FlashScore

Analysis and Design Document

Dura Dragos Catalin

**Group:30233**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <22/04/2019> | <1.0> | Iteration 1.1 | Dura Catalin |
| <02/05/2019> | <2.0> | Iteretion 1.2 | Dura Catalin |
| <25/05/2019 | <2.0> | Iteration 2, Construction and Transition | Dura Catalin |
|  |  |  |  |

Table of Contents

I. Project Specification 4

II. Elaboration – Iteration 1.1 4

1. Domain Model 4

2. Architectural Design 4

2.1 Conceptual Architecture 4

2.2 Package Design 4

2.3 Component and Deployment Diagrams 4

III. Elaboration – Iteration 1.2 4

1. Design Model 4

1.1 Dynamic Behavior 4

1.2 Class Design 4

2. Data Model 4

3. Unit Testing 4

IV. Elaboration – Iteration 2 4

1. Architectural Design Refinement 4

2. Design Model Refinement 4

V. Construction and Transition 5

1. System Testing 5

2. Future improvements 5

VI. Bibliography 5

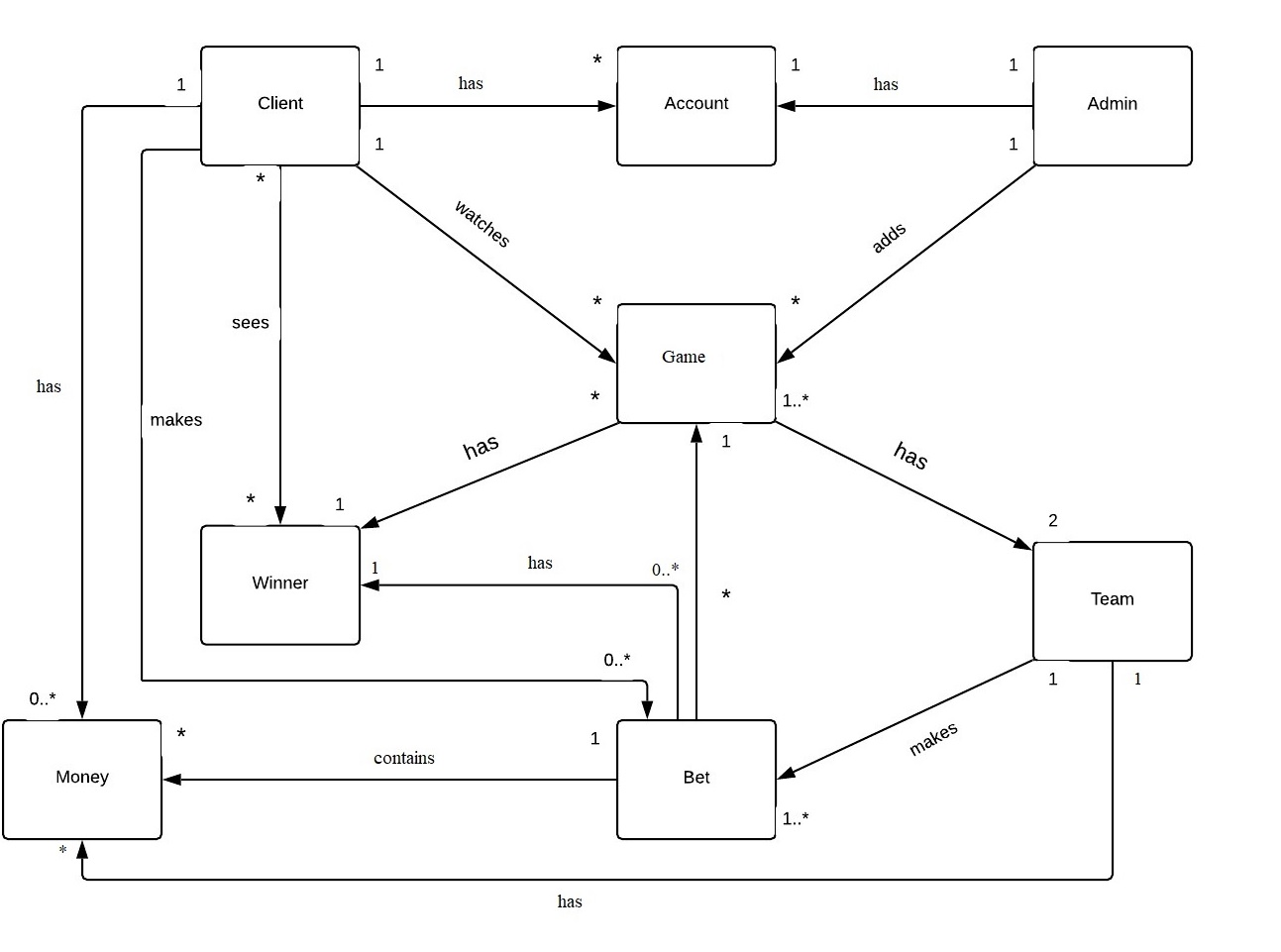
# Project Specification

The application is an online platform that allows user to see matches from different categories. The main purpose of the application is to allow people to see the results of matches and the matches that are about to start, from many categories of sports. The users can also place bets on the matches and to win money based on the matches results . The application will be split in 2 actual programs. One that program that is based on Spring Boot and fallows MVC architecture and Layered Architecture that will implement the view and the operations that can be done by the client and admin.

