

SPAD

RELATÓRIO

TPSIP\_10.17

Bruno Ferreira

João Santos

Luís Passeira

Outubro de 2018

# Índice

[Índice 3](#_Toc493864898)

[1. Introdução 5](#_Toc493864899)

[1.1 Enquadramento 5](#_Toc493864900)

[1.2 Objetivos a Cumprir 5](#_Toc493864901)

[1.3 Organização do Relatório 5](#_Toc493864902)

[2 Análise de Sistemas 7](#_Toc493864903)

[2.1 Requisitos Funcionais 7](#_Toc493864904)

[2.2 Requisitos Não Funcionais 7](#_Toc493864905)

[*2.3* Diagrama de *Use Cases* 7](#_Toc493864906)

[2.4 *Use Case XPTO – Descrição Estruturada* 7](#_Toc493864907)

[2.5 Diagrama de Classes 7](#_Toc493864908)

[*2.6* Diagrama de Sequência para o *Use Case XPTO* 7](#_Toc493864909)

[2.7 Fluxograma para o *Use Case XPTO* 7](#_Toc493864910)

[*2.8* Diagrama de Fluxo de Dados para o Processo XPTO 7](#_Toc493864911)

[2.9 Diagrama E-R 7](#_Toc493864912)

[2.10 Modelo de Dados 7](#_Toc493864913)

[2.11 Dicionário de Dados 7](#_Toc493864914)

[3 Desenvolvimento da Aplicação 9](#_Toc493864915)

[3.1 Arquitetura da Aplicação 9](#_Toc493864916)

[3.2 Tecnologias 9](#_Toc493864917)

[*3.3* *Pontos Importantes* 9](#_Toc493864918)

[3.4 Descrição do uso da aplicação e suas funcionalidades 9](#_Toc493864919)

[4 Conclusão 11](#_Toc493864920)

[4.1 Objectivos Cumpridos 11](#_Toc493864921)

[4.2 Contratempos 11](#_Toc493864922)

[4.3 Dificuldades Encontradas 11](#_Toc493864923)

[4.4 Apreciação pessoal 11](#_Toc493864924)

[5 Referências 13](#_Toc493864925)

[6 Anexos 15](#_Toc493864926)

# Introdução

Com o advento da internet surgiu também um mundo sem fim de novas oportunidades.

Entre elas a disponibilização de serviços on-line de casas de apostas de futebol, as quais se têm multiplicado nos últimos anos.

O homem, é, historicamente um ser atraído pelo jogo e pelo risco, consequentemente o mercado das apostas desportivas online, é um mercado que vem crescendo consistentemente ao longo do tempo.

Aliado ao facto do forte investimento que estas casas de apostas realizam em publicidade, juntamente com a cada vez maior facilidade de acesso à internet que as pessoas em todo mundo possuem, é de prever que este mercado continue a crescer nos próximos anos.

Atualmente existem milhões de pessoas em todo o mundo que realizam apostas desportivas online diariamente, a grande maioria sem conhecimento nem ferramentas adequadas para realizar uma aposta desportiva consciente.

Assim fica claro que este facto representa uma oportunidade de negócio ainda pouco explorada, gerada pelo mercado “principal”, o mercado das apostas desportivas online.

O projeto SPAD é, portanto, uma iniciativa de criação de um serviço online baseado em análise estatística, de prognósticos de apostas desportivas, que visa fornecer aos apostadores online, uma ferramenta que os ajude a apostar de forma mais consciente e consistente.

## Enquadramento

SPAD ou Sistema de Prognósticos de Apostas Desportivas, enquadra-se no âmbito do Projeto Final do curso TPSIP\_10.17.

## Objetivos a cumprir

Entre os vários objetivos que este projeto se propõe a alcançar, destacam-se os seguintes:

- Obter dados de eventos desportivos de futebol de forma automática (Web-API).

- Validar a fidedignidade dos dados obtidos através da Web-API.

- Persistir os dados no nosso sistema (Base de dados MS SQL Server).

- Aplicar algoritmos estatísticos sobre os dados recolhidos

- Gerar prognósticos para futuros eventos desportivos de futebol.

- Guardar o histórico dos prognósticos gerados pelo sistema.

- Permitir o registo de utilizadores no sistema e a sua gestão.

- Permitir a criação e envio de newsletter para utilizadores registados.

- Permitir a consulta do histórico dos prognósticos gerados pelo sistema.

- Criar duas Aplicações Web (Front Office; Back Office) que permitam a automatização e a fácil gestão de todo o sistema, bem como a consulta dos dados gerados de forma simples.

- Garantir o acesso ao Front Office através de dispositivos móveis.

## Organização do Relatório

Nota: deve descrever-se a estrutura do relatório em termos de capítulos principais, explicando o que se vai abordar em cada um.

# Análise de Sistemas.

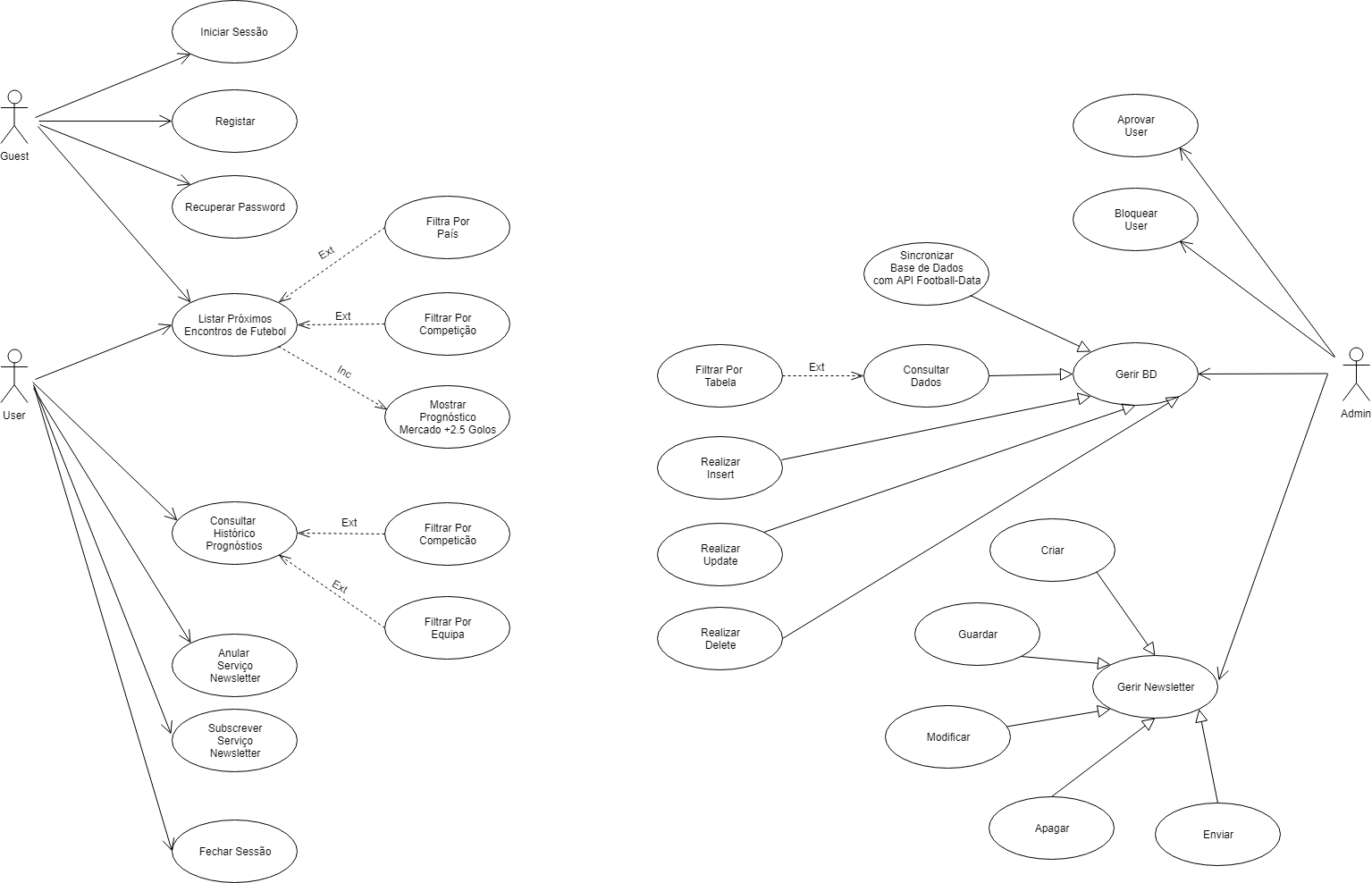
## Requisitos Funcionais

* Cada Utilizador deve poder fazer registo.
* Cada Utilizador deve poder iniciar sessão e terminar a mesma, caso já a tenha iniciado.
* Cada Utilizador deve poder recuperar a password da sua conta, caso não tenha sessão iniciada.
* Cada Utilizador pode guardar uma fotografia associada ao seu perfil.
* Um Utilizador não autenticado (sem sessão iniciada) pode ver a listagem de encontros de futebol e pode também fazer filtragem por competição e/ou equipa, mas não têm acesso aos prognósticos disponíveis no serviço.
* Um Utilizador autenticado (com sessão iniciada) tem acesso a todas as funcionalidades anteriormente mencionadas, e também à subscrição da Newsletter.
* Cada utilizador deve confirmar o e-mail após o registo para que um administrador o possa aprovar. Depois de aprovado, o utilizador consegue ter acesso às funcionalidades do sistema.
* Cada Administrador pode aprovar Utilizadores.
* Cada Administrador pode bloquear Utilizadores.
* Cada Administrador pode gerir a base de dados, de acordo com as seguintes ações:
  + Sincronizar com a API;
  + Filtrar dados;
  + Introduzir dados;
  + Modificar dados;
  + Eliminar dados.
* Cada Administrador pode gerir o serviço de Newsletter, de acordo com as seguintes ações:
  + Criar Newsletter;
  + Guardar Newsletter na base de dados;
  + Modificar Newsletter;
  + Apagar Newsletter;
  + Enviar Newsletter.

## Requisitos Não Funcionais

* A Base de Dados deve ser desenvolvida em *Transact-SQL*.
* A aplicação deve seguir uma arquitetura N-TIER de 3 camadas (*User Interface, Business Logic Layer e Data Access Layer*).
* Deve ser utilizada a programação orientada a objetos (*Object-oriented Programming*).

## Diagrama de Use Cases



## Use Case Listar Próximos Encontros – Descrição Estruturada

Pré-Condição:

O dispositivo deve ter conexão à internet.

Descrição:

1. O Utilizador seleciona uma Liga.
2. Apenas o botão da liga selecionada é que fica destacado.
3. O sistema procura a próxima partida de cada equipa distinta dessa liga.
4. O sistema vai obter os prognósticos correspondentes às partidas obtidas.

Pós-Condição:

Os jogos são apresentados com os respetivos prognósticos, com indicação de “Bet/No Bet” no mercado 2,5.

