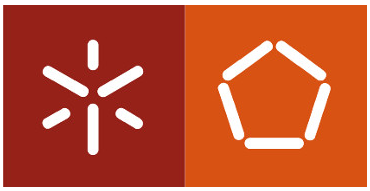


UMinho

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

RASBET

Grupo PL 3 - 2 | Entrega 2

Gonçalo Braz (a93178)
Tiago Silva (a93277)
Simão Cunha (a93262)
Gonçalo Pereira (a93168)

Braga, 7 de dezembro de 2022

Conteúdo

Lista de Figuras

Lista de Tabelas

Contribuições de cada membro da equipa

De forma a identificar-mos o contributo de cada membro da equipa, elaboramos uma tabela para esse efeito. A medição do contributo toma três valores: `-` (alunos que contribuíram significativamente abaixo da média), `0` (alunos cujo contributo foi próximo da média) e `+` (alunos que contribuíram significativamente acima da média.

Membros da equipa	Contribuição em relação à média
Gonçalo Braz	0
Tiago Silva	0
Simão Cunha	0
Gonçalo Pereira	0

1. Introdução e objetivos

Nesta segunda entrega do trabalho prático da UC de Requisitos e Arquiteturas de Software, procedemos à implementação dos requisitos funcionais que ficaram por implementar na fase anterior, à correção de alguns aspetos que foram detetados pela equipa docente aquando da primeira apresentação e ao tratamento dos requisitos não funcionais, onde pensamos sobre a arquitetura do sistema, com a inclusão de alguns *design patterns*.

O primeiro passo foi a correção do documento de requisitos. Aqui efetuamos o seguinte:

- Reparamos que interpretamos erradamente os requisitos não funcionais de escalabilidade, tendo procedido à sua retificação - encontram-se assinalados de acordo com as normas estabelecidas para a escrita de requisitos. Assim, surgiram os requisitos não funcionais #14 e #15;
- Decidimos incluir alguns requisitos não funcionais de usabilidade que antes constavam na secção de requisitos não funcionais de escalabilidade (requisitos não funcionais #7, #8 e #9);
- Alteramos o número do tipo de requisito não funcional de segurança (17 $\rightarrow$ 15) em todos os requisitos deste tipo (requisito não funcional #15)
- No requisito funcional #7, esclarecemos o conceito de "consulta estatística": o utilizador pode ver o montante total depositado, o montante levantado, o montante apostado, ganho com apostas, ganho com promoções, percentagem de apostas simples ganhas e percentagem de apostas múltiplas ganhas.
- Adicionamos um requisito funcional #10, referente à capacidade do sistema poder incorporar vários idiomas para a visualização das suas páginas;
- No requisito funcional #15, acrescentamos o novo estado da aposta ("para aprovação");
- No requisito funcional #20, retiramos o acesso de um jogo através da sua data;
- No requisito funcional #21, adicionamos o facto de poder ordenar jogos por data;
- No requisito funcional #28, alteramos o número de estados da aposta, passando de três para quatro, assim como a nomenclatura do novo estado em questão;
- No requisito funcional #32, alteramos o local onde o utilizador pode ver oportunidades de aposta: antes era na página inicial mas agora é redirecionado para uma outra página;
- No requisito funcional #42, adicionamos o facto de o utilizador ter acesso ao resultado final de uma jogo na página das apostas efetuadas;
- Retiramos o antigo requisito funcional #44, pois era igual ao requisito funcional #20;

1 Requisitos funcionais

1. Um utilizador do sistema tem obrigatoriamente nome de utilizador, uma password, uma data de nascimento, um e-mail e um saldo monetário na sua carteira da aplicação;
2. Dois utilizadores são considerados diferentes se tiverem o e-mail diferente;
3. O Apostador tem de se registar na aplicação para a poder utilizar;
4. O Utilizador edita o seu perfil;
5. O utilizador consulta o seu histórico de apostas;
6. O utilizador, ao efetuar uma aposta, escolhe a quantia a apostar oriunda do seu saldo, o tipo de aposta a fazer (simples ou múltipla) e o resultado de cada jogo da sua aposta.;
7. O utilizador realiza uma consulta estatística dos seus ganhos na aplicação: o montante total depositado, o montante levantado, o montante apostado, ganho com apostas, ganho com promoções, percentagem de apostas simples ganhas e percentagem de apostas múltiplas ganhas;
8. Utilizador consulta histórico de transações;
9. Utilizador consulta os seus desportos favoritos;
10. Utilizador cancela a aposta se esta ainda se encontrar aberta e recebe o montante apostado na íntegra de volta no seu saldo.;
11. Utilizador faz aposta com montante mínimo de 0.10 numa moeda suportada na aplicação;
12. Utilizador faz aposta com montante máximo de 500 numa moeda suportada na aplicação;
13. Utilizador aposta montante não superior ao disponível na sua carteira.
14. O utilizador com credenciais de especialista adiciona e altera as odds associadas a um jogo já existente no sistema;
15. Utilizador com credenciais de administrador pode alterar estado de uma aposta (aberta, suspensa, para aprovação, fechada);
16. Utilizador com credenciais de administrador pode criar promoções;
17. O utilizador com credenciais de administrador pode criar eventos desportivos no sistema;
18. O sistema deve garantir que nenhum utilizador tem acesso a outros perfis de utilizador;
19. O sistema deve permitir o acesso a um jogo através das seguintes formas: pelo desporto, pelo país e pela competição;
20. O sistema deve permitir ao utilizador a visualização de uma lista de jogos que contém uma referência ao desporto, ao país, à competição, à data, às equipas envolvidas e às odds, podendo esta ser ordenada por datas;
21. A aplicação deve permitir ao utilizador realizar uma aposta simples num evento desportivo disponível na aplicação;
22. A aplicação deve permitir ao utilizador realizar várias apostas simples e múltiplas em eventos desportivos disponíveis na aplicação;
23. A aplicação permite o registo de novos utilizadores;
24. A aplicação permite realizar apostas, recorrendo ao saldo da carteira;
25. A aplicação permite o carregamento/levantamento de saldo da sua carteira através de MBWay e cartão de débito;
26. Um jogo tem obrigatoriamente um país, uma competição, uma equipa visitada, uma equipa visitante e uma data (no formato Dia/Mês/Ano Hora:Minuto);
27. Uma aposta tem obrigatoriamente um de 4 estados, Aberta, Fechada, Para Aprovação, Suspensa;
28. O sistema tem de suportar apostas em futebol;
29. O sistema tem de notificar os apostadores dos resultados das suas apostas;
30. O sistema deve permitir o utilizador adicionar e remover desportos como favoritos;
31. O sistema deve redirecionar para outra página oportunidades de apostas de acordo com os favoritos do utilizador;
32. O sistema deve notificar os utilizadores por email e na aplicação de novas promoções que estejam disponíveis na aplicação;
33. O sistema deve permitir ao utilizador filtrar as partidas mais apostadas de qualquer desporto disponível na aplicação;

34. O sistema permite um depósito com montante mínimo de 5 numa moeda suportada na aplicação;
35. O sistema permite um depósito com montante máximo de 500 numa moeda suportada na aplicação;
36. O sistema permite uma aposta múltipla com no mínimo 2 jogos;
37. O sistema permite uma aposta múltipla com no máximo 10 jogos;
38. O sistema deve impedir que haja duas apostas para o mesmo jogo num boletim de aposta múltipla;
39. O sistema permite ao utilizador apostar apenas em odds de jogos não inferiores a 1.20 na modalidade simples;
40. O sistema permite ao utilizador apostar apenas em odds de jogos não inferiores a 1.10 na modalidade múltipla;
41. O sistema deve mostrar o resultado final de um jogo no fim deste na página das suas apostas efetuadas;
42. O sistema deve mostrar a distribuição das apostas por percentagem de um jogo;
43. O sistema tem de fechar as apostas de um jogo após o seu início;
44. O sistema tem de notificar os apostadores dos resultados das suas apostas e creditá-los com o devido prémio (caso seja aplicável)

2 Requisitos não funcionais

Entre os vários requisitos não funcionais tratados no documento de requisitos, relevamos os seguintes:

- O sistema tem de ser um serviço web;
- Após o clique de criação de aposta, a aplicação não deve demorar mais de 2 segundos a registar a aposta;
- Após o carregamento da conta o saldo tem que estar disponível para utilização em menos de 30 segundos;
- O sistema deverá preservar a integridade dos dados dos utilizadores;
- O sistema só poderá ser usado por utilizadores com uma idade igual ou superior a 18 anos;

Estes requisitos são fundamentais para a implementação do sistema. É essencial que o sistema seja uma aplicação Web de forma a permitir que vários utilizadores possam acedê-la em diferentes *browsers*. Além disso, é necessário que o tempo de criação da aposta seja muito baixo para impedir que o utilizador tenha tempo de ponderar a realização da mesma, levando a que a equipa responsável não receba dinheiro - o mesmo se aplica no depósito do saldo. Também é importante que o sistema impeça que utilizadores não tenham acesso a dados de outros, o que levaria a uma descredibilidade da aplicação. Por último, a maioridade imposta em novos utilizadores é importante na medida em que obriga a aplicação a cumprir restrições legais em Portugal.

3 Cliente, Consumidores e StakeHolders

Ao longo do desenvolvimento deste projeto verificamos a existência de diversas partes envolvidas, nomeadamente: clientes, consumidores e *stakeholders*.

- **Cliente:** Os nossos clientes serão empresas que actuem no ramo de apostas associadas a eventos desportivos, com a necessidade de apresentar e vender aos seus clientes (casas de apostas) uma solução *user-friendly*, 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.

- **Consumidores:** Como referido anteriormente, o produto informático será apresentado e vendido pelas empresas assim destinadas a todas as casas de apostas interessadas. Assim, neste contexto, o consumidor são as casas de apostas, que são empresas registadas e licenciadas para aceitar apostas dos clientes na previsão de um certo acontecimento e com o potencial lucro. Ou seja, caso estas pretendam usufruir de um sistema *online*, que permita efetuar apostas apenas ao nível do mercado desportivo, com o benefício de ser instantâneo e, consequentemente atingir e "trazer"para este mercado um maior número de utilizadores/apostadores, necessitam de recorrer à empresa destinada à criação e venda do produto informático que permita efetuar tais apostas desportivas.

- **Outros *Stakeholders*:** Uma das principais partes interessadas no produto informático serão os apostadores, definindo-os como todas as pessoas cuja idade é superior a 18 anos que efetuam apostas ao nível do mercado desportivo, e que pretendem "mudar-se"para o sistema *online*, para que possam efetuar as suas apostas desportivas de forma rápida e intuitiva, ao alcance de um *"click"*, sem existir a necessidade de se deslocarem a uma casa de apostas física para esse efeito. Outros, serão o administrador e o apostador, que terão cargos essenciais para o 'bom' funcionamento do produto.

2. Restrições

Nesta secção, iremos falar acerca das restrições que são impostas obrigatoriamente ao nosso sistema, desde as restrições de ambiente do sistema, restrições temporais e restrições orçamentais.

1 Restrições da usabilidade do sistema

- **Descrição:** O apostador tem acesso a uma listagem de desportos, e após a seleção dessa modalidade, a uma listagem de jogos.

  Justificação: De forma a poder apostar na equipa favorita, caso seja um desporto coletivo, ou em um único participante, no caso de desporto individual, o apostador tem que ter acesso a toda a lista com os jogadores intervenientes.

- **Descrição:** A aplicação tem de ser um *web site*.

  Justificação: A aplicação deverá poder ser usada em qualquer lugar, a partir do seu computador ou *smartphone*.

- **Descrição:** O sistema terá de incluir a API de eleição da equipa docente, cuja informação pode ser obtida do seguinte link: `http://ucras.di.uminho.pt/swagger/`

  Justificação: A aplicação deverá receber informação do desporto futebol através desta API.

2 Restrições do ambiente de implementação do sistema

- A Empresa irá alocar os recursos necessários para manter a aplicação;
- As sessões de teste da aplicação serão realizadas nos *browsers* Google Chrome, Microsoft Edge e Safari;
- A Empresa quer monitorizar o desenvolvimento da aplicação via GitHub.

3 Restrições Prazo/Agendamento

- **Descrição:** O documento de requisitos terá se ser entregue numa fase inicial até dia 04 de novembro de 2022.

  Justificação: De forma a poder ser avaliado o estado do projeto numa fase inicial, é necessário que seja feita uma entrega que contenha a primeira fase deste projeto, que abarca a contextualização e a definição dos requisitos da solução.

- **Descrição:** O presente relarório terá de ser entregue até dia 5 de dezembro de 2022 até às 11.59h.

  Justificação: Permitirá à equipa docente avaliar o que a equipa de desenvolvedores tem a mostrar nesta segunda entrega, desde a nova versão do documento de requisitos até ao sistema pronto.

4 Restrições Orçamento

- **Descrição:** O orçamento total para o desenvolvimento do projeto é de 20 000€ (vinte mil euros), durante um período de 4 meses.

  Justificação: A equipa responsável pelo desenvolvimento do projeto é constituída por quatro engenheiros de *software*. Para além de ter em conta os salários dos elementos, é preciso também a compra de um domínio, bem como a de servidores para alojar todos os dados da aplicação.

