Managementul Ligii de Fotbal “Premier League”

Analysis and Design Document

Student: Victor-Petru Almasan

**Group: 30235**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 13/04/18 | 1.0 | <details> | Victor-Petru Almasan |
| 11/05/18 | 2.0 | <details> | Victor-Petru Almasan |
| 01/06/18 | 3.0 | <details> | Victor-Petru Almasan |
|  |  |  |  |

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 6

2.3 Component and Deployment Diagrams 6

III. Elaboration – Iteration 1.2 7

1. Design Model 7

1.1 Dynamic Behavior 7

1.2 Class Design 8

2. Data Model 9

IV. Elaboration – Iteration 2 9

1. Architectural Design Refinement 9

2. Design Model Refinement 10

V. Construction and Transition 12

1. System Testing 12

2. Future improvements 12

VI. Bibliography 14

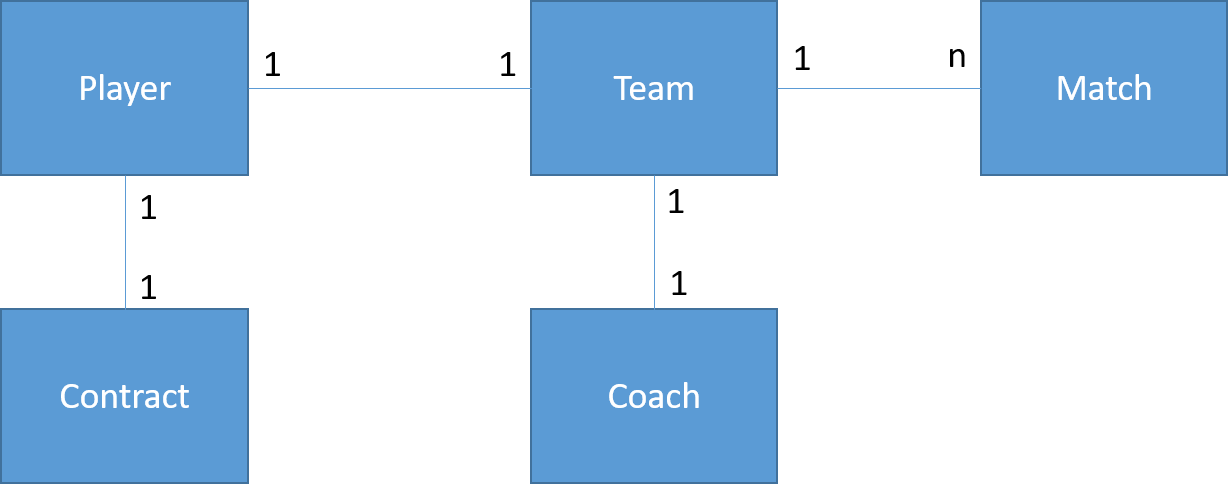
# Project Specification

Tema proiectului de fata este realizarea unei aplicatii pentru facilitarea managementului unei ligi de fotbal. Dat fiind ca ligile de fotbal sunt organizate dupa legile UEFA dar si dupa regulile din tara de origine, am decis realizarea managementului ligii „Premier League”. Aceasta este o competitie profesionista pentru cluburi de fotbal, localizata in primul esalon al sistemului de ligi ale fotbalului englez. Este cea mai profitabila liga in ierarhia UEFA. Pe baza performantelor din cupele europene din ultimii 5 ani, „Premier League” este situata pe prima pozitie.

Sistemul contine o baza de date in care vor fi stocate informatii despre echipe, jucatori, contracte si meciuri. Utilizatorii vor interactiona cu sistemul printr-o interfata prietenoasa. Exista doua tipuri de utilizatori: administrator – persoana care se ocupa de configurarea sistemului (poate adauga/edita/sterge un jucator, o echipa, un contract, poate genera etapele din campioant si poate da start sezonului); utilizator – poate vizualiza diverse statistici.

# Elaboration – Iteration 1.1

# Domain Model



# Architectural Design

## Conceptual Architecture

Proiectul este bazat pe o arhitectura de tip *MVC*(Model-View-Controller):

*Model*: reprezinta pachetul de baza al aplicatiei si contine clasele corespunzatoare bazei de date.

*View*: este un pachet care contine practice implementarea interfetei grafice a aplicatiei.

*Controller*: este un pachet care implementeaza functionalitatile partii grafice.

**

*Schema conceptuala a modelului.*

*Modelul* este reprezentat de clasele:

* Application.java
* Coach.java
* Contract.java
* Match.java
* MatchRound.java
* Nationality.java
* Player.java
* PlayerPosition.java
* PlayerRole.java
* Team.java

*View-ul* este reprezentat de fxml-urile:

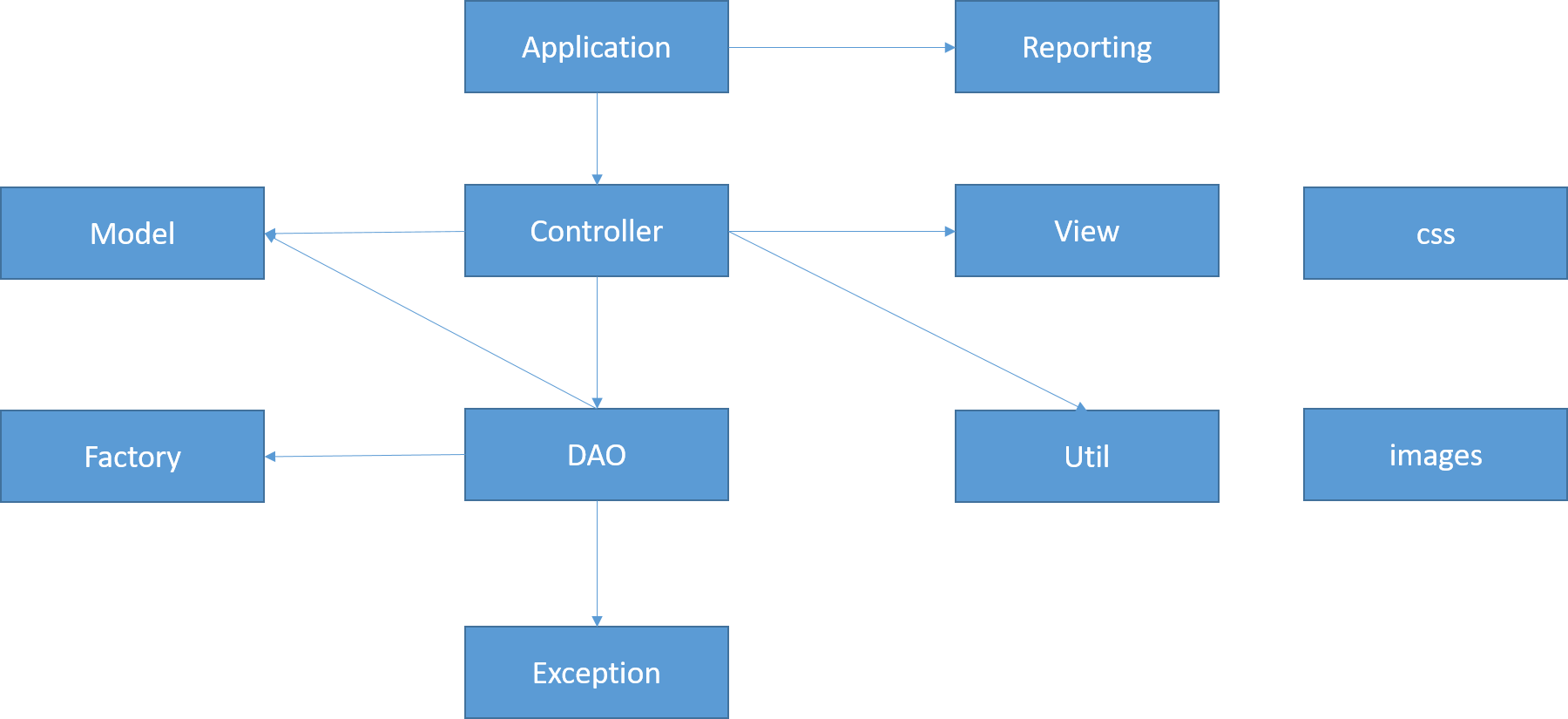
* ContractAddDialog.fxml
* ContractEditDialog.fxml
* ContractsOverview.fxml
* LeagueManagement.fxml
* MainLayout.fxml
* MatchResultsDialog.fxml
* MatchScheduleDialog.fxml
* ScheduleCreateOverview.fxml
* TeamEditDialog.fxml
* TeamsOverview.fxml

*Controller-ul* este reprezentat de clasele:

* ContractAddDialogController.java
* ContractEditDialogController.java
* ContractsOverviewController.java
* LeagueManagementController.java
* MainLayoutController.java
* MatchResultsDialogController.java
* MatchScheduleDialogController.java
* SchedulteCreateOverviewController.java
* TeamEditDialogController.java
* TeamsOverviewController.java

## Package Design

O prima diagram de pachete ar arata in felul urmator:



## Component and Deployment Diagrams

Diagrama de componente:

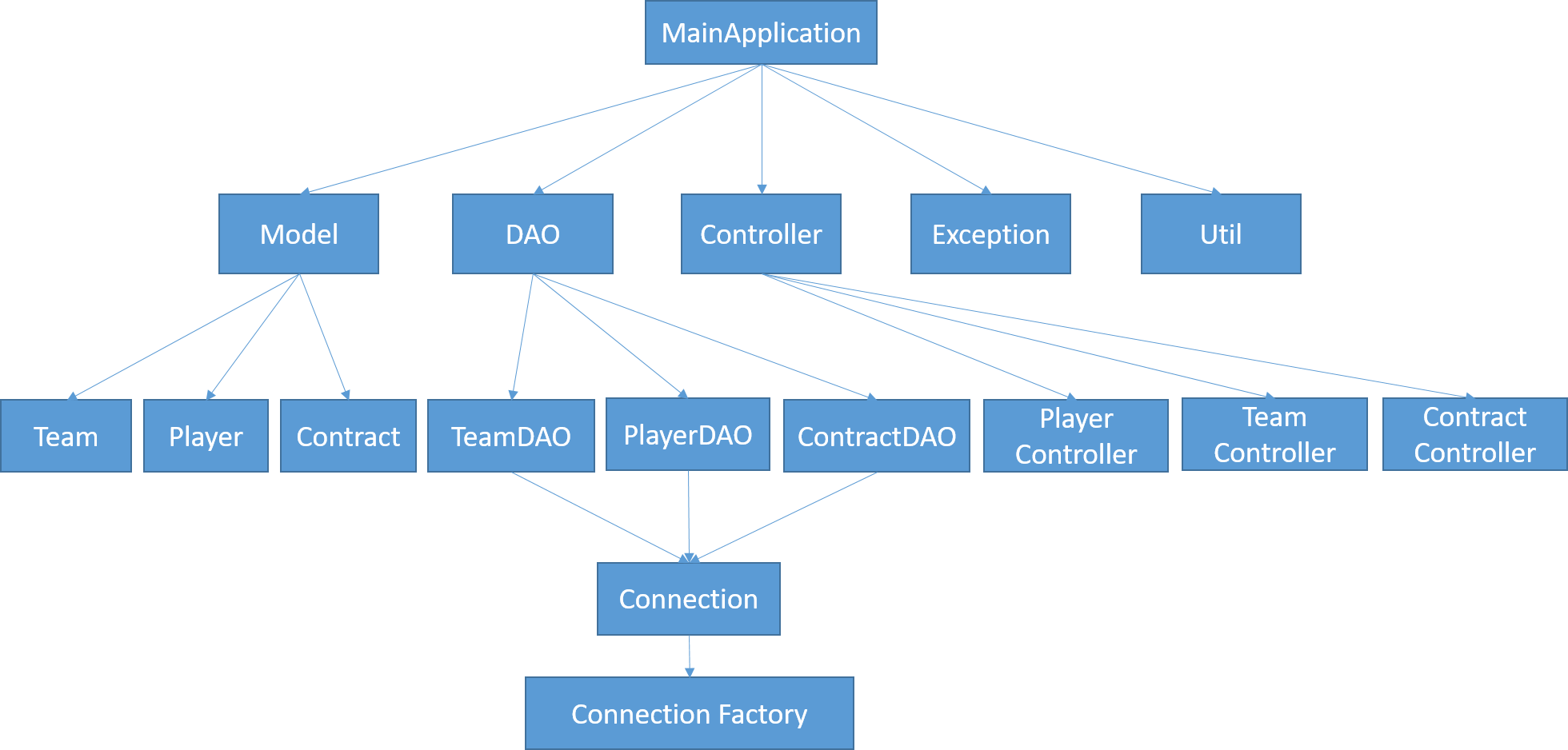
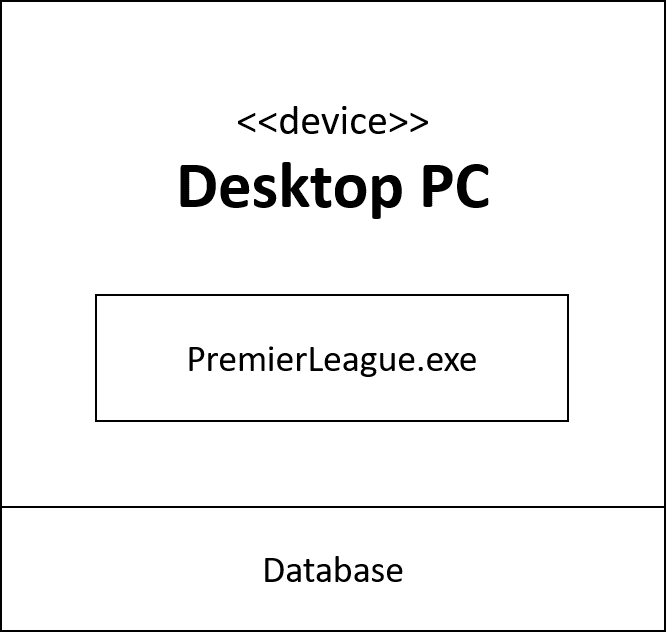


Diagrama de deployment:



# Elaboration – Iteration 1.2

# Design Model

## Dynamic Behavior

Diagrame de secventa:

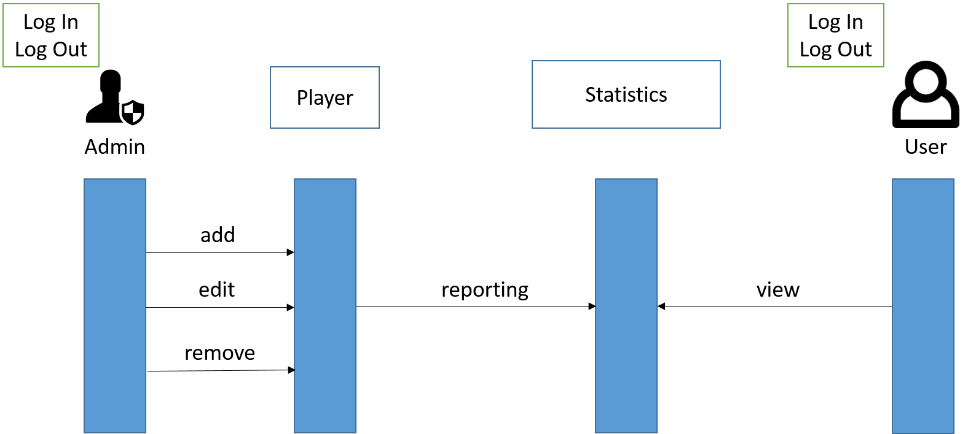
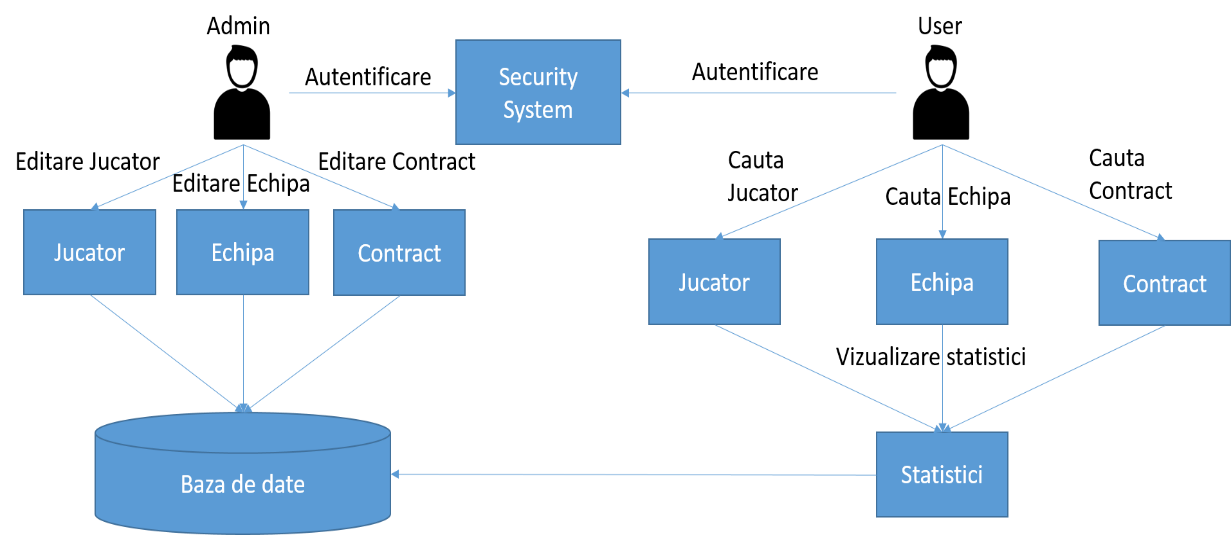
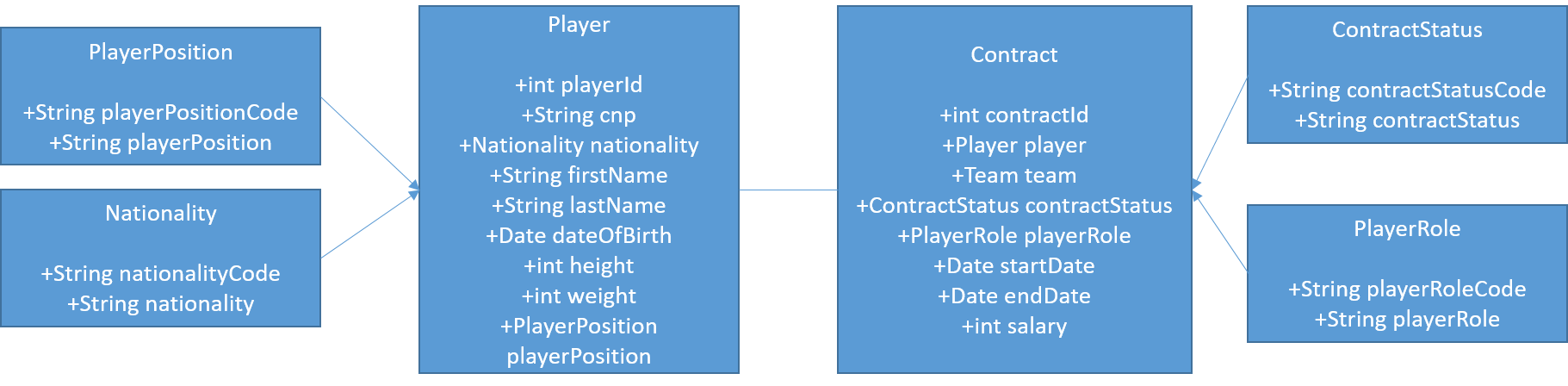