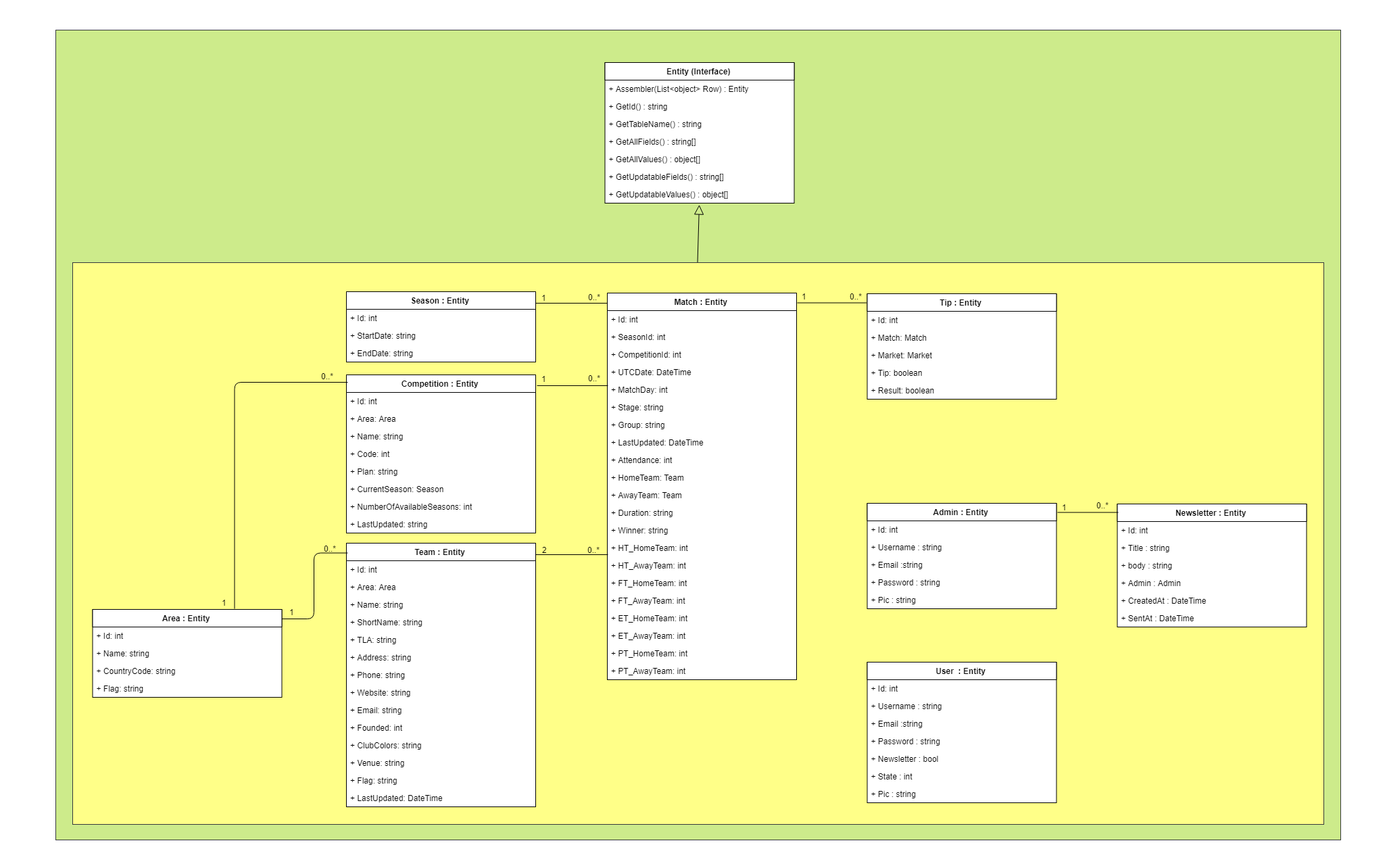
Cenário Secundário:

* 1. O utilizador pode selecionar várias ligas ao mesmo tempo.
  2. No caso de o utilizador ter selecionado várias ligas os botões das mesmas permanecerão destacados.
  3. Se não houver encontros a aplicação mostra uma mensagem de erro e volta ao ponto 1.

## Diagrama de Classes

- Representação das classes que mapeiam as tabelas mais importantes da nossa base de dados.

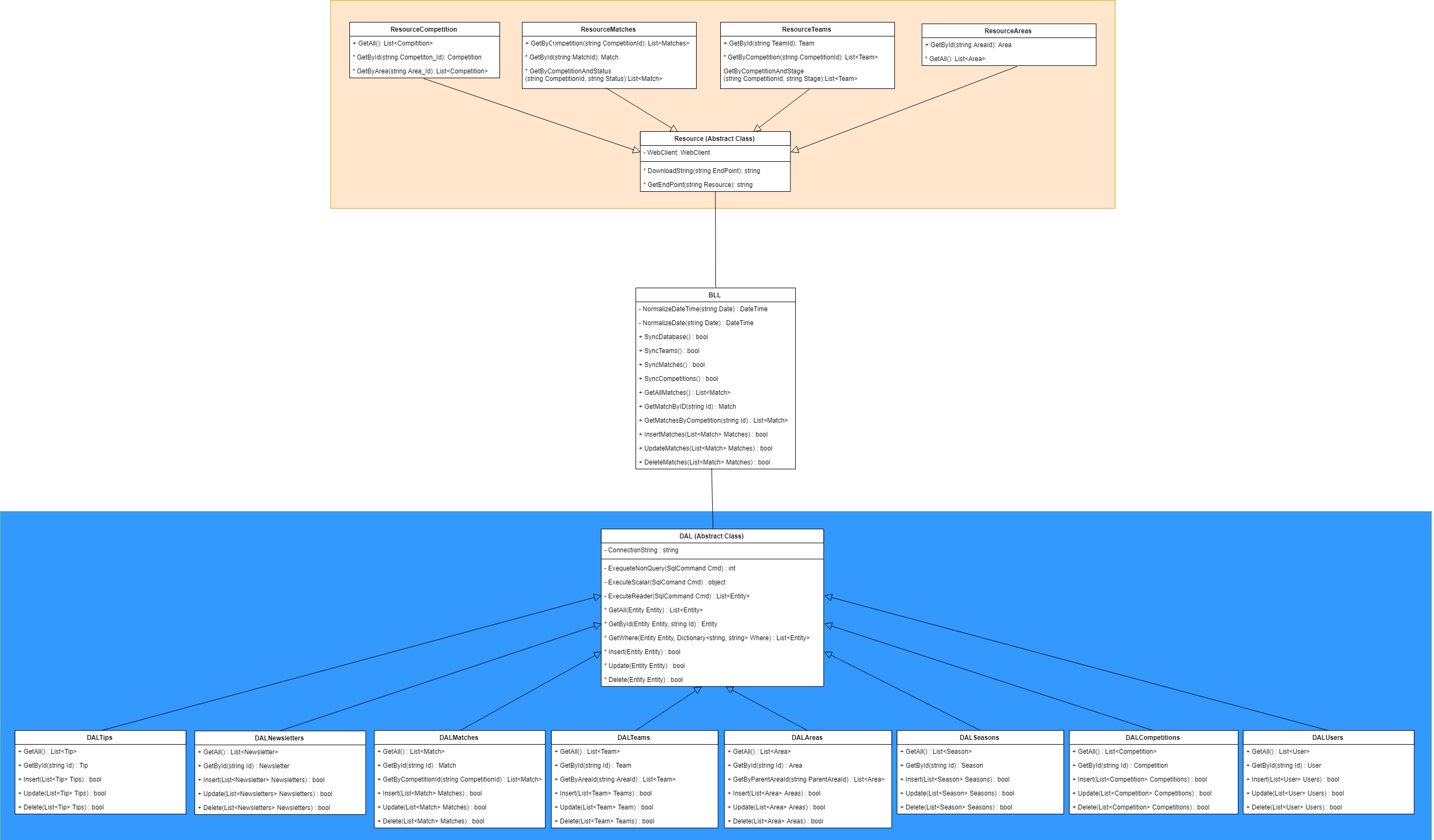
- Todas estas classes implementam a interface Entity permitindo assim o desenvolvimento de uma DAL genérica que trabalha apenas com objetos do tipo Entity.



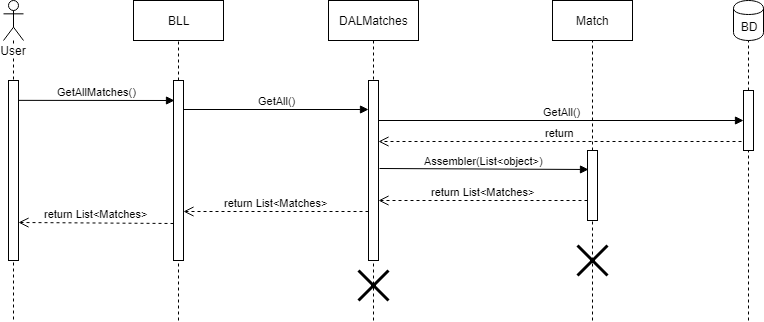
- Representação da classe abstrata DAL genérica, e das suas classes filhas. Cada uma responsável por expor os métodos necessários para comunicar com a tabela correspondente da base de dados.

- Representação da classe abstrata Resource genérica, e das suas classes filhas. Cada uma responsável por expor os métodos necessários para comunicar com os diferentes “endpoints” da API Football-Data.Org.

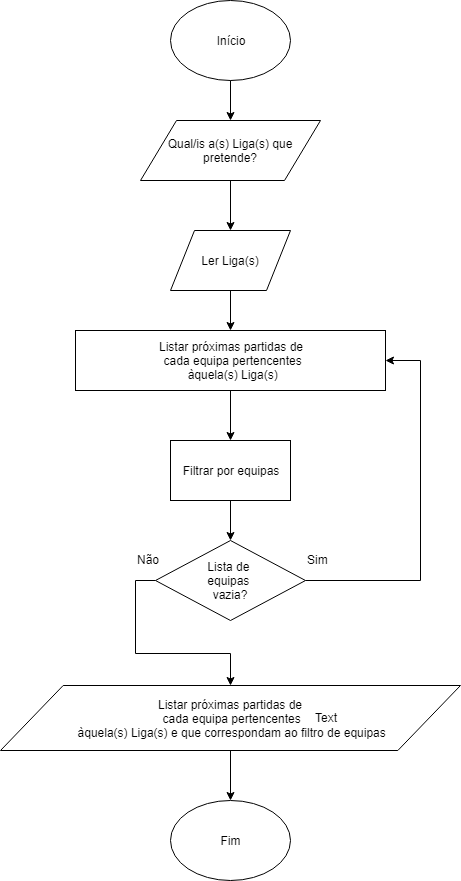
- Representação da BLL que expões os mais variados métodos para comunicar com a API, DAL e também métodos de validação.



