U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Relatório Final

Métodos Formais em Engenharia de Software
2018/2019

Mestrado Integrado em Engenharia Informática e Computação

Agenda Viral

Bárbara Sofia Silva	up201505628@fe.up.pt
Julieta Frade	up201506530@fe.up.pt

7 de Janeiro 2019

Conteúdo

1 Descrição do Sistema

Este projeto tem como finalidade modelar uma agenda de eventos culturais, à semelhança da plataforma **Agenda Viral**, mas ainda com a possibilidade de comprar bilhetes. Esta tem como objetivo informar o utilizador do que se passa à sua volta, ajudando-o a descobrir os seus eventos preferidos: concertos, exposições, festas, workshops, conferências, etc.

De forma a prestar uma ajuda mais precisa e rápida, é possível aplicar filtros à pesquisa, respetivamente por distrito, cidade, categoria e data. Cada utilizador pode também sugerir eventos que ainda não estejam presentes na aplicação através de um formulário e comprar bilhetes para os eventos já disponíveis na plataforma.

Por último, cabe ao administrador adicionar eventos à agenda, assim como aceitar ou rejeitar os eventos sugeridos por um utilizador habitual. Pode também visualizar algumas estatísticas da sua aplicação.

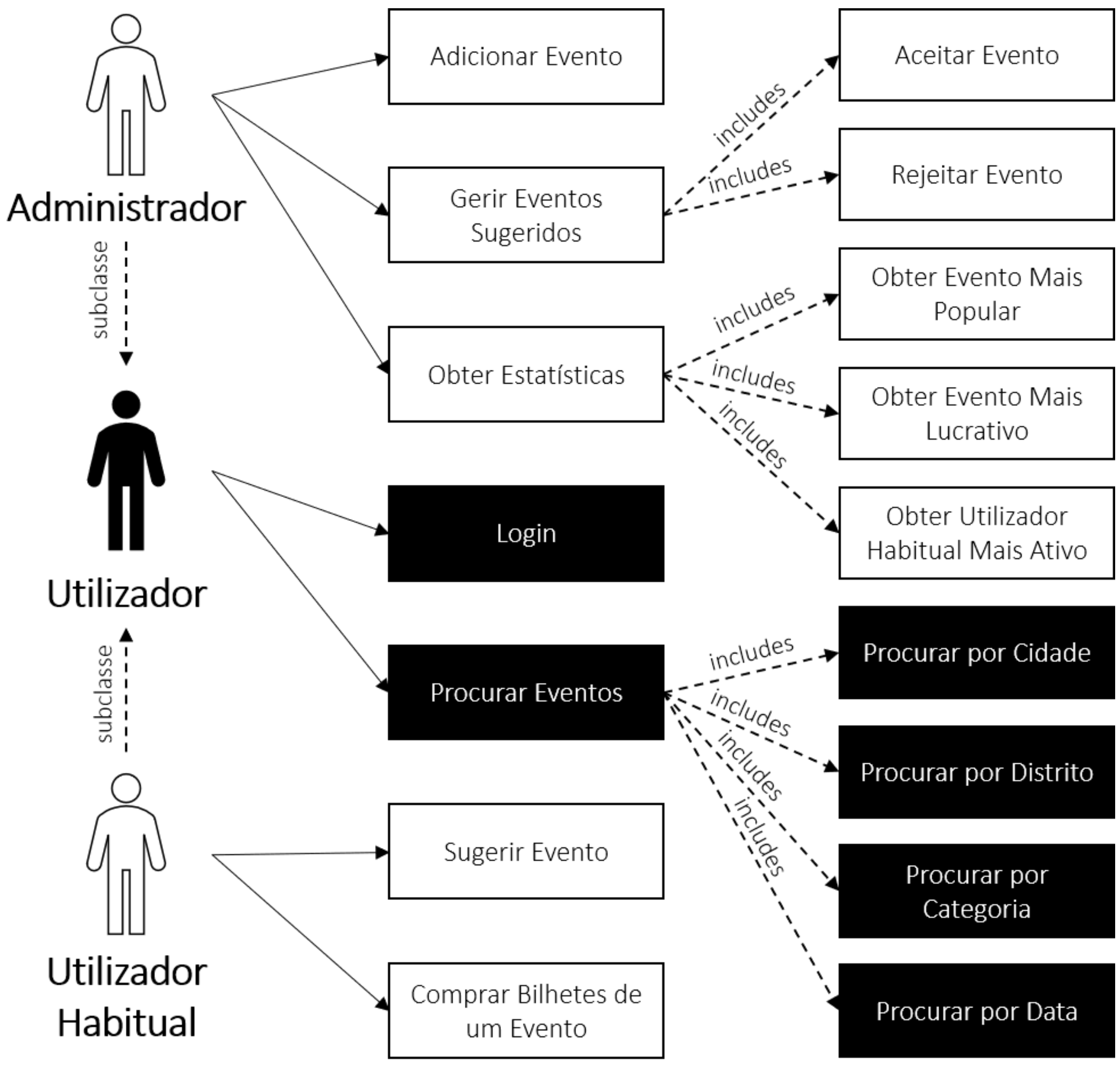
1.1 Lista de Requisitos

Para executar o sistema descrito, são necessários os seguintes requisitos:

ID	Prioridade	Descrição
R01	Obrigatória	Um utilizador pode iniciar sessão
R02	Obrigatória	Um administrador pode adicionar novos eventos
R03	Obrigatória	Um administrador pode aceitar eventos propostos
R04	Obrigatória	Um administrador pode rejeitar eventos propostos
R05	Obrigatória	Um utilizador habitual pode propor eventos
R06	Obrigatória	Um utilizador habitual pode comprar bilhetes para um evento
R07	Obrigatória	Um utilizador pode procurar eventos por distrito
R08	Obrigatória	Um utilizador pode procurar eventos por cidade
R09	Obrigatória	Um utilizador pode procurar eventos por categoria
R10	Opcional	Um utilizador pode procurar eventos por data
R11	Opcional	Um utilizador pode procurar eventos por múltiplos filtros
R12	Obrigatória	Um administrador pode obter o evento mais popular
R13	Opcional	Um administrador pode obter o evento mais lucrativo
R14	Opcional	Um administrador pode obter o utilizador habitual mais ativo

2 Modelo Visual UML

2.1 Modelo de Caso de Uso



Cenário	Iniciar sessão
Descrição	Um utilizador iniciar sessão no sistema
Pré-Condições	1. O email existir no sistema 2. A password ter mais que 8 caracteres
Pós-Condições	1. O email corresponder à password 2. A *token* de sessão estar preenchido OU 1. O email não corresponder à password
Passos	1. No menu inicial, *Login*, preencher os campos necessários
Exceções	1. O utilizador terminar o programa

Cenário	Adicionar um evento
Descrição	Um administrador adicionar um evento
Pré-Condições	1. O utilizador estar com sessão iniciada 2. O utilizador ser administrador 3. A categoria existir no sistema 4. A cidade existir no sistema 5. Não existir nenhum evento com o mesmo título, categoria, data de início, data de fim e cidade no sistema
Pós-Condições	1. O evento existir na lista de eventos
Passos	1. No menu principal escolher a opção *Add Event* 2. Preencher os campos necessários
Exceções	1. O utilizador terminar o programa

Cenário	Aceitar um evento proposto
Descrição	Um administrador aceitar evento proposto
Pré-Condições	1. O utilizador estar com sessão iniciada 2. O utilizador ser administrador 3. O evento existir na lista de eventos propostos 4. O evento não existir na lista de eventos
Pós-Condições	1. O evento existir na lista de eventos 2. O evento não existir na lista de eventos propostos
Passos	1. No menu principal escolher a opção *Proposed Events* 2. Inserir o identificador do evento 3. Aceitar o evento
Exceções	1. O utilizador terminar o programa

Cenário	Rejeitar um evento proposto
Descrição	Um administrador rejeitar evento proposto
Pré-Condições	1. O utilizador estar com sessão iniciada 2. O utilizador ser administrador 3. O evento existir na lista de eventos propostos
Pós-Condições	1. O eventos não existir na lista de eventos propostos
Passos	1. No menu principal escolher a opção *Proposed Events* 2. Inserir o identificador do evento 3. Rejeitar o evento
Exceções	1. O utilizador terminar o programa

Cenário	Propor um evento
Descrição	Um utilizador habitual propor um evento
Pré-Condições	1. O utilizador estar com sessão iniciada 2. O utilizador ser do tipo Regular 3. A categoria existir no sistema 4. A cidade existir no sistema 5. Não existir nenhum evento com o mesmo título, categoria, data de início, data de fim e cidade no sistema
Pós-Condições	1. O evento existir na lista de eventos propostos
Passos	1. No menu principal escolher a opção *Propose Event* 2. Preencher os campos necessário
Exceções	1. O utilizador terminar o programa

Cenário	Comprar bilhetes
Descrição	Um utilizador habitual comprar bilhetes para evento
Pré-Condições	1. O utilizador estar com sessão iniciada 2. O utilizador ser do tipo *Regular* 3. O evento ter bilhetes disponíveis suficientes 4. O evento estar no estado *Available* 5. O utilizador ter um balanço igual ou superior ao preço do bilhetes
Pós-Condições	1. O número de bilhetes vendidos para o evento incrementar as unidades de bilhetes comprados 2. O número de bilhetes comprados pelo utilizador incrementar as unidades de bilhetes comprados 3. O balanço do utilizador decrementar a soma do preço dos bilhetes comprados
Passos	1. Procurar um eventos 2. Selecionar o evento pelo seu id 3. Escolher a opção *Buy Tickets* 4. Introduzir o número de bilhetes pretendidos 5. Submeter
Exceções	1. O utilizador terminar o programa

Cenário	Procurar por distrito
Descrição	Utilizador procurar eventos por distrito
Pré-Condições	1. O utilizador existir no sistema 2. Existir pelo menos um evento no sistema 3. O distrito escolhido existir no sistema
Pós-Condições	1. Os eventos retornados serem no distrito escolhido
Passos	1. No menu principal escolher a opção *Find by District"* 2. Escolher o distrito 3. Submeter
Exceções	1. O utilizador terminar o programa

Cenário	Procurar por cidade
Descrição	Utilizador procurar eventos por cidade
Pré-Condições	1. O utilizador existir no sistema 2. Existir pelo menos um evento no sistema 3. A cidade escolhida existir no sistema
Pós-Condições	1. Os eventos retornados serem na cidade escolhido
Passos	1. No menu principal escolher a opção *Find by City* 2. Escolher a cidade 3. Submeter
Exceções	1. O utilizador terminar o programa

Cenário	Procurar por categoria
Descrição	Utilizador procurar eventos por categoria
Pré-Condições	1. O utilizador existir no sistema 2. Existir pelo menos um evento no sistema 3. A categoria escolhida existir no sistema
Pós-Condições	1. Os eventos retornados serem da categoria escolhida
Passos	1. No menu principal escolher a opção *Find by Category* 2. Escolher a categoria 3. Submeter
Exceções	1. O utilizador terminar o programa

