UMinho

Mestrado Engenharia Informática

Requisitos e Arquitetura de Software

PL3, Grupo 1 / ENTREGA 2

PG47071, Bruno Barbosa
PG50419, Hugo Fernandes
PG50736, Rui Moreira
PG50519, José Gonçalves
PG50259, Bernardo Saraiva

Ano Letivo 2022/2023
Braga, dezembro de 2022

Prefácio

A especificação estrutural de um sistema, dá-se como a representação dos requisitos de forma arquitetural e numa perspetiva mais da solução; assim, existem muitas características a considerar que são cruciais para um documento completo e sucinto. Uma delas é a definição de metodologias que abordagens para a resolução de potenciais problemas ou objetivos a alcançar e, nesse aspeto, o documento está repleto de restrições, caracterizações tanto de entidades como do contexto onde se insere o sistema, além dos padrões arquiteturais e de *design* selecionados para compor o produto e o tornar mais robusto.
Um ponto a ter em consideração é a complexidade dos diagramas de sequência, contudo, elementos como a descrição do *pipeline* de tarefas empregues, a matriz de compoenentes e a tabela de interação dos métodos, representam auxiliares de compreensão e agilizam o processo de perceção do comportamento de cada um dos métodos.
É de frisar também a utilização de alguns dos tópicos presentes no *template Arc42*[3] e a consulta de alguns conceitos na bibliografia[1].
Finalmente, e no que diz respeito à contribuição dos elementos integrantes da equipa desenvolvedora, é de salientar que nenhum se destaca em relação aos outros, fruto de uma divisão justa de tarefas e um cumprimento assiduo das respetivas:

- PG47071, Bruno Barbosa - 0
- PG50419, Hugo Fernandes - 0
- PG50736, Rui Moreira - 0
- PG50519, José Gonçalves - 0
- PG50259, Bernardo Saraiva - 0

Resumo

O presente relatório divide-se em sete pontos distintos. No primeiro, é feita uma breve introdução em que se abordam os objetivos e as funcionalidades implementadas, de uma forma resumida. São ainda mencionados os *Stakeholders*, e as suas expectativas em relação ao projeto.

No segundo ponto, abordam-se as restrições colocadas na especificação/modelação e implementação, sejam elas técnicas, organizacionais, ou até conveções utilizadas.

Segue-se, no terceiro ponto, é abordado o contexto do projeto, enquadrando-o com as diversas peças em movimento nesta secção da indústria.

No quarto ponto, temos a estratégia adotada na solução, começando por uma árvore de objetivos de qualidade, seguindo-se um quadro que apresenta as abordagens tomadas em cada um desses objetivos. Segue-se ainda uma apresentação de padrões arquiteturais - *Model-View-Controller* e *Middleware* -, e de *design* - *Observer* e *Adapter*. Em seguida, no quinto ponto, a Modelação estrutural, abordada em três camadas, contendo cada uma visão progressivamente mais detalhada da arquitetura.

No sexto ponto, é abordada a Modelação comportamental, começando com uma enumeração das diversas funcionalidades presentes no sistema, seguindo-se uma secção de diagramas de sequência para cada uma dessas. Por fim, no sétimo ponto, um diagrama a exibir os diferentes componentes do sistemas completo, proprietários ou não, e a forma como a sua comunicação é feita.

Conteúdo

Lista de Figuras

Lista de Tabelas

1 Introdução e ojetivos

A realização deste documento tem por base a detonação de aspetos que articulam os requisitos levantados previamente com a implementação propriamente dita.

1.1 Síntese das funcionalidades

O produto tem como foco principal, a disponibilização de funcionalidades que trazem conforto e entretenimento aos Utilizadores. Deste modo, e adotando as secções relativas ao Âmbito do produto e Requisitos Funcionais do documento de requisitos, seguem o conjunto de funcionalidades agregadas pelo seu papel no sistema.

Acesso

1. Registo
2. Login

Consulta

3. Consultar histórico de apostas
4. Consultar histórico de transações
5. Consultar notificações
6. Consultar desportos
7. Consultar eventos

Administração Pessoal

8. Alterar Moeda
9. Editar Perfil
10. Movimentar conta

Administração Geral

11. Alterar estado do evento
12. Criar promoção

Atividade

13. Efetuar aposta
14. Seguir apostador
15. Cancelar aposta
16. Copiar aposta
17. Seguir evento

Geração

18. Gerar eventos
19. Gerar taxa de conversão de moeda

1.2 Objetivos da qualidade

Uma vez que se trata de um produto destinado a um número elevado de Consumidores, surge a necessidade de garantir um conjunto de aspetos no que diz respeito à qualidade do produto. Assim, e considerando o modelo de qualidade de produto ISO/IEC 25010[2], é possível identificar propriedades, prioridades e elementos que distinguem o produto dos restantes sobre a alçada dos requisitos não-funcionais já sublinhados no documento anterior; ora, mediante os variados elementos a selecionar é possível destacar alguns com um relevo superior, elegendo uma sub caracterísitica de cada objetivo de qualidade, 1:

Requisitos não-funcionais de maior relevo

1. O produto deverá garantir um tempo de resposta, na pior das hipóteses, de 1 segundo entre qualquer comunicação entre o utilizador e o sistema.
2. O produto deverá possibilitar a administração de capital via pagamentos com MBway, Multibanco, Visa e Bitcoin.
3. O produto deverá ser intuitivo e inclusivo à utilização por parte de todos os intervenientes.
4. O produto deverá estar funcional pelo menos 99% do tempo, com o intuito de preservar 1% para manutenção em horas com atividade muito baixa e outras problemáticas.
5. O produto deverá garantir a total privacidade dos dados fornecidos pelo utilizador.
6. O produto deverá possuir um código fonte legível e apropriadamente comentado, de forma a agilizar o processo de manutenção e correção de erros.

Objetivo de qualidade	Descrição
1 (Performance - Comportamento temporal)	Grau em que os tempos de resposta e de processamento e as taxas de produção de um produto ou sistema, satisfazem os requisitos no desempenho das suas funções
2 (Compatibilidade - Interoperabilidade)	Grau em que dois ou mais sistemas, produtos ou componentes podem trocar informação e utilizar a informação que foi trocada
3 (Usabilidade - Acessibilidade)	Grau em que um produto ou sistema pode ser utilizado por pessoas com a mais ampla gama de características e capacidades para atingir um objectivo especificado num contexto de utilização
4 (Fiabilidade - Disponibilidade)	Grau em que um sistema, produto ou componente está operacional e acessível quando necessário para utilização
5 (Segurança - Confidencialdiade)	Grau em que um sistema, produto ou componente impede o acesso não autorizado a, ou a modificação de, programas ou dados informáticos
6 (Manutenção - Modularidade)	Grau em que um sistema é composto por componentes discretos de tal forma que uma mudança para um componente tem um impacto mínimo sobre restantes

Tabela 1: Objetivos de qualidade primordiais

1.3 *Stakeholders*

Em conformidade com os intervenientes do projeto identificados no capítulo Clientes, Consumidores e *Stakeholders* do documento prévio, resta identificar cada um dos elementos consoante aquilo que é expetável dos correspondentes em articulação com o nome das companhias/empresas que representam. 2.

Stakeholder	**Identificação de tipologias**	**Expetativas**
Equipa de desenvolvimento	Programadores e Engenheiros	Ficar a cargo da implementação do produto, seja a nível estrutural, seja ao nível da documentação
Equipas de gestão do sistema	Gestores de projeto e gestores de mercado	Assegurar a harmonia entre entidades, o cumprimento de restrições e a evolução do produto
Equipas de análise de negócio	Analistas de processos e sistemas, contabilistas e administradores	Salvaguardar o crescimento do produto em termos de receitas, projeção e integração na indústria.
Investidores	Venture capital, Business angel e Corporate venture capital	Investir numa percentagem mínima do produto com um rendimento expetável do crescimento do respetivo
Patrocinadores	Nike, Adidas, Sagres, Vodafone e Eleven Sports	Contribuir e promover para o crescimento sob a forma de entreajuda com o produto, financeiramente e via publicidade, respetivamente
Entidades bancárias	Paypal, Mastercard, Banco de portugal e Abanca Serviços financeiros	Disponibilizar um inventário de métodos de transferência
Entidades governamentais	`SRIJ` - Serviço de regulação inspeção de jogos - e Governo de Portugal	Fiscalização do cumprimento de regras legais em conjugação com a atribuição de taxas sobre os lucros a satisfazer
Entidades desportivas	`IPDJ` - Instituto português do desporto e da juventude	Proporcionar a expansão do produto para mercados vinculados a outros desportos e eventos
Concorrência	Betclic, Betano, Solverde, Esc Online, etc	Garantir o insucesso do produto, indiretamente
União de trabalhadores	Recursos Humanos	Zelar pela harmonia entre trabalhadores intervenientes, a nível pessoal e também a nível de definição de vencimentos
Media	Jornais, Televisão, Internet e Rádio	Assegurar a divulgação do produto e fomentar a crítica construtiva, por parte do público em geral

Tabela 2: Cargos e identificação de *stakeholders*

2 Restrições

2.1 Restrições técnicas

As restrições tecnicas prendem-se com as ferramentas e utilidades escolhidas para a implementação, sejam elas linguagens de progrmação, bibliotecas, ou APIs.

Restrição	Panorama e/ou motivação
Ambiente de implementação	Implementação via *JavaScript*, `HTML`, `CSS` e *Mongoose*
Frontend framework	*Frontend* desenvolvido via *React* com recurso a *Bootstrap*
Backend framework	*Backend* desenvolvido com num ambiente *Node.js* através de *Express.js*
Bases de dados	Preservação dos dados na *cloud* via *MongoDB*
RASBet API	API fornecedora de dados a propósito dos eventos e desportos correspondentes
Coin Exchanger API	API fornecedora de dados a propósito das taxas de conversão entre moedas

Tabela 3: Restrições técnicas

2.2 Restrições organizacionais

As restrições organizacionais representam métodos de desenvolvimento, limites no orçamento ou calendarização, ou as ferramentas usadas, sendo que estas podem ir desde IDEs a sistemas de controlo de versão.

Restrição	Panorama e/ou motivação
Equipa, Agenda e Orçamento	Aspetos já ilustrados sob a forma de Restrições do projeto no documento de requisitos preliminar.
Ética de trabalho e modelo de desenvolvimento	Modelo de desenvolvimento orientado ao risco e faseado; ora, a documentação mediante o template *Arc42* é a chave para o sucesso do produto
Ferramentas de desenvolvimento	Implementação via *Visual Studio Code*, documentação em *LaTex* via *Overleaf* e controlo da Base de dados via *MongoDB Atlas*
Controlo de versões	Monitorização e preservação das etapas de implementação via *Github*
Ferramentas de teste e processos de teste	Supervisão da carga de trabalho e do suporte da respetiva via *JUnit* através de simulações da realidade

Tabela 4: Restrições organizacionais

2.3 Convenções

Convenções referem-se a padrões de indústria adotados, sejam eles a nível da especificação, modelação ou implementação.

Convenção	Panorama e/ou motivação
Volere shell	Escrita e visualização de requisitos funcionais através da tabela do modelo de *Volere*
Arc42 Template	Modelação estrutural e topologias mediante a sequência de pontos do *template Arc42*
Diretrizes de codificação em *JavaScript*	Comentários associados ao desenvolvimento na própria implementação em *JavaScript*, com o objetivo de elucidar o propósito dos métodos para iterações futuras

Tabela 5: Convenções

3 Âmbito e contexto

3.1 Contexto de negócio

De forma a representar de uma forma sucinta a interação entre entidades participantes do sistema, é necessário demonstrar o contexto em que se inserem e como comunicam entre si; ora, o diagrama 1 mostra todos os *stakeholder* já realçados em 2 e é possível verificar que apesar de todos serem participantes no sistema, à partida não interagem diretamente com os Utilizadores do mesmo. Outro aspeto que é possível averiguar do contexto de negócio do sistema, é o fluxo de comunicação entre entidades, principalmente na distinção entre entidades que fornecem dados e entidades que apenas participam de uma forma superficial no produto. Há que salientar também a divergência entre a entidade Ator, e a entidade *stakeholder*.

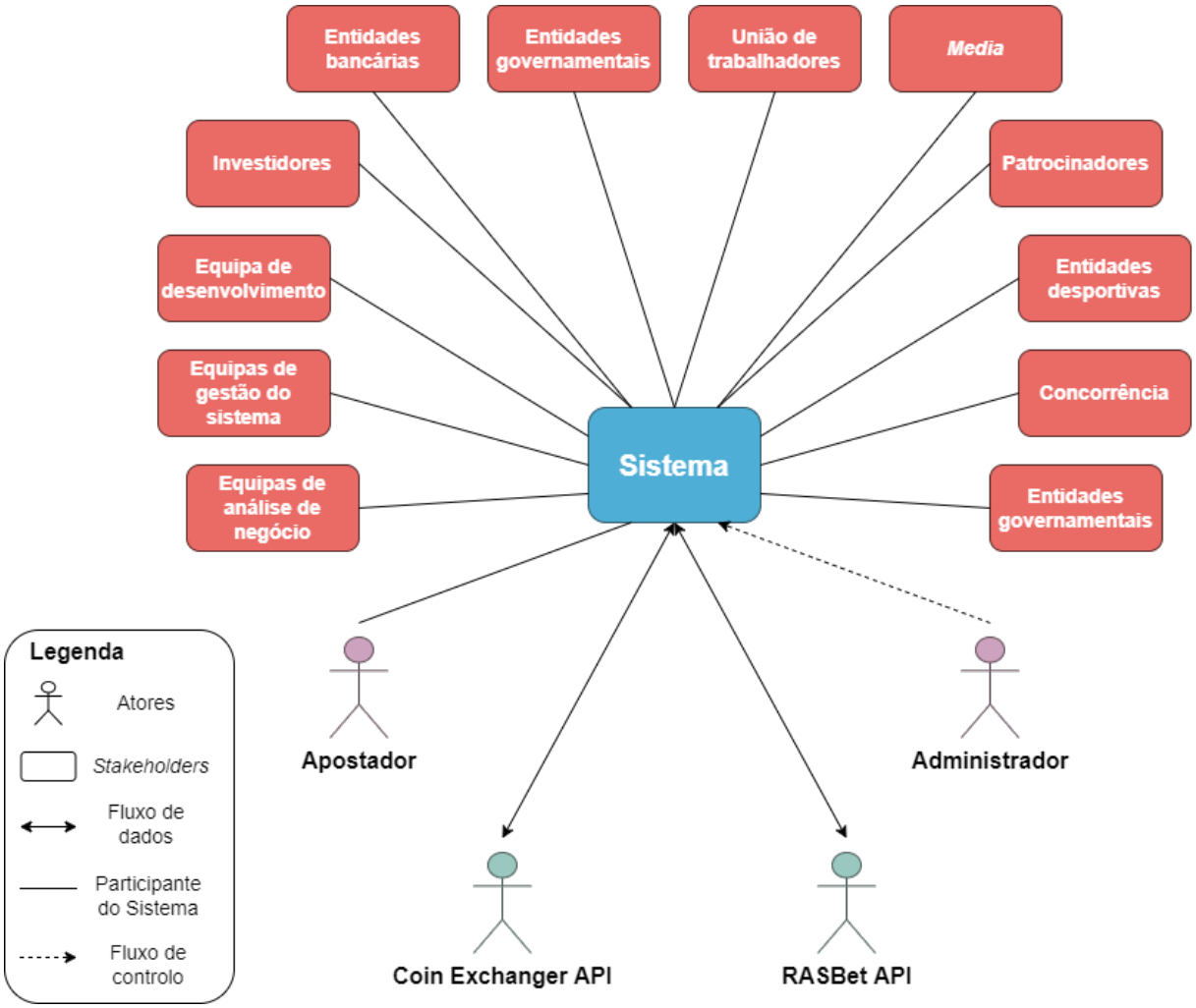

Figura 1: Diagrama de contexto de negócio

4 Estratégia da solução

4.1 Árvore dos objetivos de qualidade

Previamente foram identificados alguns objetivos de qualidade representantes de requisitos não-funcionais, contudo, e tendo em consideração a dimensão da amostra dos requisitos de maior relevo, não é possível identificar a dimensão desta categoria; assim sendo, segue um diagrama que demonstra precisamente a posição dos objetivos mencionados no espaço da qualidade de produto, 2. É de frisar que ainda restam duas categorias a explorar, e talvez se fossem representados todos os requisitos não funcionais, poderia surgir algum numa delas.