## Diagrama de Sequência para o Use Case Listar Próximos Jogos

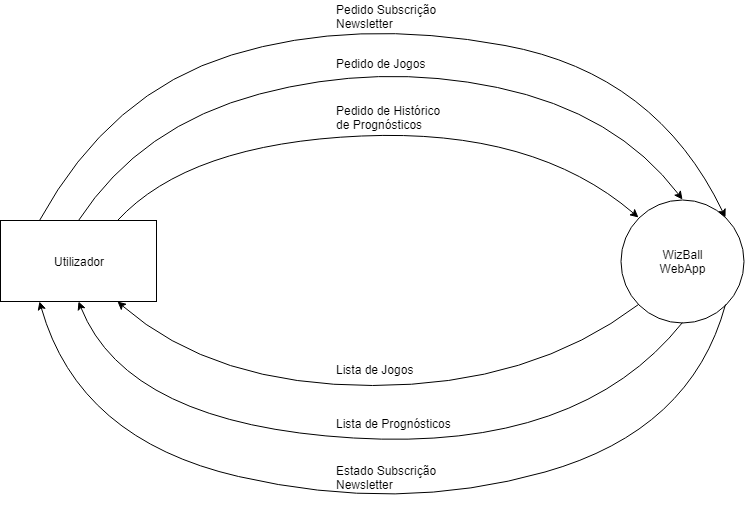


## Fluxograma para o Use Case Listar Próximos Jogos

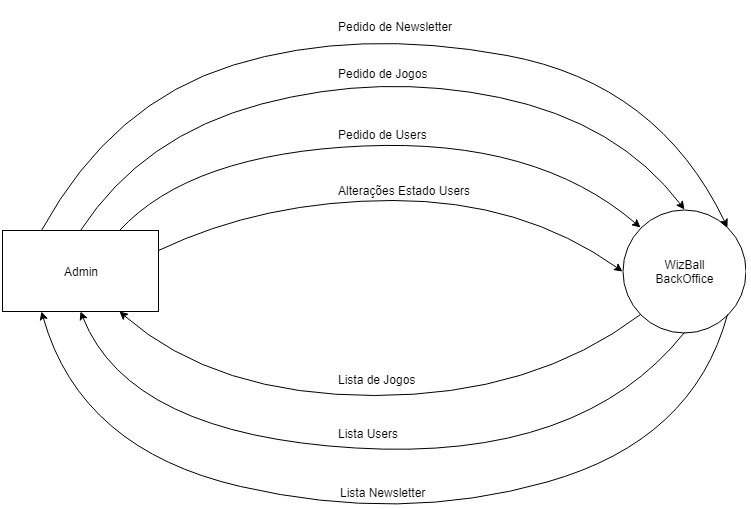


## Diagrama de Contexto

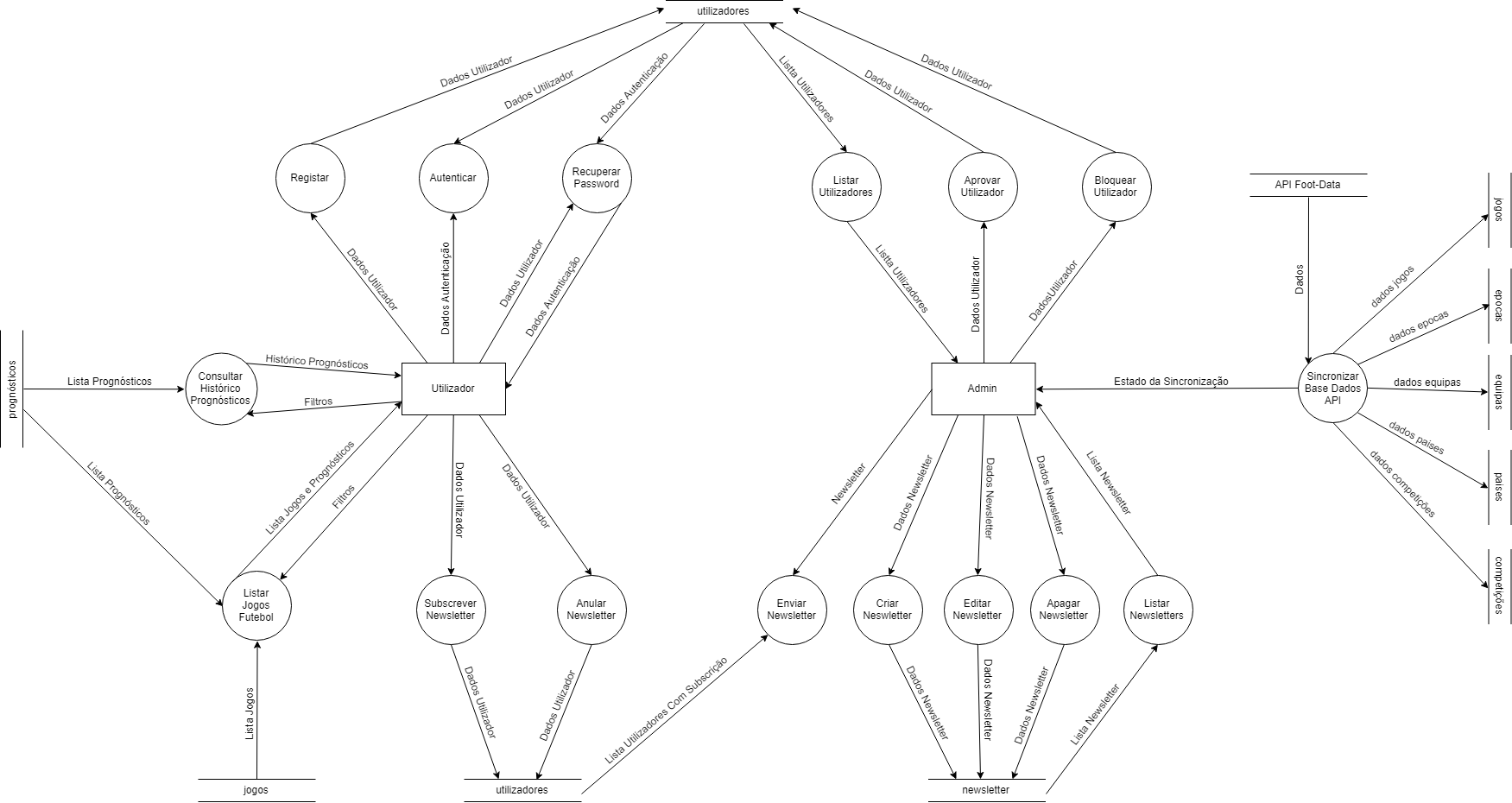
**Front Office**



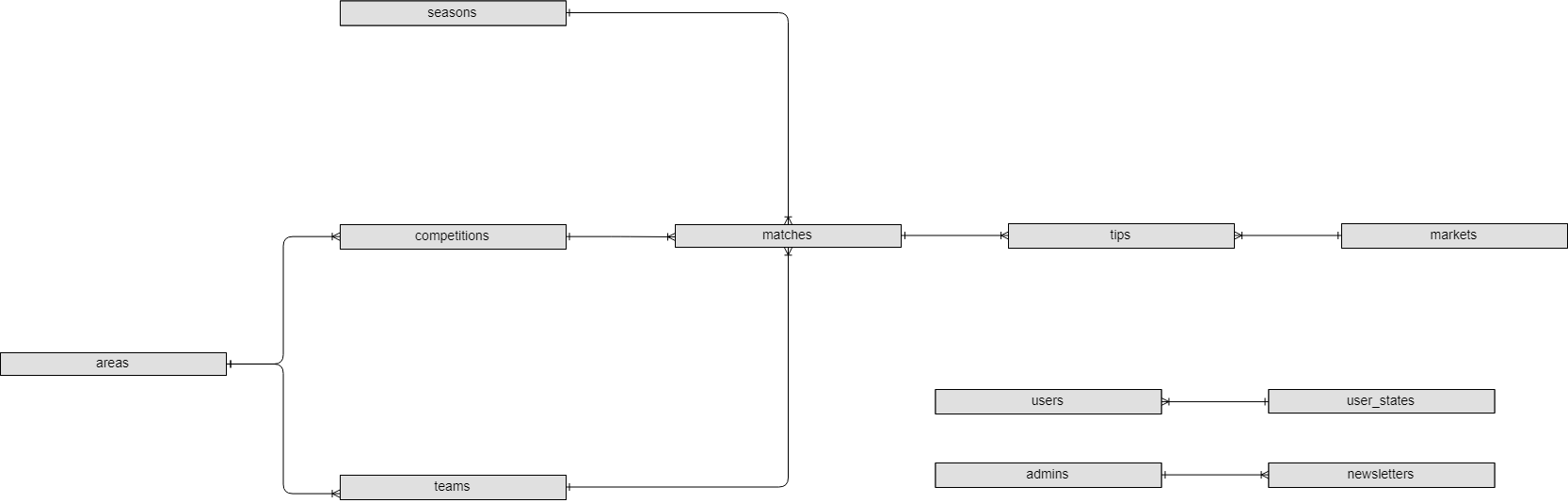
**Back Office**



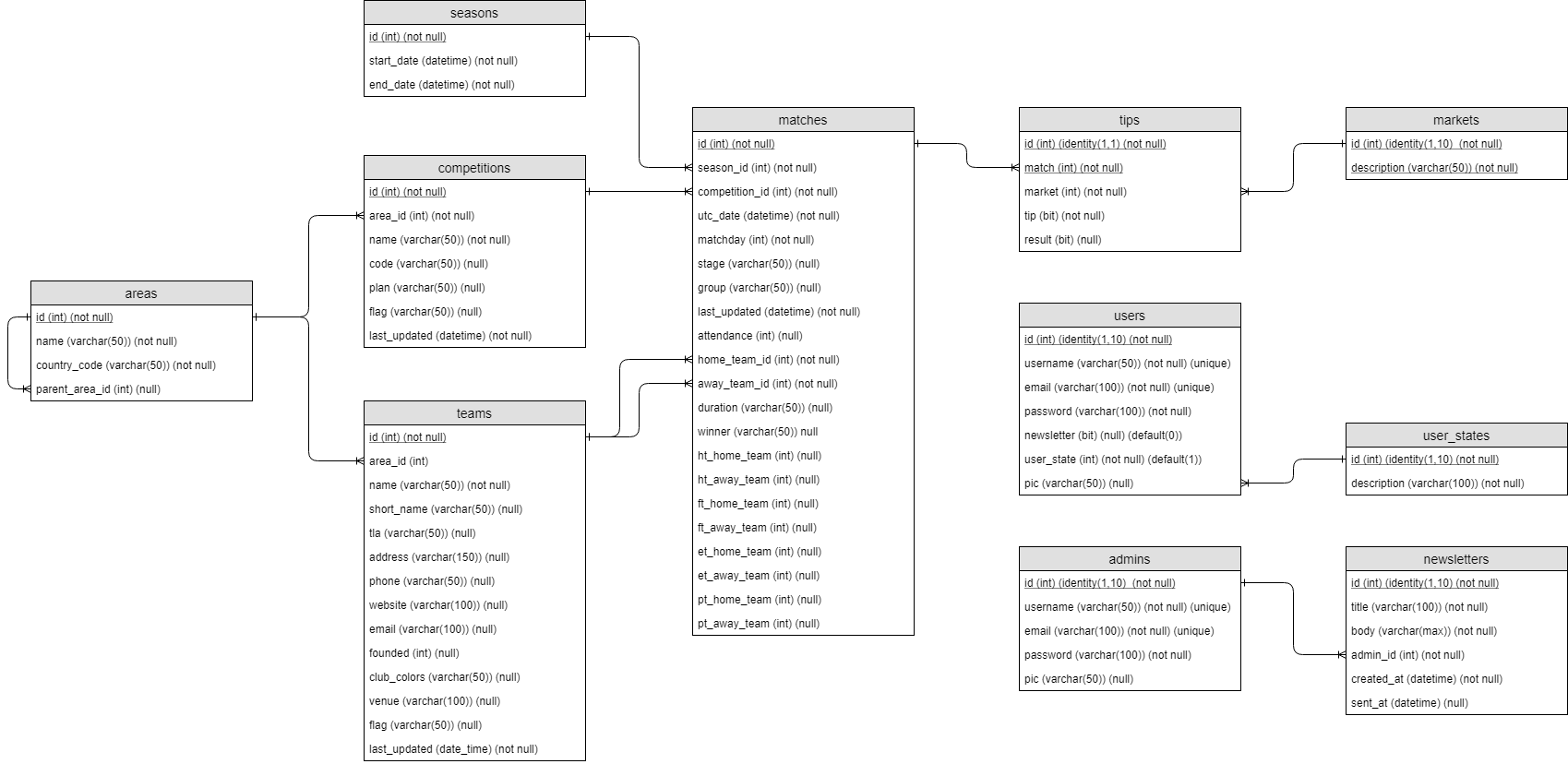
## Diagrama de Fluxo de Dados



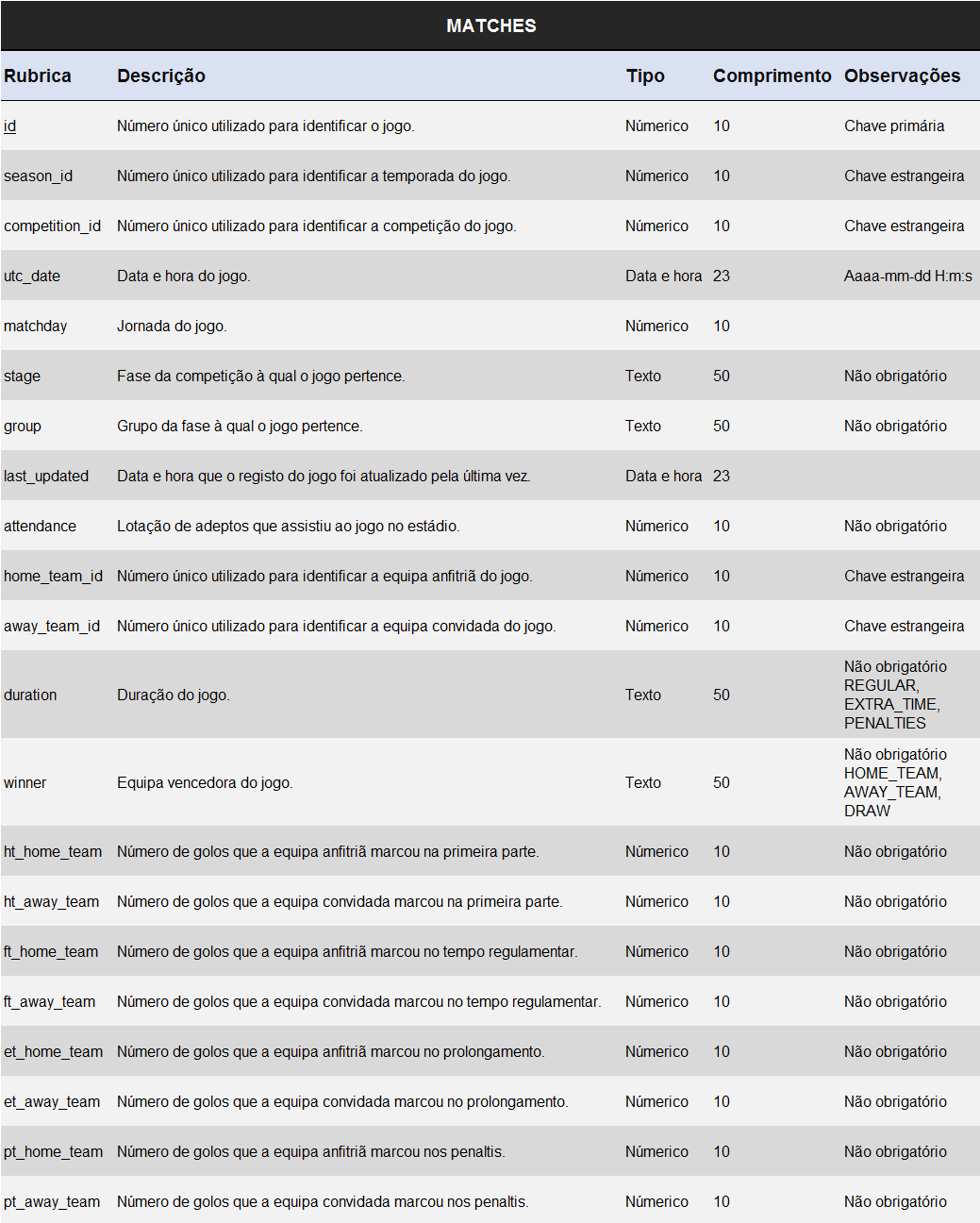
## Diagrama E-R

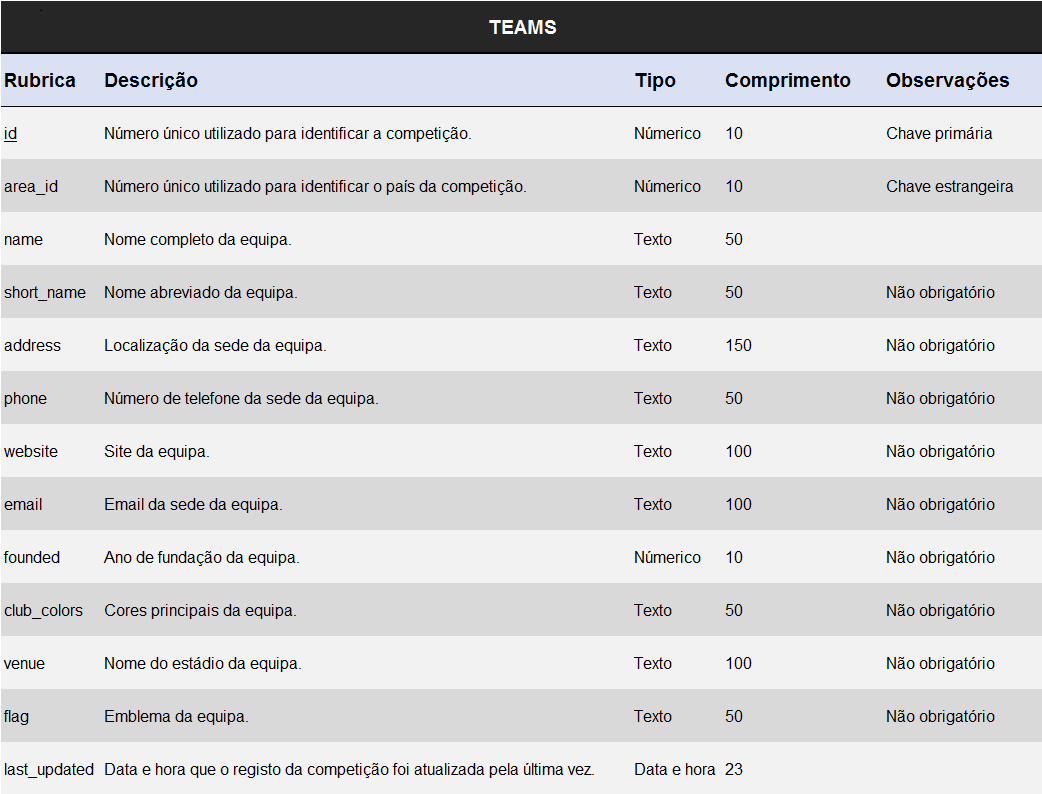


## Modelo de Dados



## Dicionário de Dados







# Desenvolvimento da Aplicação

## Módulos

Após a primeira reunião com o nosso cliente Ricardo Baptista, ficou claro que o projeto SPAD teria de ser segmentado em vários subprojectos, para que desta forma pudéssemos atender de forma mais eficiente e versátil às necessidades colocadas pelo nosso cliente. Portanto o sistema SPAD é composto por 3 módulos principais.

### Módulo - Business Logic

O módulo Business Logic é um projeto do tipo Class Library, que pode ser entendida como a espinha dorsal de todo o projeto SAPD.

A responsabilidade desta class library é encapsular toda a lógica de negócios da aplicação, e disponibilizar as funcionalidades, que, tanto o Front Office, como o Back Office vão necessitar.

Por sua vez o módulo Business Logic é também ele dividido em partes lógicas menores:

1. Entities (Conjunto de classes que mapeiam as entidades da base de dados).