Cenário	Procurar por data
Descrição	Utilizador procurar eventos por data
Pré-Condições	1. O utilizador existir no sistema 2. Existir pelo menos um evento no sistema 3. A data escolhida ser válida
Pós-Condições	1. Os eventos retornados ocorrem no dia escolhido
Passos	1. No menu principal escolher a opção *Find by Date* 2. Escolher a data 3. Submeter
Exceções	1. O utilizador terminar o programa

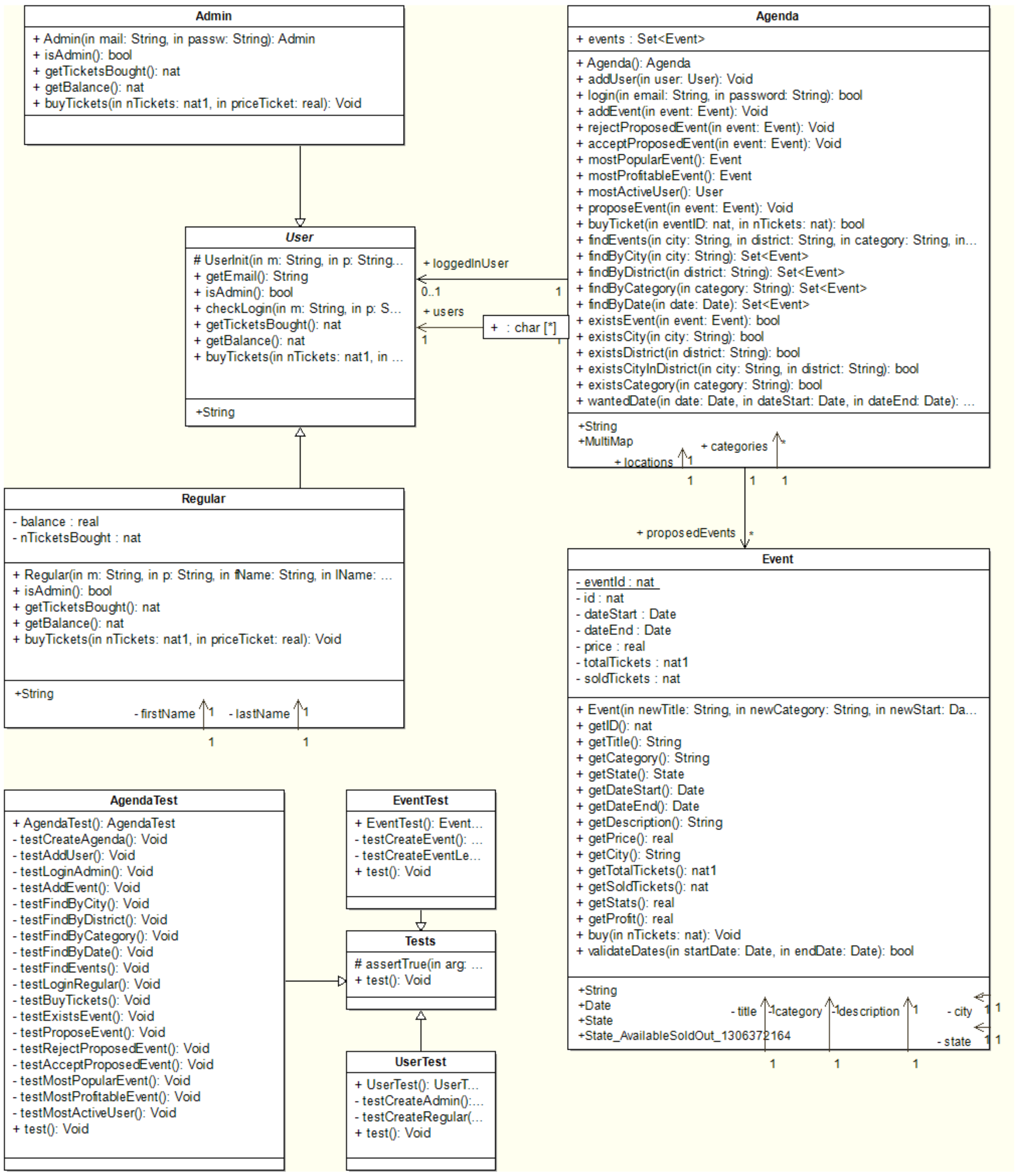
Cenário	Procurar por vários filtros
Descrição	Utilizador procurar eventos por vários filtros
Pré-Condições	1. O utilizador existir no sistema 2. Existir pelo menos um evento no sistema 3. Escolher pelo menos um filtro 4. Não pode escolher os filtros cidade e distrito na mesma procura 5. O distrito escolhido existir no sistema 6. A cidade escolhida existir no sistema 7. A categoria escolhida existir no sistema 8. A data escolhida ser válida
Pós-Condições	1. Os eventos retornados obedecerem aos filtros escolhidos
Passos	1. No menu principal escolher a opção *Find by Multiple Filters* 2. Preencher os filtros desejados 3. Submeter
Exceções	1. O utilizador terminar o programa

Cenário	Obter evento mais popular
Descrição	Um administrador obter o evento mais popular, ou seja, o que tem a maior percentagem de bilhetes vendidos face ao número total de lugares
Pré-Condições	1. O utilizador estar com sessão iniciada 2. O utilizador ser administrador 3. Existir pelo menos um evento na lista de eventos
Pós-Condições	Nenhuma
Passos	1. No menu principal escolher a opção *Most Popular Event*
Exceções	1. O utilizador terminar o programa

Cenário	Obter evento mais lucrativo
Descrição	Um administrador obter o evento mais lucrativo
Pré-Condições	1. O utilizador estar com sessão iniciada 2. O utilizador ser administrador 3. Existir pelo menos um evento na lista de eventos
Pós-Condições	Nenhuma
Passos	1. No menu principal escolher a opção *Most Profitable Event*
Exceções	1. O utilizador terminar o programa

Cenário	Obter utilizador habitual mais ativo
Descrição	Um administrador obter o utilizador habitual mais ativo, ou seja, aquele que comprou mais bilhetes no total
Pré-Condições	1. O utilizador estar com sessão iniciada 2. O utilizador ser administrador 3. Existir pelo menos um utilizador na lista de eventos
Pós-Condições	Nenhuma
Passos	1. No menu principal escolher a opção *Most Active User*
Exceções	1. O utilizador terminar o programa

2.2 Modelo de Classe


Admin
+ Admin(in mail: String, in passw: String): Admin
+ isAdmin(): bool
+ getTicketsBought(): nat
+ getBalance(): nat
+ buyTickets(in nTickets: nat1, in priceTicket: real): Void
User
# UserInit(in m: String, in p: String...
+ getEmail(): String
+ isAdmin(): bool
+ checkLogin(in m: String, in p: S...
+ getTicketsBought(): nat
+ getBalance(): nat
+ buyTickets(in nTickets: nat1, in ...
+String
+ loggedInUser
0..1
1
+ users
+ : char [*]
1
Agenda
+ events : Set<Event>
+ Agenda(): Agenda
+ addUser(in user: User): Void
+ login(in email: String, in password: String): bool
+ addEvent(in event: Event): Void
+ rejectProposedEvent(in event: Event): Void
+ acceptProposedEvent(in event: Event): Void
+ mostPopularEvent(): Event
+ mostProfitableEvent(): Event
+ mostActiveUser(): User
+ proposeEvent(in event: Event): Void
+ buyTicket(in eventID: nat, in nTickets: nat): bool
+ findEvents(in city: String, in district: String, in category: String, in...
+ findByCity(in city: String): Set<Event>
+ findByDistrict(in district: String): Set<Event>
+ findByCategory(in category: String): Set<Event>
+ findByDate(in date: Date): Set<Event>
+ existsEvent(in event: Event): bool
+ existsCity(in city: String): bool
+ existsDistrict(in district: String): bool
+ existsCityInDistrict(in city: String, in district: String): bool
+ existsCategory(in category: String): bool
+ wantedDate(in date: Date, in dateStart: Date, in dateEnd: Date): ...
+String
+MultiMap
+ categories
*
+ locations
1
1
1
1
+ proposedEvents
*
Regular
- balance : real
- nTicketsBought : nat
+ Regular(in m: String, in p: String, in fName: String, in lName: ...
+ isAdmin(): bool
+ getTicketsBought(): nat
+ getBalance(): nat
+ buyTickets(in nTickets: nat1, in priceTicket: real): Void
+String
- firstName
1
- lastName
1
1
1
Event
- eventId : nat
- id : nat
- dateStart : Date
- dateEnd : Date
- price : real
- totalTickets : nat1
- soldTickets : nat
+ Event(in newTitle: String, in newCategory: String, in newStart: Da...
+ getID(): nat
+ getTitle(): String
+ getCategory(): String
+ getState(): State
+ getDateStart(): Date
+ getDateEnd(): Date
+ getDescription(): String
+ getPrice(): real
+ getCity(): String
+ getTotalTickets(): nat1
+ getSoldTickets(): nat
+ getStats(): real
+ getProfit(): real
+ buy(in nTickets: nat): Void
+ validateDates(in startDate: Date, in endDate: Date): bool
+String
+Date
+State
+State_AvailableSoldOut_1306372164
- title
1
- category
1
- description
1
- city
1
1
- state
1
1
1
1
1
AgendaTest
+ AgendaTest(): AgendaTest
- testCreateAgenda(): Void
- testAddUser(): Void
- testLoginAdmin(): Void
- testAddEvent(): Void
- testFindByCity(): Void
- testFindByDistrict(): Void
- testFindByCategory(): Void
- testFindByDate(): Void
- testFindEvents(): Void
- testLoginRegular(): Void
- testBuyTickets(): Void
- testExistsEvent(): Void
- testProposeEvent(): Void
- testRejectProposedEvent(): Void
- testAcceptProposedEvent(): Void
- testMostPopularEvent(): Void
- testMostProfitableEvent(): Void
- testMostActiveUser(): Void
+ test(): Void
EventTest
+ EventTest(): Event...
- testCreateEvent(): ...
- testCreateEventLe...
+ test(): Void
Tests
# assertTrue(in arg: ...
+ test(): Void
UserTest
+ UserTest(): UserT...
- testCreateAdmin():...
- testCreateRegular(...
+ test(): Void