3. Contextualização e foco

1 Contexto do negócio

Nesta secção iremos abordar o contexto de negócio do nosso sistema, isto é, delimitaremos o domínio do nosso sistema a um conjunto limitado de funcionalidades e dados, passíveis de serem expandidos no futuro. Para tal, elaboramos um modelo de domínio para ilustrar o contexto de negócio - o mesmo já se encontrava presente no documento de requisitos:

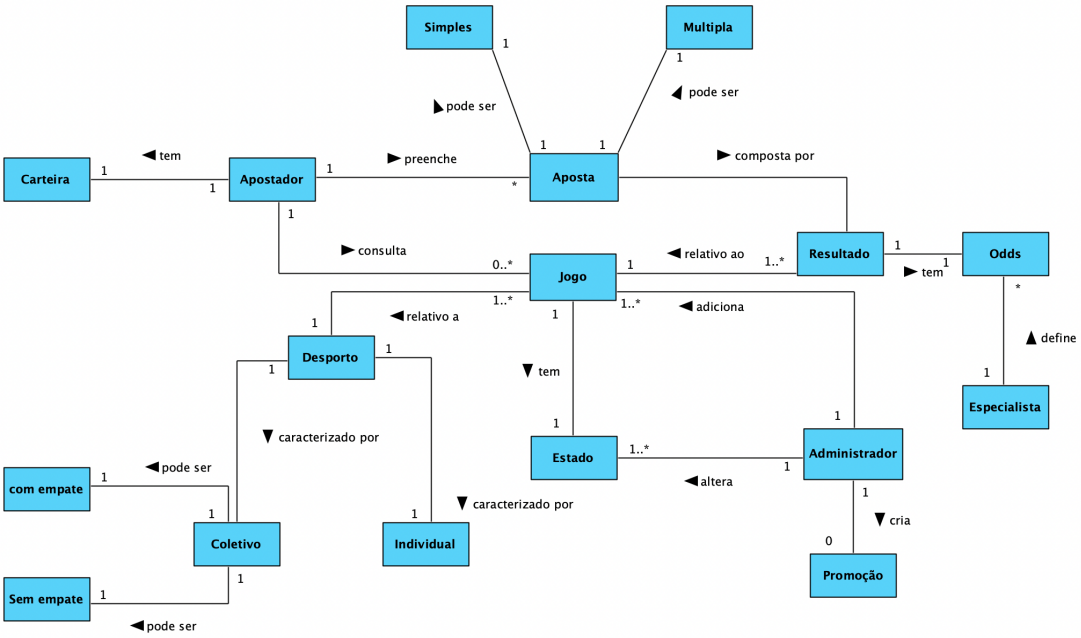


Figura 3.1: Modelo de domínio

Entidade	Descrição
Apostador	Possui uma carteira com montante para apostar Responsável por efetuar apostas Consulta jogos para apostar
Aposta	Pode tomar dois tipos: Simples e Múltipla Tem em conta o resultado de um jogo
Resultado	Possui odds definidas pelo especialista
Jogo	Incluído num desporto Possui um estado que é alterado pelo Administrador
Desporto	Pode tomar dois tipos: Coletivo (c/ ou s/ empate) e individual
Administrador	Responsável por criar promoções

Tabela 3.1: Tabela com as entidades do modelo de domínio

2 Contexto técnico

Nesta fase, o contexto técnico no nosso sistema é relativamente simples, pois existe apenas uma ligação a um componente externo: a API proposta pela equipa docente (obtida em `http://ucras.di.uminho.pt/v1/games/`). Através deste sistema vizinho, conseguimos povoar a nossa base de dados local de partidas para apostar em futebol através de um pedido HTTP GET, cuja resposta vem num formato JSON. Na secção 7, iremos mostrar um diagrama que permite visualizar em que medida este componente externo influencia a nossa aplicação.

4. Estratégia da solução

Como já ficou definido anteriormente, a aplicação RASBET tem como propósito a sua comercialização, com principal alvo empresas de casas de apostas online, de forma que o site deve ser estruturado de forma a ser modificável, como explicitado pelos requisitos anteriormente discutidos, e sempre ter em conta o utilizador final, os apostadores.

Assim sendo, existiu uma necessidade de manter um padrão de qualidade de forma a reconhecer e cumprir as várias propriedades que o software deve apresentar, em geral evidenciados através da enumeração dos requisitos não funcionais.

Discutiremos então quais foram tidos em conta e o modo como foram verificados.

Sendo que o sistema surge a partir de um modelo de domínio com uma delimitação das componentes principais, sendo estas o utilizador, apostas e jogos, seguimos com a estratégia de programação orientada a objetos. Para este fim, e tendo em vista que a aplicação teria um foco na sua usabilidade web, seguimos com a utilização da framework Django, baseada em python, que permite uma bem preparada consolidação entre a modelagem de um programa, a sua divisão entre diferentes views respetivas a um package, e o respetivo tratamento em frontend. Além disso, a própria framework apresenta à partida uma robusta API para ligação com uma base de dados (sqlite3), orientada a objetos e mesmo uma própria estrutura para exploração e tratamento da mesma para admins do sistema. Este último modelo foi especialmente útil, pois sendo extremamente expansível e versátil, permitiu-nos adaptar os admins da base de dados para os admins do RASBET.

A bem definida divisão entre front e backend permitida, naturalmente sugere uma arquitetura MVC para o sistema como uma simples solução para a interatividade entre o cliente e o servidor, para aceder aos dados do sistema.

Os dados do sistema, são essencialmente internos, como é o caso dos apostadores e das apostas, mas para a germinação dos jogos na base de dados é utilizada a API proposta pelos docentes. De forma a permitir uma possível expansão do sistema para receber novas API, com mais informação, ou jogos de outros desportos, o tratamento de dados destas é contido de forma a ser normalizado num único modelo e apenas posteriormente passado ao sistema para os guardar/atualizar.

Categoria	Objetivos	Estratégias da solução
Usabilidade	- O sistema tem de ser um serviço web - A aplicação deve permitir realizar uma aposta com, no máximo, 5 cliques - O sistema deve ter uma versão em Português de Portugal e outra em Inglês do Reino Unido, de forma a demonstrar a expansibilidade para outras linguagens - O sistema deve permitir ao utilizador depósitos em Euros e Libras, do mesmo modo permitindo a sua portabilidade para outras sociedades - O sistema deve permitir adicionar novos desportos, novos países e novas competições para apostar - O sistema deve permitir adicionar novos tipos de aposta	- Utilização da framework Django, própria para a criação de programas com interações entre clientes e servidor - Páginas html criadas direcionadas à comodidade do utilizador - Novas linguagens e moedas apenas abrangem uma breve modificação das views, e a criação dos respetivos htmls - A expansão de novos tipos de desportos e apostas é conseguida através de uma implementação direcionada aos objetos, utilizando hierarquia e abstração em ambos os casos - Este processo aplica-se ainda para outras entidades, como tipo de participantes, equipas ou individuais, competições, países,etc.
Desempenho	- Após o clique de criação de aposta, a aplicação não deve demorar mais de 2 segundos a registar a aposta - Após o carregamento da conta o saldo tem que estar disponível para utilização em menos de 30 segundos - Rápido acesso à página web e ao resto da sua navegação	- Aplicação de diversas estratégias, tais como a utilização do servidor como cache para busca dos dados das APIs periodicamente, de modo que o cliente apenas é limitado pela resposta do servidor à base de dados e não da resposta de qualquer API - Estilo Observer para a notificação do término de apostas de um dado jogo, algo que como apenas só pode ser realizado pelo admin, assim que este fecha um jogo ou um conjunto de jogos, os interessados do sistema (apostas), são notificadas para realizar a sua alteração de estado e, possivelmente, notificar os respetivos apostadores dos seus resultados finais
Aparência	- Palete de cores fixa - Nav bar bem definida, de modo a fomentar interatividade	- Mecanismos e ferramentas front-end, tal como o uso de html e css
Segurança	- O sistema deverá preservar a integridade dos dados dos utilizadores - O sistema deve garantir uma distinção de privilégios de modificação de dados, baseada em cargos de utilizador (apostador, administrador,especialista)	- Encapsulamento de objetivos e definição de funcionalidades baseadas no tipo de utilizador - Utilização da ferramenta admin do Django para administração de dados - Páginas e redirecionamentos de front-end definidos para os diferentes tipos de utilizador
Legalidade	- O sistema só poderá ser usado por utilizadores com uma idade igual ou superior a 18 anos	- Condições restritivas durante o cadastro de um novo apostador com intuito de impedir a criação de conta a menores de idade

Tabela 4.1: Objetivos de qualidade e suas soluções

5. Building block view

De modo a decompor o sistema nos seus diferentes blocos, que representam as suas principais entidades e respetivas funcionalidades, geramos diferentes diagramas, de modo a expor esta divisão em diferentes níveis: entre os principais subsistemas, através do diagrama de componentes (nível 1) e, posteriormente, uma melhor especificação destes e das suas interações com o diagrama de classes (nível 2).

1 Diagrama de componentes - Nível 1

Começamos pelo desenho da divisão do sistema nas suas principais componentes, cada uma responsável pelo tratamento de diferentes funcionalidades, sendo estas o **Accounts**, responsável pelo tratamento de todos os dados referentes aos utilizadores (especialistas incluídos), **Game**, responsável pelo tratamento dos jogos, suas competições, países e desportos e por último **Gamble**, responsável por todos os mecanismos de realização de apostas do sistema, assim como ligação entre os jogos e utilizadores que nestes apostam. Por fim tem o **Admin**, que trata da manipulação especial de dados, que apenas pode ser feita por administradores, como por exemplo, fechar jogos, criar novas promoções ou mandar notificações. Este último sistema foi criado utilizando as ferramentas inerentes do Django, como anteriormente referido.

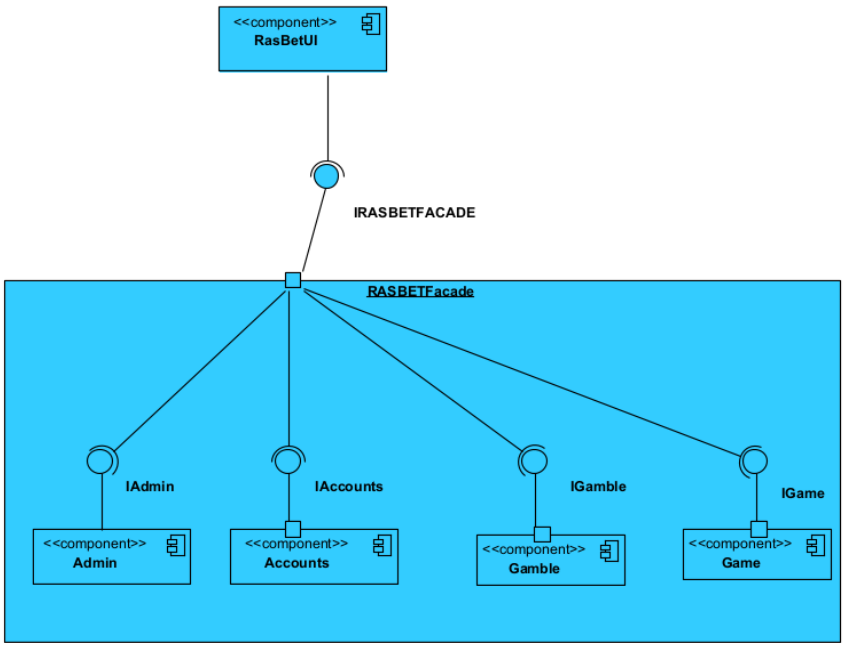


Figura 5.1: Diagrama de componentes

Analisando os use cases definidos na fase anterior, podemos fixar as responsabilidades destes sistemas da seguinte forma:

Use case	Sistema
login	Accounts
registo	Accounts
fazer aposta	Gamble
consultar histórico de apostas	Accounts
consultar jogos	Game
alterar informações de perfil	Accounts
depositar dinheiro	Accounts
levantar dinheiro	Accounts
consultar histórico de transações	Accounts
inserir e alterar odds	Game
alterar estado da aposta	Admin
gerir notificações	Admin
criar promoções	Admin
adicionar jogos	Admin

Tabela 5.1: Use cases e sistemas responsáveis

2 Diagrama de classes - Nível 2

O diagrama de classes procura fornecer uma visão mais detalhada de todas as componentes do sistema, a maneira como elas coexistem, as suas interdependências e uma simples noção das suas principais funcionalidades. Tendo em vista a estruturação particular do Django, o diagrama que se segue é uma abstração do diagrama real, visto que esta framework trabalha a partir de uma divisão de aplicações/modelos, para os quais cada uma tem a sua view, tornando em formato de diagrama mais confuso e com muitos mais elementos. Mesmo com este adendo, conseguimos ver as dependências entre as diferentes classes e modelos.

