UMinho
Mestrado em Engenharia Informática
Requisitos e Arquiteturas de Software

Grupo ?, PL5 / Entrega 2

PG46983	Ana Luísa Lira Tomé Carneiro
PG46988	Ana Rita Abreu Peixoto
PG47194	Francisco José Vilão Matos Queiróz Peixoto
A89582	Henrique Manuel Ferreira da Silva Guimarães Ribeiro
PG47428	Luís Miguel Lopes Pinto





Dezembro de 2021 2021/2022

Conteúdo

Prefácio

No decorrer da realização deste documento foram encontradas algumas dificuldades, tais como a escolha dos *design patterns*. Esta será sempre uma decisão difícil e subjetiva pois não existe uma resposta certa, cada caso é um caso e tiveram de ser medidos os pontos positivos e negativos dos diferentes padrões de *design* (estruturais, comportamentais e de criação).

Por último, consideramos relevante analisar o contributo de cada elemento do grupo para o desenvolvimento do trabalho. Desta forma, cada elemento é classificado com + para alunos que contribuíram significativamente acima da média, 0 para alunos cujo contributo foi próximo da média e - para alunos que contribuíram significativamente abaixo da média:

- Ana Luísa Carneiro **0**
- Ana Rita Peixoto **0**
- Francisco Peixoto **0**
- Henrique Ribeiro **0**
- Luís Miguel Pinto **0**

1. Introdução e Objetivos

Nesta 2ª fase do trabalho foi elaborado um documento de arquitetura para a aplicação RASBet. Este documento aborda diversas questões relevantes para a implementação futura da aplicação, tais como restrições da arquitetura, estratégias da solução, modelação do bloco de construção e da execução, e ainda conceitos transversais, decisões arquiteturais e requisitos de qualidade.

Esta aplicação tem como principal objetivo o registo de apostas desportivas por parte dos utilizadores (público alvo). Desta forma é importante a criação de uma arquitetura divida em componentes de forma a permitir a separação das diferentes funcionalidades do sistema e que consiga manter sempre atualizados todos os eventos, notificar os apostadores e que consiga armazenar informação sobre os apostadores e eventos de forma eficiente. Além disso, é necessário uma arquitetura que permita notificar os apostadores quando um evento é fechado, através do padrão ***observer*** e que permita acrescentar facilmente novos métodos no futuro através do padrão ***facade***. No presente documento é possível analisar com detalhe os tópicos referidos anteriormente e obter uma visão da aplicação que será implementada no futuro.

Visão Geral dos Requisitos

Os requisitos foram identificados e explicitados no relatório da **Fase 1** no capítulo de **Requisitos Funcionais** na secção de **Requisitos funcionais e de dados**.

Metas de qualidade

No contexto deste trabalho, consideramos importante realçar os seguintes objetivos de qualidade, tendo em conta os requisitos não funcionais:

- **Gestor de Eventos**: O sistema deverá garantir que só os gestores de apostas é que gerem as *odds* e os eventos.
- **Atualização rápida de informações**: O sistema deve atualizar rapidamente as *odds* e resultado dos eventos.
- **Formato *WEB***: Disponibilizar o sistema em formato *WEB* e apresentar compatibilidade com os vários *browsers*.
- **Maioridade**: O sistema não deve ser utilizado por indivíduos menores do que 18 anos.

Estes requisitos são fundamentais para a implementação do sistema. É essencial que apenas o gestor seja capaz de alterar *odds* de apostas e gerir os eventos, de modo a garantir o correto funcionamento do programa. No que toca

às informações atualizadas pelo gestor, é de grande importância que estas sejam atualizadas de forma rápida e eficaz, de forma a apresentar informações corretas aos apostadores e em tempo próximo do real. Por conseguinte, o formato *WEB* da aplicação permite que os utilizadores tenham acesso à *app*. Finalmente, a maioridade é importante na medida em que obriga o programa a cumprir uma restrição legal de existirem apenas apostadores maiores de 18 anos.

Stakeholder

Os *stakholders*, clientes e consumidor foram identificados no relatório da **Fase 1** no capítulo de ***Triggers* do projeto** na secção de **clientes, consumidor, *stakeholders***.

2. Restrições de Arquitetura

As restrições de arquitetura, tal como as restrições de limite técnico e limite organizacional foram identificadas e explicitadas no relatório da **Fase 1**, no capítulo de **Restrições do Projeto**, na secção de **Restrições Obrigatórias**.

3. Âmbito e Contexto

Neste capitulo será abordado o âmbito e o contexto do sistema, permitindo assim delimitar o sistema (ou seja, o seu âmbito) de todos os seus parceiros de comunicação (sistemas vizinhos).

Contexto de Negócio

Nesta secção abordamos o contexto do nosso sistema, isto é, delimitar o domínio do nosso sistema, casa de apostas, a um conjunto limitado de funcionalidades e dados, como por exemplo o número de desportos ou o tipo de apostas a implementar. Este secção foi abordada **na fase 1**, mais concretamente no capítulo **Requisitos funcionais**, na secção **Âmbito de Trabalho** no tópico **Modelo de domínio**.

Contexto de Técnico

Nesta secção abordamos o contexto técnico no nosso sistema, isto é, apresentar os limites do sistema evidenciando fontes de informação vizinhas e externas à aplicação. No nosso caso, como não temos nenhuma interação com componentes externas não existe nenhum sistema vizinho que a nossa aplicação interage.

4. Estratégias de Solução

Objetivos de Qualidade

Objetivo de qualidade	Cenário	Abordagens de solução
- Acessibilidade	- Facilidade de analisar e estudar a infraestrutura do sistema.	- *overview* arquitetural estruturada no formato arc42. - modelo de domínio explícito e orientado a objetos.
- Sistema preparado para expansões/alterações	- Implementação de diferentes tipos de apostas, de diferentes desportos ou de diferentes tipos de notificações	- Uso de interfaces na implementação dos componentes mais importantes do sistema, como nas apostas e eventos - Uso de um sistema abstrato e mutável na verificação de apostas. - Uso de um sistema abstrato na definição de eventos desportivos. - Distinção entre histórico e notificações.
- Fácil e intuitiva navegação na aplicação	- Utilizador e gestor percorre uma lista de eventos desportivos.	- Categorização de dados por tipo (tipo de apostas). - Separação de informação de diferentes categorias por diferentes estruturas de dados. - Mecanismos *front-end* que permitam a paginação e/ou navegação de listas expansivas de dados
- Informação consistente e atualizada	- Atualização consistente e rápida das *odds*	- Atualização minuto-a-minuto de toda a informação pertinente sobre eventos desportivos e apostas.
- Respostas rápidas e interativas a inputs do utilizador bem como acontecimentos do sistema	- Utilizador efetua aposta, aposta efetuada é verificada	- Definição de sistema de notificações que causam "alertas- notificações. - Definição de interfaces *front-end* com a capacidade de reagir aos "alertas"definidos. - Sistema que permite enviar mensagens dinâmicas ao utilizador.
- O sistema deverá garantir que só os gestores de apostas é que gerem as *odds* e os eventos	- Gestor altera *odds* de um certo evento.	- Uso de interfaces para distinguir funcionalidades de cada utilizador do sistema. - Uso de sistema de registo e credenciação para garantir integridade dos acessos ao sistema.
- Disponibilizar o sistema em formato *WEB* e apresentar compatibilidade com os vários \textit{browsers}.	- Apostador usa diferentes browsers para aceder à aplicação.	- Verificação do código HTML e CSS. - Testagem da aplicação em diferentes *browsers*.
- O sistema não deve ser utilizado por indivíduos menores do que 18 anos.	- Menor de idade é impossibilitado de criar uma conta no sistema.	- Sistema de registo/credenciação que apenas permite criação de contas a maiores de 18 anos.

Tabela 5.1: Objetivos de Qualidade

Estrutura do Programa RASBet

O sistema RASBet é baseado num diagrama de classes simples e representativo do programa a desenvolver, das quais o apostador, o gestor, o evento e a aposta são as mais importantes.

Estas estruturas, bem como a implementação da aplicação para os gestores e os apostadores estarão abstraídas devido a implementação de interfaces (arquitetura Facade).

Esta aplicação será implementada em **Java**, devido à:

- Fácil tradução de um modelo de domínio orientado a objetos para uma aplicação concreta.
- Simples implementação de interfaces.
- Fácil integração de componentes de interação com o utilizador, bem como interfaces gráficas.
- Familiaridade que os *developers* têm com a linguagem.