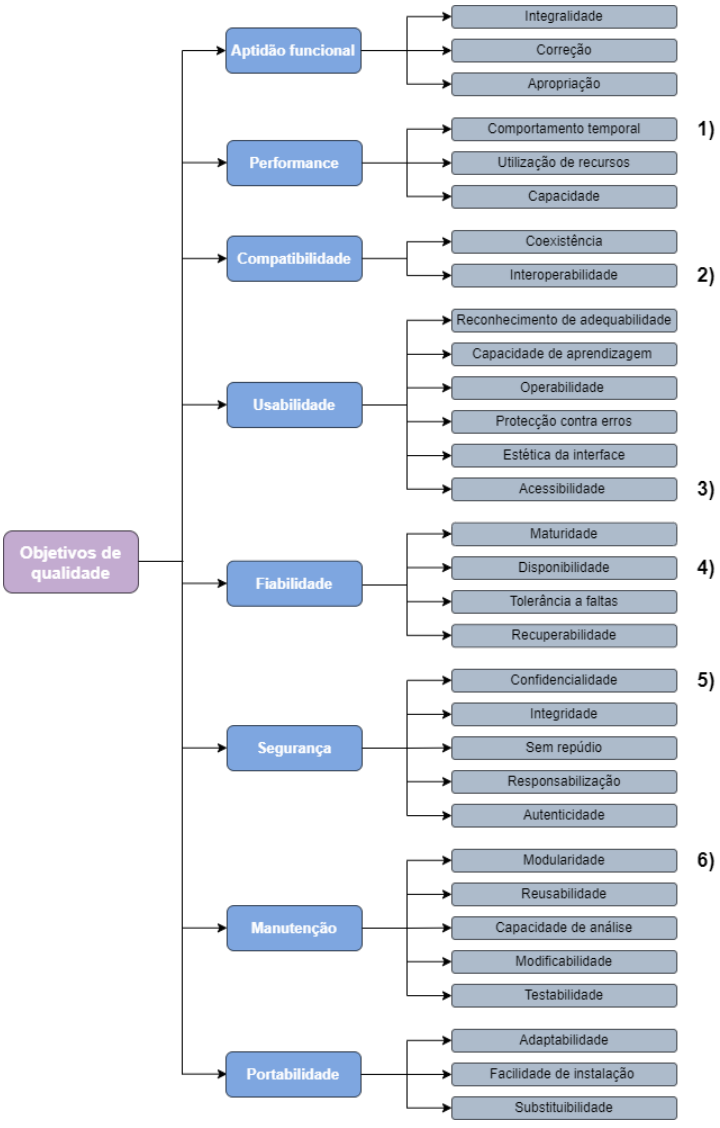

Figura 2: Árvore dos objetivos de qualidade

4.2 Estratégias dedicadas aos objetivos de qualidade

Além da identificação dos objetivos, é necessário levantar formas ou processos para a sua resolução e preservação; deste modo, segue uma tabela que conjuga cada um dos objetivos de qualidade, 6, com metodologias impostas sobre a arquitetura e outros tópicos que delimitem as capacidades do sistema. É de realçar novamente a dimensão dos elementos de qualidade de produto e as propostas que o sistema a desenvolver tem para cada um deles, todavia, manteve-se a amostra uma vez que representa a fatia imprescindível no que diz respeito a objetivos a alcançar.

Objetivo de qualidade	Abordagens de correspondência na solução
1 (Performance - Comportamento temporal)	• Integração de testes com limites temporais de resposta.
2 (Compatibilidade - Interoperabilidade)	• Parcerias com entidades bancárias e utilização das API's respetivas.
3 (Usabilidade - Acessibilidade)	• Interface de utilizador minimalista que engloba grande parte das funcionalidades • Funcionalidades de gestão pessoal são todas colocadas no perfil
4 (Fiabilidade - Disponibilidade)	• Recurso a alocação de servidores, dinamicamente, consoante a carga de trabalho presente no sistema. • Manutenção semanal por parte da equipa desenvolvedora, sob a alçada do horário de menor utilização do sistema, calculado via equipas de análise de negócio perante a concorrência.
5 (Segurança - Confidencialdiade)	• Preservação dos dados de cada Apostador de forma privada, em contas encapsuladas nas bases de dados.
6 (Manutenção - Modularidade)	• Código fonte separado por módulos e packages de identificação trivial. • Utilização de padrões de *design* específicos para a abstração e flexibilidade da solução.

Tabela 6: Estratégias dedicadas aos objetivos de qualidade primordiais

4.3 Estratégias baseadas em padrões

Além dos tópicos um pouco mais superficiais acerca de abordagens a seguir em 4.2, surgem outras problemáticas e funcionalidades que requerem um pouco de atenção estrutural e a delimitação de alguns padrões pré-definidos que sejam robustos para arquitetura e para a solução final do produto; assim sendo, seguem os padrões a implementar de maior destaque no núcleo do sistema, o padrão `MVC`, 4.3.1, o padrão *Middleware*, 4.3.2, o padrão *Observer* 4.3.3 e o padrão *Adpater* 4.3.4.

4.3.1 Padrão arquitetural *Model, View e Controller* (MVC)

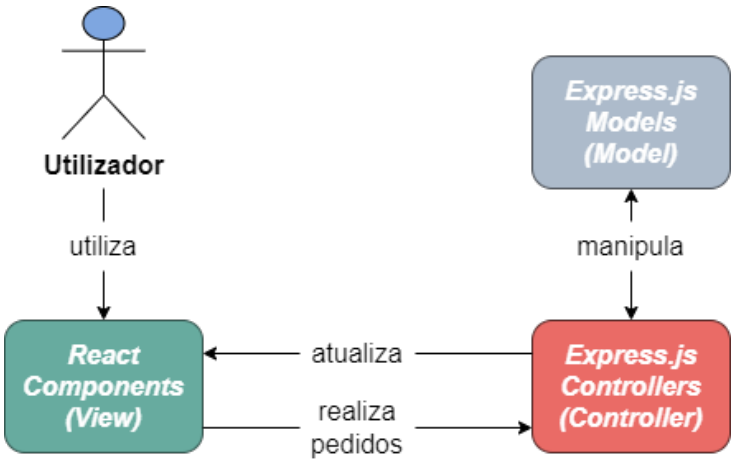

Figura 3: Padrão arquitetural - MVC

A arquitetura do sistema a desenvolver terá por base o conceito de `MVC`, dadas as suas características funcionais simplistas e a sua flexibilidade de componentes, contudo, não será uma representação convencional do mesmo, onde a *View* e o *Model* estão articulados (modelo triangular); ao invés da abordagem realçada, cada elemento terá algumas caracterísitcas específicas, como em 3:

Model Representa os componentes que mapeiam as tabelas da Base da dados, de forma a serem manipulados pelo *Controller*.

View É apresentada como a interface de utilizador (UI) e tem como papel principal, fornecer ao utilizador propriamento dito, uma ponte entre a lógica da aplicação e o que está a visualizar; por outras palavras, é aqui que as funcionalidades se iniciam e por sua vez, terminam.

Controller Trata de praticamente toda a lógica da aplicação, e tem acesso aos dados preservados na Base de dados; é o *Controller* que se responsabiliza do tratamento de dados mediante a resposta a pedidos provenientes da *View*.

4.3.2 Padrão arquitetural *Middleware*

A utilização deste padrão depende muito daquilo que é o contexto do sistema. De forma geral, o *Middleware* é um pedaço de *software* desenhado para tratar de pedidos e respostas; por outras palavras é um redirecionador de tráfego que tem a possibilidade de seleção dos pedidos que quer receber e enviar.
No âmbito deste produto, este padrão é utilizado devido à forma como uma aplicação em *Express.js* funciona, mediante um componente *Router*. Além de todas as funcionalidades que estes possibilitam, servem também como mais uma camada de abstração que encapsula o *Backend*, 6.

4.3.3 Padrão de *design Observer*

O padrão *Observer* é muitas vezes utilizado para definir entidades que observam outras entidades, com o propósito de notificar ou alertar algo; ora, para este efeito, a integração deste padrão dá-se para a resolução de dois *Use cases* - Seguir apostador e Seguir evento -, além de tratar da lógica de resolução de apostas quando os eventos são fechados, e notficações vinculadas a mudanças de estados - interações estas, já representadas em diagramas de sequência no documento prévio nos capítulos Representação das mudanças de estado dos eventos e Representação da interação de entidades na realização de uma aposta.
Assim, e tendo em conta o diagrama 7, surge a necessidade de identificar as *subjects* - entidades observadas - e os observadores - entidades que observam - realçados em 5.3; a interação destes componente é tão simplista como: o observador subscreve a *subject* que pretende seguir, e sempre que os *triggers* definidos forem ativados, ele notifica os utilizadores.

4.3.4 Padrão de *design Adapter*

Finalmente, e num contexto de representar a flexibilidade total no que diz respeito a fontes de dados, a equipa decidiu incluir mais uma camada de abstração que relaciona as API's com o *Backend* propriamente dito; isto funciona através de um componente *Adapter* que, neste caso, faz os pedidos à API, trata os dados, e devolve ao *controller* vinculado os dados de forma universal. Desta maneira, é possível incluir novas fontes de dados no futuro sem alterar o código dos *controllers* nem dos *models*.

5 Modelação estrutural

Passando à modelação estrutural, segue a estrutura do sistema, começando de um ponto de vista mais abstraído, e olhando, progressivamente, em mais detalhe.

5.1 Modelação estrutural - 1ª Camada

No que diz respeito ao sistema, numa visão mais superficial, de primeira camada, 4, podemos dividí-lo em diversas partes: *database*, *system* e API's. O *system*, divide-se em dois *subsystems*: *Backend* e *Frontend*. O *Backend* é onde se encontra toda a lógica de negócio do sistema, onde são feitas as interações com a base de dados, e onde é feita a conexão com as API's. Já o *Frontend* prende-se com a interação com o utilizador, seja este um administrador ou um apostador. Este subsistema tem a responsabilidade de comunicar com o *Backend*, pedindo-lhe os diversos dados, que deverão então ser apresentados ao utilizador, bem como o contrário, enviar ao *Backend* informação comunicada pelos utilizadores. A *database*, implementada aqui em *MongoDB*, guarda os diversos dados relativos ao sistema, que são então acedidos pelo *Backend*. Finalmente, as API's, servem para obter dados necessários ao funcionamento do sistema, que são fornecidos por *third-parties* e podem mudar em tempo real. Existem duas, sendo a primeira a RASBet API, que providencia informação sobre jogos, e as respetivas odds, e a segunda é uma Coin Exchanger API - nome fictício, como já foi mencionado anteriormente, que oferece ao sistema, conversões de moeda atualizadas. Estes componentes serão abordados em mais detalhe nas seguintes secções do relatório.

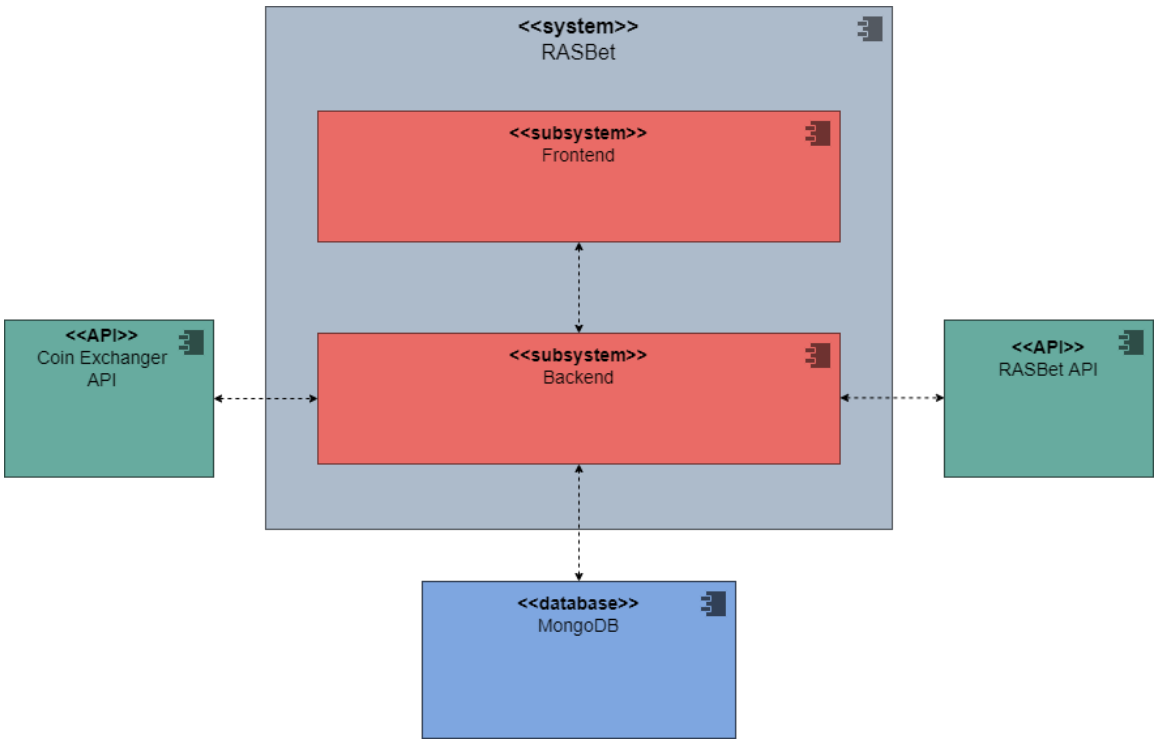

Figura 4: Diagrama de componentes - 1ª Camada

5.2 Modelação estrutural - 2ª Camada

Passando à segunda camada, podemos examinar os subsistemas *Frontend*, 5, e *Backend*, 6, em maior detalhe.

5.2.1 *Frontend*

Como já referido, O *Frontend* é responsável por exibir a informação aos utilizadores do sistema. Neste sentido, tem diversos componentes, representativos das diversas páginas e que a aplicação pode exibir, com o objetivo de segmentar essa informação de modo a tornar-se mais facilmente consumivel. Posto isto, existem componentes para que os utilizadores se registem, e entrem na sua conta - *Register* e *Login* -, componentes relativos aos dados pessoais do utilizador, como o seu histórico de apostas, dados de perfil, transações, notificações e administração de capital - *BetHistory*, *Profile*, *Transactions*, *NotificationsDropdown* e *Movement* - e o componente relativos aos eventos, desportos e às apostas propriamente ditas - *Events*. Existem ainda componentes auxiliares à navegabilidade da aplicação - *Home* e *Navbar* -, um componente exclusivo aos administradores do sistema - *Admin* - com as respetivas funcionalidades e um componente dedicado às funcionalidades responsáveis por seguir e deixar de seguir Apostadores - *CopyBetter*. Por fim, este subsistema deve comunicar com o Backend, quer para obter, quer pra devolver informação.