The second application fallows layered architecture and this application will implement the system that generates the scores of the games in a random way and also will process all the bets that were made on the games by clients.

# Elaboration – Iteration 1.1

# Domain Model



# Architectural Design

## Conceptual Architecture

*[Define the system’s conceptual architecture; use an architectural style and pattern - highlight its use and motivate your choice.]*

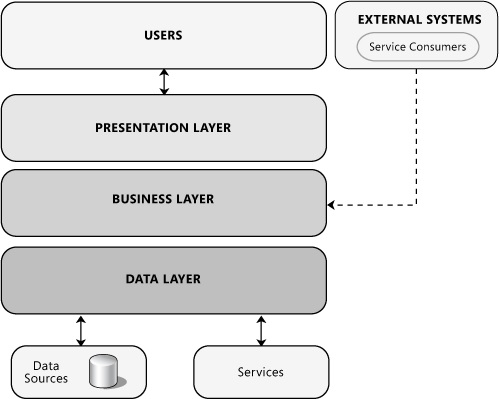
Since it will be a web application, this project will work on an MVC architecture.

**Model** - Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.

**View** - View represents the visualization of the data that model contains.

**Controller** - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

The application also fallows a Layered Arhitectural Pattern, one of the powerful features of the layered architecture pattern is the separation of concerns among components.



Data layer: In this layer we have the operations that are made on the database and also contains POJO Classes.

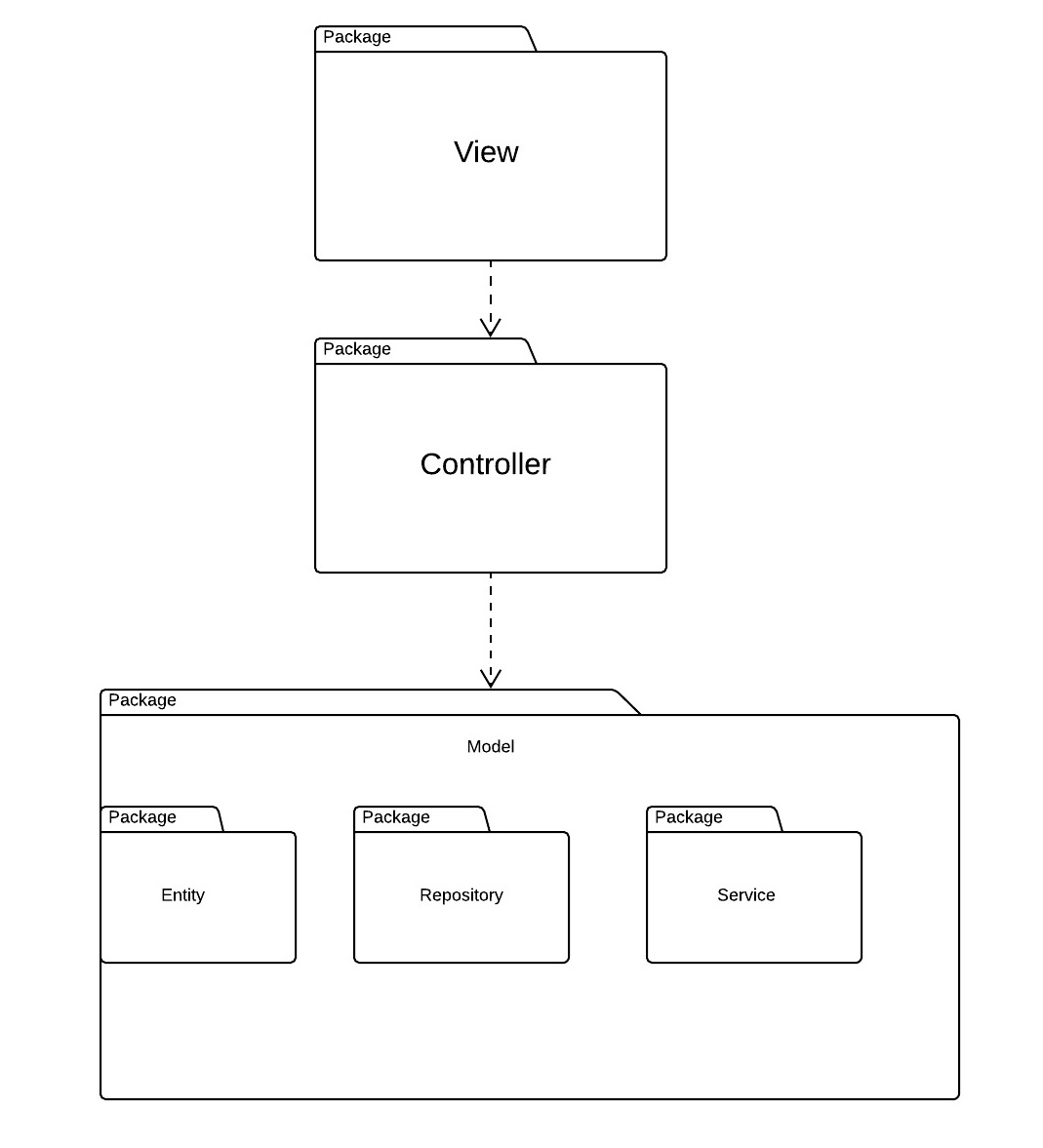
Business layer: In this layer we have the core functionality of the application

Presentation layer: In this layer we have UI and also the controllers of the application which use the business functionality to perform the tasks that the user requested from the UI

In a layered architecture application a component from a layer knows only about the components that are in the layer below the current layer. For the example if we have a component in Business Layers that component can use some component from Data Layer, but can not know about components from Presentation Layer.

