



UMinho

**Mestrado Engenharia Informática
Requisitos e Arquiteturas de Software
(2022/23)**

**RASBET
Solução Arquitetural**

Cristiano Pereira (PG50304)

João Martins (PG50463)

Jorge Lima (PG50506)

Rúben Santos (PG50733)



Braga, 4 de dezembro de 2022

Conteúdo

1 O propósito do Projeto	3
1.1 Introdução	3
1.2 Objetivos	3
2 Restrições do Projeto	4
2.1 RASBet API	4
2.2 The Odds API	4
2.3 Spring Security	4
3 Contexto e Âmbito do Projeto	5
3.1 Cliente	5
3.2 Apostador	5
3.3 Especialista	6
3.4 Administrador	6
3.5 RASBet API e The odds-api	6
4 Estratégia da Solução do Projeto	7
4.1 Client-server pattern	7
4.2 Model-view-controller pattern	8
4.3 Master-slave pattern	8
5 Vista em Bloco	9
5.1 Diagrama de Componentes	9
5.2 Diagrama de Classes	10
5.2.1 Diagrama de classes : Controller	10
5.2.2 Diagrama de classes : Database	11
5.2.3 Diagrama de classes : Entities	11
5.2.4 Diagrama de classes : GamesAPI	12
5.2.5 Diagrama de classes : Security	12
6 Vista em Tempo de Execução	13
6.1 Diagrama de sequência: getEvents	13
6.2 Diagrama de sequência: makeBet	14
6.3 Diagrama de sequência: login	14
7 Vista de Deployment	15
7.1 Diagrama de Instalação	15

Lista de Figuras

3.1	Diagrama de <i>Contexto</i>	5
4.1	Diagrama C4 Model Container	7
5.1	Diagrama de <i>Componentes</i>	9
5.2	Diagrama de <i>Packages</i>	10
5.3	Diagrama de Classes (Controller).	10
5.4	Diagrama de Classes (Database).	11
5.5	Diagrama de Classes (Entities).	11
5.6	Diagrama de Classes (GamesAPI).	12
5.7	Diagrama de Classes (Security).	12
6.1	Diagrama de <i>Sequência</i>	13
6.2	Diagrama de <i>Sequência</i>	14
6.3	Diagrama de <i>Sequência</i>	14
7.1	Diagrama de <i>Instalação</i>	15

1. O propósito do Projeto

Short description of the requirements, driving forces, extract (or abstract) of requirements (present a summary of the requirements document). Top three (max five) quality goals (non functional requirements) for the architecture which have highest priority.

[requisitos com prioridade] [joão]

1.1 Introdução

Neste documento apresentamos uma descrição geral da solução arquitetural do projeto **Rasbet**. Começamos por apresentar os nossos principais objetivos com o desenvolvimento do projeto, depois aprofundamos os requisitos não funcionais implementados, posteriormente expomos o contexto e o âmbito do Projeto, desenvolvendo o papel de cada ator do sistema e também das duas *api's* implementadas, após exibimos a nossa estratégia da solução do projeto em conjunto com alguns diagramas mas técnicos. Por fim, demonstramos a nossa vista de como executávamos o **deployment** do projeto.

1.2 Objetivos

Devido a natureza a qual o **Rasbet** se caracteriza, o mais importante para o sucesso deste projeto é a quantidade de apostadores, para isto tivemos que moldar o projeto sempre a pensar de como otimizar a experiência do apostador. Para isto vimos como um dos principais objetivos tornar o **website** o mais apelativo e fácil de utilizar para o utilizador final.

Como qualquer outra casa de apostas o **Rasbet** tem que ter suporte tanto a levantar como a depositar dinheiro e também vai ter acesso a vários dados privados do utilizador sabendo isto vimos como extrema importância a segurança do website, protegemos os nossos utilizadores através do uso de **JWT (Json Web Tokens)** que são tokens únicos por utilizador com informação do mesmo assim conseguimos identificar e ao mesmo tempo manter a segurança de dados do utilizador. Também tivemos como objetivo na segurança de manter a comunicação entre o cliente e o servidor de dados do sistema encriptado, para obter isto aplicamos *HTTPS* em todas as nossas comunicações externas.

Com a grande influencia que a internet e em particular as redes sócias têm sobre o que nos gostamos e consumimos consideramos que o **website** deve facilmente suportar novos desportos e tipos de apostas com o mínimo de mudança de código, com isto em mente tentamos minimizar as restrições que colocamos tanto na estrutura que guarda as apostas como os eventos.

2. Restrições do Projeto

Num projeto de desenvolvimento de software existem, genericamente, três tipos de restrições: tempo, âmbito e custo. O projeto está constrangidos a nível de tempo, sendo que a data para a entrega da segunda fase está fixada para o dia 2 de dezembro de 2022. No que diz respeito às restrições de custo, estas não são relevantes, dado que não existe um orçamento do projeto. No entanto, estamos limitados em termos de hardware e mão de obra, não havendo possibilidade de contratar mais pessoal ou de investir na expansão do hardware (p.ex cloud). Estamos ainda a utilizar outra API de eventos desportivos, com apenas um número reduzido de chamadas gratuitas. Por fim, as restrições do âmbito do projeto que, como o nome indica, restringem os objetivos e concentram o foco do desenvolvimento.

As secções seguintes definem as restrições aos aspetos não funcionais do sistema.

2.1 RASBet API

Uma das restrições impostas pelo enunciado é a utilização de uma API específica para a obtenção de alguns resultados de eventos desportivos. Esta API, oferece apenas um desporto, limitando inicialmente os desportos disponibilizados pela plataforma. Devolve os dados num formato específico, o que implica o desenvolvimento de uma interface que interaja com estes dados e os descodifique de forma a serem introduzidos na base de dados do projeto. Por vezes os pedidos demoram algum tempo, ou seja, não podemos simplesmente fazer chamadas à API sempre que necessitarmos de dados. Como tal foi implementado um mecanismo para atualizar a nossa base de dados em segundo plano, não sobrecarregando a API e oferecendo uma experiência de navegação rápida pela plataforma. A utilização desta API restringe portanto o design do controlador da aplicação, ainda que não tenha muito impacto na performance do serviço.

