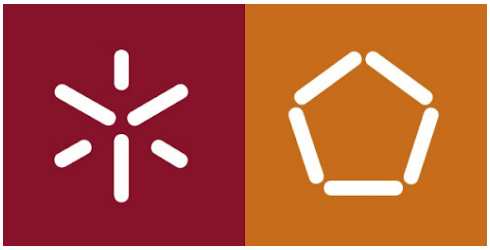

Universidade do Minho

Mestrado em Engenharia Informática

RasBet

Requisitos e Arquiteturas de Software

Guilherme Martins (pg47225)
José Santos (a84288)
Leonardo Marreiros (pg47398)
Pedro Fernandes (pg47559)

17 de dezembro de 2021

Conteúdo

Lista de Figuras

Lista de Tabelas

1 Introdução e Objetivos

A RasBet é uma aplicação de apostas desportivas dos Jogos Santa Casa, onde se poderá apostar em eventos de vários desportos. A cada acontecimento está associada uma *odd* que, multiplicada pelo valor da aposta, determina os seus ganhos possíveis (prémio).

1.1 Visão geral dos requisitos

O principal propósito da RasBet é proporcionar aos aficionados de desporto e apostas uma forma de manifestarem as suas opiniões prevendo resultados desportivos ao colocar em jogo uma quantia monetária nos desfechos destes eventos.

De forma simples, os requisitos funcionais principais são os seguintes:

- Um apostador pode consultar a lista de eventos ativos.
- Um apostador pode criar um boletim de apostas.
- Um apostador pode ver o seu histórico de apostas.
- Um apostador pode ver o estado das suas apostas.
- Um apostador é notificado com o resultado dos eventos em que apostou.
- Um especialista adiciona, cancela e termina eventos.

Um descrição mais detalhada dos requisitos pode ser encontrada no documento de requisitos da RasBet.

1.2 Objetivos de qualidade

Prioridade	**Qualidade**	**Motivação**
1	Aptidão funcional	O sistema contém funções que vão de acordo às necessidades estabelecidas.
2	Eficiência	Criar um boletim de apostas deve ser uma tarefa simples. Escolher os acontecimentos que pretende apostar e inserir o montante a apostar.
3	Compreensibilidade	Os requisitos funcionais são simples o suficiente para permitir uma solução simples e compreensível.
4	Performance	A obtenção de dados como a lista de eventos ativos deve demorar menos de 3 segundos.
5	Atratividade	As listas de eventos, históricos, boletins, etc, apresentados devem ser fáceis de entender e minimalistas.

Tabela 1: Objetivos de qualidade

1.3 *Stakeholders*

Papel	**Objetivo**
Apostadores	Apostadores aficionados de desporto que procuram uma aplicação fácil de utilizar, confiável e onde podem maximizar os seus lucros.
Bancos	Os bancos poderão financiar monetariamente o sistema. O banco pode retornar o seu investimento com as apostas falhadas pelos apostadores.
Sponsors	Entidades como eventos, influências, clubes, etc podem dar maior visibilidade ao sistema em troco de contratos monetários. Da mesma forma, clubes ou atletas menos conhecidos podem ganhar visibilidade ao usar a marca do sistema.
Developers	O desenvolvimento do produto irá gerar oportunidades de emprego na área de desenvolvimento de *software*.
Advogados	Sendo este um sistema de apostas desportivas, é natural que exista regulação legislativa rigorosa para a sua aprovação. O desenvolvimento do produto promoverá a criação de trabalho na área legal.
CNPD	O produto a desenvolver deverá estar de acordo as normas da CNPD para a gestão e utilização dos dados recolhidos ao utilizador.
Casas de apostas	*Stakeholder* negativo. O sucesso da aplicação, pode pôr em causa o possível lucro que estas poderiam fazer.

Tabela 2: Stakeholders

2 Restrições

2.1 Restrições Técnicas

	Restrição	**Motivação**
Restrições de software e programação		
RT1	Implementação em Java	A aplicação deve ser desenvolvida em Java, utilizando Spring Boot.
RT2	Base de Dados *MySQL*	A aplicação deve utilizar uma base de dados MySQL.
RT3	*Frontend* em React	O *frontend* da aplicação deve ser desenvolvido em React.
RT4	O software de terceiros deve estar disponível sob uma licença *open source* e instalável via *package manager*	Deve ser possível verificar as fontes, compilar e executar a aplicação sem problemas de compilação ou instalação de dependências. Todas as dependências externas devem estar disponíveis por meio do *package manager* do sistema operativo ou pelo menos por meio de um instalador.
Restrições do sistema operativo		
RT5	Desenvolvimento independente do sistema operativo	A aplicação deve ser compilável em todos os três sistemas operativo principais (Mac OS X, Linux e Windows).
Restrições de Hardware		
RT6	Baixa utilização de memória	A aplicação deve consumir poucos recursos de memória, mantendo apenas a *cache* necessária.

Tabela 3: Restrições técnicas

2.2 Restrições Organizacionais

	Restrição	**Motivação**
RO1	Equipa	Equipa de engenheiros informáticos da Universidade do Minho contratados pelos Jogos Santa Casa.
RO2	Agenda	A aplicação deve estar operacional para o Desporto Principal (Futebol) até dia 1 de janeiro de 2022. No entanto a adição dos restantes desportos deverá estar operacional até meados de janeiro do mesmo ano.
RO3	Orçamento	O orçamento total para o desenvolvimento da aplicação, incluindo contratação de engenheiro de requisitos, arquitetos de *software*, testador e *developers* deverá rondar os 62 400 €.
RO4	Controlo de versões	Repositório git privado com um histórico de *commits* completo e um *branch master* público no GitHub.

Tabela 4: Restrições Organizacionais

3 Delimitação do Contexto

3.1 Contexto de negócio

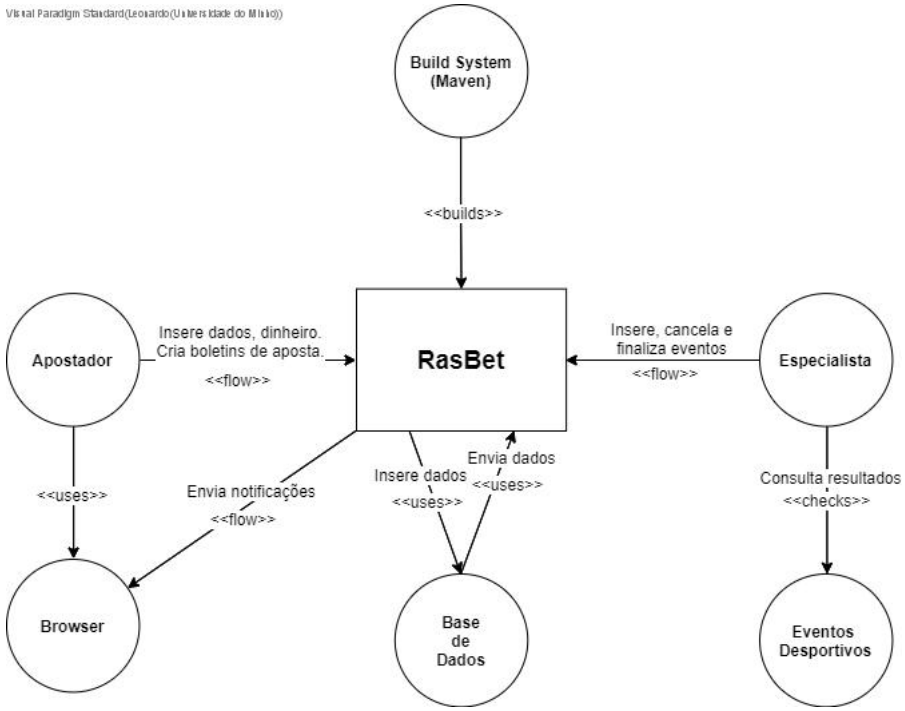

Figura 1: Diagrama de contexto de negócio

Apostador

Um apostador dedicado utiliza a Rasbet para ver os próximos eventos nos quais pode apostar, cria os seus boletins e possivelmente ganha dinheiro com as suas apostas acertadas. Também pretende ser capaz de ver o estado das suas apostas e o seu histórico.

Browser

Um apostador utiliza um browser para aceder à RasBet. É neste browser que o apostador irá receber notificações com os resultados dos acontecimentos nos quais apostou.

Base de Dados

A aplicação mantém a informação necessária acerca de utilizadores, eventos, boletins de apostas, etc, na base de dados. Periodicamente, existem consultas e inserções de dados.

Especialista

Entidade encarregue de inserir, cancelar e finalizar eventos. Insere informações acerca de cada evento como as equipas que jogam, *odds* e tipo de aposta. Um especialista vai acompanhado um jogo adicionando os factos do jogo assim como o seu término.

Eventos Desportivos

O especialista consulta os eventos desportivos que vão ter lugar no futuro para inserir no sistema. Do mesmo modo, o especialista consulta os factos dos eventos.

3.2 Contexto técnico

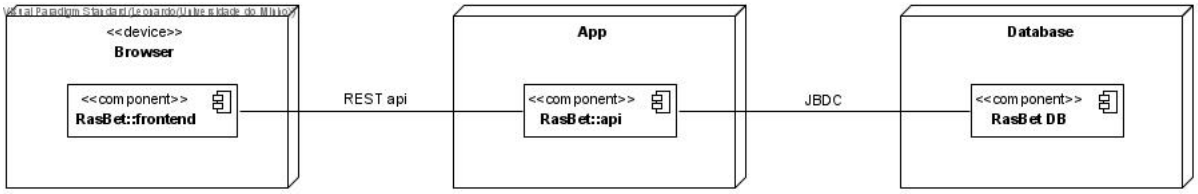

Figura 2: Diagrama de contexto técnico

4 Estratégia da Solução

A aplicação Rasbet pode ser dividida em quatro partes essenciais: *frontend*, *business*, *persistance* e base de dados.

