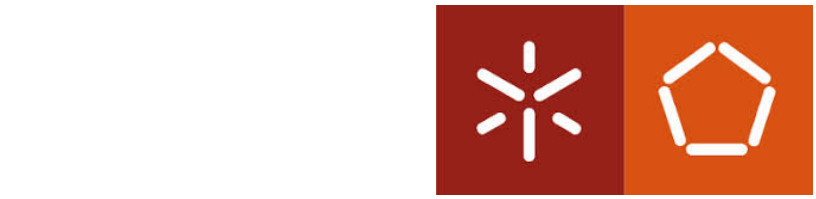

Universidade do Minho
Mestrado em Engenharia Informática

Requisitos e Arquiteturas de Software

Fase 2

13 de dezembro de 2021





António Santos (PG47031)
Manuel Moreira (PG47439)
Maria Marques (PG47489)
Pedro Pereira (PG47584)
Sara Dias (PG47667)

Conteúdo

Lista de Figuras

Lista de Tabelas

1 Introdução e objetivos

RASBet é um projeto que está a ser desenvolvido com o propósito de criar um suporte a casas de apostas *online*. Este permite aos seus utilizadores realizar apostas nos seus eventos desportivos favoritos de uma forma fácil e segura. De momento, a RASBet suporta desportos coletivos e individuais, sendo eles o futebol e o ténis, respetivamente.

Com este projeto, pretendemos atingir alguns objetivos:

- O sistema deverá suportar múltiplos utilizadores.
- O sistema deverá suportar diferentes tipos de eventos desportivos.
- O sistema deverá permitir a visualização, em tempo real, de uma lista de apostas ativas consoante o desporto.
- O sistema deverá suportar múltiplas apostas por parte dos utilizadores do sistema.
- O sistema deverá suportar transferências bancárias seguras.

1.1 Visão Geral dos Requisitos

Id	**Requisito**	**Explicação**
F1	Registar	O sistema deve permitir que os utilizadores se registem na aplicação através de um *email*, *username*, data de nascimento e *password*.
F4	*Login*	O sistema deve permitir que os utilizadores iniciem sessão nas suas contas através do seu *email* e *password*.
NF21	Alterar Idioma	O sistema deve suportar uma interface do utilizador multilingue.
F5	Depositar Dinheiro	O sistema deve permitir que os utilizadores façam depósitos na sua conta.
F6	Levantar Dinheiro	O sistema deve permitir que os utilizadores façam levantamentos do saldo disponível.
F12	Realizar Aposta	O sistema deve permitir que os utilizadores façam apostas.
F13	Visualizar Histórico	O sistema deve permitir que os utilizadores tenham acesso ao seu histórico de apostas efetuadas.
F2 F3	Editar Perfil	O sistema deve permitir que os utilizadores registados alterem a sua *password* e *username*.

Tabela 1: Requisitos Prioritários

Para uma melhor compreensão aconselha-se à leitura do relatório da Fase 1, mais especificamente o capitulo 3, sub-capitulo 3.3 e o capitulo 4.

1.2 Objetivos de qualidade

Nr	Qualidade	Motivação
1	Utilidade	O sistema suporta funções que atendem às necessidades declaradas no tópico acima mencionado.
2	Manutenção	O sistema pode ser modificado, corrigido, adaptado ou melhorado devido a mudanças no ambiente ou requisitos.
3	Usabilidade	O sistema pode ser compreendido, aprendido, usado, e é apelativo para os usuários.
4	Segurança	A aplicação deve fornecer proteção do sistema contra acesso, uso, modificação, destruição ou divulgação acidental ou mal-intencionada.
5	Eficiência	O produto deve fornecer o desempenho adequado, em relação à quantidade de recursos usados, nas condições estabelecidas.

Tabela 2: Objetivos de qualidade

1.3 Stakeholder

A lista a seguir contém as entidades mais importantes para esta aplicação:

Nome	Função	Expectativas
Developers	Desenvolver a aplicação	*Developers* que desejam aprender como desenvolver aplicações modernas e vários *front-ends*, de preferência usando Java 8 no *back-end*.
Casas de apostas	Incorporar a aplicação no seu negócio	Aumentar o seu número de apostadores anuais.
Utilizadores	Usufruir da aplicação	Entusiastas de deporto e de apostas *online* que pretendem fazer as suas apostas num ambiente simples e seguro.
InsighT	Investir no desenvolvimento da aplicação	Aumentar o valor da empresa ao entrar numa nova área tecnológica com margem para grandes lucros.

Tabela 3: Stakeholder

2 Restrições

2.1 Restrições Técnicas

Id	Restrição	Motivação
RT1	Implementação em Java	A aplicação deve ser desenvolvida utilizando o Java 8. A interface (ou seja, a API) deve ser independente de linguagem e estrutura.
RT2	Desenvolvimento independente do sistema operacional	A aplicação deve ser compilável em todos os três sistemas operativos principais: MacOS X, Linux e Windows.
RT3	Model-View-Controller (MVC)	A aplicação deve ser implementada segunda a arquitetura Model-View-Controller.

Tabela 4: Restrições Técnicas

2.2 Restrições e convenções organizacionais

Id	Restrição	Motivação
CO1	Prazos de Entrega	19 de novembro de 2021: entrega de toda a planificação inicial necessária do projeto. 17 de dezembro de 2021: conclusão de toda a modelação do sistema e dos seus componentes. 21 de janeiro de 2022: produto final pronto para a apresentação ao público.
CO2	Configuração e controle e gestão de versões	Repositório *github* privado com um histórico de *commits* completo.

Tabela 5: Convenções Organizacionais

2.3 Convenções

Id	Convenção	Motivação
C1	Documentação de arquitetura	Estrutura baseada no modelo arc42 em português.
C2	Convenções de codificação	O projeto usa as convenções de código para a linguagem de programação Java.
C3	Idioma	Multilingue. O projeto é direccionados a um público internacional, portanto foi desenvolvida a possibilidade de escolha do idioma na aplicação. No entanto, uma vez que a empresa financiadora se trata de uma empresa nacional, o português foi usado em toda a documentação do projeto.
C4	Convenções de nomenclatura	Existem algumas convenções de nomenclatura que podem ser encontradas tanto no capitulo 2 ,sub capitulo 2.2, do relatório da 1ª fase como aqui .

