Fitness Application

Métodos Formais em Engenharia de Software

4º ano - 1º semestre

Mestrado Integrado em Engenharia Informática e Computação

Bernardo Ferreira dos Santos Aroso Belchior - up201405381@fe.up.pt
Nuno Miguel Mendes Ramos - up201405498@fe.up.pt

3 de Janeiro de 2018

Conteúdo

1 Descrição do Sistema 3
1.1 Requisitos . . . 3

2 Modelo UML 4
2.1 Use Case Model . . . 4
2.2 Class Model . . . 9

3 Modelo Formal 10
3.1 Admin . . . 10
3.2 CalorieChallenge . . . 10
3.3 Challenge . . . 11
3.4 DistanceChallenge . . . 12
3.5 FitnessApp . . . 13
3.6 RhythmChallenge . . . 16
3.7 Route . . . 17
3.8 Types . . . 18
3.9 User . . . 19
3.10 Workout . . . 21

4 Validação do Modelo 24
4.1 Test . . . 24
4.2 CalorieChallengeTest . . . 24
4.3 DistanceChallengeTest . . . 25
4.4 FitnessAppTest . . . 26
4.5 RhythmChallengeTest . . . 29
4.6 RouteTest . . . 30
4.7 UserTest . . . 31
4.8 WorkoutTest . . . 32

5 Verificação do Modelo 34
5.1 Exemplo de Verificação de um Domínio . . . 34
5.2 Exemplo de Verificação de uma Invariante . . . 34

6 Geração de Código 36
6.1 Main . . . 36
6.2 CommandLineInterface . . . 36

7 Conclusões 53
7.1 Resultados Obtidos . . . 53
7.2 Possíveis Melhoramentos . . . 53
7.3 Contribuição . . . 53

8 Referências 54

1 Descrição do Sistema

O nosso projeto é modelar uma aplicação de fitness, onde o objetivo desta aplicação é o auxílio na execução do exercício físico. Nesta aplicação é possível criar exercícios e rotas específicas. Tem, também, desafios que o utilizador pode tentar bater, por exemplo, distância percorrida ou número de calorias queimadas. Por fim, é possível adicionar amigos.

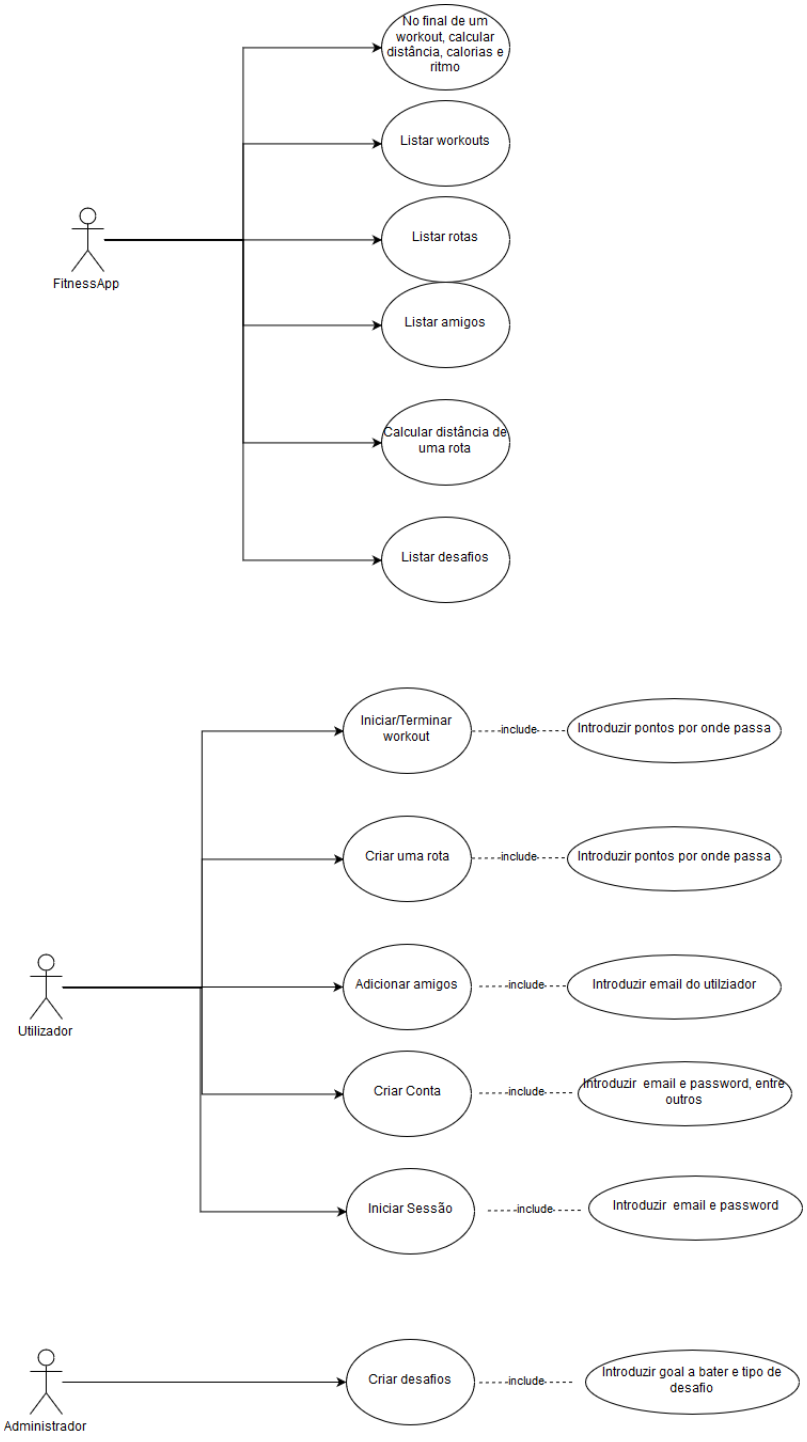
1.1 Requisitos

Para executar as funcionalidades descritas em cima, são necessários os seguintes requisitos:

ID	Prioridade	Descrição
R01	Obrigatória	Um utilizador iniciar e terminar um workout
R02	Obrigatória	Um utilizador no final de um workout obter duração, distância, calorias queimadas e ritmo. E saber se quebrou algum desafio
R03	Obrigatória	Listar, a um utilizador, os seus workouts e respetivas informações
R04	Obrigatória	Um utilizador definir uma nova rota
R05	Obrigatória	Um utilizador obter a distância de uma rota
R06	Obrigatória	Listar, a um utilizador, as suas rotas e respetivas informações
R07	Obrigatória	Um administrador do sistema adicionar um novo desafio (distância, calorias ou ritmo)
R08	Obrigatória	Listar, a um utilizador, os desafios ativos e respetivas informações
R09	Obrigatória	Um utilizador pode criar conta
R10	Obrigatória	Um utilizador pode fazer log-in
R11	Opcional	Um utilizador adicionar amigos
R12	Opcional	Listar, a um utilizador, os seus amigos