Quanto ao *business* este será construído utilizando a *framework Spring Boot* uma vez que reduz o tempo geral de desenvolvimento e aumenta a eficiência e integração. Por um lado, não há necessidade de entender uma configuração XML complexa e, por outro, fornece serviços de *middleware*.

O objetivo de qualidade relativo à performance será atingido na camada de persistência pelo uso do driver JDBC para conetar à base de dados aliado ao padrão de desenho DAO. Ao manter uma *cache* da lista de eventos ativos em memória, prevenimos acessos demorados à base de dados. Sempre que a tabela referente aos eventos ativos na base de dados é alterada, a lista em memória também o é.

É importante referir que esta lista nunca irá conter uma quantidade significativa de eventos que leve a uma grande ocupação de memória.

A API irá utilizar JSON sobre protocolo HTTP simples. Numa primeira fase, a segurança não é o foco principal desta aplicação. Deve ser segura o suficiente para evitar que outros utilizadores alterem os dados de outros, confidencialidade não é a principal preocupação. Apenas é prevista a implementação de um método muito simples de *hashing* para as palavras-passe dos utilizadores.

Em relação ao *frontend*, será implementada com recurso à ferramenta *React*. Foi escolhida esta biblioteca devido à sua simplicidade e facilidade de uso, não comprometendo a qualidade visual da *interface*.

A atratividade será atingida com a implementação de uma interface minimalista com recurso a poucos cliques para executar as tarefas pretendidas.

5 Vista de Blocos de Construção

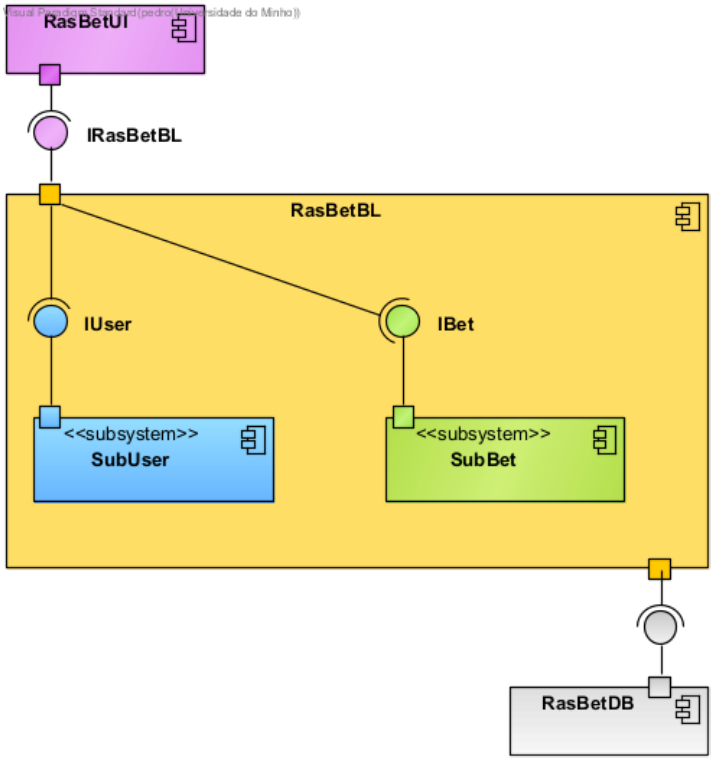

Figura 3: Diagrama de componentes - Nível 0

A lógica de negócio da RasBet é constituida por duas partes: a componente dos utilizadores (*SubUser*) e a componente das apostas (*SubBet*). Além disso, existe também uma componente de interface gráfica com o utilizador e uma base de dados.

A separação de componentes em utilizadores e apostas é uma decisão lógica pois constituem partes praticamente independentes do sistema. Para estes dois componentes irá ser feita uma análise mais detalhada nas seguintes secções. Quanto à interface gráfica com o utilizador (*RasBetUI*), esta será desenvolvida utilizando React cuja documentação pode ser encontrada aqui.

***Blackboxes* Contidas**

Nome	**Responsabilidade**
SubUser	Trata de gerir especialistas e apostadores assim como as suas notificações.
SubBet	Trata de gerir eventos, as suas ocorrências e boletins de apostas.

Tabela 5: Descrição das *blackboxes*

Interfaces

Nome	Responsabilidade
IRasBetBL	REST api. Contém métodos para ler, escrever, editar dados do sistema .

Tabela 6: Descrição das interfaces

5.1 RasBet::SubUser

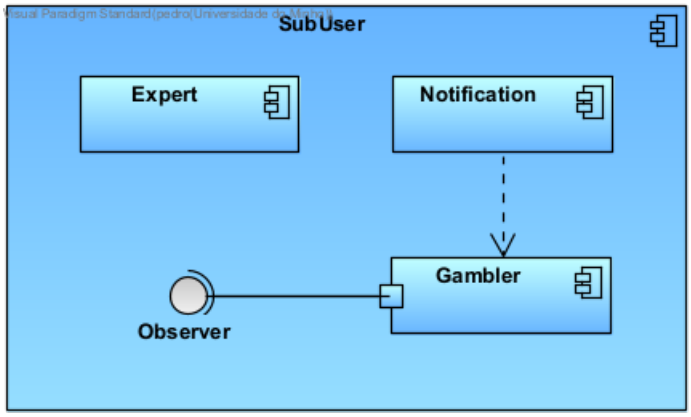

Figura 4: Diagrama de componentes *SubUser* - Nível 1

***Blackboxes* Contidas**

Nome	Responsabilidade
Gambler	Objeto que representa um apostador.
Expert	Objeto que representa um especialista.
Notification	Objeto que representa uma notificação.

Tabela 7: Descrição das *black boxes* do componente SubUser

Interfaces

Nome	Responsabilidade
Observer	Contém métodos para enviar notificações quando o estado de um evento muda.

Tabela 8: Descrição das interfaces

5.2 RasBet::SubBet

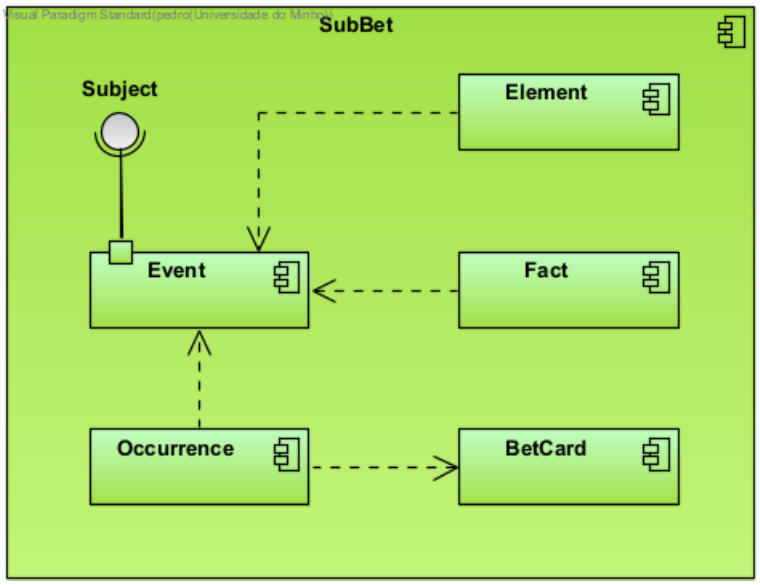

Figura 5: Diagrama de componentes *SubBet* - Nível 1

Blackboxes Contidas

Nome	**Responsabilidade**
Element	Objeto que representa uma equipa/participante.
Fact	Objeto que representa um facto.
Event	Objeto que representa um evento.
Occurrence	Objeto que representa uma ocorrência.
BetCard	Objeto que representa um boletim de aposta.

Tabela 9: Descrição das *black boxes* do componente SubBet

Interfaces

Nome	**Responsabilidade**
Subject	Contém métodos para inserir, remover e notificar apostadores interessados em eventos.