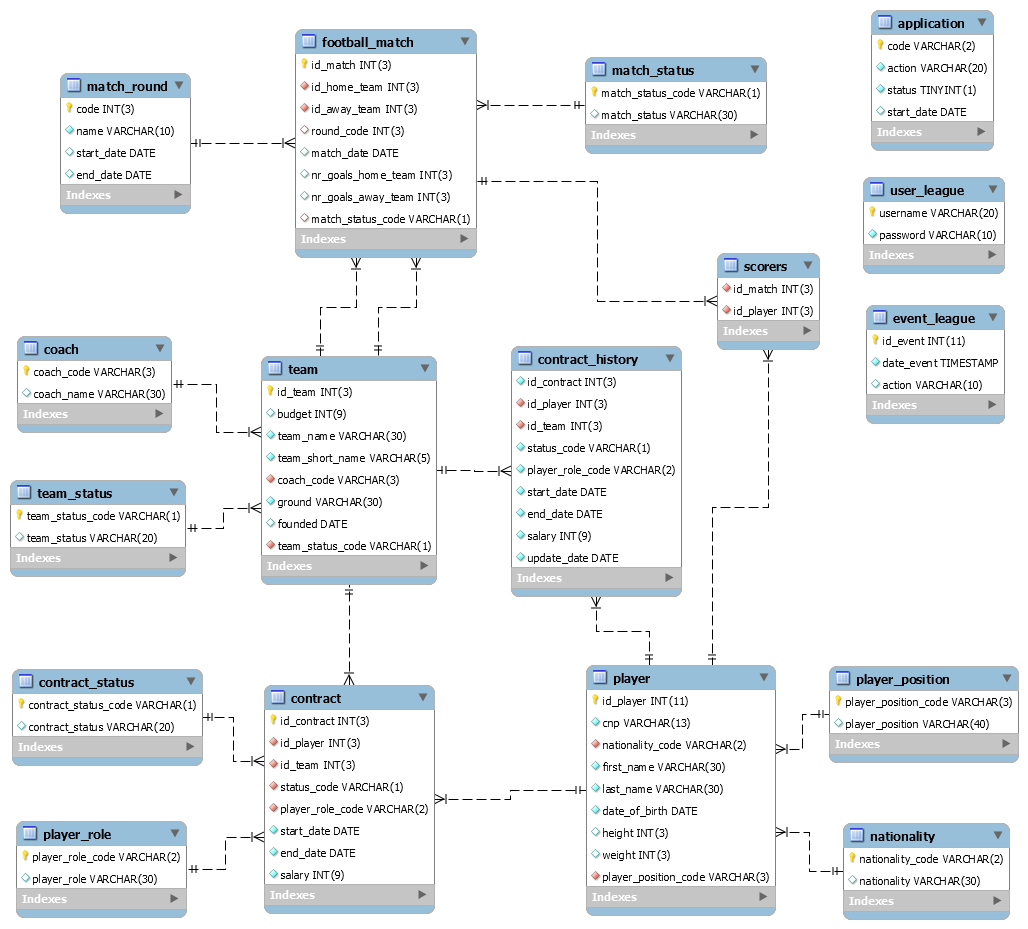
Diagrama de comunicare:

****

## Class Design



# Data Model



# Elaboration – Iteration 2

# Architectural Design Refinement

Pe langa MVC s-a folosit si design pattern-uri precum Singleton, Factory sau DAO.

Design pattern-ul singleton s-a folosit pentru crearea conexiunii la baza de date. Astfel nu este creata o conexiune la fiecare accesare a bazei de date ci sunt create spre exemplu maximum 5 conexiuni (extindere la Singleton) si acestea sunt folosite pe parcursul utilizarii aplicatiei de catre utilizator.

Design patter-ul Factory s-a folosit pentru crearea de instante ale unor clase model. Astfel atunci cand a fost nevoie de crearea unei instante s-a folosit clasa ModelFactory si s-a apelat metode getModel cu un parametru de tip String.

Design pattern-ul DAO este folosit mai ales pentru aplicatii enterprise. Este un design pattern folosit pentru a sepera operatiile low level de layerele superioare. Astfel au fost create clase DAO pentru clasele model pentru a interactiona cu baza de date.

**GRASP**

Controller-ul GRASP este controller-ul din MVC deoarece metodele din controller tind sa reprezinte evenimente sistem, care raspund la input-urile utilizatorului.

Pe partea de Creator pentru GRASP s-a folosit design pattern-ul Factory Method. Astfel a fost create o clasa ModelFactory care este responsabila pentru a noi instante pentru clasele model.

High cohesion – clasele au un scop foarte bine definit. Astfel ele sunt foarte usor de inteles si de reutilizat. Spre exemplu au fost create clase controller pentru aproape fiecare parte din model. De asemenea au fost create clase care extind clasa Exception. Aceste clase sunt folosite in clasele DAO si astfel se arunca exceptii acolo unde este cazul si se stie clar de unde provin aceste exceptii.

Polimorfism – in acest proiect a fost creat un pachet cu numele exception. Acest pachet contine clase care extind clasa Exception si astfel in cadrul metodelor este folosit super().