2 Modelo UML

2.1 Use Case Model



Cenário	Configuração
Descrição	1. Adicionar um workout
Pré-Condições	1. O utilizador pertencer ao sistema
Pós-Condições	1. O workout adicionado tem de pertencer aos workouts do utilizador
Passos	1. Após criar conta e fazer log-in, o utilizador escolhe a opção "My Workouts" 2. No novo menu escolhe a opção "Start New Workout" 3. Preenche os diferentes parâmetros e termina o Workout 4. O utilizador volta ao menu anterior
Exceções	1. O utilizador interromper o workout sem o terminar

Cenário	Configuração
Descrição	1. Um utilizador obter duração, distância, calorias queimadas e ritmo
Pré-Condições	1. O utilizador pertencer ao sistema 2. O utilizador ter 1 ou mais workouts 3. A rota do workout ter mais do que um ponto (latitude, longitude) 4. Todos os pontos terem latitude e longitude entre -90 e 90
Pós-Condições	1. Duração maior ou igual a 0 2. Distância maior ou igual a 0 3. Calorias queimadas maior ou igual a 0 4. Ritmo maior ou igual a 0
Passos	1. Após criar conta e fazer log-in, o utilizador escolhe a opção "My Workouts" 2. No novo menu escolhe a opção "View My Workouts" 3. Aparece uma lista de workouts 4. Procurar o workout desejado e verificar as informações 5. O utilizador volta ao menu anterior
Exceções	1. Não existir workouts 2. Dados inválidos que impossibilitem o cálculo dos valores

Cenário	Configuração
Descrição	1. Listar os Workouts de um utilizador
Pré-Condições	1. O utilizador pertencer ao sistema 2. O utilizador ter 0 ou mais workouts
Pós-Condições	1. Aparecer todos os workouts do utilizador no ecrã 2. Apenas aparecerem workouts do utilizador
Passos	1. Após criar conta e fazer log-in, o utilizador escolhe a opção "My Workouts" 2. No novo menu escolhe a opção "View My Workouts" 3. Aparece uma lista de workouts 4. O utilizador volta ao menu anterior
Exceções	1. O utilizador terminar o programa

Cenário	Configuração
Descrição	1. Adicionar uma rota
Pré-Condições	1. O utilizador tem de pertencer ao sistema
Pós-Condições	1. A rota adicionada tem de pertence às rotas do utilizador 2. Todos os pontos (latitude, longitude) serem válidos, ou seja, valores entre -90 e 90
Passos	1. Após criar conta e fazer log-in, o utilizador escolhe a opção "My Routes" 2. No novo menu escolhe a opção "Create New Route" 3. Preencher os diferentes parâmetros 4. O utilizador volta ao menu anterior
Exceções	1. Coordenadas inválidas 2. O utilizador terminar o programa sem terminar a rota

Cenário	Configuração
Descrição	1. Listar as rotas de um utilizador
Pré-Condições	1. O utilizador pertencer ao sistema 2. O utilizador ter 0 ou mais rotas
Pós-Condições	1. Aparecer todas as rotas do utilizador 2. Apenas aparecerem rotas do utilizador
Passos	1. Após criar conta e fazer log-in, o utilizador escolhe a opção "My Routes" 2. No novo menu escolhe a opção "View My Routes" 3. Aparece uma lista de rotas 4. O utilizador volta ao menu anterior
Exceções	1. O utilizador terminar o programa

Cenário	Configuração
Descrição	1. Um utilizador obter a distância de uma rota
Pré-Condições	1. O utilizador ter 1 ou mais rotas 2. A rota ter mais do que um ponto (latitude, longitude) 3. Todos os pontos terem latitude e longitude entre -90 e 90
Pós-Condições	1. Aparecer a distância da rota no ecrã 2. A distância ser maior ou igual a 0
Passos	1. Após criar conta e fazer log-in, o utilizador escolhe a opção "My Routes" 2. No novo menu escolhe a opção "View My Routes" 3. Aparece uma lista de rotas 4. Procurar a rota desejada e verificar a distância 5. O utilizador volta ao menu anterior
Exceções	1. Não existir rotas 2. O utilizador terminar o programa

Cenário	Configuração
Descrição	1. Um administrador adicionar um novo desafio (distância, calorias ou ritmo)
Pré-Condições	1. Ser administrador do sistema
Pós-Condições	1. O objectivo ser maior que 0 2. O desafio adicionado tem de pertencer aos desafios do sistema
Passos	1. Fazer log-in, email: ”a@d.min”e password: ”adminadmin” 2. Escolher a opção ”Challenges” 3. No novo menu escolhe a opção ”Add Challenge” 4. Preencher os parâmetros, entre eles o tipo de desafio 5. O utilizador volta ao menu anterior
Exceções	1. O utilizador não ser administrador do sistema

Cenário	Configuração
Descrição	1. Listar os desafios do sistema
Pré-Condições	1. O utilizador pertencer ao sistema 2. O número de desafios do sistema ser igual ou superior a 0
Pós-Condições	1. Aparecer todos os desafios do sistema, no ecrã
Passos	1. Após criar conta e fazer log-in, o utilizador escolhe a opção ”Challenges” 2. No novo menu escolhe a opção ”View Challenges” 3. Aparece uma lista de desafios 4. O utilizador volta ao menu anterior
Exceções	1. O utilizador terminar o programa