Sendo uma aplicação de apostas, a grande parte das funcionalidades do sistema estarão associadas a métodos de visualização e transformação de dados. Logo, a qualidade da implementação das várias estruturas bem como os processos que permitem o seu manuseamento serão os maiores fatores que influenciam as decisões de *design*.

O modelo MVC surge então como uma escolha óbvia para a arquitetura do sistema, dado que oferece uma simples solução para a necessidade de interação entre utilizadores com os dados do sistema bem como à necessidade de proporcionar diferentes funcionalidades a diferentes tipos de utilizador (neste caso, entre os apostadores e os gestores).

Os dados serão persistidos recorrendo a uma base de dados em *mySQL*, por se tratar de uma ferramenta intuitiva e que permite dar suporte às funcionalidades do sistema. Apesar desta base de dados em *mySQL*, a realização de *queries* dependerá principalmente de funcionalidades do Java como *maps*, *reduces* e *streams*.

É de notar que nesta aplicação não foi necessário implementar um componente que fornecesse dados ao sistema, como uma API, o que torna o *design* da arquitetura do sistema mais simplista, visto que, para manter a atualização rápida e consistente da informação disponibilizada aos utilizadores com uma API, seria necessário uma solução de maior complexidade.

5. Modelação do Bloco de Construção

De modo a modelar o sistema e os diferentes blocos de construção que o compõe, foram realizados diferentes diagramas: de *packages*, componentes e classes. Além disso, é importante ter consideração os diferentes níveis de abstração associados a cada diagrama construído. Deste modo, esta secção está organizada por ordem crescente de especificidade, ou seja, os diagramas são apresentados do mais geral para o particular. De seguida está presente uma imagem geral do diagrama de componentes no nível 1 e do diagrama de classes (nível 2) dividido pelos diferentes componentes. Por questões de simplicidade e visualização, o diagrama de *packages* (nível 1) nao foi incluido nesta representação.

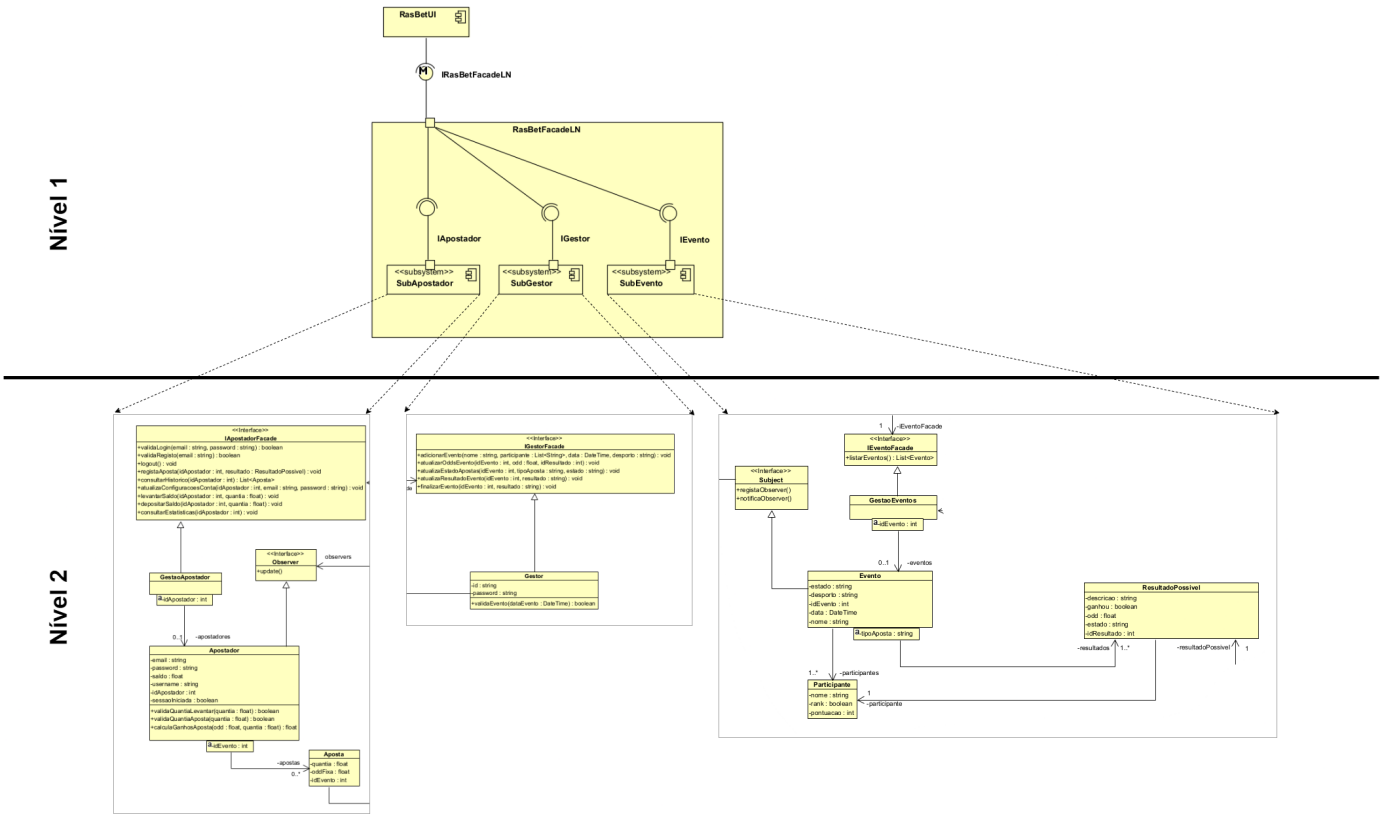

Figura 1: Diagrama de Visão de Bloco

Nível 1 - Diagrama de *Packages*

Num **nível mais abrangente**, podemos considerar o diagrama de *packages*, onde conseguimos observar as dependências entre as diferentes classes do sistema.

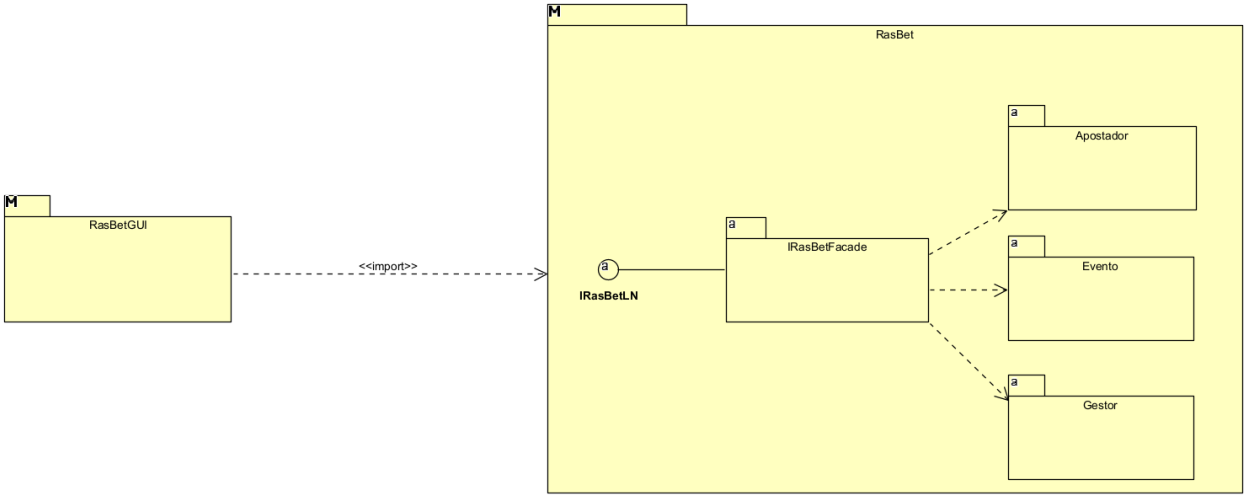

Figura 2: Diagrama de *Packages*

Nível 1 - Diagrama de Componentes

O diagrama de componentes oferece uma visão geral e de **alto nível** dos diferentes componentes do sistema. Este modelo permite observar os diferentes subsistemas do programa e as dependências existentes entre si. Consideramos relevante dividir o programa em 3 subsistemas: apostador, gestor e evento, de forma a suportar as diversas funcionalidades do sistema.