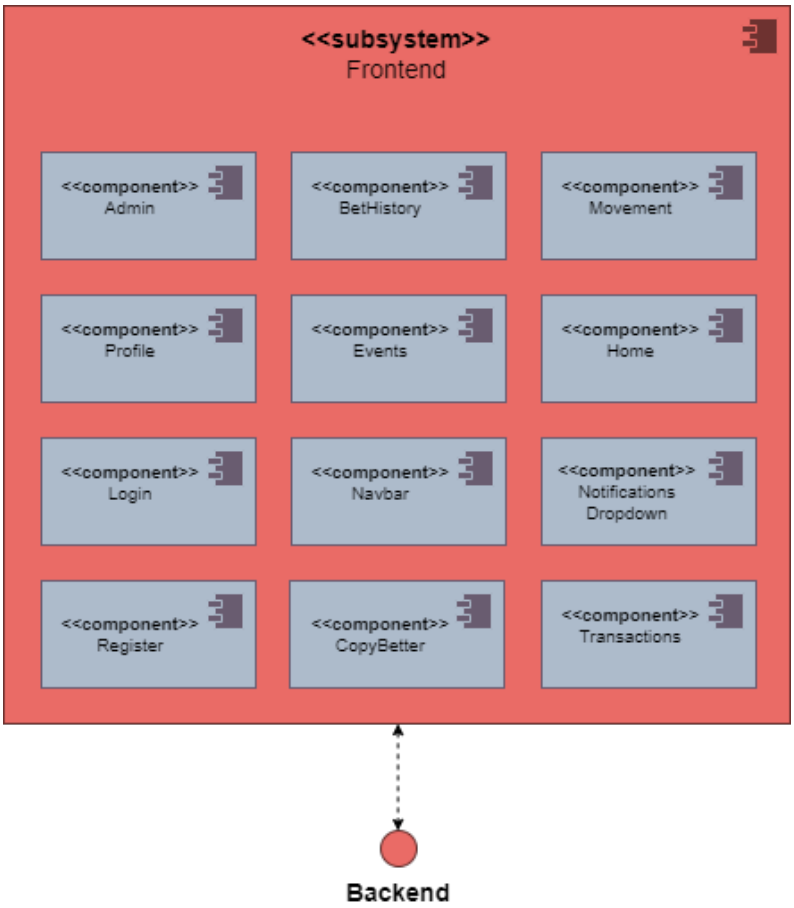

Figura 5: Diagrama de componentes - 2ª Camada - *Frontend*

5.2.2 *Backend*

O *Backend* contém toda a lógica de negócio do sistema, e, como tal, é o subsistema mais denso na aplicação. É possível identificar três tipos de componentes: *models*, componentes do *middleware*, 4.3.2, e *controllers*. Os *models* são representativos dos diferentes tipos de entradas na base de dados, já que identificam os diferentes campos de cada entrada, os tipos desses campos, e outras cláusulas relativamente a estes. Existem *models* para representar utilizadores, promoções, boletins de apostas, apostas, e eventos - *User*, *Promotions*, *Bulletin*, *Bet* e *Event*.
Quanto aos *controllers*, é onde a lógica propriamente dita está contida. É onde são guardadas as funções que afetam a base de dados, e que geram os *outputs* pedidos pelo *Frontend*. Existem cinco, o *Account*, o *Admin*, o *Event* e os dois *Observers*. O *Account*, tal como o nome indica, lida com as contas dos utilizadores, e os seus dados pessoais. Funções como criar, editar ou consultar um utilizador, ou retornar todos os boletins de um utilidor são feitas aqui. De modo homólogo, o *Event* tem o mesmo tipo de funções, mas, desta feita, relativamente aos eventos. Retornar todos os eventos, fechar um eventos, ou atualizá-lo são o tipo de coisas pelas quais este componente é responsável. O *Admin* apenas existe para tornar o código mais limpo e separado, apenas contendo os métodos relativos às funcionalidades do Administrador. Finalmente, os *Observers* e os *adapters*, representam componentes mais específicos a tratar na camada posterior.
Por terem acesso tanto ao pedido como à resposta, os componentes do *middleware* servem de ponte na comunicação entre o Frontend e o Backend, que é feito através de pedidos `HTTP`, get e post. Quanto a este sistema, estão presentes quatro componentes do *middleware*, divididos em duas camadas. Na de baixo encontram-se os componentes do *middleware* associados aos *controllers*. Estes reencaminham os tráfegos relativos aos dados de utilizador, administradores, eventos e mecanismos relacionados com os observers, respetivamente. Para este efeito, utilizam o componente do *middleware* da camada de cima, o *Router*.

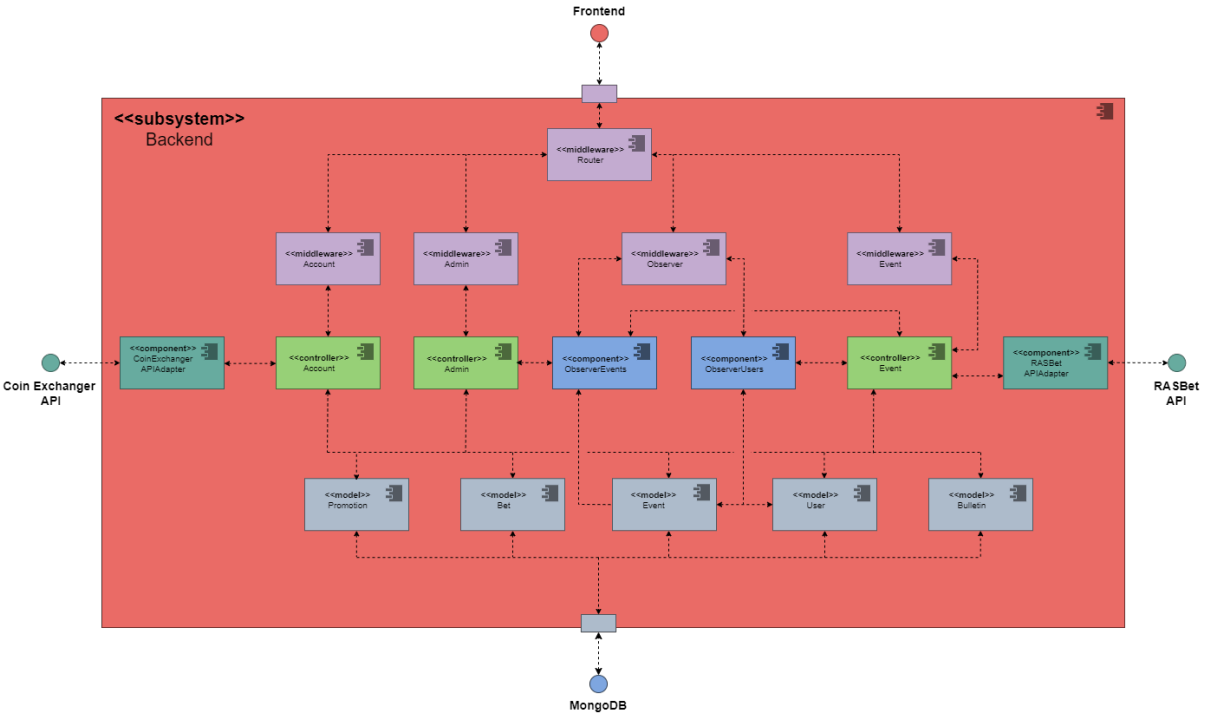

Figura 6: Diagrama de componentes - 2ª Camada - *Backend*

5.3 Modelação estrutural - 3ª Camada

No que diz respeito à terceira e última camada de abstração, existem quatro vertentes a deliminar primeiro: a primeira está diretamente relacionada com a representação dos dois últimos padrões mencionados em 4.3.
O recurso ao padrão *Observer*, 4.3.3, dá-se não só no âmbito das funcionalidades de seguir entidades, sejam eventos ou outros utilizadores para cópia de apostas, mas também na representação de notificações com mudanças de estados de eventos; assim os componentes em 7 representados a cor azul identificam-se como dois *controllers* que tratam da subscrição de *subjects* - *models* User e User/Event para *ObserverUsers* e *ObserverEvents*, respetivamente, para eventuais notficações para os casos realçados previamente.
Em conformidade com o padrão *Adapter*, é de salientar a sua utilidade no futuro do sistema, uma vez que, surgirão oportunidades para a inserção de novas API's, o que apenas implica a adição do respetivo adaptador, dado que o *Backend* já está preparado apenas para receber um inventário fixo de dados de cada um deles. Como já mencionado em 4.3.4, o objetivo é que a arquitetura seja fixa, e seja possível adicionar elementos com um número mínimo de alterações, e com a adição deste padrão apenas é necessário chamar um novo método que realize a transição entre o novo adaptador e o *Backend* e devolve os dados necessários da forma correta. Em contraste, surge ainda um ponto crucial, no que toca à capacidade do sistema e na sua flexibilidade; ora, a utilização de atributos que não são obrigatórios nos *models* - atributos que não estão representados com - permite utilizar o *model Event* para qualquer tipo de desporto, dado que as suas tabela na base de dados não criam os atributos se estes não forem passados na sua criação. Neste momento, apenas desportos do tipo coletivo com empates estão a ser utilizados, mas o arquitetura existente suportas os outros tipos, apenas com a adição das respetivas *API's* e verficações funcionais. Todos os outros componentes presentes, já foram realçados previamente em 5 e 6.

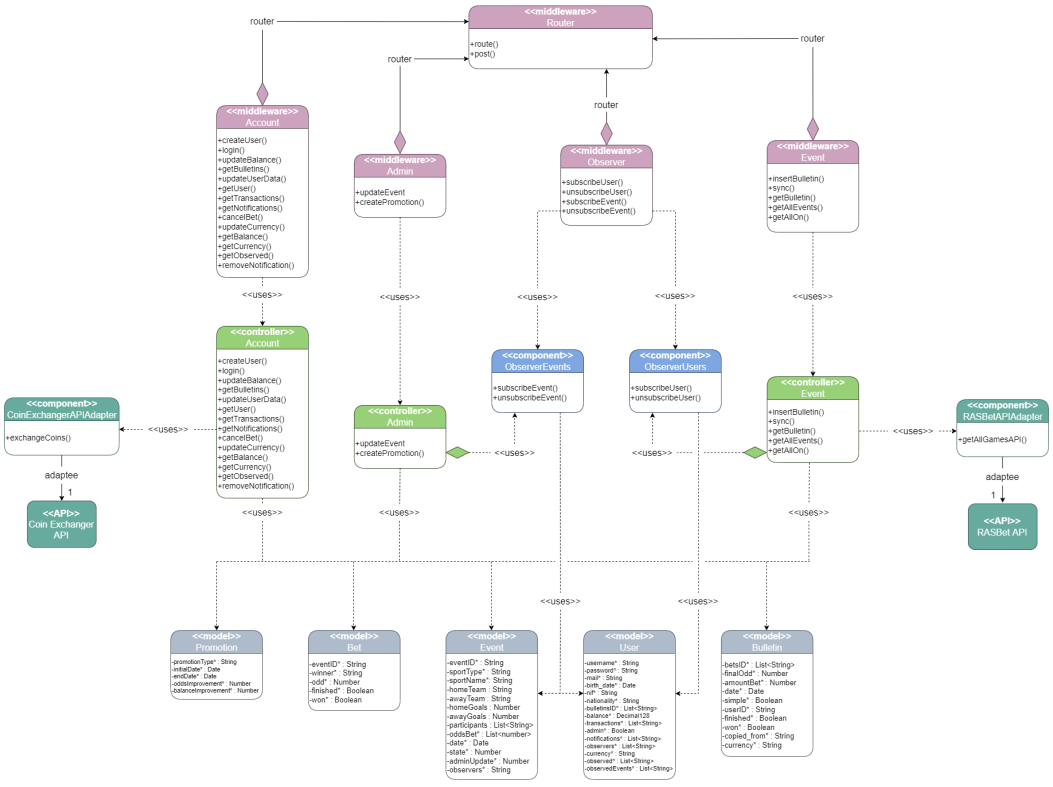

Figura 7: Diagrama de componentes - 3ª Camada - *Backend*

6 Modelação comportamental

6.1 Interação das funcionalidades com os componentes

Método	Use Case(s)	Componente do *Backend* vinculado
createUser	Registo	Account
login	Login	
updateBalance	Movimentar conta	
getBulletins	Consultar histórico de apostas	
updateUserData getUser	Editar Perfil	
getTransactions	Consultar histórico de transações	
removeNotification getNotifications	Consultar notificações	
cancelBet	Cancelar aposta	
getBalance getCurrency updateCurrency	Alterar Moeda e Gerar taxa de conversão de moeda	
getObserved subscribeUser unsubscribeUser	Seguir apostador	ObserverUsers
subscribeEvent unsubscribeEvent	Seguir evento	ObserverEvents
createPromotion	Criar promoção	Admin
updateEvent	Alterar estado do evento	
insertBulletin getBulletin	Efetuar aposta e Copiar aposta	Event
sync	Gerar eventos	
getAllEvents getAllsOn	Consultar eventos e Consultar desportos	

Tabela 7: Levantamento de métodos

Após a identificação dos métodos e os correspondentes componentes do *Backend*, 7, segue o pipeline de interações desde que as funcionalidades são chamados até a sua função estar finalizada; ora, para este efeito, é necessário padronizar os diferentes componentes mediante acrónimos simplistas de forma a integrarem todos a tabela 8.

Frontend linked component - FLC

- Admin - A
- BetHistory - BH
- CopyBetter - CB
- Movement - M
- Profile - P
- Events - Es
- Home - H

- Login - L
- Navbar - Nb
- NotficationsDropdown - ND
- Register - R
- Transactions - T

Backend linked component - BLC

- Account - Acc
- ObserverUsers - OU
- ObserverEvents - OE
- Admin - Adm
- Event - E
- RASBet API Adapter - RAA
- Coin Exchanger API Adapter - CEAA

API

- RASBet API - RA
- Coin Exchanger API - CEA

Outros

- MongoDB - MDB
- User Interface - UI

Use Case	UI	FLC	BLC	API	MDB
Consultar histórico de apostas	×	BH	Acc		×
Cancelar Aposta	×	BH	Acc		×
Alterar Moeda	×	Nb	Acc		×
Efetuar aposta	×	Es	E		×
Editar Perfil	×	P	Acc		×
Registo	×	R	Acc		×
Login	×	L	Acc		×
Consultar notificações	×	ND	Acc		×
Copiar aposta	×	ND	Acc		×
Seguir apostador	×	CB	OU		×
Consultar desportos	×	Es	E		×
Consultar eventos	×	Es	E		×
Seguir evento	×	Es	OE		×
Movimentar conta	×	M	Acc		×
Consultar histórico de transações	×	T	Acc		×
Alterar estado do evento	×	A	Adm		×
Criar promoção	×	A	Adm		×
Gerar eventos			E, RAA	RA	×
Gerar taxa de conversão de moeda			Acc, CEAA	CEA	×

Tabela 8: Matriz de componentes

6.2 Diagramas de sequência

No âmbito da representação comportamental dos métodos levantados, é necessário, além do respetivo diagrama, de identificar as tarefas desempenhadas pelo mesmo; assim sendo seguem, para cada um dos métodos associados às funcionalidades - *Use cases* - um diagrama de sequência e o respetivo pipeline de tarefas a desempenhar e as restrições em tomar em consideração.

6.2.1 createUser

1. Estabelecimento dos campos de registo.
2. Confirmação dos campos e verificação dos respetivos. - **Passo 1**
3. Se existirem promoções para novos Apostadores, atualizar o saldo da nova conta. - **Passo 1.4.3**
4. Criar o novo utilizador na base de dados. - **Passo 1.4.5**

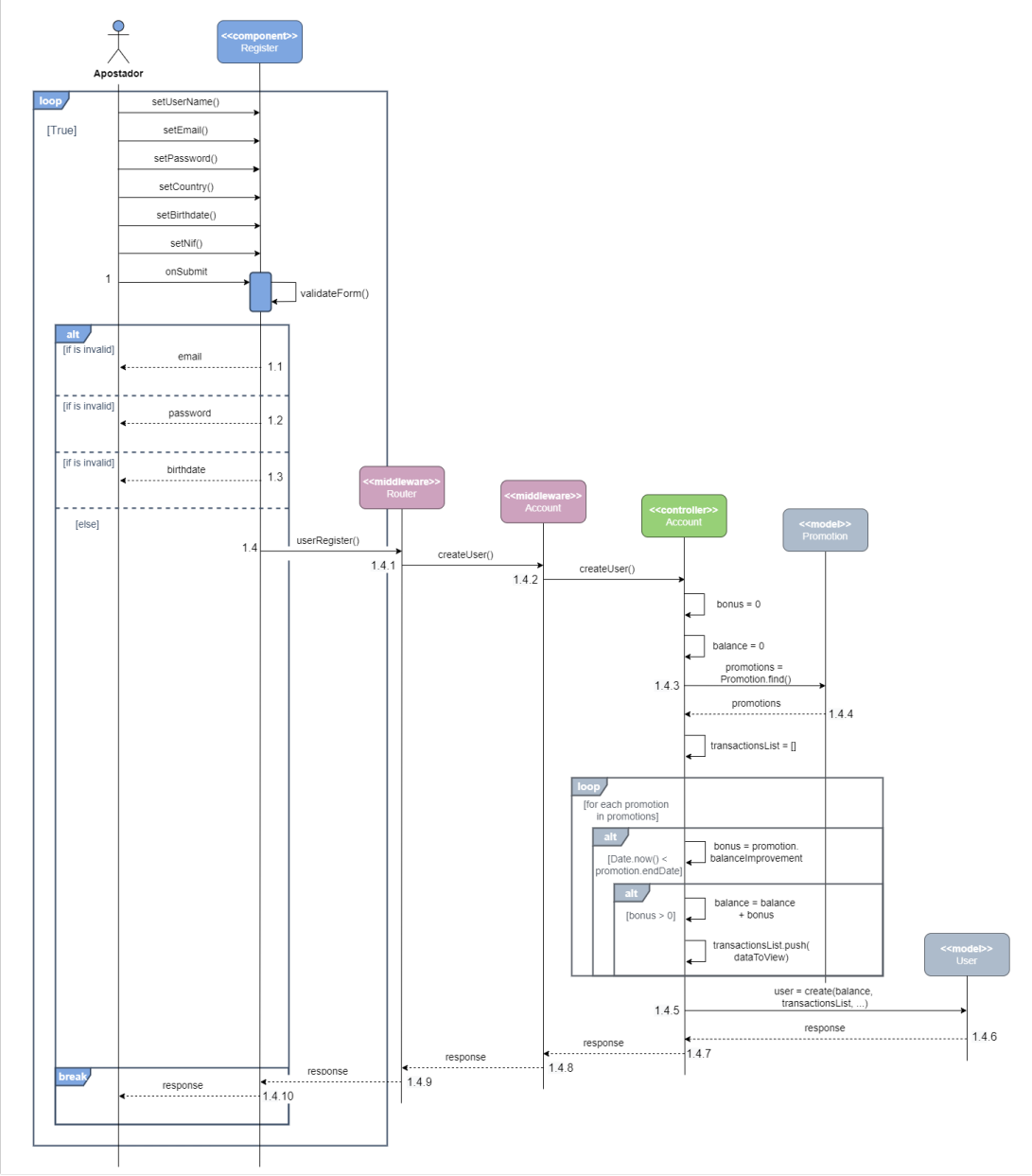

Figura 8: Diagrama de sequência - createUser

6.2.2 login

1. Estabelecimento dos campos de login.
2. Confirmação dos campos e verificação dos respetivos. - **Passo 1**
3. Procurar o utilizador na base de dados via email. - **Passo 1.3.3**
4. Se o utilizador não existir o sistema avisa que o email está errado. - **Passo 1.3.5.1**
5. Se a palavra estiver incorreta o sistema avisa acerca da falha. - **Passo 1.3.5.2**
6. Se o utilizador existir no sistema e for Administrador, procede ao login de Administrador com sucesso. - **Passo 1.3.5.3**
7. Se o utilizador existir no sistema e não for Administrador procede ao login com sucesso. - **Passo 1.3.5.4**