Tabela 6: Stakeholder

3 Contexto e alcance

Esta secção descreve o ambiente do RASBet: quem são seus utilizadores e com quais outros sistemas ele interage.

3.1 Contexto empresarial

3.1.1 Utilizador

Um utilizador da plataforma é alguém interessado em apostar em eventos desportivos do seu interesse. Estes utilizadores podem realizar qualquer tipo de ação relacionada com fazer uma aposta, tal como verificar todos os eventos disponíveis, seguir o seu historial de apostas e até mesmo movimentar o seu saldo. Outra ação possível não ligada às apostas em si consiste em poderem alterar os seus dados pessoais caso assim o desejem.

3.1.2 RASBet

Plataforma encarregue de processar os pedidos do utilizador de forma eficiente e correta.

3.1.3 APIs

Sistemas externos que fornecerão ao sistema informação atualizada e útil para o seu funcionamento.

Atualiza de forma autónoma e disponibiliza, num ficheiro JSON, toda a informação relativa a um evento. É importante referir que são utilizadas duas APIs distintas para suportar a plataforma: uma fornece-nos a informação necessária sobre eventos desportivos de futebol e a outra é responsável pela informação sobre eventos desportivos de ténis.

A API utilizada para futebol é disponibilizada pela plataforma API-Sports [1] e a API utilizada para ténis encontra-se disponível em Rakuten RapidAPI[2].

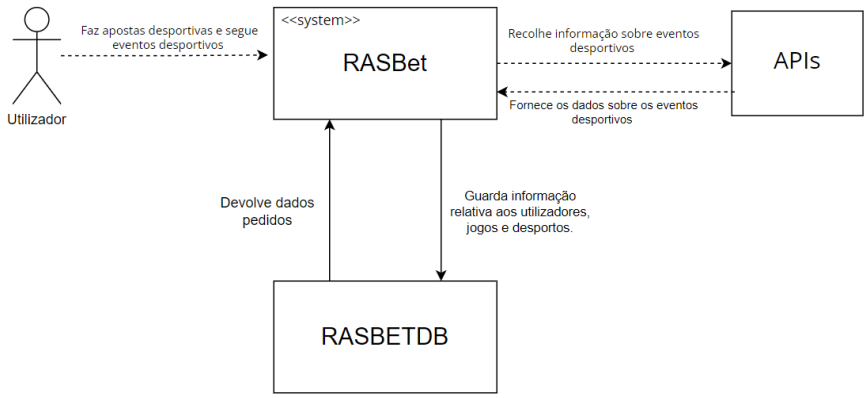

Figura 2: Diagrama de Contexto do Negócio

4 Estratégia de Solução

No processo de formulação de qualquer aplicação, é fundamental ter em conta as estratégias e recursos que serão necessários, uma vez que sem eles é impossível implementar o produto final.

Para isso, apresentamos de seguida uma breve enumeração daqueles considerados imprescindíveis para conseguir um bom produto final:

1. A aplicação será feita na linguagem de programação Java.
2. Várias ferramentas serão utilizadas, tanto no desenvolvimento da aplicação como no seu posterior uso, das quais se salientam: Visual Paradigm, IntelliJ, SceneBuilder.

[1] https://api-sports.io/
[2] https://english.api.rakuten.net/BettingAPI/api/tennis-odds/endpoints

3. Utilização de APIs externas que nos irão fornecer dados sobre jogos de futebol e ténis, e as suas respetivas *odds*. [3] [4]

4. Utilização de uma base de dados para preservar a informação pessoal de cada utilizador, bem como informação sobre os eventos desportivos e respetivas *odds* para apostas.

5. Separação da aplicação por diferentes subsistemas, tendo cada um deles diferentes responsabilidades, como por exemplo: gestão dos utilizadores, gestão dos eventos e comunicação com o utilizador.

5 Vista de Bloco de Construção

Como é frequente em projetos deste tipo, a aplicação pode-se tornar bastante complexa bastante rápido, principalmente numa fase futura, caso sejam adicionadas funcionalidades novas no sistema ou sejam modificadas as previamente feitas.

Nesse caso, é necessário dividir o sistema nos seus diferentes componentes e tentar manter as coisas simples e fáceis de gerir, impedindo assim a dificuldade de ter a enorme quantidade de código e funcionalidades num único sítio.

Desta forma e relembrando o diagrama de contexto empresarial apresentado no capitulo 3, que representa o nível 0 da visualização da construção por blocos, passamos a apresentar os restantes níveis.

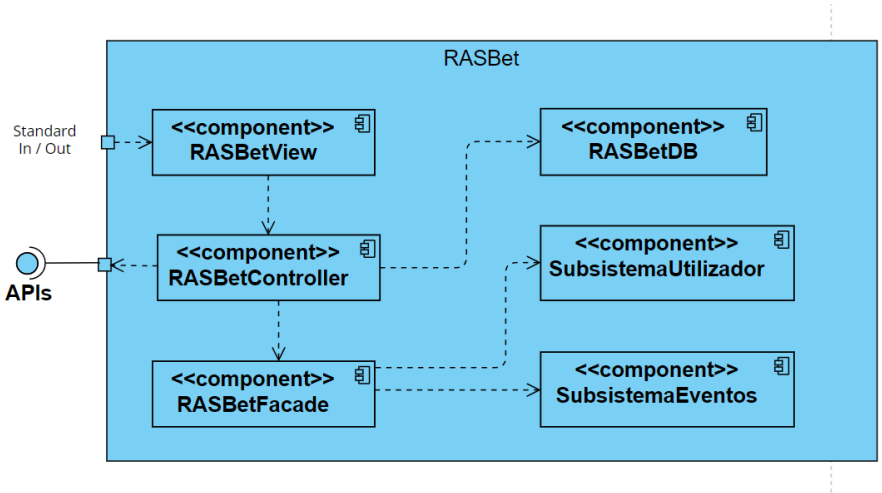

Figura 3: RASBet, visualização do bloco de construção, nível 1

O sistema RASBet divide-se em dois principais subsistemas: SubsistemaUtilizador e SubsistemaEventos, conforme apresentado na figura 3. Dentro do sistema, temos ainda classes com responsabilidades distintas, mas todas necessárias para o bom funcionamento do sistema como um todo.

