1. Introdução

Neste trabalho prático da unidade curricular de Requisitos e Arquiteturas de Software, foi pedido aos alunos que desenvolvessem uma aplicação, chamada RasBet, que permitisse aos seus utilizadores apostar em eventos desportivos. A cada acontecimento está associada uma odd que, multiplicada pelo valor da aposta, determina os ganhos possíveis.

Assim sendo, a primeira etapa do projeto focou-se no levantamento dos requisitos funcionais e sua respetiva implementação. A segunda fase do projeto concentrou-se no levantamento dos requisitos não funcionais e sua respetiva implementação. Na terceira etapa era pedido aos alunos que fossem incorporadas mais duas funcionalidades: a primeira seria o sistema passar a considerar apostas compostas por vários jogos (apostas múltiplas); a segunda consiste na possibilidade de os utilizadores seguirem um jogo, caso estejam interessados nesse jogo, isto é, de ser alertado sempre que as odds forem modificadas ou sobre outras alterações do estado do jogo.

1.1. Visão geral dos requisitos

O principal objetivo da RasBet é proporcionar aos amantes de desporto e apostas uma forma de manifestarem as suas opiniões prevendo resultados desportivos ao colocar em jogo uma quantia monetária nos desfechos destes eventos, através de uma aplicação fácil de utilizar, confiável e onde podem maximizar os seus lucros.

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

- Um apostador pode consultar a lista de eventos disponíveis.
- Um apostador pode criar um boletim de apostas.
- Um apostador pode ver o seu histórico de transações.
- Um apostador é notificado com o resultado dos eventos em que apostou.
- Um especialista é a entidade responsável por definir as odds dos resultados dos jogos.
- Um administrador é a entidade responsável por abrir, fechar ou suspender as apostas e criar promoções.

Em seguida apresentamos alguns dos requisitos não funcionais por ordem do mais relevante ao menos relevante:

- Um utilizador já autenticado e com saldo suficiente na carteira, na página inicial, poderá completar uma aposta simples em menos de 7 cliques.
- O sistema deve ser capaz de suportar múltiplas conexões simultâneas.
- A aplicação deve ser cross-platform, permitindo a execução em diferentes sistemas operativos.
- O sistema deve ser capaz de suportar um grande nível de operações sem restrições ou pontos de falha estruturais
- O sistema deverá garantir a idade mínima de 18 anos no registo de utilizadores.

2. Restrições

2.1. Restrições Técnicas

Restrição	Motivação
Implementação (Backend) em Java	A aplicação deve ser desenvolvida em Java
Frontend em JavaFx	O frotend da aplicação deve ser desenvolvido em JavaFx.O JavaFX fornece um conjunto de bibliotecas e APIs para construir interfaces gráficas com o utilizador em aplicações Java.
Base de Dados SQLite	A aplicação deve utilizar uma base de dados SQLite
Desenvolvimento independente do sistema operativo	A aplicação deve ser cross-platform, permitindo a execução em diferentes sistemas operativos.

2.2. Restrições Organizacionais

Restrição	Motivação
Equipa	Equipa de três engenheiros informáticos da Universidade do Minho
Prazo	A aplicação final deve estar operacional até dia 12 de Janeiro de 2023.
Orçamento	O orçamento total para o desenvolvimento da aplicação é de 50 000€
Monitorização	A empresa monitoriza o desenvolvimento da aplicação via github.

3. Âmbito e contexto do sistema

O sistema desenvolvido apenas interage com os utilizadores (Especialista, Administrador e Apostador) e Bookmaker. É um modelo de aplicação com dois componentes principais: Frontend e Backend. Em relação ao Frontend, este será feito através de uma interface gráfica usando o JavaFx. Quanto ao Backend, este será constituído por dois elementos fundamentais: lógica de negócio e base de dados.

Apostador: Principal entidade do sistema, que pode ver os próximos eventos nos quais pode apostar, criar boletins de apostas e possivelmente ganhar dinheiro com as apostas acertadas.

Especialista: Entidade responsável por definir as odds dos resultados dos jogos.

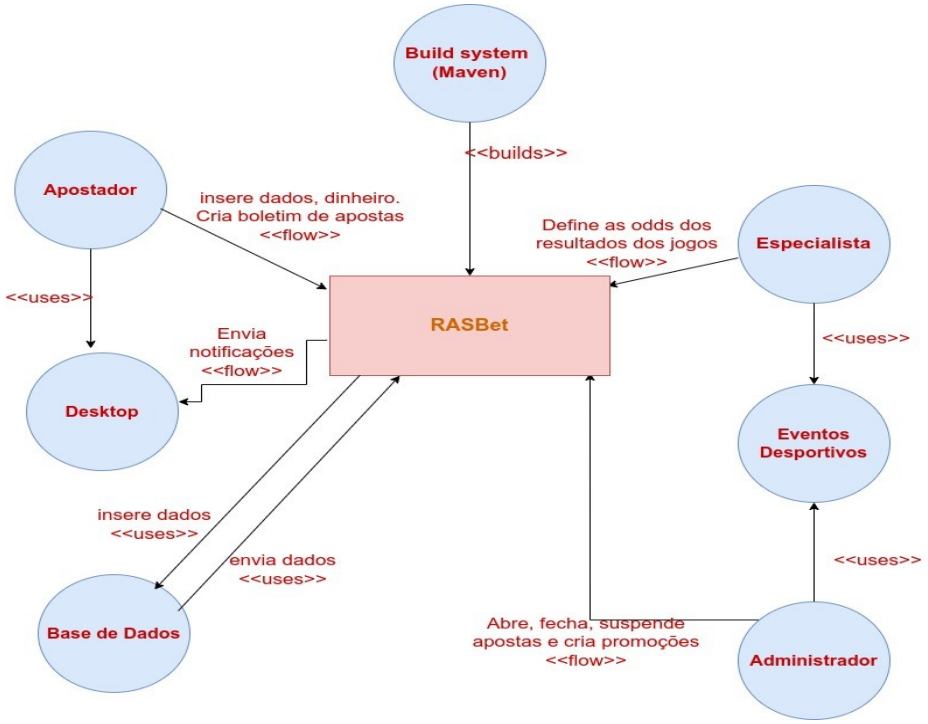
Administrador: Entidade responsável por abrir, fechar ou suspender as apostas e criar promoções.

Aposta: Principal funcionalidade do sistema. Esta é composta por resultados de jogos, podendo ser simples ou múltipla, consoante o número de resultados de jogos diferentes selecionados. Uma aposta simples contém apenas um resultado de um jogo, enquanto que uma aposta múltipla contém 2 ou mais resultados de jogos diferentes, em que todos terão de ser vencedores para que a aposta seja ganha. Associado às apostas temos o conceito de odds, que são as cotações dadas a um determinado jogo, isto é, as probabilidades de um determinado evento ocorrer.

Notificações: Notificações do sitema que anunciam o resultado das apostas quando um jogo acaba.

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

Em seguida apresentamos um diagrama de contexto, que fornece uma visão geral da situação atual do sistema e de como ele se relaciona com o seu ambiente.