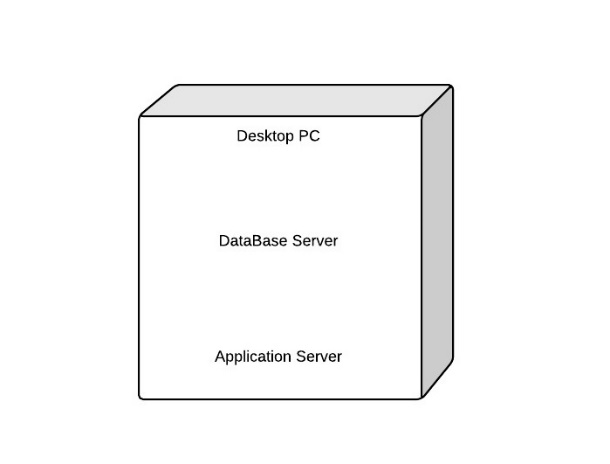
## Package Design

Package Diagram:



## Component and Deployment Diagrams

Deployment Diagram

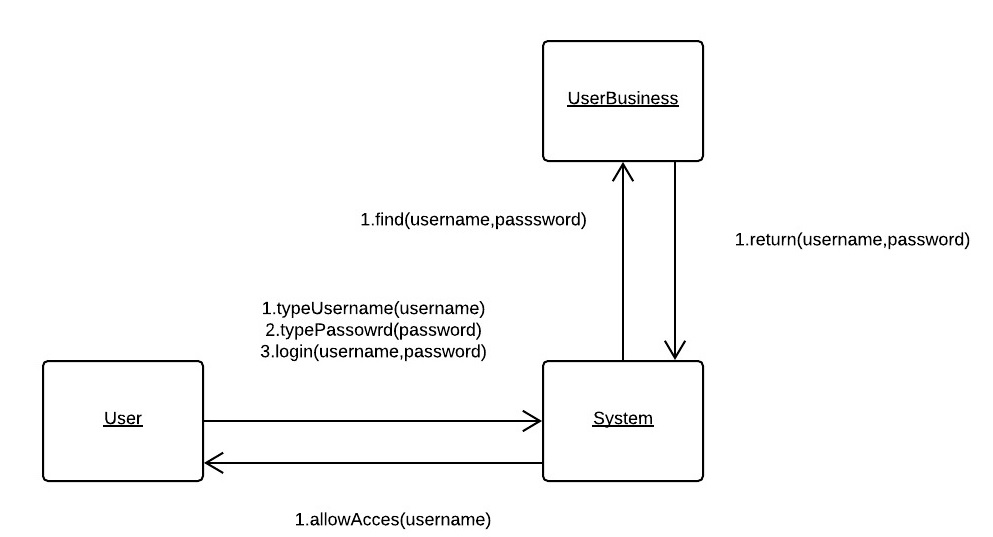


# Elaboration – Iteration 1.2

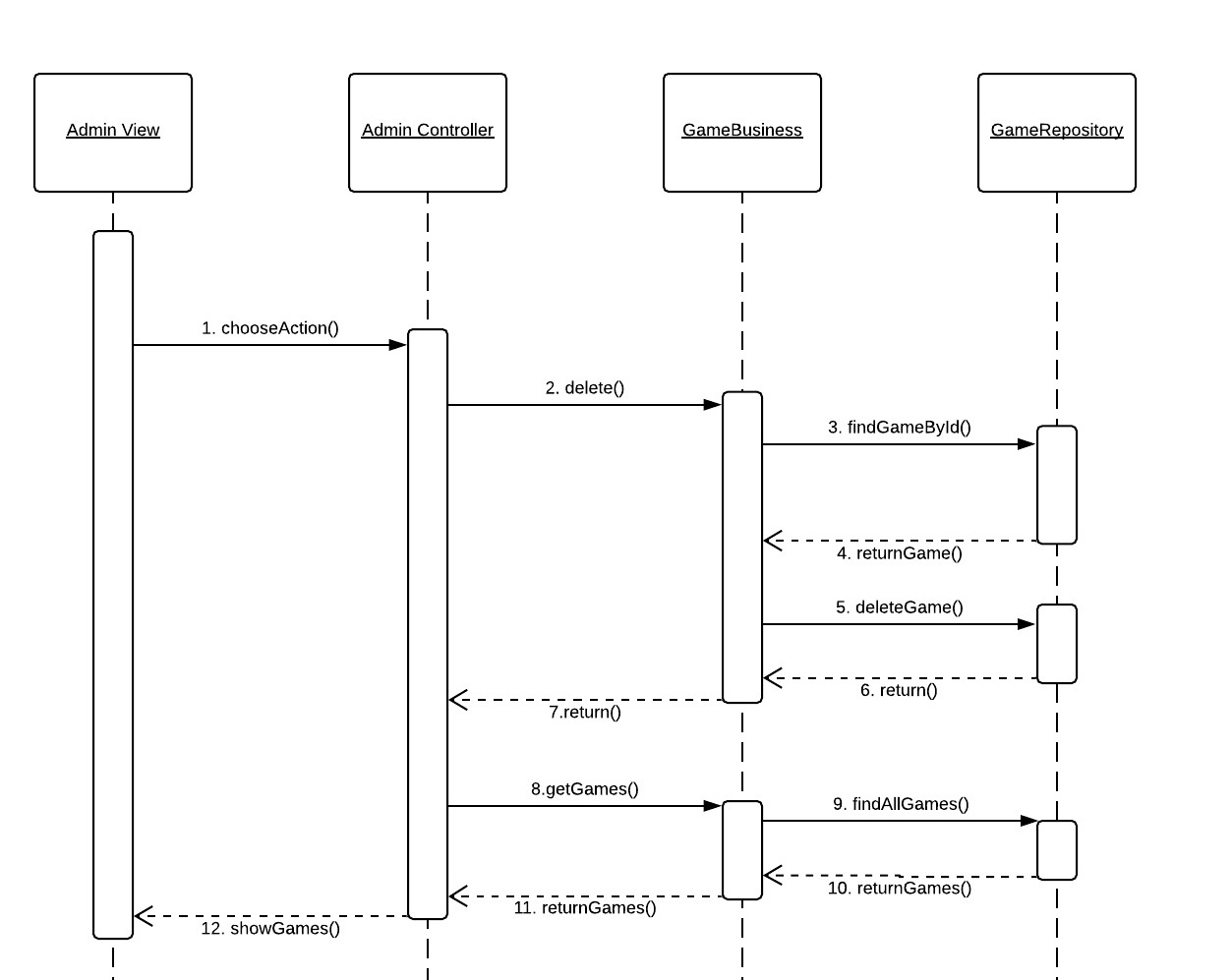
# Design Model

## Dynamic Behavior

Communication diagram: login client

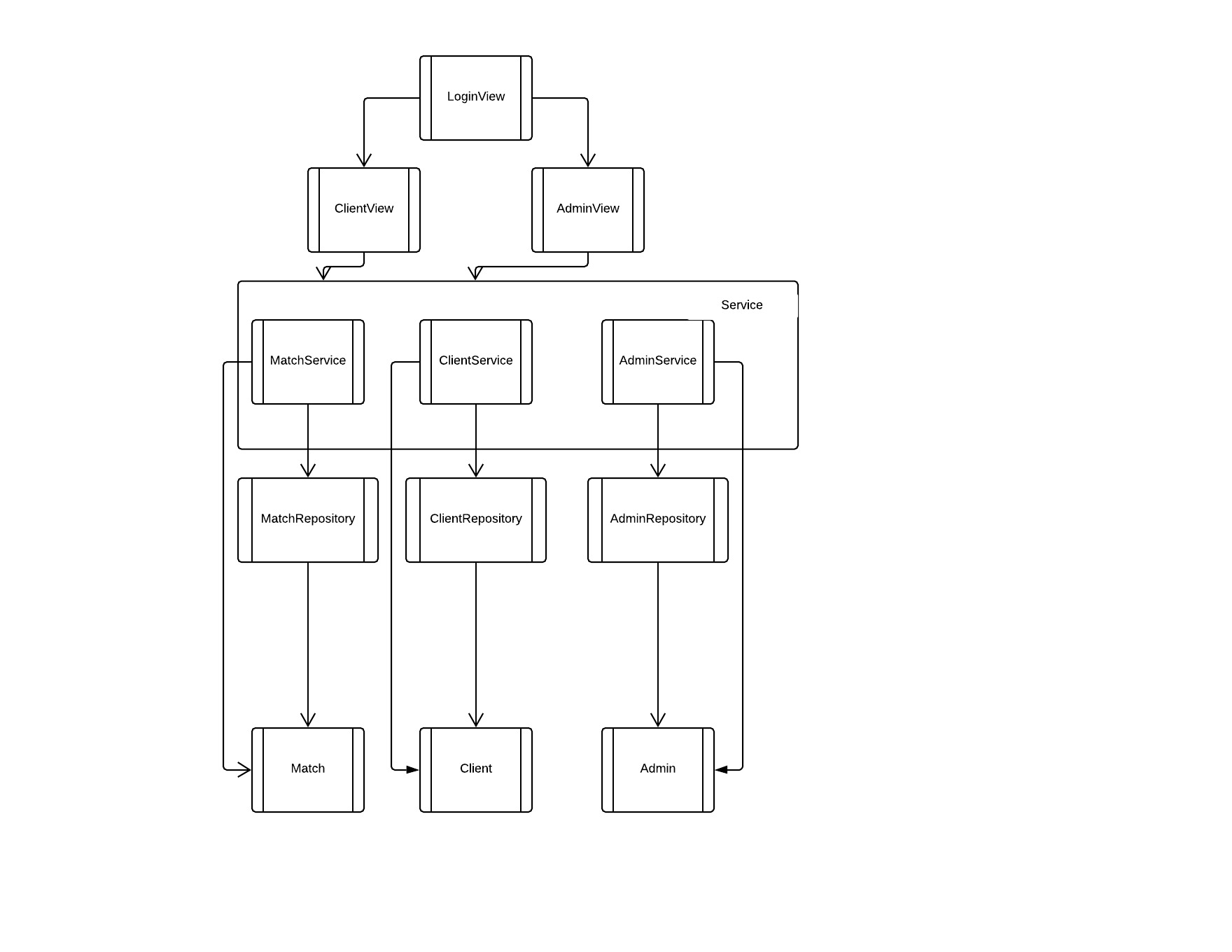