[3] https://api-sports.io/
[4] https://english.api.rakuten.net/BettingAPI/api/tennis-odds/endpoints

As setas tracejadas representam dependências lógicas entre os subsistemas. As caixas quadradas presentes nos limites do sistema RASBet são pontos de interação ("portas") com o mundo externo, isto é, o utilizador e as APIs responsáveis por fornecerem os dados das apostas.

Subsistemas	Descrição
RASBetView	É o elo de ligação entre todo o processamento executado na classe RASBetController e um utilizador. Após um dado *input* por parte do utilizador, este será enviado e processado no RASBetController, e posteriormente o *output* será apresentado ao utilizador através desta mesma componente.
RASBetController	É o "motor" do sistema, fazendo o elo de ligação entre os dados fornecidos por um utilizador na RASBetView e os dados existentes, que serão acessíveis através da RASBetFacade. É também este o componente responsável por estabelecer ligações à RASBetDB, que será hospedada no computador local, e à API.
RASBetFacade	É um dos principais componentes, uma vez que é nele que existe todo o processamento próprio ao sistema de apostas. Os métodos desta classe serão utilizados pelo RASBetController.
RASBetDB	Tal como já foi referido, será o responsável por fornecer informação ao RASBetController acerca dos utilizadores, etc.

Tabela 7: Descrição dos Subsistemas

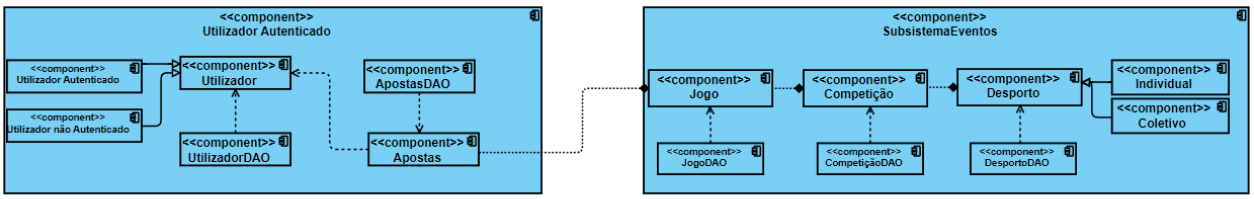

Figura 4: RASBet, visualização do bloco de construção, nível 2

Como se pode observar a partir da imagem acima inserida, referente ao nível 2, o SubsistemaUtilizador apresenta o Utilizador como classe principal (existindo 2 tipos de utilizador, o Utilizador Autenticado e o Utilizado não Autenticado) que está ligado a dois DAOs, UtilizadorDAO e ApostasDAO, responsáveis pela comunicação com a base de dados. Este último DAO encontra-se ligado a uma outra classe, Aposta, tendo esta os métodos responsáveis por cada aposta.

Em relação ao SubsistemaEventos, este possui o Jogo como classe principal, que faz parte de uma Competição que, por sua vez, faz parte de um Desporto, podendo este ser considerado Individual (ténis) ou Coletivo (futebol). Como uma aposta poderá ter um ou mais eventos, foi criada uma ligação entre as duas classes, permitindo assim a conexão entre estes diferentes subsistemas.

Assim, produzimos o seguinte diagrama de classes:

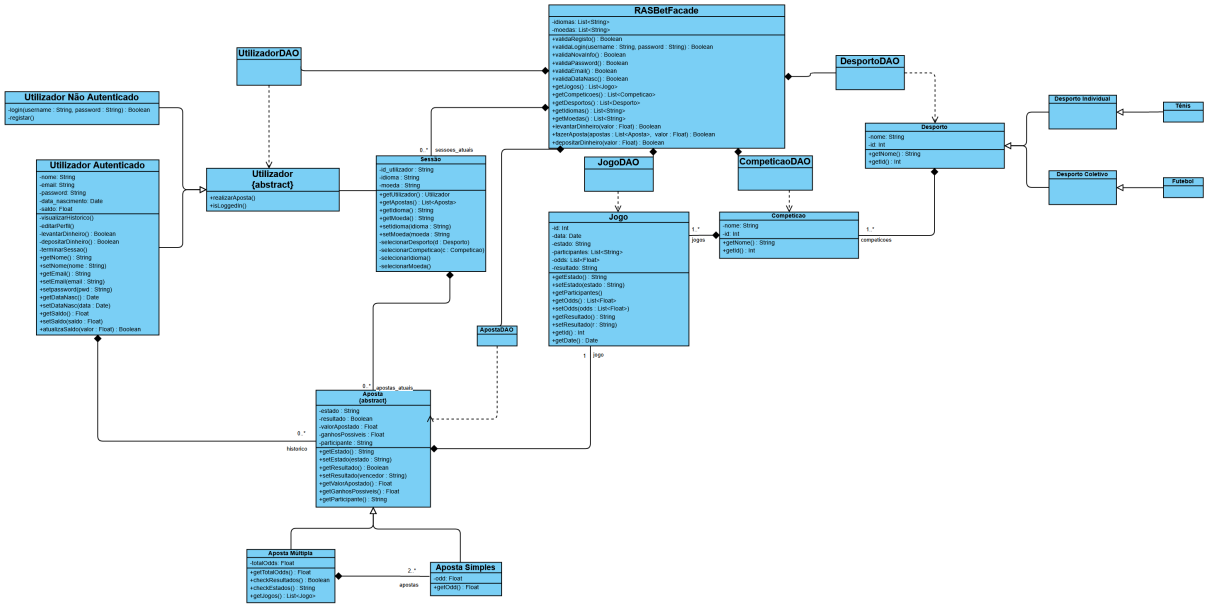

Figura 5: Diagrama de classes

No diagrama, podemos ver que a classe Utilizador é uma classe abstrata, sendo Utilizador Autenticado e Utilizador não Autenticado as subclasses que a especificam, com os atributos adequados a cada uma. Por exemplo, apenas um utilizador não autenticado é passível de fazer *login*.