Tabela 10: Descrição das interfaces

5.3 RasBet::DB

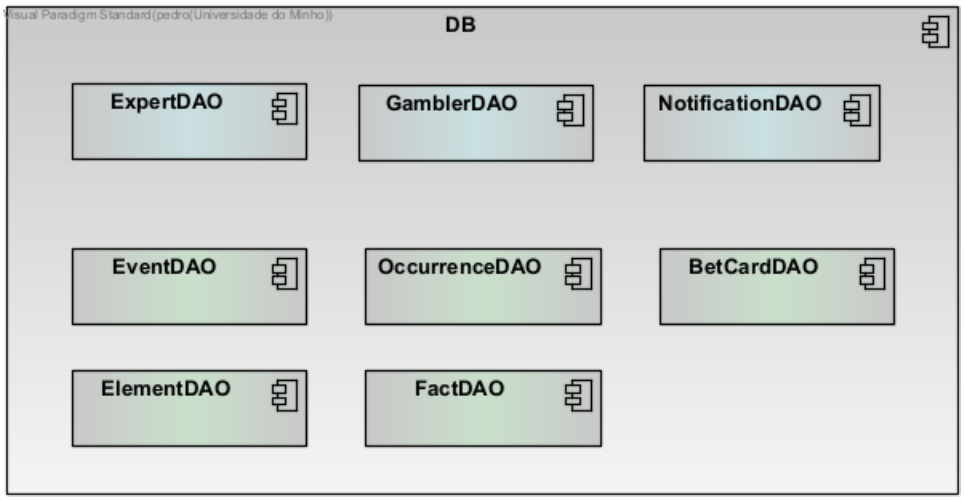

Figura 6: Diagrama de componentes *DB* - Nível 1

Blackboxes Contidas

Nome	**Responsabilidade**
GamblerDAO	Usado para aceder a dados da base de dados referentes a apostadores.
ExpertDAO	Usado para aceder a dados da base de dados referentes a especialistas.
NotificationDAO	Usado para aceder a dados da base de dados referentes a notificações.
ElementDAO	Usado para aceder a dados da base de dados referentes a equipas/participantes de eventos.
FactDAO	Usado para aceder a dados da base de dados referentes a um facto que ocorreu num evento.
EventDAO	Usado para aceder a dados da base de dados referentes a eventos.
OccurrenceDAO	Usado para aceder a dados da base de dados referentes uma ocorrência de um evento.
BetCardDAO	Usado para aceder a dados da base de dados referentes a boletins de aposta.

Tabela 11: Descrição das *black boxes* do componente DB

6 Vista de Execução

A interação do utilizador com a aplicação RasBet é bastante simples. De seguida são apresentados alguns exemplos de diagramas de sequência para as principais funcionalidades do sistema. Existem algumas funcionalidades cujos métodos em que se materializam são muito semelhantes. Nesse caso foi apenas realizado um diagrama de sequência referente ao método de uma delas.

6.1 Registar utilizador

O método *registerUser* tem como função, tal como o próprio nome indica, registar um novo utilizador na aplicação.

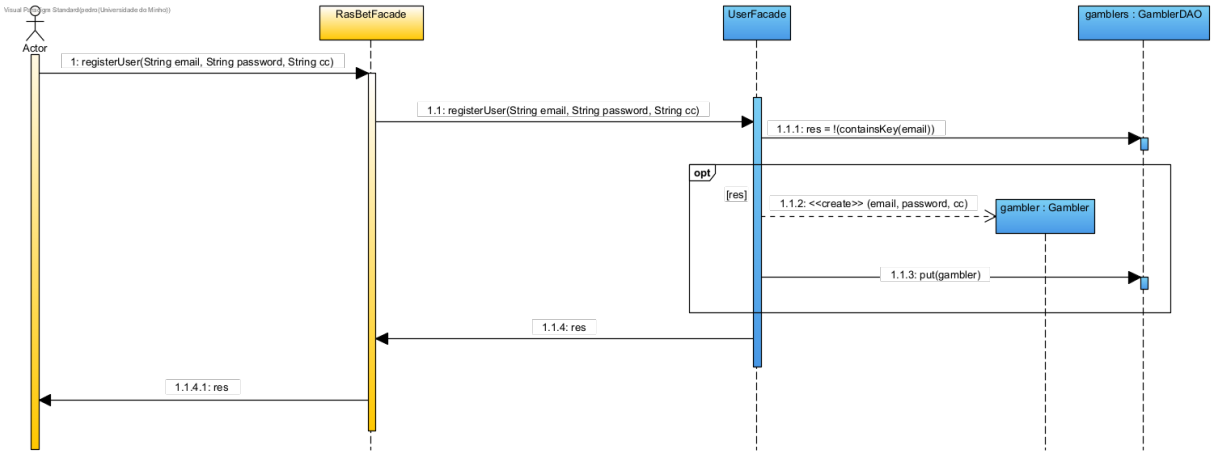

Figura 7: Diagrama de Sequência: Registar utilizador

6.2 Autenticar utilizador

Após um utilizador estar registado, para entrar novamente na aplicação é necessário proceder à validação dos seus dados de acesso. Para isso foi necessário a introdução do método *authUser*. A lógica por detrás deste é análoga à lógica do método *registerUser* com a única diferença ser que em vez de adicionar registos, vai apenas verificar se existe um registo cujas credenciais sejam as mesmas que o *input* da autenticação.

6.3 Alterar *email* ou *password* do utilizador

Por algum motivo um utilizador pode querer alterar algum dos campos relativos à sua conta daí a necessidade dos métodos *updateUserEmail* e *updateUserPass* que, como os seus nomes indicam alteram o *email* e a *password* respectivamente de um utilizador. Devido a estes métodos serem uma simples alteração de uma variável de um objecto, os diagramas de sequência correspondentes não se justificam.

6.4 Depositar dinheiro

Para um apostador(*gambler*) conseguir realizar apostas, necessita de ter algum dinheiro na sua *wallet*. Desse modo é necessário o método *depositMoney* que permite a um *gambler* aumentar o seu saldo.

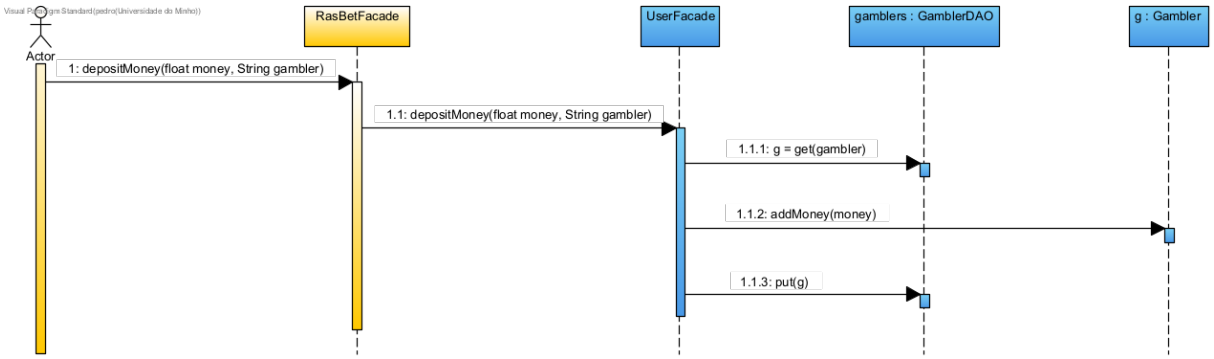

Figura 8: Diagrama de Sequência: Depositar dinheiro

6.5 Levantar dinheiro

Após a realização de uma ou mais apostas, um *gambler* pode querer levantar o seu dinheiro. Assim foi criado o método *withdrawMoney* que apresenta uma grande semelhança com o método *depositMoney* com a única diferença que em vez de aumentar o dinheiro na *wallet*, neste caso ele vai diminuir.

6.6 Consultar lista de eventos ativos

Na aplicação existem diversos *events* ativos em que cada *gambler* pode apostar. Assim o método *listEvents* permite a um *gambler* visualizar a lista de *events* em que pode apostar de momento.

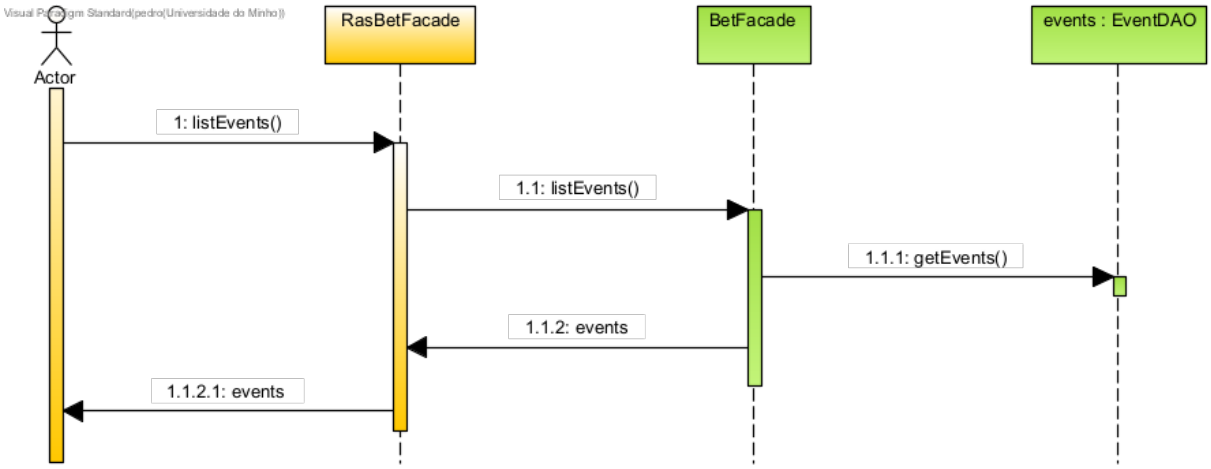

Figura 9: Diagrama de Sequência: Consultar lista de eventos ativos

6.7 Criar boletim de apostas

Fazer uma aposta significa criar um boletim de apostas(*BetCard*). Assim, o método *createBetCard* demonstra como é possível a sua criação.