3 Modelo Formal VDM++

3.1 Classe Agenda

```
class Agenda

  types

    public String = seq of char;
    public MultiMap = map String to set of String;

  instance variables

    -- Agenda's list of categories
    public categories: set of String := {};

    -- Agenda's list of locations (multimap district->cities)
    public locations: MultiMap := {|->};

    -- Agenda's map of users
    public users: map seq of char to User := {|->};

    -- Agenda's list of events
    public events: set of Event := {};

    -- Logged in user. If nil means there is no user logged in.
    public loggedInUser: [User] := nil;
    -- TODO: depois por a private

    -- Agenda's list of proposed events waiting for admin's approval
    public proposedEvents: set of Event := {};

  operations

    -- Constructor
    public Agenda: () ==> Agenda
    Agenda() == (
      categories :=  {"Concertos", "Exposicoes", "Gastronomia", "Moda"
   , "Desporto", "Natureza"};
      locations := {"Porto" |-> {"Porto", "Matosinhos", "Maia", "Vila
   Nova de Gaia"},
                    "Lisboa" |-> {"Lisboa", "Amadora", "Cascais", "
   Sintra"},
                    "Faro" |-> {"Faro", "Albufeira", "Portimao"}};

      return self;
    );

    -- Adds user to Agenda's list of users
    public addUser: User ==> ()
    addUser(user) == users := users ++ {user.getEmail() |-> user}
    pre user.getEmail() not in set dom users
    post user.getEmail() in set dom users;

    -- Logs the user in. Returns true if successful and updates the
   loggedInUser variable
    public login: String * String ==> bool
    login(email, password) == (
      dcl user: User := users(email);

      -- verifies if password matches to email
      if user.checkLogin(email, password)
      then (
          loggedInUser := user;
          return true;
      );
      return false;
    )
    pre email in set dom users and len password > 8
    post (RESULT = true and loggedInUser <> nil) or RESULT = false;

    -- *ADMINISTRATOR ONLY*

    -- Adds event to Agenda's list of events
    public addEvent: Event ==> ()
    addEvent(event) == events := events union {event}
    pre loggedInUser <> nil and loggedInUser.isAdmin() and
   existsCategory(event.getCategory()) and existsCity(event.getCity()
   ) and not existsEvent(event)
    post event in set events;

    -- Removes event from Agenda's list of proposed events
    public rejectProposedEvent: Event ==> ()
    rejectProposedEvent(event) == proposedEvents := proposedEvents \ {
   event}
    pre loggedInUser <> nil and loggedInUser.isAdmin() and event in
   set proposedEvents
    post event not in set proposedEvents;

    -- Adds event to Agenda's list of events and removes from Agenda's
    list of proposed events
    public acceptProposedEvent: Event ==> ()
    acceptProposedEvent(event) == (
      events := events union {event};
      proposedEvents := proposedEvents \ {event};
    )
```

```
  pre loggedInUser <> nil and loggedInUser.isAdmin() and event in
 set proposedEvents and event not in set events
  post event not in set proposedEvents and event in set events;

  -- Returns most popular event
  public mostPopularEvent: () ==> Event
  mostPopularEvent () == (
    dcl popularEvent: [Event] := nil;
    dcl value: real := 0;

    for all event in set events do (
      if(event.getStats() >= value)
      then (
        value := event.getStats();
        popularEvent := event;
      );
    );

    return popularEvent;
  )
  pre loggedInUser <> nil and loggedInUser.isAdmin() and events <>
 {};

  -- Returns most profitable event
  public mostProfitableEvent: () ==> Event
  mostProfitableEvent () == (
    dcl profitableEvent: [Event] := nil;
    dcl value: real := 0;

    for all event in set events do (
      if(event.getProfit() >= value)
      then (
        value := event.getProfit();
        profitableEvent := event;
      );
    );

    return profitableEvent;
  )
  pre loggedInUser <> nil and loggedInUser.isAdmin() and events <>
 {};

  -- Returns most active user
  public mostActiveUser: () ==> User
  mostActiveUser () == (
    dcl allUsers: set of User := rng users;
    dcl activeUser: [User] := nil;
    dcl value: nat := 0;

```

```
      for all user in set allUsers do (
        if(isofclass(Regular, user))
        then (
          if(user.getTicketsBought() >= value)
          then (
            value := user.getTicketsBought();
            activeUser := user;
          );
        );
      );

      return activeUser;
    )
    pre loggedInUser <> nil and loggedInUser.isAdmin() and users <>
   {|->};

    -- *REGULAR USER ONLY*

    -- Adds event to Agenda's list of proposed event
    public proposeEvent: Event ==> ()
    proposeEvent(event) == proposedEvents := proposedEvents union {
   event}
    pre loggedInUser <> nil and not loggedInUser.isAdmin() and
   existsCategory(event.getCategory()) and existsCity(event.getCity()
   ) and not existsEvent(event)
    post event in set proposedEvents;

    -- Returns if the purchase was successful
    public buyTicket: nat * nat ==> bool
    buyTicket(eventID, nTickets) == (
      dcl eventBought : Event;
      for all event in set events do (
          if event.getID() = eventID
          then eventBought := event;
      );
      -- verifies if there are enough tickets to sell
      if eventBought.getTotalTickets() >= eventBought.getSoldTickets()
    + nTickets and
          loggedInUser.getBalance() >= nTickets * eventBought.getPrice
   ()
      then (
        loggedInUser.buyTickets(nTickets, eventBought.getPrice());
        eventBought.buy(nTickets);
        return true;
      )
      else return false
    )
    pre loggedInUser <> nil and not loggedInUser.isAdmin();

```

```
  -- *FIND EVENTS*

  -- Returns a set of events based on filters
  public findEvents: String * String * String * [Event`Date] ==> set
  of Event
  findEvents(city, district, category, date) == (
    dcl foundEvents : set of Event := {};

    if city <> ""
      then foundEvents := findByCity(city)
    elseif district <> ""
      then foundEvents := findByDistrict(district);

    if category <> ""
    then (
      if city = "" and district = ""
      then foundEvents := findByCategory(category)
      else foundEvents := foundEvents inter findByCategory(category)
;
    );

    if date <> nil
    then (
      if city = "" and district = "" and category = ""
      then foundEvents := findByDate(date)
      else foundEvents := foundEvents inter findByDate(date);
    );

    return foundEvents;
  )
  pre loggedInUser <> nil and events <> {} and
      not (city <> "" and district <> "") and
      not (city = "" and district = "" and category = "" and date =
nil)
  post (if city <> "" then forall e in set RESULT & e.getCity() =
city else true) and
       (if district <> "" then forall e in set RESULT &
existsCityInDistrict(e.getCity(), district) else true) and
       (if category <> "" then forall e in set RESULT & e.
getCategory() = category else true) and
       (if date <> nil then forall e in set RESULT & wantedDate(date
, e.getDateStart(), e.getDateEnd()) else true);

  -- by city
  public findByCity: String ==> set of Event
  findByCity(city) == (
    dcl cityEvents : set of Event := {};
    for all event in set events do (
        if event.getCity() = city
```

```
          then cityEvents := cityEvents union {event}
      );
      return cityEvents;
    )
    pre loggedInUser <> nil and existsCity(city) and events <> {}
    post forall e in set RESULT & e.getCity() = city;

    -- by district
    public findByDistrict: String ==> set of Event
    findByDistrict(district) == (
      dcl districtEvents : set of Event := {};
      if existsDistrict(district)
      then (
        for all event in set events do (
          if existsCityInDistrict(event.getCity(), district)
          then districtEvents := districtEvents union {event}
        );
        return districtEvents;
      )
      else return {}
    )
    pre loggedInUser <> nil and events <> {}
    post forall e in set RESULT & existsCityInDistrict(e.getCity(),
  district);


    -- by category
    public findByCategory: String ==> set of Event
    findByCategory(category) == (
      dcl categoryEvents : set of Event := {};
      for all event in set events do (
          if category = event.getCategory()
          then categoryEvents := categoryEvents union {event}
      );
      return categoryEvents;
    )
    pre loggedInUser <> nil and existsCategory(category) and events <>
    {}
    post forall e in set RESULT & e.getCategory() = category;

    -- by date
    public findByDate: Event`Date ==> set of Event
    findByDate(date) == (
      dcl dateEvents : set of Event := {};
      for all event in set events do (
          if wantedDate(date, event.getDateStart(), event.getDateEnd()
  )
          then dateEvents := dateEvents union {event}
      );
      return dateEvents;
    )
    pre loggedInUser <> nil and events <> {}
    post forall e in set RESULT & wantedDate(date, e.getDateStart(), e
    .getDateEnd());

    -- *AUX*

    -- Returns if events exists in the agenda's list of events
    public pure existsEvent: Event ==> bool
    existsEvent(event) == (
      for all e in set events do(
        if (event.getTitle() = e.getTitle() and
            event.getCategory() = e.getCategory() and
            event.getDateStart() = e.getDateStart() and
            event.getDateEnd() = e.getDateEnd() and
            event.getCity() = e.getCity())
        then return true
      );
      return false;
    );

    -- Returns if city exists in the agenda's list of cities
    public pure existsCity: String ==> bool
    existsCity(city) == (
      dcl citiesSet : set of set of String := rng locations;
      dcl cities: set of String := {};
      for all cs in set citiesSet do(
        cities := cities union cs
      );
      return city in set cities;
    )
    pre locations <> {|->};

    -- Returns if district exists in the agenda's list of districts
    public pure existsDistrict: String ==> bool
    existsDistrict(district) == (
      dcl districts : set of String := dom locations;
      return district in set districts;
    )
    pre locations <> {|->};

    -- Returns if city exists in a certain district
    public pure existsCityInDistrict: String * String ==> bool
    existsCityInDistrict(city, district) == (
      dcl districtCities : set of String := locations(district);
      return city in set districtCities;
    )
    pre existsDistrict(district) and locations <> {|->};

    -- Returns if category exists in the agenda's list of categories
    public pure existsCategory: String ==> bool
    existsCategory(category) == return category in set categories
    pre categories <> {};

    -- Returns if date belongs to the interval [dateStart,dateEnd]
    public pure wantedDate: Event`Date * Event`Date * Event`Date ==>
   bool
    wantedDate(date, dateStart,dateEnd) == (
      dcl natDateStart : nat := dateStart.year * 10000 + dateStart.
   month * 100 + dateStart.day;
      dcl natDateEnd : nat := dateEnd.year * 10000 + dateEnd.month *
   100 + dateEnd.day;
      dcl natDate : nat := date.year * 10000 + date.month * 100 + date
   .day;

      return (natDateStart <= natDate) and (natDate <= natDateEnd);
    );

end Agenda
```