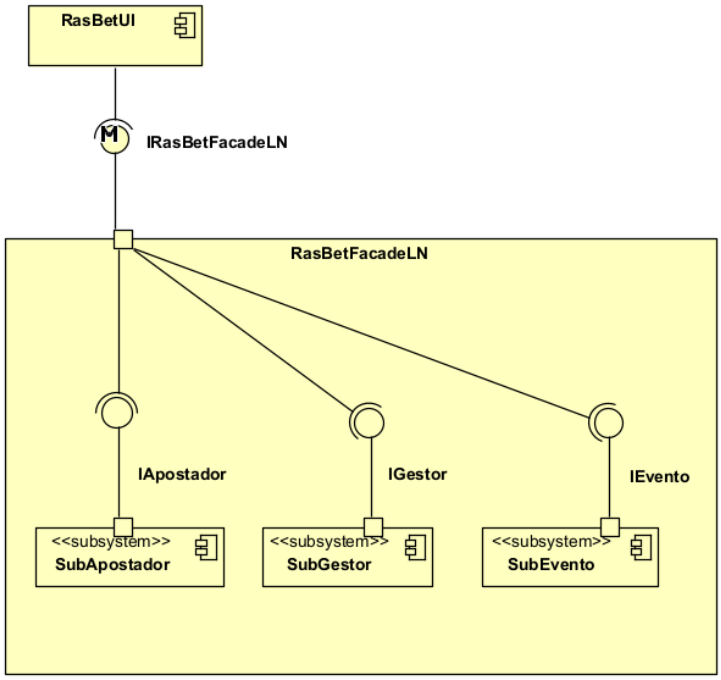

Figura 3: Diagrama de Componentes

Divisão Das Responsabilidades

Tendo agora por base os componentes adotados para o nosso sistema, bem como os use cases definidos na fase 1, fará sentido apresentar as responsabilidades atribuídas a cada componente, ou seja, os métodos associados a cada uma das componentes do sistema.

Neste sentido, surge na tabela abaixo a ligação entre cada use case e a componente onde ocorre a sua implementação.

Use Case	**Componente Responsável**
Login	Apostador
Registo	Apostador
Logout	Apostador
Consultar Eventos	Evento
Consultar Apostas Realizadas	Apostador
Consultar Notificações	Apostador
Alterar Configuracoes da Conta	Apostador
Levantar Saldo	Apostador
Depositar Saldo	Apostador
Consultar Estatisticas da conta	Apostador
Fazer Aposta	Apostador
Atualizar Odds	Gestor
Adicionar Evento	Gestor
Atualizar Aposta	Gestor
Finalizar Evento	Gestor

Tabela 6.1: Tabela de divisão das responsabilidades

Nível 2 - Diagrama de Classes

O diagrama de classes fornece uma visão das classes do sistema com um um **nível de maior de especificidade**. É possível observar as diferentes classes que vão constituir o programa e as suas relações, assim como as suas variáveis de instância e métodos.

O ponto de partida para a construção das classes foram os requisitos abordados previamente e o modelo de domínio do problema. Deste modo, as principais classes do sistema são:

- **Apostador:** É a entidade que irá utilizar o sistema e realizar apostas.
- **Gestor:** Classe que irá suportar o gestor do sistema e as suas funcionalidades.
- **Evento:** Contém informações relevantes sobre o programa e os eventos existentes.
- **RasBetModelFacade:** Classe principal do programa que dá suporte a todas as funcionalidades do sistema.

O **Apostador** terá uma lista de apostas, onde estarão todas as apostas efetuadas independentemente do seu estado. Esta representação permite ao utilizador consultar o histórico de apostas e as notificações associadas a cada aposta (acionadas pela mudança de estado de uma aposta). Além disso, o apostador também possui diversos atributos que o caracterizam e permitem interação com o sistema.

A classe **Evento** possui uma estrutura que vai associar um tipo de aposta a todos os resultados possíveis desse tipo. Tal como estava presente no modelo de domínio, o tipo de aposta pode ser: 1X2, por pontos ou apostar num vencedor (no caso de jogos sem empate). O resultado possível representa cada aposta passível de ser registada por um apostador, e está associada a um participante do evento.

Relativamente a classe **Gestor**, através do atributo *gestaoEventos* que representa a classe *GestaoEventos* conseguimos permitir o controlo propriamente dito dos eventos, quer seja o adicionar, remover ou a atualização de um evento. Esta classe possui ainda atributos ligados a conta de um gestor e o um método para validar os eventos criados pelo gestor (*validaEvento*).

Por fim, é de notar que para a realização deste diagrama consideraram-se dois padrões de *design* distintos: ***Facade*** e ***Observer***, de forma a modelar o sistema consoante diferentes interfaces e para implementar a noção de notificação, respetivamente.

O padrão comportamental ***Observer*** foi usado de forma a implementar as notificações ligadas aos apostadores sobre o estado das suas apostas. Deste modo, quando um evento é finalizado, é invocado o método *notificaObserver()* que irá notificar todos os apostadores que realizaram apostas nesse evento, e através do método *update()* estes conseguem obter a informação atualizada sobre o evento que foi finalizado e consequentemente sobre o estado das suas apostas.

O padrão estrutural ***Facade*** foi usado de forma a que a classe que irá implementar todas as funcionalidades do sistema, *RasBetModelFacade*, tivesse acesso a *interfaces* que disponibilizam um conjunto de métodos que estão a ser implementados nas restantes classes. Desta forma, promovemos a abstração dos dados, pois deixamos ao encargo de cada classe a implementação, não só dos métodos a serem utilizados pelo *RasBetModelFacade* mas também a implementação de outros métodos utilizados internamente em cada classe e que não se encontram nas *interfaces*. A implementação deste padrão também teve como objetivo permitir que o nosso sistema consiga facilmente ser integrado noutros sistemas ou até integrar novas componentes, como API's.

Tal como podemos ver no diagrama abaixo, existem um conjunto de classes que irão representar os diversos **DAO's do sistema** (classes com cor verde), isto é, as classes cujos dados vamos persistir numa base de dados. As informações

sobre os apostadores, apostas, eventos, resultado possíveis e participantes vão ser armazenados num sistema de base de dados pois são dados principais no nosso sistema e há uma necessidade de ter acesso historial tanto do apostador como os eventos após um apostador finalizar a sessão no sistema.