Sequence diagram: delete match by admin



## Class Design

*[Create the UML class diagram; apply GoF patterns and motivate your choice]*

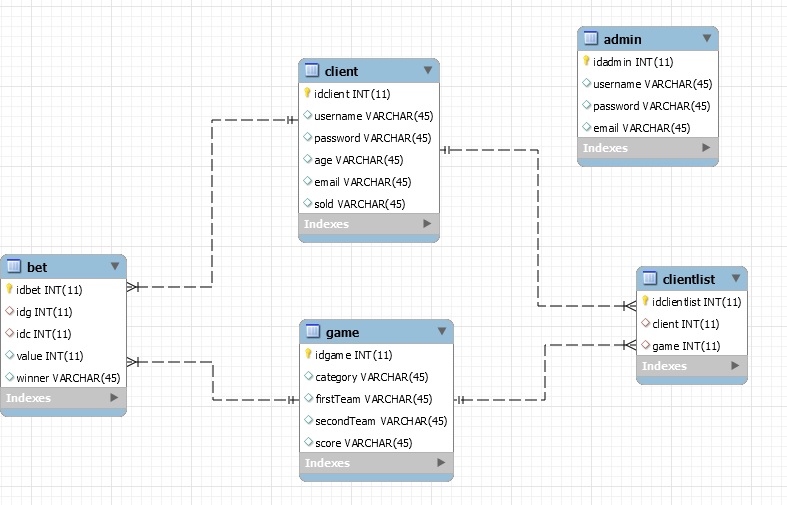
**

Class Diagram:

# Data Model

*[Create the data model for the system.]*

The DataBase that will be used for this project is MySql DataBase which is a relational DataBase.