2. DAL (Conjunto de classes que providenciam todos os métodos necessários para interagir com a base de dados).
3. API (Conjunto de classes que providenciam todos os métodos necessários para interagir com football-data.org).
4. BLL (Classe que expõe todos os métodos tanto da DAL como da API).
5. Web Services (Classe que expõe parcialmente os métodos da DAL e da API, para ser possível realizar chamadas AJAX apenas quando apropriado e apenas para os métodos que façam sentido).

Desta forma a Business Logic encapsula toda a lógica de negócios, bem como a disponibilização de métodos de acesso aos dados, tanto à API como à base de dados, permitindo aos desenvolvedores que usarem esta Business Logic, concentrarem-se apenas na implementação das funcionalidades que esta expõe.

### Módulo – Back Office

O módulo Back Office é um projeto do tipo Web Application, que pode ser entendida como o cérebro do projeto.

A responsabilidade desta Web Application é a de gerir todo sistema SPAD. Ou seja, nesta aplicação podem-se realizar tarefas como:

1. Gerir utilizadores.
2. Sincronização de dados entre a API e a Base de Dados.
3. Gerar prognósticos para os próximos eventos de futebol.
4. Gerar prognósticos para um histórico de eventos de futebol (Funcionalidade que serve para testar novos algoritmos, contra um conjunto de eventos de futebol já ocorridos).
5. Consultar os dados da Base de Dados.
6. Enviar Newsletters.

### Módulo – Front Office

O módulo Front Office é um projeto do tipo Web Application, que pode ser entendida como a cara do projeto.