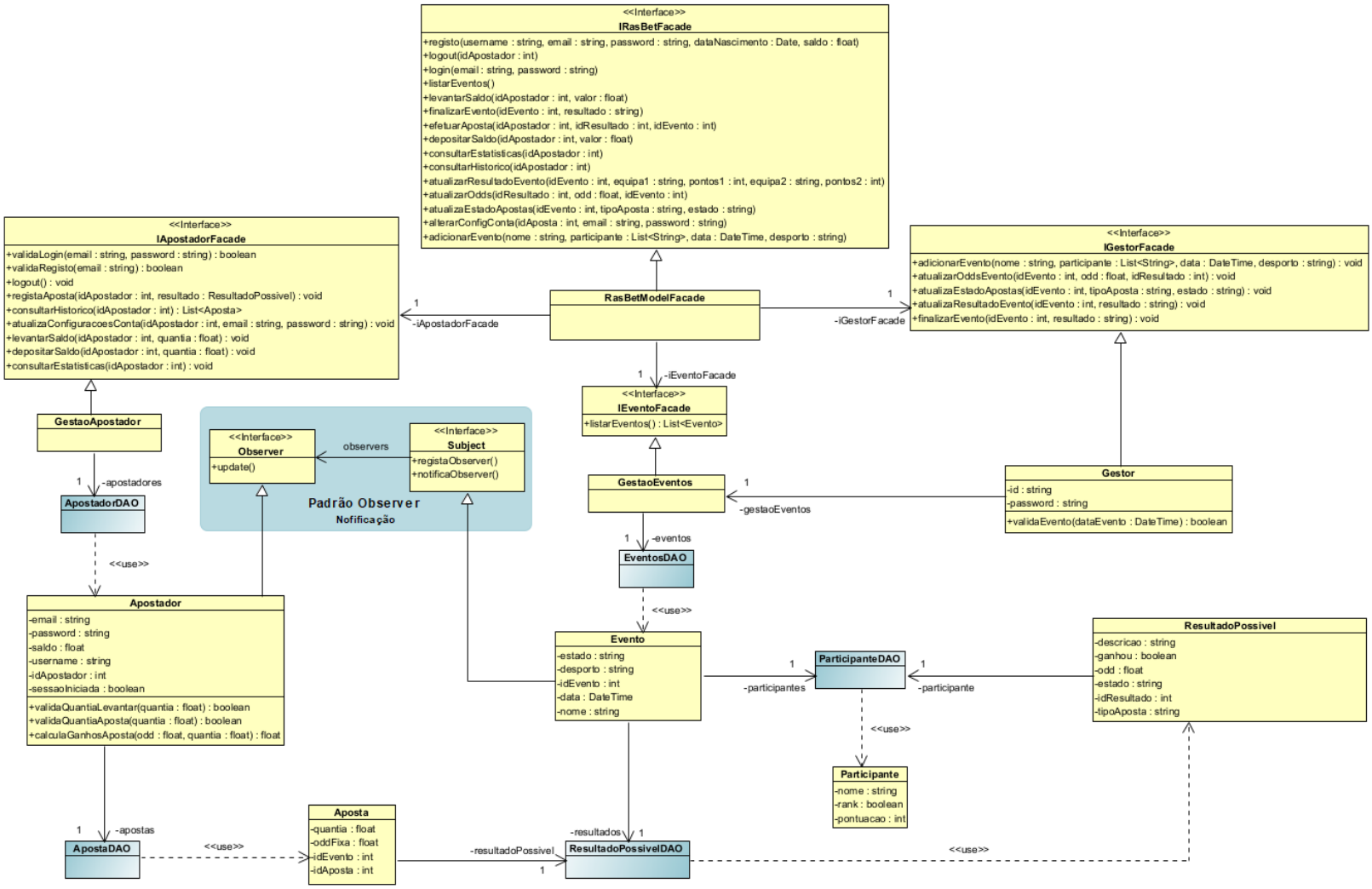

Figura 4: Diagrama de classes

Modelo Conceptual

Na figura abaixo apresentamos o modelo conceptual da base de dados desenvolvido de forma a persistir toda a informação acerca de apostadores, apostas, resultados possíveis, eventos e participantes.

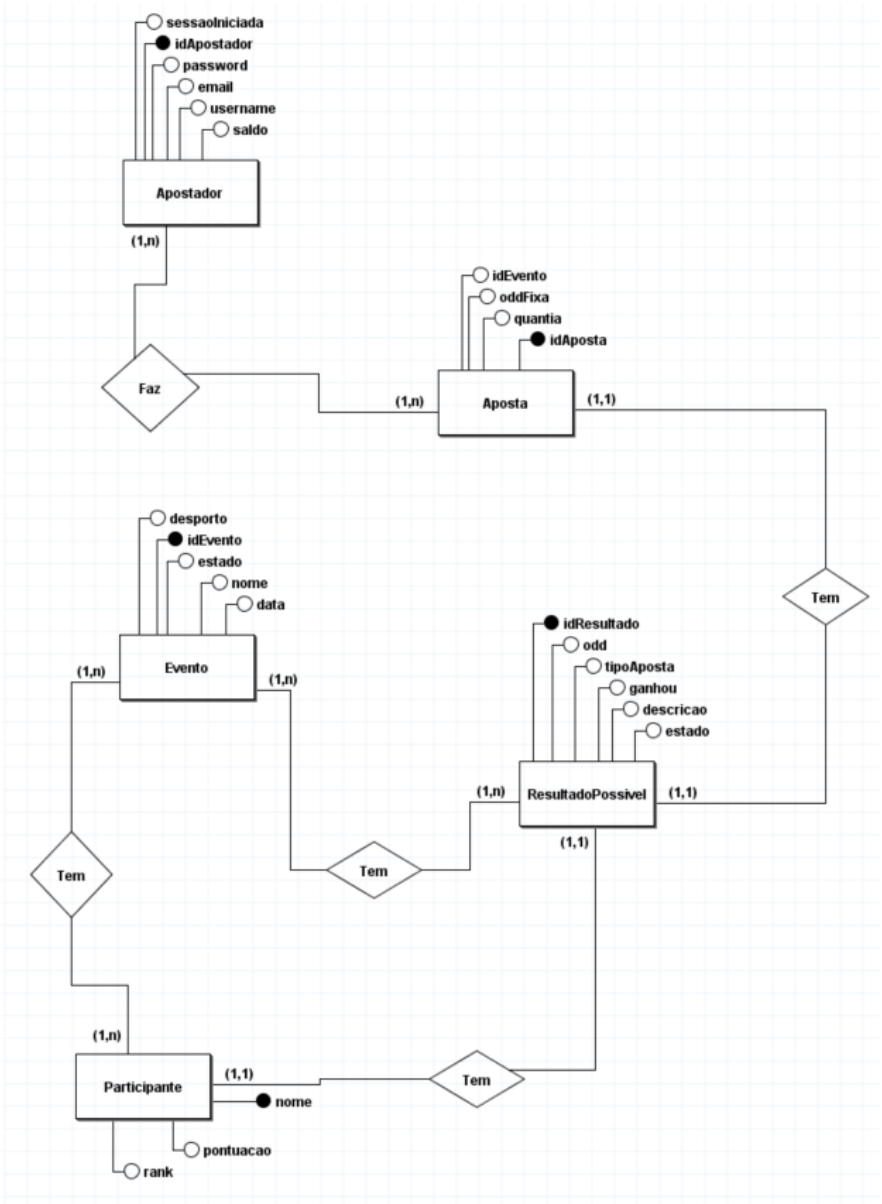

Figura 5: Modelo Conceptual

Tal como podemos ver os atributos de cada entidade neste modelo conceptual correspondem às variáveis de instância de cada classe que vamos persistir. As relações presentes também evidenciam a interação entre cada classe no diagrama de classes.

Um apostador terá um conjunto de apostas em que apostou, onde cada aposta será associada a um evento através do *idEvento* e a um resultado possível. Esta relação vai permitir a criação de um *map* que vai associar a cada *idEvento* uma lista de apostas na classe apostador.

O evento terá um conjunto de participantes, identificados pelo o nome, e um conjunto de resultados possíveis que serão identificados pelo tipo de aposta a que pertencem e pelo *idResultadoPossivel*.

Finalmente, é possível ver que cada resultado possível está associado a um participante, fazendo com que um dos atributos do resultado possível seja o participante.

Modelo Lógico

Na figura abaixo apresentamos o modelo lógico da base de dados desenvolvido de forma a obter todas as tabelas de base de dados, quer sejam entidades como relações, necessários para a implementação do sistema de base de dados.