2.2 The Odds API

Por motivos de expansão, de forma a oferecer mais desportos e mais tipos de apostas, foi integrada uma outra API de eventos desportivos. No entanto, esta API tem um número limitado e reduzido de chamadas gratuitas. Para além disto, o formato dos dados muda subtilmente, sendo necessário adaptar ligeiramente o código para integrar a informação na base de dados. Por consequência, o design do controlador tem de ser adaptado para limitar ainda mais o número de chamadas a esta API.

2.3 Spring Security

Na segunda fase do projeto, para implementação de requisitos não funcionais relacionados com a segurança do sistema, foi necessário recorrer ao módulo Spring Security. Esta funcionalidade restringe o acesso à plataforma sendo que, mais uma vez, tem impacto em decisões tomadas acerca da implementação da solução.

3. Contexto e Âmbito do Projeto

Neste capítulo pretendemos obter uma visão geral do projeto, delimitando o sistema pelas suas comunicações externas e explicando o modo como cada interface vai interagir com o sistema e vice-versa.

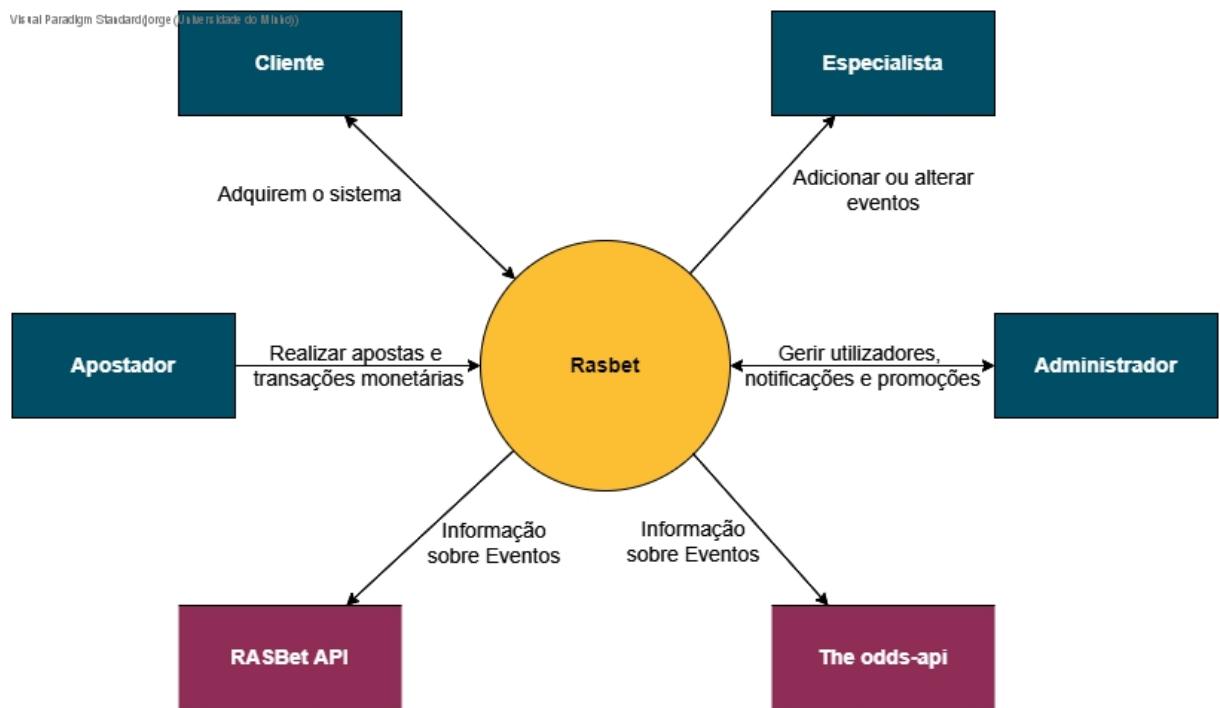


Figura 3.1: Diagrama de *Contexto*

3.1 Cliente

Os clientes do *Rasbet* são as empresas que estão no ramo de apostas em eventos desportivos, e que pretendem vender esta solução aos seus clientes (casas de apostas), ou seja, um produto informático de suporte a apostas desportivas de alta usabilidade, e consequentemente utilização simples, instantânea e intuitiva, para realizar as suas apostas.

3.2 Apostador

O Apostador é o usufruidor da aplicação, tendo acesso a todas as funcionalidades que a nossa plataforma disponibiliza ao público. Além das apostas em si é pretendido que usa funcionalidades de auxílio as apostas como levantar dinheiro, depositar dinheiro ou consultar o seu histórico de transações.

3.3 Especialista

O Especialista é o ator que se responsabiliza pela atualização dos eventos, ou seja, para além do login, este tem acesso apenas a alterar odds e a adição de novos eventos.

3.4 Administrador

O Administrador do sistema é responsável pelo registo de outros administradores e também dos especialistas. Além disso tem a possibilidade de controlar o estado de um evento e também criar e gerir notificações e promoções.

3.5 RASBet API e The odds-api

RASBet API e The odds-api são as fontes de informação do Rasbet, ou seja onde toda a informação dos eventos e das suas odds é extraída, bem como os seus resultados.