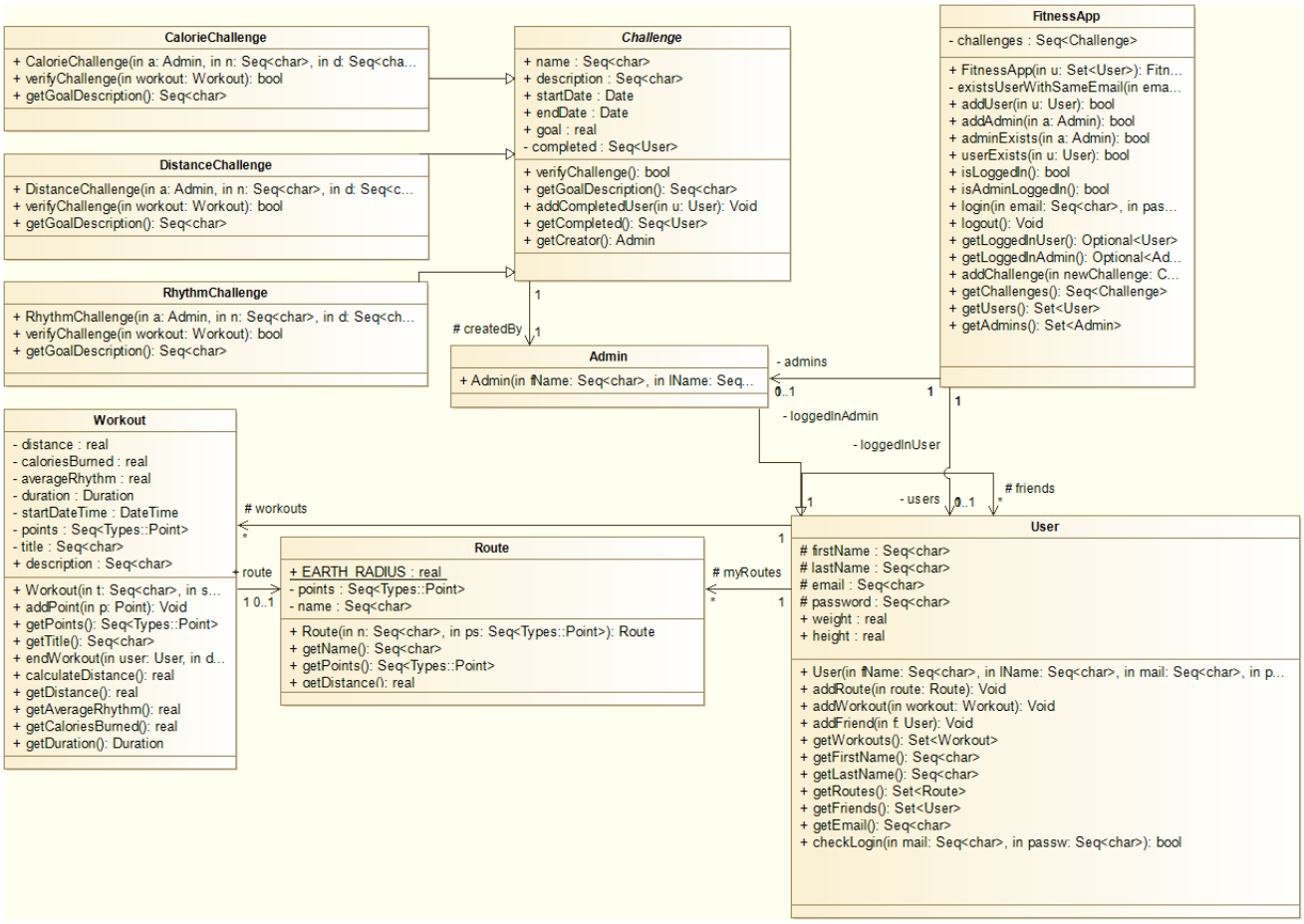
Cenário	Configuração
Descrição	1. Um utilizador criar conta
Pré-Condições	(none)
Pós-Condições	1. O email utilizado não existir em nenhuma conta 2. A conta adicionada tem de pertence às contas do sistema 3. Altura e peso maior ou igual a 0 4. Sexo igual a masculino ou feminino
Passos	1. No menu principal escolher a opção ”Create Account” 2. Preencher os parâmetros necessários 3. O utilizador volta ao menu principal
Exceções	1. Terminar o programa sem criar a conta

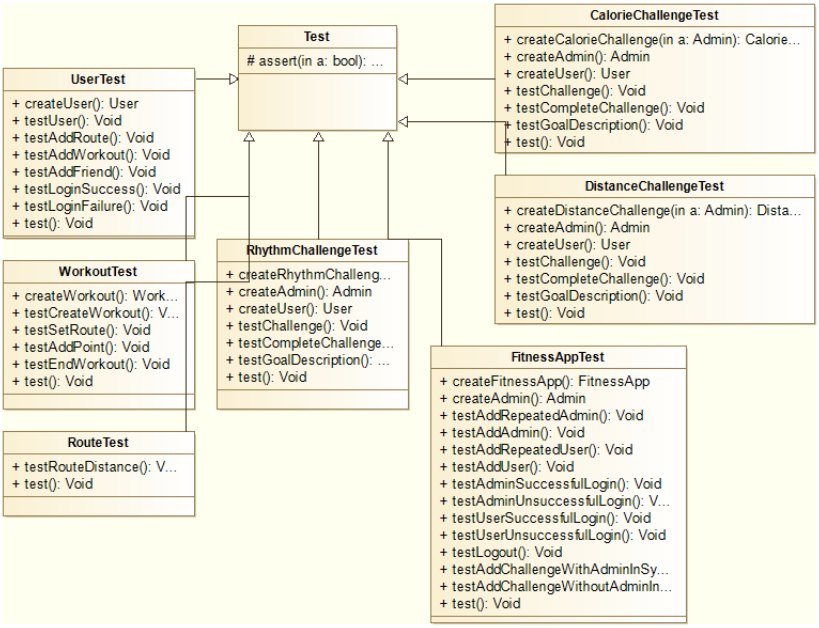
Cenário	Configuração
Descrição	1. Um utilizador fazer log-in
Pré-Condições	1. A conta existir no sistema
Pós-Condições	1. Introduzir dados válidos
Passos	1. No menu principal escolher a opção ”Login” 2. Preencher os parâmetros necessários 3. O utilizador vai para o menu principal quando tem a sessão iniciada
Exceções	1. A conta não existir no sistema 2. O utilizador terminar o programa

Cenário	Configuração
Descrição	1. Um utilizador adicionar amigos
Pré-Condições	1. O utilizador existir no sistema 2. O novo amigo existir no sistema 3. Introduzir um email válido
Pós-Condições	1. O novo amigo adicionado pertencer aos amigos do utilizador
Passos	1. Após criar conta e fazer log-in, o utilizador escolhe a opção ”Manage Friends” 2. No novo menu escolhe a opção ”Add New Friend” 3. Escrever o email do novo amigo, aparecerá uma mensagem dando conta do estado 4. O utilizador volta ao menu anterior
Exceções	1. O novo amigo não existir no sistema 2. O utilizador não existir no sistema

Cenário	Configuração
Descrição	1. Listar os amigos de um utilizador
Pré-Condições	1. O utilizador existir no sistema 2. O utilizador ter 0 ou mais amigos
Pós-Condições	1. Aparecer os amigos do utilizador no ecrã
Passos	1. Após criar conta e fazer log-in, o utilizador escolhe a opção ”Manage Friends” 2. No novo menu escolhe a opção ”View My Friends” 3. Aparece uma lista de amigos 4. O utilizador volta ao menu anterior
Exceções	1. O utilizador terminar o programa

2.2 Class Model





3 Modelo Formal

3.1 Admin

```
-- The Admin class is a User with augmented privileges.
class Admin is subclass of User
operations
 -- Constructor. Equal to a User's.

 public Admin : seq of char * seq of char * seq of char * seq of char * real * real *
     Types`Gender ==> Admin
 Admin(fName, lName, mail, passw, w, h, g) == (
  firstName := fName;
  lastName := lName;
  email := mail;
  password := passw;
  weight := w;
  height := h;
  gender := g;
 )
 pre len mail >= 5 and len passw >= 8
 post firstName = fName and lastName = lName and email = mail and password = passw and weight
     = w and height = h and gender = g;
end Admin
```

Function or operation	Line	Coverage	Calls
Admin	5	100.0%	195
Admin.vdmpp		100.0%	195