4. Estratégia da Solução

Em seguida apresentamos as principais linhas de raciocínio seguidas durante o planeamento e concretização da arquitetura do sistema.

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

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

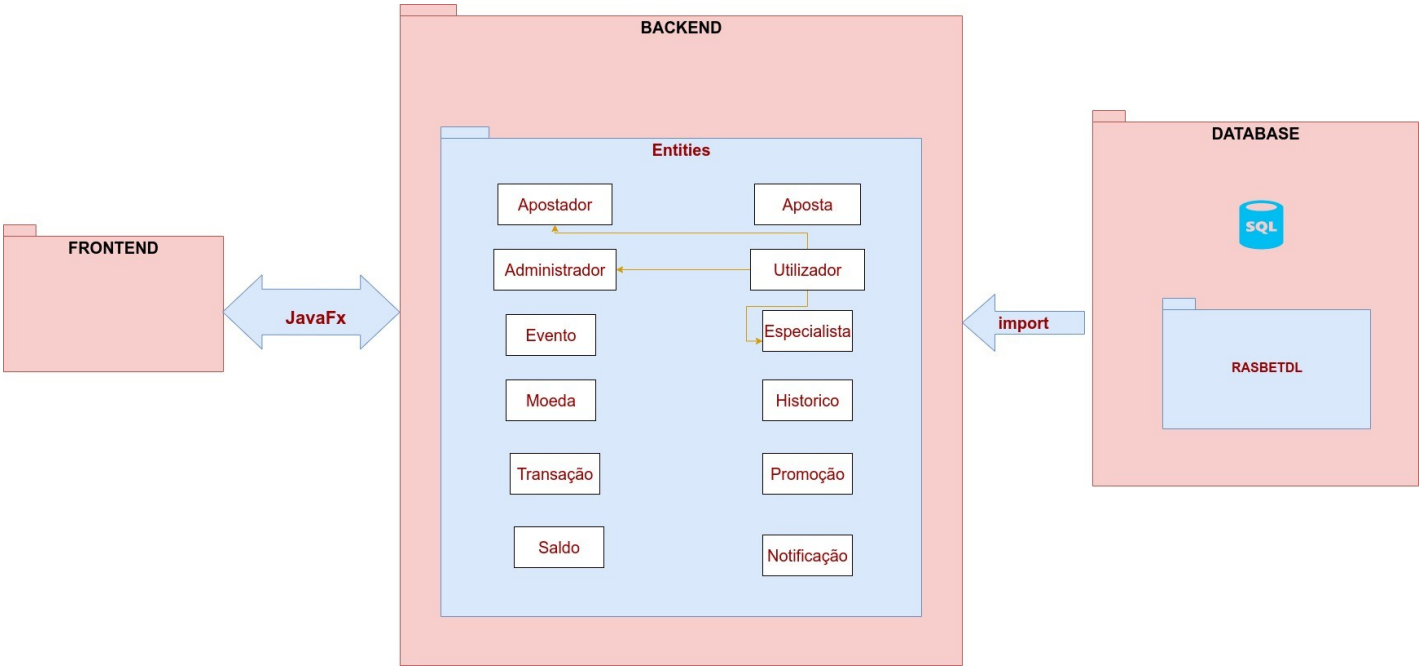
A API irá utilizar SQLite. Esta plataforma deve ser segura o suficiente para evitar que outros utilizadores alterem os dados de outros.

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

5. Building Block View

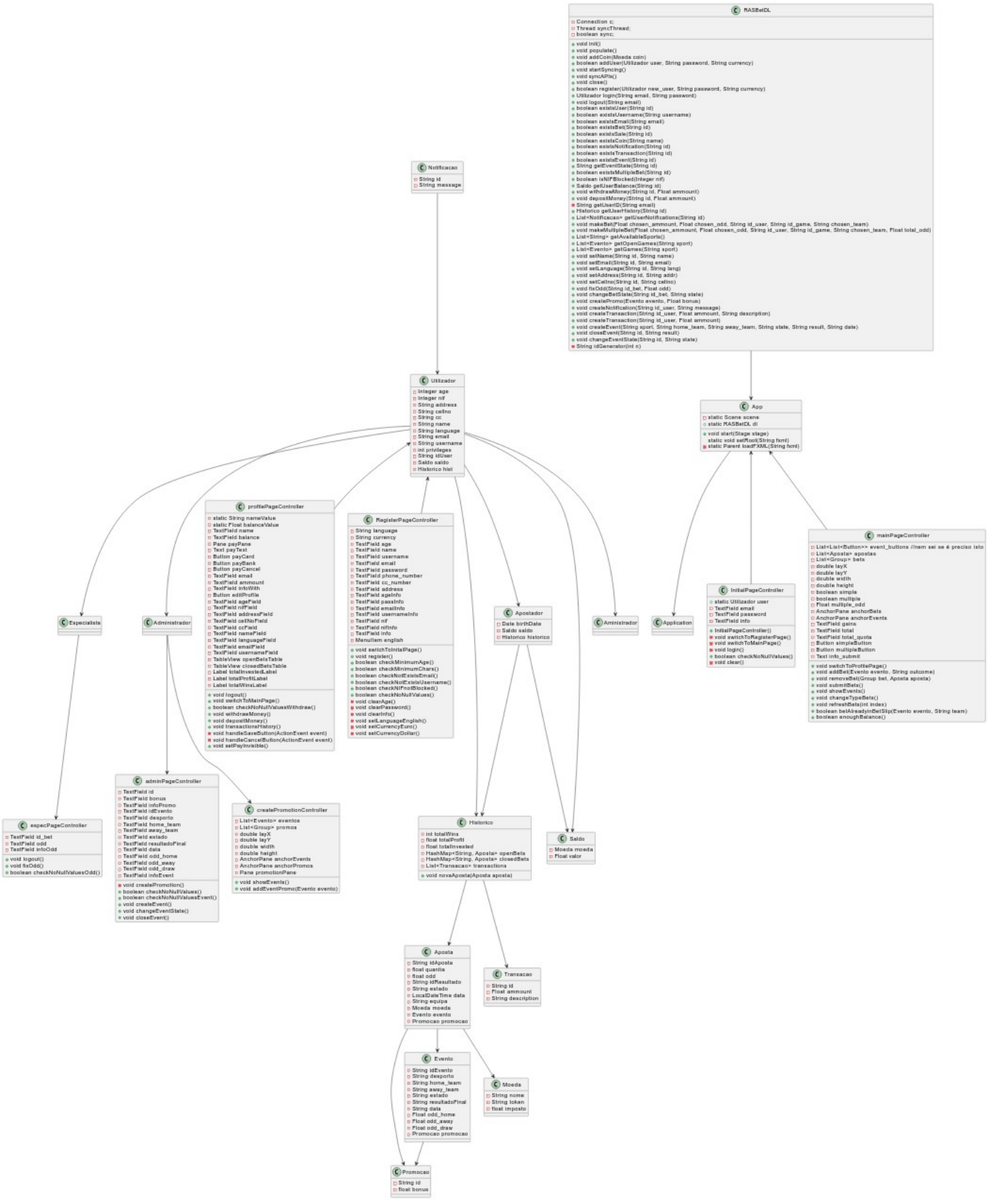
A arquitetura do sistema começa o seu desenvolvimento a partir da identificação de possíveis subsistemas. Esta estratégia permite uma maior organização, uma vez que tanto as classes e os seusrespetivos métodos serão agrupados de acordo com o subsistema onde melhor se adequam.