# Design Model Refinement

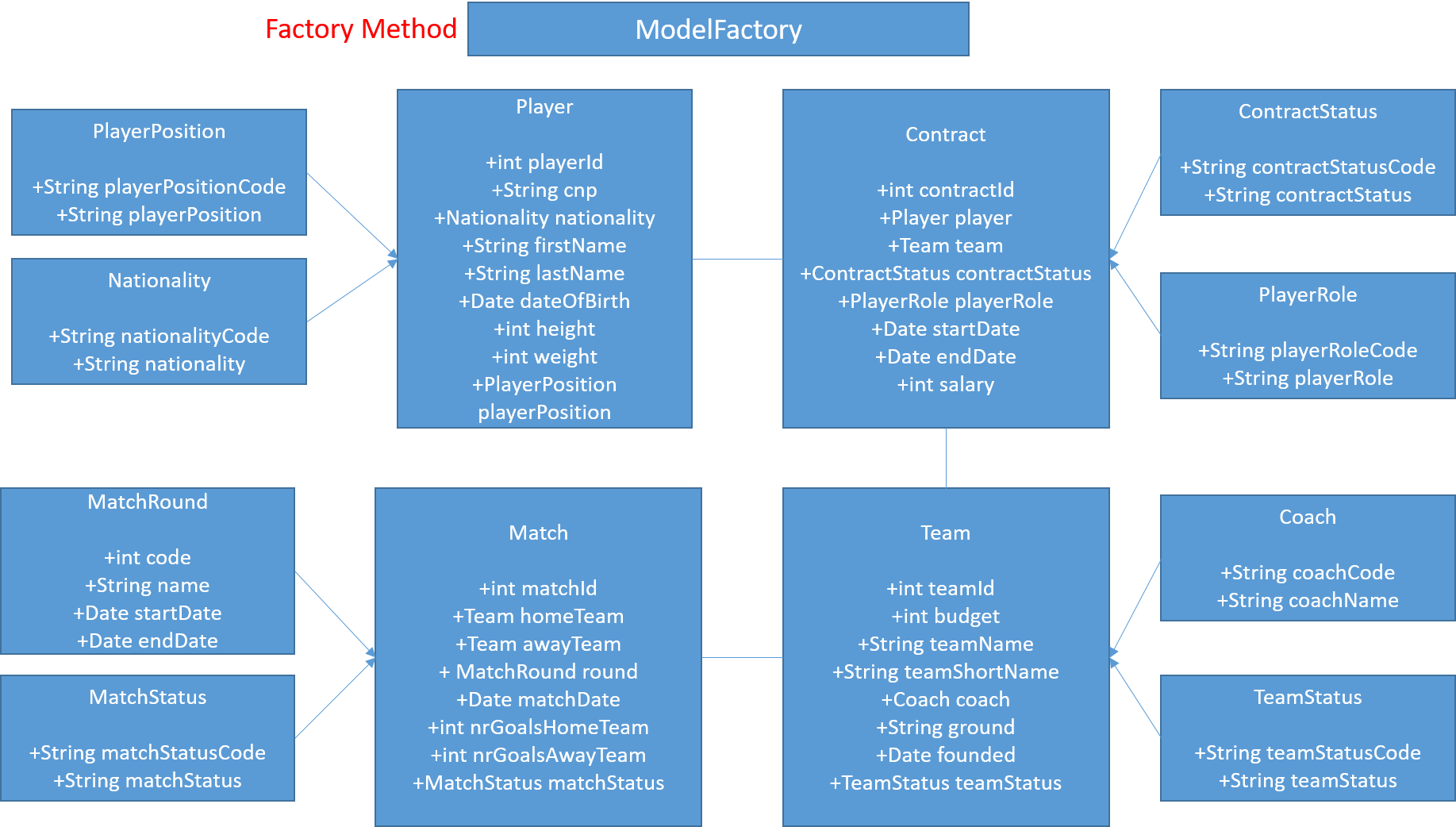


Fig. Design Model (1)