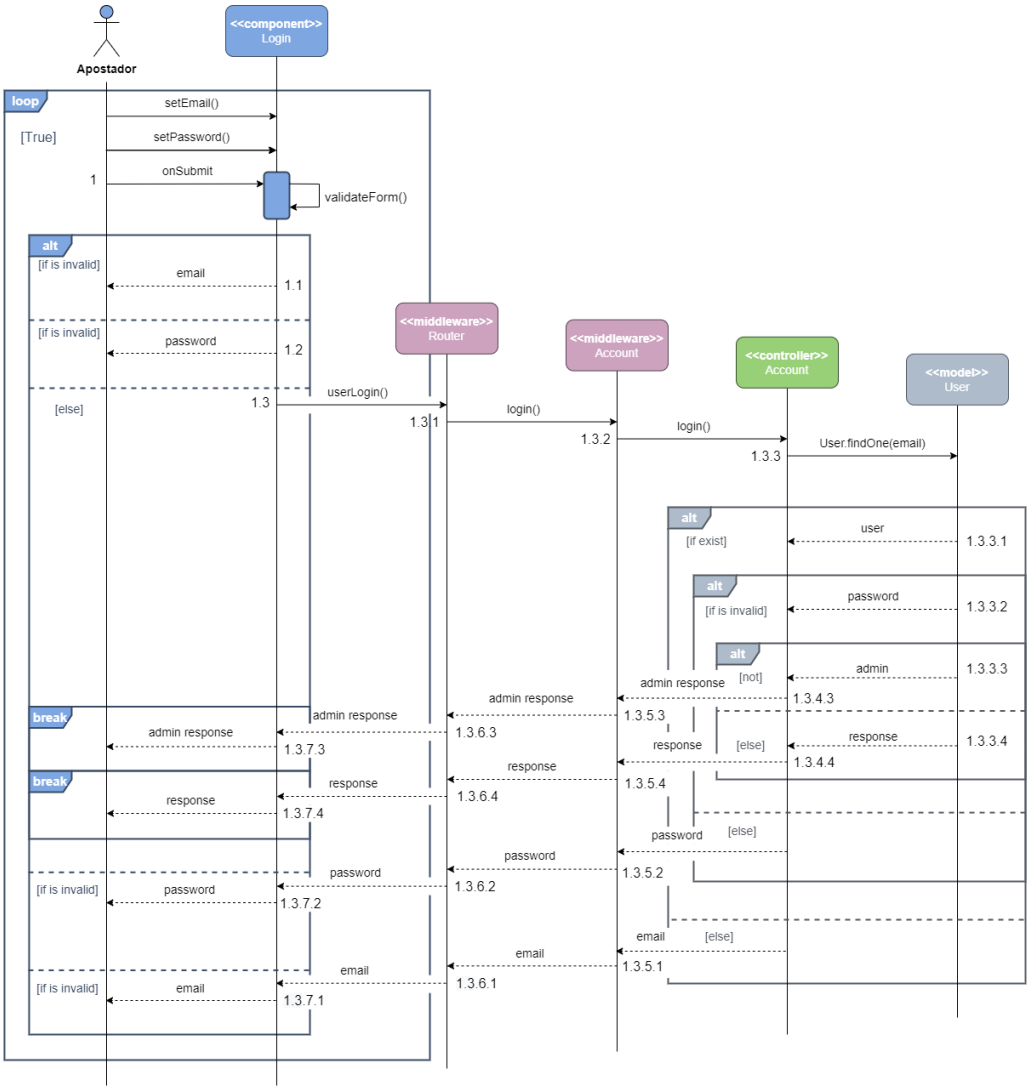

Figura 9: Diagrama de sequência - login

6.2.3 updateBalance

1. Estabelecimento da modalidade de movimento (depósito ou levantamento).
2. Estabelecimento dos campos necessários ao movimento.
3. Confirmação dos campos. - **Passo 1**
4. Segue a atualização do saldo do utilizador, que é realizada de forma similar, nas duas modalidades, uma vez que a lógica de escolha é feita em *Frontend*.
5. Procurar o utilizador na base de dados via identificador. - **Passo 1.1.3**
6. Verificar a modalidade e atualizar a lista de transações do utilizador conforme a correspondente.
7. Verificar se o valor a atualizar é válido, e se for, segue-se a salvaguarda do respetivo utilizador na base de dados com o novo saldo confirmando o movimento. - **Passo 1.1.4.1**

8. Na eventualidade de um movimento inválido, alertar o utilizador para o efeito. - **Passo 1.1.4.2**

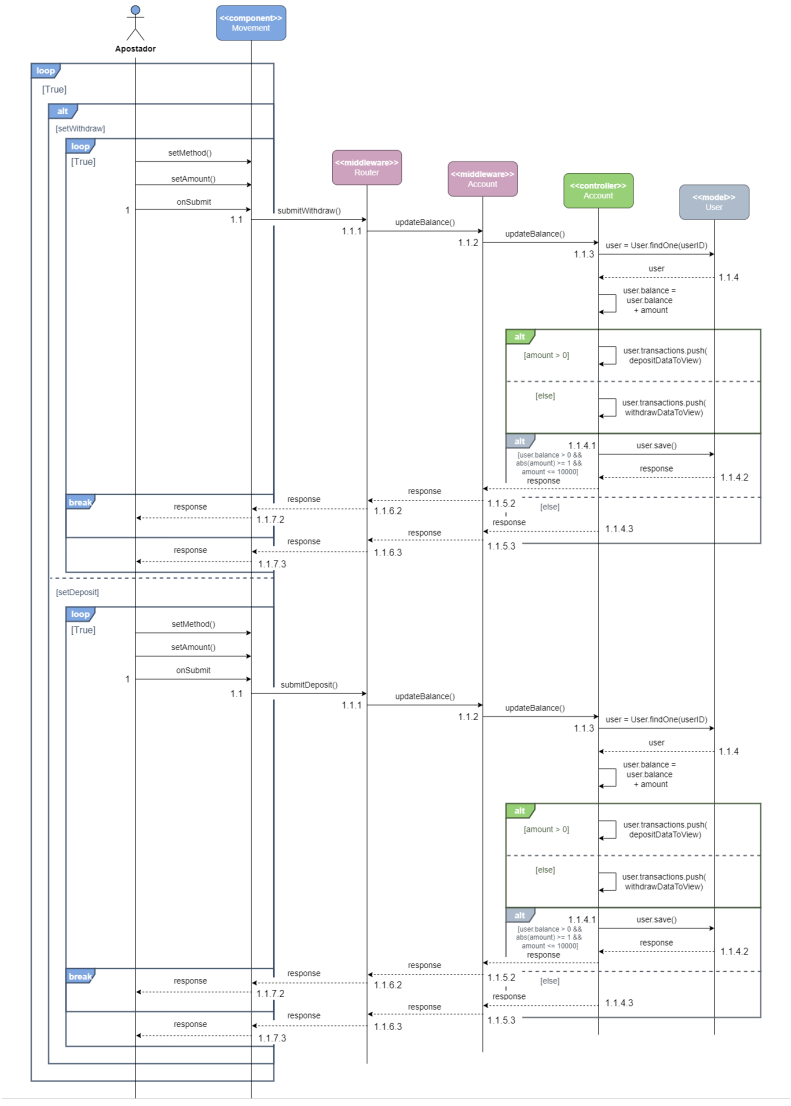

Figura 10: Diagrama de sequência - updateBalance

6.2.4 getBulletins

1. Procurar o utilizador na base de dados via identificador. - **Passo 5**
2. Procurar na base de dados os boletins associados ao utilizador em questão.
3. Procurar na base de dados os eventos associados a cada uma das apostas dos boletins do utilizador.
4. Criar o objeto a enviar para o *Frontend* com os dados a visualizar.
5. Enviar o histórico de apostas sob a forma definida. - **Passo 6**

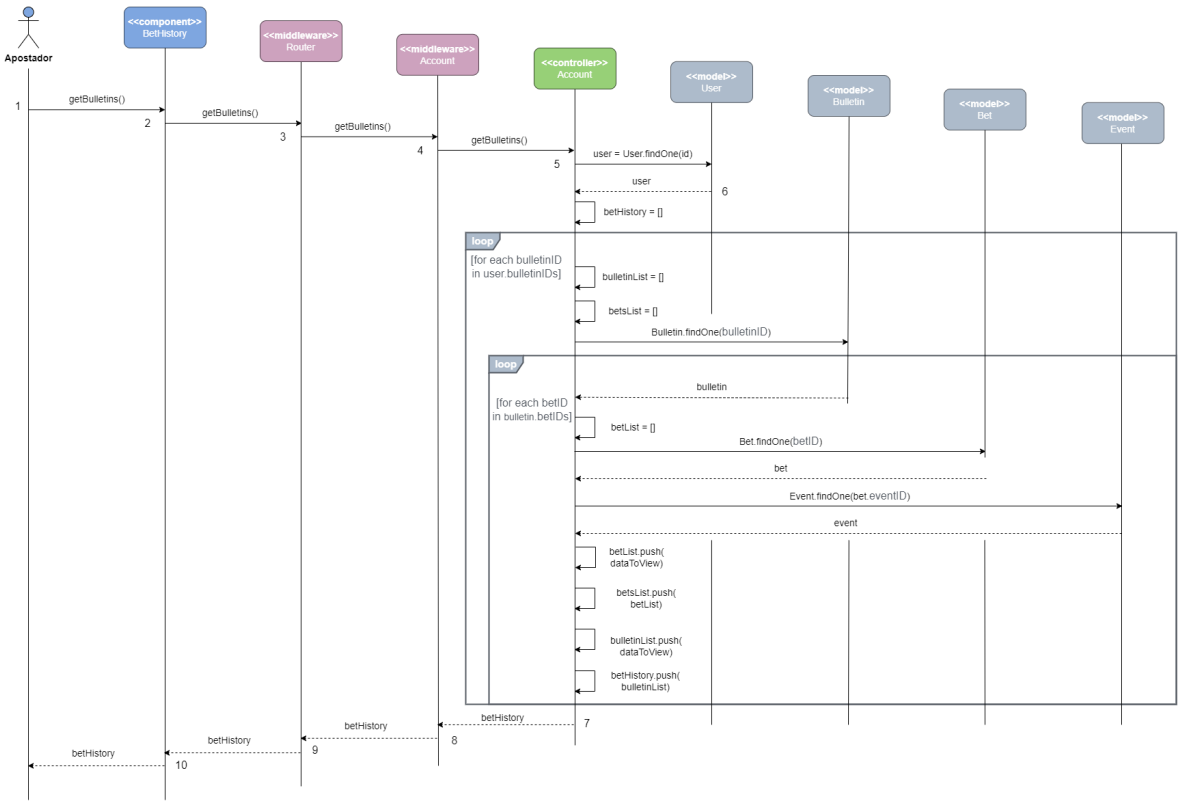

Figura 11: Diagrama de sequência - getBulletins

6.2.5 updateUserData

1. Estabelecimento dos campos a modificar - a confirmação da palavra-passe é realizada em *Frontend* através do método *getUser* e não é necessária ao contexto.
2. Confirmação dos campos e verificação dos respetivos. - **Passo 1**
3. Se o email novo não for nulo, procurar o utilizador na base de dados e atualizar o seu email salvaguardando-se na base de dados. - **Passo 1.6.5**
4. Se a palavra-passe nova não for nula, procurar o utilizador na base de dados e atualizar a sua palavra-passe salvaguardando-se na base de dados. - **Passo 1.6.10**

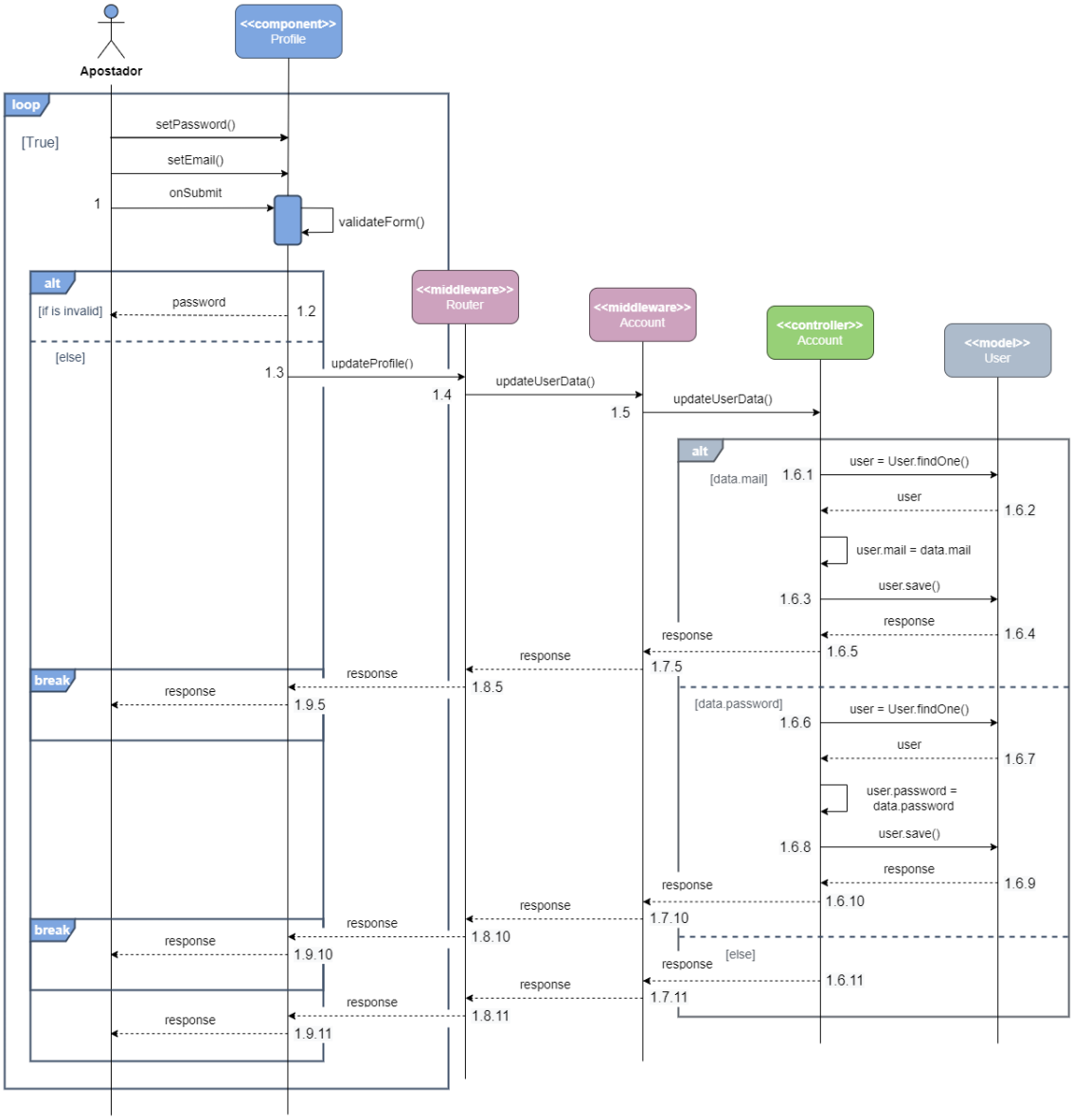

Figura 12: Diagrama de sequência - updateUserData

6.2.6 getUser

1. Procurar o utilizador na base de dados via identificador. - **Passo 5**
2. Criar o objeto a enviar para o *Frontend* com os dados a visualizar.
3. Enviar os dados do utilizador sob a forma definida. - **Passo 7**