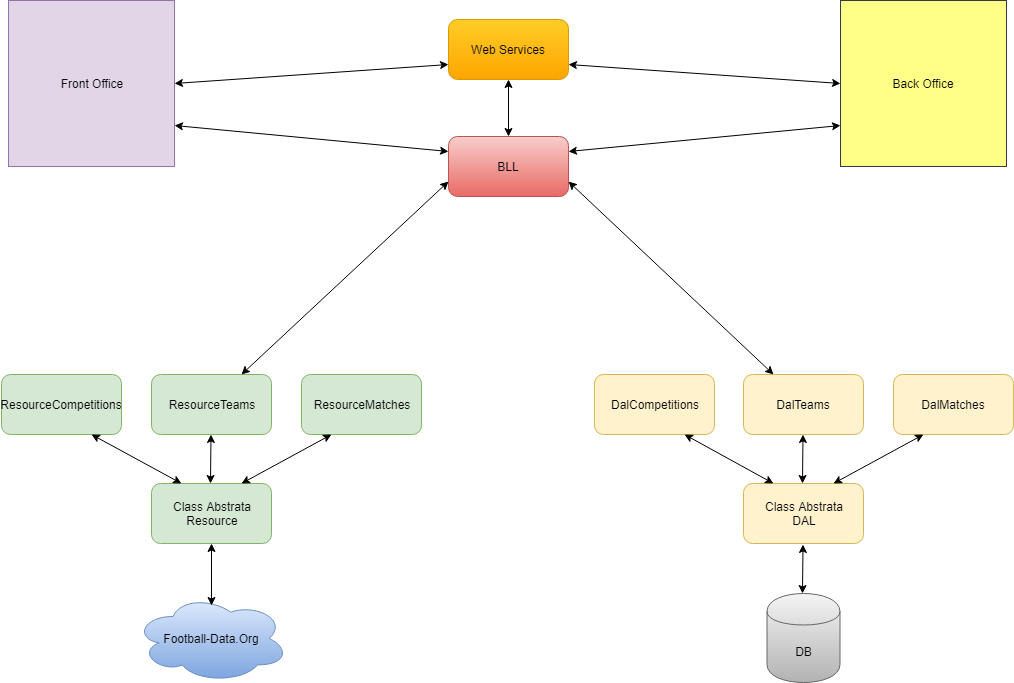
A responsabilidade desta Web Application é permitir a consulta de prognósticos de eventos desportivos de futebol de forma rápida, simples e eficiente, a partir de qualquer lugar através da internet, e de ser capaz de se adaptar à dimensão de qualquer dispositivo, garantido uma experiência de utilização sempre fluida.

## Arquitetura da Aplicação

Ambas as Web Applications (Front Office, Back Office), são aplicações n-tier de 5 camadas.

1. User interface (Front Office; Back Office)
2. Business Logic Layer (Modelo de Negócio)
3. Data Access Layer (Acesso a Dados: Ms Sql Server)
4. API Access Layer (Acesso a Dados: API Football-Data.org)
5. Web Services Layer (Acesso a Dados Assíncronos)

A imagem seguinte é uma representação das diferentes camadas da aplicação e das suas relações.



Como se pode concluir pela imagem, tanto o Front Office como o Back Office podem comunicar ambas com a BLL e WS, no entanto será sempre responsabilidade da BLL de comunicar com as camadas mais baixas da aplicação.

## Tecnologias

- Microsoft SQL Server, T-SQL.

- ASP.NET, C#.

- API football-data.org.

- HTML 5, Javascript, jQuery, CSS 3, Bootstrap 4.

- Font Awesome, Sweet Alert.

- Git.

- Gantt.

## *Pontos Importantes*

### **Business Logic**

#### Interface Entity

Antes de falar sobre qualquer outro tema relativamente ao Business Logic, é necessário primeiro abordar o nosso conceito de Entity, que existe dentro da Business Logic.

Entity é uma Interface, que diz que, qualquer classe que queira implementar esta interface, deve implementar uma série de métodos. Esta interface foi desenvolvida por forma a possibilitar o desenvolvimento de uma DAL genérica. Assim sendo, as classes que vão implementar esta interface são classes como Match, Team, Competition, ou seja, classes que têm como função mapear as tabelas da base de dados, e, também os objetos recebidos pela API Football-data.org. Estas classes além de terem as propriedades correspondentes às colunas das tabelas da base de dados a que correspondem, devem também implementar os seguintes 7 métodos da interface Entity.

public interface Entity

{

Entity Assembler(List<object> Row);

string GetId();

string GetTableName();

string[] GetInsertableFields();

string[] GetInsertableValues();

string[] GetUpdatableFields();

string[] GetUpdatableValues();

}

Desta forma, quaisquer métodos de qualquer classe da Business Logic, podem saber de forma dinâmica, informações (meta dados) sobre uma determinada Entity, independentemente da Entity em questão ser um Match ou uma Competition.

Toda a Business Logic trabalha exclusivamente com objetos do tipo Entity, isto é, enviam Entities, recebem Entities, manipulam Entities, fazem validações sobre Entities etc…

#### DAL Genérica

De forma a implementar os conhecimentos adquiridos no curso, tentamos implementar a DAL de forma genérica. Digo tentamos, porque apenas os métodos mais requisitados foram implementados de forma genérica. Esta opção deveu-se essencialmente à complexidade que tal desenvolvimento acresce em termos de tempo, ao total de tempo disponível para desenvolver o resto da aplicação.

Então, o conceito a que chegamos foi o seguinte: Existe uma classe, de nome DAL. Esta classe é abstrata, e é a responsável por configurar toda a ligação à base de dados, e, também por implementar e expor todos os métodos genéricos de acesso a dados, que posteriormente as DALs especificas podem utilizar.

Método genérico que permite executar qualquer Stored Procedure na base de dados e converter os dados retornados na Entity passada por parâmetro. Tem ainda a possibilidade de passar os SqlParameters que o ST possa eventualmente necessitar.

protected List<Entity> ExecuteStoredProcedure(Entity Entity, string StoredProcedure, SqlParameter[] Parameters)

Método genérico que permite passar uma expressão SQL pura e converter o resultado para a Entity passada por parâmetro.

protected List<Entity> GetRaw(Entity Entity, string QueryRaw)

Método que permite obter registos da base de dados, relativos à tabela correspondente à Entity passada por parâmetro, bem como indicar uma lista de condições que satisfaçam as nossas necessidades.

protected List<Entity> GetWhere(Entity Entity, List<DbWhere> Where)

Esta é apenas uma amostra dos diversos métodos genéricos implementados na classe abstrata DAL. As DALs especificas devem herdar a class abstrata DAL, e assim, desta forma, poderem consumir os recursos disponibilizados na classe abstrata DAL.

#### Lazy Loading

A DAL foi desenvolvida segundo o conceito de Lazy Loading, isto significa que todos os métodos das diferentes DALs devolvem dados “puros”. Ou seja, os dados vêm da base de dados exatamente como estão guardados.

No entanto na aplicação os objetos do tipo Entity (que mapeiam as tabelas da base de dados), são objetos complexos.

Portanto, se em alguma parte da aplicação for necessário aceder a propriedades dos objetos complexos que compõem o objeto Entity, deve-se então chamar o método **EntityBuilder (Entity Obj).**

Este método tem várias versões sobrecarregadas.

Após a conclusão deste método, os objetos do tipo Entity ficam complemente carregados.

#### API football-data.org

Esta aplicação perderia todo o sentido se os dados de que precisa fossem manualmente introduzidos na base de dados, uma vez que uma das necessidades do nosso cliente era exatamente automatizar o processo de obtenção de dados. Assim sendo, a conceção e implementação deste módulo, foi em tudo muito semelhante à da DAL genérica.

Assim, existe também uma classe abstrata de nome Resource, a qual implementa e expõe todos os métodos genéricos que os Resource específicos vão necessitar para comunicar com a API. Desta forma a classe abstrata Resource encapsula em si toda a lógica, devendo-se apenas implementar as chamadas especificas para cada End Point nos Resource específicos.

Método genérico que recebe um EndPoint ex: http:// api.football-data.org /v2/matches e que retorna uma string no formato JSON pronta a ser utilizada nas classes Resources especificas.

protected virtual string DownloadString(string EndPoint)

#### Sincronização API / Base de dados

A sincronização API / Base de dados, é dos pontos mais importantes da aplicação.

Como já foi explicado, para comunicar e obter dados da API football-data.org, existe uma série de classes responsáveis por estes processos, as classes do tipo Resource.

Da mesma forma para comunicar e trabalhar com a base de dados, existe também um conjunto de classes responsáveis por esses processos, as DALs especificas.

Também como já explicado anteriormente, tanto os Resources e as DALs conhecem e trabalham exclusivamente com objetos do tipo Entity.

Então, na BLL existem vários métodos responsáveis chamar métodos de ambas as classes, Resources e DALs especificas, realizar vários tipos de validações aos dados obtidos da API, confirmar se estes são válidos, e se, se devem guardar/atualizar na base de dados.

Estes métodos existentes na BLL foram denominados de Sincronizadores e são então os responsáveis por fazer a ponte entre a API e a Base de dados, fazendo para tal recursos dos objetos Entity.

#### Geração / atualização de prognósticos

A geração de prognósticos e a sua posterior atualização, é, por assim dizer o culminar de todo o esforço que foi colocado no desenvolvimento da Business Logic e no projeto SPAD em geral. Na BLL existem, portanto, diversos métodos responsáveis por garantir que estes processos de geração / atualização de prognósticos ocorrem na ordem devida, e, apenas para os eventos elegíveis.

Por exemplo, existe um método responsável por validar os eventos desportivos de futebol, que são elegíveis para receber um novo prognóstico. Antes de mais, é responsabilidade deste método confirmar que a tabela Matches na base de dados está devidamente sincronizada, utilizando para isso os métodos já anteriormente mencionados, os sincronizadores. Posteriormente deve requisitar através de um outro métodos, os jogos que são elegíveis para geração de prognósticos, e sobre estes, validar se já lhes foi atribuído algum prognóstico, e, só em caso negativo, proceder a geração de um novo prognóstico, recorrendo uma vez mais a um outro método, cuja a responsabilidade é apenas a de fazer realmente a geração de um novo prognóstico para um determinado evento.

// TIPS GENERATION

public void RunNextMatchesTips()

{

// Runs matches sync.

SyncMatchesTierOne();

// Decides which matches are eligible for the Full Time Over Two and Half Goals generation tip.

SetNextMatchesTips();

// Updates previous tips for the matches which have been played in the mean while.

SetTipsResults();

}

Este método apresenta um exemplo da lógica que é aplicada sempre que é necessário trabalhar com prognósticos. Nada pode ser deixado ao acaso, e tudo deve ocorrer por uma ordem devidamente estabelecida, garantido a correta execução dos diversos processos inerentes a este método.

#### Teste de novos algoritmos de geração de prognósticos sobre históricos de eventos.

Uma necessidade sentida imediatamente após o inicio da implementação dos diversos métodos relativos à geração / atualização de prognósticos, foi a impossibilidade de testar se o algoritmo criado seria eficaz ou não. Apenas com o passar do tempo e com cada vez mais jogos de futebol na base de dados, poderíamos chegar a alguma conclusão. O mesmo seria verdade sempre que quiséssemos implementar um novo algoritmo, ou seja, teríamos de voltar a aguardar que novos eventos de futebol ocorressem para podermos perceber a performance desse mesmo algoritmo.