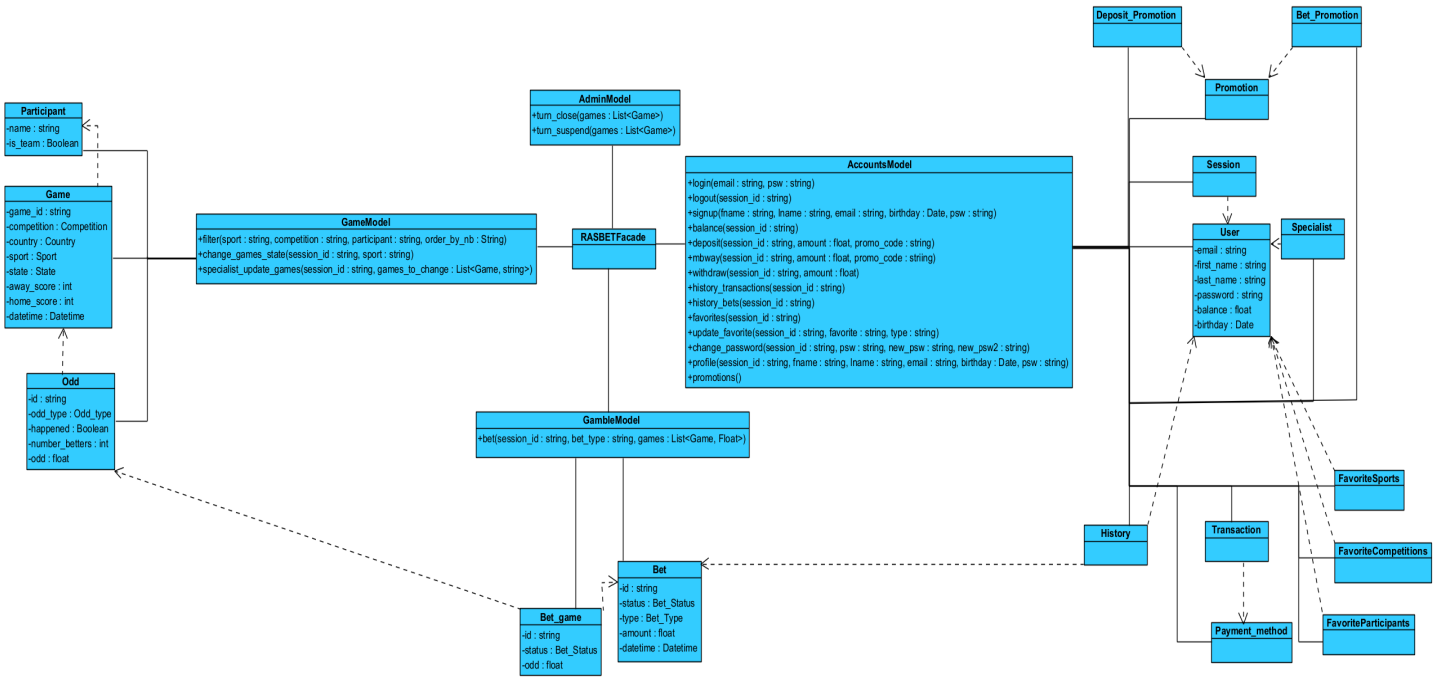


Figura 5.2: Diagrama de classes

Como mencionado anteriormente, os principais subsistemas são **Admin,Game,Accounts** e **Gamble** que, passadas para o diagrama de classes, cada uma é representada por uma classe

Model, que serve como Facade de cada uma destas, implementando os seus respetivos métodos.

O modelo **Admin** tem como métodos principais a mudança do estado dos jogos de aberto para fechado ou suspenso, isto, consequentemente, terá uma cascada de outros efeitos, com o fecho das respetivas apostas, atualização dos balanços dos apostadores vencedores e respetiva notificação.
Além disso, e como mencionado anteriormente, este sistema tem ainda outras funcionalidades, mais orientadas aos dados, como a criação de novos admins, jogos, promoções, entre outras, que são disponibilizadas pela embutida página de admin da framework.

O modelo **Accounts**, será o mais complexo e que criará mais dependências no sistema, pelo simples facto de possuír todos os dados sobre os utilizadores, os sujeitos que o irão utilizar e que, sem os quais, não seria possível realizar apostas. Os utilizadores dividem-se entre apostadores normais e os especialistas, que são responsáveis pela aprovação dos jogos, para poderem ser apostados, e pela modificação das suas odds.
A classe Transaction representa um registo de uma transação da conta do utilizador, e depende deste e do modo de pagamento.
A classe History faz a ligação entre um utilizador e uma aposta por ele realizada.
Um utilizador possuí ainda uma lista dos diferentes favoritos que ele pode ter escolhido no sistema, sendo que neste momento, apresentamos neste conjunto deportos, competições e participantes.
Por último, a classe Session representa uma sessão de um dado utilizador, permitindo que este não tenha de estar sempre a fazer login entre as diferentes páginas e limitando a que apenas consiga fazer login uma vez num dispositivo, uma vez que a sessão será passada como cookie entre o browser e o servidor.

O modelo **Game** possui as classes dos jogos, com a sua informação, como o desporto, país,etc. e as respetivas odds. Permite a filtragem de jogos e as funcionalidades do especialista.

A classe **Gamble** é responsável por disponibilizar ao apostar realizar as suas apostas. A classe Bet representa uma aposta genérica, podendo depois esta ser simples ou múltipla. Ela possuí portanto uma lista de Bet_games onde cada entidade é realmente uma aposta única num jogo, com o seu dado valor apostado e resultado atual. Assim sendo, o estado de uma Bet, apenas é alterado para vitória se todos os seus Bet_game forem vencedores, ou derrota se qualquer um for derrota. Esta mudança de estado é realizada em funções separadas de acordo com o tipo de aposta, sendo que novos tipos de aposta adicionados teoricamente apenas necessitariam de uma adição dessa função de mudança de estado (por exemplo apostas menos rigorosas em que se pode perder pelo menos x número de jogos, ou em que uma aposta para ser ganha deve coincidir com um dado score).

Numa outra perspetiva , muito mais orientado aos dados, obtivemos um modelo gerado automaticamente pela framework que pode ser considerado como um modelo lógico de uma base de dados pois inclui os detalhes inerentes a um modelo deste tipo: chaves, entidades, ...

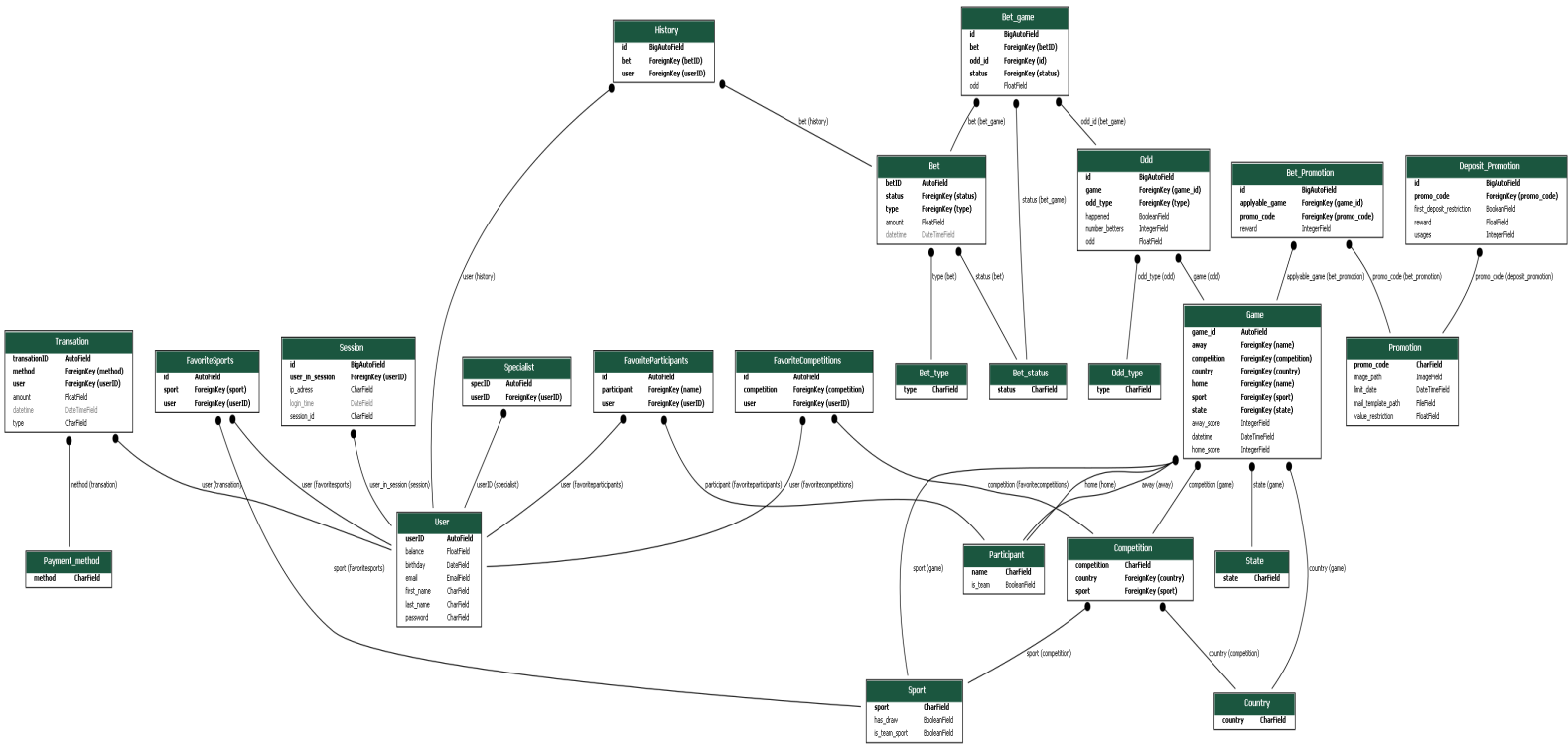


Figura 5.3: Diagrama de classes Django

6. Runtime view

Nesta secção, descreveremos o comportamento e a interação dos diferentes componentes do sistema. Esta descrição será feita recorrendo ao uso de diagramas de sequência que representam a forma como o sistema executa os diferentes Use Cases ou funcionalidades. Iremos abordar os seguintes casos de uso:

- Consultar jogos;
- Consultar desportos;
- Registo;
- Inserir e alterar odd;
- Fazer aposta;
- Consultar histórico de transações;
- Depositar dinheiro;
- Login;
- Alterar informações de perfil;
- Consultar histórico de apostas;
- Levantar dinheiro;
- Alterar estado de aposta;
- Criar promoções;
- Adicionar jogos

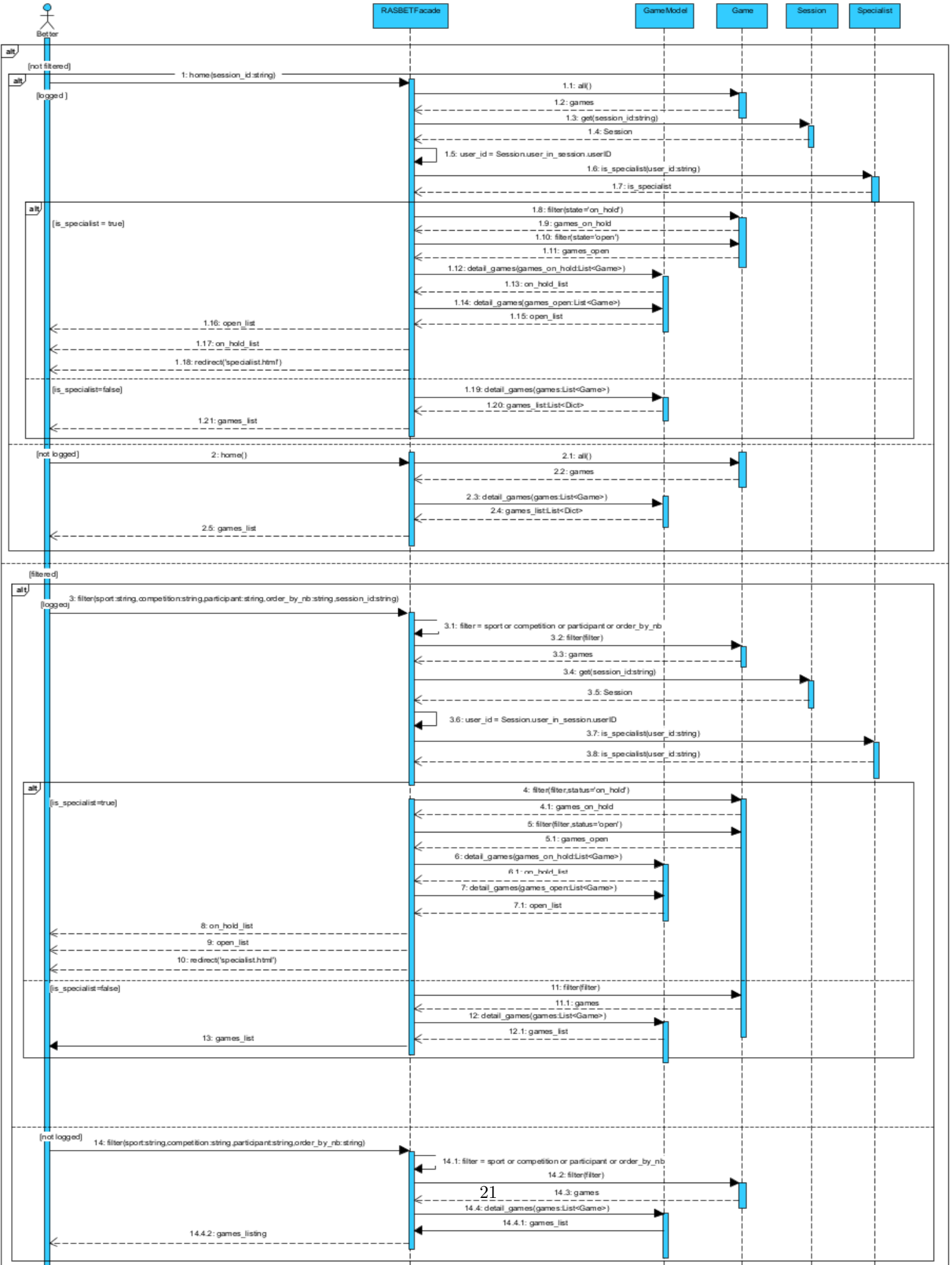
1 Consultar jogos

O primeiro diagrama que discutiremos é o de consulta de jogos. Esta é realizada por norma na página inicial do website, com dois principais modos, com e sem filtros.