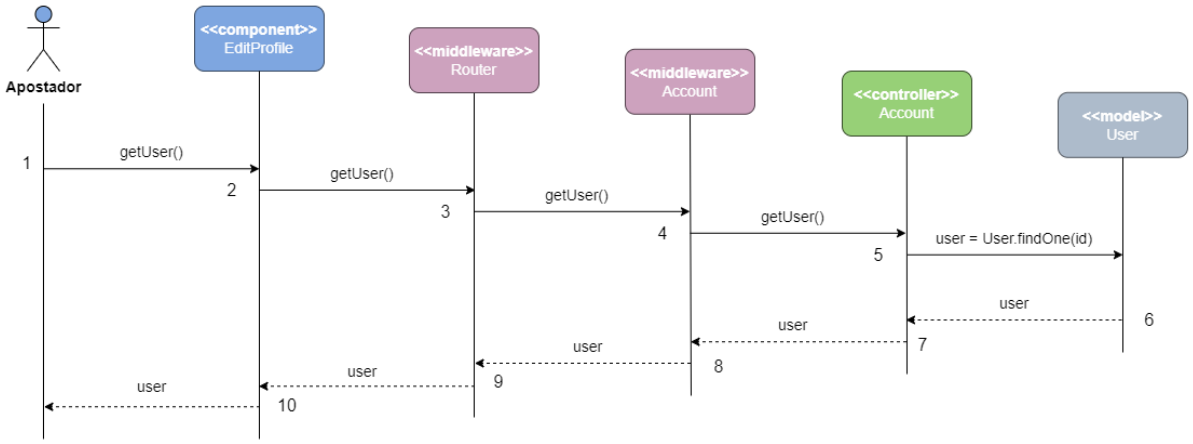

Figura 13: Diagrama de sequência - getUser

6.2.7 getTransactions

1. Procurar o utilizador na base de dados via identificador. - **Passo 5**
2. Criar o objeto a enviar para o *Frontend* com os dados a visualizar.
3. Enviar a lista de transações sob a forma definida. - **Passo 7**

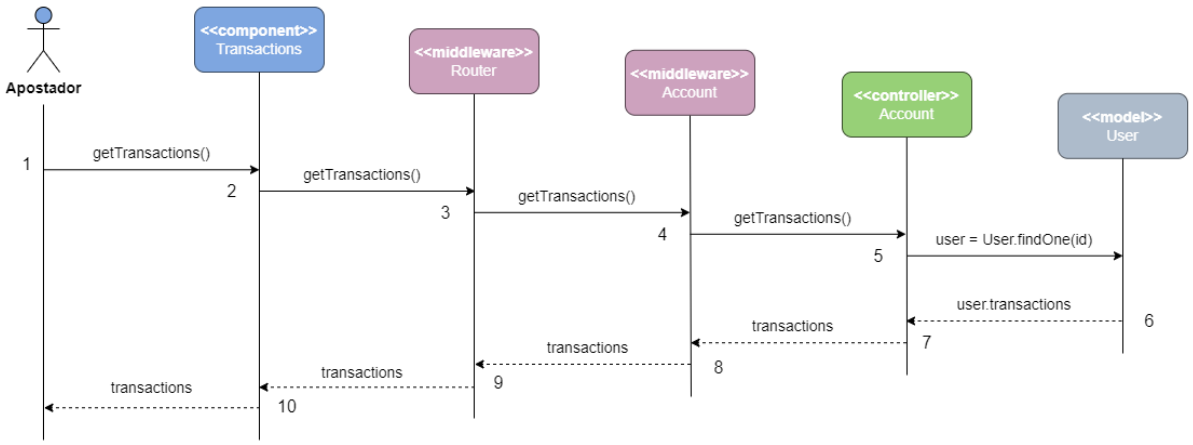

Figura 14: Diagrama de sequência - getTransactions

6.2.8 getNotifications

1. Procurar o utilizador na base de dados via identificador. - **Passo 5**
2. Criar o objeto a enviar para o *Frontend* com os dados a visualizar.
3. Enviar a lista de notificações sob a forma definida. - **Passo 7**

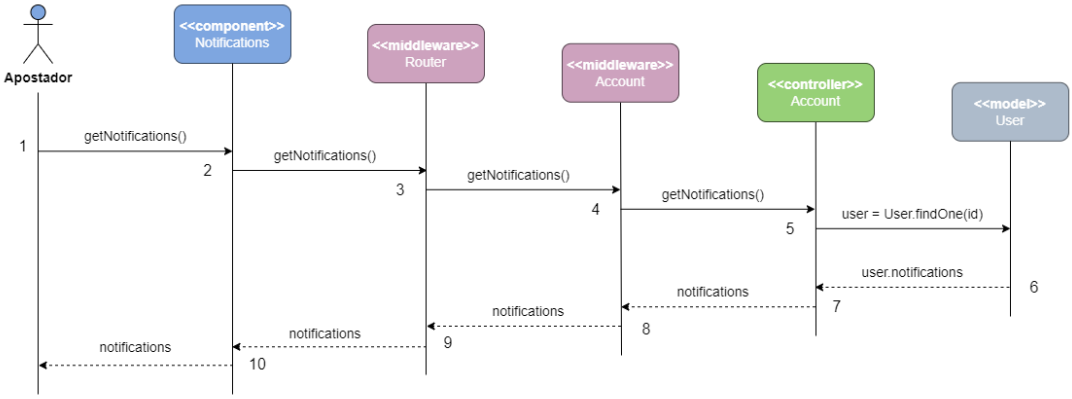

Figura 15: Diagrama de sequência - getNotifications

6.2.9 removeNotification

1. Procurar o utilizador na base de dados via identificador. - **Passo 5**
2. Remover a notificação da lista de notificações do utilizador e salvaguardar o respetivo na base dados. - **Passo 7**

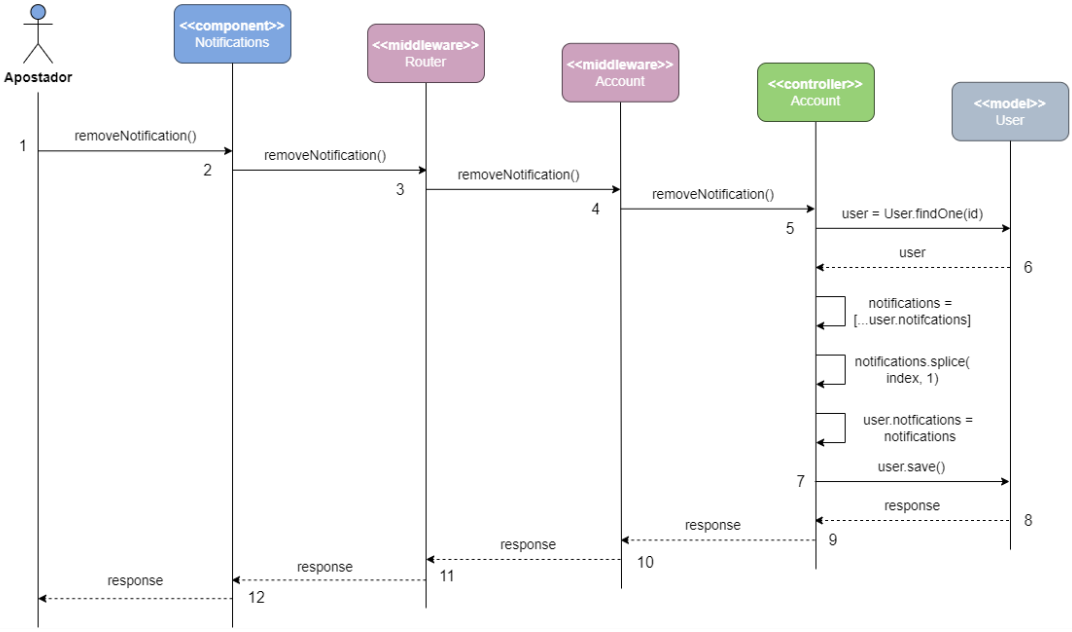

Figura 16: Diagrama de sequência - removeNotification

6.2.10 cancelBet

1. Procurar o boletim na base de dados via identificador. - **Passo 5**
2. Procurar na bases de dados as apostas associadas ao *Bulletin*.
3. Verifcar se alguma das apostas já acabou ou ainda está a acabar, e caso isso aconteça, alertar para a impossibilidade de cancelar a aposta. - **Passo 6.6**
4. Caso todas as apostas sejam aptas para cancelamento, retirar os eventos associados às mesmas da lista dos *ObserverEvents* do utilizador.
5. Procurar na bases de dados o utilizador associado ao *Bulletin*. - **Passo 6.1**
6. Atualizar a sua lista de transações, a sua lista de boletins e o seu saldo, uma vez que a aposta será cancelada, seguindo-se da salvaguarda do respetivo utilizador na base de dados. - **Passo 6.3**

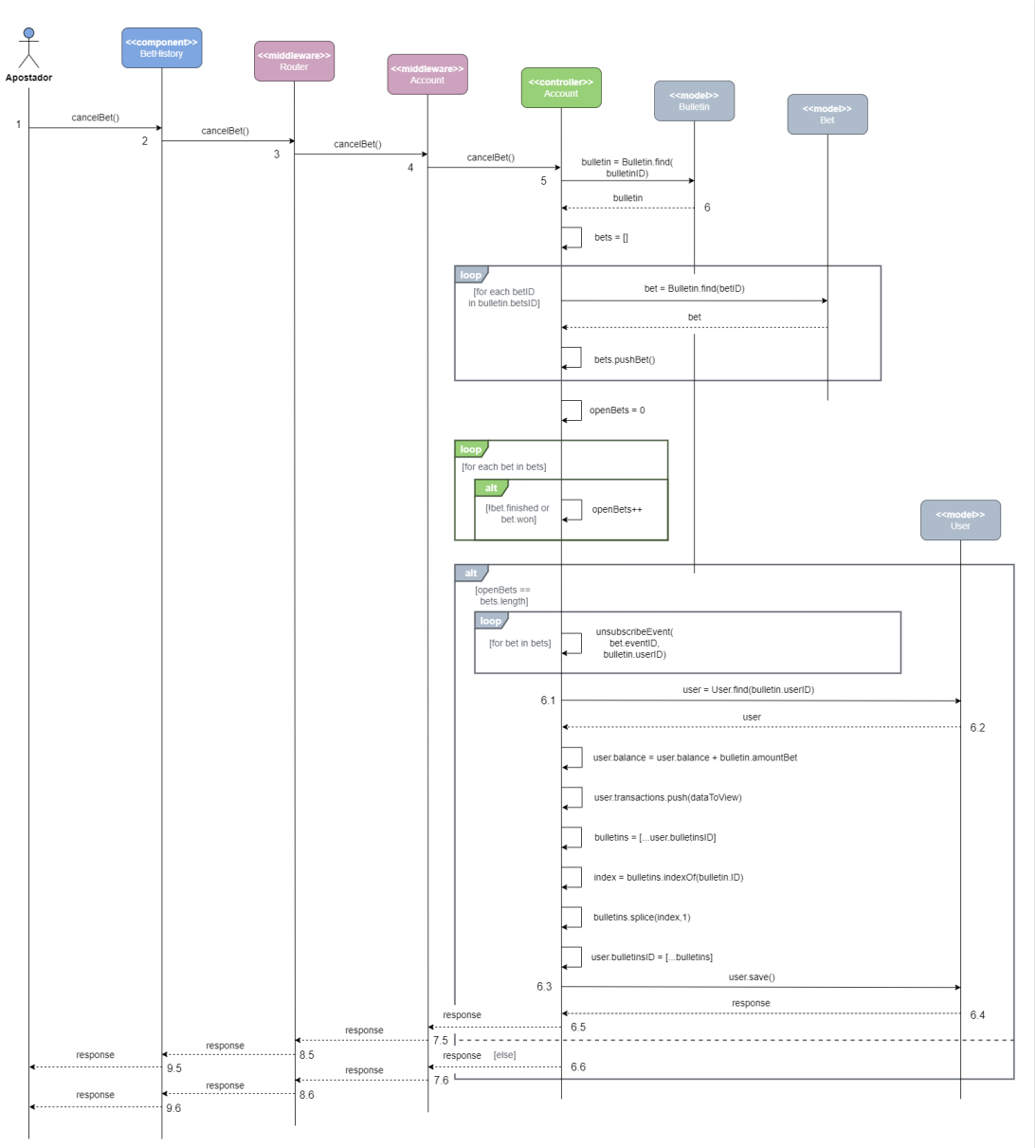

Figura 17: Diagrama de sequência - cancelBet

6.2.11 getBalance

1. Procurar o utilizador na base de dados via identificador. - **Passo 5**
2. Criar o objeto a enviar para o *Frontend* com os dados a visualizar.
3. Enviar o saldo sob a forma definida. - **Passo 7**

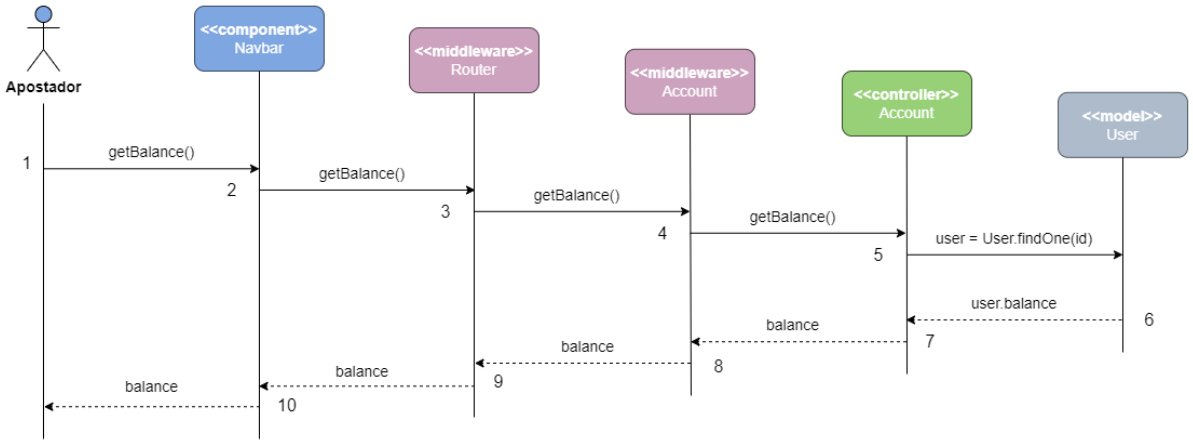

Figura 18: Diagrama de sequência - getBalance

6.2.12 getCurrency

1. Procurar o utilizador na base de dados via identificador. - **Passo 5**
2. Criar o objeto a enviar para o *Frontend* com os dados a visualizar.
3. Enviar a moeda sob a forma definida. - **Passo 7**
4. Atualizar a moeda na *Navbar*. - **Passo 11**