4. Estratégia da Solução do Projeto

A implementação do projeto levou-nos a ter que tomar algumas decisões quanto a solução arquitetural do projeto. Como podemos ver na figura abaixo optamos por uma solução simples que pode ser facilmente explicada pelos três padrões arquiteturais utilizados:

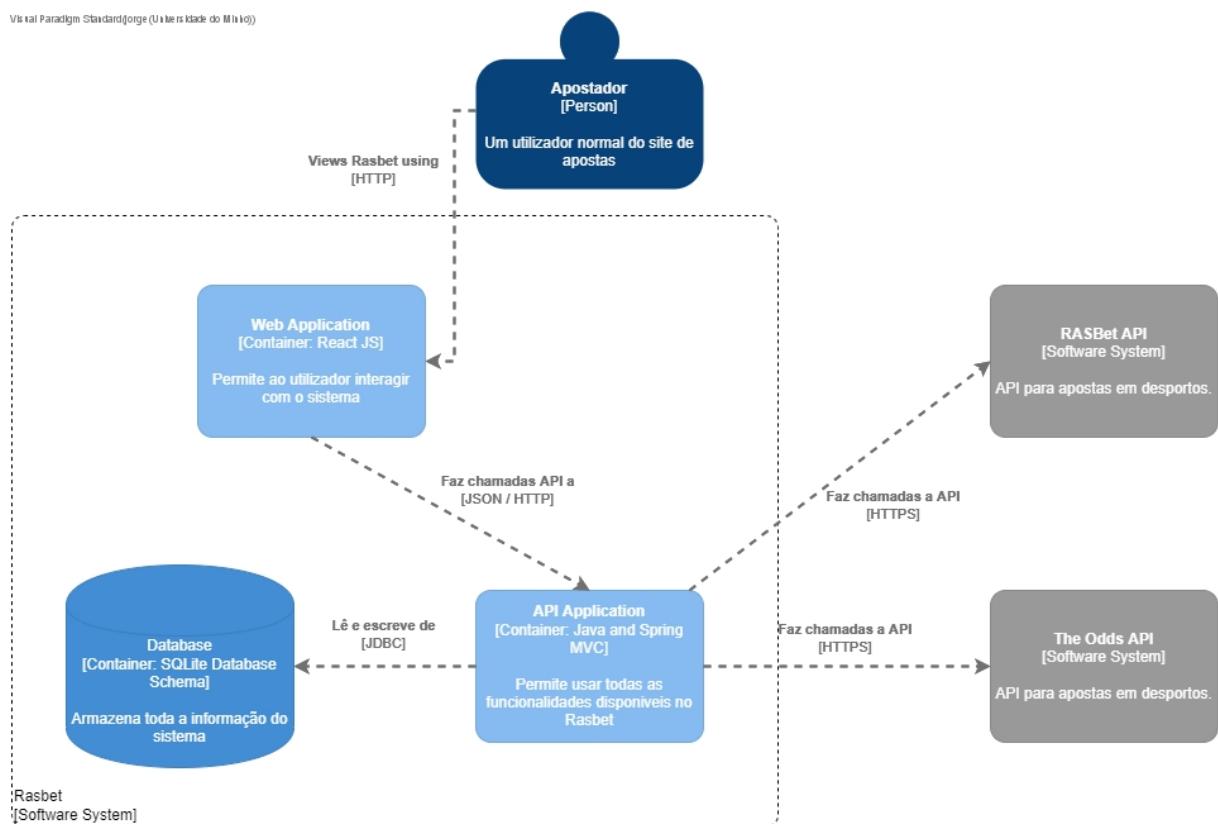


Figura 4.1: Diagrama C4 Model Container

4.1 Client-server pattern

O modelo cliente-servidor é uma estrutura de uma aplicação distribuída que distribui as tarefas e cargas de trabalho entre servidores, fornecedores de um ou vários serviços, e clientes, utilizadores do serviço, que comunicam através de uma rede de computadores.

Um servidor é um host que disponibiliza funcionalidades e compartilha informação com os clientes. No nosso caso o servidor é implementado com Spring Boot, uma framework que nos permite criar uma REST API. Esta framework foi escolhida pois tem a vantagem de automatizar toda a configuração, sendo assim de fácil incorporação no projeto.

Os clientes iniciam sessões de comunicação e fazem vários pedidos com o servidor, que podem aceitar esses pedidos, processá-los e retornar as informações solicitadas de volta ao cliente, que neste programa é uma interface web construída em React.

4.2 Model-view-controller pattern

Conseguimos facilmente dividir o padrão MVC na nossa solução do Rasbet:

A componente view também corresponde a componente cliente no modelo cliente-servidor pois é a interface em que o utilizador vai usar para interagir com o sistema.

O controller do programa interpreta todos os pedidos que o utilizador fez na interface e atualiza o model, no nosso caso, depois de o utilizador fazer um pedido, como uma aposta, o controller deve enviar-lha ao model e deixar que este atualize os dados.

Como já referido o model é a principal componente do programa, atualizando toda a informação que recebe do controller e da view, no caso do Rasbet, implica atualizar toda a base de dados, sendo assim é nesta componente em que é aplicada toda a logica dos pedidos, por exemplo, no caso das apostas, a sua introdução, verificação do resultado, introdução da notificação, atualização da carteira do utilizador, ...

4.3 Master-slave pattern

Na construção da solução surgiu um problema na ligação as APIs para obter informação. Tendo em conta que queremos um tempo de resposta aos utilizadores, precisamos de ter algum mecanismo que nos permite não esperar pela resposta da API quando é feito um pedido do cliente. Para isso aplicamos o padrão Master-slave, ou seja, quando o programa é iniciado o programa principal cria uma slave thread que vai ser sempre responsável pela ligação as APIs e a atualização da informação obtida.

Ou seja, sempre que chega um pedido de um cliente a master thread, este acorda a thread slave que começa por verificar se já passaram 5 minutos desde os últimos pedidos a API e em caso positivo faz as chamadas e o processamento da informação. No fim a thread slave volta a esperar por um sinal da master thread.