3.2 CalorieChallenge

```
-- Represents a challenge where its goal is measured in calories.
class CalorieChallenge is subclass of Challenge
operations

 public CalorieChallenge: Admin * seq of char * seq of char * Types`Date * Types`Date * real
     ==> CalorieChallenge
 CalorieChallenge(a, n, d, s, e, g) == (
  createdBy := a;
  name := n;
  description := d;
  startDate := s;
  endDate := e;
  goal := g;
 )
 post createdBy = a and name = n and description = d and startDate = s and endDate = e and
     goal = g;

 -- Returns true if the challenge is completed, returning false otherwise.

 public verifyChallenge: Workout ==> bool
 verifyChallenge(workout) == (
  return workout.getCaloriesBurned() >= goal;
 );

 -- Returns a message describing the goal.
```

```
 public getGoalDescription: () ==> seq of char
 getGoalDescription() == return "Burn %s kcal";
end CalorieChallenge
```

Function or operation	Line	Coverage	Calls
CalorieChallenge	4	100.0%	45
getGoalDescription	22	100.0%	15
verifyChallenge	16	100.0%	15
CalorieChallenge.vdmpp		100.0%	75

3.3 Challenge

```
-- Abstract class Challenge that represents a generic challenge with a goal. Subclasses
    should provide a way to interpret the goal.
class Challenge
instance variables
 -- Challenge name
 public name: seq of char;

 -- Challenge Description
 public description: seq of char;

 -- Challenge start date
 public startDate: Types`Date;

 -- Challenge end date
 public endDate: Types`Date;

 -- Challenge goal
 public goal: real;

 -- Challenge creator
 protected createdBy: Admin;

 -- Sequence of users who have completed the challenge chronologically sorted.
 private completed: seq of User := [];
operations
 -- Returns true if the challenge is completed, returning false otherwise.

 public verifyChallenge: Workout ==> bool
 verifyChallenge(-) == is subclass responsibility;

 -- Returns a message describing the goal.

 public getGoalDescription: () ==> seq of char
 getGoalDescription() == is subclass responsibility;

 -- Adds a user to the sequence of users who completed the challenge.

 public addCompletedUser: User ==> ()
 addCompletedUser(u) == (completed := [u] ^ completed)
 post completed = [u] ^ completed~;

 -- Returns the sequence of users who completed the challenge.

 public getCompleted: () ==> seq of User
```

```
 getCompleted() == return completed;

 -- Returns the creator of this challenges.

 public getCreator: () ==> Admin
 getCreator() == return createdBy;
end Challenge
```

Function or operation	Line	Coverage	Calls
addCompletedUser	34	100.0%	45
getCompleted	39	100.0%	45
getCreator	43	100.0%	16
getGoalDescription	30	100.0%	16
verifyChallenge	26	100.0%	16
Challenge.vdmpp		100.0%	138

3.4 DistanceChallenge

```
-- Represents a challenge where its goal is measured in distance.
class DistanceChallenge is subclass of Challenge
operations

 public DistanceChallenge: Admin * seq of char * seq of char * Types`Date * Types`Date * real
     ==> DistanceChallenge
 DistanceChallenge(a, n, d, s, e, g) == (
  createdBy := a;
  name := n;
  description := d;
  startDate := s;
  endDate := e;
  goal := g;
 )
 post createdBy = a and name = n and description = d and startDate = s and endDate = e and
      goal = g;

 -- Returns true if the challenge is completed, returning false otherwise.

 public verifyChallenge: Workout ==> bool
 verifyChallenge(workout) == (
  return workout.getDistance() >= goal;
 );

 -- Returns a message describing the goal.

 public getGoalDescription: () ==> seq of char
 getGoalDescription() == return "Run %s km";
end DistanceChallenge
```

Function or operation	Line	Coverage	Calls
DistanceChallenge	4	100.0%	61
getGoalDescription	22	100.0%	15
verifyChallenge	16	100.0%	15

DistanceChallenge.vdmpp		100.0%	91

3.5 FitnessApp

```
-- Represents all the system, includes all the information about it and handles outside
    interactions.
class FitnessApp
instance variables
 -- Users with accounts in the system. Includes both normal users and admins.
 users: map seq of char to User := {|->};

 -- Admins with accounts in the system
 admins: map seq of char to Admin := {|->};

 -- Reference to the logged in user. If nil means there is no user logged in.
 loggedInUser: [User] := nil;

 -- Reference to the logged in admin. If nil means there is no admin logged in.
 -- Note that this can be nil, but loggedInUser may not be if the user does not have admin
     privileges.
 loggedInAdmin: [Admin] := nil;

 -- All challenges in the system
 challenges: seq of Challenge := [];

 -- Establishes the relation between loggedInUser and loggedInAdmin. Ensures loggedInUser is
     never nil if an admin is logged in.
 inv (loggedInUser = nil and loggedInAdmin = nil) or (loggedInAdmin <> nil and loggedInUser =
     loggedInAdmin) or (loggedInUser <> nil and loggedInAdmin = nil);

 -- Ensures the map is consistent with the object information.
 inv forall email in set dom users & users(email).getEmail() = email;
operations
 -- Constructor. Allows the system to be created with some Users.

 public FitnessApp: set of User ==> FitnessApp
 FitnessApp(u) ==
 for all user in set u do (
  users := users ++ {user.getEmail() |-> user}
 )
 pre forall u1, u2 in set u & u1 <> u2 => u1.getEmail() <> u2.getEmail()
 post forall user in set u & user in set rng users;

 -- Checks if there is a user with the same email as provided.

 private existsUserWithSameEmail: seq of char ==> bool
 existsUserWithSameEmail(email) == return email in set dom users;

 -- Adds a user if there is no user with the same email. If the user cannot be added, the
     operation will return false.
 -- Otherwise, it will return true.

 public addUser: User ==> bool
 addUser(u) == (
  if(not existsUserWithSameEmail(u.getEmail())) then (
   users := users ++ {u.getEmail() |-> u};
   return true;
  );

  return false;
 )
```

```
post (RESULT = true and users = users~ ++ {u.getEmail() |-> u}) or (RESULT = false and users
    = users~);

-- Adds a admin if there is no user with the same email. If the admin cannot be added, the
    operation will return false.
-- Otherwise, it will return true.

public addAdmin: Admin ==> bool
addAdmin(a) == (
 if(not existsUserWithSameEmail(a.getEmail())) then (
  admins := admins ++ {a.getEmail() |-> a};
  users := users ++ {a.getEmail() |-> a};
  return true;
 );

 return false;
)
post (RESULT = true and admins = admins~ ++ {a.getEmail() |-> a} and users = users~ ++
    {a.getEmail() |-> a}) or (RESULT = false and users = users~ and admins = admins~);

-- Returns true if the given Admin is in the system.

public adminExists: Admin ==> bool
adminExists(a) == return a.getEmail() in set dom admins
post RESULT = a.getEmail() in set dom admins;

-- Returns true if the given user is in the system.

public userExists: User ==> bool
userExists(u) == return u.getEmail() in set dom users
post RESULT = u.getEmail() in set dom users;

-- Returns true if there is a user logged in.

public isLoggedIn: () ==> bool
isLoggedIn() == return loggedInUser <> nil
post RESULT = (loggedInUser <> nil);

-- Returns true if there is a admin logged in.

public isAdminLoggedIn: () ==> bool
isAdminLoggedIn() == return loggedInAdmin <> nil
post RESULT = (loggedInAdmin <> nil);

-- Logs the user (or admin) in. Returns true if successful and updates the loggedInUser and
    loggedInAdmin variables.

public login: seq of char * seq of char ==> bool
login(email, password) == (
 dcl admin: Admin;
 dcl user: User;

 if (len email < 5 or len password < 8) then
  return false;

 if (email in set dom admins) then (
  admin := admins(email);
  if admin.checkLogin(email, password) then (
   loggedInUser := admin;
   loggedInAdmin := admin;
   return true;
  );
 );
```

```
  if (email in set dom users) then (
   user := users(email);
   if user.checkLogin(email, password) then (
    loggedInUser := user;
    return true;
   );
  );

  return false;
 )
 post (RESULT = true and loggedInUser <> nil) or RESULT = false;

 -- Logs the user out, updating the state variables.

 public logout: () ==> ()
 logout() == (
  loggedInUser := nil;
  loggedInAdmin := nil;
 )
 pre loggedInUser <> nil
 post loggedInUser = nil and loggedInAdmin = nil;

 -- Returns the user currently logged in.

 public getLoggedInUser: () ==> [User]
 getLoggedInUser() == return loggedInUser;

 -- Returns the admin currently logged in.

 public getLoggedInAdmin: () ==> [Admin]
 getLoggedInAdmin() == return loggedInAdmin;

 -- Adds a challenge to the system.

 public addChallenge: Challenge ==> ()
 addChallenge(newChallenge) == (
  if(newChallenge.getCreator() = loggedInAdmin) then (
   challenges := [newChallenge] ^ challenges
  );
 )
 post (loggedInAdmin <> nil and challenges = [newChallenge] ^ challenges~) or (loggedInAdmin
     = nil and challenges = challenges~);

 -- Returns the sequence of challenges

 public getChallenges: () ==> seq of Challenge
 getChallenges() == return challenges;

 -- Returns the set of users.

 public getUsers: () ==> set of User
 getUsers() == return rng users;

 -- Returns the set of admins.

 public getAdmins: () ==> set of Admin
 getAdmins() == return rng admins;
end FitnessApp
```