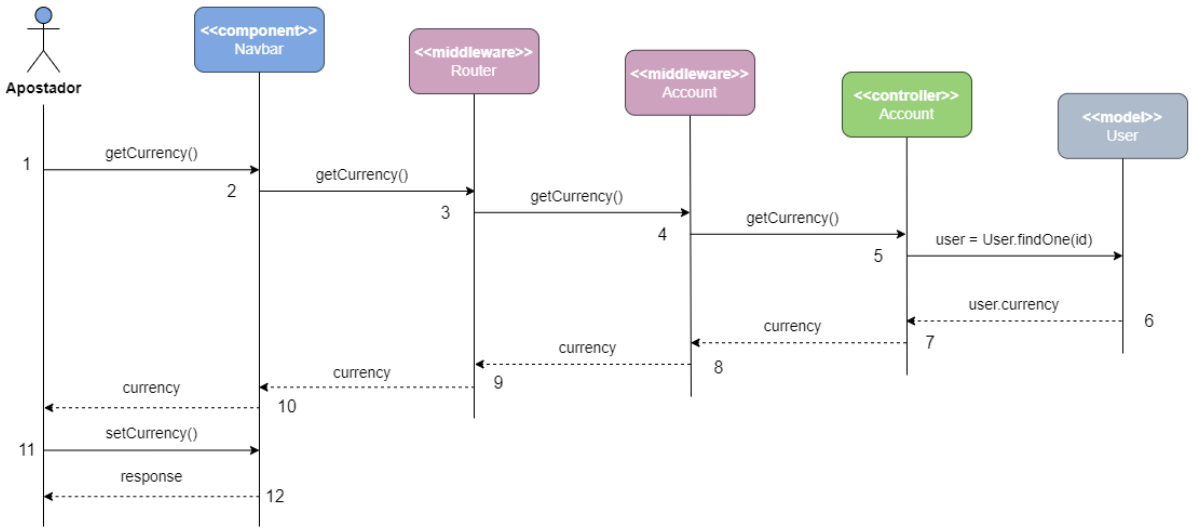

Figura 19: Diagrama de sequência - getCurrency

6.2.13 updateCurrency

1. Procurar o utilizador que foi seguido na base de dados via identificador, se a moeda selecionada não for nula. - **Passo 5.1**
2. Calcular o novo saldo do utilizador através da chamada do método auxiliar *exchangeCoins* do componente *CoinExchangerAPIAdapter*.
3. Fazer um pedido à CoinExchangerAPI com as moedas para conversão e o valor a converter. **Passo 2 - *exchangeCoins Aux method***
4. Atualizar o saldo do utilizador, a sua lista de transações e a moeda com que atua, seguindo-se da sua salvaguarda na base de dados. - **Passo 5.3**

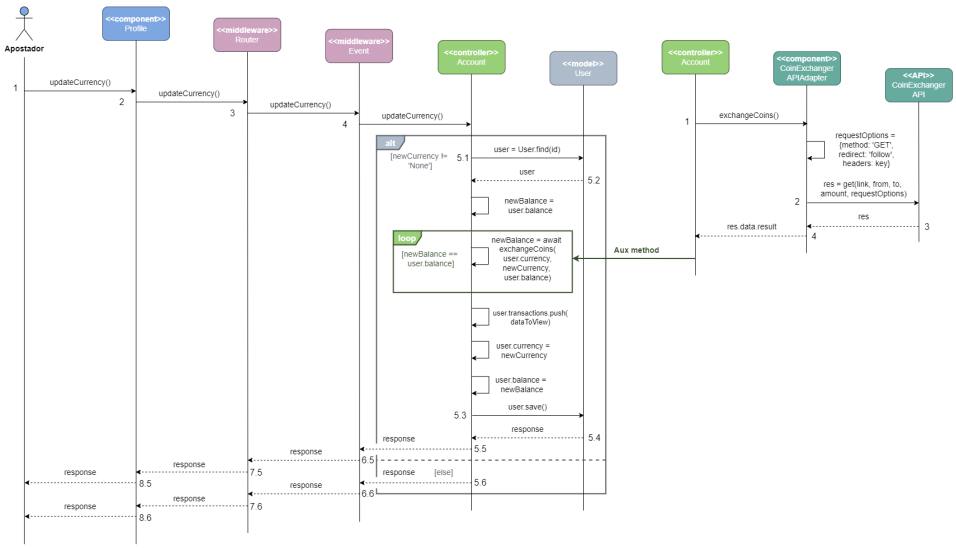

Figura 20: Diagrama de sequência - updateCurrency

6.2.14 getObserved

1. Procurar o utilizador que foi seguido na base de dados via identificador. - **Passo 5**
2. Criar o objeto a enviar para o *Frontend* com os dados a visualizar.
3. Enviar os dados do utilizador que foi seguido sob a forma definida. - **Passo 7**

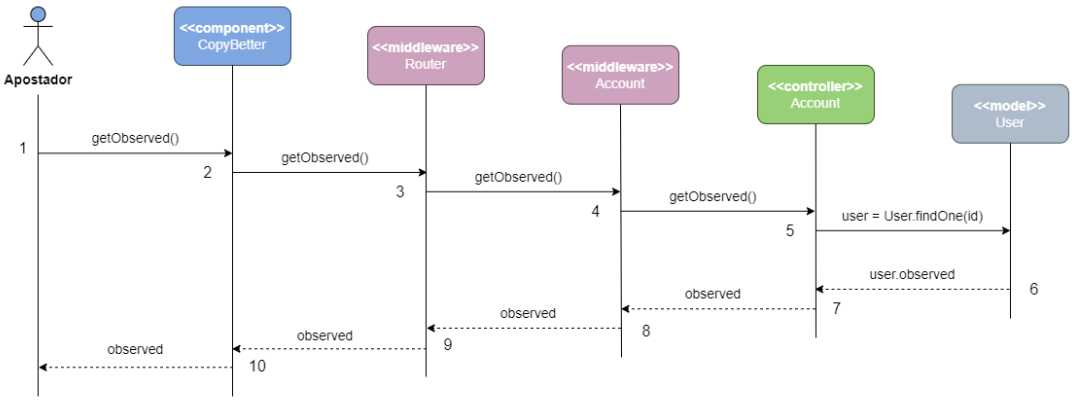

Figura 21: Diagrama de sequência - getObserved

6.2.15 subscribeUser

1. Procurar o utilizador a seguir na base de dados via nome de utilizador. - **Passo 5**
2. Se o utilizador a seguir não existir no sistema, alertar para a falha. - **Passo 5.4**
3. Procurar o utilizador que segue na base de dados via identificador. - **Passo 5.2**
4. Verificar se o utilizador já segue o utilizador que pretende seguir, caso isso aconteça, alertar para o efeito. - **Passo 5.3.1**
5. Se ainda não seguir esse utilizador, atualizar a lista de utilizadores que segue, e por sua vez, atualizar a lista de seguidores do outro utilizador, salvaguardando ambos na base de dados. - **Passos 5.3.2 e 5.3.4**

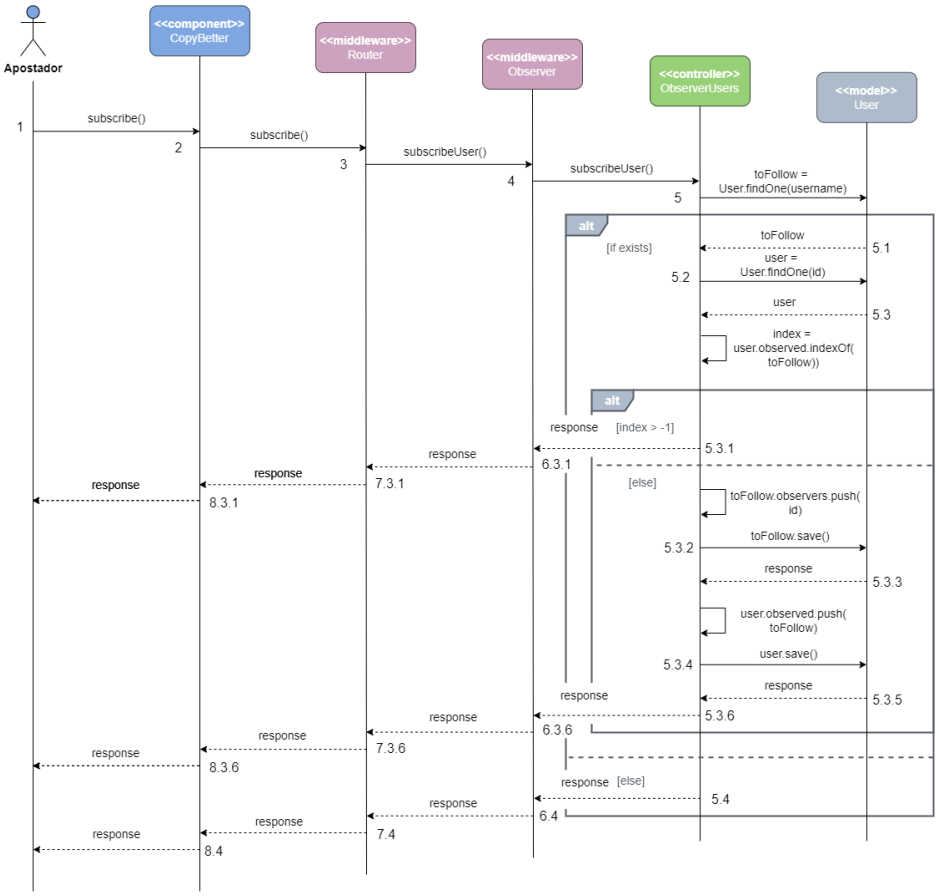

Figura 22: Diagrama de sequência - subscribeUser

6.2.16 unsubscribeUser

1. Procurar o utilizador a deixar de seguir na base de dados via nome de utilizador. - **Passo 5**
2. Se o utilizador a deixar de seguir não existir no sistema, alertar para a falha. - **Passo 5.4**
3. Verificar se o utilizador a deixar de seguir é seguido pelo utilizador que o pretende seguir, caso isso aconteça, remove-lo da lista de seguidores do outro utilizador e salvaguarda-lo na base dados. - **Passo 5.1.1**
4. Procurar o utilizador na base de dados via identificador. - **Passo 5.2**
5. Verificar se o utilizador já segue o utilizador que pretende seguir, caso isso aconteça, remover o outro utilizador da lista de utilizadores a seguir e salvaguardar o utilizador que pretende seguir na base dados. - **Passo 5.3.1**
6. Se não for possível deixar de seguir o utilizador, alertar para o efeito. - **Passo 5.3.4**

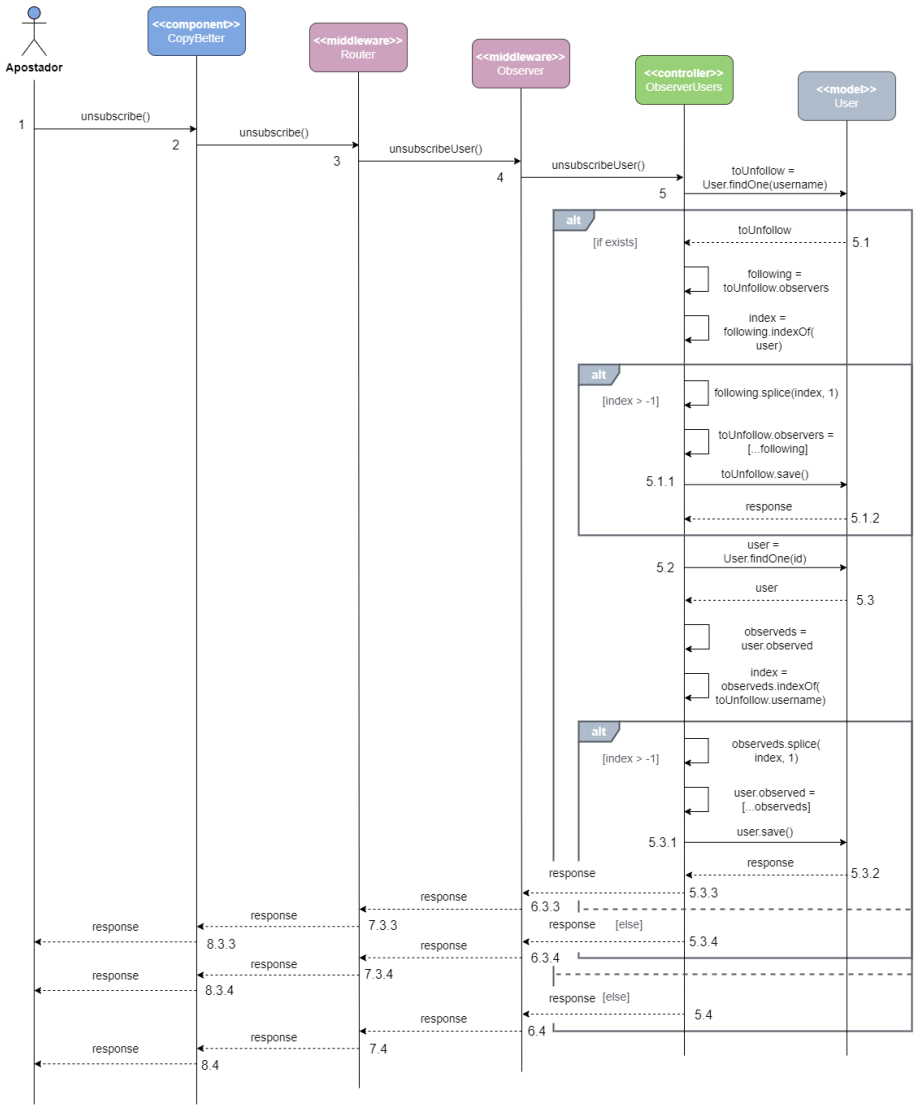

Figura 23: Diagrama de sequência - unsubscribeUser

6.2.17 subscribeEvent

1. Procurar o evento a seguir na base de dados via identificador. - **Passo 5**
2. Verificar se o evento não é seguido pelo utilizador que o pretende seguir, caso isso aconteça, atualizar a lista de utilizadores que o seguem e salvaguarda-lo na base de dados. - **Passo 6.2**
3. Procurar o utilizador que pretende seguir o evento na base de dados via identificador. - **Passo 6.3**
4. Atualizar a lista de eventos a seguir do utilizador com o novo evento e salvaguardar o utilizador na base dados. - **Passos 6.5**

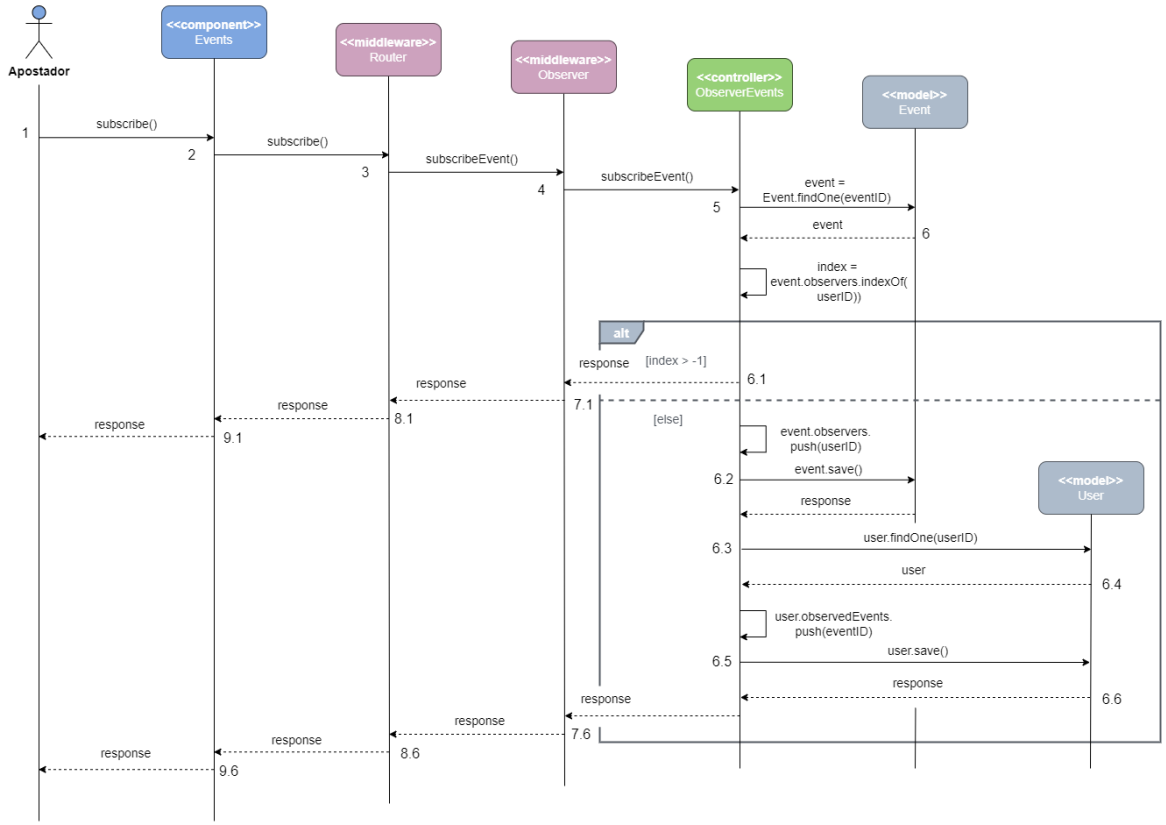

Figura 24: Diagrama de sequência - subscribeEvent

6.2.18 unsubscribeEvent

1. Procurar o evento a deixar de seguir na base de dados via identificador. - **Passo 5**
2. Verificar se o evento é seguido pelo utilizador que o pretende seguir, caso isso aconteça, atualizar a lista de utilizadores que o seguem e salvaguarda-lo na base de dados. - **Passo 6.1**
3. Procurar o utilizador que pretende seguir o evento na base de dados via identificador. - **Passo 6.3**
4. Remover o evento da lista de eventos a seguir do utilizador, e salvaguardar o utilizador na base dados. - **Passos 6.5**