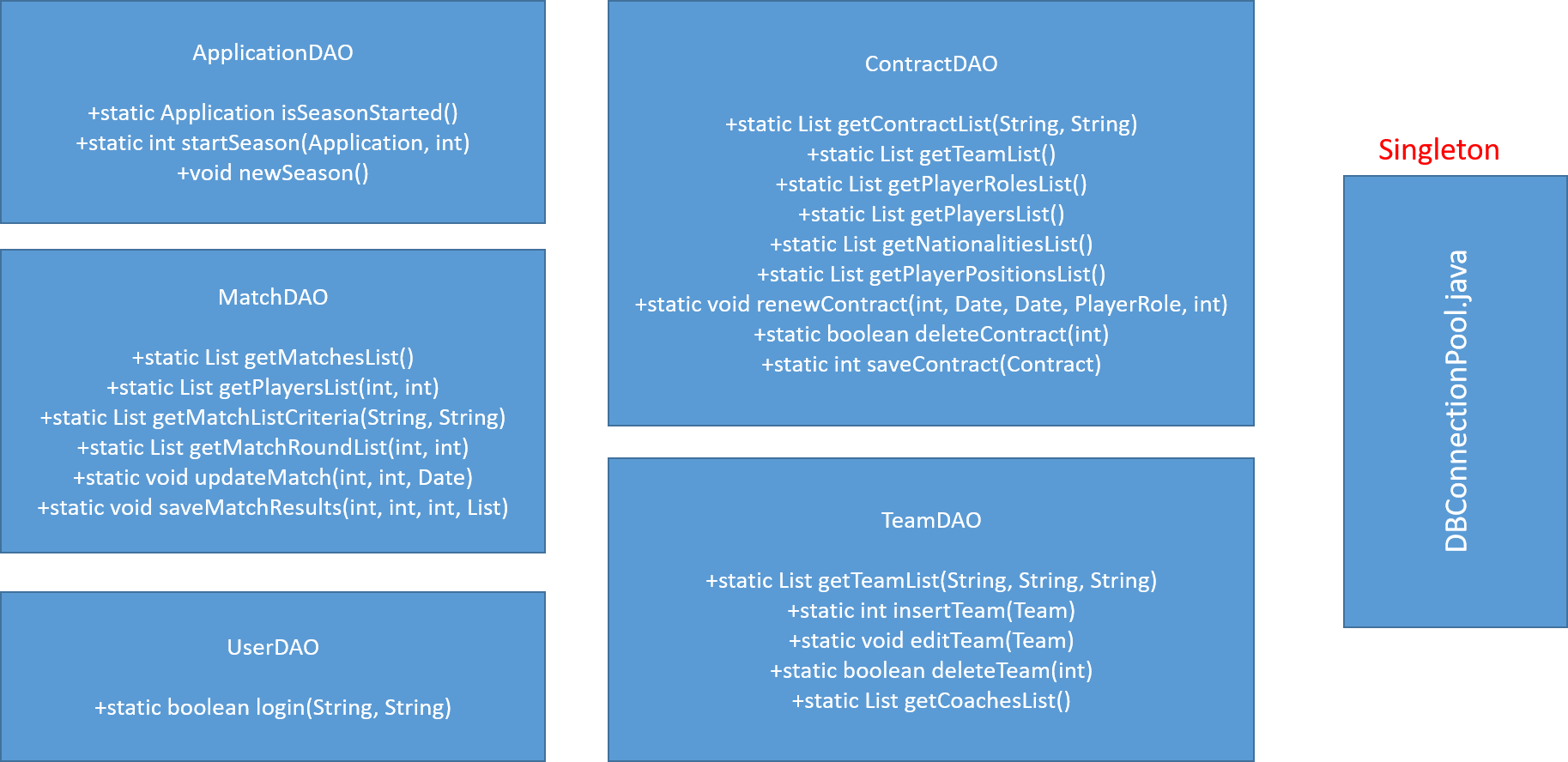


Fig. Design Model (2)

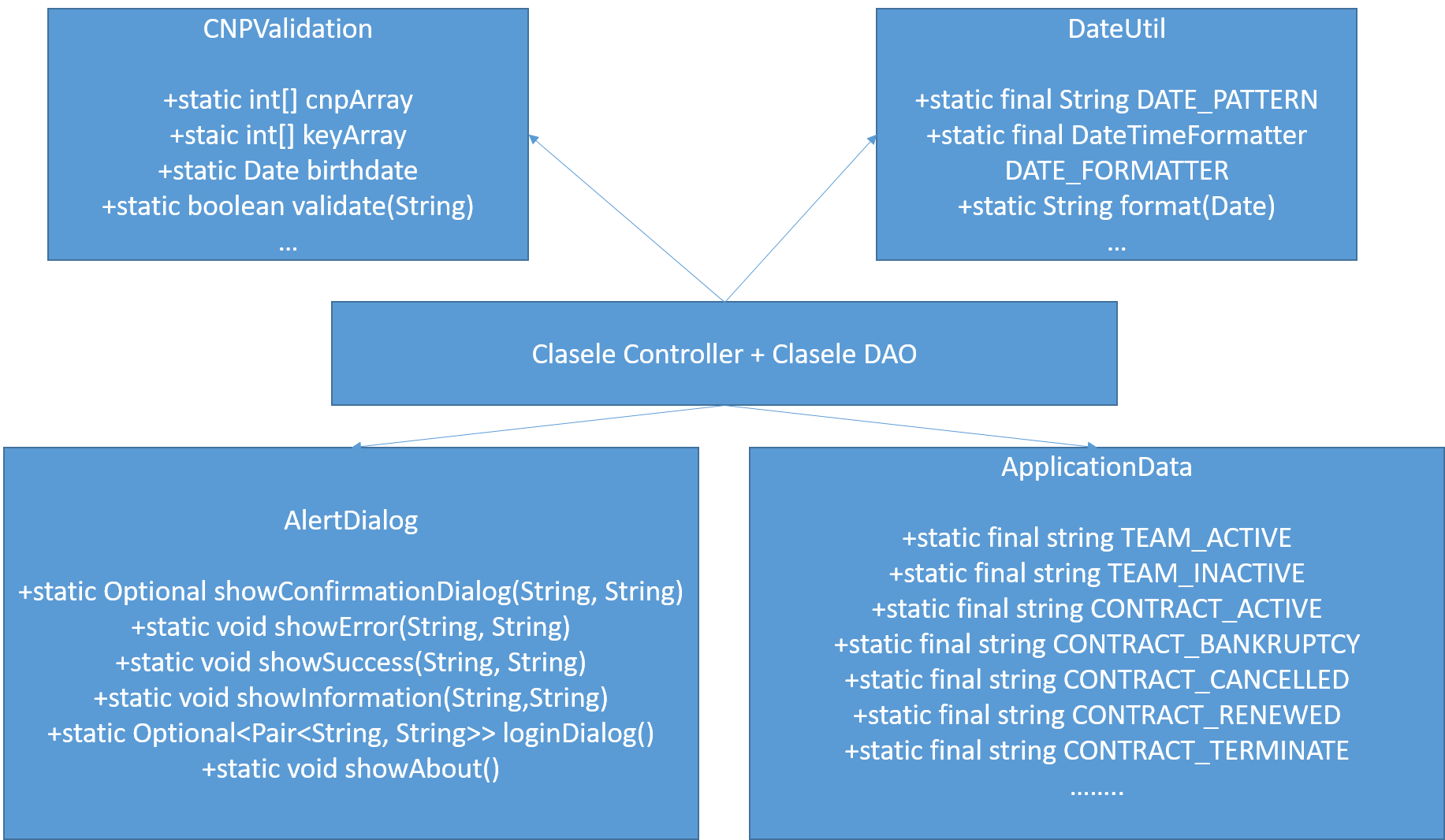


Fig. Design Model (3)