Function or operation	Line	Coverage	Calls
FitnessApp	27	100.0%	168

addAdmin	54	100.0%	36
addChallenge	135	100.0%	10
addUser	41	100.0%	36
adminExists	67	100.0%	18
existsUserWithSameEmail	36	100.0%	72
getAdmins	152	100.0%	24
getChallenges	144	100.0%	10
getLoggedInAdmin	131	100.0%	22
getLoggedInUser	127	100.0%	22
getUsers	148	100.0%	24
isAdminLoggedIn	82	100.0%	78
isLoggedIn	77	100.0%	90
login	87	100.0%	8
logout	118	100.0%	5
userExists	72	100.0%	36
FitnessApp.vdmpp		100.0%	659

3.6 RhythmChallenge

```
-- Represents a challenge where its goal is measured in rhythm.
class RhythmChallenge is subclass of Challenge
operations

 public RhythmChallenge: Admin * seq of char * seq of char * Types`Date * Types`Date * real
     ==> RhythmChallenge
 RhythmChallenge(a, n, d, s, e, g) == (
  createdBy := a;
  name := n;
  description := d;
  startDate := s;
  endDate := e;
  goal := g;
 )
 post createdBy = a and name = n and description = d and startDate = s and endDate = e and
      goal = g;

 -- Returns true if the challenge is completed, returning false otherwise.

 public verifyChallenge: Workout ==> bool
 verifyChallenge(workout) == (
  return workout.getAverageRhythm() >= goal;
 );

 -- Returns a message describing the goal.

 public getGoalDescription: () ==> seq of char
 getGoalDescription() == return "Run at a pace of %s min/km";
end RhythmChallenge
```

Function or operation	Line	Coverage	Calls
RhythmChallenge	4	100.0%	45
getGoalDescription	22	100.0%	15
verifyChallenge	16	100.0%	15

RhythmChallenge.vdmpp		100.0%	75

3.7 Route

```
-- Represents a route taken in an activity.
class Route
values
 -- Earth radius in kilometers
 public EARTH_RADIUS: real = 6373;
instance variables
 -- Points this route goes through.
 points: seq of Types`Point := [];

 -- This route's name.
 name: seq of char;

 -- Ensure sure this route always has at least 2 points.
 inv len points > 1;
operations

 public Route: seq of char * seq of Types`Point ==> Route
 Route(n, ps) == (
   name := n;
   points := ps;
  )
  post len points = len ps and points = ps and name = n;

 -- Returns the route's name.

 public getName: () ==> seq of char
 getName() == return name;

 -- Returns the points this route goes through

 public getPoints: () ==> seq of Types`Point
 getPoints() == return points;

 -- Calculates the real distance between Points

  public getDistance: () ==> real
  getDistance() == (
 dcl totalDistance: real := 0;

  -- Source: http://www.geodatasource.com/developers/javascript
 dcl upperBound: real := (len points) - 1;
 for i = 1 to upperBound do (
   dcl radLat1: real := MATH`pi * points(i).lat / 180;
   dcl radLat2: real := MATH`pi * points(i+1).lat / 180;
   dcl theta: real := points(i).long - points(i+1).long;
   dcl radTheta: real := MATH`pi * theta / 180;
   dcl dist: real := MATH`sin(radLat1) * MATH`sin(radLat2) + MATH`cos(radLat1) *
       MATH`cos(radLat2) * MATH`cos(radTheta);
   dist := MATH`acos(dist);
   dist := dist * 180 / MATH`pi;
   dist := dist * 60 * 1.1515;
   dist := dist * 1.609344;
   totalDistance := totalDistance + dist;
  );

  return totalDistance;
 );
end Route
```

Function or operation	Line	Coverage	Calls
Route	16	100.0%	45
getDistance	32	100.0%	15
getName	24	100.0%	15
getPoints	28	100.0%	15
Route.vdmpp		100.0%	90

3.8 Types

```
-- Class that houses all the types used in the system.
class Types
types
 -- Represents a moment in time by its date and time.
 public DateTime :: date: Date
           time: Time;

 -- Represents a duration.
 public Duration :: hours : int
            minutes: int
            seconds: int
  inv duration ==
   duration.minutes >= 0 and duration.minutes < 60 and
   duration.seconds >= 0 and duration.seconds < 60 and
   duration.hours >= 0;

 -- Represents a moment of a day.
 public Time :: hours : int
            minutes: int
            seconds: int
  inv time ==
   time.minutes >= 0 and time.minutes < 60 and
   time.seconds >= 0 and time.seconds < 60 and
   time.hours >= 0 and time.hours < 24;
 -- Ensures time

 -- Represents a date.
 public Date :: year : int
        month: int
        day: int
 inv date ==
  date.year >= 0 and date.month > 0 and date.month <= 12 and date.day > 0 and date.day <= 31;

 -- Represents the type of an activity
 public Activity = <Bycicle> | <Running> | <Walking>;

 -- Represents a gender
 public Gender = <Masculine> | <Feminine>;

 -- Represents a point with latitude and longitude.
 public Point :: lat : real
         long : real;
 functions

 public static toMinutes: Types`Duration -> real
 toMinutes(duration) == duration.minutes + duration.hours*60 + duration.seconds/60;
end Types
```