De seguida, procedeu-se à modelação dos requisitos levantados na etapa anterior deste projeto. Para tal, foi elaborado um diagrama de componentes que descreve os componentes do sistema e a dependência entre eles, permitindo assim, identificar o que é necessário para construir o sistema. Além disso, foi criado também um diagrama de packages, de modo a facilitar a gestão do número crescente de classes, e ainda a identificar o papel de uma classe no sistema tal como as dependências entre as diversas classes presentes. Isto permite à equipa descrever informação importante para a evolução do sistema.



O diagrama de classes trata-se de uma visão geral, isto é, uma visão mais abrangente de como os subsistemas estão relacionados, e como são tratadas as dependências entre as classes pertencentes a cada um deles.

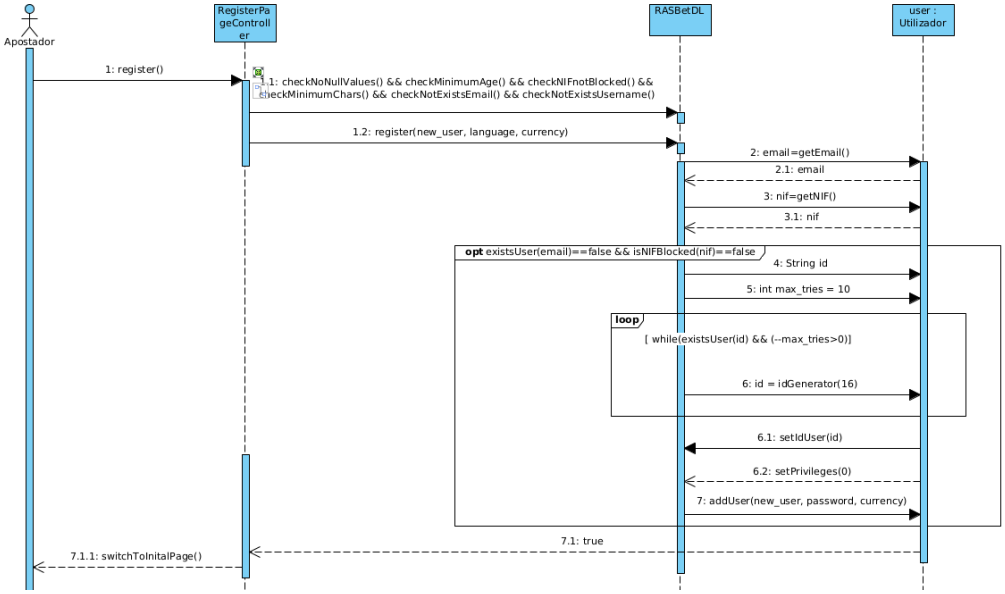
Primeiramente, é na interface RASBetDL onde se encontram os métodos principais do sistema, que representam as funcionalidades da aplicação a desenvolver. Estes métodos irão chamar “sub-métodos” que estão definidos nas interfaces de cada subsistema. Deste modo, diminui-se a dependência entre classes, e dá-se prioridade ao encapsulamento de dados.


MASBetDL
- Connection c;
- Thread syncThread;
- boolean sync;
+ void init()
+ void populate()
+ void addCoin(Moeda coin)
+ boolean addUser(Utilizador user, String password, String currency)
+ void startSyncing()
+ void syncAPIs()
+ void close()
+ boolean register(Utilizador new_user, String password, String currency)
+ Utilizador login(String email, String password)
+ void logout(String email)
+ boolean existsUser(String id)
+ boolean existsUsername(String username)
+ boolean existsEmail(String email)
+ boolean existsBet(String id)
+ boolean existsSale(String id)
+ boolean existsCoin(String name)
+ boolean existsNotification(String id)
+ boolean existsTransaction(String id)
+ boolean existsEvent(String id)
+ String getEventState(String id)
+ boolean existsMultipleBet(String id)
+ boolean isNIFBlocked(Integer nif)
+ Saldo getUserBalance(String id)
+ void withdrawMoney(String id, Float ammount)
+ void depositMoney(String id, Float ammount)
- String getUserID(String email)
+ Historico getUserHistory(String id)
+ List<Notificacao> getUserNotifications(String id)
+ void makeBet(Float chosen_ammount, Float chosen_odd, String id_user, String id_game, String chosen_team)
+ void makeMultipleBet(Float chosen_ammount, Float chosen_odd, String id_user, String id_game, String chosen_team, Float total_odd)
+ List<String> getAvailableSports()
+ List<Evento> getOpenGames(String sport)
+ List<Evento> getGames(String sport)
+ void setName(String id, String name)
+ void setEmail(String id, String email)
+ void setLanguage(String id, String lang)
+ void setAddress(String id, String addr)
+ void setCellno(String id, String cellno)
+ void fixOdds(String id_bet, Float odd)
+ void changeBetState(String id_bet, String state)
+ void createPromo(Evento evento, Float bonus)
+ void createNotification(String id_user, String message)
+ void createTransaction(String id_user, Float ammount, String description)
+ void createTransaction(String id_user, Float ammount)
+ void createEvent(String sport, String home_team, String away_team, String state, String result, String date)
+ void closeEvent(String id, String result)
+ void changeEventState(String id, String state)
- String idGenerator(int n)
Notificacao
- String id
- String message
Utilizador
- Integer age
- Integer nif
- String address
- String cellno
- String cc
- String name
- String language
- String email
- String username
- int privileges
- String idUser
- Saldo saldo
- Historico hist
App
- static Scene scene
- static MASBetDL dl
+ void start(Stage stage)
static void setRoot(String fxml)
- static Parent loadFXML(String fxml)
profilePageController
- static String nameValue
- static Float balanceValue
- TextField name
- TextField balance
- Pane payPane
- Text payText
- Button payCard
- Button payBank
- Button payCancel
- TextField email
- TextField ammount
- TextField infoWth
- Button editProfile
- TextField ageField
- TextField nifField
- TextField addressField
- TextField cellNoField
- TextField ccField
- TextField nameField
- TextField languageField
- TextField emailField
- TextField usernameField
- TableView openBetsTable
- TableView closedSetsTable
- Label totalInvestedLabel
- Label totalProfitLabel
- Label totalWinsLabel
+ void logout()
+ void switchToMainPage()
+ boolean checkNoNullValuesWithdraw()
+ void withdrawMoney()
+ void depositMoney()
+ void transactionsHistory()
- void handleSaveButton(ActionEvent event)
- void handleCancelButton(ActionEvent event)
+ void setPayInvisible()
RegisterPageController
- String language
- String currency
- TextField age
- TextField name
- TextField username
- TextField email
- TextField password
- TextField phone_number
- TextField cc_number
- TextField address
- TextField ageInfo
- TextField passInfo
- TextField emailInfo
- TextField usernameInfo
- TextField nif
- TextField nifInfo
- TextField info
- MenuItem english
+ void switchToInitalPage()
+ void register()
+ boolean checkMinimumAge()
+ boolean checkMinimumChars()
+ boolean checkNotExistsEmail()
+ boolean checkNotExistsUsername()
+ boolean checkNIFnotBlocked()
+ boolean checkNoNullValues()
- void clearAge()
- void clearPassword()
- void clearInfo()
+ void setLanguageEnglish()
- void setCurrencyEuro()
- void setCurrencyDollar()
mainPageController
- List<List<Button>> event_buttons //nem sei se é preciso isto
- List<Aposta> apostas
- List<Group> bets
- double layX
- double layY
- double width
- double height
- boolean simple
- boolean multiple
- float multiple_odd
- AnchorPane anchorBets
- AnchorPane anchorEvents
- TextField gains
- TextField total
- TextField total_quota
- Button simpleButton
- Button multipleButton
- Text info_submit
+ void switchToProfilePage()
+ void addBet(Evento evento, String outcome)
+ void removeBet(Group bet, Aposta aposta)
+ void submitBets()
+ void showEvents()
+ void changeTypeBets()
+ void refreshBets(int index)
+ boolean betAlreadyInBetSlip(Evento evento, String team)
+ boolean enoughBalance()
InitialPageController
+ static Utilizador user
- TextField email
- TextField password
- TextField info
+ InitialPageController()
- void switchToRegisterPage()
- void switchToMainPage()
- void login()
+ boolean checkNoNullValues()
- void clear()
Especialista
Administrador
Apostador
Administrador
Application
especPageController
- TextField id_bet
- TextField odd
- TextField infoOdd
+ void logout()
+ void fixOdd()
+ boolean checkNoNullValuesOdd()
adminPageController
- TextField id
- TextField bonus
- TextField infoPromo
- TextField idEvento
- TextField desporto
- TextField home_team
- TextField away_team
- TextField estado
- TextField resultadoFinal
- TextField data
- TextField odd_home
- TextField odd_away
- TextField odd_draw
- TextField infoEvent
- void createPromotion()
+ boolean checkNoNullValues()
+ boolean checkNoNullValuesEvent()
+ void createEvent()
+ void changeEventState()
+ void closeEvent()
createPromotionController
- List<Evento> eventos
- List<Group> promos
- double layX
- double layY
- double width
- double height
- AnchorPane anchorEvents
- AnchorPane anchorPromos
- Pane promotionPane
+ void showEvents()
+ void addEventPromo(Evento evento)
Apostador
- Date birthDate
- Saldo saldo
- Historico historico
Historico
- int totalWins
- float totalProfit
- float totalInvested
- HashMap<String, Aposta> openBets
- HashMap<String, Aposta> closedBets
- List<Transacao> transactions
+ void novaAposta(Aposta aposta)
Saldo
- Moeda moeda
- Float valor
Aposta
- String idAposta
- float quantia
- float odd
- String idResultado
- String estado
- LocalDateTime data
- String equipa
- Moeda moeda
- Evento evento
- Promocao promocao
Transacao
- String id
- Float ammount
- String description
Evento
- String idEvento
- String desporto
- String home_team
- String away_team
- String estado
- String resultadoFinal
- String data
- Float odd_home
- Float odd_away
- Float odd_draw
- Promocao promocao
Moeda
- String nome
- String token
- float imposto
Promocao
- String id
- float bonus