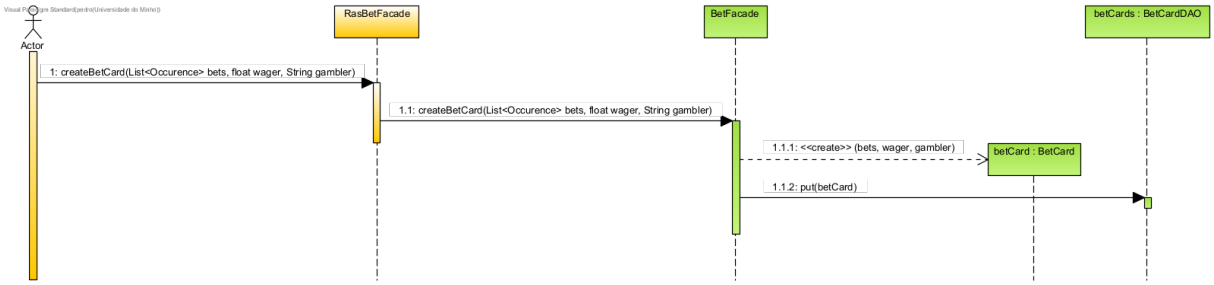

Figura 10: Diagrama de Sequência: Criar boletim de apostas

6.8 Consultar histórico de apostas

Um *gambler* pode querer ver o seu historial de apostas, assim sendo é necessário o método *viewHistory*, que vai mostrar ao utilizador todos os seus boletins já finalizados.

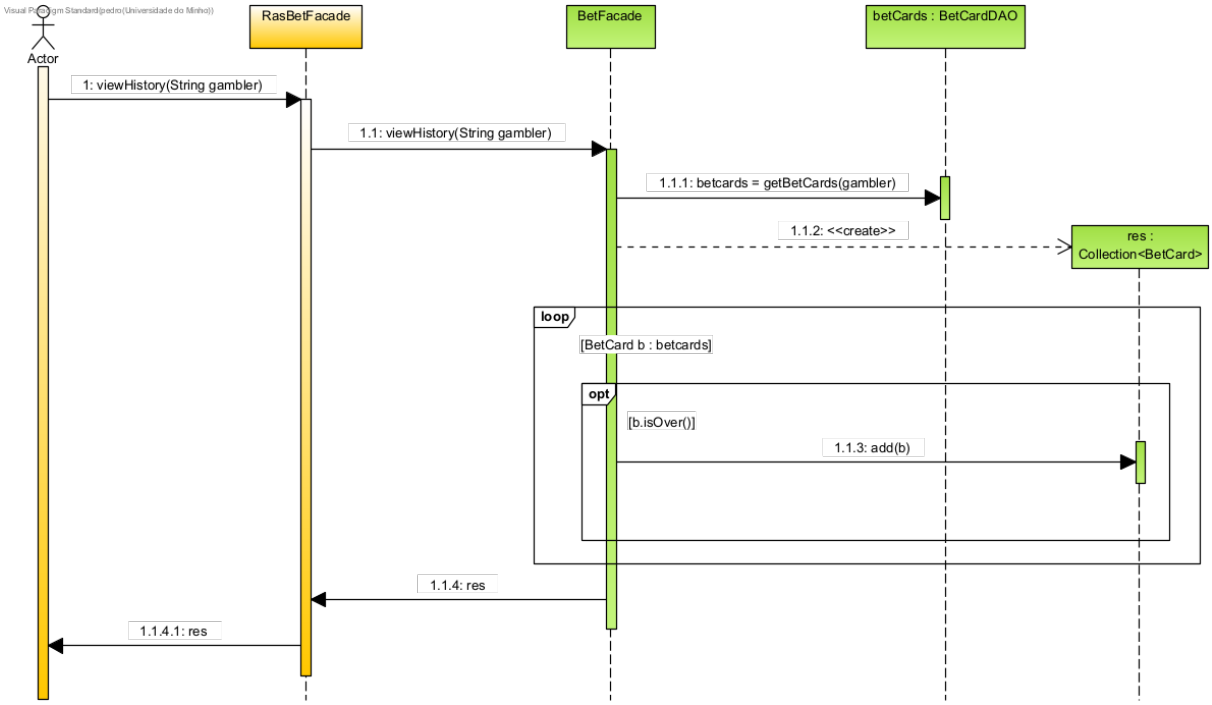

Figura 11: Diagrama de Sequência: Consultar histórico de apostas

6.9 Consultar apostas ativas

Após a criação de um boletim, é natural que um *gambler* queira saber o estado do mesmo ao longo dos eventos. Dessa forma foi criada o método *viewState* que irá mostrar-lhe todos os boletins que ainda não estejam finalizados. Este método é análoga ao *viewHistory*, sendo que a única diferença é que este em vez de mostrar os boletins já acabados, mostra os que ainda têm eventos a decorrer.

6.10 Atribuir prémio

Após o termino de um boletim é necessário fornecer o prémio ao *gambler* que o possui. Desta forma o método *deliverPrize* da forma que se encontra em seguida, vai garantir que o dono de cada boletim recebe o prémio que lhe pertence.

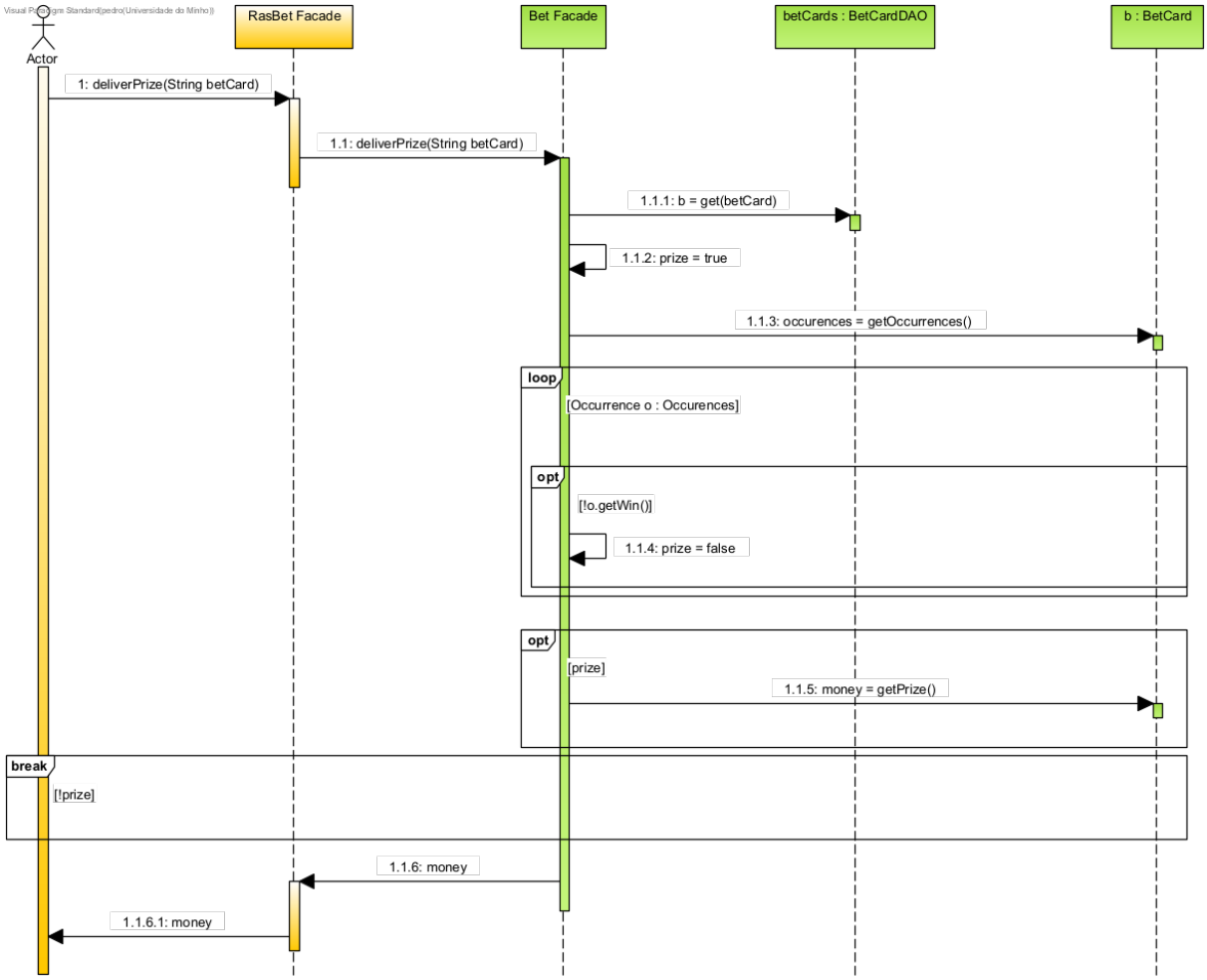

Figura 12: Diagrama de Sequência: Atribuir prémio

6.11 Criar evento

Sempre que vá acontecer um evento de interesse, um especialista(*expert*) irá pretender adicioná-lo à RasBet. De modo a satisfazer tal requisito é necessário recorrer ao método *addEvent* que se encontra de seguida.