Function or operation	Line	Coverage	Calls
toMinutes	44	0.0%	0
Types.vdmpp		100.0%	0

3.9 User

```
class User
instance variables
 -- User's first name
 protected firstName: seq of char;

 -- User's last name
 protected lastName: seq of char;

 -- Workouts the user has completed
 protected workouts: set of Workout := {};

 -- Routes the user has created
 protected myRoutes: set of Route := {};

 -- Friends the user has added
 protected friends: set of User := {};

 -- User's email
 protected email: seq of char;

 -- User's password
 protected password: seq of char;

 -- User's gender
 public gender: Types`Gender;

 -- User's weight in kilograms
 public weight: real;

 -- User's height in meters
 public height: real;

 -- Restricts some invalid emails. At least 5 characters are needed to write a simple e-mail,
     eg.: a@b.c
 inv len email >= 5;
operations

 public User : seq of char * seq of char * seq of char * seq of char * real * real *
     Types`Gender ==> User
 User(fName, lName, mail, passw, w, h, g) == (
  firstName := fName;
  lastName := lName;
  email := mail;
  password := passw;
  weight := w;
  height := h;
  gender := g;
 )
 pre len mail >= 5 and len passw >= 8 and h > 0 and w > 0
 post firstName = fName and lastName = lName and email = mail and password = passw and weight
     = w and height = h and gender = g;

 -- Adds a route to user's routes

 public addRoute: Route ==> ()
```

```
  addRoute(route) == myRoutes := myRoutes union {route}
  post route in set myRoutes;

 -- Adds a complete workout to the set of complete workouts

 public addWorkout: Workout ==> ()
  addWorkout(workout) == workouts := workouts union {workout}
  post workout in set workouts;

 -- Add a friend to user's set of friends

 public addFriend: User ==> ()
  addFriend(f) == (
   friends := friends union {f};
   if self not in set f.getFriends()
   then f.addFriend(self);
  )
  pre f <> self
  post f in set friends;

 -- Returns the user's workouts

 public getWorkouts: () ==> set of Workout
 getWorkouts() == return workouts;

 -- Returns the user's first name

 public getFirstName: () ==> seq of char
 getFirstName() == return firstName;

 -- Returns the user's last name

 public getLastName: () ==> seq of char
 getLastName() == return lastName;

 -- Returns the user's routes

 public getRoutes: () ==> set of Route
 getRoutes() == return myRoutes;

 -- Returns the user's friends

 public getFriends: () ==> set of User
 getFriends() == return friends;

 -- Returns the user's email

 public pure getEmail: () ==> seq of char
 getEmail() == return email;

 -- Checks if the email and password combination matches

 public checkLogin: seq of char * seq of char ==> bool
 checkLogin(mail, passw) == (
  return email = mail and password = passw;
 )
 post RESULT = (email = mail and password = passw);
end User
```

Function or operation	Line	Coverage	Calls
User	36	100.0%	437

addFriend	60	100.0%	30
addRoute	50	100.0%	15
addWorkout	55	100.0%	15
checkLogin	94	100.0%	89
getEmail	90	100.0%	1732
getFirstName	74	100.0%	15
getFriends	86	100.0%	45
getLastName	78	100.0%	15
getRoutes	82	100.0%	15
getWorkouts	70	100.0%	15
User.vdmpp		100.0%	2423

3.10 Workout

```
class Workout
types
instance variables
 distance: real := 0; -- In km
 caloriesBurned: real := 0; -- In km
 averageRhythm: real := 0; -- In min/km
 duration: Types`Duration; -- In min
 startDateTime: Types`DateTime;
 activity: Types`Activity;
 points: seq of Types`Point := [];
 title: seq of char;
 public route: [Route] := nil;
 public description: seq of char;

 inv distance >= 0;
 inv caloriesBurned >= 0;
 inv averageRhythm >= 0;
operations

 public Workout: seq of char * Types`DateTime * Types`Activity * Types`Point ==> Workout
 Workout(t, sd, a, p) == (
  title := t;
  startDateTime := sd;
  activity := a;
  points := points ^[p];
 )
 post startDateTime = sd and activity = a and title = t and len points > 0 and points(len
     points) = p and route = nil;


 public addPoint: Types`Point ==> ()
 addPoint(p) == points := points ^ [p]
  post len points > 0 and points(len points) = p;


 public getPoints: () ==> seq of Types`Point
 getPoints() == return points;


 public getTitle: () ==> seq of char
 getTitle() == return title;


 public endWorkout: User * Types`Duration ==> ()
 endWorkout(user, d) == (
```

```
  duration := d;
  distance := calculateDistance();
  caloriesBurned := distance * user.weight * 1.036;
  averageRhythm := distance / Types`toMinutes(duration);
 );


 public calculateDistance: () ==> real -- returns in kms
  calculateDistance() == (
  dcl totalDistance: real := 0;

   -- Source: http://www.geodatasource.com/developers/javascript
  dcl upperBound: real := (len points) - 1;
  for i = 1 to upperBound do (
    dcl radLat1: real := MATH`pi * points(i).lat / 180;
    dcl radLat2: real := MATH`pi * points(i+1).lat / 180;
    dcl theta: real := points(i).long - points(i+1).long;
    dcl radTheta: real := MATH`pi * theta / 180;
    dcl dist: real := MATH`sin(radLat1) * MATH`sin(radLat2) + MATH`cos(radLat1) *
        MATH`cos(radLat2) * MATH`cos(radTheta);
    dist := MATH`acos(dist);
    dist := dist * 180 / MATH`pi;
    dist := dist * 60 * 1.1515;
    dist := dist * 1.609344;
    totalDistance := totalDistance + dist;
   );

   return totalDistance;
  );


  public getDistance: () ==> real
  getDistance() == return distance;


  public getAverageRhythm: () ==> real
  getAverageRhythm() == return averageRhythm;


  public getCaloriesBurned: () ==> real
  getCaloriesBurned() == return caloriesBurned;


  public getDuration: () ==> Types`Duration
  getDuration() == return duration;
end Workout
```

Function or operation	Line	Coverage	Calls
Workout	19	100.0%	120
addPoint	28	100.0%	75
calculateDistance	46	100.0%	60
endWorkout	38	100.0%	60
getAverageRhythm	71	100.0%	30
getCaloriesBurned	74	100.0%	30
getDistance	68	100.0%	30
getDuration	77	100.0%	15
getPoints	32	100.0%	30
getTitle	35	100.0%	15