6. Runtime View

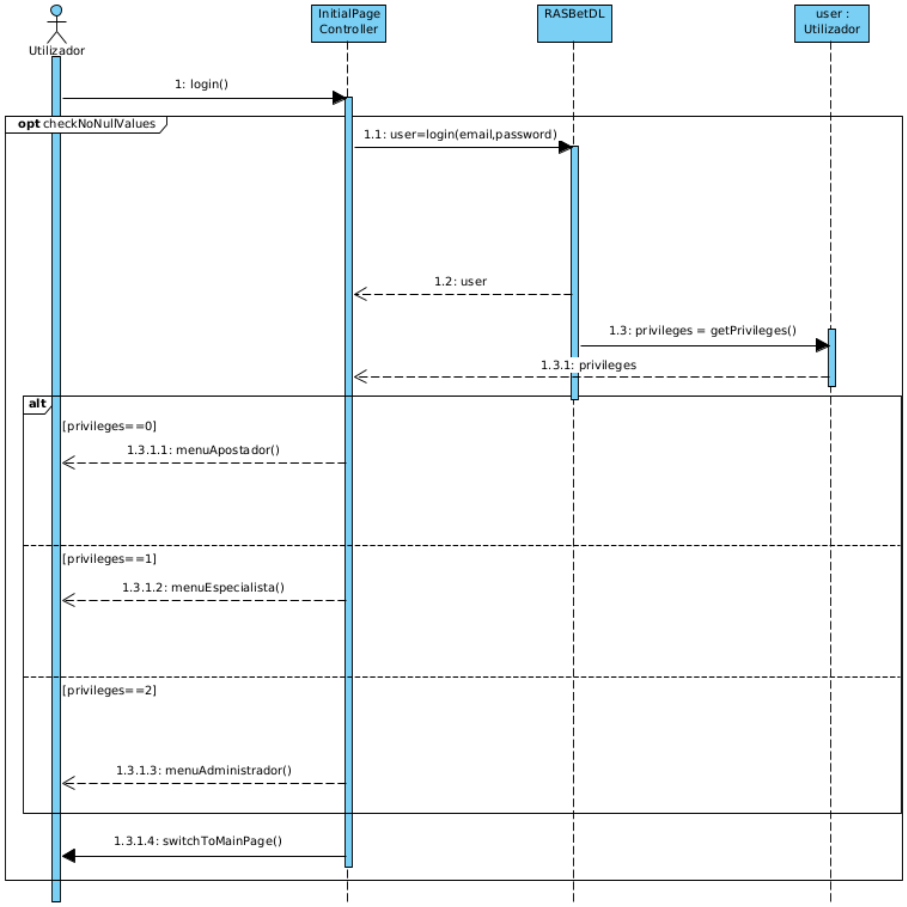
De seguida, apresentamos alguns diagramas de sequência em UML para descrever de forma mais aprofundada a interação entre os diversos blocos do nosso sistema. Para isso foram escolhidas algumas funcionalidades principais que a aplicação implementa.