Começando por discutir a consulta sem filtros, teremos uma nova divisão no caso de o cliente estar logged ou não no sistema, sendo que não estando logged, ele irá pedir para ir para a página inicial ao RASBETFacade, sem passar nenhum id de sessão, e este irá pedir à base de dados que devolva todos os objetos Game, que depois serão tratados pelo método no Game Model, que os torna em dicionários para uso na frontend. O Facade recebendo esta lista de dicionários de jogos irá devolve-la ao cliente.

No caso de o cliente estar logged, novamente serão pedidos todos os jogos, porém também será buscado a Sessão respondente ao session_id passado ao Facade. Com esta, poderemos saber o id do utilizador e, consequentemente, saber se ele tem ou não privilégios de especialista. No caso de não ter, o fluxo seguirá os passos iguais ao caso de ele não estar logged mas, se for um especialista, então o servidor pede envés todos os jogos, mas apenas aqueles que estão em estado de open ou on_hold, pois estes são aqueles que podem ser modificados pelo especialista, e devolve a lista dos dicionários destes ao cliente e redireciona-o para a página de especialista.

No caso do uso de filtros, então o cliente pedirá ao Facade os jogos através da função filter, onde passará os filtros que escolheu na página, de desporto, competição, participante ou ordenado. O Facade pedirá então os jogos de acordo com o filtro e, novamente no caso do utilizador ser um especialista, apenas os com o estado específico deste. O Facade devolve então a lista dos jogos filtrados e, no caso do especialista, redireciona-o para a sua página.


Better
RASBETFacade
GameModel
Game
Session
Specialist
alt
[not filtered]
alt
[logged]
1: home(session_id:string)
1.1: all()
1.2: games
1.3: get(session_id:string)
1.4: Session
1.5: user_id = Session.user_in_session.userID
1.6: is_specialist(user_id:string)
1.7: is_specialist
alt
[is_specialist = true]
1.8: filter(state='on_hold')
1.9: games_on_hold
1.10: filter(state='open')
1.11: games_open
1.12: detail_games(games_on_hold:List<Game>)
1.13: on_hold_list
1.14: detail_games(games_open:List<Game>)
1.15: open_list
1.16: open_list
1.17: on_hold_list
1.18: redirect('specialist.html')
[is_specialist=false]
1.19: detail_games(games:List<Game>)
1.20: games_list:List<Dict>
1.21: games_list
[not logged]
2: home()
2.1: all()
2.2: games
2.3: detail_games(games:List<Game>)
2.4: games_list:List<Dict>
2.5: games_list
[filtered]
alt
[logged]
3: filter(sport:string,competition:string,participant:string,order_by_nb:string,session_id:string)
3.1: filter = sport or competition or participant or order_by_nb
3.2: filter(filter)
3.3: games
3.4: get(session_id:string)
3.5: Session
3.6: user_id = Session.user_in_session.userID
3.7: is_specialist(user_id:string)
3.8: is_specialist(user_id:string)
alt
[is_specialist=true]
4: filter(filter,status='on_hold')
4.1: games_on_hold
5: filter(filter,status='open')
5.1: games_open
6: detail_games(games_on_hold:List<Game>)
6.1: on_hold_list
7: detail_games(games_open:List<Game>)
7.1: open_list
8: on_hold_list
9: open_list
10: redirect('specialist.html')
[is_specialist=false]
11: filter(filter)
11.1: games
12: detail_games(games:List<Game>)
12.1: games_list
13: games_list
[not logged]
14: filter(sport:string,competition:string,participant:string,order_by_nb:string)
14.1: filter = sport or competition or participant or order_by_nb
14.2: filter(filter)
14.3: games
14.4: detail_games(games:List<Game>)
14.4.1: games_list
14.4.2: games_listing

2 Consultar desportos

A consulta de desportos abrange a listagem numa aba da esquerda na página principal de site, assim sendo, esta consulta abrange categorias ainda mais detalhadas do que apenas o desporto, incluindo os países que têm este desporto e as competições de cada um destes países relativas ao dado desporto.

Assim, quando o utilizador pede para ir à página principal, o Facade irá preparar a listagem de desportos para o contexto, através do pedido ao GamesModel com a função sports_listing, que irá buscar todos os desportos existentes na base de dados e todos os países e, num dicionário, irá colocar para cada desporto, os países que o praticam, e para cada país as respetivas competições de cada desporto.

O modelo responde ao Facade com este dicionário que irá adicionar ao contexto, posteriormente passado ao cliente e utilizado na frontend.

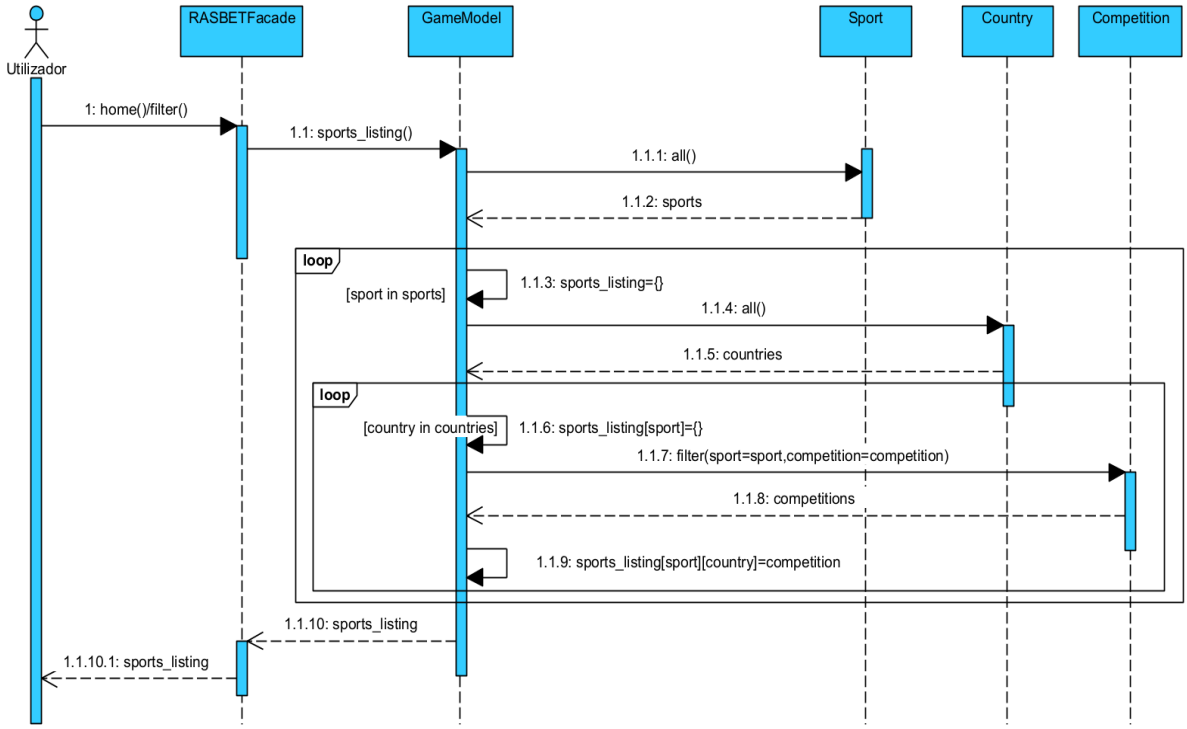


Figura 6.2: Diagrama de sequência de consulta de desportos

3 Registo

O registo começa a partir da página de signup do website, onde, após colocar os seus dados de conta, que serão verificados em frontend (em termos de validade de formato), o cliente pede para registar uma nova conta.

Este pedido é passado para o Facade que o irá redirecionar para o modelo Accounts, que tentará, com os dados fornecidos, pedir à classe User que insira uma nova instância na base de dados. Caso o email fornecido, que serve como identificador único dos utilizadores, já esteja a ser usado por algum utilizador, a inserção não será sucedida.

Caso o User tenha sido criado, o modelo Accounts irá criar uma Sessão para este User no seu IP atual. Isto serve para permitir que ele se mantenha logado entre as diferentes páginas e que não consiga fazer login em dois dispositivos diferentes ao mesmo tempo.

Por fim o modelo responde ao Facade com o resultado da criação da conta, e, em caso de sucesso este irá redirecionar o cliente para a página principal do site senão, irá transmitir uma

mensagem de erro ao cliente, avisando que o email escolhido já está em uso.

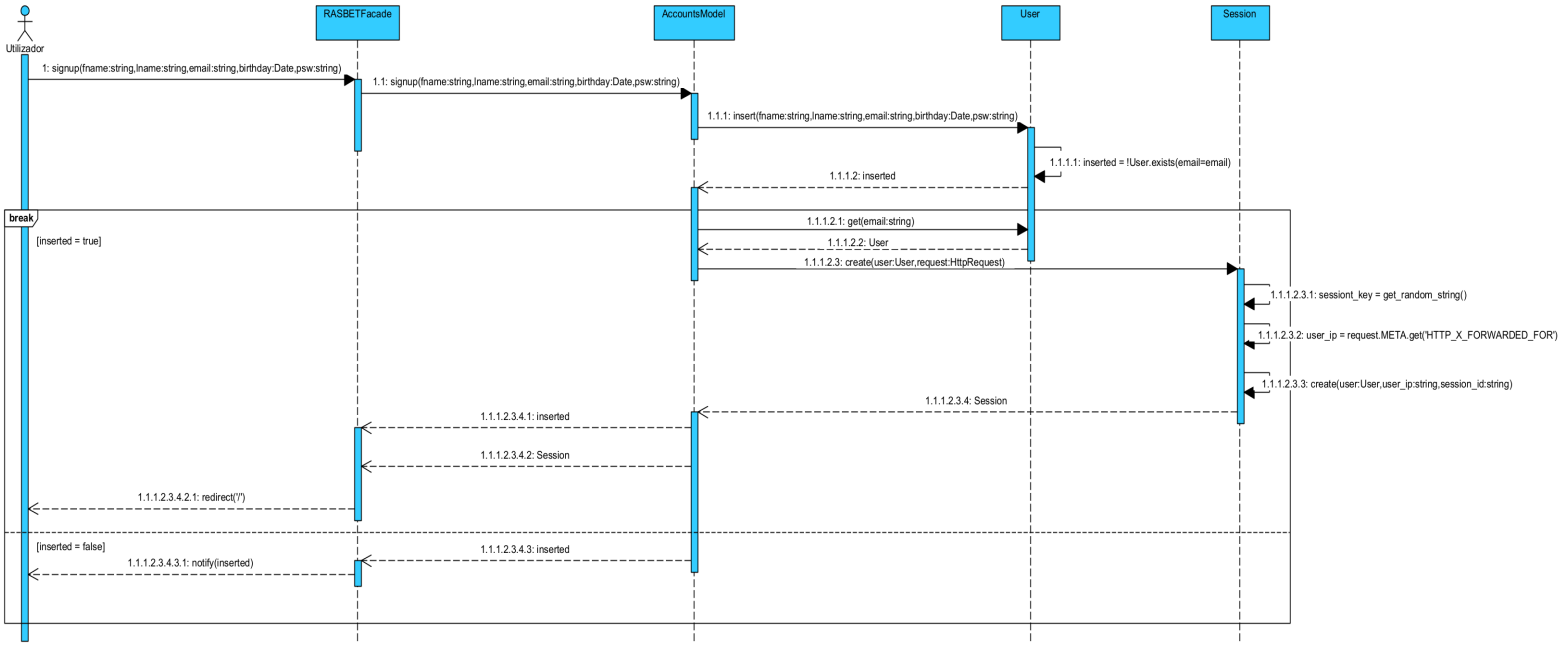


Figura 6.3: Diagrama de sequência de registo

4 Alterar odd e aprovar jogo

Este diagrama descreve as funcionalidades de um especialista, ou seja, a alteração das odds de um jogo, e a aprovação destes, isto é, mudar o estado de um jogo para open.

Após o especialista realizar as mudanças que pretende na aplicação web, ele passará um dicionário com todos os jogos modificados para o Facade, que será posteriormente tratado pelo GamesModel.

Para cada entidade no dicionário pode haver uma de duas mudanças a ser feita, ou foi modificada a odd de um jogo, ou o jogo foi aprovado.
No primeiro caso, o model chamará a função db_change_gameodd que, caso a odd ainda não exista, cria uma Odd para o jogo, senão altera o seu valor.
No segundo caso, o model chama a função db_change_gamestate que recebe um jogo e, neste caso irá mudar o seu estado para open. Esta função pode no entanto ser usada para alterar o estado de um jogo para qualquer um dos outros estados de jogo possíveis.