# Unit Testing

*[Present the used testing methods and the associated test case scenarios.]*

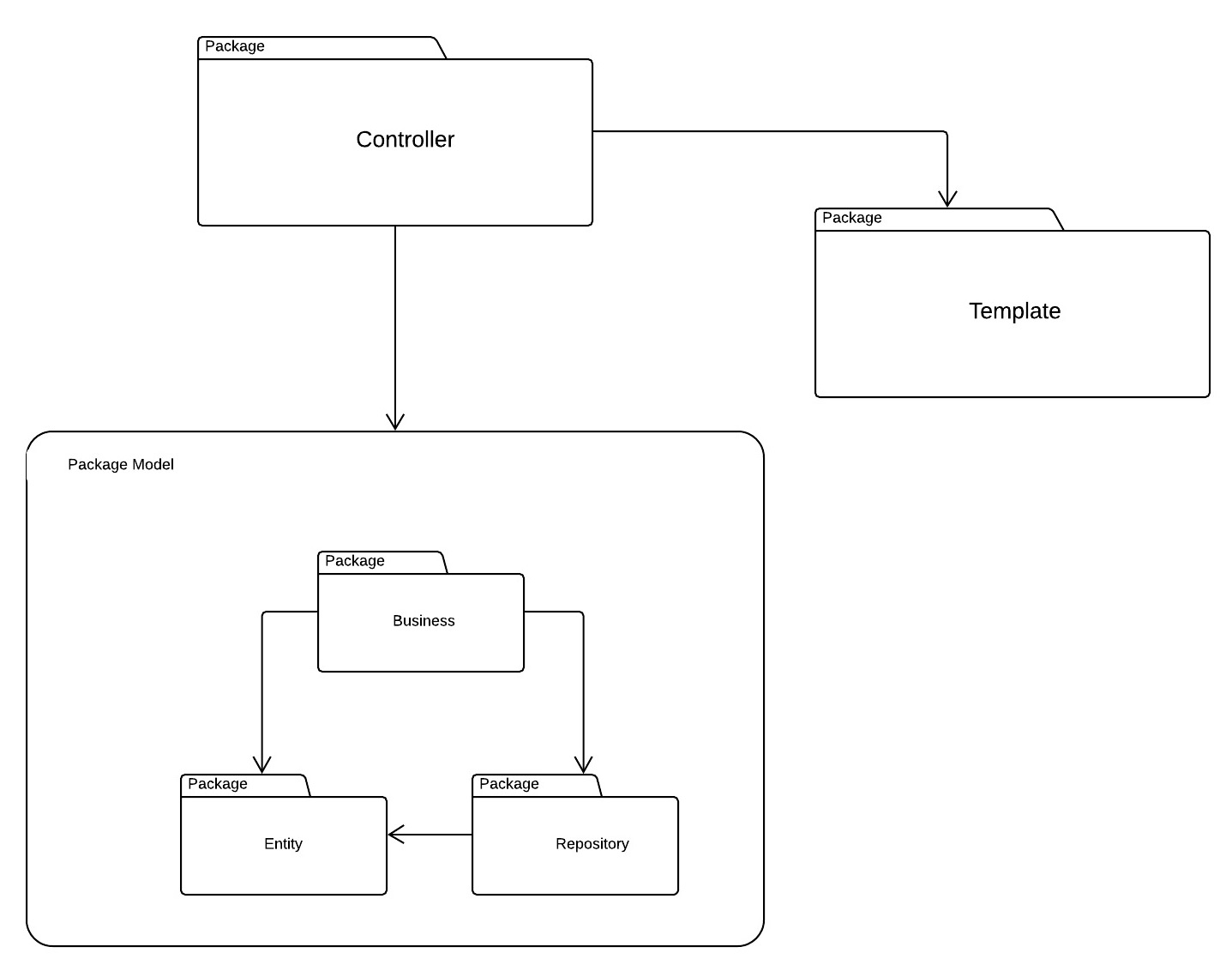
For the testing I am going to use Junit4 where I will test scenarios such as: login as client or admin, CRUD operations on matches and adding matches by clients into their favourite list.