6.1. Registo do Apostador



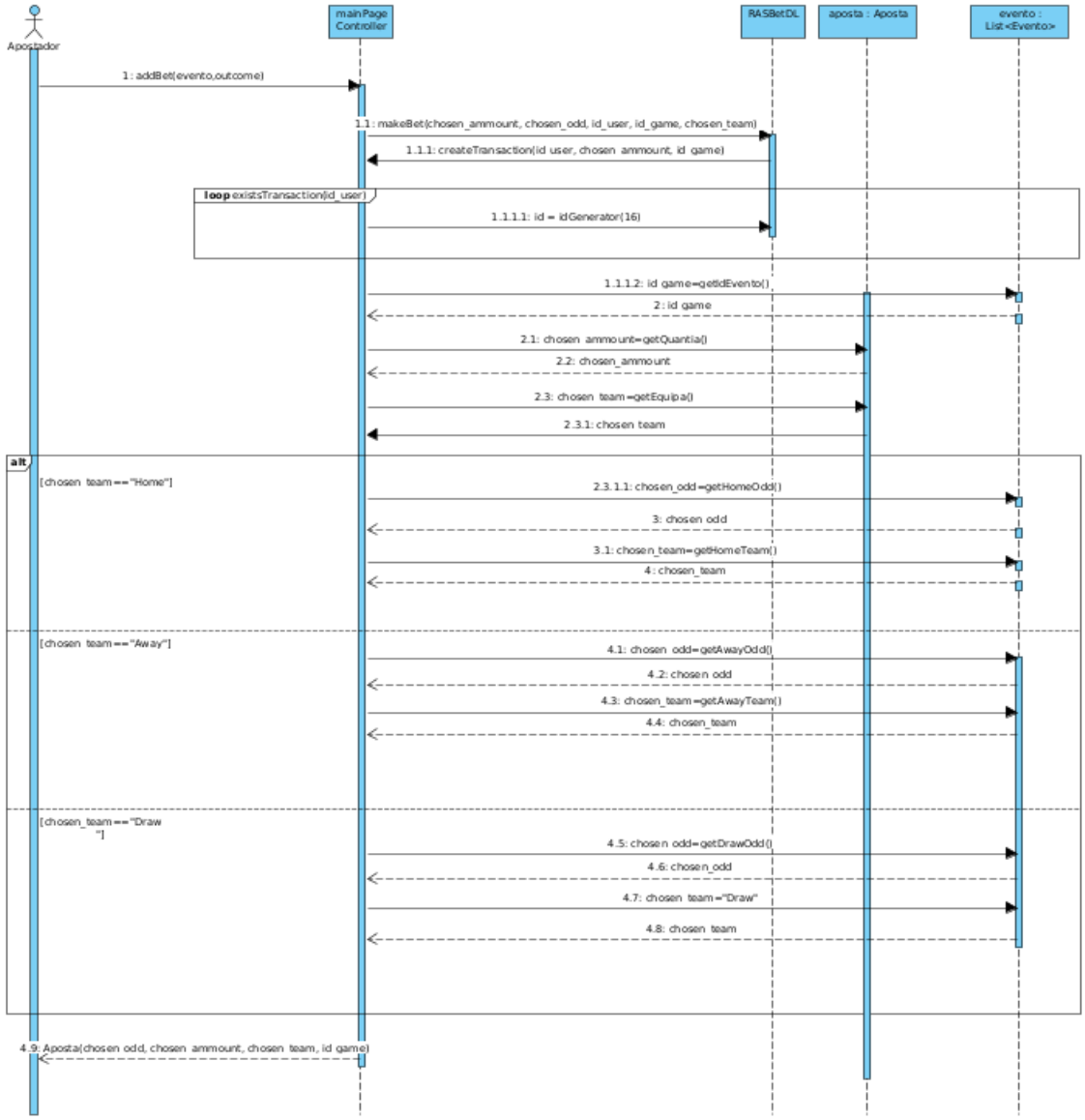
O método register() tem como objetivo registar um novo apostador na aplicação. Para fazer isso, é necessário verificar que todos os campos estão preenchidos, verificar que o apostador têm a idade mínima (18 anos) para apostar, verificar se o NIF é válido e verificar que já não existe um utilizador com esse e-mail. Após todas essas verificações, o utilizador fica registado no sistema e no final, volta à página inicial da aplicação.

6.2. Login Utilizador



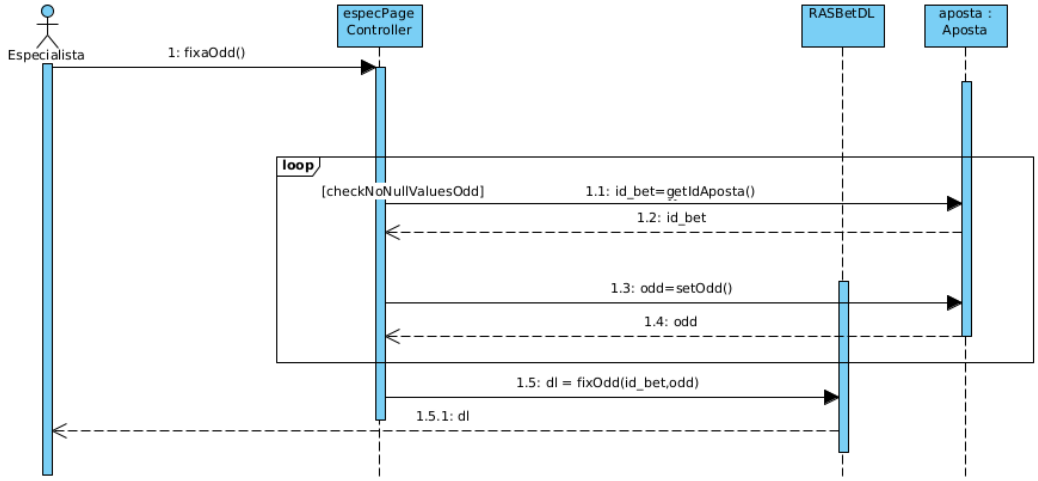
Após um utilizador estar registado é necessário proceder à validação dos seus dados de acesso para entrar novamente na aplicação. Para isso foi necessário a introdução do método login(email, password). A lógica por detrás deste é análoga à lógica do método register(), contudo em vez de adicionar registos, vai verificar se existe um registo cujas credenciais sejam as mesmas que o input da autenticação. Depois de verificar isso, vê que tipo de preivilégios tem o utilizador e direciona-o para o menu correspondente.