Por fim, o Facade dá reload à página em que o especialista se encontra.

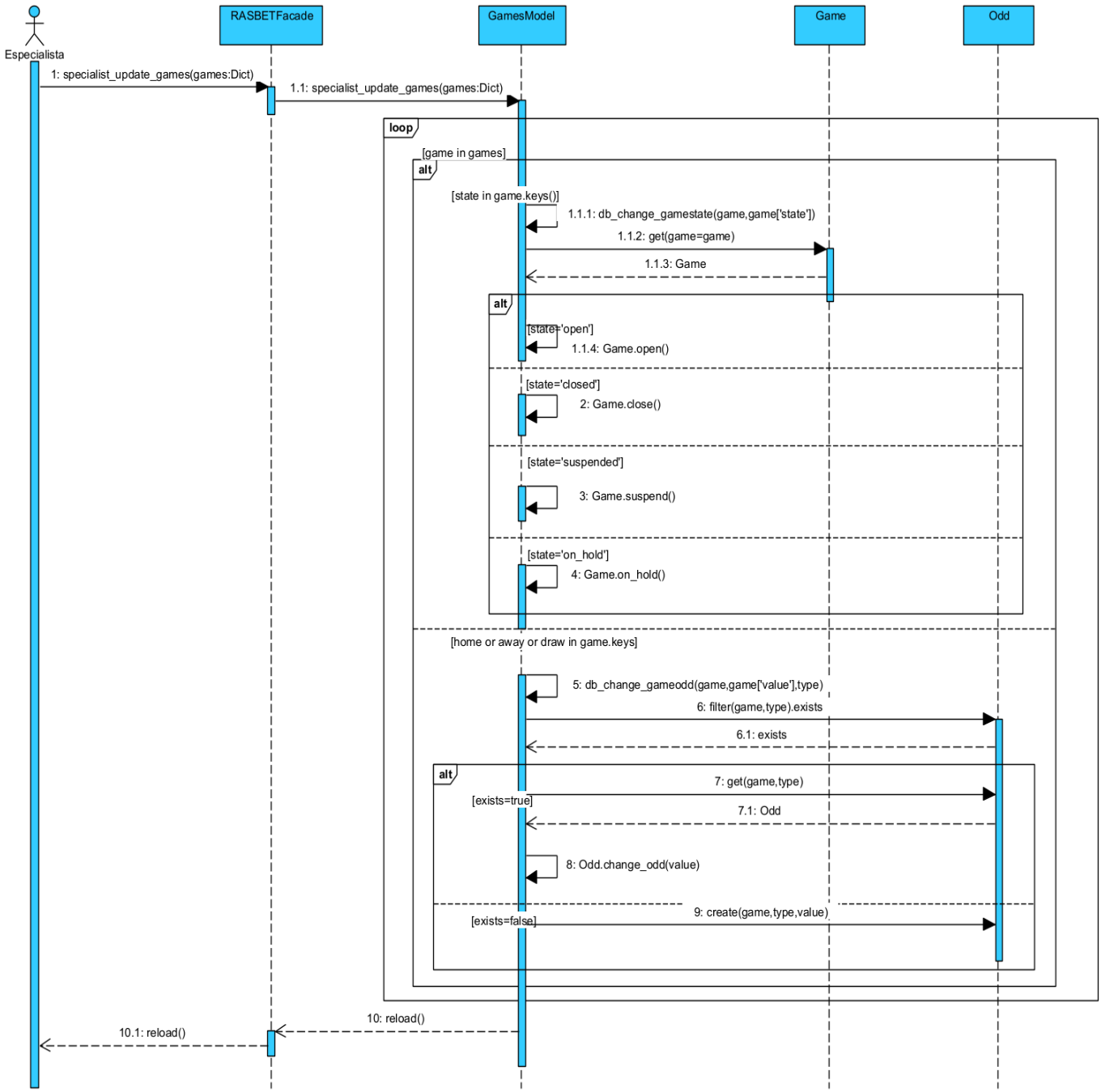


Figura 6.4: Diagrama de sequência de alteração de odd ou estado de jogo

5 Fazer aposta

O seguinte diagrama representa a principal interação do sistema, a realização de uma aposta, ou a tentativa de, visto que pode ou não ser realizada dependendo do saldo do usuário. O fluxo começa a partir do pedido do apostador para realizar uma aposta, esta pode no entanto ser simples ou múltipla, tendo mais tarde tratamentos diferentes.

Este pedido vai para o RASBETFacade, que o direciona para o modelo Gamble, onde é tentado criar uma transaction. Aqui, faz-se a verificação do saldo do utilizador e, caso este seja suficiente, a quantidade apostada será retirada do apostador e será criado um novo registo de transação.

O resultado desta Transaction será devolvido ao Gamble model e, em caso de sucesso, serão registadas as apostas, utilizando o método place_simple no caso de uma aposta simples e

place_multiple para múltipla. Este primeiro método irá criar uma instância de Bet_Game, que une uma aposta, uma odd, referente a um dado jogo e por fim cria uma instância History para conectar esta aposta ao utilizador que a realizou.
O place_multiple irá funcionar de forma semelhante, porém em loop, criando um Bet_Game para cada jogo apostado, mas apenas um registo de histórico.

Por fim, o Gamble Model irá anunciar ao Facade o resultado da tentativa de aposta, e este notificará o utilizador do seu sucesso ou insucesso.

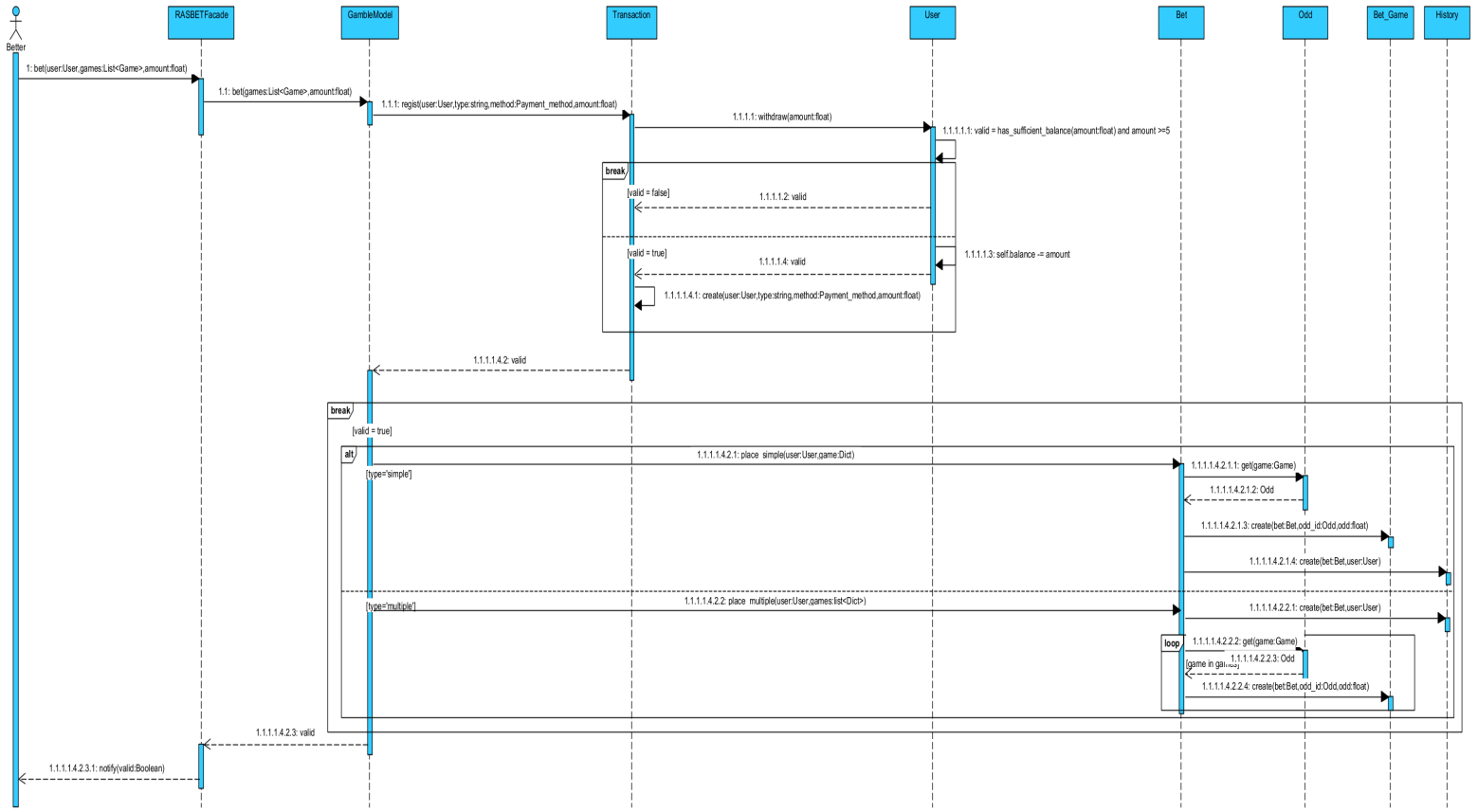


Figura 6.5: Diagrama de sequência de aposta

6 Consultar histórico de transações

Um utilizador pode consultar o seu histórico de transações através da página da mesma na web app. No caso do nosso site, esta página, além das transações, também fornecerá um conjunto de estatísticas relevantes ao histórico de apostas do utilizador, tais como taxa de vitória de apostas, número de apostas realizadas,etc.

Um utilizador pede então ao Facade que para lhe entreguer o histórico de transações, passando-lhe o session_id. O Facade distribui então o trabalho para o model Accounts, que irá através da sessão, obter o User em questão. Através deste User, o model pede todas as Transactions deste à base de dados, e todos os History (registos de bets do utilizador), estes últimos para a realização de estatísticas.
Seguidamente, a função history_transactions trata de todas as transactions, de modo a colocá-las numa lista de dicionários para serem utilizadas na frontend, e faz os cálculos necessários para apresentar o dicionário de estatísticas através do conjunto de History do User.

Por último, o Models devolve estes dois conjuntos ao Facade que os retorna ao Utilizador.

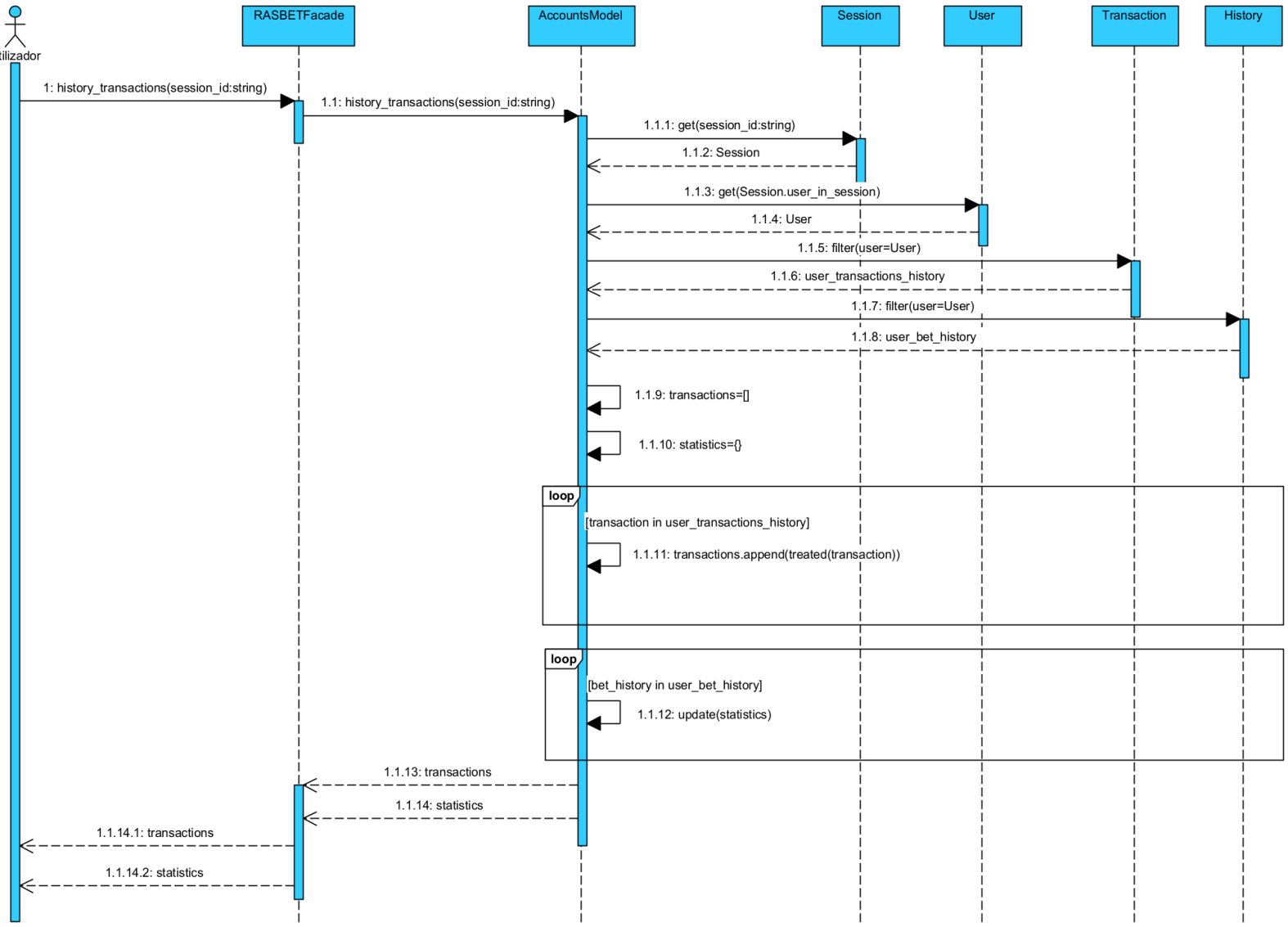


Figura 6.6: Diagrama de consulta de transações

7 Depositar dinheiro

O processo de depósito de dinheiro na conta de um utilizador começa com ele a pedir ao Facade que realize um depósito de uma dado montante, utilizando um certo método e utilizando ou não um código de promoção (caso este seja válido no cenário de utilização, a quantidade depositada irá ser superior à pedida).