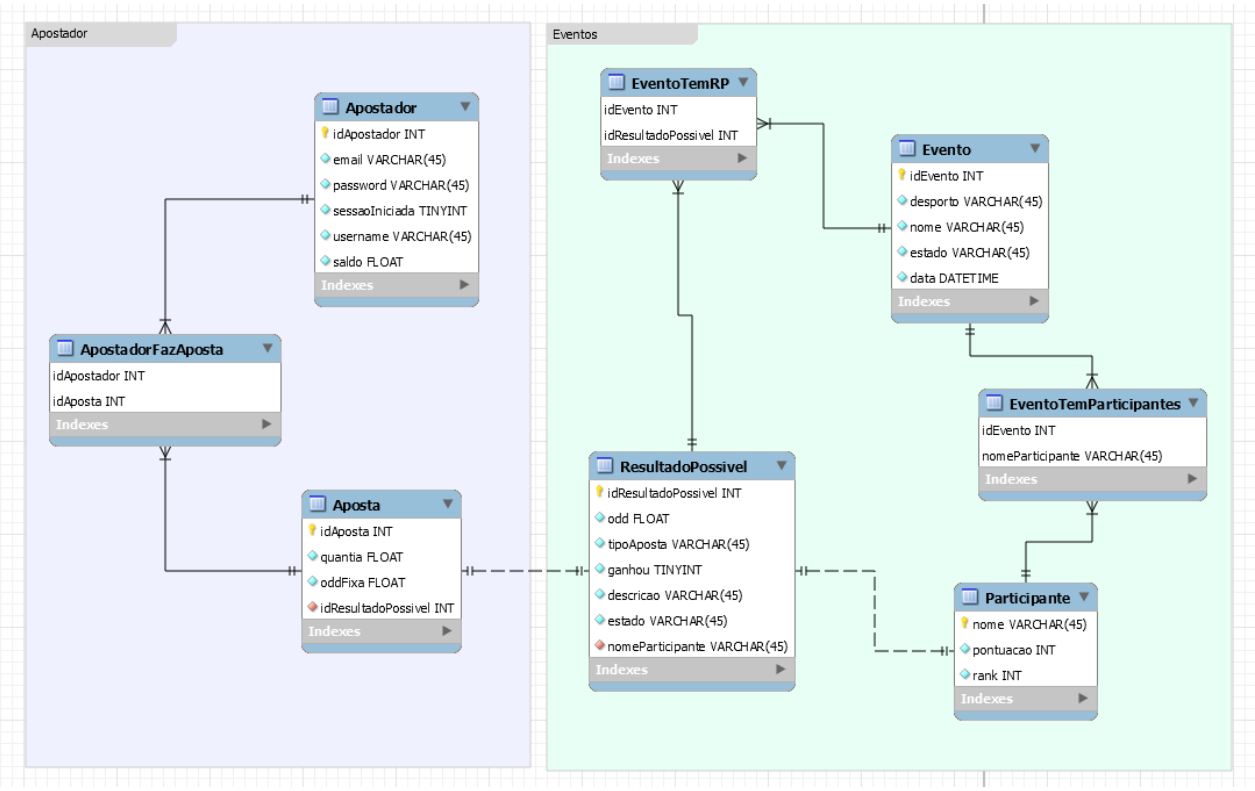

Figura 6: Modelo Lógico

6. Modelação da execução

Diagrama de Sequências

Nesta secção apresentamos os diagramas de sequência dos métodos mais importantes e representativos do sistema. Nestes diagramas podemos ver a modelação comportamental ao longo do tempo de execução para os métodos de : fazer aposta, adicionar eventos, atualizar estado das apostas, atualizar *odds* e finalizar evento.

Fazer Apostas

Neste diagrama apresentamos o fluxo de interação entre cada classe de forma a conseguir adicionar uma nova aposta na lista de aposta de um determinado apostador. Para isso, iniciamos o fluxo na classe *RasBetModelFacade* que vai aceder à *GestaoEventos* e obter o resultado possível que o apostador apostou. De seguida vai invocar outro método para adicionar esse resultado possível na aposta realizada pelo utilizador. Na classe *GestaoAposta* criamos uma nova aposta com a quantia indicada pelo utilizador e com o resultado possível apostado. Internamente esta função vai armazenar a *odd* do resultado possível para que posteriormente seja utilizada para a obtenção do ganhos caso a aposta seja a vencedora. Finalmente, essa aposta é armazenada na base de dados do apostador.

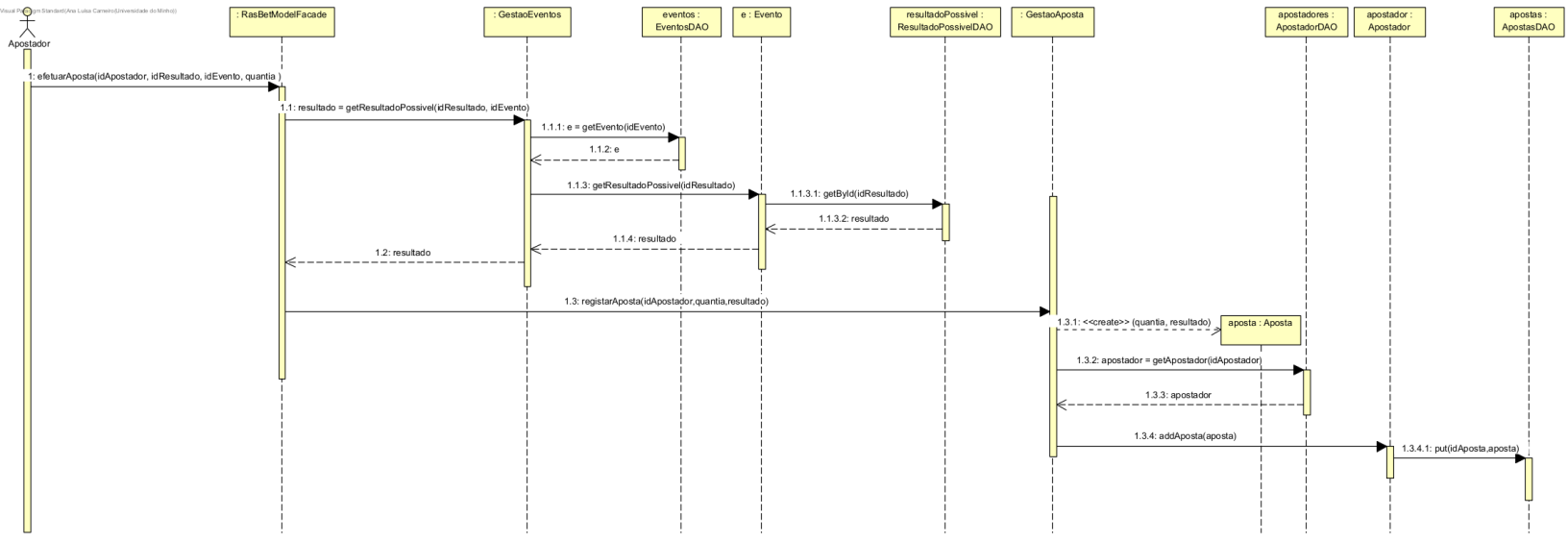

Figura 1: Diagrama Sequência - Fazer Apostas

Adicionar Evento

Neste diagrama apresentamos o fluxo de interação entre cada classe de forma a conseguir adicionar um novo evento ao sistema. Para isso, iniciamos o fluxo na classe *RasBetModelFacade* que vai aceder ao *Gestor* e criar um evento com a informação recebida como parâmetro. De seguida invoca-se o método para adicionar o evento criado na base de dados.

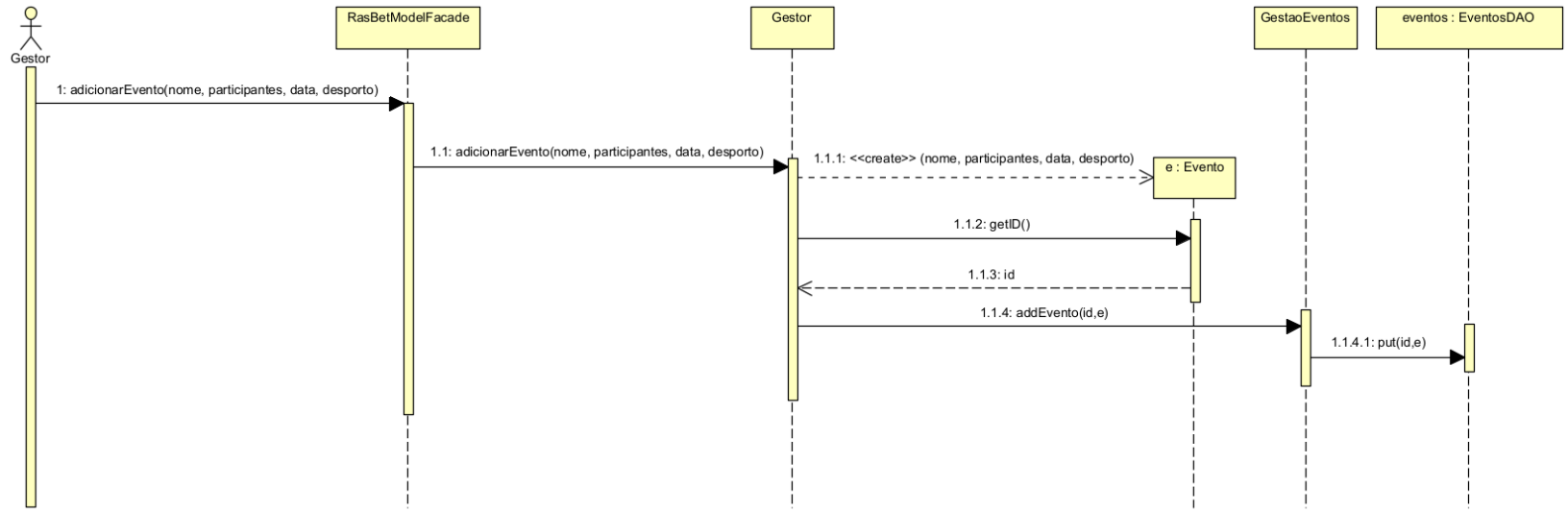

Figura 2: Diagrama Sequência - Adicionar Evento

Atualizar Estado das Apostas

Neste diagrama apresentamos o fluxo de interação entre cada classe de forma a conseguir atualizar o estado das apostas de um dado tipo num dado evento. Para isso, iniciamos o fluxo na classe *RasBetModelFacade* que vai aceder ao *Gestor* e vai, a partir do identificador de evento e do tipo de apostas, aceder aos resultados possíveis. De seguida, vai percorrer cada resultado possível alterando a cada iteração o seu estado para o que foi fornecido em parâmetro, isto é, "Suspenso", "Fechado"ou "Aberto".