3.2 Classe Event

```
class Event

  types

    public String = seq of char;

    -- Represents a date
    public Date :: day : nat1
                   month: nat1
                   year : nat1

    -- Ensures a valid date
    inv date == if date.year mod 400 = 0 or (date.year mod 100 <> 0
   and date.year mod 4 = 0)
                 then date.month <= 12 and date.day <= [31, 29, 31, 30,
    31, 30, 31, 31, 30, 31, 30, 31](date.month)
                 else date.month <= 12 and date.day <= [31, 28, 31, 30,
    31, 30, 31, 31, 30, 31, 30, 31](date.month);

    -- Represents the event's state
    public State = <Available> | <SoldOut>;

  instance variables
```

```

    private static eventId : nat := 1;

    -- Event's id
    private id : nat;

    -- Event's title
    private title: String;

    -- Event's category
    private category: String;

    -- Event's state
    private state: State;

    -- Event's starting date
    private dateStart: Date;

    -- Event's ending date
    private dateEnd: Date;

    -- Event's description
    private description: String;

    -- Event's price
    private price: real;

    -- Event's city
    private city: String;

    -- Event's total amount of tickets
    private totalTickets: nat1;

    -- Event's amount of sold tickets
    private soldTickets: nat;

    -- Ensures inexistence of overbooking
    inv totalTickets >= soldTickets;

  operations

    -- Constructor
    public Event: String * String * Date * Date * String * real *
   String * nat1 ==> Event
    Event(newTitle, newCategory, newStart, newEnd, newDescription,
   newPrice, newCity, newTotal) == (
      id := eventId;
      eventId := eventId + 1;

      title := newTitle;
      category := newCategory;
      state := <Available>;
      dateStart := newStart;
      dateEnd := newEnd;
      description := newDescription;
      price := newPrice;
      city := newCity;
      totalTickets := newTotal;
      soldTickets := 0;

      return self;
    )
    pre len newTitle > 0 and validateDates(newStart, newEnd)
    post title = newTitle;

    -- Returns event's id
    public pure getID: () ==> nat
    getID () == return id;

    -- Returns event's title
    public pure getTitle: () ==> String
    getTitle () == return title;

    -- Returns event's category
    public pure getCategory: () ==> String
    getCategory () == return category;

    -- Returns event's state
    public pure getState: () ==> State
    getState () == return state;

    -- Returns event's starting date
    public pure getDateStart: () ==> Date
    getDateStart () == return dateStart;

    -- Returns event's ending date
    public pure getDateEnd: () ==> Date
    getDateEnd () == return dateEnd;

    -- Returns event's description
    public pure getDescription: () ==> String
    getDescription () == return description;

    -- Returns event's price
    public pure getPrice: () ==> real
    getPrice () == return price;

    -- Returns event's city
    public pure getCity: () ==> String
    getCity () == return city;

    -- Returns event's total tickets
    public pure getTotalTickets: () ==> nat1
    getTotalTickets () == return totalTickets;

    -- Returns event's sold tickets
    public pure getSoldTickets: () ==> nat
    getSoldTickets () == return soldTickets;

    -- Returns event's percentage of tickets sold
    public pure getStats: () ==> real
    getStats () == return soldTickets * 100 / totalTickets;

    -- Returns event's total money raised
    public pure getProfit: () ==> real
    getProfit () == return soldTickets * price;

    -- Updates event's sold tickets and state if needed
    public buy: nat ==> ()
    buy (nTickets) == (
      soldTickets := soldTickets + nTickets;
      if(soldTickets = totalTickets)
      then state := <SoldOut>;
    )
    pre totalTickets >= soldTickets + nTickets and state = <Available>
    post soldTickets = soldTickets~ + nTickets;

    -- *AUX*
    -- Verifies if the start date is before the end date
    public pure validateDates: Event`Date * Event`Date ==> bool
    validateDates(startDate,endDate) == (
      dcl natDateStart : nat := startDate.year * 10000 + startDate.
   month * 100 + startDate.day;
      dcl natDateEnd : nat := endDate.year * 10000 + endDate.month *
   100 + endDate.day;

      return (natDateStart <= natDateEnd);
    );

end Event
```

3.3 Classe User

```
class User

```

```
 types

   public String = seq of char;

 instance variables

   -- User's email
   protected email: String := "";

   -- User's password
   protected password: String := "";

 operations

   -- Constructor
   protected UserInit : String * String ==> ()
   UserInit(m, p) == (
     email := m;
     password := p;
   )
   pre len m >= 5 and len p >= 8
   post email = m and password = p;

   -- Returns the user email
   public pure getEmail: () ==> String
   getEmail () == return email ;

   -- Returns if user is admin
   public pure isAdmin: () ==> bool
   isAdmin() == is subclass responsibility;

   -- Checks if the email and password combination matches
   public checkLogin: String * String ==> bool
   checkLogin(m, p) == return email = m and password = p
   post RESULT = (email = m and password = p);

   -- Returns number of bought tickets
   public pure getTicketsBought: () ==> nat
   getTicketsBought() == is subclass responsibility;

   -- Returns balance
   public pure getBalance: () ==> nat
   getBalance() == is subclass responsibility;

   -- Makes a purchase
   public buyTickets: nat1 * real ==> ()
   buyTickets(nTickets, priceTicket) == is subclass responsibility;

end User
```

3.4 Classe Admin

```
class Admin is subclass of User

  operations

    -- Constructor
    public Admin : String * String ==> Admin
    Admin(mail, passw) == (
      UserInit(mail, passw);
      return self;
    );

    -- Returns if user is admin
    public pure isAdmin: () ==> bool
    isAdmin() == return true;

    -- Returns number of bought tickets
    public pure getTicketsBought: () ==> nat
    getTicketsBought() == is not yet specified;

    -- Returns balance
    public pure getBalance: () ==> nat
    getBalance() == is not yet specified;

    -- Makes a purchase
    public buyTickets: nat1 * real ==> ()
    buyTickets(nTickets, priceTicket) == is not yet specified;

end Admin
```

3.5 Classe Regular

```
class Regular is subclass of User

  types

      public String = seq of char;

  instance variables

    private firstName: String;
    private lastName: String;
    private balance: real;
    private nTicketsBought: nat;

  operations

    -- Constructor
    public Regular : String * String * String * String * real * real
   ==> Regular
    Regular(m, p, fName, lName, b, nt) == (
      UserInit(m, p);
      firstName := fName;
      lastName := lName;
      balance := b;
      nTicketsBought := nt;
      return self;
    );

    -- Returns if user is admin
    public pure isAdmin: () ==> bool
    isAdmin() == return false;

    -- Returns number of bought tickets
    public pure getTicketsBought: () ==> nat
    getTicketsBought() == return nTicketsBought;

    -- Returns balance
    public pure getBalance: () ==> nat
    getBalance() == return balance;

    -- Makes a purchase
    public buyTickets: nat1 * real ==> ()
    buyTickets(nTickets, priceTicket) == (
      balance := balance - (nTickets * priceTicket);
      nTicketsBought := nTicketsBought + nTickets
    )
    pre balance >= nTickets * priceTicket
    post balance = balance~ - (nTickets * priceTicket);

end Regular
```

4 Validação do Modelo

4.1 Classes de Teste

4.1.1 Classe Tests

```
class Tests
  instance variables

  operations

    protected assertTrue: bool ==> ()
    assertTrue(arg) == return
    pre arg;

    public test: () ==> ()
    test() == (
      dcl agendaTest: AgendaTest := new AgendaTest();
      dcl eventTest: EventTest := new EventTest();
      dcl userTest: UserTest := new UserTest();
      agendaTest.test();
      eventTest.test();
      userTest.test();
    );

end Tests
```