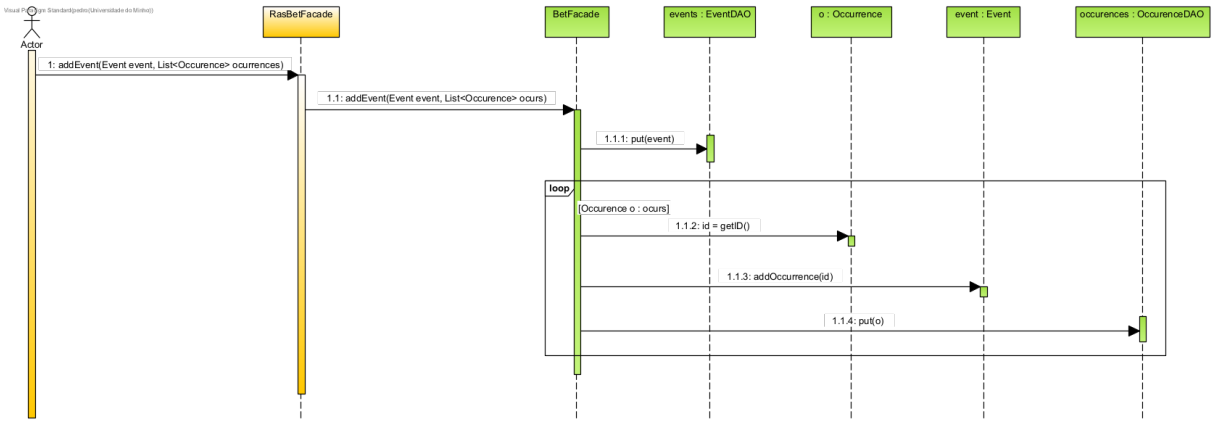

Figura 13: Diagrama de Sequência: Criar evento

6.12 Terminar evento

Quando um evento acaba é necessário que o *expert* adicione essa informação no sistema. Em seguida apresenta-se o diagrama de sequência do método *endEvent* que pretende descrever o mesmo:

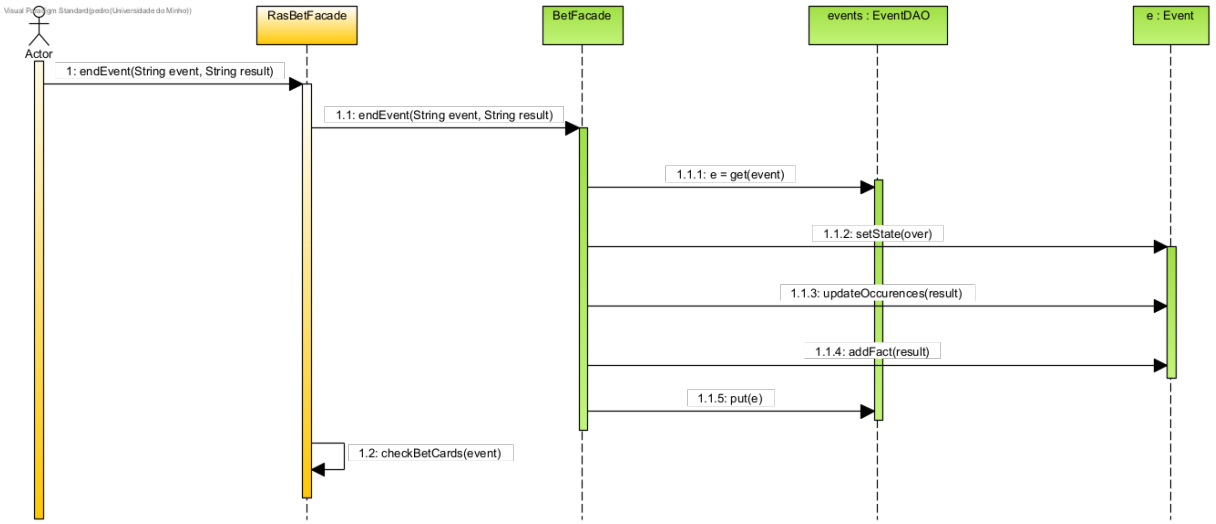

Figura 14: Diagrama de Sequência: Terminar evento

6.13 Cancelar evento

Por algum motivo um evento pode ter um fim inesperado e não deve ser tido em conta para apostas. Dessa forma é necessário que um *expert* possa cancelar determinados eventos que tenham tido um final "irregular", para isso foi criado o método *cancelEvent*. A sua lógica é muito semelhante à do método *endEvent* com a diferença que o evento em vez de ficar finalizado fica cancelado.

6.14 Adicionar facto ao evento

Muitos *gamblers* pretendem ser informados com diversos factos que sucedam durante os eventos em que apostam. Então é necessário que um *expert* adicione informação relevante dos eventos enquanto este decorram. O método *addFactToEvent* irá permitir que o sistema não só armazene os factos relativos de cada evento, mas também que notifique os *gamblers* que estão interessados no mesmo.

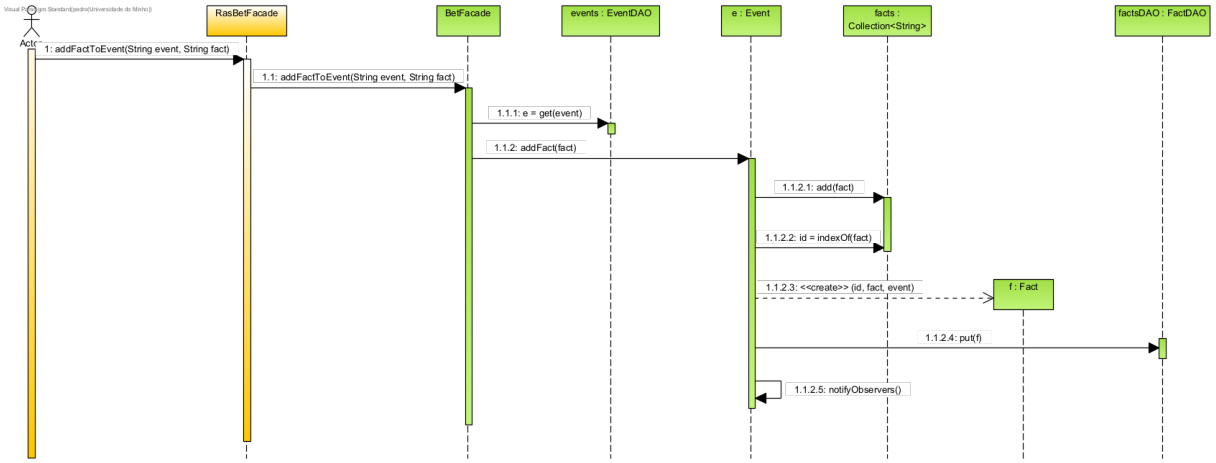

Figura 15: Diagrama de Sequência: Adicionar facto ao evento

7 Visão de Instalação

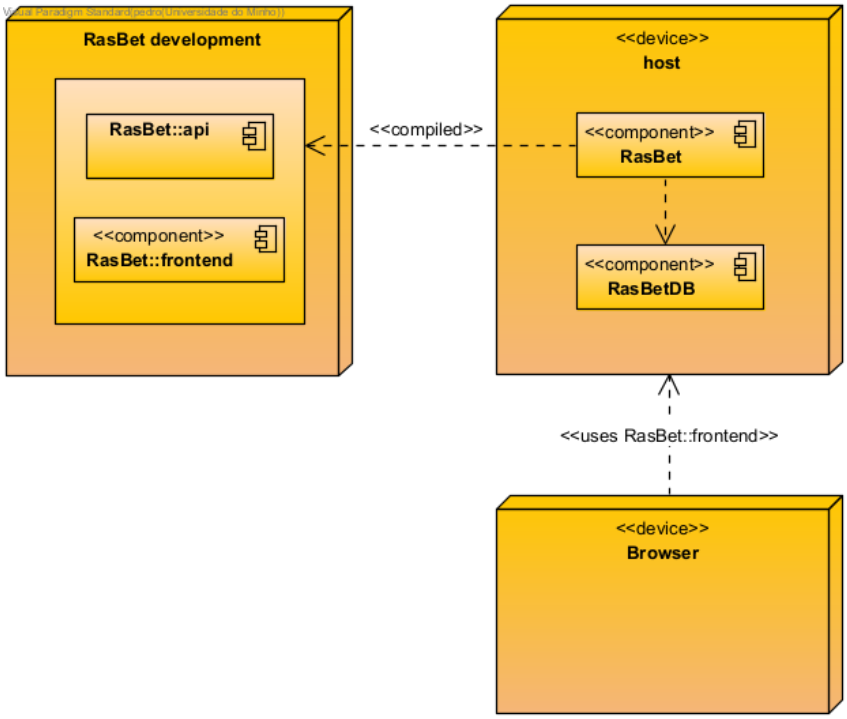

Figura 16: Diagrama de *Deployment*

Componente	**Descrição**
RasBet development	Onde ocorre o desenvolvimento da RasBet, computador com JDK 17 e Maven instalados.
Host	Cliente onde é compilada aplicação.
Browser	Um navegador recente para aceder à interface gráfica com o utilizador da aplicação. Todos os principais navegadores (Chrome, Firefox, Safari, IE / Edge) devem funcionar.
RasBet	Ficheiro compilado a partir dos componentes de desenvolvimento da aplicação e das suas dependências.

Tabela 12: Componentes da instalação

8 Conceitos Transversais

8.1 Conceitos de Domínio

A aplicação Rasbet revolve bastante à volta dos dados que irão estar contidos na base de dados. O modelo ER foi desenvolvido e deu origem ao seguinte modelo lógico.