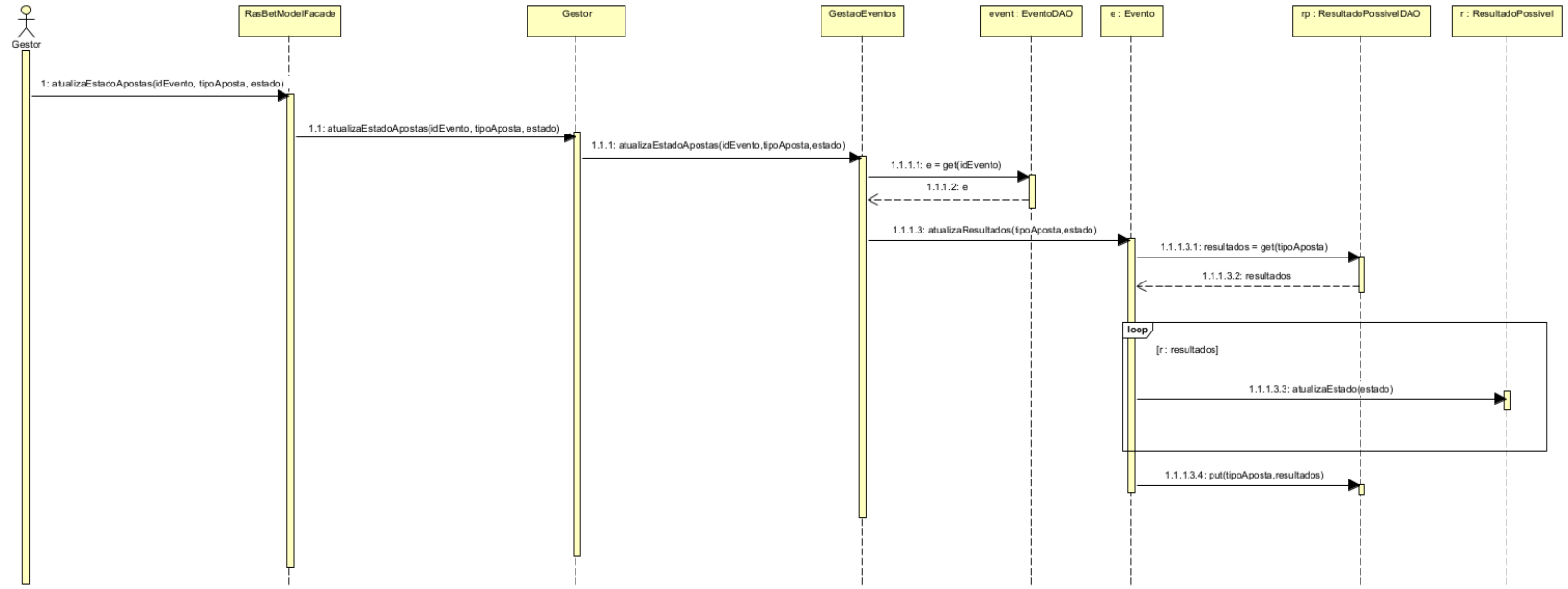

Figura 3: Diagrama Sequência - Atualizar Estado das Apostas

Atualizar *Odds*

Neste diagrama apresentamos o fluxo de interação entre cada classe de forma a conseguir atualizar a *odd* de um dado resultado possível. Para isso, iniciamos o fluxo na classe *RasBetModelFacade* que vai aceder ao *Gestor* e vai, a partir do identificador de evento e do identificador do resultado possível, aceder ao resultado possível que pretendemos alterar. De seguida, vai atualizar a *odd* do resultado para a fornecida como parâmetro.

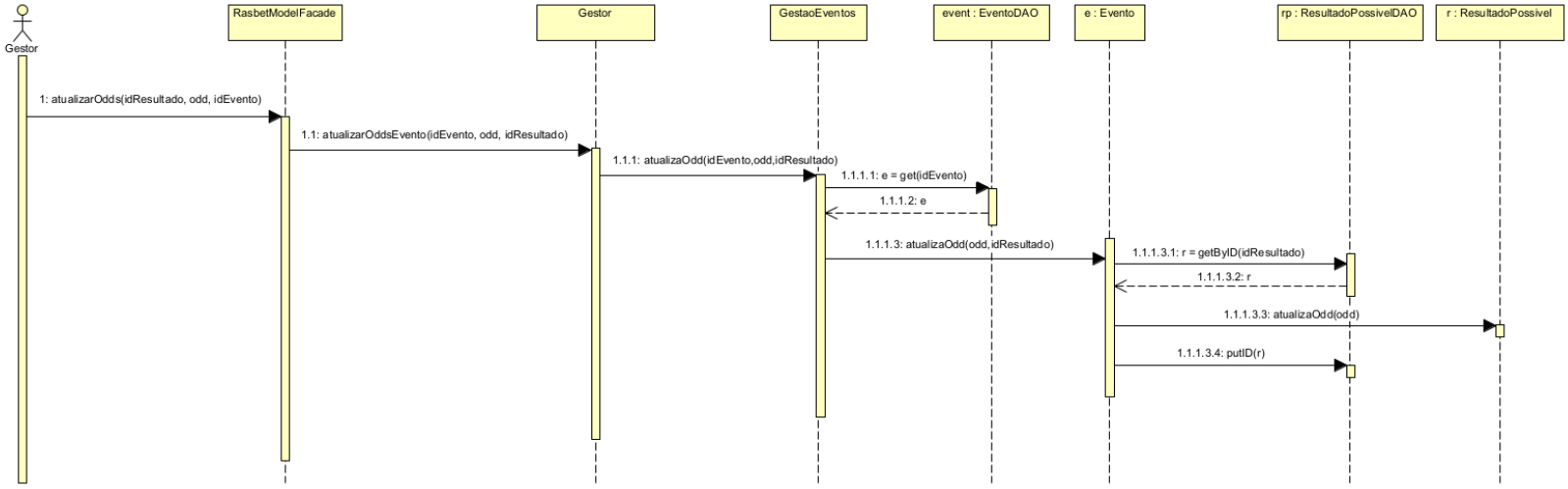

Figura 4: Diagrama Sequência - Atualizar *Odds*

Finalizar Evento

Neste diagrama apresentamos o fluxo de interação entre cada classe de forma a conseguir finalizar um evento. Para isso, iniciamos o fluxo na classe *RasBetModelFacade* que vai aceder ao *Gestor* e chamar dois métodos distintos, um que atualiza o estado de todos os tipos de apostas para o estado de "Fechado", tal como foi mencionado no diagrama de sequência da figura 3 e outro que vai atualizar o resultado final do evento. Neste segundo método vamos atualizar o número de pontos finais ou o *ranking* de cada participante no evento e vamos indicar qual foi o resultado possível que realmente aconteceu. De seguida, vamos notificar todos os apostadores (observers) que apostaram nalgum resultado possível do evento, através do método *notifyObserver()*

