If these characteristics have special significance, such as safety, security or privacy implications, they must be clearly delineated.]