6.3. Fazer Aposta



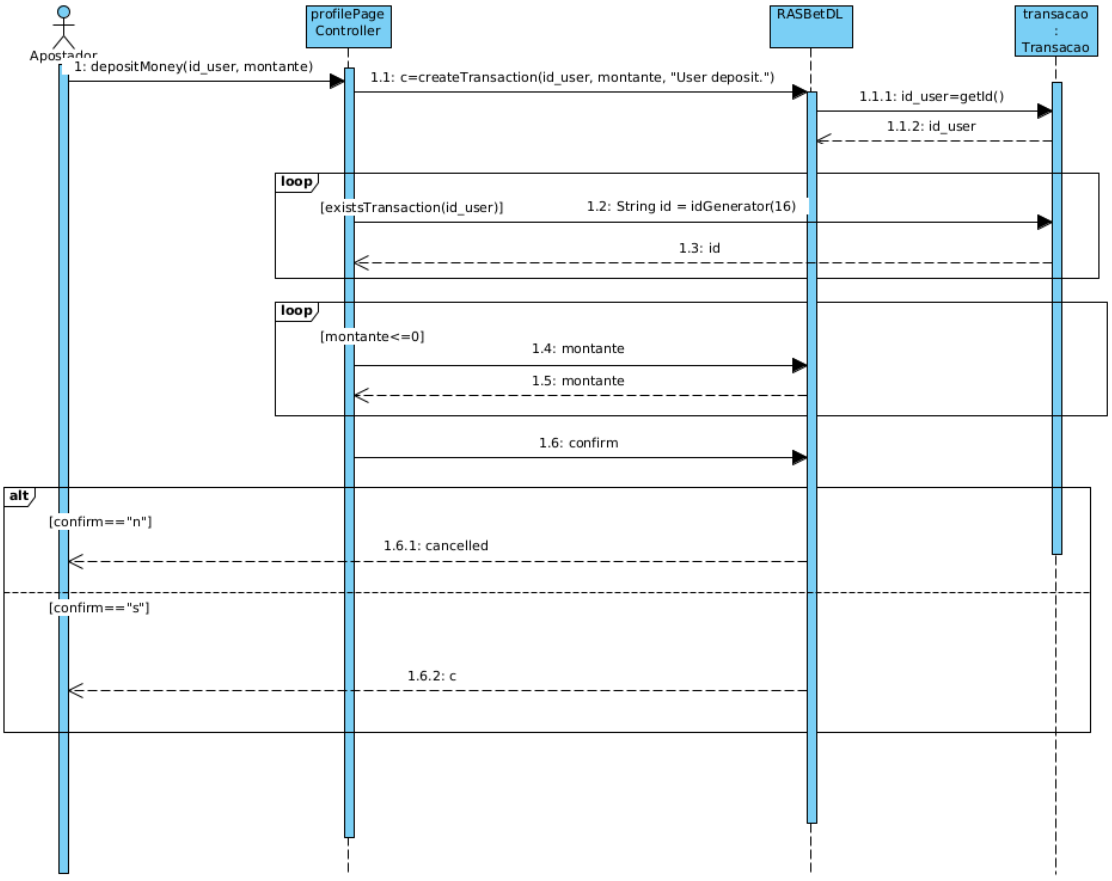
Após o utilizador selecionar a opção de fazer aposta, é invocado o método addBet com o evento que o utilizador pretende apostar e o outcome. Por sua vez, é invocado o metodo makeBet, para que o utilizador selecione o evento em que pretende apostar, a quantia que deseja apostar, a equipa em que deseja apostar e a respetiva odd associada à equipa escolhida.

6.4. Fixar Odd



O especialista é a entidade encarregue por fixar as odd's nos eventos. Para isso ele invoca o méotodo fixOdd, no qual terá que fornecer id do evento e a odd que pretende alterar/fixar.

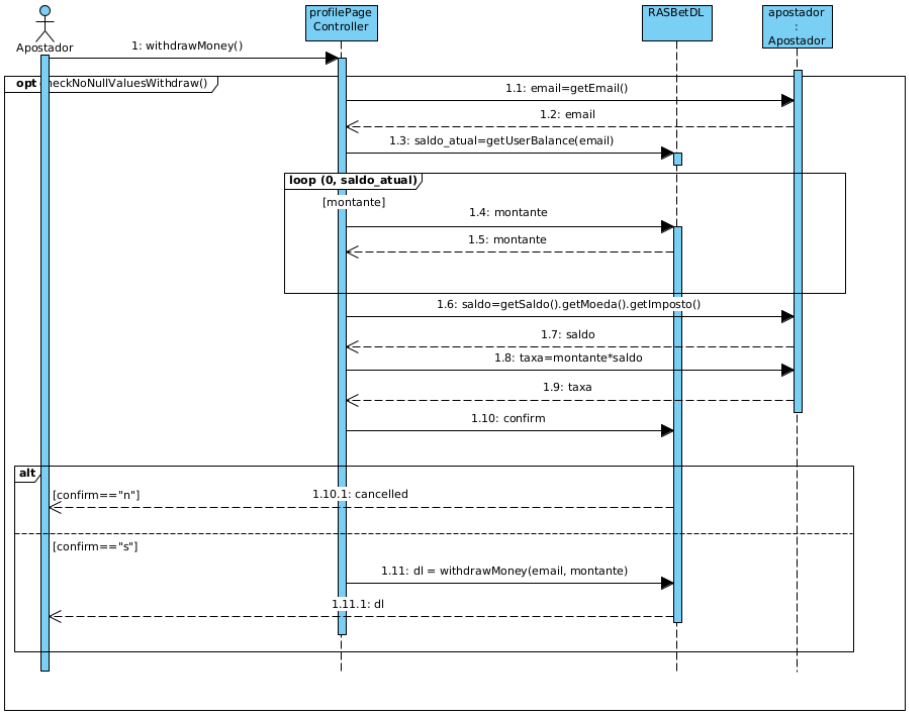
6.5. Depositar Dinheiro



Para depositar dinheiro, o utilizador indica o montante respetivo e o Frontend irá fazer o devido pedido ao Backend. O método depositMoney da classe profilePageController irá fazer todo o processo de atualização da carteira do utilizador.

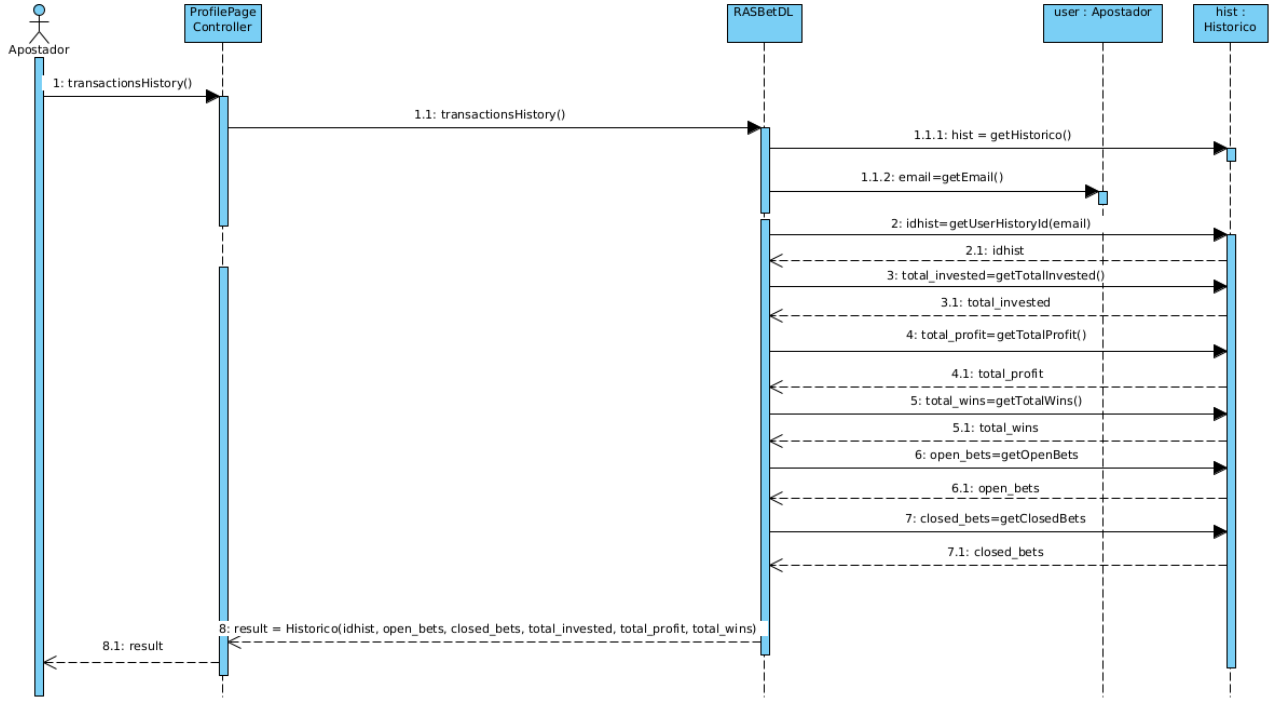
Assim sendo, tanto para levantar como para depositar, será necessário, em primeiro lugar obter o utilizador que efetua a transação e por fim atualizar o saldo da sua carteira.