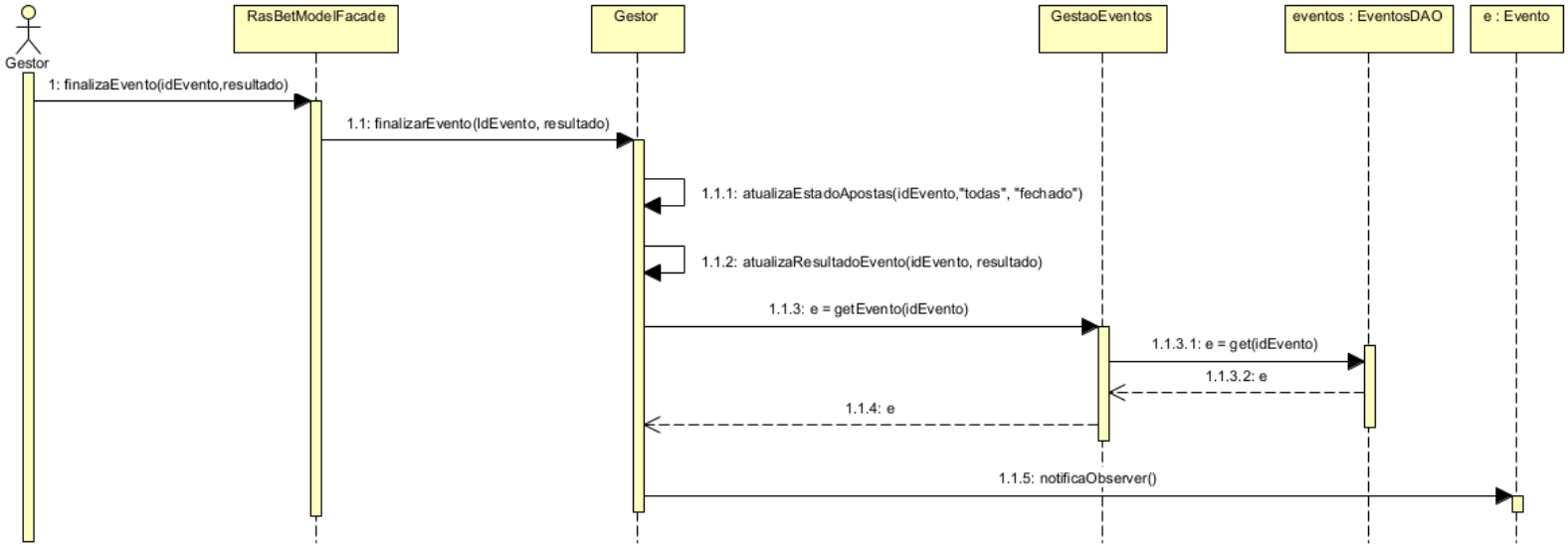

Figura 5: Diagrama Sequência - Finalizar Evento

Diagrama de Atividades

Os diagramas de atividades foram desenvolvidos anteriormente na **fase 1** deste trabalho prático, para as funcionalidades mais importantes e representativas do sistema. (consultar: relatório - **Fase 1**, tópico 8 - **Âmbito do Sistema** , subtópico - **Diagramas de Atividades**).

Máquina de Estados

A máquina de estado foi desenvolvido anteriormente na **fase 1** deste trabalho prático, para a mudança de estados do evento. (consultar: relatório - **Fase 1**, capitulo - **requisitos Funcionais**, tópico 7 - **Âmbito do trabalho** , subtópico - **Diagrama de Estados do evento**).

7. Modelação de desenvolvimento

O **diagrama de *deployment*** permite visualizar qual o *hardware* onde o programa implementado será executado. Numa primeira versão do sistema, todos os componentes do programa (sistema e base de dados) serão executados na mesma máquina, tal como se reflete na imagem abaixo.

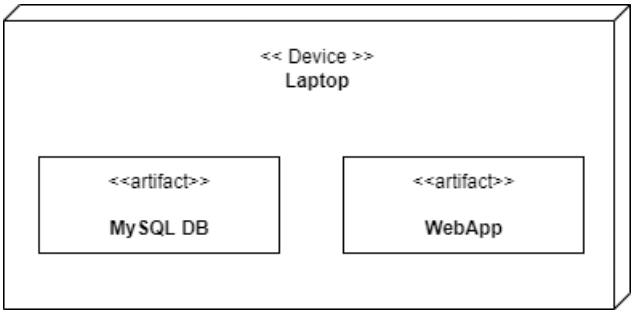

Figura 1: Diagrama de *Deployment*

8. Conceitos Transversais

Neste capitulo serão abordados vários tópicos relevantes relativos às diferentes componentes do sistema, de modo a demonstrar um corte transversal da aplicação.

Conceitos de Domínio

Os Conceitos de Domínio foram identificados e explicitados no relatório da **Fase 1**, no capítulo de **Requisitos Funcionais**, na secção **Modelo de Domínio do Sistema**.

Experiência do Utilizador

A interface com o utilizador tem por base o GUI apresentado anteriormente, nos *Mockups* presentes na **Fase 1**, no capítulo de **Requisitos Funcionais**, na secção **Âmbito do sistema**, no tópico ***Mockups* do sistema**. O utilizador interage com a plataforma via aplicação web ou aplicação *desktop* e tem a possibilidade de realizar apostas desportivas com base num valor escolhido, contudo limitado pelo valor do saldo na carteira.

As ações do utilizador são registadas pelo sistema, o que permite uma gestão e armazenamos dos dados para serem usados na implementação do histórico de apostas do utilizador.

Proteção e Segurança

De forma a garantir uma boa politica de proteção de dados relativa aos utilizadores, todos os dados são guardados segundo um protocolo de segurança, que aplica encriptação ao nível de dados pessoais, como por exemplo a palavra-passe.

Os utilizadores só conseguem aceder à plataforma via *login*, sendo por isso necessário uma rotina de autenticação sempre que o utilizador pretenda entrar na aplicação. Desta forma o serviço garante a proteção dos dados do utilizador.

Componentes de *Back-end* / *Under-the-Hood*

Nesta secção pretende-se destacar implementações que se virão a realizar de forma a garantir certos requisitos importantes do sistema, como por exemplo a base de dados e mensagens de erro.

Começando pela persistência de dados, assegurada através de uma base de dados *mySQL*, onde se pretende guardar dados relativos aos utilizadores e aos eventos desportivos. A conexão da base de dados ao serviço é feita através de *Data Access Objects*.

A aplicação prevê ainda executar diversas mensagens e alertas para controlo de erros, que garantem uma boa experiência de uso por parte do utilizador e impedem certas ações indesejadas de serem realizadas. As mensagens de erro são comunicadas ao utilizador sob a forma de *pop ups*.

9. Decisões de Arquitetura

Notificações e Histórico

Problema:

Será necessário implementar notificações e histórico como elementos separados do sistema?

Alternativas consideradas:

- Implementar notificações como um sistema dentro do histórico.
- Implementar notificações e histórico como componentes separados.

Decisão:

As notificações do sistema servirão para alertar o utilizador de certos acontecimentos, sendo que, neste sistema, o único uso implementado de momento será para dar informação sobre apostas realizadas. O problema, de ponto de vista de *design*, era que esta componente e o histórico eram demasiado semelhantes, porque ambas possuíam informações pertinentes a apostas. Decidimos, então integrar as notificações dentro histórico, não havendo no modelo uma componente "notificação"mas sim um "histórico"e um "observador".

Padrão de design das apostas

Problema:

Tendo a aplicação vários tipos de apostas, será necessário implementar métodos que verifiquem se estas foram vencedoras ou não.

Alternativas consideradas:

- Definir tipos de apostas e métodos de verificação individuais.
- Definir tipos e verificações de apostas como um sistema abstrato e com base em instanciação.

Decisão:

Tendo em conta, mais uma vez, a expansão do sistema, decidimos implementar um sistema de apostas abstrato e instanciado, visto que perante possível necessidade de implementar diferentes tipos de apostas ou eventos desportivos, basta definir os novos parâmetros de sucesso tendo em conta o "resultado possível"e não sendo necessário fazer alterações à arquitetura do sistema.

Base de dados

Problema:

Será necessário implementar uma base de dados?

Alternativas consideradas:

- Não implementar base de dados, sendo os dados guardados em formato JSON.
- Implementar base de dados e integrar DAOs no sistema.

Decisão:

Decidimos que o sistema necessita de uma base de dados devido às grandes quantidades de dados a serem armazenados, bem como a simplicidade de editar e encontrar informação.

Arquitetura do sistema de notificações

Problema:

Arquitetura do sistema de notificações.

Alternativas consideradas:

- Implementar o módulo das notificações com o padrão de *design*, *observer*.
- Implementar o módulo das notificações sem *observer*, integrando o envio de notificações nas funcionalidades já existentes do sistema.

Decisão:

Sem a componente *observer* na arquitetura das notificações, seria necessário incorporar este mecanismo em certos métodos relativos a funcionalidades críticas do sistema, como por exemplo, a verificação de apostas ou a introdução de novos eventos, pelo que seria complicado controlar quais destas ações deveriam gerar uma notificação ou não. A componente *observer*, por outro lado, garante maior controlo e interatividade aos utilizadores, permitindo "escolher"que tipo de notificações deverão ser enviadas e consequentemente, evitar frustrações sobre avisos indesejados. Assim, decidimos incluir o "observer"na arquitetura das notificações.