O Facade irá chamar o método deposit no AccountsModel, onde primeiramente será verificado a existência do código de promoção, se este tiver sido passado. Caso o código não exista, será enviada uma mensagem de erro ao utilizador a dizer que o código é inválido.
Caso o código exista ou não tenha sido passado código, o método deposit passa a redirecionar o cliente para a página de método de pagamento, mbway ou cartão.

Sendo este apenas um trabalho com âmbito escolar, a única diferenciação entre os dois métodos em termos de código são os valores passados aos seus métodos no model, uma vez que não conseguimos fazer as verificações que num contexto real se iriam fazer. Assim, no método de pagamento por mbway, será passado o número de telemóvel representativo da conta que irá pagar, enquanto que no método de cartão será passado o número do cartão, o nome do titular e o número de segurança.

Dentro da função de pagamento no Accounts model, começa-se por iniciar a variável reward, valor que será acrescido pela possível promoção, a 0 e através do id de sessão passado, vai-se buscar o User da sessão.
De seguida é realizada uma validação da promoção, em termos da sua data e do montante limite a ser utilizado, e em caso de erro, o Utilizador será redirecionado para a página de balanço.
Caso seja válida a promoção, será realizada a transação de depósito por método de pagamento, seguida de uma transação de depósito por promoção, onde é depositado o reward na conta do User. Caso não tenha sido usado um código de promoção, apenas a primeira transação se aplica.

Por fim, o utilizador é redirecionado para a página do balanço.

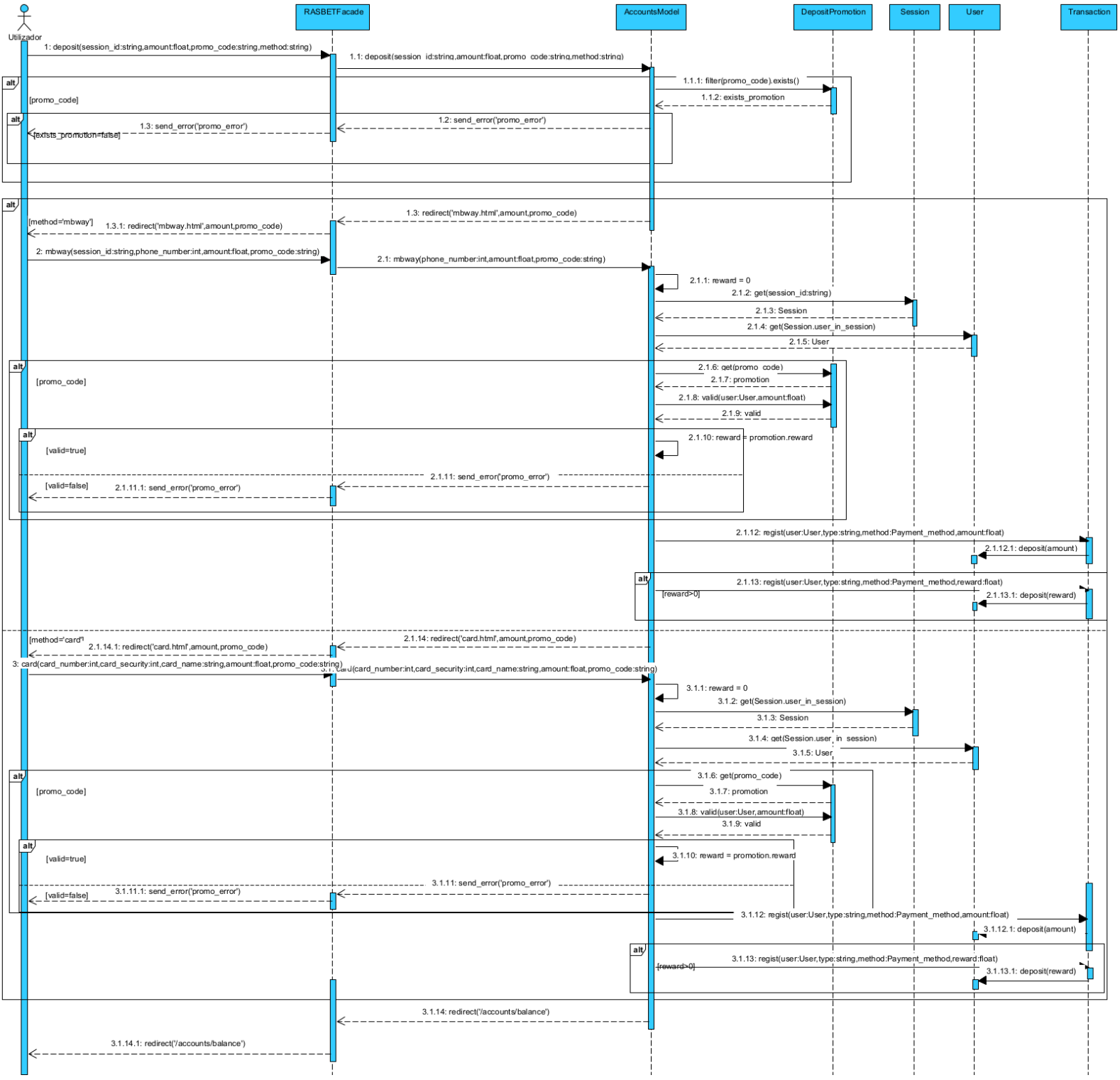


Figura 6.7: Diagrama de sequência de depósito

8 Login

Neste diagrama, o utilizador pretende realizar login no sistema. Assim, o utilizador manda um pedido de login que vai se recebido pelo RASBETFacade, que o orienta para o AccountsModel, componente responsável pelas contas de utilizadores no sistema. De seguida, esta componente verifica se existe algum utilizador com o email e password de input com a função `verify_login`.

Caso existe algum *user* com os parâmetros de input, verifica se o mesmo já tem uma sessão iniciada: em caso afirmativo, cria uma *cookie* com a mesma; caso contrário, cria uma sessão nova e cria uma cookie. Ora, caso exista um utilizador, depois de tratar da cookie e da sessão, redireciona o utilizador para o diretório `"/"`; caso contrário, envia uma mensagem de erro ao utilizador que está a tentar fazer login no sistema.

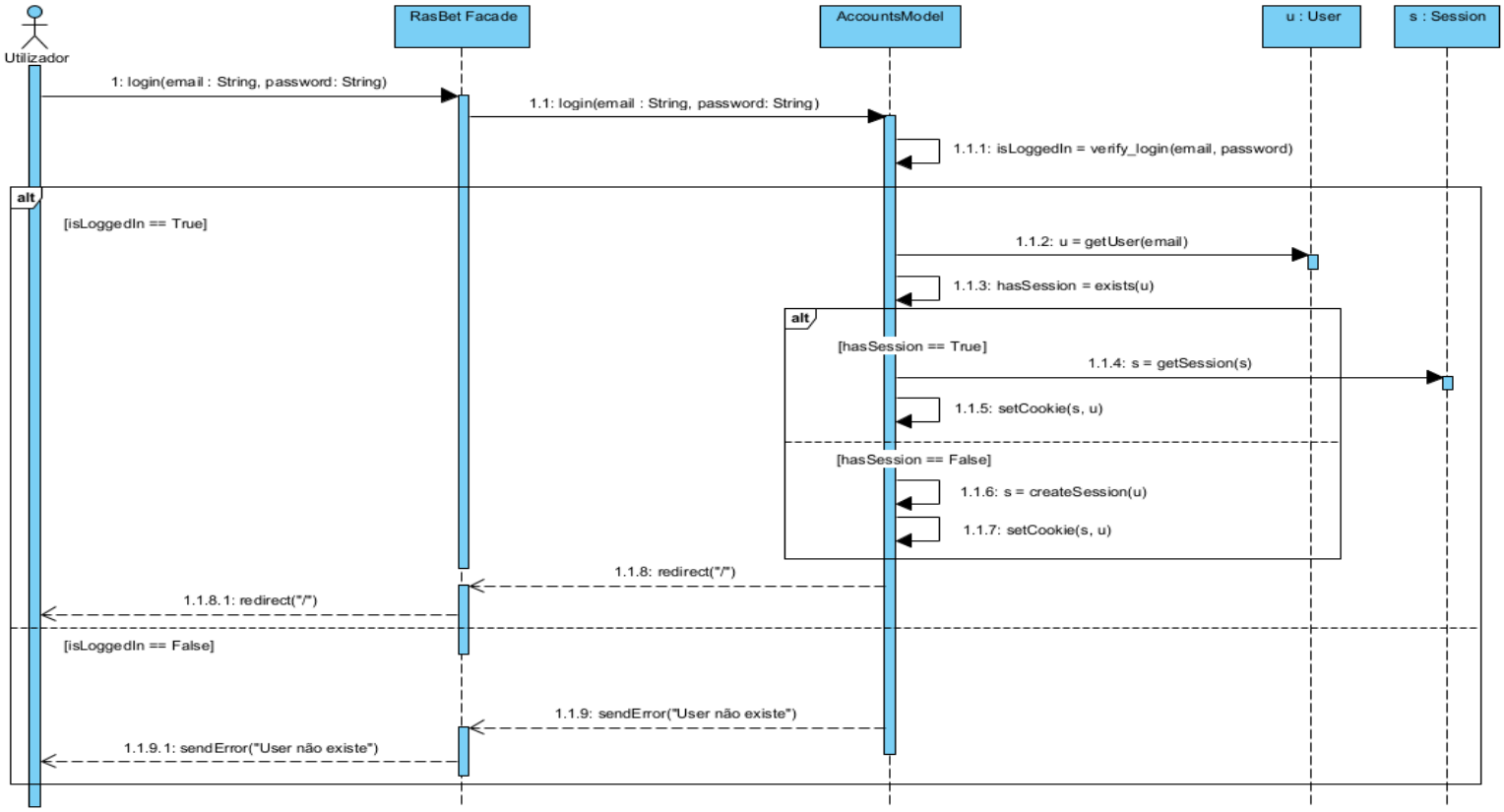


Figura 6.8: Diagrama de sequência de Login

9 Alterar informações de perfil

Neste diagrama, o utilizador pretende alterar as suas informações do perfil. Pode alterar o seu primeiro nome, último nome, e-mail, data de nascimento e password. Com intuito de simplificar o processo de desenho do diagrama, assume-se no mesmo que o utilizador pretende alterar todos os seus parâmetros de perfil de uma só vez, mas pode não acontecer na realidade. Com isto, o utilizador manda um pedido ao RASBETFacade que o reencaminha para o AccountsView. Aí obtem-se a sessão e o utilizador em questão e alteramos diretamente todos os parâmetros (exceto a password) com os novos valores, onde se faz um *refresh* (redireciona para a mesma página) permitindo ver os valores atualizados. Neste momento, o utilizador precisa de alterar a sua password e, para tal, tem de enviar um pedido ao RasBetFacade que contêm três strings: password atual, nova password e confirmação da nova password. Este reencaminha-o para o AccountView que verifica se a palavra passe assumida como a atual é realmente a verdadeira e se as duas strings seguintes coincidem. Caso isto se verifique, o utilizador é redirecionado para a página do seu perfil com a password atualizada; caso contrário, envia uma mensagem de erro a alertar que a password não foi alterada.