# Elaboration – Iteration 2

# Architectural Design Refinement

*[Refine the architectural design: conceptual architecture, package design (consider package design principles), component and deployment diagrams. Motivate the changes that have been made.]*

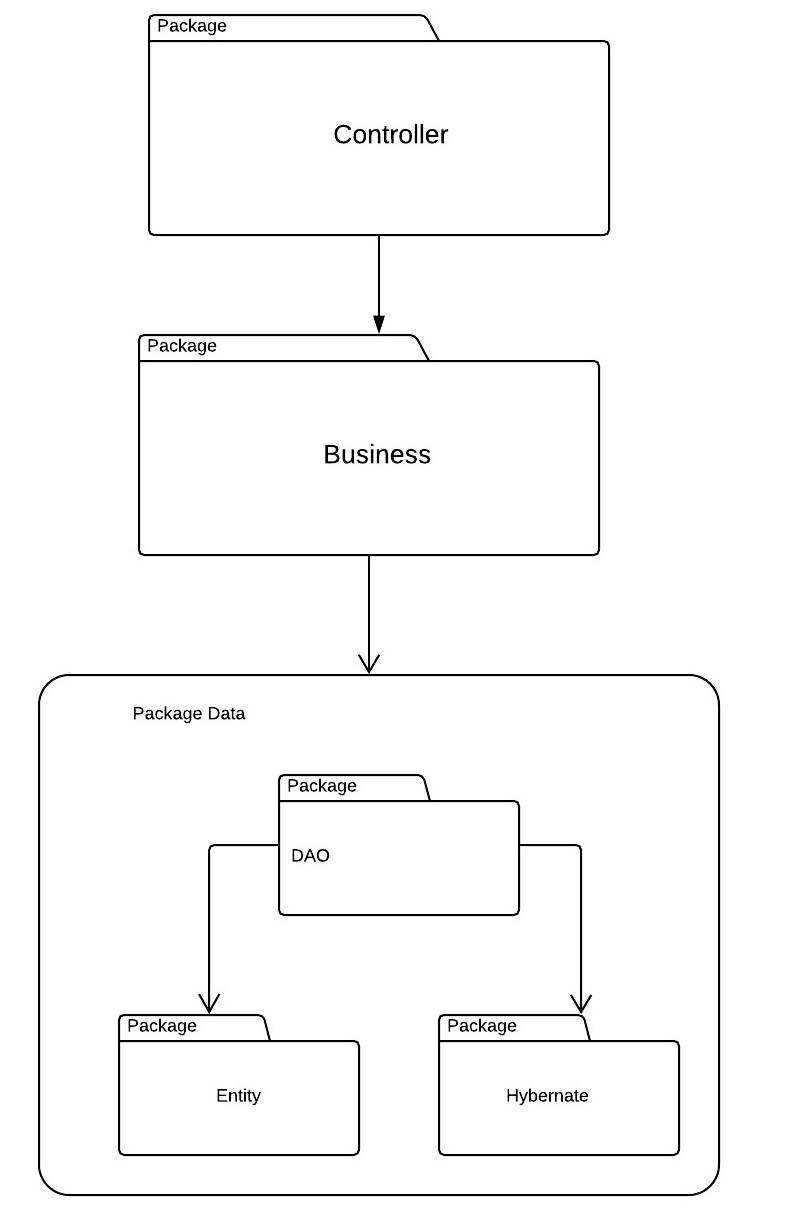
Package Diagram for the Spring Boot application:



In this class package diagram I modified the way that package Model looks, the layered architecture inside this package being now easier to be seen.

I also modified the view, because the view is not using the controller as I represented it in the previous diagram, but the controller is using the view, in my case the view being represented by the templates that I created in package Template.