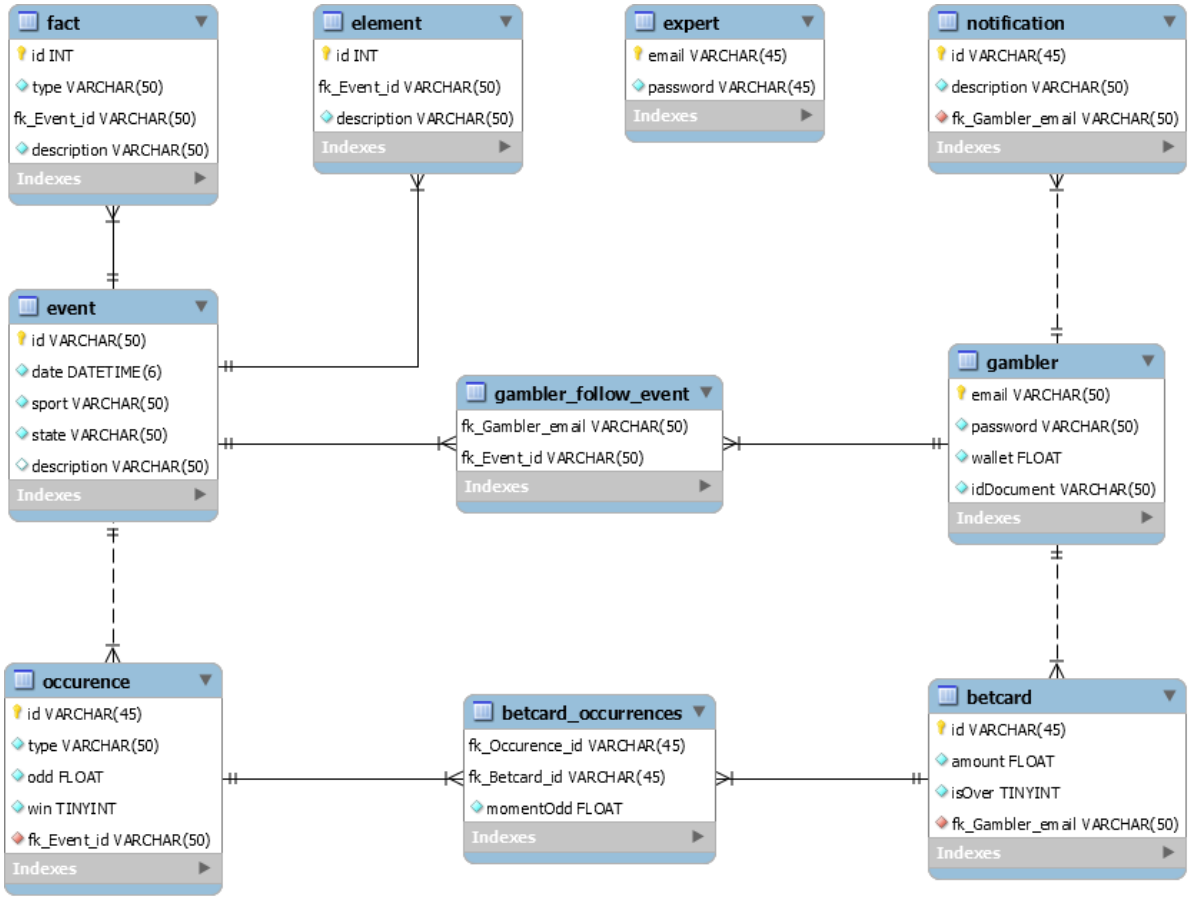

Figura 17: Modelo lógico da base de dados

Nome	Descrição
expert	Armazena as informações da conta dos especialistas. Contém um email que funciona como chave primária e um *hash* da sua palavra-passe.
event	Armazena as informações de um evento. Contém uma data, uma identificação do desporto, o seu estado e uma descrição opcional.
element	Armazena os participantes de evento. Contém uma identificação da sua designação.
fact	Armazena os factos que aconteceram num evento. Contém uma identificação do tipo (ex: 1x2 Int, 1º golo) e o resultado.
occurence	Armazena as ocorrências que podem acontecer num evento. Contém uma identificação do tipo (ex: 1x2 Int, 1º golo) e a sua *odd*.
betcard	Armazena os boletins de apostas. Contém a quantidade apostada e um estado.
gambler	Armazena as informações da conta dos apostadores. Contém um email que funciona como chave primária, um *hash* da sua palavra-passe, o montante na sua carteira e o seu documento de identificação.
notification	Armazena as notificações associadas a apostadores. Contém uma descrição.
betcard_occurences	Tabela que resulta da relação de N para M entre boletins de apostas e ocorrências. Contém a *odd* no momento da aposta.
gambler_follow_event	Tabela que resulta da relação de N para M entre eventos e apostadores. Um apostador pode seguir múltiplos eventos e um evento é seguido por múltiplos apostadores.

Tabela 13: Tabelas da base de dados

Estas tabelas estão mapeadas no diagrama de classes seguinte:

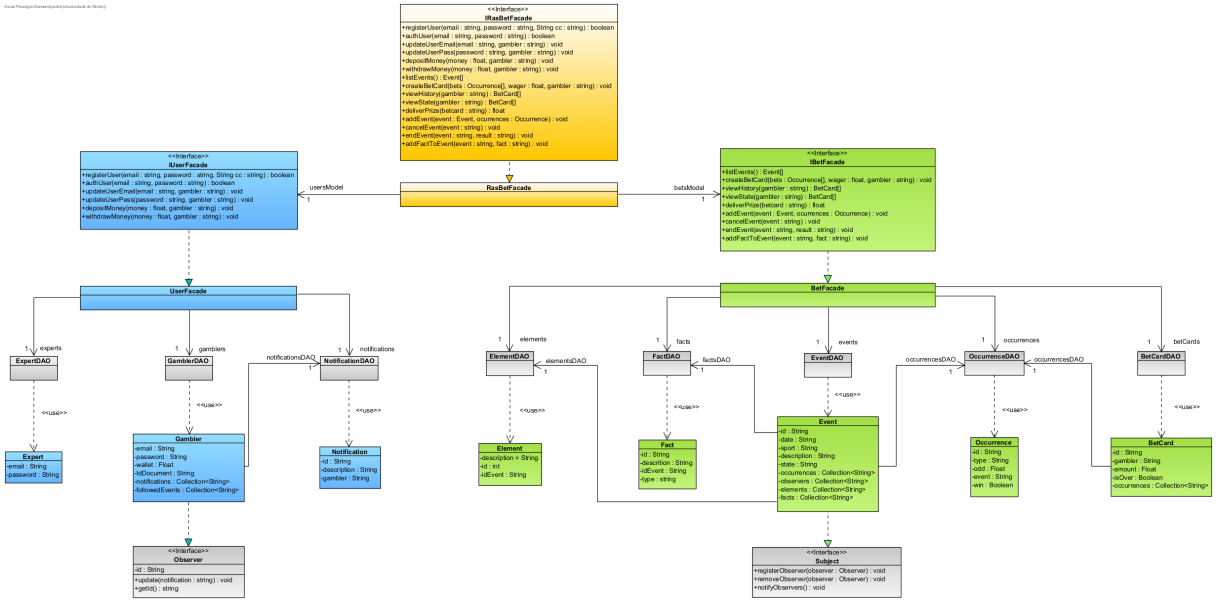

Figura 18: Diagrama de classes

Nome	**Descrição**
UserFacade	Para lidar com a inserção de utilizadores no sistema e a sua autenticação. No caso dos apostadores, irá conter métodos para editar as suas informações, levantar e depositar dinheiro da sua carteira.
BetFacade	No contexto das apostas, lidará com funcionalidades como obter a lista de eventos ativos, criar boletins de apostas, obter o histórico de apostas de um apostador. Conterá também métodos para inserir, cancelar e encerrar eventos.
RasBetFacade	Irá importar os métodos das facades interiores, funciona como controlador da REST api.

Tabela 14: Classes principais

Nome	**Descrição**
registerUser	Insere utilizador na base de dados.
authUser	Autentica utilizador.
updateUserEmail	Atualiza email do utilizador.
updateUserPass	Atualiza palavra-passe do utilizador.
depositMoney	Insere montante na carteira do utilizador.
withdrawMoney	Retira montante da carteira do utilizador.

Tabela 15: Métodos de negócio principais em UserFacade

Nome	**Descrição**
listEvents	Obtém lista de eventos ativos.
createBetCard	Cria e insere boletim de apostas na base de dados.
viewHistory	Obtém lista de apostas passadas de um apostador.
viewState	Obtém estado de um boletim de apostas.
deliverPrize	Atribui o prémio a um boletim de apostas.
addEvent	Insere novo evento na base de dados.
cancelEvent	Cancela evento. Retira todas as suas ocorrências em boletins de apostas.
endEvent	Termina evento. Atualiza o estado das ocorrências em boletins de apostas que contém o evento
addFactToEvent	Adiciona facto de um evento à base de dados.

Tabela 16: Métodos de negócio principais em BetFacade

8.2 Arquitetura e *design* de padrões

A RasBet será uma aplicação web. Tem como utilização básica um (ou mais) utilizador(es) (cliente) que interage(m) com o sistema enviando mensagens a um servidor (pedidos). O servidor por sua vez, interpreta os pedidos e envia uma mensagem de resposta ao cliente.

8.2.1 Arquitetura

Tendo em conta o método de utilização do sistema e os seus objetivos funcionais, optou-se por escolher o padrão arquitetural em Cliente-Servidor. Este padrão destaca-se por facilitar a comunicação entre o cliente e o servidor: um cliente solicita que recursos específicos sejam buscados ao servidor; o servidor identifica os pedidos feitos e responde ao cliente de forma adequada, enviando os recursos solicitados. Esta arquitetura é altamente flexível, pois um único servidor pode servir a vários clientes.

Concretamente, o Servidor estará dividido em três camadas pelo que se adotará uma arquitetura *three-tiered client-server*. As três camadas serão: a camada de interface com o utilizador, a camada de negócio e a camada de persistência.