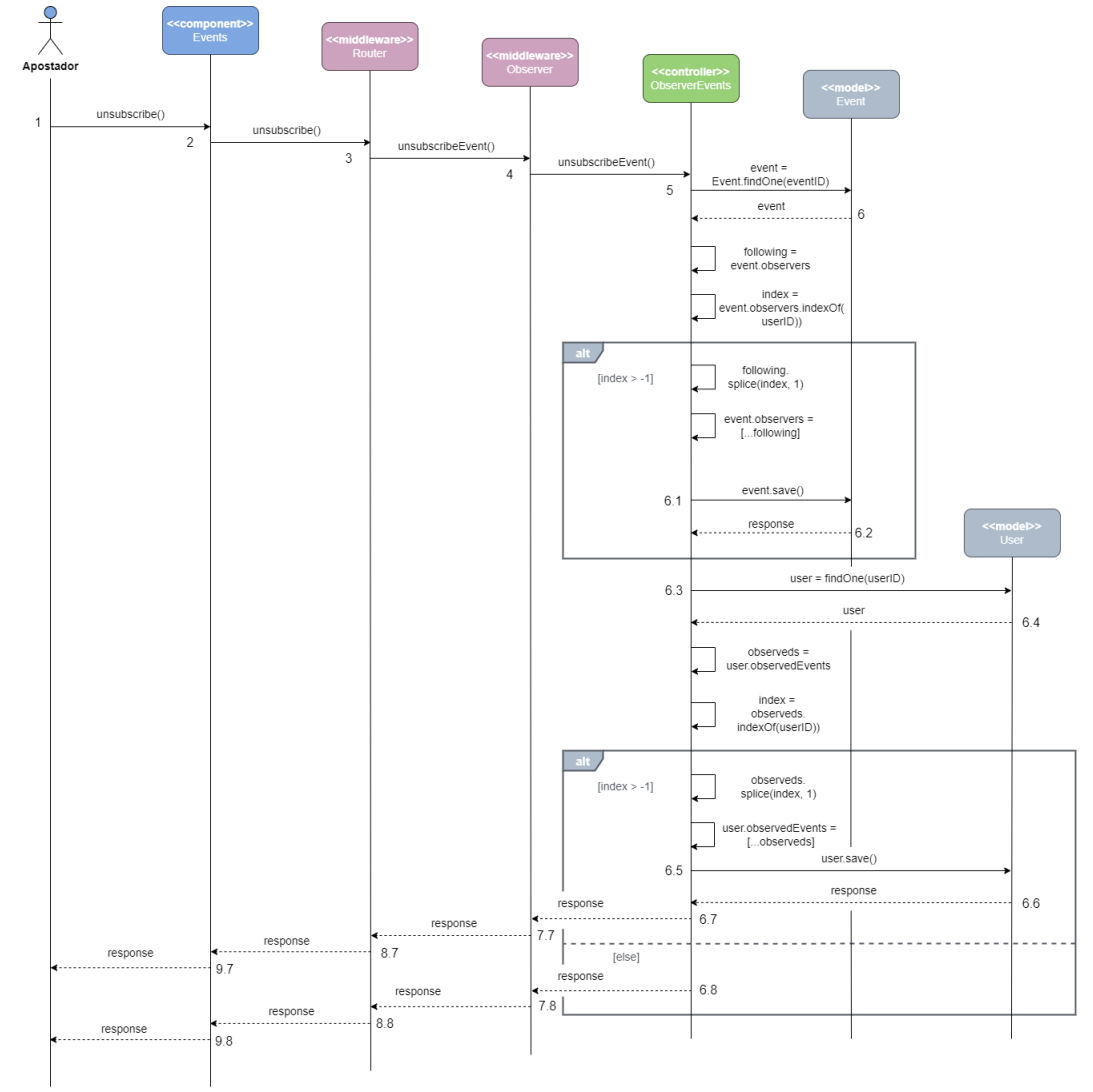

Figura 25: Diagrama de sequência - unsubscribeEvent

6.2.19 createPromotion

1. Procurar e verificar se já exite alguma promoção do tipo definido e no intervalo de tempo introduzido, e caso isso aconteça, alertar para a impossibilidade de criar a promoção solicitada. - **Passo 5**
2. Criar a nova promoção na base de dados. - **Passo 6.1**
3. Verificar se o tipo de promoção é o de acréscimo de saldo a novos Apostadores.
4. Procurar o utilizador todos os utilizadore na base de dados. - **Passo 6.2**
5. Filtrar os utilizadore que não têm nenhuma aposta realizada e atualizar o seu saldo com o novo valor, salvaguardando os respetivos utilizadores na base de dados.

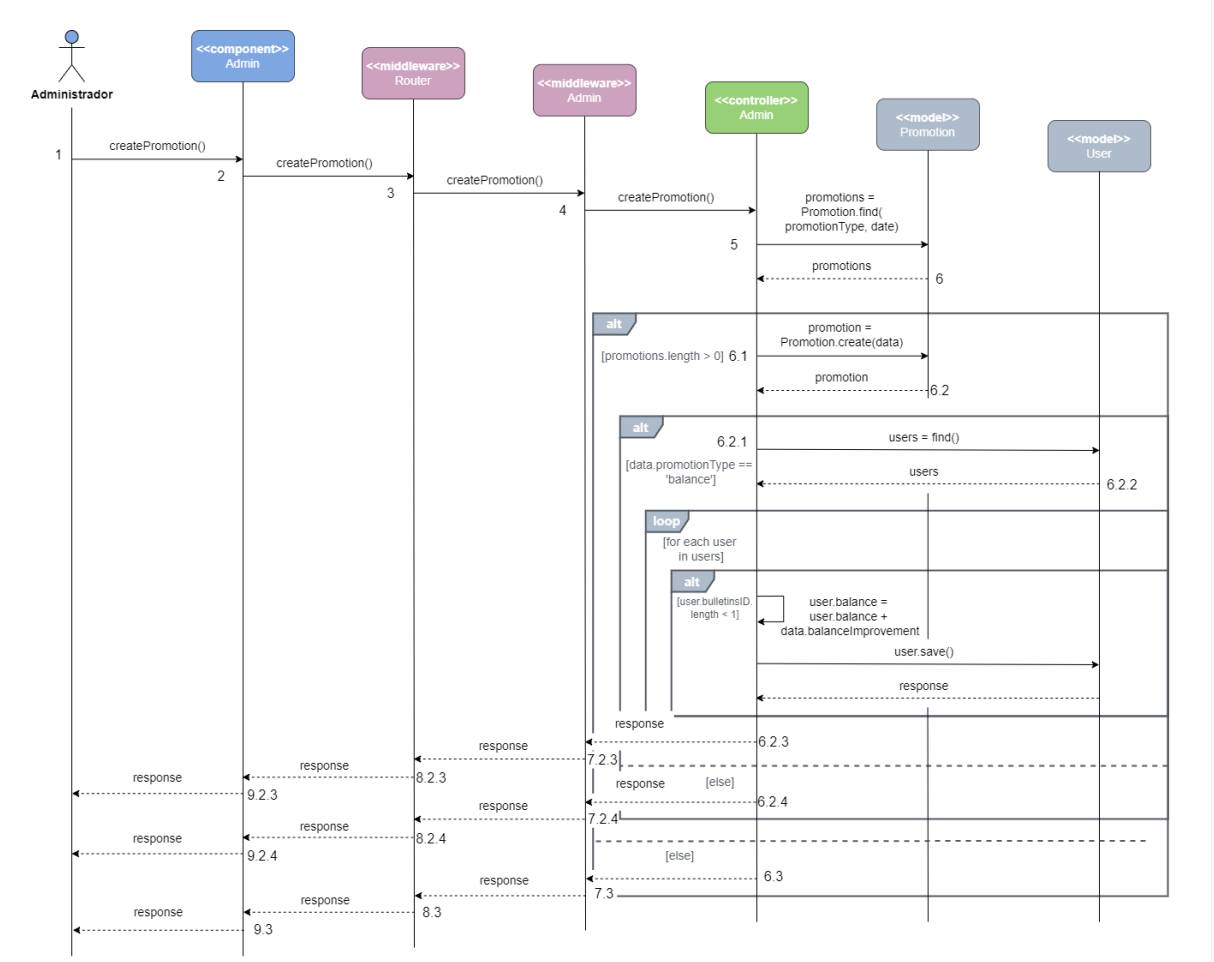

Figura 26: Diagrama de sequência - createPromotion

6.2.20 updateEvent

1. Procurar o *Event* na base de dados via identificador. - **Passo 5**
2. Atualizar o estado do *Event* para o definido e salvaguardar o mesmo na base de dados. - **Passo 7**

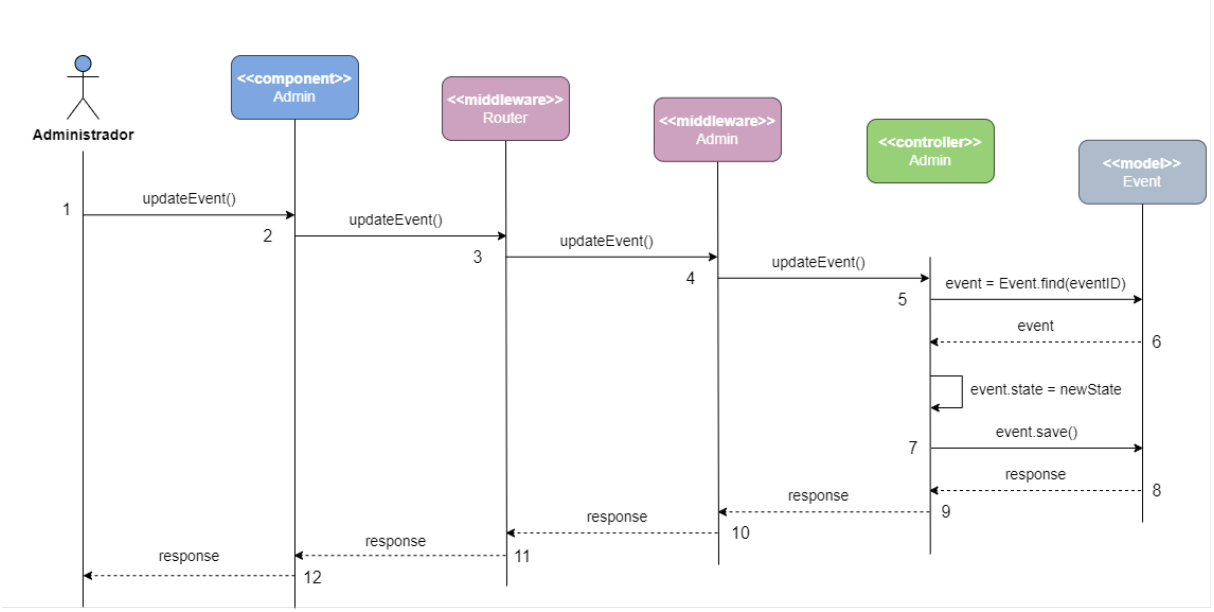

Figura 27: Diagrama de sequência - updateEvent

6.2.21 insertBulletin

1. Criação do boletim em *Frontend* e a confirmação do respetivo. - **Passo 1**
2. Verificação dos campos do boletim.
3. Procurar o utilizador na base de dados associado ao boletim. - **Passo 1.3.3**
4. Procurar todas as promoções na base de dados. - **Passo 1.3.5**
5. Verificar se existe alguma promoção a decorrer e atualizar a *odd* final do boletim.
6. Somar o valor apostado em todas as apostas do boletim.
7. Procurar na base de dados e verificar se algum dos eventos do boletim não está aberto para apostas, e no caso de pelo menos um não estar aberto, alertar para o efeito. - **Passo 1.3.6.3**
8. Verficar se a aposta é simples ou múltipla.
9. Caso seja simples, verificar se o saldo do utilizador é suficiente para realizar a aposta, e no caso de não ser, alertar para o efeito. - **Passo 1.3.6.1**
10. Uma vez verificado o saldo, se for possível a realização da aposta simples, criar cada boletim com cada aposta na base de dados, calculando a *odd* final de cada uma; é de frisar que todos os evento serão seguidos pelo utilizador e o saldo do mesmo será atualizado, assim com a sua lista de transações salvaguardando-se na base de dados.
11. Caso seja múltipla, verificar se o saldo do utilizador é suficiente para realizar a aposta, e no caso de não ser, alertar para o efeito. - **Passo 1.3.6.2.2**
12. Uma vez verificado o saldo, se for possível a realização da aposta múltipla, criar o boletim com cada aposta na base de dados, calculando a *odd* final do boletim; é de frisar que todos os evento serão seguidos pelo utilizador e o saldo do mesmo será atualizado, assim com a sua lista de transações salvaguardando-se na base de dados.
13. Alertar acerca do registo da aposta. - **Passo 1.3.6.2.1**

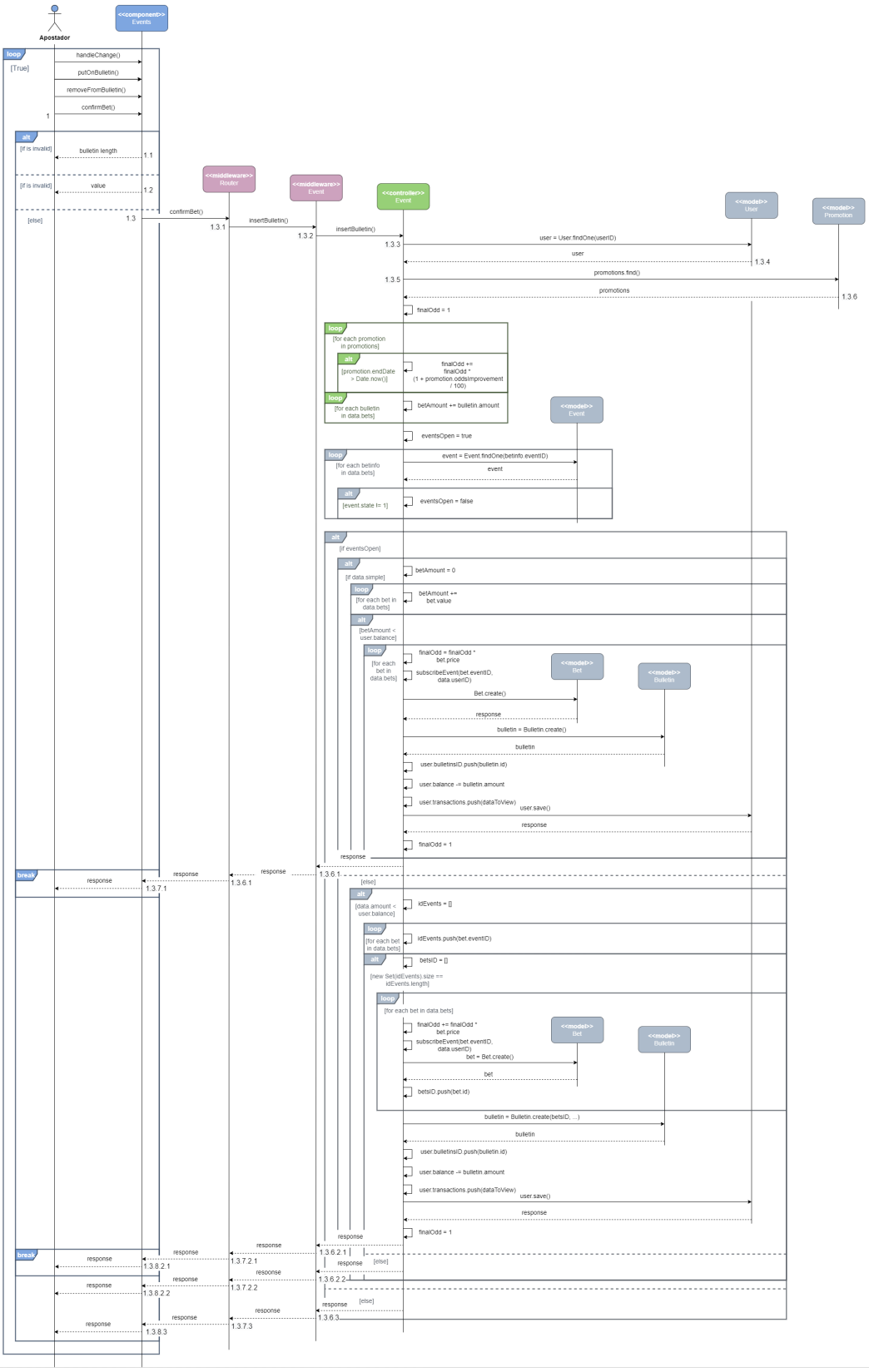

Figura 28: Diagrama de sequência - insertBulletin

6.2.22 sync

1. Quando iniciada a aplicação, num intervalo de 30 segundos é chamado o método *updateFootballGames*.
2. Atualizar os eventos na base de dados através da chamada do método auxiliar *getAllGamesAPI* do componente *RASBetAPIAdapter*.
3. Fazer um pedido à RASBetAPI e retirar o os dados releventes de cada evento. **Passo 2 - *getAllGamesAPI Aux method***
4. Procurar cada evento na base de dados, e caso existam, verificar a mudança de estado - se o estado de um evento mudar, todos os seguidores desse devem ser notificados, e se o evento terminar, as apostas sobre esse evento devem ser resolvidas.
5. Salvaguardar os eventos com os novos dados ou criar os eventos na base de dados, se forem novos.