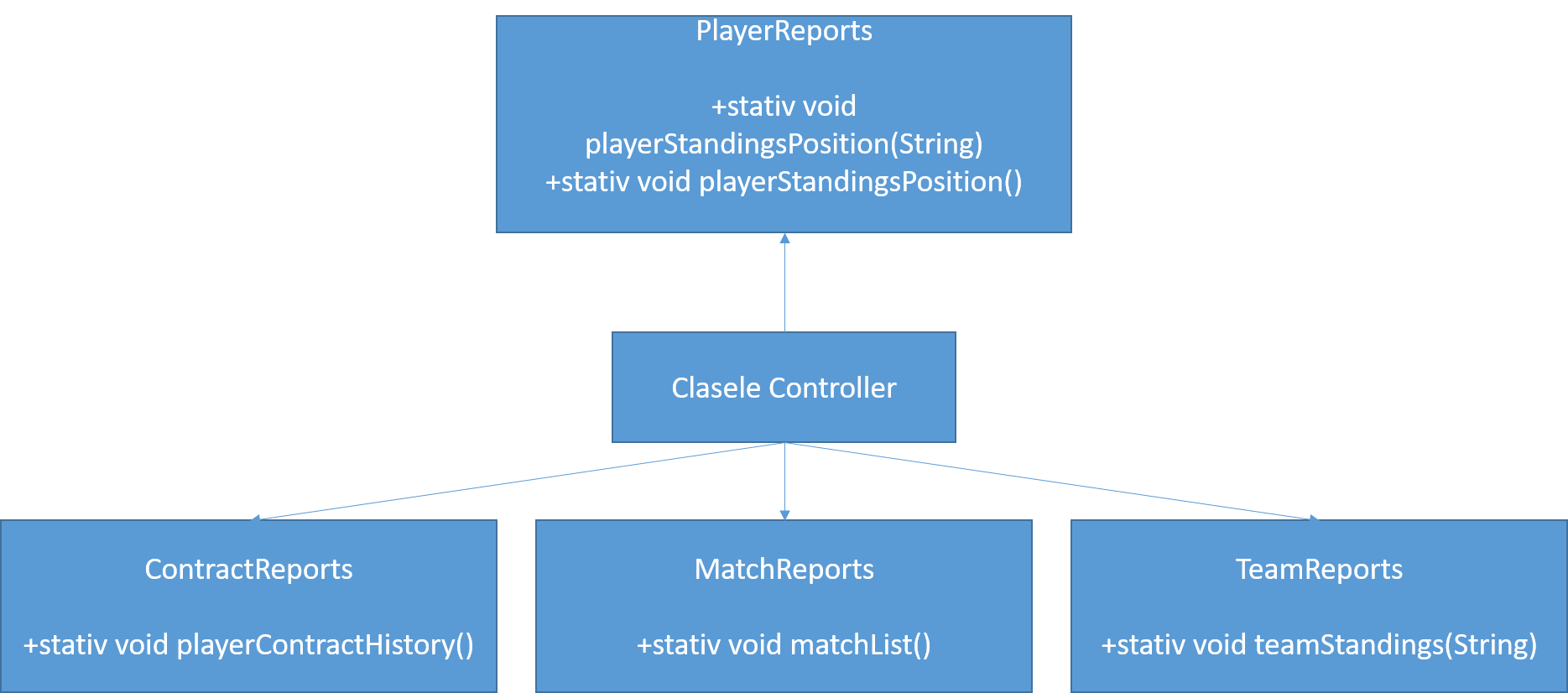


Fig. Design Model (4)

# Construction and Transition

# System Testing

S-a testat clasa ModelFactory folosita pentru integrarea design pattern-ului Factory Method. A fost testata instantierea claselor Coach si Nationality.

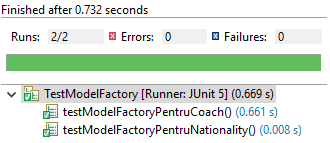


Fig. Testare clasa ModelFactory

Aplicatia dispune de asemenea de numeroase validari. Spre exemplu la introducerea unui nou jucator, CNP-ul acestuia trebuie sa fie adevarat.

# Future improvements

Pe viitor ar putea fi extinsa baza de date astfel incat sa stocheze informatii despre numarul de cartonase galbene si rosii obtinute de jucatori, numarul de lovituri de pedeapsa, numarul loviturilor libere, inlocuiri etc. Aceste date putand fi folosite pentru realizarea de raportari cat si gestinuea jucatorilor care pot juca in meciul urmator sau nu. De asemenea, ar putea fi tinuta evidenta transferurilor de jucatori. Aplicatia ar putea fi optimizata pentru lucrul cu mai multe fire de executie.

Aplicatia poate fi usor adaptata pentru a fi folosita si de alte ligi de fotbal, de exemplu “Ligue 1” in Franta.

# Bibliography

<https://docs.oracle.com/javase/8/docs/api/>

<http://code.makery.ch/library/javafx-8-tutorial/>

<http://zetcode.com/gui/javafx/>

<http://tutorials.jenkov.com/javafx/index.html>

<https://www.premierleague.com/>

<https://en.wikipedia.org/wiki/Premier_League>

<https://community.jaspersoft.com/questions/525754/how-create-pdf-file-java-class>

<https://www.w3resource.com/sql/sql-procedure.php>