Semelhante acontece com a classe Aposta. Esta é uma classe abstrata, sendo que as subclasses Aposta Múltipla e Aposta Simples a estendem, com atributos e métodos específicos a cada tipo. Uma aposta múltipla é definida por um conjunto de apostas simples, e consideramos importante distingui-las pela forma como é calculado o resultado da aposta: numa aposta múltipla, o apostador apenas ganha a quantia devida se acertar em todas as apostas que fez.

Temos ainda a classe Sessão que define uma sessão atual de um utilizador na aplicação. Assim, cada sessão está ligada a um utilizador, e este quando, por exemplo, altera o idioma da aplicação, essa modificação apenas se mantém para a sessão atual em que se encontra. Esta classe foi inicialmente pensada para preservar as apostas que estão a ser selecionadas, no caso de ser de um utilizador não autenticado. Quando o utilizador não autenticado decide apostar, é encaminhado para a página de *login*, e com esta classe é possível preservar o estado das apostas que o mesmo pretendia fazer.

As restantes classes do diagrama são bastante intuitivas e foram já mencionadas.

6 Visualização em tempo de execução

Antes de mais, para uma melhor compreensão do funcionamento do nosso sistema, aconselhamos a revisitar os diagramas de atividades e máquina de estados presentes no relatório da primeira fase, no sub-capítulo 3.2.

A interação do utilizador com a RASBet é bastante simples, tal como se pode ver, de seguida, nos quatro casos de uso apresentados abaixo.

6.1 Fazer Aposta

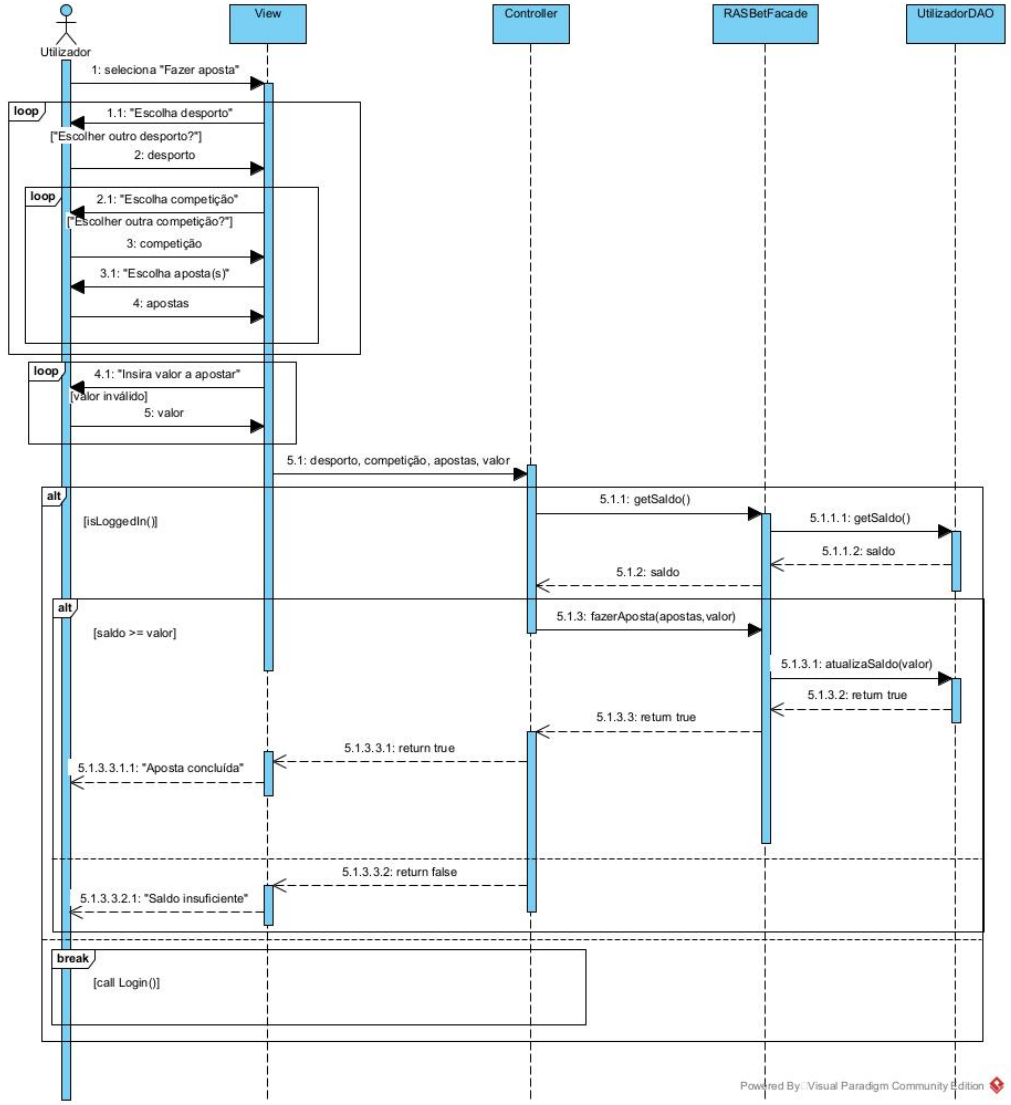

Figura 6: Diagrama de Sequência: Fazer Aposta

Apesar de ser um diagrama de fácil compreensão, apresentamos uma breve explicação do seu significado:

1. O Utilizador seleciona a opção para fazer uma aposta e de seguida tem de escolher o desporto, a competição, os jogos em que deseja apostar e finalmente tem de inserir o valor monetário.
2. O RASBetFacade verifica se o utilizador está autenticado.
3. Se este se encontra autenticado, então verifica-se na base de dados se o utilizador tem saldo suficiente.
4. Se o saldo for suficiente então é feita a aposta, atualiza-se o saldo do utilizador e este recebe uma mensagem de sucesso.
5. Caso o saldo seja insuficiente, o utilizador recebe uma mensagem de erro.
6. No caso de o utilizador não se encontrar autenticado, o *RASBetFacade* chama o método de *login*, pois o utilizador pode realizar uma aposta se e só se encontrar devidamente autenticado.