Workout.vdmpp		100.0%	465

4 Validação do Modelo

4.1 Test

```
class Test
operations

 protected assert : bool ==> ()
 assert(a) == return
 pre a
end Test
```

4.2 CalorieChallengeTest

```
class CalorieChallengeTest is subclass of Test
operations

 public createCalorieChallenge: Admin ==> CalorieChallenge
 createCalorieChallenge(a) ==
  return new CalorieChallenge(a, "Testing Challenge", "Lets test Challenge class",
      mk_Types`Date(2017,12,24), mk_Types`Date(2017,12,30), 20);


 public createAdmin: () ==> Admin
 createAdmin() ==

  return new Admin("Bernardo", "Belchior", "b@b.c", "12345678", 70, 1.70, <Masculine>);

 public createUser: () ==> User

 createUser() ==
  return new User("Nuno", "Ramos", "a@b.c", "12345678", 70, 1.70, <Masculine>);

 public testChallenge: () ==> ()
 testChallenge() == (
  dcl c: CalorieChallenge := createCalorieChallenge(createAdmin());
  assert(c.name = "Testing Challenge");
  assert(c.description = "Lets test Challenge class");

  assert(c.startDate = mk_Types`Date(2017,12,24));
  assert(c.endDate = mk_Types`Date(2017,12,30));
 );

 public testCompleteChallenge: () ==> ()
 testCompleteChallenge() == (
  dcl c: CalorieChallenge := createCalorieChallenge(createAdmin());
  dcl w: Workout := new Workout("Morning Run", mk_Types`DateTime(mk_Types`Date(2017,12,21),
      mk_Types`Time(16, 37, 00)), <Running>, mk_Types`Point(0, 0));
  dcl u: User := createUser();
  dcl p: Types`Point := mk_Types`Point(10, 10);
  w.addPoint(p);
  w.endWorkout(u, mk_Types`Duration(0, 10, 0));

  assert(true = c.verifyChallenge(w));
  c.addCompletedUser(u);
  assert(u in set elems c.getCompleted());
 );
```

```
 public testGoalDescription: () ==> ()

 testGoalDescription() == (
  dcl c: CalorieChallenge := createCalorieChallenge(createAdmin());
  dcl a: seq of char := c.getGoalDescription();
  assert(a = "Burn %s kcal");
 );

 public test: () ==> ()
 test() == (
  testChallenge();
  testCompleteChallenge();
  testGoalDescription();
 )
end CalorieChallengeTest
```

4.3 DistanceChallengeTest

```
class DistanceChallengeTest is subclass of Test
operations

 public createDistanceChallenge: Admin ==> DistanceChallenge
 createDistanceChallenge(a) ==
  return new DistanceChallenge(a, "Testing Challenge", "Lets test Challenge class",
      mk_Types`Date(2017,12,24), mk_Types`Date(2017,12,30), 20);


 public createAdmin: () ==> Admin
 createAdmin() ==

  return new Admin("Bernardo", "Belchior", "b@b.c", "12345678", 70, 1.70, <Masculine>);

 public createUser: () ==> User

 createUser() ==
  return new User("Nuno", "Ramos", "a@b.c", "12345678", 70, 1.70, <Masculine>);

 public testChallenge: () ==> ()
 testChallenge() == (
  dcl c: DistanceChallenge := createDistanceChallenge(createAdmin());
  assert(c.name = "Testing Challenge");
  assert(c.description = "Lets test Challenge class");

  assert(c.startDate = mk_Types`Date(2017,12,24));
  assert(c.endDate = mk_Types`Date(2017,12,30));
 );

 public testCompleteChallenge: () ==> ()
 testCompleteChallenge() == (
  dcl c: DistanceChallenge := createDistanceChallenge(createAdmin());
  dcl w: Workout := new Workout("Morning Run", mk_Types`DateTime(mk_Types`Date(2017,12,21),
      mk_Types`Time(16, 37, 00)), <Running>, mk_Types`Point(0, 0));
  dcl u: User := createUser();
  dcl p: Types`Point := mk_Types`Point(10, 10);
  w.addPoint(p);
  w.endWorkout(u, mk_Types`Duration(0, 10, 0));

  assert(true = c.verifyChallenge(w));
  c.addCompletedUser(u);
  assert(u in set elems c.getCompleted());
```

```
 );

 public testGoalDescription: () ==> ()

 testGoalDescription() == (
  dcl c: DistanceChallenge := createDistanceChallenge(createAdmin());
  dcl a: seq of char := c.getGoalDescription();
  assert(a = "Run %s km");
 );

 public test: () ==> ()
 test() == (
  testChallenge();
  testCompleteChallenge();
  testGoalDescription();
 )
end DistanceChallengeTest
```