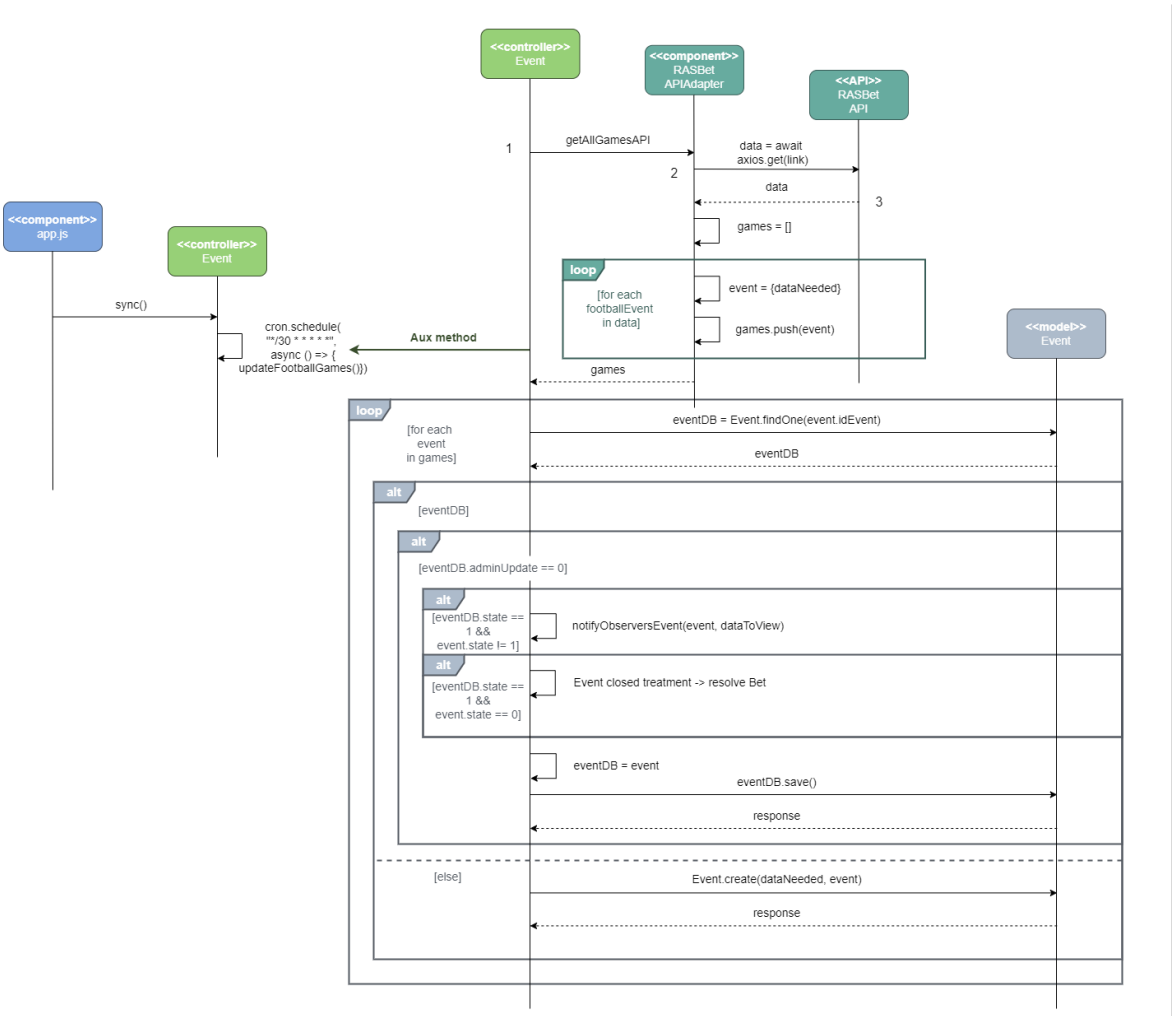

Figura 29: Diagrama de sequência - sync

6.2.23 getBulletin

1. Procurar o boletim na base de dados via identificador. - **Passo 5**
2. Procurar na base de dados e copiar todos os eventos presentes em todas as apostas do boletim.
3. Criar a lista de objetos a enviar para o *Frontend*.
4. Enviar o novo boletim sob a forma definida. - **Passo 7**

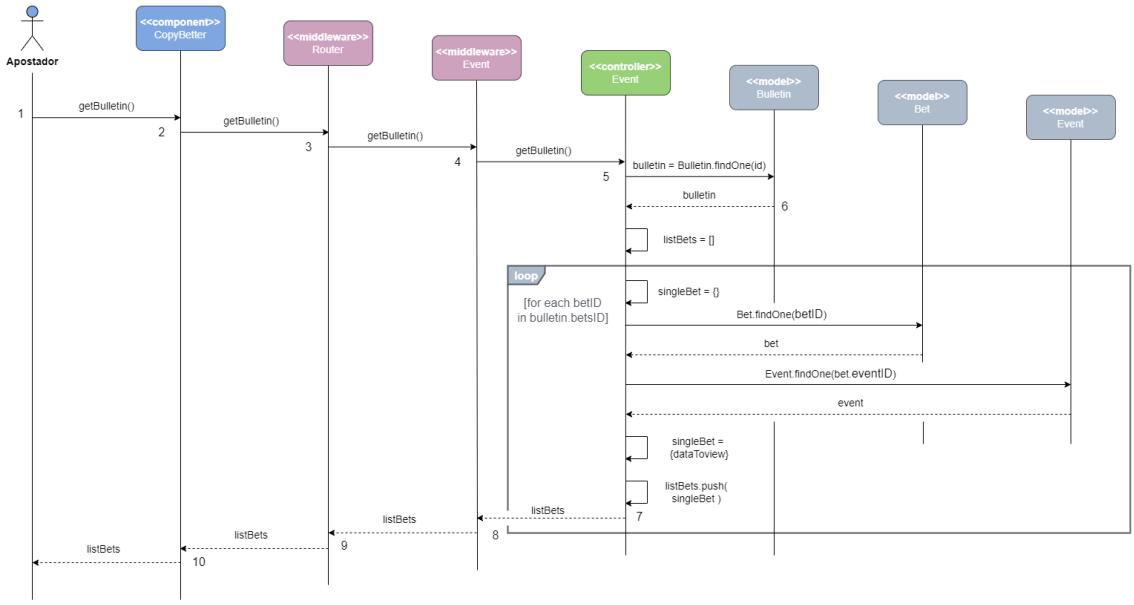

Figura 30: Diagrama de sequência - getBulletin

6.2.24 getAllEvents

1. Procurar todos os *Event's* na base de dados. - **Passo 5**
2. Criar o objeto a enviar para o *Frontend* com os dados a visualizar.
3. Enviar os eventos sob a forma definida. - **Passo 7**

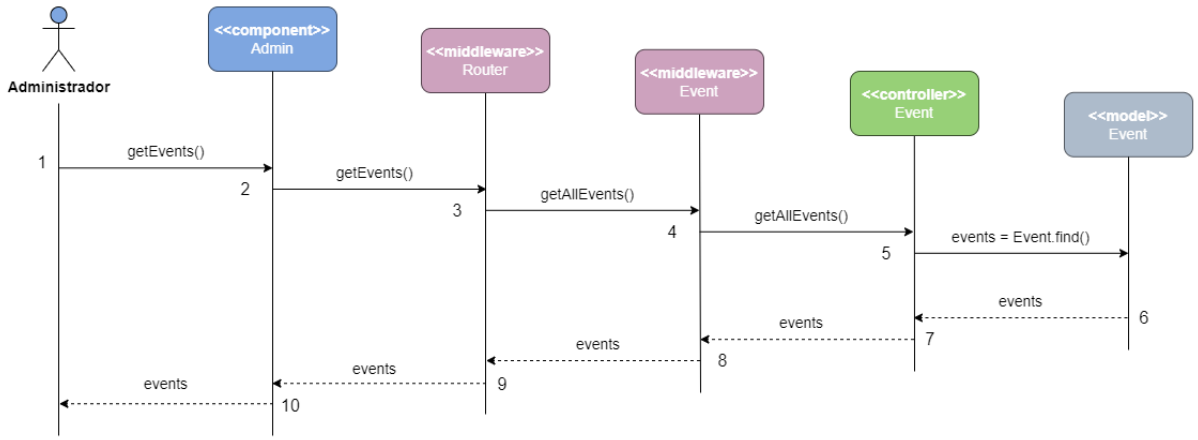

Figura 31: Diagrama de sequência - getAllEvents

6.2.25 getAllEventsOn

1. Procurar todos os *Event's* na base de dados. - **Passo 5**
2. Filtrar os eventos através dos nomes dos desportos e pela data dos repetivos - os eventos que já aconteceram são excluídos.
3. Criar o objeto a enviar para o *Frontend* com os dados a visualizar.
4. Enviar os eventos sob a forma definida. - **Passo 7**

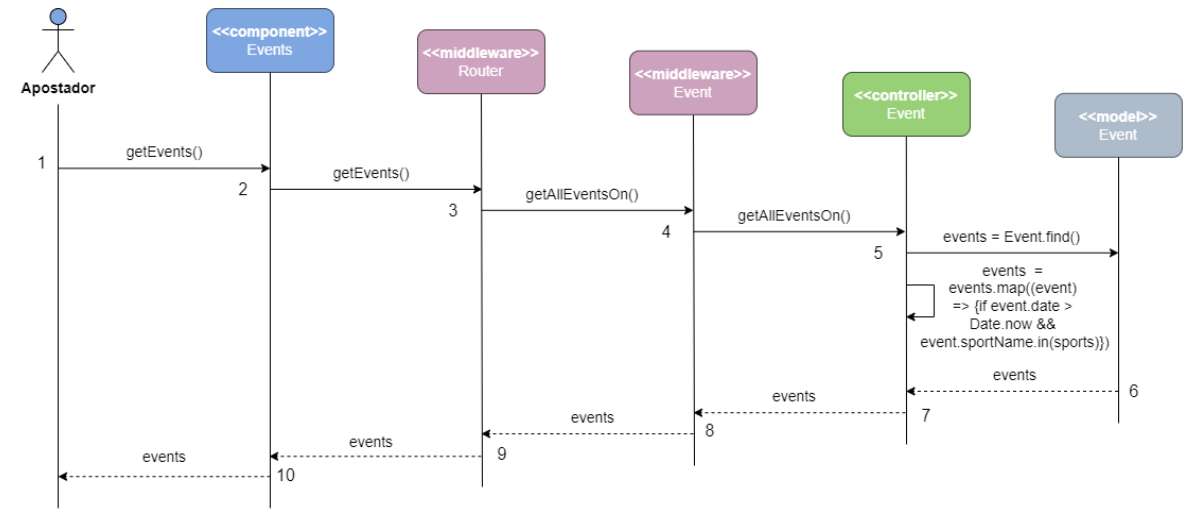

Figura 32: Diagrama de sequência - getAllEventsOn

7 Modelação de instalação

No que diz respeito à modelação de instalação, podemos observar cinco elementos. O elemento central é o *web server*, num ambiente *Node.js*. Este é composto pela lógica da aplicação, pelo *middleware Express*, e pelo *Route* - componentes já realçados com algum relevo 5.2.2. Este comunica com um servidor *Atlas*, para uma base de dados *MongoDB*, através de um *driver MongoDB* do *Node.js*. Finalmente comunica por HTTP, com as API's (Coin Exchanger e RASBet), e para os *web browsers* dos utilizadores da aplicação.

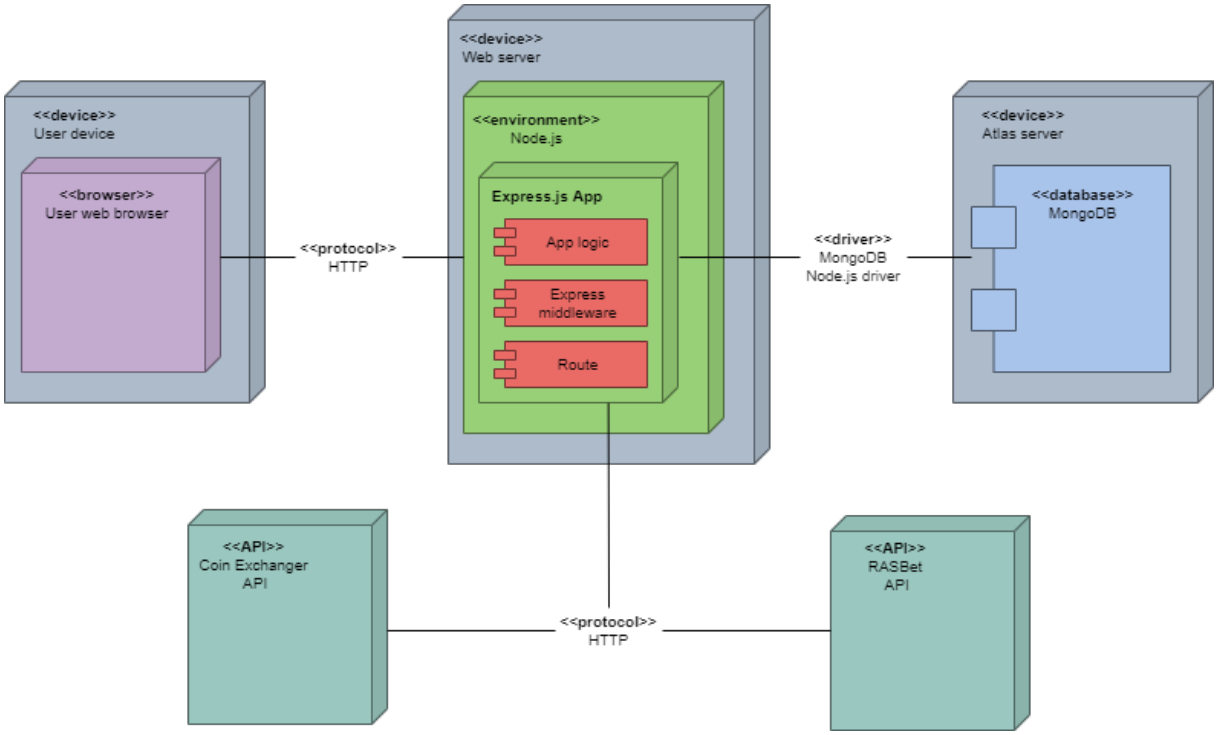

Figura 33: Diagrama de instalação do sistema

Bibliografia

1. Requirements in Engineering Projects - João M. Fernandes, Ricardo J. Machado
2. ISO/IEC 25010 - *Software product quality*
   `https://iso25000.com/index.php/en/iso-25000-standards/iso-25010`
3. Arc42 Template - `https://arc42.org/overview`