6.2 Levantar Dinheiro

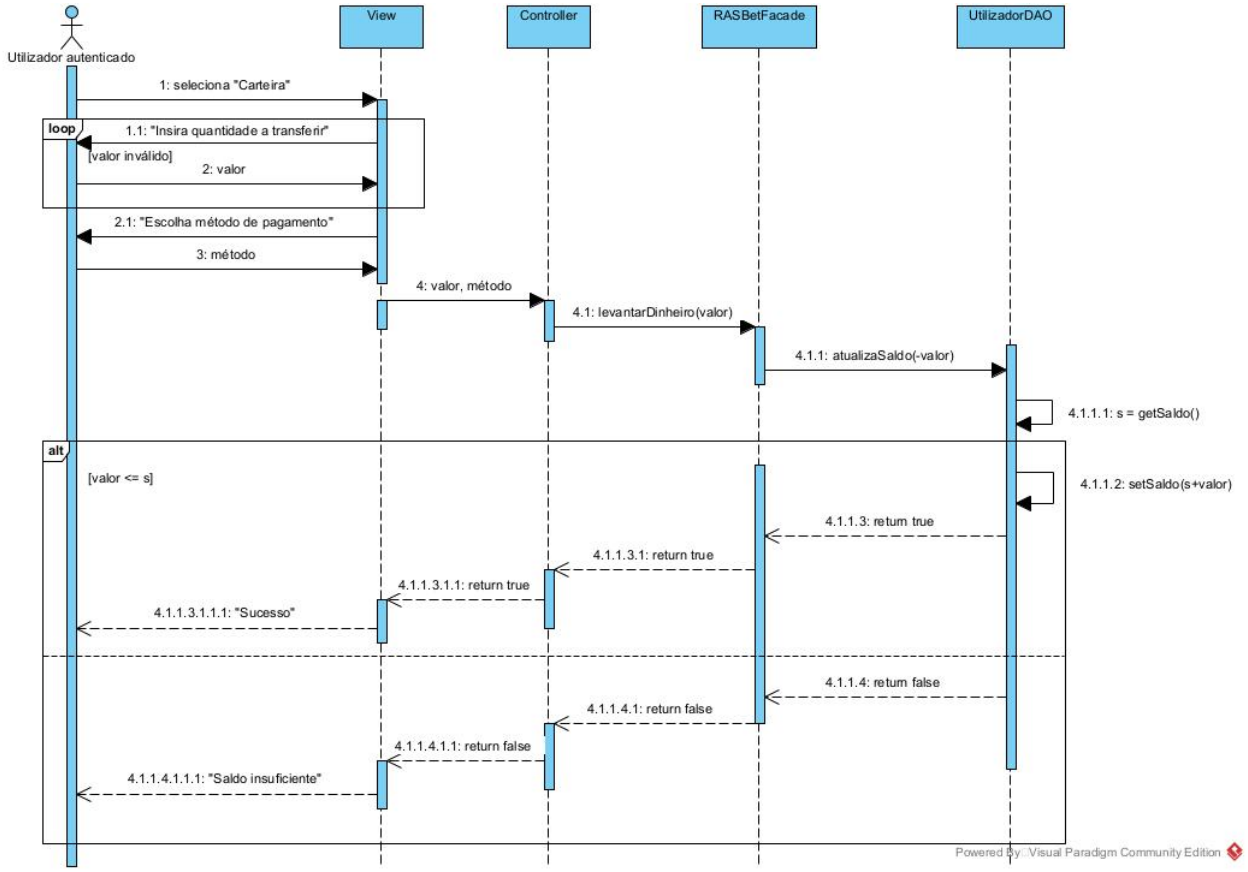

Figura 7: Diagrama de Sequência: Levantar Dinheiro

Apesar de ser um diagrama de fácil compreensão, apresentamos uma breve explicação do seu significado:

1. O Utilizador inicialmente tem de abrir a sua carteira digital.
2. De seguida o utilizador insere o valor que deseja levantar e seleciona o método de pagamento.
3. Se o utilizador tiver saldo suficiente, é chamado o método *atualiza* para que seja removido o valor apostado à sua carteira, e de seguida é enviada uma mensagem de sucesso.
4. Caso contrário, é enviada uma mensagem ao utilizador para que este tenha conhecimento que o seu saldo é insuficiente.