6.6. Levantar Dinheiro



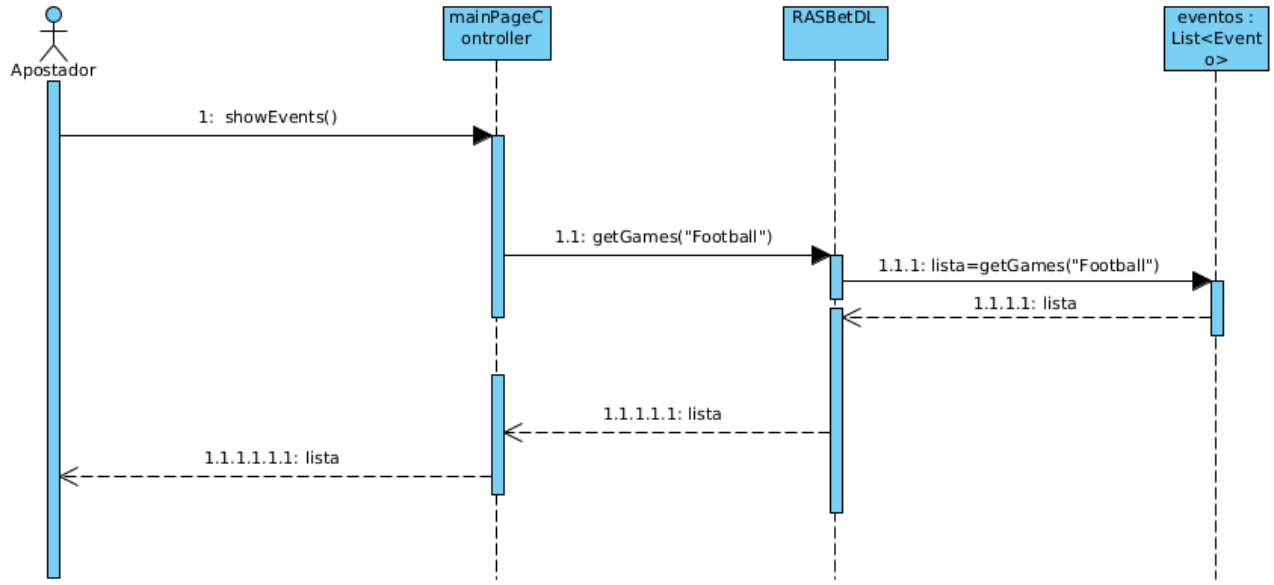
Para levantar dinheiro, a estratégia é semelhante ao de depositar dinheiro. Contudo, neste caso há a exceção de o utilizador não ter dinheiro suficiente.

6.7. Consultar Histórico



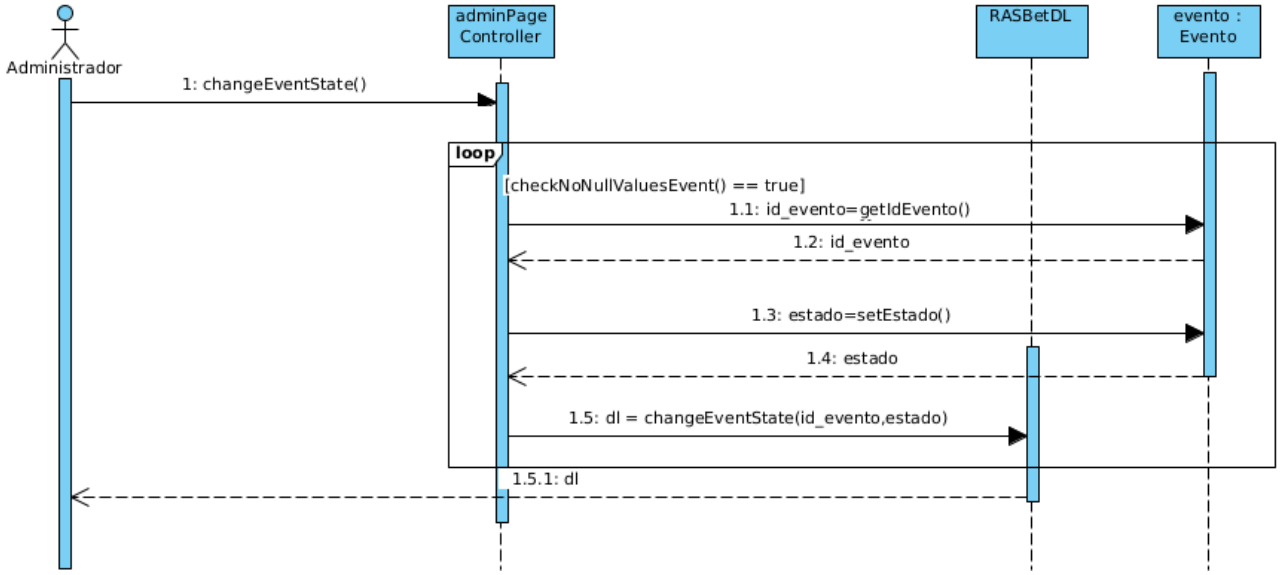
Um apostador pode querer ver o seu historial de apostas, assim sendo é necessário o método transactionsHistory(), que vai mostrar ao utilizador todo o seu histórico de transações, isto é, total investido, total de lucro ganho até ao momento, total de apostas ganhas, número de apostas em aberto e número de apostas terminadas.

6.8. Ver Jogos



Quando um utilizador acede à aplicação pela primeira vez é-lhe mostrado na página inicial uma lista de eventos nos quais pode apostar. Para obter essa lista é invocado o método showEvents() do mainPageController que obtém todos os eventos guardados na base de dados. É ainda possível efetuar uma filtragem do feed por desporto.