Apercebendo-nos deste facto, foi desenvolvido um método que aplica qualquer novo algoritmo de prognósticos, sobre um histórico de eventos de futebol, possibilitando-nos assim, saber de forma imediata a performance que este novo algoritmo teria, se tivesse sido também ele previamente implementado.

// Gets a list of history matches (from season starting till yesterday)

// Then for each one of those matches this method calls the SetTip (Match, PlayedMatches).

private void SetHistoryMatchesTips()

{

// Ensure the beginning of the season.

DateTime startDate = DateTime.UtcNow.AddYears(-1);

// Today 00:00:00 this ensures only matches before today.

DateTime finalDate = DateTime.UtcNow.Date;

foreach (Competition competition in TierOneCompetitions())

{

// Gets a list with matches that already have been played for a given competition.

List<Match> playedMatches = GetMatchesByCompetitionAndRangeDates(competition.Id.ToString(), startDate, finalDate);

foreach (Match match in playedMatches)

{

// First we try to get from the database the tip corresponding to the current match.

Tip matchTip = GetTipByMatchId(match.Id.ToString());

// Only if this match does not already has a tip in the database then we will generate one.

if (matchTip is null)

{

// Method call which will generate the tip.

SetTip(match, playedMatches);

}

}

}

}

### **Back Office**

### **Front Office**

#### Vanilla Javascript e CSS 3 puro.

Toda a aplicação Front Office utiliza apenas Vannila Javascript e CSS 3 puro, uma decisão que provavelmente atrasou o desenvolvimento geral da aplicação, mas que foi uma decisão consciente, e que resultou numa aprendizagem adicional muito grande, que de outra forma seria muito difícil de alcançar.

Exceção são os botões e os inputs que utilizam Bootstrap 4.

#### Geração dinâmica de conteúdos

Devido a não utilização de templates ou plug-ins, todos os conteúdos do Front Office, quer sejam os menus do utilizador, os modais, ou ainda as tabelas apresentadas nas diferentes páginas da aplicação, são tudo conteúdos gerados de forma dinâmica, dependendo de diversos aspetos, como por exemplo: dimensão do dispositivo de acesso à aplicação, se o utilizador é um utilizador registado ou não, ou ainda consoante a página em que o utilizador se encontra num determinado momento.

#### Validações

Todas as validações necessárias no Front Office, são, numa primeira instância realizadas via Javascript, e só quando todas as validações são positivas, e apenas se necessário, se realiza um Post Back ao servidor. Ainda assim é sempre feita uma segunda validação no servidor.

#### AJAX

O Front Office faz recurso de AJAX em situações especificas como por exemplo, quando um utilizador está a escolher o seu username, o Front Office faz validação via AJAX para informar o utilizador on-the-fly se o username é válido. O mesmo acontece com o email, com o login, recuperar password etc… evitando-se chamadas desnecessárias ao servidor com Post Backs.

#### Regex

Ainda que de forma subtil, foi utilizado Regex para validação de emails.

#### Responsividade

Um dos requisitos mais importantes do projeto SPAD, era que este fosse acessível via dispositivos móveis. Como tal, foi dada grande enfase à responsividade do Front Office, estando este totalmente responsivo na página Home (a mais importante) e na página Tips History (segunda mais importante). As restantes páginas ficaram parcialmente responsivas.

#### Imagens com preview sem necessidade de servidor

Uma das features que poderá ser diferenciadora no Front Office, é a funcionalidade de preview de imagem que o utilizador escolher no seu perfil, sem necessidade de a guardar previamente no servidor. Esta funcionalidade foi alcançada via Javascript.

#### Modals personalizados

No sentido de melhorar a experiência de utilização da Web Application e de conseguir alcançar um design diferenciado, foram desenvolvidos vários modelos de modais para o Front Office, tendo sido para tal necessário o recurso à manipulação de HTML, CSS e Javascript.

#### Interação com tabelas

As funcionalidades das diferentes tabelas do Front Office, como ordenação por coluna, filtragem de conteúdos ou ainda a conversão automática da Utc Date para Local Date, foram alcançadas através da manipulação da DOM via Javascript.

#### Bubble Sort

Sendo que o Front Office é totalmente gerado de forma dinâmica, sem recurso a qualquer template ou plug-in, e sendo o mesmo verdade para as tabelas que nele se encontram, tornou-se imperativo desenvolver uma forma de aplicar ordenação por coluna, para as diferentes tabelas. Assim, o algoritmo escolhido, por simplicidade e rapidez de implementação, foi o Bubble Sort.

#### Utc Date to Local Date

O Front Office é uma Web Application que inevitavelmente lista vários conjuntos de informação, onde a data é extremamente importante. Por conveniência todas as datas são guardadas no base de dados no formato Utc Date. Ter a datas neste formato possibilita a conversão da mesma para qualquer data de uma região diferente, sendo apenas necessário saber a hora padrão local.

Assim foi desenvolvido do lado do cliente (pois queremos converter a Utc Date para Local Date, e apenas estando do lado do cliente conseguimos saber, através de informações do seu sistema, qual é a sua hora padrão local) um método em Javascript, de nome UtcDateToLocalDate(), que acuta imediatamente após qualquer tabela ser gerada. Assim o cliente vê sempre a data dos jogos, que vão decorrer, por exemplo, na sua hora local, sem necessidade de o utilizador intervir na aplicação.

## Descrição do uso da aplicação e suas funcionalidades

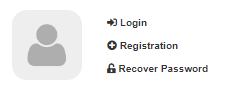
Nota: Utilização de caso de sucesso principal

### Header

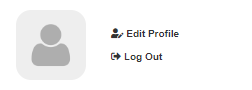


Figura 1 – Header

* O header, que está presente na maioria das páginas do FrontOffice, serve como ponto central de navegação, possibilitando sempre o acesso às seguintes páginas:
  + *Tips*;
  + *Tips History*;
  + *About Us*;
  + *Contact Us*.
* Este dá também acesso condicional às seguintes páginas:
  + Caso o Utilizador não tenha sessão iniciada, em que o painel de Utilizador terá uma imagem escolhida pelo próprio ou um *placeholder* caso o mesmo não tenha escolhido nenhuma:



* + - Login, que permite ao Utilizador iniciar sessão, caso esteja registado e aprovado;
    - Registration, que permite ao Utilizador criar uma conta;
    - Recover Password, que permite ao Utilizador recuperar as credenciais através de E-mail;
  + Caso o Utilizador tenha sessão iniciada:



* + - Edit Profile;
    - Logout;

### Página *Tips*

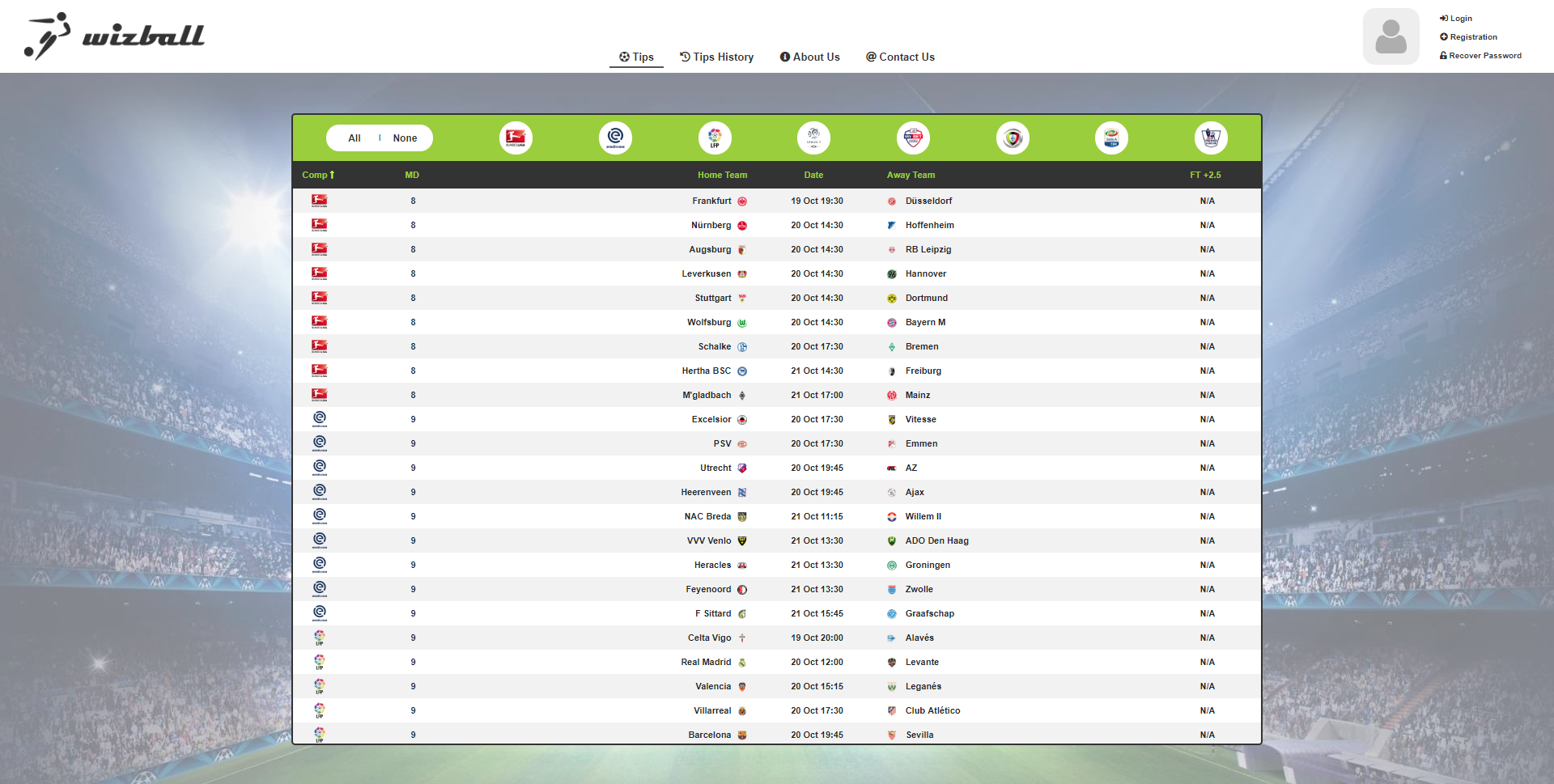


Figura 2 - Página "Tips"

* A página inicial que o Utilizador visita contém partidas futuras, e prognósticos para as mesmas, tendo o Utilizador a possibilidade de filtrar as partidas por campeonato(s), clicando no ícone do(s) mesmo(s), e ordená-las por campeonato, data, ou previsão, clicando no cabeçalho específico de cada coluna para alterar a seleção ou a ordem da mesma.