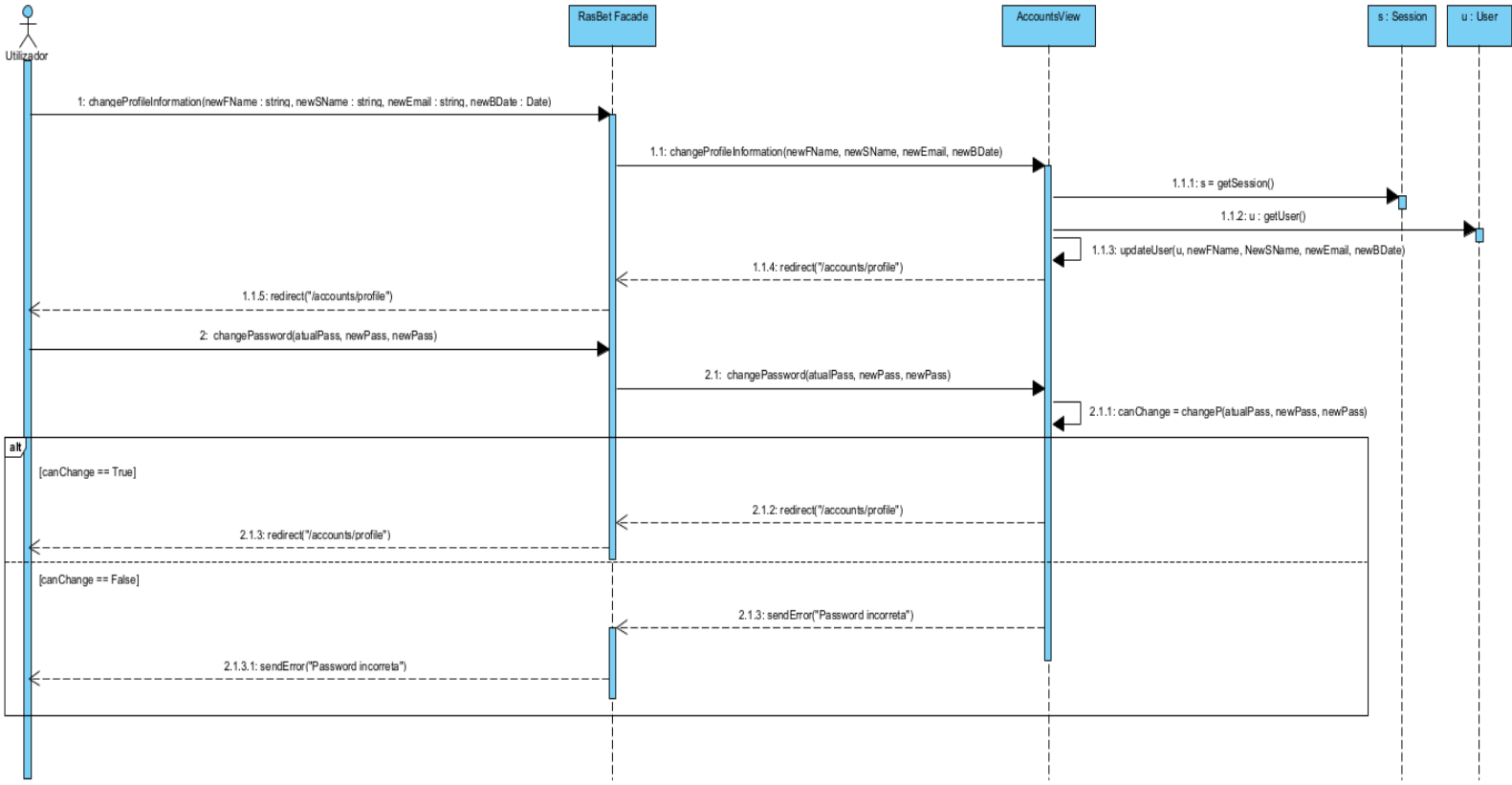


Figura 6.9: Diagrama de sequência de Alteração das informações do perfil

10 Consultar histórico de apostas

Neste diagrama, o utilizador pretende consultar o seu histórico de apostas realizadas na plataforma RASBet. Assim, envia um pedido ao RASBETFacade que o reencaminha para o AccountsView. O utilizador só receberá o seu histórico de apostas se tiver uma sessão iniciada, correndo o risco de ser redirecionado para a página de login. Caso já tenha uma sessão, a AccountsView obterá o utilizador em questão e vai consultar todas as apostas simples e múltiplas, redirecionando o utilizador para uma página com toda esta informação, incluindo apostas cujo resultado ainda não foi divulgado.

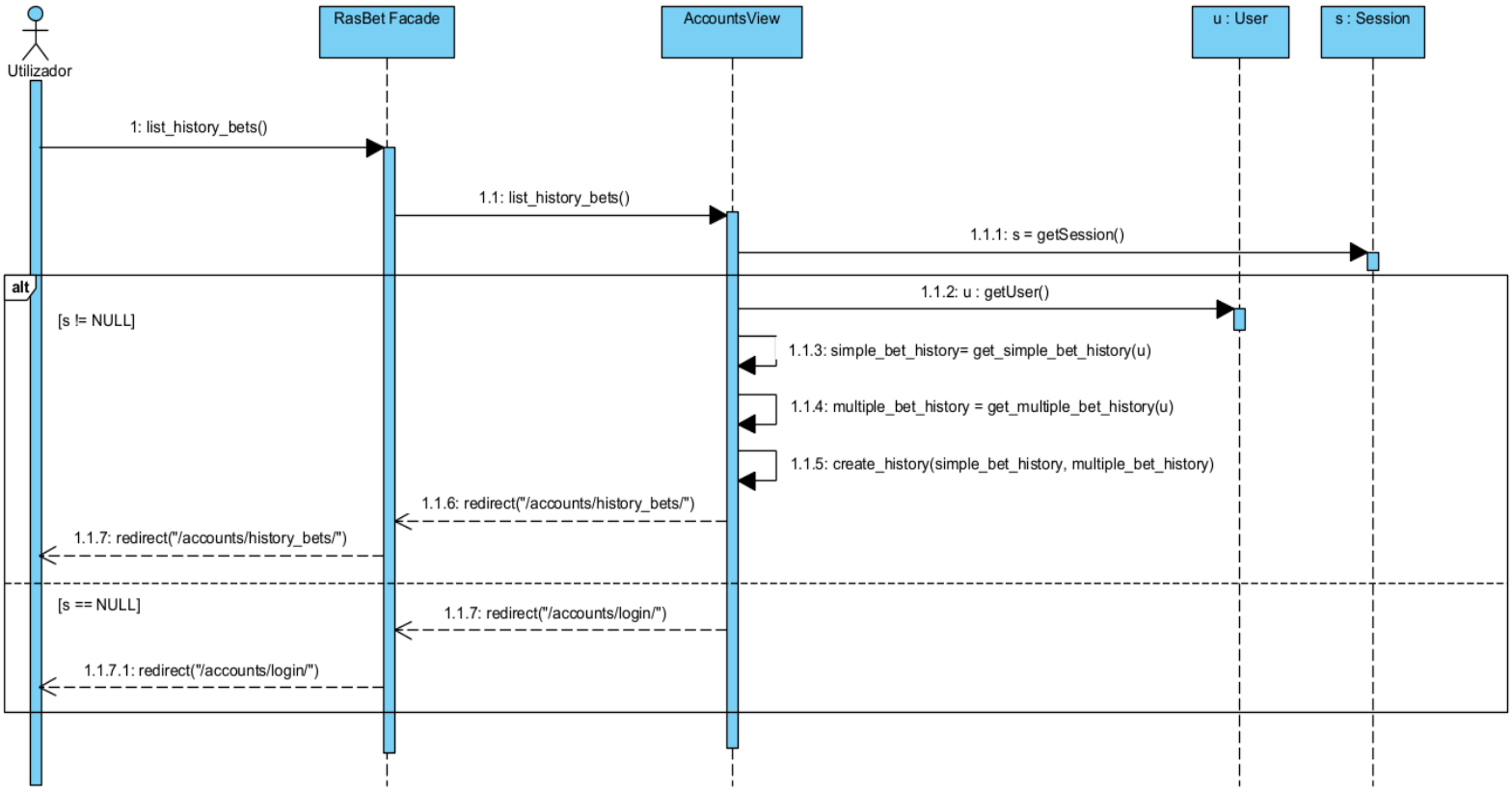


Figura 6.10: Diagrama de sequência de Consulta de histórico de apostas

11 Levantar dinheiro

Neste diagrama, o utilizador pretende levantar dinheiro da sua carteira do sistema. Para tal, faz um pedido ao RASBETFacade que o reencaminha para o AccountsModel. De seguida, obtém o utilizador em questão e verifica se pode efetuar um levantamento (i.e. montante não inferior a 5€ e montante a levantar ser inferior ao saldo da carteira). Em caso afirmativo, efetua uma transação na tabela de base de dados Transaction e retira o devido valor, redirecionando o utilizador para a mesma página do saldo com o valor atualizado. Caso contrário, envia uma mensagem de erro a informar que o levantamento pretendido é inválido.