6.9. Mudar estado da aposta

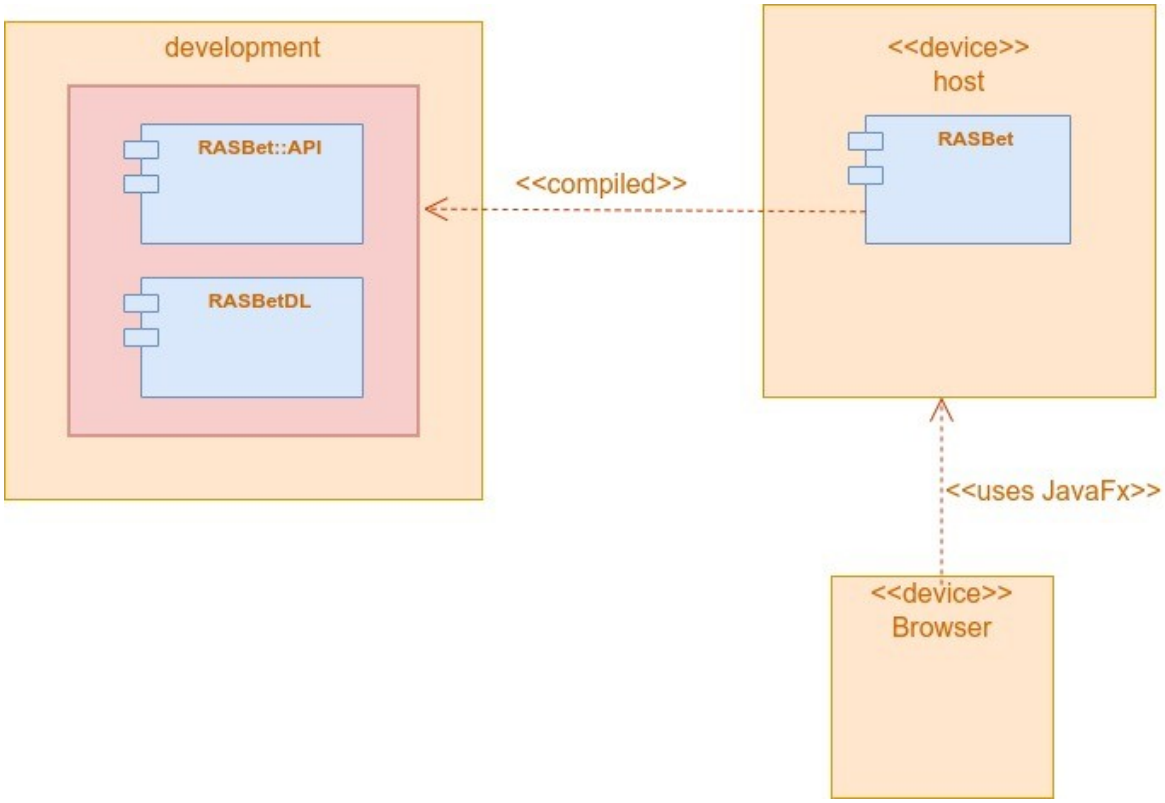


Uma das funcionalidades do administrador é a possibilidade de alterar o estado das apostas. Desta forma, quando o administrador o desejar fazer terá que especificar não só o id das apostas que quer alterar e os seus novos estados, como também o evento no qual essas apostas foram feitas. Ao receber este pedido, o RASBetDL desencadeia o processo de atualização de estado das apostas.

7. Deployment View

Componente	Descrição
RasBet development	Onde ocorre o desenvolvimento da RasBet, computador com JDK 17 e Maven instalados.
Host	Cliente onde é compilada a aplicação.
Desktop	Um desktop para aceder à interface gráfica com o utilizador da aplicação.
RasBet	Ficheiro compilado a partir de componentes de desevolvimento da aplicação e das suas dependências.

No diagrama de deployment, é possível representar a estrutura física onde os vários componentes de software do sistema estão contidos. Assim sendo, apenas existe um device que é o laptop. Este faz uso do JavaFx para comunicar com o utilizador e também de outro componente componente que representa a base de dados sql do sistema.



8. Alterações no código

Para satisfazer as novas funcionalidades, fizeram-se as seguintes mudanças:

- **Base de Dados**

  Aquando da criação das tabelas na base de dados, adicionou-se a tabela ‘Following’:

```
s.executeUpdate(
    "CREATE TABLE IF NOT EXISTS Following (" +

        // FIELDS:
        "id_user        VARCHAR(16)     NOT NULL    ," +
        "id_event       VARCHAR(32)     NOT NULL    ," +

        // CONSTRAINTS:

        // KEYS:
        "PRIMARY KEY (id_user,id_event)," +
        "FOREIGN KEY (id_user) REFERENCES User(id)," +
        "FOREIGN KEY (id_event) REFERENCES Event(id)" +

    ") WITHOUT ROWID;"
);
```

  Desta maneira, haverá uma relação User <-> Evento (muitos para muitos), que registará os seguidores de cada evento, ou os eventos seguidos por cada utilizador.

- **Data Layer**

  Com as mudanças na base de dados, quando há uma alteração nas odds de um evento, facilmente se consultam (e notificam) os utilizadores que o seguem:

```
public void createNotification(String id_user, String message) { try {
    Statement s = c.createStatement();

    String id;

    // Find suitable id for notification
    do { id = idGenerator(n: 16); } while(existsNotification(id));

    s.executeUpdate("INSERT INTO Notification (id, message, id_user) VALUES ('"+id+ "', '"+message+"', '"+id_user+"');");

    s.close();

    } catch (SQLException e) {
        System.out.println(e.getMessage());
        return;
    }
}
```

Figure 1: Função 'createNotification'

```
ResultSet rs = s.executeQuery("SELECT * FROM Outcome WHERE team=='"+home_team+"' AND id_event=='"+id+"';");
if(rs.next()) if(rs.getFloat(columnLabel: "odd") != final_odd_home) {
    rs = s.executeQuery("SELECT id_user FROM Following WHERE id_event=='"+id+"';");
    while(rs.next())
        createNotification(
            rs.getString(columnLabel: "id_user"),
            "Odd for team "+home_team+" on event "+id+" has changed to "+final_odd_home);
}

ResultSet rs1 = s.executeQuery("SELECT * FROM Outcome WHERE team=='"+away_team+"' AND id_event=='"+id+"';");
if(rs1.next()) if(rs1.getFloat(columnLabel: "odd") != final_odd_away) {
    rs1 = s.executeQuery("SELECT id_user FROM Following WHERE id_event=='"+id+"';");
    while(rs1.next())
        createNotification(
            rs1.getString(columnLabel: "id_user"),
            "Odd for team "+away_team+" on event "+id+" has changed to "+final_odd_away);
}

ResultSet rs2 = s.executeQuery("SELECT * FROM Outcome WHERE team=='Draw' AND id_event=='"+id+"';");
if(rs2.next()) if(rs2.getFloat(columnLabel: "odd") != final_odd_tie) {
    rs2 = s.executeQuery("SELECT id_user FROM Following WHERE id_event=='"+id+"';");
    while(rs2.next())
        createNotification(
            rs2.getString(columnLabel: "id_user"),
            "Odd for Draw on event "+id+" has changed to "+final_odd_tie);
}
```

Figure 2: Notificação dos utilizadores interessados em caso de alteração da odd