5. Vista em Bloco

Neste capítulo é apresentada uma visão em bloco do sistema. Para tal é fornecido um diagrama de componentes que descreve, de forma abstracta, os componentes principais do sistema e a sua interação. Posteriormente apresentamos uma visão do sistema por classes.

5.1 Diagrama de Componentes

Tal como podemos ver na figura abaixo, o sistema tem quatro componentes fundamentais: a interface do utilizador (RASBet UI), que possibilita a interação do utilizador com o sistema; o *Controller*, que gera os pedidos *http* vindos da UI, autenticando os utilizadores e implementando a lógica da aplicação; a base de dados (*Database*) que armazena todos os dados do sistema e, por fim, o *Events*, que trata, paralelamente, de atualizar a base de dados com a informação mais atual disponibilizada pelas API's.

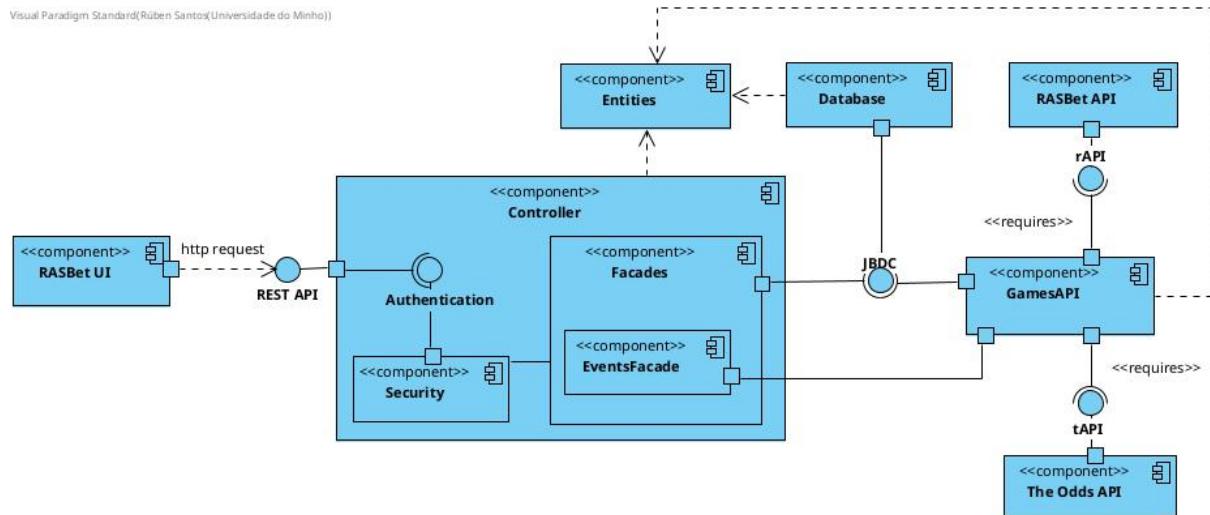


Figura 5.1: Diagrama de *Componentes*.

Se olharmos com algum detalhe, o controlador depende de um outro componente *Entities*, que define as entidades do sistema. Sem este componente é impossível traduzir os dados de forma a serem introduzidos na base de dados.

O componente *Security* é responsável por autenticar os utilizadores, tanto por fornecer um *token* aos utilizadores já registados mas não autenticados, como por validar um *token* recebido num pedido.

As classes que implementam a lógica do sistema estão contidas no componente *Facades*. Valeu a pena realçar o papel do componente *EventsFacade*, responsável pelo tratamento dos dados de ambas as API's, que por sua vez interage com o componente *Events*, responsável por obter os dados das mesmas.

5.2 Diagrama de Classes

Dado o elevado número de classes, decidimos fazer um diagrama de classes para cada *package*. No diagrama abaixo podemos, de uma forma ainda mais genérica do que no diagrama de componentes, visualizar as dependências entre *pacakges*.

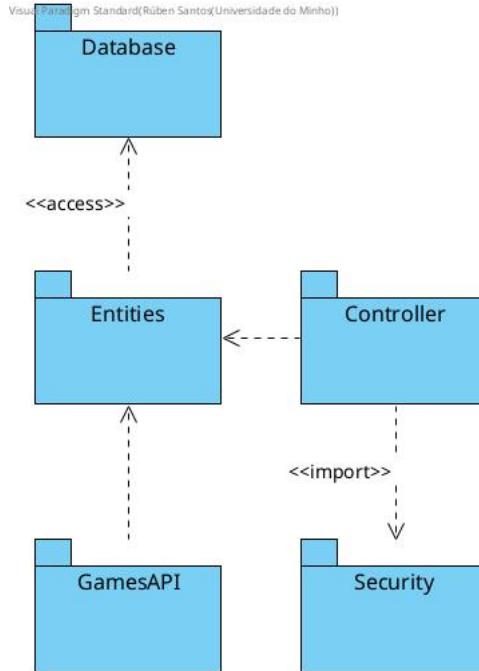


Figura 5.2: Diagrama de *Packages*.

Nas secções seguintes apresentamos, para cada *package*, o respetivo diagrama de classes.

5.2.1 Diagrama de classes : Controller

Neste diagrama estão representadas as classes do *package* Controller. Estas classes são controladores REST, que contém os *endpoints* da aplicação. Não existem dependências entre elas.