### Página *Tips History*

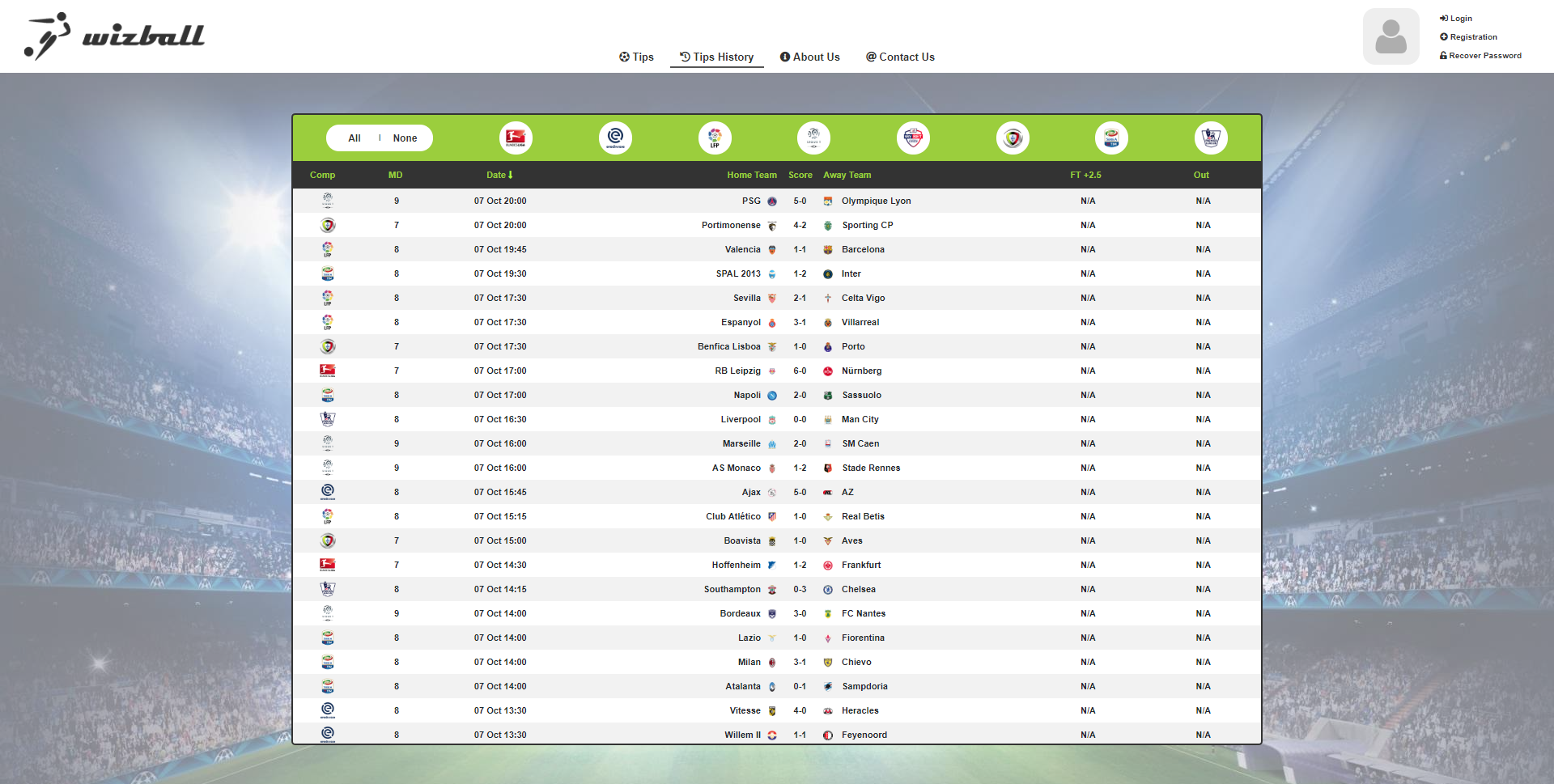
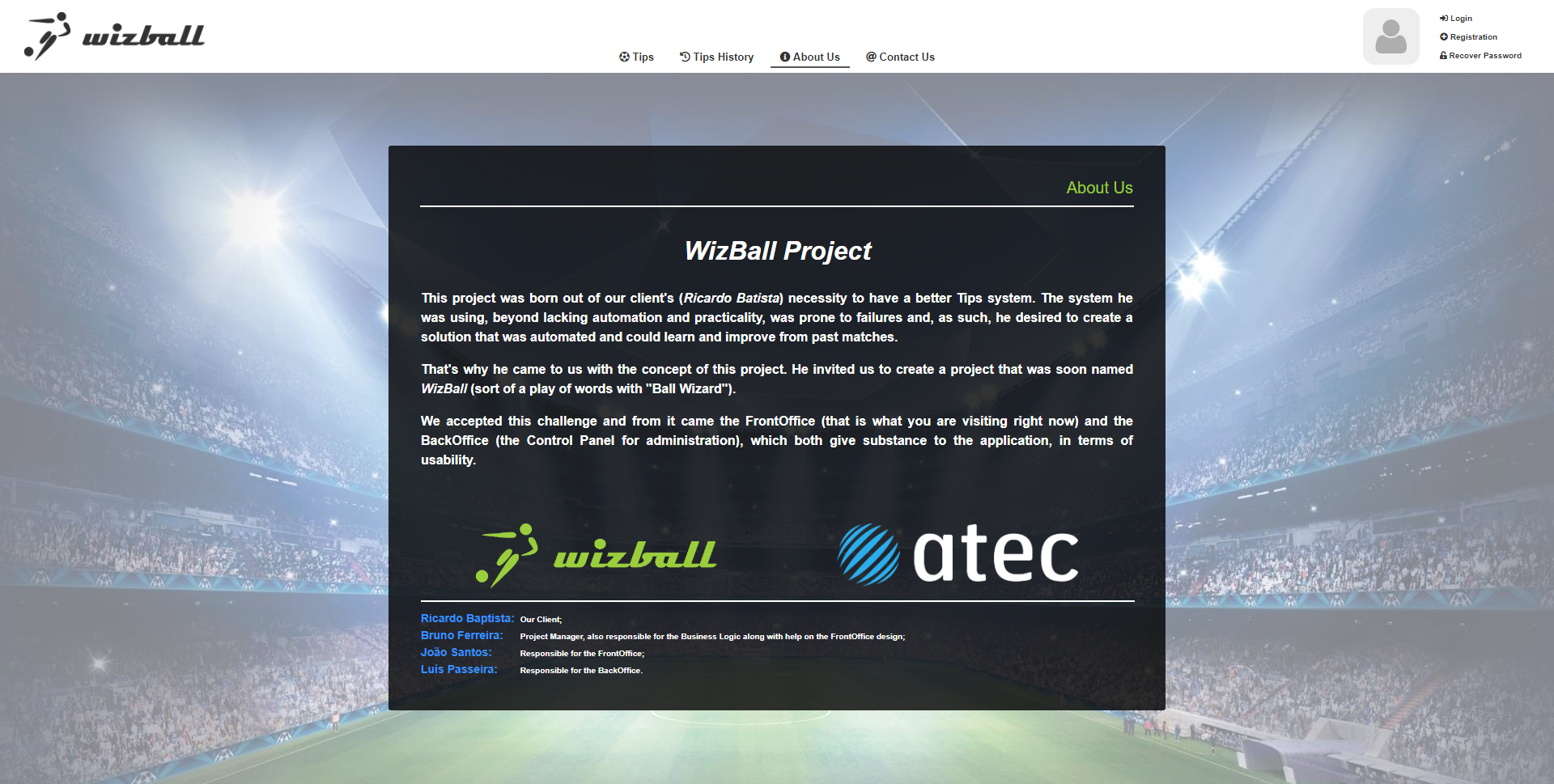


Figura 3 - Página "Tips History"

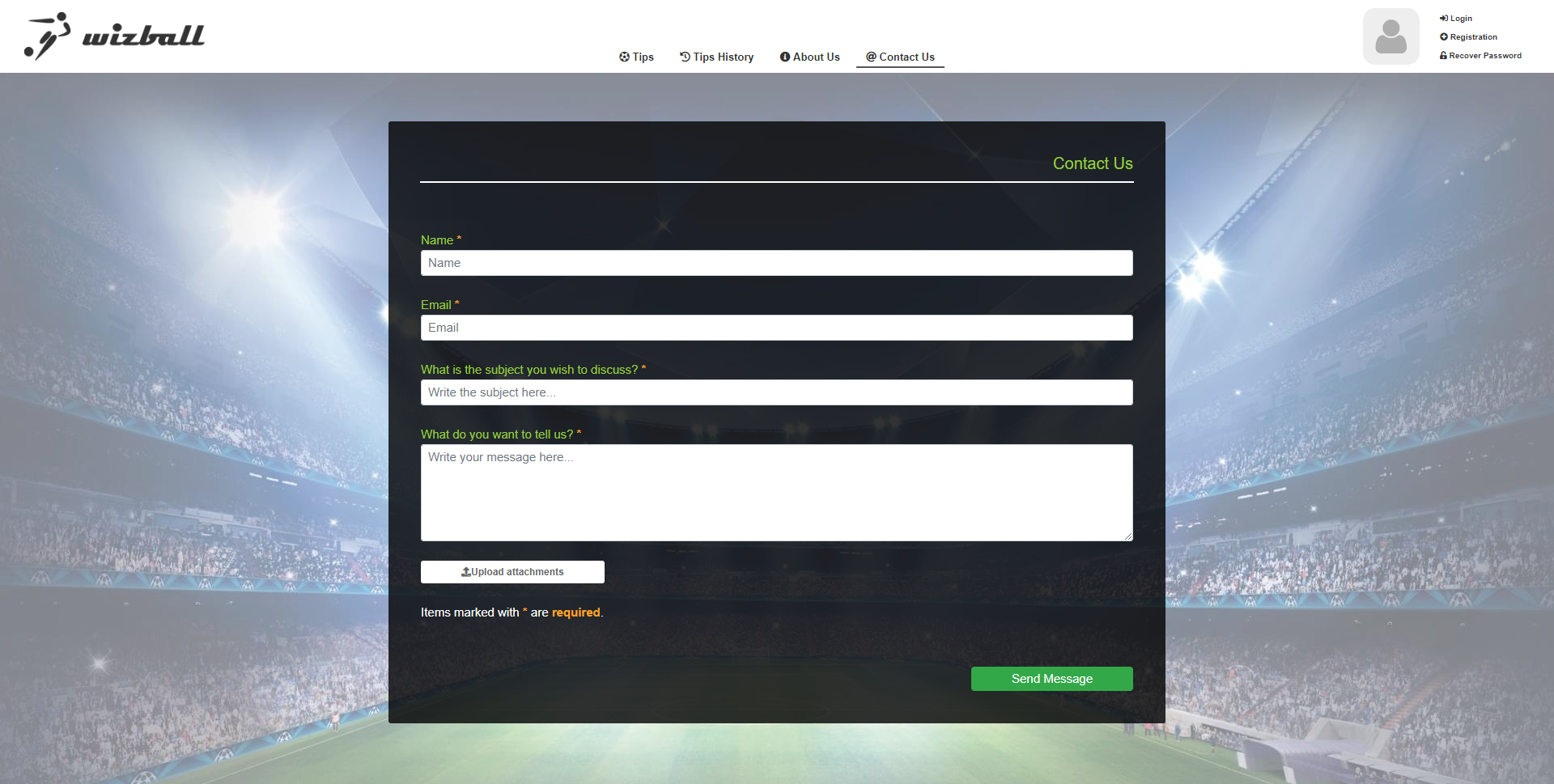
* Dentro da página de histórico de prognósticos, o Utilizador tem a possibilidade de visualizar todas as partidas que a aplicação já analisou e visualizar quais foram os prognósticos que esta deu, comparando-os assim com os resultados efetivos de cada equipa.