6.3 Depositar Dinheiro

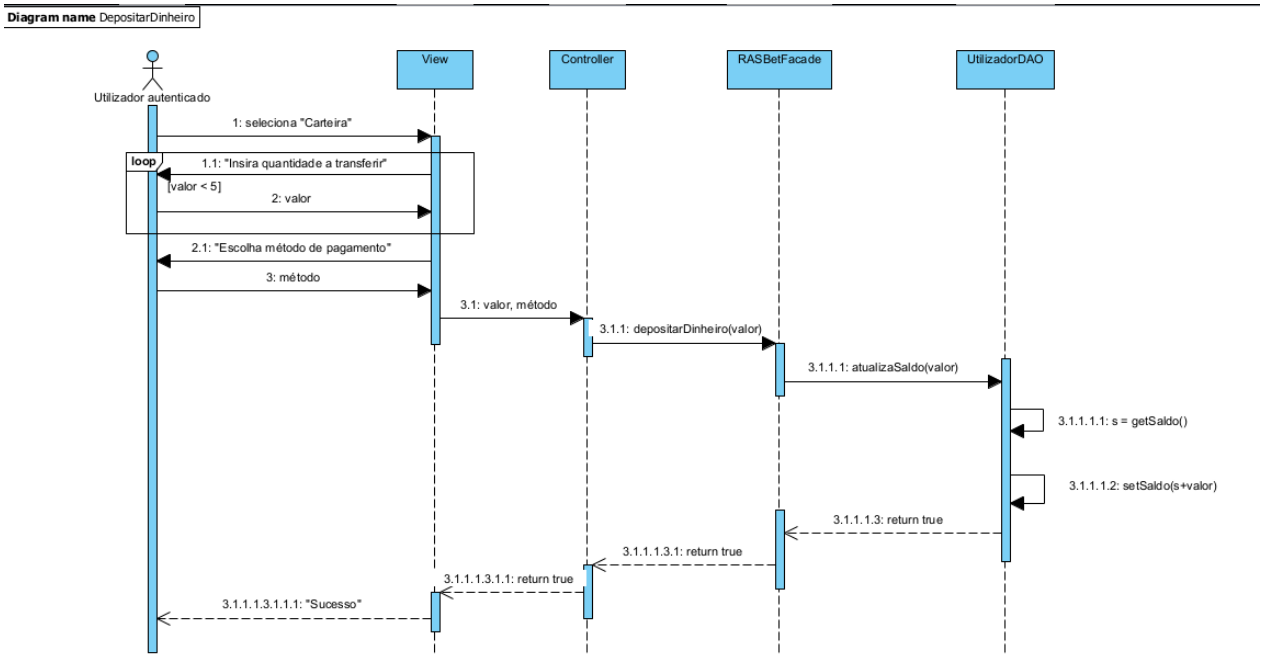

Figura 8: Diagrama de Sequência: Depositar Dinheiro

Apesar de ser um diagrama de fácil compreensão, apresentamos uma breve explicação do seu significado:

1. O Utilizador inicialmente tem de abrir a sua carteira digital.
2. De seguida o utilizador insere o valor que deseja depositar e seleciona o método de pagamento.

3. Se o valor escolhido pelo utilizador for inferior a 5, este não tem a possibilidade de continuar o processo.

4. Caso contrário, é chamado o método *atualizaSaldo* e *setSaldo*, aumentando assim o valor monetário na carteira do utilizador. De seguida, este recebe uma mensagem de sucesso.

6.4 Iniciar sessão

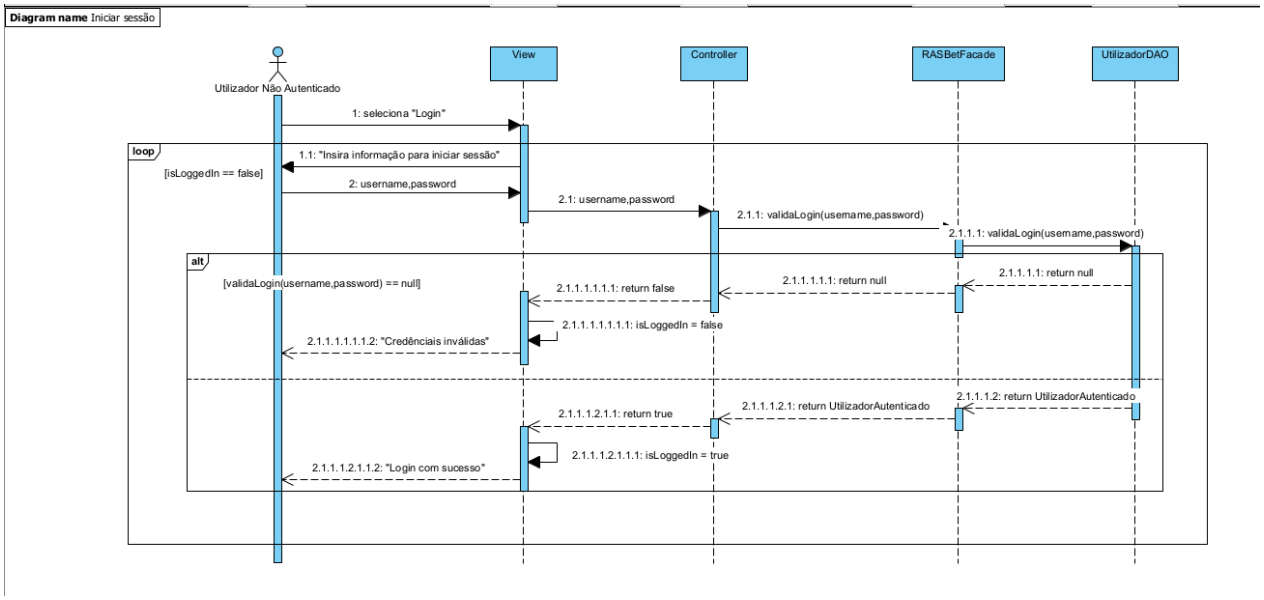

Figura 9: Diagrama de Sequência: Iniciar sessão

Apesar de ser um diagrama de fácil compreensão, apresentamos uma breve explicação do seu significado:

1. O utilizador chama o método *login*, que por sua vez, chama o método *validaLogin* para verificar na base de dados se as credenciais do utilizador (*username* e *password*) estão corretas.

2. Se as credenciais forem inválidas, o utilizador recebe uma mensagem de erro.

3. Caso contrário, recebe uma mensagem de sucesso, entrando assim na sua conta.

7 Visão de implementação

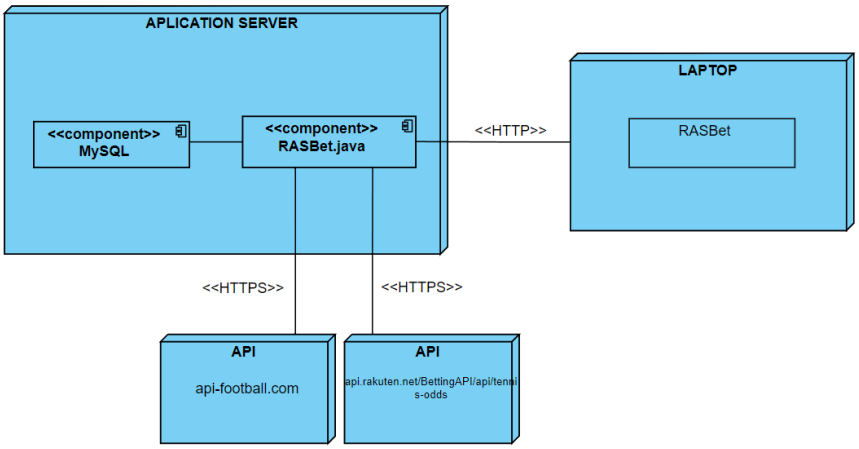

Figura 10: Diagrama de Instalação

Para a nossa aplicação, foi necessário recorrer à criação de um diagrama de implementação, que representa, de maneira geral, como funciona o nosso sistema.