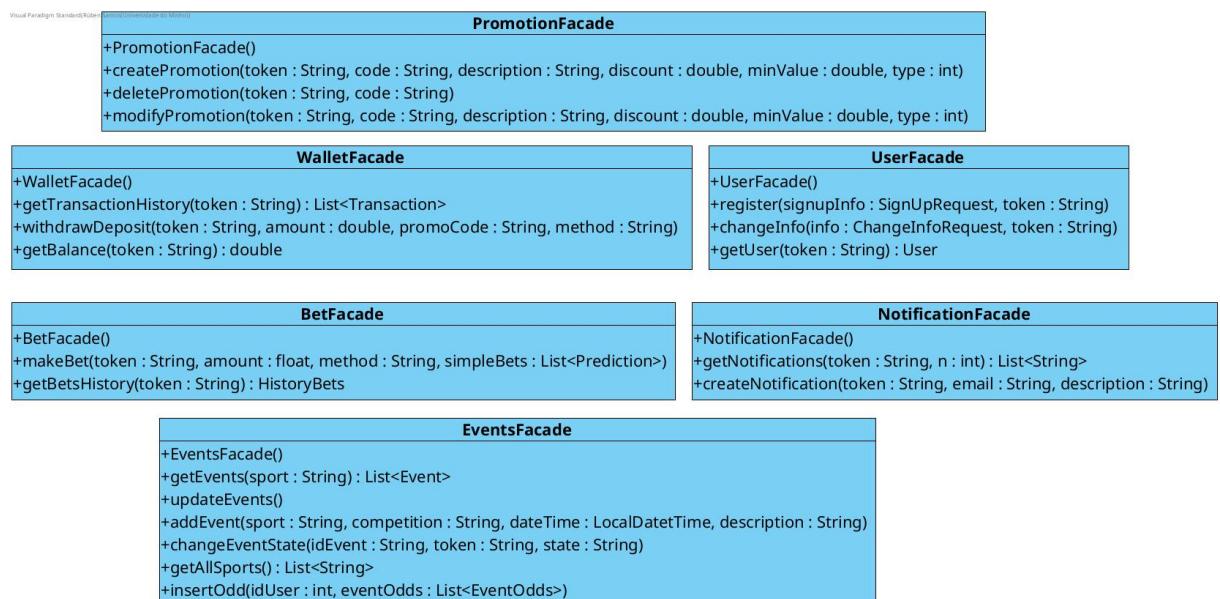


Figura 5.3: Diagrama de Classes (Controller).

5.2.2 Diagrama de classes : Database

Neste diagrama estão representadas as classes do package Database. Estas classes fornecem uma interface de acesso à base de dados. Todas dependem da classe SQLiteJDBC, que trata da gestão das conexões à base de dados.

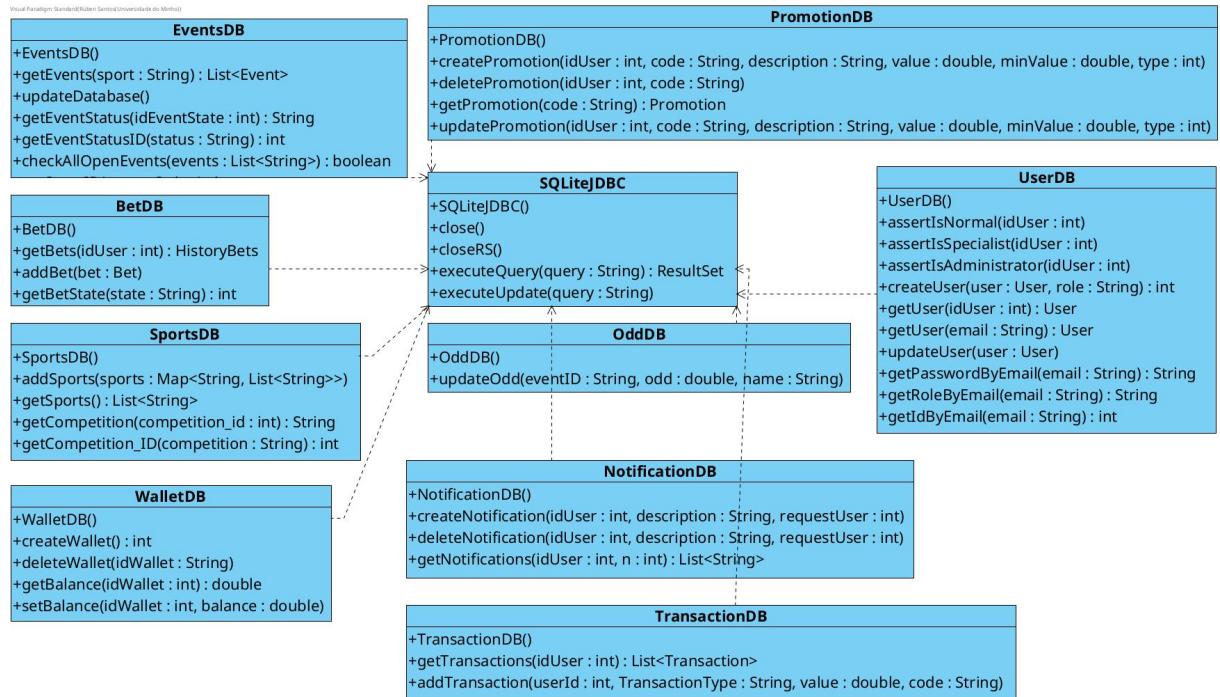


Figura 5.4: Diagrama de Classes (Database).