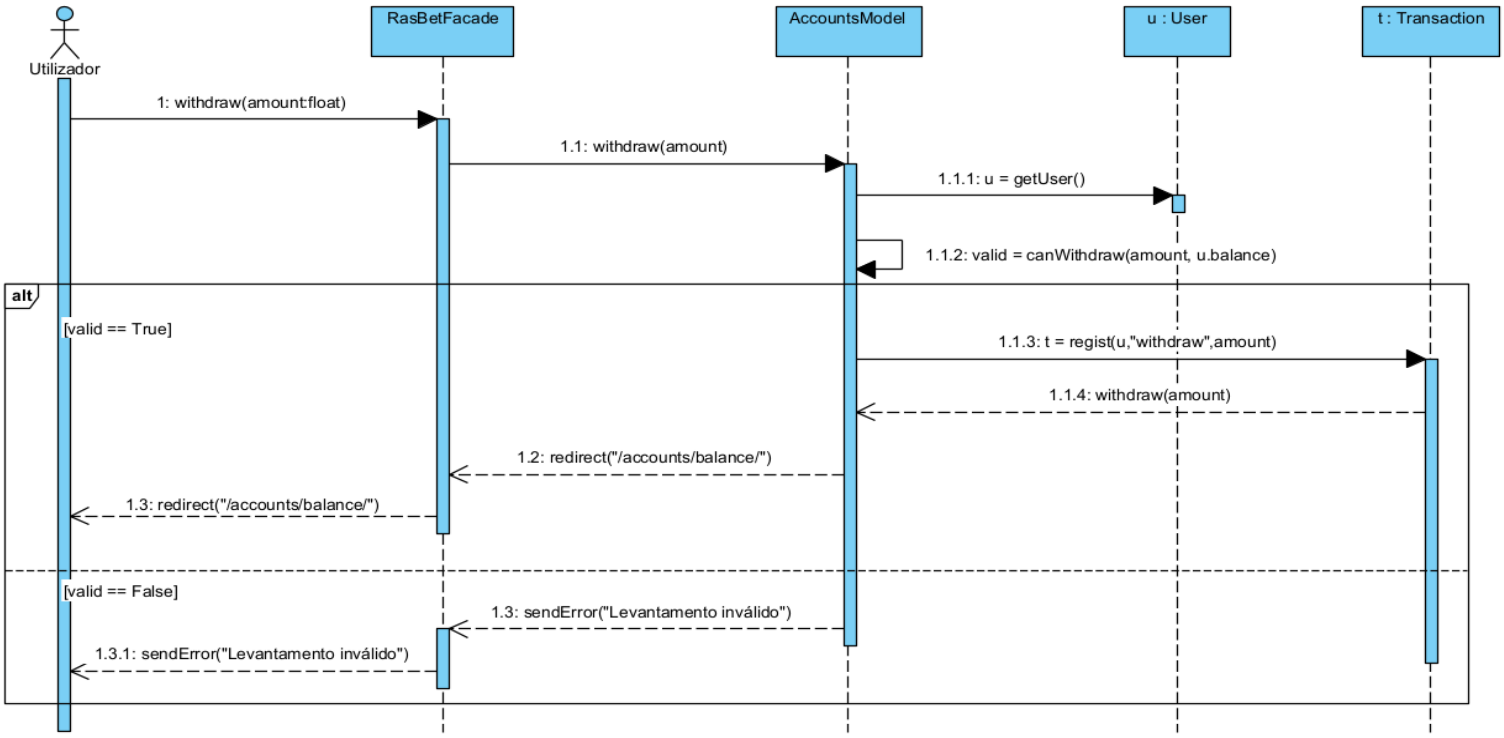


Figura 6.11: Diagrama de sequência de Levantamento de dinheiro

12 Alterar estado de aposta

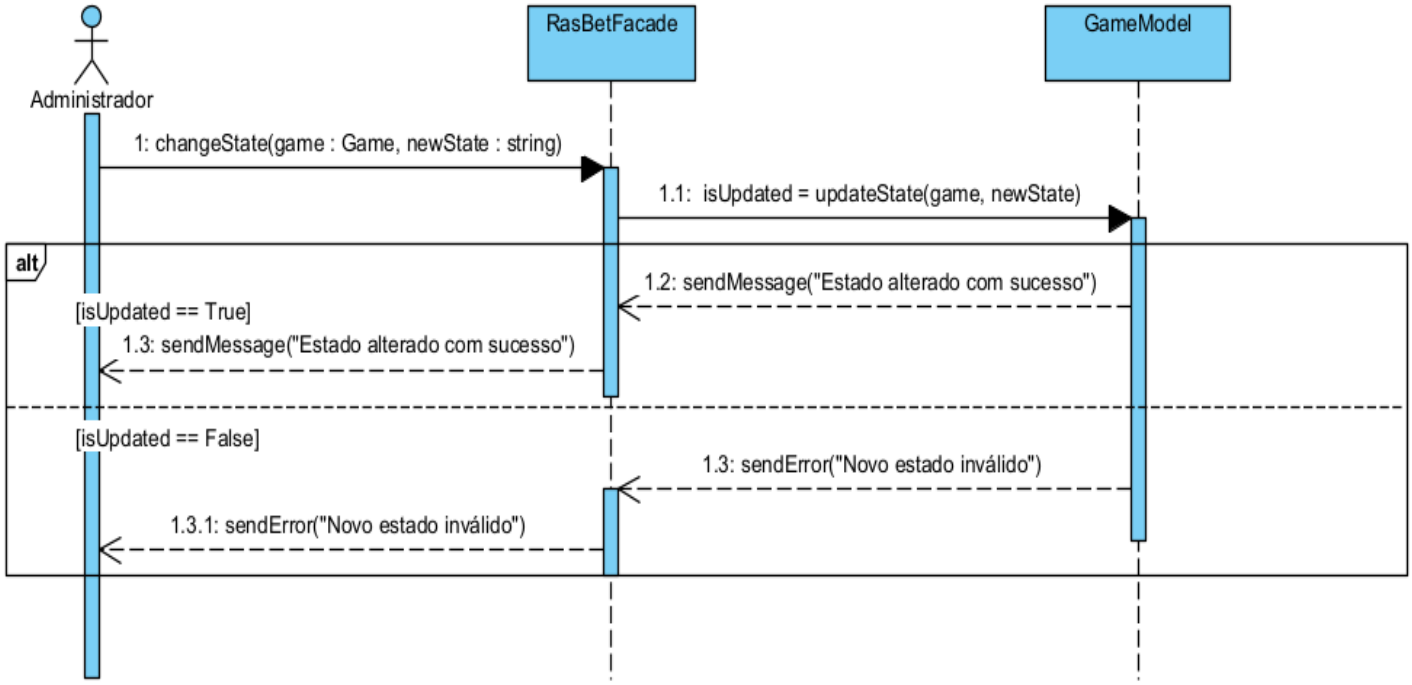


Figura 6.12: Diagrama de sequência de Alteração de estado de aposta

Neste diagrama, o administrador irá alterar o estado de uma aposta. Neste caso, o administrador manda um pedido ao RasBetFacade que o reencaminha para o GameModel. Aqui verifica se pode alterar o estado do jogo, ou seja, verifica se o novo estado não coincide com o estado antigo. Em caso afirmativo, atualiza o valor e devolve uma mensagem de sucesso, caso contrário não atualiza e devolve uma mensagem de erro.

13 Criar promoções

Neste diagrama, o administrador irá criar promoções para os utilizadores da plataforma, quer sejam promoções de apostas ou promoções de depósito. Assim, o administrador manda um pedido ao RasBetFacade que o reencaminha para o AccountsModel. Este cria uma entrada na tabela de promoções na base de dados com os parâmetros de input. De seguida, irá enviar um e-mail a todos os utilizadores com o recurso a um ficheiro HTML e uma imagem criadas especificamente para o envio desta promoção em concreto. Para tal, a equipa criou um e-mail para este propósito: rasbetpl32@gmail.com.

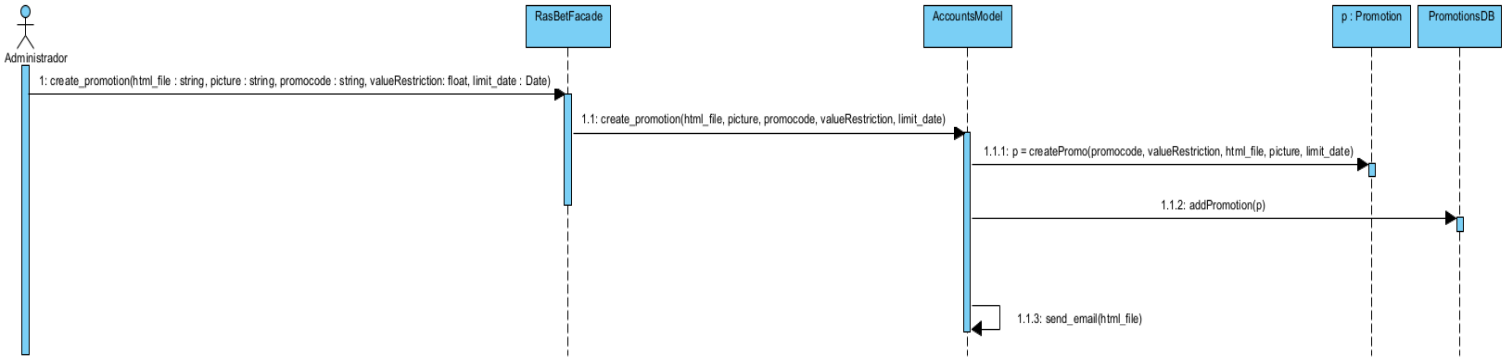


Figura 6.13: Diagrama de sequência de Criação de promoções

14 Adicionar jogo

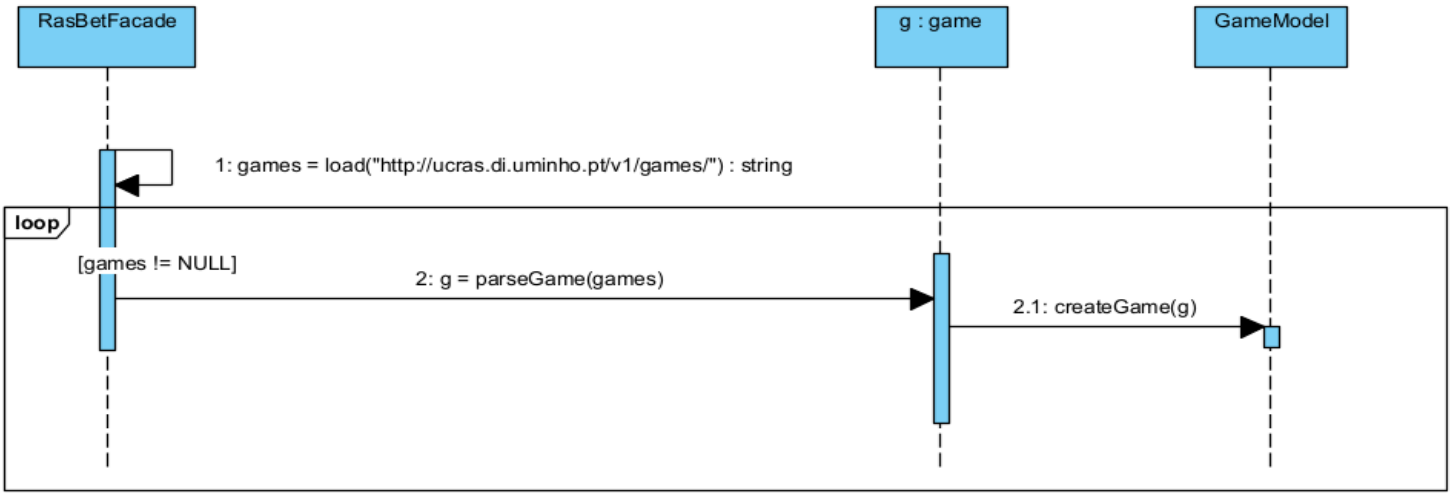


Figura 6.14: Diagrama de sequência de Adição de jogos

Neste diagrama, pretende-se adicionar jogos ao sistema. Não existem quaisquer atores neste diagrama pois quem providencia os jogos é uma API escolhida pela equipa docente (`"http://`

`ucras.di.uminho.pt/v1/games/"`), cuja resposta vem em formato JSON para o RasBetFacade. De seguida, cria-se um jogo através do *parse* de uma entrada da resposta e adiciona-se à lista de jogos do sistema.

7. Deployment view

Nesta secção, elaboramos um diagrama de *deployment* que permite visualizar qual o hardware onde o programa implementado será executado. Numa primeira versão do sistema, todos os componentes do programa (servidor, cliente, base de dados) exceto a API RASBET vão ser executados na mesma máquina, mas, uma vez que se trata de um projeto académico, no diagrama pressupõe-se que são nodos de computação em máquinas diferentes, facto que aconteceria na realidade.

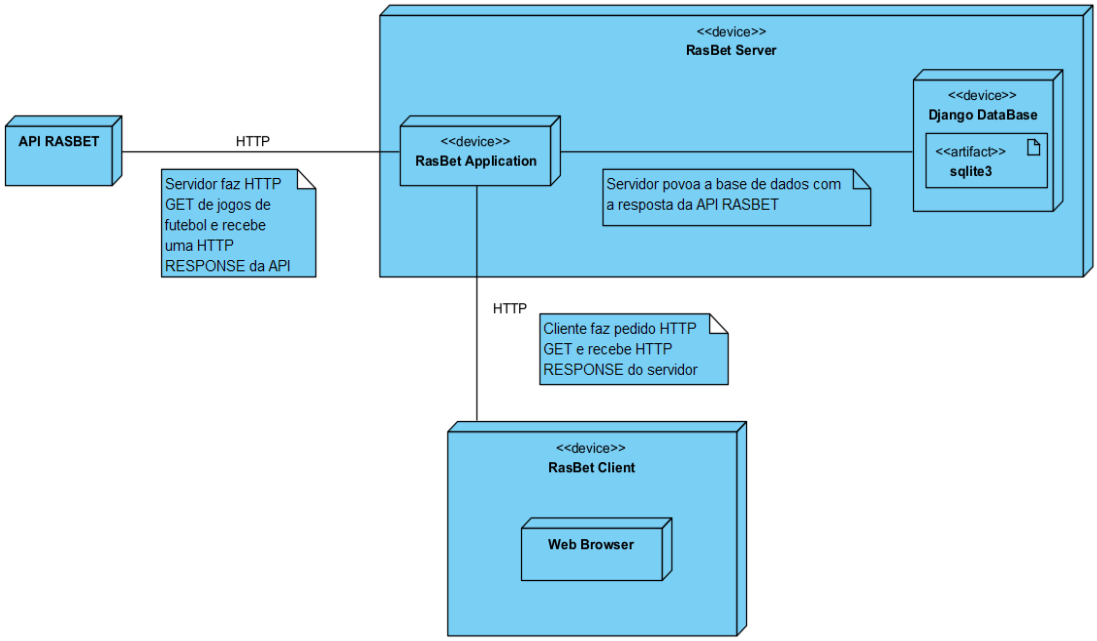


Figura 7.1: Diagrama de *deployment*

8. Decisões de arquitetura

Nesta secção discutiremos algumas decisões arquiteturais realizadas de modo a resolver de forma simples diferentes problemas que surgem da implementação.

1 API

Foi notado no inicio desta fase que a API fornecida pela equipa docente ficou notavelmente mais lenta na sua resposta, deste modo podendo impactar gravemente o desfruto e utilização da aplicação por parte dos apostadores devido ao grande tempo de espera para a receção dos dados dos jogos.

Deste modo, de forma a diminuir o impacto desta falha, a solução implementada abrangeu que o servidor realiza-se a task de ir buscar os dados dos jogos periodicamente e guardá-los na base de dados. Assim, quando um cliente quiser estes dados, o servidor apenas terá de fornecer os dados que se encontram atualmente na base de dados, não havendo qualquer delay adicional.

Além disso, de modo a permitir a expansão do programa para aceitar novas APIs, esta é lidada numa única função, que trata dos dados por ela fornecida de acordo, de forma a em seguida apenas os ter de guardar na base de dados. Assim, para novas APIs adicionadas, apenas é necessária programar esta função de parsing dos dados.

2 Notificação das apostas

De forma a notificar os diferentes apostadores dos resultados das suas apostas, uma das alternativas mais simples, seria que eles constantemente pedissem ao servidor uma atualização dos seus resultados porém, isto causa uma carga excessiva do lado do servidor, o que por sua vez traria diminuições de desempenho notáveis para o utilizador.

A solução implementada foi então do estilo observer, onde as notificações são apenas realizadas quando um admin termina um jogo, e consequentemente todas as partes interessadas neste (quem fez apostas no jogo), são notificadas.

3 Expansabilidade

Com o intuito de permitir a diferentes sites que usem a aplicação Rasbet terem a sua própria identidade, através de, por exemplo, novos desportos, ou diferentes tipos de aposta, tomamos uma abordagem na implementação de forma a permitir o incremento destes, ao criar classes genéricas como Sport e Bet_Game, que depois é incluída na mais adaptável classe Bet.

9. Conclusões

Concluída a segunda fase do trabalho, chega o momento de realizar uma introspeção sobre solução implementada e o processo de planeamento para a tal, assim como um balanço dos objetivos derivados da primeira fase.

O planeamento realizado para os requisitos na fase 1, assim como as modificações mencionadas no início do documento a estes, permitiram uma mais focada e detalhada planificação e divisão do trabalho, evitando ambiguidades.

Destes requisitos, fomos capazes de implementar na generalidade os funcionais, considerando este balanço positivo, no entanto, o esforço e tempo dedicado a estes impactou a finalização dos não funcionais, algo que pretendemos corrigir para a última fase.

As dificuldades propostas pelos docentes, tais como a diminuição da eficiência da API Rasbet, assim como a necessidade de notificar os apostadores dos resultados das suas apostas quando estas terminam, permitiram a exploração de diferentes padrões arquiteturais da indústria, os quais cremos ter aplicado de forma eficiente, como o uso de cache na base de dados, e o estilo observer.

Por último, a nossa estratégia de implementação foi realizada de modo que permite expansibilidade ao sistema, tal como o aumento dos desportos disponíveis, fácil integração de novas APIs de dados de jogos e de criação de novos tipos de apostas, algo que, em cenário real, seria extremamente desejado visto este ser um produto para vender para outras empresas adaptarem.