Podemos começar por ver o nodo referente ao *Server*, onde a aplicação irá correr, assim como a base de dados MySQL. Este nodo está conectado, através de ligações HTTPS, à API externa escolhida, que lhe irá fornecer constantemente os dados necessários sobre os jogos e as suas *odds*. O nodo do *Server* está também ligado ao nodo *Web*, que inclui o *web browser*, onde o utilizador, através de HTTP, poderá aceder e interagir com a aplicação.

8 Conceitos Transversais

8.1 Modelo Domínio

Relativamente ao modelo de domínio da aplicação, o mesmo foi mencionado e exposto na 1ª fase do trabalho prático, desta forma aconselhamos a leitura do capítulo 2.2, do relatório da 1ª fase.

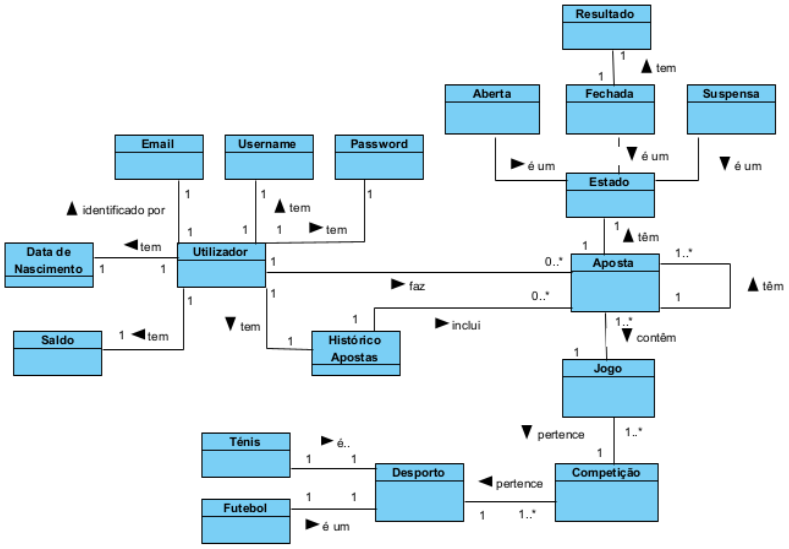

Figura 11: Diagrama Modelo Domínio

8.2 Interface Utilizador

Na primeira fase do projeto, encontram-se especificados variados *mockups* de uma possível implementação gráfica do sistema RASBet.

Desta forma, recomendamos a visualização das mesmas no capítulo 3.2 do relatório da 1ª fase do projeto.

8.3 Tratamento de Exceções e Erros

Os problemas que surjam da utilização do sistema RASBet serão apresentados através do terminal.

Poderão ser levantados dois tipos de erros no decorrer do programa. Um deles verifica-se caso o ficheiro JSON fornecido pela API seja inválido. Já o segundo ocorre caso seja impossível estabelecer uma ligação à base de dados ou à API. Nestes casos, é lançada uma exceção que é tratada no RASBetController. Todos os outros erros que surjam serão resultados de erros de programação, pelo que não é esperado que apareçam.

9 Decisões arquitetónicas

Para a resolução do projeto, será importante aplicar os nossos conhecimentos relativos ao encapsulamento de dados, classes e hierarquias de classes, classes abstratas e interfaces, tal como a implementação do padrão MVC (*Model-View-Controller*).

Para lidar com a persistência de dados será implementada uma interface com o uso de DAOs (*Data Access Objects*) que comunicarão com uma base de dados remota onde estará armaze-

nada toda a informação necessária para a realização de apostas de forma legítima, segura e eficaz.

10 Requerimentos de qualidade

Esta secção contém todos os requisitos de qualidade, apresentados através de uma árvore de qualidade com cenários.

Os mais importantes já foram descritos na secção 1.2.

10.1 Árvore de Qualidade

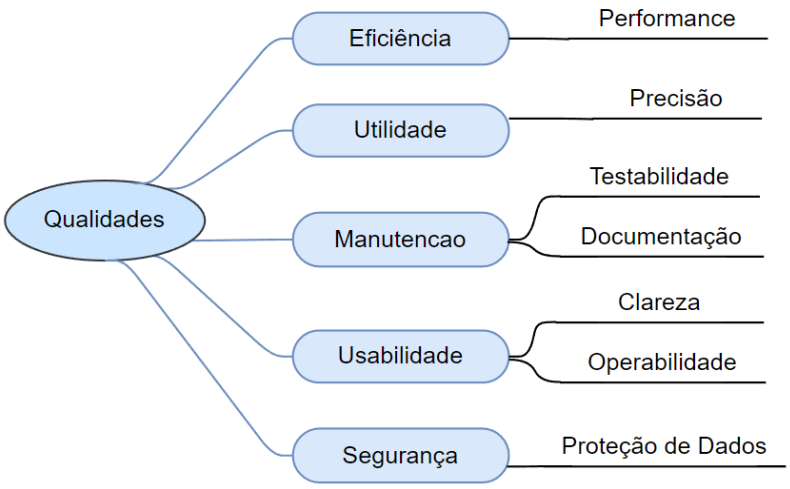

Figura 12: Árvore de Qualidade

10.2 Cenários de qualidade

10.2.1 Eficiência/Performance

Esta característica identifica a capacidade do produto fornecer um tempo de resposta apropriado ao executar a sua função nas condições estabelecidas. É um atributo que pode ser medido para cada funcionalidade do sistema.

Este pode ser medido tendo em conta diversos estímulos. Uma boa forma consiste em medir o tempo de resposta do produto a um estímulo do utilizador, bem como a sua latência.

Exemplo: Um grande número de pedidos chega ao sistema de múltiplos usuários em condições normais.