5.2.3 Diagrama de classes : Entities

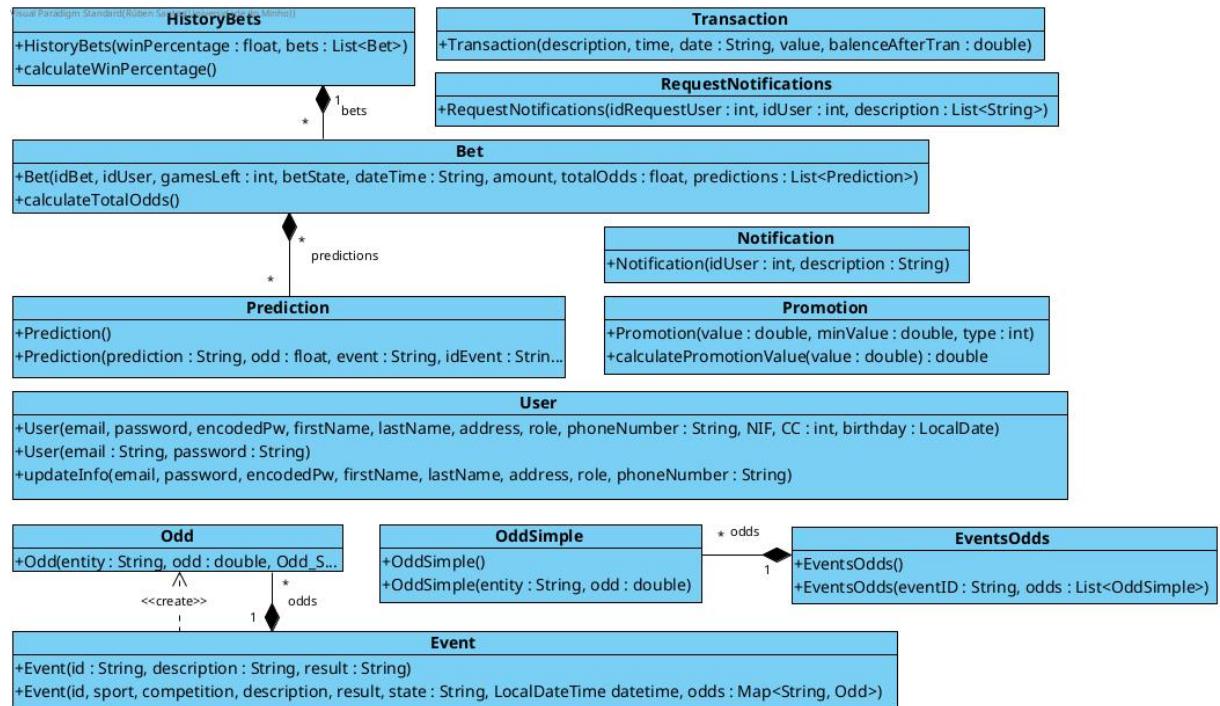


Figura 5.5: Diagrama de Classes (Entities).

No diagrama acima estão representadas as classes do *package* Entities. Estas definem os tipos utilizados em toda a aplicação.

5.2.4 Diagrama de classes : GamesAPI

Neste diagrama estão representadas as classes do *package* GamesAPI. Estas duas classes são responsáveis pela interação com as duas API's que fornecem informação acerca de eventos desportivos.

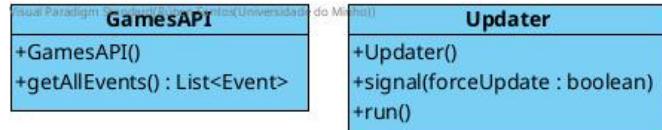


Figura 5.6: Diagrama de Classes (GamesAPI).

5.2.5 Diagrama de classes : Security

Neste diagrama estão representadas as classes do *package* Security. Estas classes são responsáveis pela gestão da autenticação dos utilizadores, assim como a codificação das palavras passe.

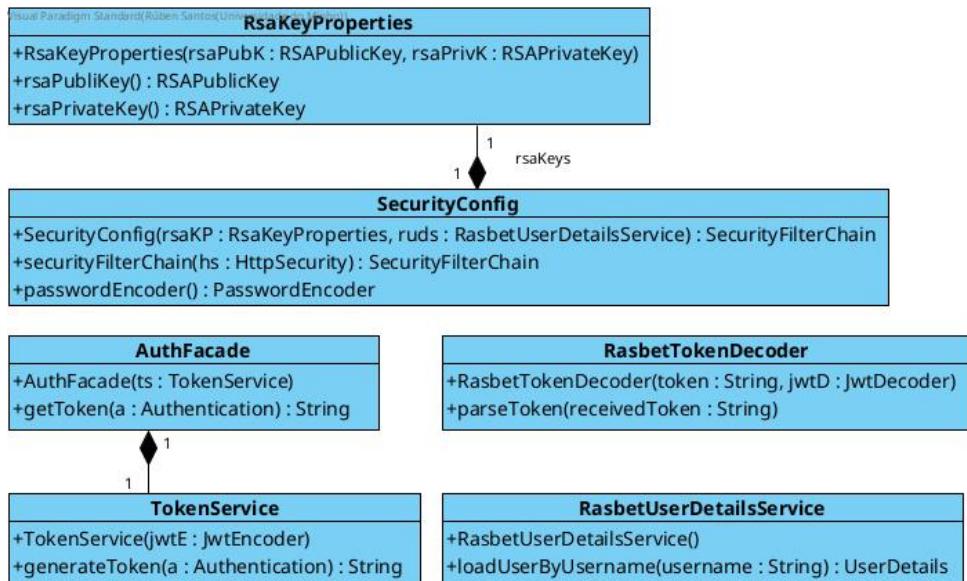


Figura 5.7: Diagrama de Classes (Security).

6. Vista em Tempo de Execução

Neste capítulo é apresentada uma visão da execução do sistema. É utilizado diagramas de sequências para três dos casos mais utilizados.

6.1 Diagrama de sequência: getEvents

No seguinte diagrama é apresentado o diagrama de sequência da execução de um pedido dos eventos pelo utilizador.