4.1.2 Classe AgendaTest

```
class AgendaTest is subclass of Tests

  instance variables
    agenda: Agenda := new Agenda();
    userAdmin: Admin := new Admin("julieta@gmail.com", "julieta12345")
   ;
    user1: Regular := new Regular("sofia@gmail.com", "sofia12345", "
   Sofia", "Silva", 500, 30);
    user2: Regular := new Regular("bibi@gmail.com", "bibi12345", "
   Beatriz", "Baldaia", 200, 5);
    user3: Regular := new Regular("carlos@gmail.com", "carlos12345", "
   Carlos", "Freitas", 800, 10);
    user4: Regular := new Regular("vicente@gmail.com", "vicente12345",
    "Vicente", "Espinha", 100, 50);
    event1: Event := new Event("Twenty One Pilots", "Concertos",
  mk_Event`Date(17,3,2019), mk_Event`Date(17,3,2019), "Twenty One Pilots, o aclamado duo norte-americano constituido por Tyler Joseph e Josh Dun. The Bandito Tour sera a digressao mundial de apresentacao do album, com a estreia da banda ao vivo em Portugal, dia 17 de Marco, na Altice Arena.", 42, "Lisboa", 20000);
    event2: Event := new Event("EXO", "Concertos", mk_Event`Date(20,5,2019), mk_Event`Date(20,5,2019), "EXO e um grupo masculino sino-coreano de Seul. Estreia em Portugal dia 20 de Maio, na Altice Arena.", 30, "Lisboa", 20000);
    event3: Event := new Event("Aberturas: Tom Emerson em conversa com o arquivo Alvaro Siza", "Exposicoes", mk_Event`Date(6,1,2019), mk_Event`Date(6,2,2019), "Visita orientada a exposicao por Matilde Seabra, educadora. Localizacao: Biblioteca de Serralves", 2.5, "Porto", 200);
    event4: Event := new Event("Brunch Mercearia Bio", "Gastronomia", mk_Event`Date(5,1,2019), mk_Event`Date(5,1,2019), "Ir as compras e aproveitar para tomar um pequeno-almoco reforcado ou antecipar a hora do almoco e a proposta do nosso Brunch, servido entre as 11h e as 16h.", 7.8, "Cascais", 30);
    event5: Event := new Event("Porto VS Belenenses", "Desporto", mk_Event`Date(30,1,2019), mk_Event`Date(30,1,2019), "Lorem ipsum.", 35, "Porto", 50000);
    event6: Event := new Event("Leixoes VS Famalicao", "Desporto", mk_Event`Date(7,4,2020), mk_Event`Date(7,4,2020), "Lorem ipsum.", 12.5, "Matosinhos", 2);

    proposed1: Event := new Event("Workshop Comida Saudavel daTerra", "Gastronomia", mk_Event`Date(15,7,2019), mk_Event`Date(15,7,2019), "Workshop de comida saudavel, daTerra baixa, 15h.", 5, "Porto", 20);
    proposed2: Event := new Event("Cozinhar Nunca Foi Facil", "Gastronomia", mk_Event`Date(20,12,2019), mk_Event`Date(20,12,2019), "Lorem.", 10, "Lisboa", 35);

  operations

    public AgendaTest: () ==> AgendaTest
    AgendaTest() == (
      return self;
    );

    -- Creates agenda and verifies the parameters
    private testCreateAgenda: () ==> ()
    testCreateAgenda() == (
      dcl a: Agenda := new Agenda();
      assertTrue(a.categories = {"Concertos", "Exposicoes", "Gastronomia", "Moda", "Desporto", "Natureza"});
      assertTrue(a.locations = {"Porto" |-> {"Porto", "Matosinhos", "Maia", "Vila Nova de Gaia"}, "Lisboa" |-> {"Lisboa", "Amadora", "
Cascais", "Sintra"}, "Faro" |-> {"Faro", "Albufeira", "Portimao"
}});
      assertTrue(a.loggedInUser = nil);
    );

    private testAddUser: () ==> ()
    testAddUser() == (
      agenda.addUser(userAdmin);
      agenda.addUser(user1);
      agenda.addUser(user2);
      agenda.addUser(user3);
      agenda.addUser(user4);
      assertTrue(agenda.users <> {|->});
    );

    private testLoginAdmin: () ==> ()
    testLoginAdmin() == (
      dcl outcome: bool;
      agenda.loggedInUser := nil;
      -- fail
      outcome:= agenda.login("julieta@gmail.com", "person12345");
      assertTrue(outcome = false);
      assertTrue(agenda.loggedInUser = nil);

      -- success
      outcome := agenda.login("julieta@gmail.com", "julieta12345");
      assertTrue(outcome = true);
      assertTrue(agenda.loggedInUser <> nil);
      assertTrue(agenda.loggedInUser = userAdmin);
    );

    private testAddEvent: () ==> ()
    testAddEvent() == (
      agenda.addEvent(event1);
      agenda.addEvent(event2);
      agenda.addEvent(event3);
      agenda.addEvent(event4);
      agenda.addEvent(event5);
      agenda.addEvent(event6);
      assertTrue(agenda.events <> {});
      assertTrue(agenda.events = {event1, event2, event3, event4,
event5, event6});
    );

    private testFindByCity: () ==> ()
    testFindByCity() == (
      dcl cityEvents : set of Event := {};
      --fail
      cityEvents := agenda.findByCity("Faro");
      assertTrue(cityEvents = {});

      --success
      cityEvents := agenda.findByCity("Porto");
      assertTrue(cityEvents <>{});
    );

    private testFindByDistrict: () ==> ()
    testFindByDistrict() == (
      dcl districtEvents : set of Event := {};
      --fail
      districtEvents := agenda.findByDistrict("Faro");
      assertTrue(districtEvents = {});

      districtEvents := agenda.findByDistrict("Braganca");
      assertTrue(districtEvents = {});

      --success
      districtEvents := agenda.findByDistrict("Lisboa");
      assertTrue(districtEvents <> {});
    );

    private testFindByCategory: () ==> ()
    testFindByCategory() == (
      dcl categoryEvents : set of Event := {};
      --fail
      categoryEvents := agenda.findByCategory("Moda");
      assertTrue(categoryEvents = {});

      --success
      categoryEvents := agenda.findByCategory("Desporto");
      assertTrue(categoryEvents <>{});
    );

    private testFindByDate: () ==> ()
    testFindByDate() == (
      dcl dateEvents : set of Event := {};
      --fail
      dateEvents := agenda.findByDate(mk_Event`Date(31,1,2018));
      assertTrue(dateEvents = {});

      --success
      dateEvents := agenda.findByDate(mk_Event`Date(5,1,2019));
      assertTrue(dateEvents <>{});

    );

    private testFindEvents: () ==> ()
    testFindEvents() == (
      dcl eventsFound : set of Event := {};
      --fail
      eventsFound := agenda.findEvents ("Matosinhos", "", "Moda",
   mk_Event`Date(7,4,2020));
      assertTrue(eventsFound = {});

      --success
      eventsFound := agenda.findEvents("Matosinhos", "", "", nil);
      assertTrue(eventsFound <>{});

      eventsFound := agenda.findEvents("", "Porto", "", nil);
      assertTrue(eventsFound <>{});

      eventsFound := agenda.findEvents("", "", "Desporto", nil);
      assertTrue(eventsFound <>{});

      eventsFound := agenda.findEvents("", "", "", mk_Event`Date
   (7,4,2020));
      assertTrue(eventsFound <>{});

      eventsFound := agenda.findEvents("Matosinhos", "", "Desporto",
   mk_Event`Date(7,4,2020));
      assertTrue(eventsFound <>{});

    );

    private testLoginRegular: () ==> ()
    testLoginRegular() == (
      dcl outcome: bool;
      agenda.loggedInUser := nil;
      -- fail
      outcome := agenda.login("sofia@gmail.com", "person12345");
      assertTrue(outcome = false);
      assertTrue(agenda.loggedInUser = nil);

      -- success
      outcome := agenda.login("sofia@gmail.com", "sofia12345");
      assertTrue(outcome = true);
      assertTrue(agenda.loggedInUser <> nil);
      assertTrue(agenda.loggedInUser = user1);
    );

    private testBuyTickets: () ==> ()
    testBuyTickets() == (
      --fail
      dcl outcome: bool := agenda.buyTicket(1, 20001);
      assertTrue(outcome = false);
      assertTrue(event1.getSoldTickets() = 0);
      assertTrue(agenda.loggedInUser.getTicketsBought() = 30);

      --success
      outcome := agenda.buyTicket(6, 2);
      assertTrue(outcome = true);
      assertTrue(event6.getSoldTickets() = 2);
      assertTrue(agenda.loggedInUser.getTicketsBought() = 32);
    );

    private testExistsEvent: () ==> ()
    testExistsEvent() == (
      --fail
      dcl outcome: bool := agenda.existsEvent(event1);
      assertTrue(outcome = true);

      --success
      outcome := agenda.existsEvent(proposed2);
      assertTrue(outcome = false);
    );

    private testProposeEvent: () ==> ()
    testProposeEvent() == (
      agenda.proposeEvent(proposed1);
      agenda.proposeEvent(proposed2);
      assertTrue(agenda.proposedEvents <> {});
      assertTrue(agenda.proposedEvents = {proposed1, proposed2});
    );

    private testRejectProposedEvent: () ==> ()
    testRejectProposedEvent() == (
      agenda.rejectProposedEvent(proposed2);
      assertTrue(agenda.proposedEvents = {proposed1});
    );

    private testAcceptProposedEvent: () ==> ()
    testAcceptProposedEvent() == (
      agenda.acceptProposedEvent(proposed1);
      assertTrue(agenda.proposedEvents = {});
      assertTrue(agenda.events = {event1, event2, event3, event4,
   event5, event6, proposed1});
    );

    private testMostPopularEvent: () ==> ()
    testMostPopularEvent() == (
      dcl outcome: Event := agenda.mostPopularEvent();
      assertTrue(outcome = event6);
    );

    private testMostProfitableEvent: () ==> ()
    testMostProfitableEvent() == (
      dcl outcome: Event := agenda.mostProfitableEvent();
      assertTrue(outcome = event6);
    );

    private testMostActiveUser: () ==> ()
    testMostActiveUser() == (
      dcl outcome: User := agenda.mostActiveUser();
      assertTrue(outcome = user4);
    );

    public test: () ==> ()
    test() == (
      testCreateAgenda();
      testAddUser();
      testLoginAdmin();
      testAddEvent();
      testFindByCity();
      testFindByDistrict();
      testFindByCategory();
      testFindByDate();
      testFindEvents();
      testLoginRegular();
      testBuyTickets();
      testExistsEvent();
      testProposeEvent();
      testLoginAdmin();
      testRejectProposedEvent();
      testAcceptProposedEvent();
      testMostPopularEvent();
      testMostProfitableEvent();
      testMostActiveUser();
    );

end AgendaTest
```

4.1.3 Classe EventTest

```
class EventTest is subclass of Tests

  operations
    public EventTest: () ==> EventTest
    EventTest() == (
      return self;
    );

    -- Creates an event, verifies the parameters and simulates a
   purchase
```

```
    private testCreateEvent: () ==> ()
    testCreateEvent() == (
      dcl event: Event := new Event("Twenty One Pilots", "Concertos",
  mk_Event`Date(17,3,2019), mk_Event`Date(17,3,2019), "Lorem ipsum
  dolor sit amet.", 42, "Lisboa", 100);
      assertTrue(event.getTitle() = "Twenty One Pilots");
      assertTrue(event.getCategory() = "Concertos");
      assertTrue(event.getState() = <Available>);
      assertTrue(event.getDateStart() = mk_Event`Date(17,3,2019));
      assertTrue(event.getDateEnd() = mk_Event`Date(17,3,2019));
      assertTrue(event.getDescription() = "Lorem ipsum dolor sit amet.
  ");
      assertTrue(event.getPrice() = 42);
      assertTrue(event.getCity() = "Lisboa");
      assertTrue(event.getTotalTickets() = 100);
      assertTrue(event.getSoldTickets() = 0);

      event.buy(100);
      assertTrue(event.getSoldTickets() = 100);
      assertTrue(event.getState() = <SoldOut>);
      assertTrue(event.getStats() = 100);
      assertTrue(event.getProfit() = 4200);
    );

    -- Creates an event, leap year
    private testCreateEventLeap: () ==> ()
    testCreateEventLeap() == (
      dcl event: Event := new Event("Twenty One Pilots", "Concertos",
  mk_Event`Date(29,2,2020), mk_Event`Date(29,2,2020), "Lorem ipsum
  dolor sit amet.", 42, "Lisboa", 100);
      assertTrue(event.getDateStart() = mk_Event`Date(29,2,2020));
      assertTrue(event.getDateEnd() = mk_Event`Date(29,2,2020));
    );

    public test: () ==> ()
    test() == (
      testCreateEvent();
      testCreateEventLeap();
    );

end EventTest
```

4.1.4 Classe UserTest

```
class UserTest is subclass of Tests

  operations

    public UserTest: () ==> UserTest
    UserTest() == (
      return self;
    );

    -- Creates a user: admin
    private testCreateAdmin: () ==> ()
    testCreateAdmin() == (
      dcl user: User := new Admin("julieta@gmail.com", "julieta12345");
      assertTrue(user.getEmail() = "julieta@gmail.com");
      assertTrue(user.isAdmin() = true);
      assertTrue(user.checkLogin("julieta@gmail.com", "julieta12345") = true);
    );

    -- Creates a user: regular and tests buy tickets
    private testCreateRegular: () ==> ()
    testCreateRegular() == (
      dcl user: User := new Regular("sofia@gmail.com", "sofia12345", "Sofia", "Silva", 500, 30);
      assertTrue(user.getEmail() = "sofia@gmail.com");
      assertTrue(user.isAdmin() = false);
      assertTrue(user.checkLogin("sofia@gmail.com", "sofia12345") = true);
      assertTrue(user.getTicketsBought() = 30);

      user.buyTickets(2, 50);
      assertTrue(user.getBalance() = 400);
      assertTrue(user.getTicketsBought() = 32);
    );

    public test: () ==> ()
    test() == (
      testCreateAdmin();
      testCreateRegular();
    );

end UserTest
```