Class Diagram for the application that generated results for all unfinished games:

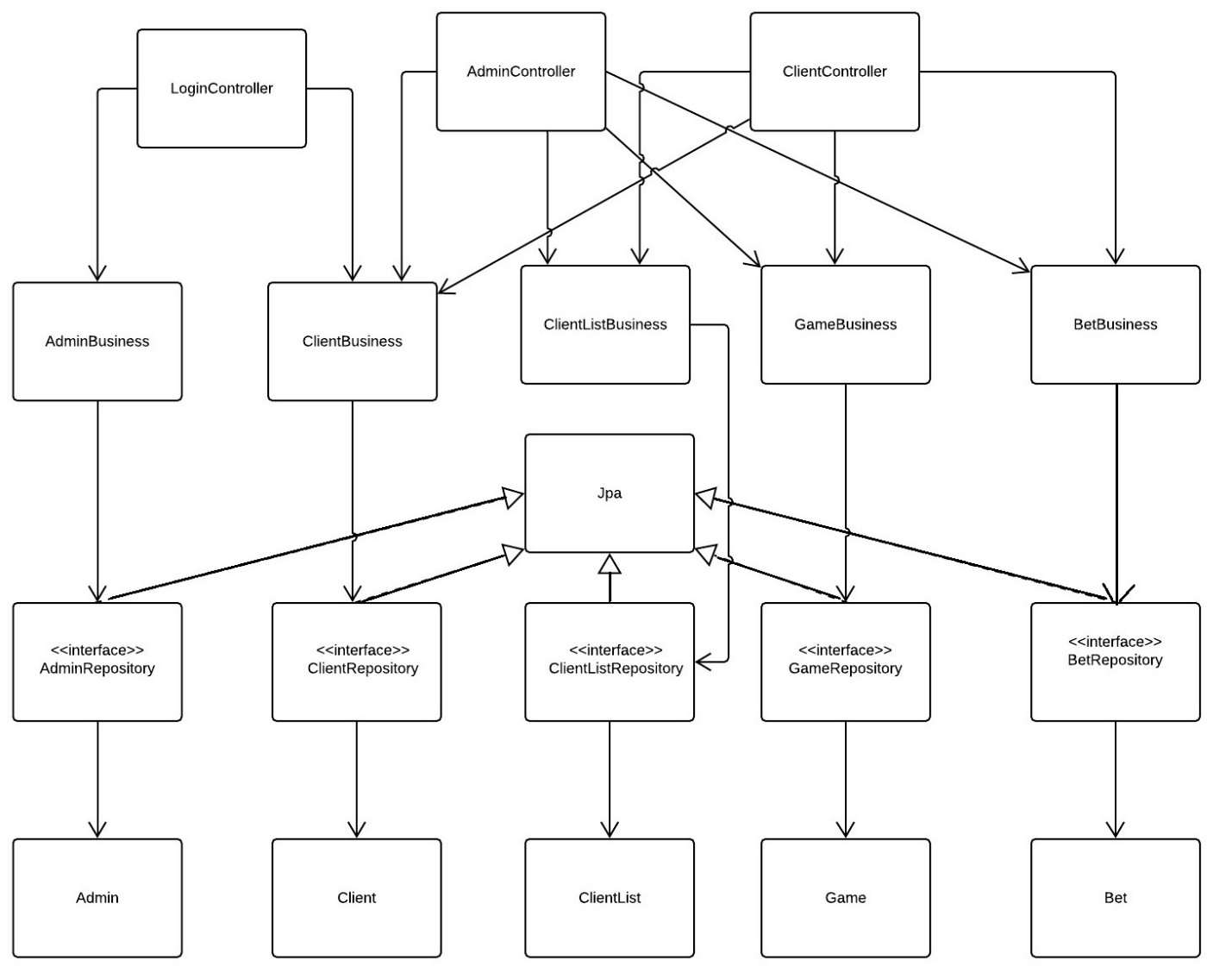


In the previous representation of the Package diagram I did not know that I would implement an application to auto-generate games results so I did not need a Package diagram for it. As it can be seen from the Diagram this second application also fallows Layered Architecture, but because it is not a Spring Boot application I used directly Hybernate instead of Jpa, so instead of package repository I have package DAO where all the queries that the application needs are implemented, being used later in the Business package.

This application being used only for the game result generation I do not need to implement a UI for the application because it will only be started and from there it will generate the results on his own.