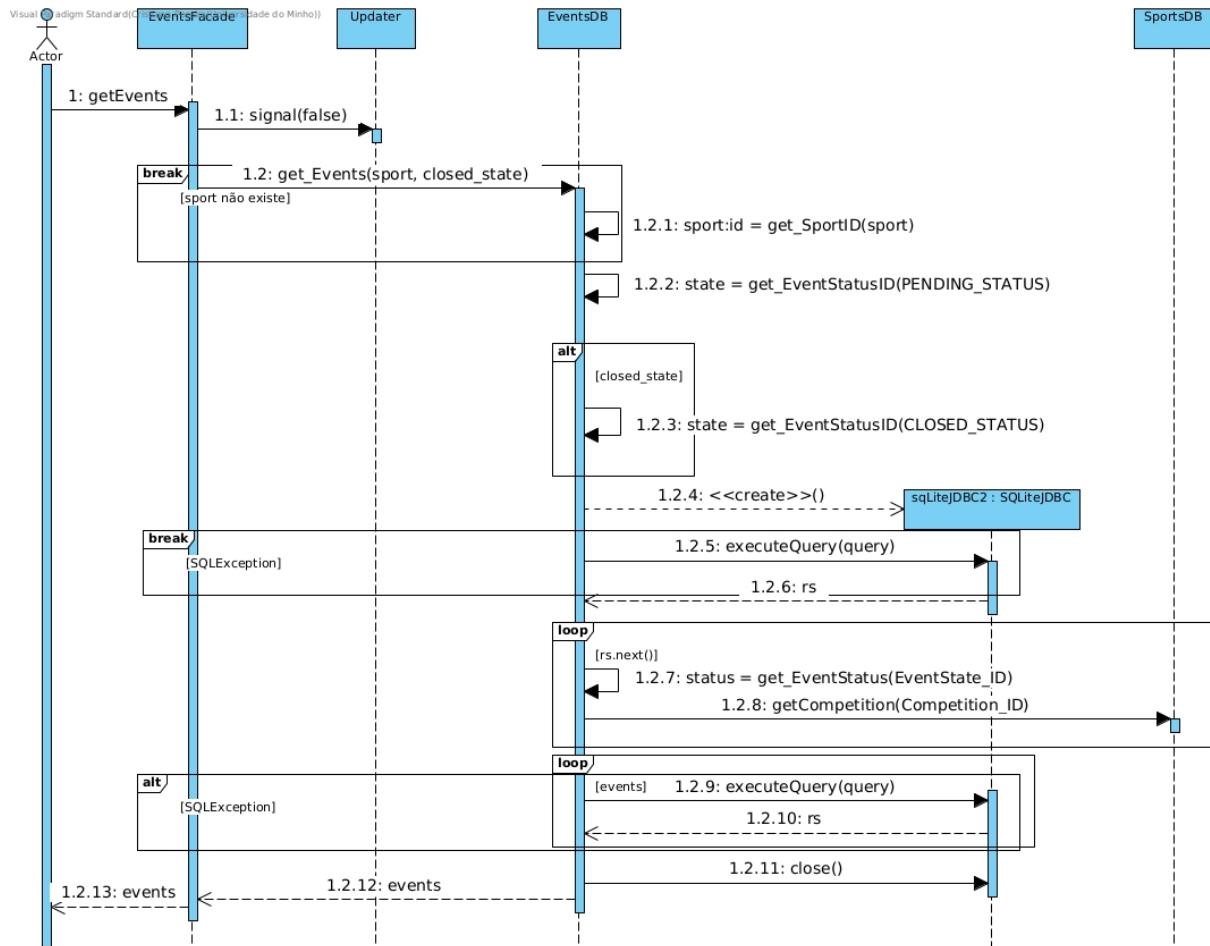


Figura 6.1: Diagrama de Sequência.

A classe EventsFacade irá receber os pedidos dos eventos e irá executar o método para obter os eventos da classe EventsDB, classe com os métodos que fazem as queries à base de dados relacionados com os eventos.

Aqui será verificado se o desporto existe na base de dados, aonde, se o desporto existir, será feita a obtenção dos eventos na base de dados. Fazendo-se um loop para escrever os dados de

cada evento e outro loop para obter as odds.

6.2 Diagrama de sequência: makeBet

Em seguida, é apresentado o diagrama de sequência de fazer uma nova aposta.

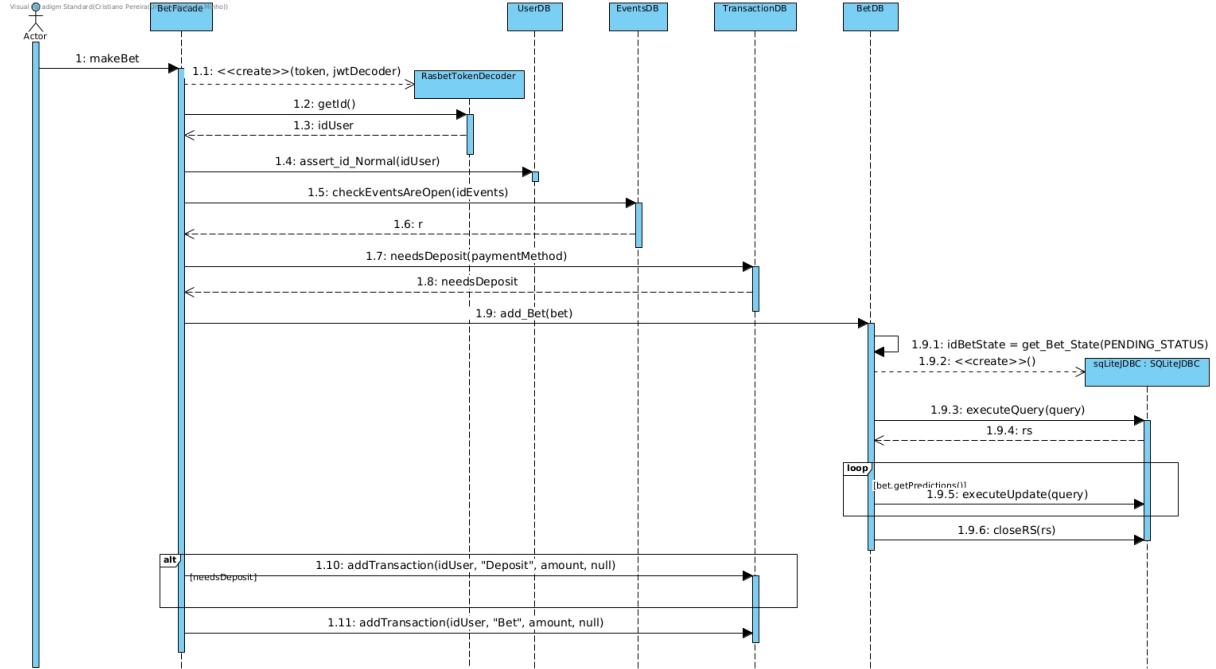


Figura 6.2: Diagrama de Sequência.