4.2 Coverage

4.2.1 Classe Agenda

Function or operation	Line	Coverage	Calls
Agenda	32	100.0%	2
acceptProposedEvent	79	100.0%	1
addEvent	67	100.0%	6
addUser	43	100.0%	5
buyTicket	154	100.0%	1
existsCategory	311	100.0%	26
existsCity	283	100.0%	39
existsCityInDistrict	303	100.0%	54
existsDistrict	295	100.0%	31
existsEvent	269	100.0%	1
findByCategory	241	100.0%	5
findByCity	210	100.0%	5
findByDate	254	100.0%	10
findByDistrict	223	100.0%	4
findEvents	176	100.0%	6
login	49	100.0%	3
mostActiveUser	124	100.0%	3
mostPopularEvent	88	100.0%	6
mostProfitableEvent	106	100.0%	1
proposeEvent	148	100.0%	2
rejectProposedEvent	73	100.0%	1
wantedDate	316	100.0%	36
Agenda.vdmpp		100.0%	248

4.2.2 Classe Event

Function or operation	Line	Coverage	Calls
Event	63	100.0%	10
buy	137	100.0%	2
getCategory	93	100.0%	50
getCity	117	100.0%	75
getDateEnd	105	100.0%	40
getDateStart	101	100.0%	40
getDescription	109	100.0%	1
getID	85	100.0%	12
getPrice	113	100.0%	3
getProfit	133	100.0%	14
getSoldTickets	125	100.0%	6
getState	97	100.0%	2
getStats	129	100.0%	14
getTitle	89	100.0%	69
getTotalTickets	121	100.0%	3
validateDates	148	100.0%	10
Event.vdmpp		100.0%	351

4.2.3 Classe User

Function or operation	Line	Coverage	Calls
UserInit	18	100.0%	7
buyTickets	48	100.0%	2
checkLogin	35	100.0%	8
getBalance	44	100.0%	2
getEmail	27	100.0%	17
getTicketsBought	40	100.0%	2
isAdmin	31	100.0%	2
User.vdmpp		100.0%	40

4.2.4 Classe Regular

Function or operation	Line	Coverage	Calls
Regular	17	100.0%	5
buyTickets	40	100.0%	2
getBalance	36	100.0%	2
getTicketsBought	32	100.0%	11
isAdmin	28	100.0%	5
Regular.vdmpp		100.0%	25

4.2.5 Classe Admin

Function or operation	Line	Coverage	Calls
Admin	6	100.0%	2
buyTickets	25	0.0%	0
getBalance	21	0.0%	0
getTicketsBought	17	0.0%	0
isAdmin	13	100.0%	12
Admin.vdmpp		100.0%	14

Nota: Algumas destas operações têm uma *coverage* de 0% devido à utilização de *is not yet specified.* Tratam-se se operações que um utilizador do tipo *Admin* não deveria ter, mas é obrigatória a sua presença por uma questão de herança de classes.

5 Verificação do Modelo

5.1 Exemplo de Verificação de um Domínio

5.1.1 Pré-condição

```
class Agenda
-- Returns if city exists in a certain district
    public pure existsCityInDistrict: String * String ==> bool
    existsCityInDistrict(city, district) == (
      dcl districtCities : set of String := locations(district);
      return city in set districtCities;
    )
    pre existsDistrict(district) and locations <> {|->};
```

A pré-condição em causa é *existsDistric* que requer a existência do distrito no sistema antes de se verificar se a cidade pertence a esse mesmo distrito.

5.1.2 Proof Obligation gerada pelo Overture

No.	Nome da Proof Obligation	Tipo
20	Agenda‘existsCityInDistrict(String, String), districCities	Legal Map Application

```
(forall city:Agenda‘String, district:Agenda‘String & ((existsDistrict(
   district) and (locations <> {|->})) => (district in set (dom
   locations))))
```

Com a expressão *(district in set (dom locations))*, a ferramenta Overture verifica se o distrito faz parte do set de keys do map *locations*, ou seja, verifica se o distrito existe no sistema.

5.1.3 Proof Sketch

```
class Agenda
  public findByDistrict: String ==> set of Event
    findByDistrict(district) == (
      dcl districtEvents : set of Event := {};
      if existsDistrict(district)
      then (
 7        for all event in set events do (
 8          if existsCityInDistrict(event.getCity(), district)
 9          then districtEvents := districtEvents union {event}
10        );
11        return districtEvents;
12      )
13      else return {}
14    )
15    pre loggedInUser <> nil and events <> {}
16    post forall e in set RESULT & existsCityInDistrict(e.getCity(),
    district);
```

Antes da chamada à função *existsCityInDistrict* na linha 8, é feita a verificação *exists-District* na linha 5, deste modo a pré-condição dentro da respetiva função é sempre preservada.

5.2 Exemplo de Verificação de uma Invariante

5.2.1 Invariante

```
class Event
  -- Ensures inexistence of overbooking
  inv totalTickets >= soldTickets;
```

O invariante em causa evita o overbooking de eventos, evitando que o número de bilhetes vendidos nunca seja maior do que o número de bilhetes disponíveis.

5.2.2 Proof Obligation gerada pelo Overture

No.	Nome da Proof Obligation	Tipo
40	Event‘buy(nat)	State Invariant Holds

```
(forall nTickets:nat & (((totalTickets >= (soldTickets + nTickets))
    and (state = <Available>)) => ((totalTickets >= soldTickets) => (
    totalTickets >= (soldTickets + nTickets)))))
```

A ferramenta Overture verifica o invariante na função *buy*, visto que é nesta que a veracidade do invariante pode ser posta em causa.

5.2.3 Proof Sketch

```
1  class Agenda
2  -- Returns if the purchase was successful
3      public buyTicket: nat * nat ==> bool
4      buyTicket(eventID, nTickets) == (
5        dcl eventBought : Event;
6        for all event in set events do (
7            if event.getID() = eventID
8            then eventBought := event;
9        );
10       -- verifies if there are enough tickets to sell
11       if eventBought.getTotalTickets() >= eventBought.getSoldTickets()
     + nTickets and
12           loggedInUser.getBalance() >= nTickets * eventBought.getPrice
   ()
13       then (
14         loggedInUser.buyTickets(nTickets, eventBought.getPrice());
15         eventBought.buy(nTickets);
16         return true;
17       )
18       else return false
19     )
20     pre loggedInUser <> nil and not loggedInUser.isAdmin();
```

Antes da chamada à função crítica *buy* na linha 15, é feita a verificação do invariante na linha 11 com o objetivo de o preservar sempre.

6 Geração de Código

O código Java foi gerado com sucesso a partir do modelo VDM++.

6.1 Main

```java
package AgendaViral;

public class Main {

  public static void main(String[] args) {
    Agenda agenda = new Agenda();
    Interface gui = new Interface(agenda);
    gui.loginMenu();
  }
}
```