No sistema, a camada apresentação corresponde ao componente de REST api que será utilizada pelo *frontend*. Esta camada é responsável pela capacidade de comunicação entre clientes e servidor. Quando um cliente faz um pedido por meio de uma API RESTful, esta transfere uma representação do estado do recurso para o cliente ou *endpoint*. Esta informação, ou representação, será entregue em formato JSON via HTTP. Dentro deste componente será possível encontrar métodos e o seu mapeamento correspondente que utilizam os métodos desenvolvidos na camada de negócio para aceder à informação desejada pelo cliente.

A camada de negócio foi dividida em duas partes tal como foi referido anteriormente, a parte dos utilizadores e a parte das apostas. A parte dos utilizadores funciona como o padrão comum de registo/autenticação com administradores e que poderia ser utilizado noutra aplicação. Contém também notificações. A parte das apostas contém os métodos para lidar com as diferentes funcionalidades esperadas no sistema que já foram referidas anteriormente.

Finalmente, a camada de persistência, representada pelas classes DAO no sistema. Cada tipo de objeto guardado na base de dados terá uma classe deste género. Cada classe tem implementado os métodos que fazem *querys* à base de dados e retorna os resultados obtidos.

8.2.2 Padrões de *Design*

Tendo em conta o sistema a desenvolver, é prevista a implementação de pelo menos três padrões de *design*: *facade*, *observer* e *DAO*.

Facade

O padrão de *design Facade* é usado quando um segmento da lógica precisa de uma interface simplificada para a funcionalidade geral de um subsistema complexo, isto é, envolve um subsistema complicado numa interface mais simples. Assim sendo, define uma interface de nível superior que torna o subsistema mais fácil de usar.

No sistema a implementar, este padrão será utilizado várias vezes. A interface *facade IUserFacade* liga as classes de Apostador, Especialista e Notificação e fornece uma api concisa para lidar com os utilizadores da aplicação. A interface *facade IUserBet* fornece uma api que lida com os diversos métodos relacionados com os eventos e apostas. Finalmente, a interface *facade IRasBetFacade* inclui todos os métodos das interfaces anteriores de forma a simplificar a criação da api REST.

Observer

O padrão *Observer* é um padrão usado quando existe uma dependência de um para muitos entre classes. A sua utilização define que sempre que um objeto atualiza o seu estado, todo os objetos dependentes dele são automaticamente avisados e atualizados.

Para a implementação deste padrão serão necessárias pelo menos duas classes e duas interfaces, sendo que cada classe implementará uma interface. Estas interfaces são denominadas *Subject* e *Observer*.

A interface *Subject* será implementada pela classe que representa o objeto cujo estado vai ser alterado. Já a interface *Observer* vai ser implementada por uma classe que depende (ou usa) o objeto que implementa a outra interface.

Com isto, sempre que o objeto altera o seu estado, a sua interface irá invocar um método da interface do *Observer*. Através disso a(as) classe(classes) que a implementam vão atualizar o seu estado para os novos valores.

No sistema a implementar, irá ser implementado este padrão na relação entre Apostadores e Eventos uma vez que existe uma relação de um para muitos pois um Evento pode estar a ser seguido por múltiplos Apostadores.

A classe *Gambler* será responsável por implementar a interface *Observer* e a classe *Event* será responsável por implementar a interface *Subject* de forma a que sempre que é adicionado um Facto a um Evento, a interface *Subject* chama a interface do *Observer* e envia uma notificação ao Apostador com a descrição do Facto.

Sempre que um Apostador cria um Boletim de apostas, passa a ser *Observer* do evento em que apostou. Assim, sempre que o evento for atualizado com um Facto, todos os Apostadores que o observam serão notificados com a nova informação do evento.

DAO

O padrão *Data Access Object* (*DAO*) é um padrão estrutural que permite isolar a camada

de negócio da camada de persistência (base de dados) fazendo uso de uma API abstrata. O objetivo desta API é ocultar a complexidade de realizar operações de criação, leitura, atualização e remoção (CRUD) na base de dados. Isto permite que ambas as camadas evoluam separadamente sem saber nada uma sobre a outra.

Para a implementação deste padrão, é necessário identificar quais as classes que representam objetos que serão guardados na base de dados. Estas classes não irão implementar métodos, apenas representam o objeto que também se encontra na base de dados. É necessário também uma classe onde se vão encontrar os métodos de operações CRUD.

No sistema a implementar, as classes que representam objetos que serão guardados na base de dados são: *Expert, Gambler, Notification, Element, Fact, Event, Occurence* e *BetCard*. Cada uma destas classes terá a respetiva classe DAO.

8.3 *User Experience (UX)*

8.3.1 Interface do Utilizador

A interface do utilizador será escrito em JXS, uma sintaxe de programação do React. A utilização de React torna fácil criar interfaces de usuário interativas.Além disso, o React atualiza e renderiza com eficiência apenas os componentes certos quando os dados forem alterados.

A interface será construída com recurso aos componentes da biblioteca MUI e eventualmente a outras bibliotecas.

Mockups da interface podem ser encontrados na secção 9 do Documento de Requisitos da RasBet.

8.3.2 Ergonomia

A interface seguirá as seguintes qualidades:

Simplicidade

Deve ter um *design* sóbrio e puro. O utilizador deve conseguir realizar as diferentes funcionalidades em menos de 6 cliques. É importante, portanto, apresentar as páginas da maneira mais rápida e simples possível, graças a botões bem posicionados e bem desenhados. Não utilizar mais do que 5 cores.

Legibilidade

Irá usar fontes bem legíveis. No máximo apenas duas utilizando pesos diferentes (claro, normal, negrito, itálico).

Navegação simples

Deve exibir o menu no topo de todas as páginas.

8.3.3 Internacionalização

Apenas é previsto suporte para linguagem portuguesa.

8.4 Confidencialidade e segurança

8.4.1 Confidencialidade

Tal como referido anteriormente, não está previsto um alto grau de confidencialidade no sistema. Serão feitos pedidos HTTP simples sem autenticação. Os utilizadores apenas terão acesso à sua informação e as suas palavras-passe serão passadas à base de dados na forma de um *hash*.

8.4.2 Segurança

A utilização da aplicação não envolve qualquer risco para a saúde.

8.5 *Under-the-hood*

8.5.1 Persistência

A Rasbet utilizará um base de dados local para armazenar dados relacionais.

Todos os acesso à base de dados irão passar pelo módulo *Spring Data JBDC*.

8.5.2 Excepções e Tratamento de erros

Erros no tratamento de dados inconsistentes, bem como falhas na validação, são mapeados para erros HTTP. Esses erros são depois tratados pelo *frontend*. Erros técnicos (hardware, base de dados, etc.) não são tratados e podem levar à falha da aplicação ou perda de dados.

8.5.3 Verificações de plausibilidade e validade

Tipos e intervalos válidos de dados são verificados no *frontend*. Exemplos são dados como emails e resultados de eventos válidos (maiores ou igual a 0).

Na camada de negócio serão feitas as seguintes verificações:

- O apostador apenas pode apostar um montante igual ou inferior ao valor que tem na carteira;
- O apostador apenas pode levantar um montante entre 5 euros e o valor que tem na carteira;
- Para fazer *login* e/ou alterar as informações da conta, é necessário validar as credenciais;
- Verificar que um boletim não tem ocorrências opostas. Por exemplo: aposta em vitória e empate de um mesmo evento.

9 Decisões Arquiteturais

As entidades principais do sistema são o Apostador, o Especialista, o Evento e o Boletim de apostas.

A Apostador interage com o *frontend* que realiza pedidos à API da lógica de negócio.

O Evento consiste numa partida de uma dada modalidade. Um dado evento pode compreender vários tipos de apostas diferentes. Chamamos de Ocorrência a cada uma das diferentes possibilidades dentro de um tipo de aposta. Cada Ocorrência tem uma *odd* que vai evoluindo com o número de apostas feitas em cada possibilidade. Um evento, dependendo da modalidade, pode ter várias equipas ou participantes pelo que cada evento conta com uma lista de Elementos. Cada evento tem três estados: ativo, pendente e terminado. Finalmente, à medida que o evento progride, são adicionados factos, isto é, ocorrências que de facto aconteceram no evento. Esta lista de factos tem como objetivo verificar o resultado das Ocorrências e servir de informação para notificar os utilizadores.

Um Boletim de apostas consiste numa lista de uma ou mais ocorrências de diversos eventos. Cada boletim está ligado a um apostador. Quando todas as ocorrências terminam, o estado do boletim passa para terminado.

O Especialista interage com a base de dados ao inserir novos eventos, cancelar eventos, terminar eventos e adicionar factos a eventos. As suas ações tem diversos efeitos nas diversas funcionalidades. Por exemplo, quando adiciona um novo evento, esse evento passa a aparecer na lista de eventos ativos; quando um evento é cancelado, todos os boletins de apostas nos quais constava ocorrências desse evento são removidos; quando termina um evento, são verificados os boletins que contém esse evento e verificado o seu resultado para depois atribuir um prémio.

9.1 Apostas múltiplas

Decidiu-se permitir apenas boletins de apostas múltiplas, isto é, sempre que num boletim aparece mais do que uma aposta, as *odds* multiplicam-se e o utilizador só ganha caso acerte todas as ocorrências. No caso de pretender fazer apostas simples, é necessário criar um boletim separado para cada uma delas.