A classe BetFacade irá receber os pedidos para adicionar apostas do utilizador. Em primeiro lugar, o token e o role do utilizador serão validados de modo a confirmar se o pedido é feito de uma conta autenticada e se um administrador ou especialista não conseguir executar apostas, respetivamente. Em seguida, será verificado se os eventos estão abertos a novas apostas e se o utilizador irá fazer um depósito ou utilizar a sua carteira para pagar a aposta. Finalizando com a adição da aposta à base de dados e da transação feita pelo utilizador para pagar a aposta.

6.3 Diagrama de sequência: login

Em seguida é apresentado o diagrama de sequência da execução do login de um utilizador no sistema.

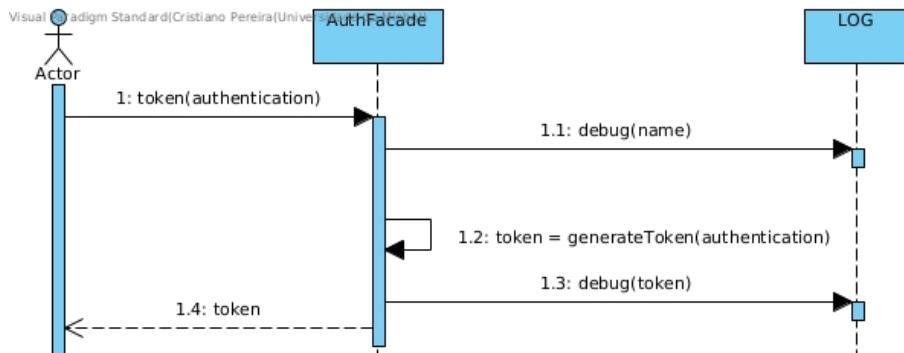


Figura 6.3: Diagrama de Sequência.

7. Vista de Deployment

De forma a apresentar uma visão da instalação do sistema, apresentamos um diagrama de instalação (deployment). Aonde são apresentados os principais nodos para um bom funcionamento deste.

7.1 Diagrama de Instalação

O funcionamento do sistema depende de três nodos principais, o servidor aonde o backend da aplicação e a base de dados estão a correr, o servidor aonde o website irá executar e um dispositivo com um browser aonde o utilizador poderá aceder ao website.

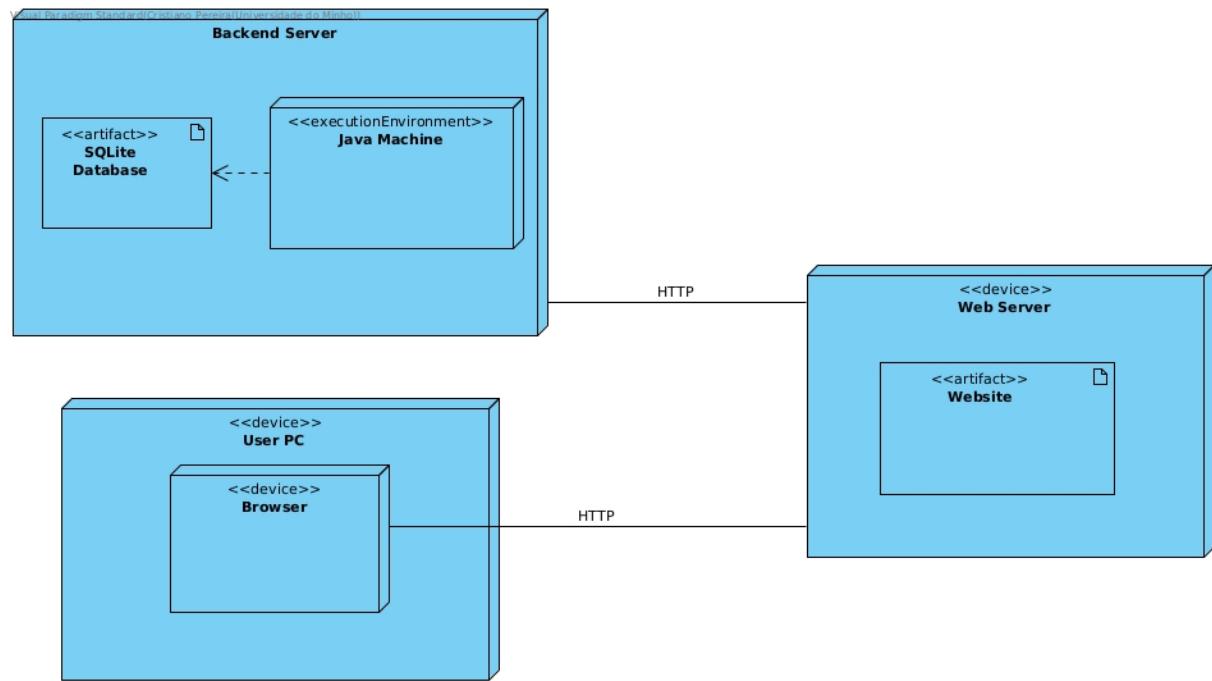


Figura 7.1: Diagrama de *Instalação*.