6.2 Interface

```java
package AgendaViral;

import java.util.Iterator;
import java.util.Scanner;

import org.overture.codegen.runtime.SetUtil;
import org.overture.codegen.runtime.VDMSet;

public class Interface {
  private Agenda agenda;
  Scanner scanner = new Scanner(System.in);

  public Interface(Agenda agenda) {
    this.agenda = agenda;

    Admin userAdmin = new Admin("julieta@gmail.com", "julieta12345");
    Regular user1 = new Regular("sofia@gmail.com", "sofia12345", "
   Sofia", "Silva", 500L, 30L);
    Regular user2 = new Regular("bibi@gmail.com", "bibi12345", "
   Beatriz", "Baldaia", 200L, 5L);
    Regular user3 = new Regular("carlos@gmail.com", "carlos12345", "
   Carlos", "Freitas", 800L, 10L);
    Regular user4 = new Regular("vicente@gmail.com", "vicente12345", "
   Vicente", "Espinha", 100L, 50L);
```

```java
 Event event1 = new Event("Twenty One Pilots", "Concertos", new Event.Date(17L, 3L, 2019L),
     new Event.Date(17L, 3L, 2019L), "The Bandito Tour, dia 17 de Marco, na Altice Arena.", 42L, "Lisboa",
     20000L);
 Event event2 = new Event("EXO", "Concertos", new Event.Date(20L, 5L, 2019L), new Event.Date(20L, 5L, 2019L),
     "Estreia em Portugal dia 20 de Maio, na Altice Arena.", 30L, "Lisboa", 20000L);
 Event event3 = new Event("Aberturas: Tom Emerson em conversa com o arquivo Alvaro Siza", "Exposicoes",
     new Event.Date(6L, 1L, 2019L), new Event.Date(6L, 2L, 2019L),
     "Visita orientada a exposicao por Matilde Seabra, educadora. Localizacao: Biblioteca de Serralves", 2.5,
     "Porto", 200L);
 Event event4 = new Event("Brunch Mercearia Bio", "Gastronomia", new Event.Date(5L, 1L, 2019L),
     new Event.Date(5L, 1L, 2019L),
     "Ir as compras e aproveitar para tomar um pequeno-almoco reforcado ou antecipar a hora do almoco e a proposta do nosso Brunch, servido entre as 11h e as 16h.",
     7.8, "Cascais", 30L);
 Event event5 = new Event("Porto VS Belenenses", "Desporto", new Event.Date(30L, 1L, 2019L),
     new Event.Date(30L, 1L, 2019L), "Lorem ipsum.", 35L, "Porto", 50000L);
 Event event6 = new Event("Leixoes VS Famalicao", "Desporto", new Event.Date(7L, 4L, 2020L),
     new Event.Date(7L, 4L, 2020L), "Lorem ipsum.", 12.5, "Matosinhos", 2000L);
 Event proposed1 = new Event("Workshop Comida Saudavel daTerra", "Gastronomia", new Event.Date(15L, 7L, 2019L),
     new Event.Date(15L, 7L, 2019L), "Workshop de comida saudavel, daTerra baixa, 15h.", 5L, "Porto", 20L);
 Event proposed2 = new Event("Cozinhar Nunca Foi Facil", "Gastronomia", new Event.Date(20L, 12L, 2019L),
     new Event.Date(20L, 12L, 2019L), "Lorem.", 10L, "Lisboa", 35L);

 agenda.addUser(userAdmin);
 agenda.addUser(user1);
 agenda.addUser(user2);
 agenda.addUser(user3);
 agenda.addUser(user4);

 agenda.login("julieta@gmail.com", "julieta12345");
 agenda.addEvent(event1);
 agenda.addEvent(event2);
 agenda.addEvent(event3);
```

```java
    agenda.addEvent(event4);
    agenda.addEvent(event5);
    agenda.addEvent(event6);

    agenda.login("carlos@gmail.com", "carlos12345");
    agenda.buyTicket(2, 5);

    agenda.login("sofia@gmail.com", "sofia12345");
    agenda.proposeEvent(proposed1);
    agenda.proposeEvent(proposed2);
    agenda.buyTicket(1, 5);

    loginMenu();
  }

  public void loginMenu() {
    System.out.println(" ------------------------------------------- ");
    System.out.println("|                   Login                   |");
    System.out.println(" ------------------------------------------- ");

    System.out.print(" > Email: ");
    String email = scanner.nextLine();

    System.out.print(" > Password: ");
    String password = scanner.nextLine();

    System.out.println("");

    if (agenda.login(email, password))
      mainMenu();
    else
      System.out.println(" > Error: login failed");
  }

  public void mainMenu() {
    if (agenda.loggedInUser.isAdmin())
      mainMenuAdmin();
    else
      mainMenuRegular();
  }

  /*
   * ADMIN
   */

  public void mainMenuAdmin() {
    System.out.println(" ------------------------------------------- "
    );
    System.out.println("|                AGENDA VIRAL               |"
    );
    System.out.println(" ------------------------------------------- "
    );
    System.out.println(" 1.  Add Event                    0.  Logout")
    ;
    System.out.println(" 2.  Proposed Events");
    System.out.println(" 3.  Find by District");
    System.out.println(" 4.  Find by City");
    System.out.println(" 5.  Find by Category");
    System.out.println(" 6.  Find by Date");
    System.out.println(" 7.  Find by Multiple Filters");
    System.out.println(" 8.  Most Popular Event");
    System.out.println(" 9.  Most Profitable Event");
    System.out.println(" 10. Most Active User");
    System.out.println(" ------------------------------------------- "
    );

    System.out.print(" > Option: ");
    int option = scanner.nextInt();
    scanner.nextLine();

    System.out.println("");

    switch (option) {
    case 1:
      addEventMenu();
      break;
    case 2:
      proposedEventsMenu();
      break;
    case 3:
      findByDistrictMenu();
      break;
    case 4:
      findByCityMenu();
      break;
    case 5:
      findByCategoryMenu();
      break;
    case 6:
      findByDateMenu();
      break;
    case 7:
      findByMultipleFiltersMenu();
      break;
    case 8:
```

```java
      mostPopularMenu();
      break;
    case 9:
      mostProfitableMenu();
      break;
    case 10:
      mostActiveMenu();
      break;
    case 0:
      loginMenu();
      break;
    default:
      loginMenu();
      break;
    }
  }

  public void addEventMenu() {
    System.out.println(" -------------------------------------------- ");
    System.out.println("|                 Add Event                  |");
    System.out.println(" -------------------------------------------- ");

    System.out.print(" > Title: ");
    String title = scanner.nextLine();

    System.out.print(" > Category: ");
    String category = scanner.nextLine();

    System.out.print(" > Starting Date[dd/mm/yy]: ");
    String startDate = scanner.nextLine();
    String[] part1 = startDate.split("/");

    int day1 = Integer.parseInt(part1[0]);
    int month1 = Integer.parseInt(part1[1]);
    int year1 = Integer.parseInt(part1[2]);

    System.out.print(" > Ending Date[dd/mm/yy]: ");
    String endDate = scanner.nextLine();
    String[] part2 = endDate.split("/");

    int day2 = Integer.parseInt(part2[0]);
    int month2 = Integer.parseInt(part2[1]);
    int year2 = Integer.parseInt(part2[2]);

    System.out.print(" > Description: ");
    String description = scanner.nextLine();

    System.out.print(" > Price: ");
    int price = scanner.nextInt();
    scanner.nextLine();

    System.out.print(" > City: ");
    String city = scanner.nextLine();

    System.out.print(" > Total Tickets: ");
    int tickets = scanner.nextInt();
    scanner.nextLine();

    Event event = new Event(title, category, new Event.Date(day1, month1, year1),
        new Event.Date(day2, month2, year2), description, price, city, tickets);

    agenda.addEvent(event);

    mainMenuAdmin();
  }

  public void proposedEventsMenu() {
    System.out.println(" ------------------------------------------- ");
    System.out.println("|               Proposed Events             |");
    System.out.println(" ------------------------------------------- ");
    for (Iterator iter = agenda.proposedEvents.iterator(); iter.hasNext();) {
      Event event = (Event) iter.next();
      System.out.println(" Id: " + event.getID());
      System.out.println(" Title: " + event.getTitle());

      if (iter.hasNext())
        System.out.println("");
    }

    if (agenda.proposedEvents.isEmpty())
      System.out.println("              No proposed events             ");

    System.out.println(" ------------------------------------------- ");
    System.out.println("                                   0. Return ");
    System.out.println(" ------------------------------------------- ");

    System.out.print(" > Event Id: ");
    int option = scanner.nextInt();
    scanner.nextLine();

    System.out.println("");

    if (option == 0) {
      mainMenu();
    }

    for (Iterator iter = agenda.proposedEvents.iterator(); iter.hasNext();) {
      Event event = (Event) iter.next();

      if (option == event.getID().intValue()) {
        proposedEventMenu(event);
      }
    }

    mainMenuAdmin();
  }

  public void proposedEventMenu(Event event) {
    System.out.println(" ------------------------------------------- ");
    System.out.println("|               Proposed Event             |");
    System.out.println(" ------------------------------------------- ");

    System.out.println(" Id: " + event.getID());
    System.out.println(" Title: " + event.getTitle());
    System.out.println(" Category: " + event.getCategory());
    System.out.println(" City: " + event.getCity());
    System.out.println(" Date: from " + event.getDateStart().day + "/" + event.getDateStart().month + "/"
        + event.getDateStart().year + " to " + event.getDateEnd().day + "/" + event.getDateEnd().month + "/"
        + event.getDateEnd().year);
    System.out.println(" Price: " + event.getPrice() + " euros");
    System.out.println(" Total Tickets: " + event.getTotalTickets() + " | Sold Tickets: " + event.getSoldTickets());
    System.out.println(" Descriprion: " + event.getDescription());

    System.out.println(" ------------------------------------------- ");

    System.out.print(" > Accept or Reject (A/R): ");
    String action = scanner.nextLine();

    System.out.println("");

    if (action.equals("A"))
      agenda.acceptProposedEvent(event);
    else if (action.equals("R"))
      agenda.rejectProposedEvent(event);

    mainMenuAdmin();
  }

  public void mostPopularMenu() {
    Event event = agenda.mostPopularEvent();

    System.out.println(" ------------------------------------------- "
   );
    System.out.println("|             Most Popular Event            |"
   );
    System.out.println(" ------------------------------------------- "
   );

    System.out.println(" Id: " + event.getID());
    System.out.println(" Title: " + event.getTitle());
    System.out.println(" Category: " + event.getCategory());
    System.out.println(" City: " + event.getCity());
    System.out.println(" Date: from " + event.getDateStart().day + "/"
    + event.getDateStart().month + "/"
        + event.getDateStart().year + " to " + event.getDateEnd().day
   + "/" + event.getDateEnd().month + "/"
        + event.getDateEnd().year);
    System.out.println(" Price: " + event.getPrice() + " euros");
    System.out.println(" Total Tickets: " + event.getTotalTickets() +
   " | Sold Tickets: " + event.getSoldTickets());
    System.out.println(" Descriprion: " + event.getDescription());

    System.out.println(" ------------------------------------------- "
   );

    System.out.println(" > Press Enter to continue...");
    try {
      System.in.read();
    } catch (Exception e) {
    }
    mainMenuAdmin();
  }

  public void mostProfitableMenu() {
    Event event = agenda.mostProfitableEvent();
```

```java

    System.out.println(" ------------------------------------------- ");
    System.out.println("|           Most Profitable Event           |");
    System.out.println(" ------------------------------------------- ");

    System.out.println(" Id: " + event.getID());
    System.out.println(" Title: " + event.getTitle());
    System.out.println(" Category: " + event.getCategory());
    System.out.println(" City: " + event.getCity());
    System.out.println(" Date: from " + event.getDateStart().day + "/" + event.getDateStart().month + "/"
        + event.getDateStart().year + " to " + event.getDateEnd().day + "/" + event.getDateEnd().month + "/"
        + event.getDateEnd().year);
    System.out.println(" Price: " + event.getPrice() + " euros");
    System.out.println(" Total Tickets: " + event.getTotalTickets() + " | Sold Tickets: " + event.getSoldTickets());
    System.out.println(" Descriprion: " + event.getDescription());

    System.out.println(" ------------------------------------------- ");

    System.out.println(" > Press Enter to continue...");
    try {
      System.in.read();
    } catch (Exception e) {
    }
    mainMenuAdmin();
  }

  public void mostActiveMenu() {
    User user = agenda.mostActiveUser();

    System.out.println(" ------------------------------------------- ");
    System.out.println("|               Most Active User            |");
    System.out.println(" ------------------------------------------- ");

    System.out.println(" Email: " + user.getEmail());
    System.out.println(" No. Bought Tickets: " + user.getTicketsBought());

    System.out.println(" ------------------------------------------- ");

    System.out.println(" > Press Enter to continue...");
    try {
      System.in.read();
    } catch (Exception e) {
    }
    mainMenuAdmin();
  }

  /*
   * REGULAR USER
   */

  public void mainMenuRegular() {
    System.out.println(" ------------------------------------------- ");
    System.out.println("|                AGENDA VIRAL               |");
    System.out.println(" ------------------------------------------- ");
    System.out.println(" 1.  Propose Event                0.  Logout");
    System.out.println(" 2.  Find by District");
    System.out.println(" 3.  Find by City");
    System.out.println(" 4.  Find by Category");
    System.out.println(" 5.  Find by Date");
    System.out.println(" 6.  Find by Multiple Filters");