### Página *About Us*

Figura 4 - Página "About Us"

* Esta página dá uma breve introdução à motivação do projeto, e os membros nele envolvidos, bem como as suas posições no mesmo.

### Página *Contact Us*

Figura 5 - Página "Contact Us"

* A página Contact Us contém um formulário disponível para o Utilizador contactar a Administração, para colocar questões, dar feedback, etc.;
* Permite anexar ficheiros, como por exemplo screenshots, para o caso de o Utilizador querer reportar um bug, ou outra situação que requira pequenos ficheiros;
* O Utilizador também receberá um E-mail automático com a mensagem que enviou;

# Conclusão

Quando penso nas conclusões do projeto SPAD, o primeiro pensamento que nos surge é que, foi um projeto que se revelou maior do que o antecipado.

Embora como grupo tenhamos dispensado bastante tempo na conceptualização e arquitetura do projeto, sinto que demos muita estrutura a coisas demasiadas pequenas, e outros assuntos, como por exemplo a estrutura de base de dados, relativamente aos prognósticos foi um pouco negligenciada.

Faltou um pouco de antecipação e experiência para perceber este tipo de nuances, que, aparentemente são impercetíveis quando o projeto está numa fase embrionária, mas que, à medida que o projeto vai crescendo, vão se tornando cada vez mais proeminentes e limitadoras, e principalmente complicadas de mudar a meio de um projeto com tempo contado.

Embora com algumas limitações de conceptualização, foi ainda assim relativamente fácil de alcançar os objetivos inicialmente propostos. No entanto, temos consciência que se fosse necessário que o projeto ganhasse uma dimensão superior, em certos aspetos teríamos de adaptar o projeto de tal forma, que teríamos necessidade realizar alterações até à base de dados.

Relativamente ao cumprimento dos objetivos propostos, acredito que alcançamos tudo a que nos propusemos no inicio do projeto, mas, acredito também que as várias funcionalidades implementadas, apesar de concluídas, poderiam estar mais requintadas. Por exemplo a responsividade ficou parcialmente implementada, a tabela que mostra o histórico de prognósticos pediria ter um rodapé com estatísticas, os sincronizadores poderiam evidenciar as diferenças entre a API e a Base de dados, entre muito outros pequenos detalhes que trariam outro brilho à aplicação no seu geral.

## Objetivos cumpridos

**- Obter dados de eventos desportivos de futebol de forma automática (Web-API).**

Cem por cento concluído. Está totalmente automatizado com recurso à Web Api football-data.org, e ao desenvolvimento de diversas classes Resources, já previamente explicadas. Esta funcionalidade pode ser utilizada através do Back Office.

**- Validar a fidedignidade dos dados obtidos através da Web-API.**

Cem por cento concluído. Implementado com recurso a várias classes e métodos que comparam se os valores que vêm da API são os antecipados, isto é, se tipo de dados é válido ou se existe integridade referencial, entre outros. Estas validações são realizadas na BLL, também já previamente explicado o seu funcionamento.

**- Persistir os dados no nosso sistema (Base de dados MS SQL Server).**

Cem por cento concluído. Implementado com recurso às várias classes DAL.

**- Aplicar algoritmos estatísticos sobre os dados recolhidos.**

Cem por cento concluído. Este passo dá-se também na BLL, e é despoletado através do Back Office. No entanto acredito que é importante ressaltar que, desenvolver um algoritmo que consiga ter uma taxa de acerto superior aos 60% é realmente muito difícil.

**- Guardar o histórico dos prognósticos gerados pelo sistema.**

Cem por cento concluído. Este processo é umas das funcionalidades disponibilizadas pelo Back Office, e ocorre através da BLL que por sua vez utiliza à DAL.

**- Permitir o registo de utilizadores no sistema e a sua gestão.**

Cem por cento concluído. Esta é uma funcionalidade que necessita tanto da Business Logic, Front Office e Back Office, uma vez que o utilizador se regista no Front Office, é autorizado no Back Office, e ambas utilizam métodos disponibilizados na BLL e nos WS, que por sua vez usam a DAL.

**- Permitir a criação e envio de newsletter para utilizadores registados.**

Cem por cento concluído. Esta funcionalidade faz parte do conjunto de funcionalidade do Back Office, e os métodos que utiliza estão implementados na BLL.

**- Permitir a consulta do histórico dos prognósticos gerados pelo sistema.**

Cem por cento concluído. Uma das funcionalidades principais do sistema, e que está disponível através do Front Office. Para tal foi necessário desenvolver todo o conjunto de classes Resources, classes DAL, Entities, Sincronizadores, Validações e Geradores de Prognósticos, até se poder ter dados para apresentar ao utilizador.

**- Garantir o acesso ao Front Office através de dispositivos móveis.**

Diria que sensivelmente sessenta por cento ficou implementado, uma vez que as duas páginas mais importantes (Home e Tips History) estão totalmente responsivas, e as restantes, estão parcialmente responsivas (o necessário para o utilizador conseguir interagir com a aplicação, mas a experiência de utilização está medíocre.

**- Criar duas Aplicações Web (Front Office; Back Office) que permitam a automatização e a fácil gestão de todo o sistema, bem como a consulta dos dados gerados de forma simples.**

Cem por cento. Tudo o que nos comprometemos, de uma forma ou de outra, está implementado.

## Contratempos

### Emblemas das equipas e das competições do SAPD

Sendo o SPAD uma Web Application que visa listar uma série de eventos desportivos de futebol, fazia todo o sentido que cada equipa, competição e país, tivessem o seu respetivo emblema/bandeira associados. Como os nossos dados provêm todos da API Football-data.org, e esta API não disponibiliza os respetivos emblemas/bandeiras, nem dos países, nem das competições, e nem das equipas, tivemos de procurar na internet packages com estas imagens, para posteriormente podermos associar cada emblema/bandeira a cada equipa, competição e país, para posteriormente apresentar a informação não apenas num formato de texto mas também num formato mais visual e apelativo para o utilizador.

Além da procura dos emblemas/bandeiras tivemos de criar um script em T-Sql que associasse cada imagem à respetiva equipa/competição/país. É ainda de salientar que por motivos de consistência, as imagens deveriam satisfazer as seguintes condições: tamanho 256\*256 ou 128\*128 e serem transparentes, o que por si só dificultou ainda mais o processo de obtenção das imagens. Outro fator que tornou o processo ainda mais demorado, foi o facto de não conhecermos todos os emblemas das equipas e competições, pelo que tivemos de ter a certeza que um determinado emblema pertence efetivamente a uma determinada equipa.

exemplo

update teams set flag = 'benfica.ico' where id = 1903;

update competitions set flag = 'primeira\_liga.png' where id = 2017

Este processo foi totalmente realizado de forma manual, para sensivelmente 200 equipas/competições diferentes.

Pode-se imaginar o trabalho que envolveu, encontrar/obter da internet as diferentes imagens/emblemas, identificar um determinado registo de uma equipa pelo seu id, posteriormente saber qual nome da imagem que este registo vai ficar associado, e ainda ter a certeza de que não nos enganamos no processo de escrita do script.

Foi de facto um “pequeno” contra tempo.

## Dificuldades Encontradas

### Datas e conversões

As datas no SPAD são um tema de grande importância, pois a finalidade desta aplicação é listar prognósticos de eventos desportivos de futebol, que por sua vez acontecem numa determinada data. No entanto essa data tem diferentes representações dependendo de onde o evento vai ser realizado (Portugal, Alemanha, Itália etc…), e ainda do local no planeta, onde os utilizadores estão a aceder à nossa aplicação.

Após alguma investigação, tornou-se obvio que a forma de alcançar uma consistência na forma como apresentamos as datas na aplicação foi com recurso ao formato UTC.

Todas as datas são guardadas no servido no formato UTC Date. Com esta pequena alteração no nosso modelo conceptual, o processo de conseguir apresentar as datas de forma consistente e segura tornou-se bastante simplificado.

### Obtenção de dados para aplicação de forma automatizada e API Football-data.org

Como já foi referido, o SPAD vive de dados, obter de dados, manipular de dados, gerar novos dados etc…

Era, portanto, imperativo para o sucesso do projeto, encontrar uma forma de obter dados de forma fidedigna, consistente, segura, rápida, mas acima de tudo de forma automatizada.

Uma vez mais, e após alguma investigação, encontramos um serviço gratuito, que nos permite através de pedidos HTTP consumir uma REST API.

A API Football-data.org, foi, portanto, o nosso ponto de partida. Na altura em que começamos o projeto, e a implementação das classes que iriam “consumir” a API, ainda não tínhamos falado sobre APIs no curso, pelo que foi um processo de aprendizagem do zero.

### Performance geral da aplicação

A medida que o SPAD foi crescendo, apercebemo-nos imediatamente que teríamos de ter diversas dificuldades a nível da performance da aplicação. No sentido de evitar o deterioramento da performance da aplicação, definimos o seguinte:

1. Instruções SQL:
   1. Apenas requisitar os dados que realmente façam sentido.
   2. Otimizar as querys de forma a evitar inner selects.
   3. Otimizar as querys de forma a evitar inner joins.
   4. Não realizar validações desnecessárias
   5. Converter para Stored Procedure as instruções mais complexas.
   6. Guardar imagens no servido web e deixar nas tabelas apenas o caminho para a imagem correspondente no servido web.
2. Tamanho das imagens:
   1. Imagens de fundo foram drasticamente reduzidas de tamanhos acima de 2Mb para 100Kb.
   2. As imagens que os utilizadores podem escolher para o seu perfil têm tamanho máximo de 500Kb.
3. Manipulação DOM
   1. Evitar estar constantemente a correr a DOM à procura de um determinado elemento. Se esse elemento é constantemente necessário, é criada uma referência para esse elemento a nível global.
   2. Otimização de algoritmos de sorting.
   3. Evitar animações.
4. Post Backs
   1. Evitar post backs, implementando sempre em primeira instância validações Javascript.
   2. Evitar post backs, recorrendo a chamadas AJAX, quando apropriado.

### Mapeamento do objetos da API com os objeto que mapeiam a base de dados

Conseguir de forma consistente obter dados “objetos” da API, e usar esses mesmos objetos no resto da Business Logic foi também um ponto que nos deu algum trabalho conceptual.

A forma que encontramos foi: Criar uma classe do tipo Entity, que tanto se adapta ao formato de dados que provêm da API, como se adapta ao nosso modelo de dados. Desta forma conseguimos uniformizar o tipo de objetos com que toda a Business Logic trabalha.

## Apreciação pessoal