O sistema deve transferir dados dentro de um determinado período de tempo aceitável.

10.2.2 Utilidade/Precisão

Esta característica identifica a capacidade do produto fornecer funções que atendam às necessidades declaradas e implícitas quando o *software* é usado sob condições especificadas.

Este pode verificar-se nos resultados obtidos, ou seja, se a resposta obtida por parte do sistema foi a esperada. Pode ainda ser medido por um atributo no código, verificando a precisão no valor obtido desse atributo.

Exemplo: O Utilizador pretende fazer uma aposta num determinado jogo que está prestes a terminar.

Como resposta a este estímulo, pretende-se que os dados fornecidos sejam o mais próximo dos dados em tempo real, de forma a garantir que a aposta é feita dentro do tempo de jogo.

10.2.3 Manutenção/Testabilidade

Esta característica identifica a capacidade do produto ser validado. Isto implica que os atributos do código têm uma complexidade relativamente baixa, e isso é calculado principalmente pelo tamanho.

Pode ser medido usando a framework JUnit. Espera-se assim garantir ausência de erros com pelo menos 90% de confiança.

10.2.4 Manutenção/Documentação

Esta característica identifica a capacidade do produto ser modificado. As modificações podem incluir correções, melhorias ou adaptações do *software* para mudanças no ambiente e nos requisitos e especificações funcionais.

Para que tal seja possível, é de enorme importância uma boa documentação do projeto de forma a que todos aqueles que necessitam de modificar o sistema, tanto para correção como melhorias, tenham facilidade em entender o código e as funcionalidades implementadas.

Exemplo: Pretende-se criar uma nova versão melhorada da aplicação onde passará a existir transferências bancárias como possível método de pagamento.

O projeto, estando sustentado por uma boa documentação, permitirá ao *developer* uma rápida e fácil compreensão do código, permitindo orientar-se no mesmo sem grande dificuldade.

10.2.5 Usabilidade/Clareza

Esta característica identifica a capacidade do produto permitir que o utilizador o entenda, se o *software* é adequado, e como é que pode ser usado para tarefas e condições de uso específicas.

Pode ser medido através de inquéritos de satisfação ou até pela adesão dos utilizadores à plataforma.

Exemplo: Um novo utilizador regista-se na aplicação.

Espera-se como resposta a este estímulo que o utilizador consiga em alguns minutos navegar pela aplicação sem dificuldades.

10.2.6 Usabilidade/Operabilidade

Esta característica identifica a capacidade do produto permitir ao usuário operá-lo e controlá-lo.

Pode ser medido através da adesão dos utilizadores, ou até pela visualização do números de apostas feitas ao longo de um ano na plataforma. Se o número de apostas cresce potencialmente com o número de utilizadores novos então é um bom indicador de usabilidade do *software*.

Exemplo: Um novo utilizador pretende fazer uma aposta.

Espera-se como resposta a este estímulo que o utilizador consiga em alguns minutos navegar pela aplicação e realizar uma aposta com sucesso.

10.2.7 Segurança/Proteção de Dados

Esta característica identifica a capacidade do produto impedir o acesso não autorizado a programas ou dados.

Exemplo: Um novo utilizador pretende adicionar a sua informação bancária na aplicação.

Espera-se como resposta a este estímulo que as transações feitas na aplicação sejam 99,99% das vezes seguras, tal como a autorização do banco de dados do cliente funcione 99,999% do tempo. Espera-se ainda que os dados públicos do utilizador sejam transformados em dados privados, como a sua *password*.

11 Riscos e dívida técnica

Na realização do projeto, tomamos consciência de que lidar com transferências de dinheiro é algo que merece um nível elevado de segurança e privacidade do lado do programa. Para tal, é necessário considerar uma lista de riscos técnicos, para os quais precisamos de encontrar solução.

1. O programa não pode guardar as *passwords* dos utilizadores.
2. O programa tem de ter um serviço de transferência de dinheiro seguro e sem falhas.
3. Como iremos utilizar APIs externas, temos de ter em consideração que estas podem apresentar riscos para o nosso programa, por motivos de falhas de conectividade e segurança.

12 Glossário

Vocábulo	**Definição**
Odds	Probabilidades
Username	Nome de utilizador
Bets	Apostas
API	*interface* de programação de aplicação
MVC	*Model-View-Controller* - Arquitetura de software no qual é feita a separação de conceitos em três camadas interconectadas
SQL	*Structured Query Language* - Linguagem de pesquisa para base de dados relacional
JSON	*JavaScript Object Notation* - Formato compacto, de troca de dados simples e rápida entre sistemas
Github	Plataforma de hospedagem de código e arquivos com controle de versão usando o Git
UML	Linguagem padrão para a elaboração da estrutura de projetos de software

Tabela 8: Vocabulário

13 Conclusões

O objetivo desta fase do projeto era aprofundar as decisões de arquitetura de *software*. Para isso, procedemos inicialmente à elaboração dos objetivos e apresentação geral dos requisitos (assunto já abordado na 1ª fase do projeto) e dos objetivos de qualidade. De seguida foram discutidas algumas estratégias, e dividimos o projeto em diferentes componentes de forma a apresentar uma melhor organização.

Para as ações mais importantes na aplicação, como realizar uma aposta ou levantar dinheiro, idealizamos de que forma a aplicação irá correr em tempo real. Para tal, foram realizados diagramas de sequência, que especificam o procedimento de cada caso de uso a nível do código.

De forma a representar melhor o funcionamento do produto, bem como identificar as diferentes entidades fundamentais, os seus atributos e os relacionamentos existentes entre elas, foi desenvolvido um diagrama de classes.

Por fim, achamos pertinente discutir os requisitos de qualidade e os riscos e problemas que podemos encontrar no desenvolvimento deste projeto.

Sentimos que esta parte do projeto foi assim essencial, uma vez que influencia diretamente a estruturação e implementação de toda a aplicação. O trabalho realizado nesta fase do projeto foi de encontro com os objetivos definidos pela equipa docente e desta forma estamos confiantes no trabalho desenvolvido.