    System.out.println(" ------------------------------------------- ");
    System.out.println(" ! Balance: " + agenda.loggedInUser.getBalance() + " euros");

    System.out.print(" > Option: ");
    int option = scanner.nextInt();
    scanner.nextLine();

    System.out.println("");

    switch (option) {
    case 1:
      proposeEventMenu();
      break;
    case 2:
      findByDistrictMenu();
      break;
    case 3:
      findByCityMenu();
      break;
    case 4:
      findByCategoryMenu();
      break;
    case 5:
      findByDateMenu();
      break;
    case 6:
      findByMultipleFiltersMenu();
      break;
    case 0:
      loginMenu();
      break;
    default:
      loginMenu();
      break;
    }
  }

  public void proposeEventMenu() {
    System.out.println(" -------------------------------------------- ");
    System.out.println("|               Propose Event                |");
    System.out.println(" -------------------------------------------- ");

    System.out.print(" > Title: ");
    String title = scanner.nextLine();

    System.out.print(" > Category: ");
    String category = scanner.nextLine();

    System.out.print(" > Starting Date[dd/mm/yy]: ");
    String startDate = scanner.nextLine();
    String[] part1 = startDate.split("/");

    int day1 = Integer.parseInt(part1[0]);
    int month1 = Integer.parseInt(part1[1]);
    int year1 = Integer.parseInt(part1[2]);

    System.out.print(" > Ending Date[dd/mm/yy]: ");
    String endDate = scanner.nextLine();
    String[] part2 = endDate.split("/");

    int day2 = Integer.parseInt(part2[0]);
    int month2 = Integer.parseInt(part2[1]);
    int year2 = Integer.parseInt(part2[2]);

    System.out.print(" > Description: ");
    String description = scanner.nextLine();

    System.out.print(" > Price: ");
    int price = scanner.nextInt();
    scanner.nextLine();

    System.out.print(" > City: ");
    String city = scanner.nextLine();

    System.out.print(" > Total Tickets: ");
    int tickets = scanner.nextInt();
    scanner.nextLine();

    Event event = new Event(title, category, new Event.Date(day1, month1, year1),
        new Event.Date(day2, month2, year2), description, price, city, tickets);

    agenda.proposeEvent(event);

    mainMenuRegular();
  }

  public void findByDistrictMenu() {
    System.out.println(" ------------------------------------------- ");
    System.out.println("|              Find By District             |");
    System.out.println(" ------------------------------------------- ");
    System.out.println(" 1.  Porto                         0. Return");
    System.out.println(" 2.  Lisboa");
    System.out.println(" 3.  Faro");
    System.out.println(" ------------------------------------------- ");

    System.out.print(" > District: ");
    int option = scanner.nextInt();
    scanner.nextLine();

    System.out.println("");

    VDMSet events = SetUtil.set();

    switch (option) {
    case 1:
      events = agenda.findByDistrict("Porto");
      break;
    case 2:
      events = agenda.findByDistrict("Lisboa");
      break;
    case 3:
      events = agenda.findByDistrict("Faro");
      break;
    case 0:
      mainMenu();
      break;
    default:
      mainMenu();
      break;
    }

    foundEventsMenu(events);
  }

  public void findByCityMenu() {
    System.out.println(" ------------------------------------------- "
   );
    System.out.println("|                Find By City               |"
   );
    System.out.println(" ------------------------------------------- "
   );
    System.out.println(" - PORTO                           0. Return")
   ;
    System.out.println(" 1.  Porto");
    System.out.println(" 2.  Matosinhos");
    System.out.println(" 3.  Maia");
    System.out.println(" 4.  Vila Nova de Gaia");
    System.out.println(" - LISBOA");
    System.out.println(" 5.  Lisboa");
    System.out.println(" 6.  Amadora");
    System.out.println(" 7.  Cascais");
    System.out.println(" 8.  Sintra");
    System.out.println(" - FARO");
    System.out.println(" 9.  Faro");
    System.out.println(" 10. Albufeira");
    System.out.println(" 11. Portimao");
    System.out.println(" ------------------------------------------- "
   );

    System.out.print(" > City: ");
    int option = scanner.nextInt();
    scanner.nextLine();

    System.out.println("");

    VDMSet events = SetUtil.set();
    switch (option) {
```

```
    case 1:
      events = agenda.findByCity("Porto");
      break;
    case 2:
      events = agenda.findByCity("Matosinhos");
      break;
    case 3:
      events = agenda.findByCity("Maia");
      break;
    case 4:
      events = agenda.findByCity("Vila Nova de Gaia");
      break;
    case 5:
      events = agenda.findByCity("Lisboa");
      break;
    case 6:
      events = agenda.findByCity("Amadora");
      break;
    case 7:
      events = agenda.findByCity("Cascais");
      break;
    case 8:
      events = agenda.findByCity("Sintra");
      break;
    case 9:
      events = agenda.findByCity("Faro");
      break;
    case 10:
      events = agenda.findByCity("Albufeira");
      break;
    case 11:
      events = agenda.findByCity("Portimao");
      break;
    case 0:
      mainMenu();
      break;
    default:
      mainMenu();
      break;
    }

    foundEventsMenu(events);
  }

  public void findByCategoryMenu() {
    System.out.println(" ------------------------------------------- "
   );
    System.out.println("|              Find By Category             |"
   );
```

```java
    System.out.println(" ------------------------------------------- ");
    System.out.println(" 1.  Concertos                     0. Return");
    System.out.println(" 2.  Exposicoes");
    System.out.println(" 3.  Gastronomia");
    System.out.println(" 4.  Moda");
    System.out.println(" 5.  Desporto");
    System.out.println(" 6.  Natureza");
    System.out.println(" ------------------------------------------- ");

    System.out.print(" > Category: ");
    int option = scanner.nextInt();
    scanner.nextLine();

    System.out.println("");

    VDMSet events = SetUtil.set();

    switch (option) {
    case 1:
      events = agenda.findByCategory("Concertos");
      break;
    case 2:
      events = agenda.findByCategory("Exposicoes");
      break;
    case 3:
      events = agenda.findByCategory("Gastronomia");
      break;
    case 4:
      events = agenda.findByCategory("Moda");
      break;
    case 5:
      events = agenda.findByCategory("Desporto");
      break;
    case 6:
      events = agenda.findByCategory("Natureza");
      break;
    case 0:
      mainMenu();
      break;
    default:
      mainMenu();
      break;
    }

    foundEventsMenu(events);
  }

  public void findByDateMenu() {
    System.out.println(" ------------------------------------------- "
   );
    System.out.println("|                  Find By Date                 |"
   );
    System.out.println(" ------------------------------------------- "
   );
    System.out.println("                                       0. Main Menu")
   ;

    System.out.print(" > Date[dd/mm/yy]: ");
    String date = scanner.nextLine();
    System.out.println("");

    if (date.equals("0")) {
      mainMenu();
    }

    String[] parts = date.split("/");

    int day = Integer.parseInt(parts[0]);
    int month = Integer.parseInt(parts[1]);
    int year = Integer.parseInt(parts[2]);

    VDMSet events = agenda.findByDate(new Event.Date(day, month, year)
   );

    foundEventsMenu(events);
  }

  public void findByMultipleFiltersMenu() {
    System.out.println(" ------------------------------------------- "
   );
    System.out.println("|                   Find Event                  |"
   );
    System.out.println(" ------------------------------------------- "
   );
    System.out.println(" - CITIES");
    System.out.println(" Porto, Matosinhos, Maia, Vila Nova de Gaia");
    System.out.println(" Lisboa, Amadora, Cascais, Sintra");
    System.out.println(" Faro, Albufeira, Portimao");
    System.out.println("");
    System.out.println(" - DISTRICTS");
    System.out.println(" Porto, Lisboa, Faro");
    System.out.println("");
    System.out.println(" - CATEGORIES");
    System.out.println(" Concertos, Exposicoes, Gastronomia, Moda");
    System.out.println(" Desporto, Natureza");
    System.out.println(" ------------------------------------------- "
   );

    System.out.print(" > City: ");
    String city = scanner.nextLine();

    String district = "";
    if (city.equals("")) {
      System.out.print(" > District: ");
      district = scanner.nextLine();
    }

    System.out.print(" > Category: ");
    String category = scanner.nextLine();

    System.out.print(" > Date[dd/mm/yy]: ");
    String date = scanner.nextLine();

    VDMSet events = null;

    if (date.equals("")) {
      events = agenda.findEvents(city, district, category, null);
    } else {
      String[] parts = date.split("/");

      int day = Integer.parseInt(parts[0]);
      int month = Integer.parseInt(parts[1]);
      int year = Integer.parseInt(parts[2]);
      events = agenda.findEvents(city, district, category, new Event.
   Date(day, month, year));
    }

    foundEventsMenu(events);
  }

  public void foundEventsMenu(VDMSet events) {
    System.out.println(" ------------------------------------------- "
   );
    System.out.println("|                 Found Events              |"
   );
    System.out.println(" ------------------------------------------- "
   );
    for (Iterator iter = events.iterator(); iter.hasNext();) {
      Event event = (Event) iter.next();
      System.out.println(" Id: " + event.getID());
      System.out.println(" Title: " + event.getTitle());

      if (iter.hasNext())
        System.out.println("");
    }

    if (events.isEmpty())
      System.out.println("                    No events                    ");

    System.out.println(" ------------------------------------------------ "
   );
    System.out.println("                                         0. Return "
   );
    System.out.println(" ------------------------------------------------ "
   );

    System.out.print(" > Event Id: ");
    int option = scanner.nextInt();
    scanner.nextLine();

    System.out.println("");

    if (option == 0) {
      mainMenu();
    }

    for (Iterator iter = events.iterator(); iter.hasNext();) {
      Event event = (Event) iter.next();

      if (option == event.getID().intValue()) {
        foundEventMenu(event, events);
      }
    }
  }

  public void foundEventMenu(Event event, VDMSet events) {
    System.out.println(" ------------------------------------------------ "
   );
    System.out.println("|                      Event                      |"
   );
    System.out.println(" ------------------------------------------------ "
   );

    System.out.println(" Id: " + event.getID());
    System.out.println(" Title: " + event.getTitle());
    System.out.println(" Category: " + event.getCategory());
    System.out.println(" City: " + event.getCity());
    System.out.println(" Date: from " + event.getDateStart().day + "/"
    + event.getDateStart().month + "/"
        + event.getDateStart().year + " to " + event.getDateEnd().day
   + "/" + event.getDateEnd().month + "/"
        + event.getDateEnd().year);
    System.out.println(" Price: " + event.getPrice() + " euros");
    System.out.println(" Total Tickets: " + event.getTotalTickets() + " | Sold Tickets: " + event.getSoldTickets());
    System.out.println(" Descriprion: " + event.getDescription());

    System.out.println(" ------------------------------------------- ");
    if (!agenda.loggedInUser.isAdmin())
      System.out.println(" 1.  Buy Tickets                    0. Return");
    else
      System.out.println("                                   0. Return ");
    System.out.println(" ------------------------------------------- ");

    System.out.print(" > Option: ");
    int option = scanner.nextInt();
    scanner.nextLine();

    System.out.println("");

    if (option == 1) {
      System.out.print(" > Number of Tickets: ");
      int nTickets = scanner.nextInt();
      scanner.nextLine();
      boolean res = agenda.buyTicket(event.getID(), nTickets);
      if (res)
        System.out.println("Tickets bought with success!");
      else
        System.out.println("Error buying tickets!");
      mainMenu();
    } else {
      foundEventsMenu(events);
    }
  }
}
```

7 Conclusões

7.1 Resultados Obtidos

A equipa conseguiu implementar toda a lista de requisitos estabelecida no ponto 1.1 e concluiu todos os objetivos delineados para o desenvolvimento esta aplicação.

Este projeto contribuiu para uma excelente aprendizagem de VDM++ e maior aprofundamento da matéria lecionada na unidade curricular de Métodos Formais em Engenharia de Software.

7.2 Possíveis Melhoramentos

O projeto poderia ter sido bem mais complexo, no sentido de ter mais classes e interação entre as mesmas, mas foi decidido optar pelas funcionalidades mais essenciais para o bom funcionamento da aplicação e pelas que faziam mais sentido. No entanto, tendo em conta o que foi desenvolvido, seria interessante implementar o registo de utilizadores.

7.3 Contribuição

O trabalho foi igualmente dividido pelos membros do grupo.

8 Referências

1. Material disponibilizado no Moodle
2. VDM-10 Language Manual, Peter Gorm Larsen et al, Overture Technical Report Series No. TR-001, March 2014
3. Overture tool website, `http://overturetool.org`
4. Agenda Viral website, `https://www.viralagenda.com`