10. Requisitos de Qualidade

Os requisitos de qualidade foram identificados e explicitados no relatório da **Fase 1**, no capítulo de **Requisitos não Funcionais**, nas diferentes secções.

Árvore de qualidade

A árvore de qualidade seguinte pretende demonstrar como os diferentes requisitos de qualidade são abordados, com recurso a exemplos práticos onde se garante a aplicabilidade dos diferentes parâmetros de qualidade. De forma a identificar onde os parâmetros foram mencionados na **Fase 1**, identificou-se cada um com o número do requisito não-funcional equivalente.

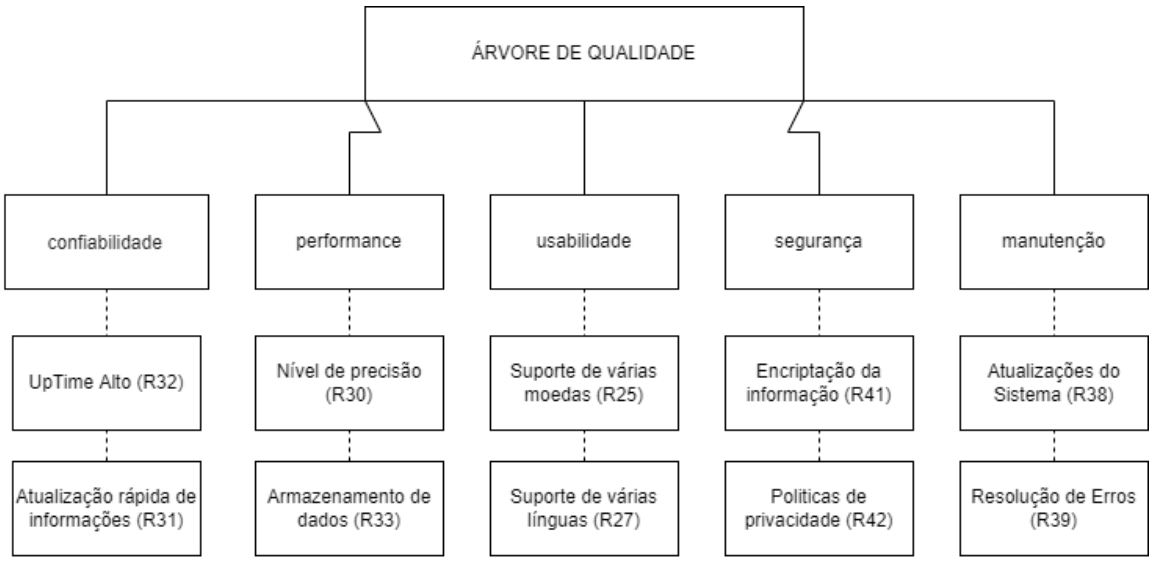

Figura 1: Árvore de Qualidade

Cenários de qualidade

Para cada requisito de qualidade referido anteriormente, criou-se um cenário de qualidade que descreve o comportamento do serviço.

- **UpTime Alto:** O serviço RASBet espera pelo menos um UpTime de 98% ao ano, sendo que atualizações do sistema e possíveis resoluções de erros que impeçam a continuação do serviço, deverão ser devidamente informadas aos utilizadores.
- **Atualização rápida de informações:** As atualizações de dados relativos ao serviço por parte do gestor estão disponíveis ao utilizador em tempo-real, considerando um intervalo de atualização de dados de 1 segundo.
- **Nível de precisão:** O sistema tem um alto nível de precisão no cálculo dos ganhos possíveis, nas transações bancárias e nas *odds*. O sistema apresenta todos os cálculos e quantias na ordem dos cêntimos e as *odds* na ordem das centésimas.
- **Armazenamento de dados:** O sistema guarda e processa os dados de grandes quantidades de utilizadores sem comprometer o funcionamento normal do sistema. O sistema não espera ter um aumento superior a 30% nos tempos de resposta quando existe 0 a 300 utilizadores ativos no sistema.
- **Suporte de várias moedas:** O sistema suporta várias moedas de modo a conseguir suportar utilizadores de diferentes regiões.
- **Suporte de várias línguas:** O sistema suporta várias línguas de modo a conseguir suportar utilizadores de diferentes regiões.
- **Encriptação de informação:** Os dados dos utilizadores do sistema estão encriptados na base de dados.
- **Politicas de Privacidade:** O sistema informa os utilizadores da mudança nas políticas de privacidade para que estes tomem conhecimento da maneira como os dados inseridos no site estão a ser usados e para que motivos.
- **Atualizações do Sistema:** O sistema permite atualizações mensais do sistema em alturas com pouco tráfego no sistema. Deve-se avaliar ao longo de 1 ano quais as 8 horas mensais onde o sistema tem um tráfego inferior a 30% de utilização e numa altura em que isso se verifica, o sistema atualiza-se.
- **Resolução de Erros:** O sistema é capaz de resolver os erros encontrados num intervalo inferior a 48h.

11. Riscos e Dívida Técnica

Neste capítulo apresentamos diversos sistemas e serviços que poderíamos no futuro incorporar na nossa aplicação, tornando-a mais robusta para ser utilizada em contexto real.

No futuro poderíamos integrar uma API de desportos, de forma a que esta seja a responsável por nos fornecer toda a informação acerca dos eventos sem que haja necessidade do gestor controlar toda a gestão de eventos.

Podia-se também expandir o sistema para apresentar informações mais detalhadas sobre os eventos desportivos, havendo a possibilidade de pagar por um novo serviço *watchers* que fornecia à nossa aplicação os dados dos eventos atualizados ao minuto. Esta expansão permitiria também a inclusão de novos de tipos de apostas que costumam estar presentes em *apps* já existentes no mercado, que não incidem sobre o resultado final de um evento desportivo mas sim sobre diferentes métricas relativas aos seus participantes, como golos marcados, cartões, etc.

Caso a aplicação fosse feita num contexto real, seria necessário implementar um sistema de pagamento e de troca de dinheiro, que integrasse diversos métodos de pagamento como *paypal*, multibanco, etc.

Na imagem abaixo encontra-se uma representação das diversas interações a sistemas e serviços externos que a aplicação poderia ter no futuro.

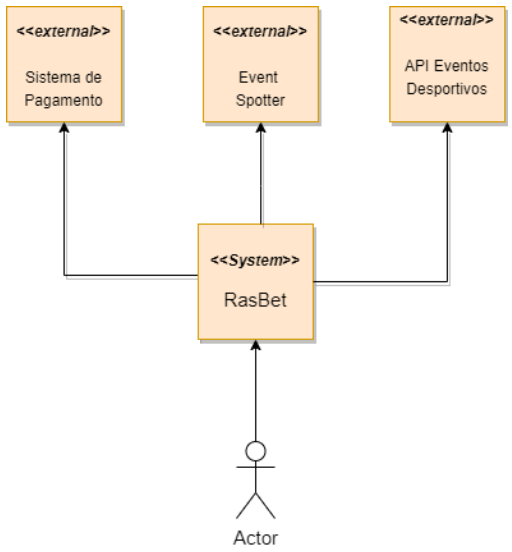

Figura 1: Diagrama de *Contexto*

12. Glossário

RASBet	Nome do sistema de apostas a implementar
RAS	Requisitos e Arquiteturas de Software
UML	*Unified Modeling Language*
IT	*Information Technology*

Conclusão

Dada por concluída a segunda etapa do projeto, faz sentido apresentar uma visão crítica, refletida e ponderada do trabalho realizado.

No espetro positivo, consideramos relevante destacar o facto de que o presente documento de arquitetura vai de acordo com o *template* fornecido e que, através desde documento, é possível ter uma visão completa e representativa dos aspetos arquiteturais do sistema, através dos diagramas mais adequados e seguindo um nível de abstração do maior para o menor.

Por outro lado, em termos de melhorias ou *upgrades* no nosso projeto, consideramos que seria benéfico implementar outras funcionalidades do programa, caso o projeto fosse desenvolvido no mundo real invés do contexto académico, tal como foi mencionado no capítulo 11.

Em suma, consideramos que o balanço do trabalho é positivo, as dificuldades sentidas foram superadas e os requisitos propostos foram cumpridos. Desta forma, com base neste documento de arquitetura, o projeto tem uma base sólida para prosseguir para a parte do desenvolvimento e implementação.