9.2 *Odds* dinâmicas

De forma a evitar implementar um valor estático para as *odds* de cada ocorrência, estas serão dinâmicas. Isto é, à medida que os apostadores apostam em determinada ocorrência de um evento, o seu valor irá oscilar. Por exemplo, caso muitos apostadores estiverem a apostar na vitória de uma equipa, então a *odd* que os próximos apostadores irão encontrar será menor que a *odd* inicial. Da mesma forma, caso se verifique a escassez de apostas numa determinada ocorrência, a sua *odd* irá aumentar.

10 Requisitos de Qualidade

10.1 Árvore de qualidade

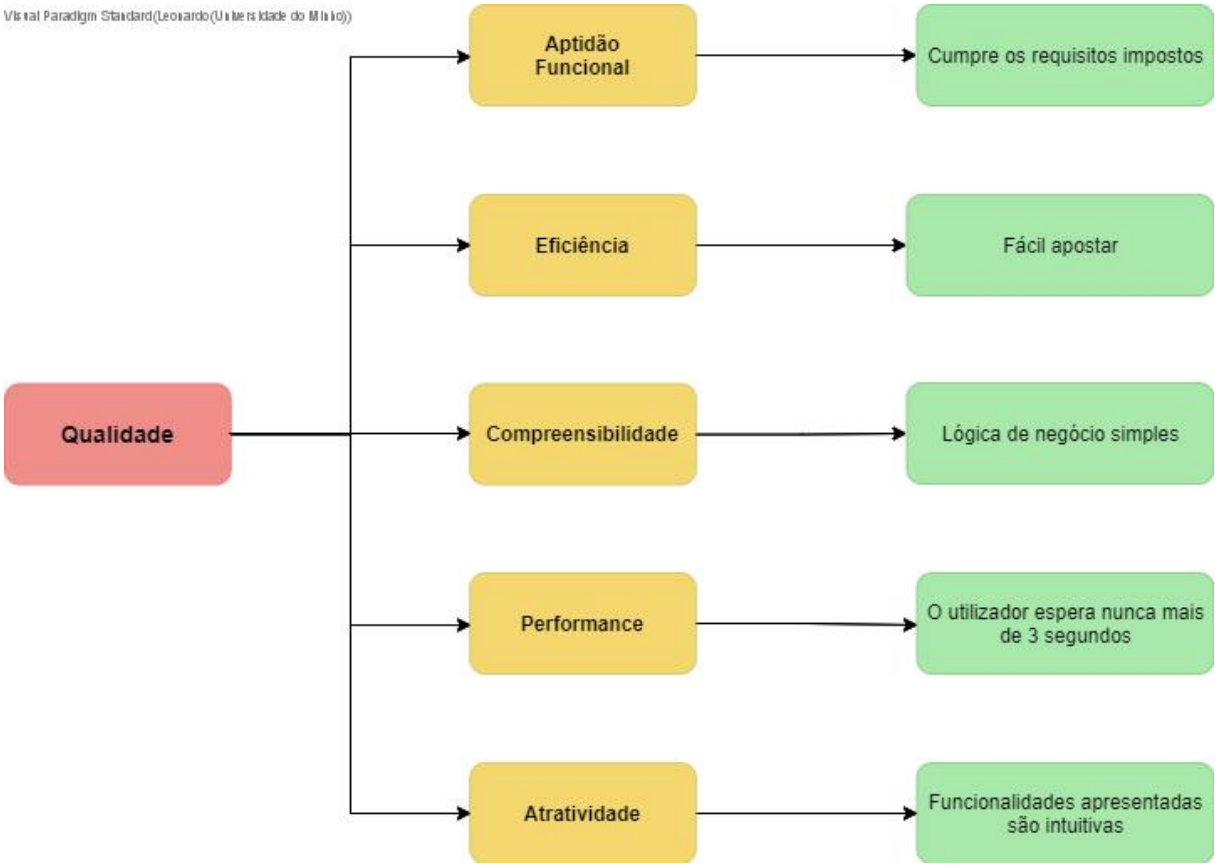

Figura 19: Árvore de qualidade

10.2 Cenários de qualidade

Aptidão Funcional

Cumpre três condições:

- **Completude funcional** - O sistema é capaz de fornecer todas as funções especificadas pelo utilizador.
- **Correção funcional** - O sistema tem comportamento e gera os resultados desejados pelo utilizador.
- **Adequação funcional** - O sistema é capaz de cumprir os requisitos necessários para os diferentes objetivos de uso que foram especificados.

Um apostador que tenha sido entrevistado (secção 3.2, Documento de Requisitos Rasbet) antes da concretização do sistema deve sentir que todas as suas necessidades foram cumpridas com o uso da aplicação. Concretamente, por exemplo, João Chaves (secção 3.2.1, Documento de Requisitos Rasbet) irá encontrar uma interface amigável; modalidades como futebol, basquetebol, MMA, etc; diversos tipos de apostas como resultado ao intervalo, resultado em tempo regulamentar, primeiro golo, etc, para um evento futebolístico; apostas e movimentos monetários sem problemas.

Eficiência

Apostar deve ser uma tarefa simples. Após a criação de três boletins de apostas, a criação de mais um deve ser uma tarefa natural a qualquer apostador.

Compreensibilidade

Seguir os requisitos funcionais e a arquitetura proposta deve resultar numa implementação sóbria apenas com o código necessário.

Por exemplo: a classe Evento é usada para qualquer modalidade ao invés de ter uma classe para cada modalidade.

Performance

Com a utilização do padrão DAO, quantidades e tipos de recursos usados são mínimos.

Adicionalmente, com a utilização de uma *cache* para a lista de eventos ativos, tempos de resposta e processamento não devem ultrapassar os 3 segundos.

Atratividade

Ao utilizar um *design* minimalista com apenas a informação necessária, um utilizador deve ser capaz de, em menos de 5 minutos, compreender todas as funcionalidades.

11 Riscos e Dívidas Técnicas

11.1 Um concorrente adiciona uma nova funcionalidade

Contexto

Existe a possibilidade de uma aplicação concorrente lançar uma nova funcionalidade que ainda não tenha sido implementada pela RasBet. Este risco traz como consequência uma possível migração dos apostadores para a aplicação concorrente. Funcionalidades como competições entre amigos, ou outras bastante inovadoras, podem chamar a atenção dos utilizadores deste tipo de aplicações.

Solução

A solução para este risco reside em implementar essa mesma funcionalidade de uma forma mais atrativa para o utilizador no futuro, aprendendo com os erros que a aplicação pioneira tenha cometido por ter sido a primeira a tentar.

11.2 Recuperação de desastres

Contexto

Existe a possibilidade de tanto o *backend* como a base de dados falharem devido a algum fator externo, causando a quebra do sistema RasBet.

Solução

A solução para este risco reside no investimento em máquinas alternativas que consigam a replicação dos componentes do sistema, de modo que consigam manter o sistema caso ocorra uma falha.

11.3 Riscos externos imprevisíveis

Contexto

A imprevisibilidade é outro fator de risco que pode ser encontrado durante todo o ciclo de vida da RasBet. Ignorar os riscos externos seria uma abordagem ingénua pois é impossível prever fatores como: mudanças nos interesses do utilizador, implementação de novas regulamentações governamentais ou o aparecimento de um concorrente com mais recursos.

Solução

Contratação de uma equipa de analistas de negócios revela-se fundamental. Esta equipa terá como tarefas: pesquisar o mercado, país de operação e tendências mundiais para analisar a situação atual. O seu objetivo é diferenciar os padrões de mercado e verificar se estes são favoráveis ao produto e ao modelo de negócio. Tecnologias mais recentes como *Machine Learning* e *Big Data Analytics*, são ferramentas que ajudam a equipa de analistas a tomar melhores decisões.

12 Glossário

Nome	**Descrição**
CNPD	Comissão Nacional de Proteção de Dados (CNPD) é uma entidade administrativa independente, com personalidade jurídica de direito público e com poderes de autoridade, dotada de autonomia administrativa e financeira, que funciona junto da Assembleia da República.
MySQL	*MySQL* é um sistema de gestão de dados, que utiliza a linguagem SQL (*Structured Query Language*) como interface.
React	*React* é uma biblioteca *JavaScript open source* com foco em criar interfaces de usuário (*frontend*) em páginas web.
GitHub	*GitHub* é uma plataforma de hospedagem de código-fonte e arquivos com controlo de versão.
Spring Boot	*Spring Boot* facilita a criação de aplicações autónomas baseados em *Spring* que são executáveis facilmente.
Spring Data JBDC	*Spring Data* JDBC faz parte da família *Spring Data*, facilita a implementação de repositórios baseados em JDBC. Este módulo trata do suporte aprimorado para camadas de acesso a dados baseadas em JDBC.
JSON	JSON (*JavaScript Object Notation*) é um formato de troca de dados leve. É fácil de ler e escrever. É um formato fácil de gerar e fazer *parsing*.
JDK 17	JDK 17 é um ambiente de desenvolvimento para construir aplicações baseadas na linguagem de programação Java.
Maven	*Maven* é uma ferramenta de gestão e compreensão de projetos de software. Com base no conceito de modelo de objeto de projeto (POM), o *Maven* pode gerir a construção, o relatório e a documentação de um projeto a partir de uma informação central.
JXS	JXS é uma extensão de sintaxe para *JavaScript*.
MUI	MUI fornece uma biblioteca robusta, personalizável e acessível de componentes básicos e avançados, que permitem construir um sistema de design e desenvolvimento de aplicações *React* com maior rapidez. .

Tabela 17: Classes principais