# Design Model Refinement

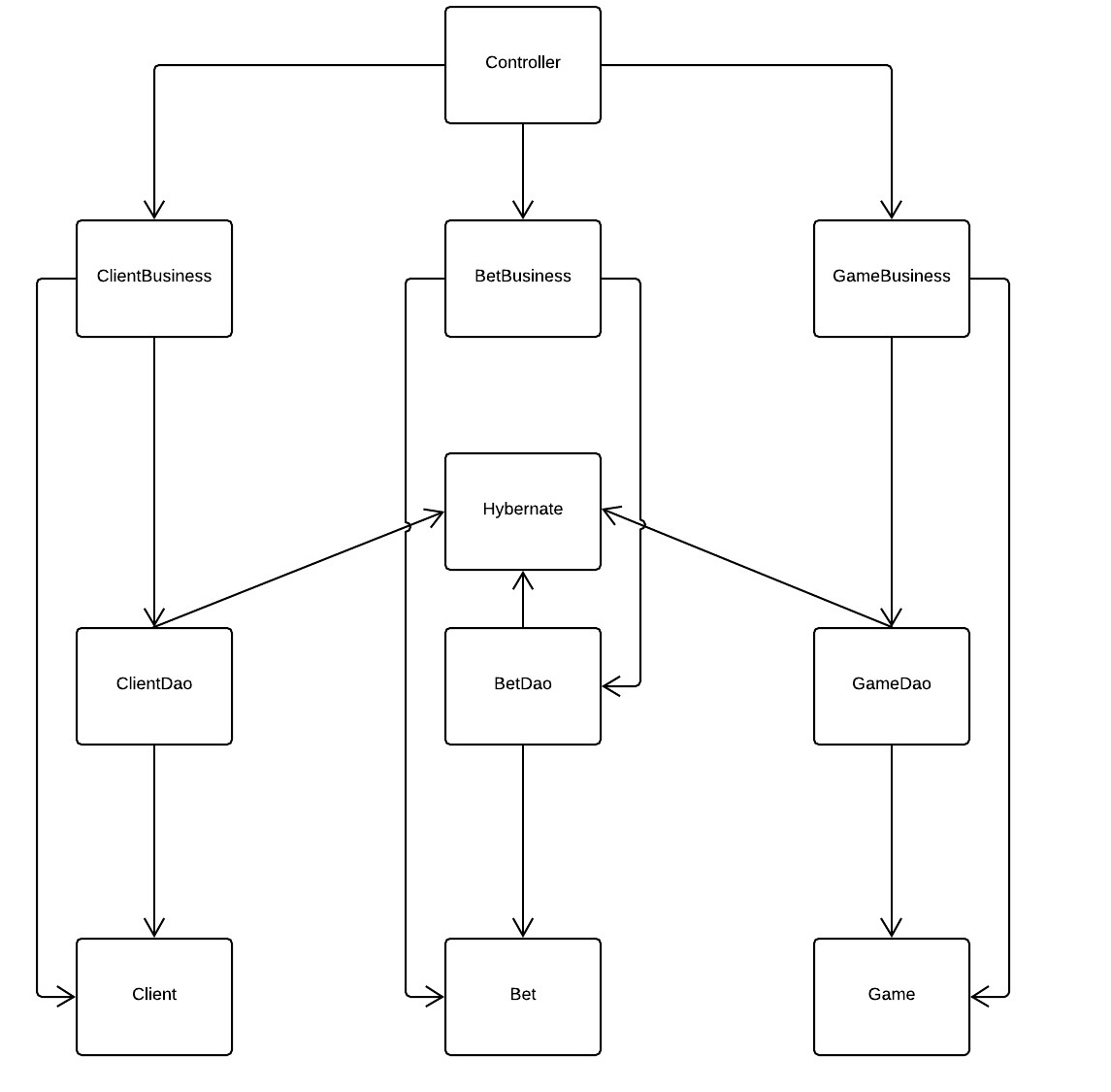
## *[Refine the UML class diagram by applying class design principles and GRASP; motivate your choices. Deliver the updated class diagrams.]*



This class diagram represents the class diagram of the main application which is implemented using Spring Boot. I implemented the classes in a way that they respect Layered Architecture Pattern, as it can be seen from the class diagram only the Business Layer of the application can communicate with the Entities and Repositories. The Presentation Layer which here is represented by the controllers can only use the Business Layer using dependency injection.

The difference from the previous version of the diagram class is that this new diagram class has more classes which were needed for the implementation of the core functionality of the application and maybe in the start when I implemented the first class diagram were not so clear. Another improvement from the previous version is adding the Jpa interface as a class and representing the interfaces that extend that interface, more exactly all the repositories of the application.

Class Diagram for the application that generated results for all unfinished games:



The class diagram for the second application is a little bit different from the first one, because the application that generated the results for the games is not implemented using Spring Boot, I implemented it using Hybernate to execute queries on the same database that the Spring Boot application is using. Because I only need to generate results for the matches and process the bets that were made I only needed the Game class which gives me the games that are not finished, the Bet class which give me the bets that were made and the Client class so the clients that placed bets could win or lose the amount they betted on a game.

This application fallows layered architecture as it can be seen from the diagram class of the application, but instead of using Jpa I used directly hibernate to execute all the queries that I needed to implement the functionality that was required.

# Construction and Transition

# System Testing

*[Describe how you applied integration testing and present the associated test case scenarios.]*

For the testing part of the application I used Junit4 where I tested the operations on games and also operations that can be made by client and admin. I tested the login operations for both admin and client, and the operations to add and delete games from client favourite list. Also I tested operations to add bets.

The testing was done in the java file that spring boot generated, where all the services that were involved in the testing were used using Injection, instead of declaring every object with the new keyword.

# Future improvements

Future improvements that could be done for the application would be implementing a system that generates match score in real time using threads. In this way you would have the start time of the game attached to every game, and after some period of time the match would start and the system would generate the score in a random way during this period of time. When the match would be over, the bets that were made on that match would be processed and all the clients that placed a bet on that game would be announced by the result of the bet using Pattern Observer.

# Bibliography