4.4 FitnessAppTest

```
class FitnessAppTest is subclass of Test
operations

 public static main() == (
  new UserTest().test();
  new WorkoutTest().test();
  new CalorieChallengeTest().test();
  new DistanceChallengeTest().test();
  new RhythmChallengeTest().test();
  new RouteTest().test();
  new FitnessAppTest().test();
 );


 public createFitnessApp: () ==> FitnessApp
 createFitnessApp() == (
  return new FitnessApp({new User("Nuno", "Ramos", "a@b.c", "12345678", 70, 1.70,
      <Masculine>), new User("Bernardo", "Belchior", "z@z.c", "12345678", 70, 1.70,
      <Masculine>)});
 );


 public createAdmin: () ==> Admin
 createAdmin() ==

  return new Admin("Bernardo", "Belchior", "b@b.c", "12345678", 70, 1.70, <Masculine>);

 public testAddRepeatedAdmin: () ==> ()
 testAddRepeatedAdmin() == (
  dcl f: FitnessApp := createFitnessApp();
  dcl a: Admin := new Admin("John", "Doe", "b@b.d", "12345678", 70, 1.70, <Masculine>);
  dcl userLen: nat := card f.getUsers();
  dcl adminLen: nat := card f.getAdmins();

  assert(not f.adminExists(a));
  assert(not f.userExists(a));
  assert(f.addAdmin(a));

  assert(f.addAdmin(a) = false);
  assert(card f.getUsers() = userLen + 1);
```

```
 assert(card f.getAdmins() = adminLen + 1);
);

public testAddAdmin: () ==> ()
testAddAdmin() == (
 dcl f: FitnessApp := createFitnessApp();
 dcl a: Admin := new Admin("John", "Doe", "b@b.d", "12345678", 70, 1.70, <Masculine>);
 assert(not f.adminExists(a));
 assert(not f.userExists(a));
 assert(f.addAdmin(a));

 assert(f.userExists(a));
 assert(f.adminExists(a));
);

public testAddRepeatedUser: () ==> ()
testAddRepeatedUser() == (
 dcl f: FitnessApp := createFitnessApp();
 dcl u: User := new User("John", "Doe", "b@b.d", "12345678", 70, 1.70, <Masculine>);
 dcl userLen: nat := card f.getUsers();
 dcl adminLen: nat := card f.getAdmins();
 assert(not f.userExists(u));

 assert(f.addUser(u));
 assert(f.addUser(u) = false);


 assert(card f.getUsers() = userLen + 1);
 assert(card f.getAdmins() = adminLen);
);

public testAddUser: () ==> ()
testAddUser() == (
 dcl f: FitnessApp := createFitnessApp();

 dcl u: User := new User("John", "Doe", "b@b.d", "12345678", 70, 1.70, <Masculine>);
 assert(not f.userExists(u));
 assert(f.addUser(u));
 assert(f.userExists(u));
);

public testAdminSuccessfulLogin: () ==> ()
testAdminSuccessfulLogin() == (
 dcl f: FitnessApp := createFitnessApp();
 dcl a: Admin := new Admin("John", "Doe", "d@d.b", "12345678", 70, 1.70, <Masculine>);
 assert(f.addAdmin(a));
 assert(not f.isAdminLoggedIn());
 assert(not f.isLoggedIn());

 assert(f.login("d@d.b", "12345678"));
 assert(f.isAdminLoggedIn());
 assert(f.isLoggedIn());
 assert(f.getLoggedInUser() = a);
 assert(f.getLoggedInAdmin() = a);
);

public testAdminUnsuccessfulLogin: () ==> ()
testAdminUnsuccessfulLogin() == (
 dcl f: FitnessApp := createFitnessApp();
 dcl a: Admin := new Admin("John", "Doe", "d@d.c", "12345678", 70, 1.70, <Masculine>);
 assert(f.addAdmin(a));
 assert(not f.isAdminLoggedIn());
 assert(not f.isLoggedIn());
 assert(not f.login("a", "invalid")); -- Tests if len email < 5
```

```
 assert(not f.isAdminLoggedIn());
 assert(not f.isLoggedIn());
 assert(not f.login("invalid", "invalid")); -- Tests if len password < 8
 assert(not f.isAdminLoggedIn());

 assert(not f.isLoggedIn());
 assert(not f.login("invalid", "invalid1")); -- Tests if combination email, password is not
     found
 assert(not f.isAdminLoggedIn());
 assert(not f.isLoggedIn());
 assert(f.getLoggedInUser() = nil);
 assert(f.getLoggedInAdmin() = nil);
);

public testUserSuccessfulLogin: () ==> ()
testUserSuccessfulLogin() == (
 dcl f: FitnessApp := createFitnessApp();
 dcl u: User := new User("John", "Doe", "d@d.b", "12345678", 70, 1.70, <Masculine>);
 assert(f.addUser(u));

 assert(not f.isAdminLoggedIn());
 assert(not f.isLoggedIn());
 assert(f.login("d@d.b", "12345678"));
 assert(not f.isAdminLoggedIn());
 assert(f.isLoggedIn());
 assert(f.getLoggedInUser() = u);
 assert(f.getLoggedInAdmin() = nil);
);

public testUserUnsuccessfulLogin: () ==> ()
testUserUnsuccessfulLogin() == (
 dcl f: FitnessApp := createFitnessApp();
 dcl u: User := new User("John", "Doe", "d@b.d", "12345678", 70, 1.70, <Masculine>);
 assert(f.addUser(u));
 assert(not f.isAdminLoggedIn());
 assert(not f.isLoggedIn());
 assert(not f.login("a", "invalid")); -- Tests if len email < 5
 assert(not f.isAdminLoggedIn());
 assert(not f.isLoggedIn());

 assert(not f.login("invalid", "invalid")); -- Tests if len password < 8
 assert(not f.isAdminLoggedIn());
 assert(not f.isLoggedIn());
 assert(not f.login("invalid", "invalid1")); -- Tests if combination email, password is not
     found
 assert(not f.isAdminLoggedIn());
 assert(not f.isLoggedIn());
 assert(f.getLoggedInUser() = nil);
 assert(f.getLoggedInAdmin() = nil);
);

public testLogout: () ==> ()
testLogout() == (

 dcl f: FitnessApp := createFitnessApp();
 dcl u: User := new User("John", "Doe", "d@b.c", "12345678", 70, 1.70, <Masculine>);
 assert(f.addUser(u));
 assert(not f.isLoggedIn());
 assert(f.login("a@b.c", "12345678"));
 assert(f.isLoggedIn());
 f.logout();
 assert(not f.isLoggedIn());
 assert(not f.isAdminLoggedIn());
);
```

```
 public testAddChallengeWithAdminInSystem: () ==> ()
 testAddChallengeWithAdminInSystem() == (
  dcl f: FitnessApp := createFitnessApp();
  dcl a: Admin := createAdmin();
  dcl c: Challenge := new DistanceChallenge(a, "Test", "Because we need to test",
      mk_Types`Date(2018, 1, 1), mk_Types`Date(2018, 2, 1), 10);
  assert(f.addAdmin(a));

  assert(f.login("b@b.c", "12345678"));
  f.addChallenge(c);
  assert(c in set elems f.getChallenges());
 );

 public testAddChallengeWithoutAdminInSystem: () ==> ()
 testAddChallengeWithoutAdminInSystem() == (
  dcl f: FitnessApp := createFitnessApp();
  dcl c: Challenge := new DistanceChallenge(createAdmin(), "Test", "Because we need to test",
      mk_Types`Date(2018, 1, 1), mk_Types`Date(2018, 2, 1), 10);
  f.addChallenge(c);
  assert(not c in set elems f.getChallenges());
 );

 public test: () ==> ()
 test() == (
  testAddAdmin();
  testAddRepeatedAdmin();
  testAddUser();
  testAddRepeatedUser();
  testAdminSuccessfulLogin();
  testAdminUnsuccessfulLogin();
  testUserSuccessfulLogin();
  testUserUnsuccessfulLogin();
  testLogout();
  testAddChallengeWithoutAdminInSystem();
  testAddChallengeWithAdminInSystem();
  )
end FitnessAppTest
```

4.5 RhythmChallengeTest

```
class RhythmChallengeTest is subclass of Test
operations


 public createRhythmChallenge: Admin ==> RhythmChallenge
 createRhythmChallenge(a) ==
  return new RhythmChallenge(a, "Testing Challenge", "Lets test Challenge class",
      mk_Types`Date(2017,12,24), mk_Types`Date(2017,12,30), 20);


 public createAdmin: () ==> Admin
 createAdmin() ==

  return new Admin("Bernardo", "Belchior", "b@b.c", "12345678", 70, 1.70, <Masculine>);

 public createUser: () ==> User

 createUser() ==
  return new User("Nuno", "Ramos", "a@b.c", "12345678", 70, 1.70, <Masculine>);
```

```
 public testChallenge: () ==> ()
 testChallenge() == (
  dcl c: RhythmChallenge := createRhythmChallenge(createAdmin());
  assert(c.name = "Testing Challenge");
  assert(c.description = "Lets test Challenge class");

  assert(c.startDate = mk_Types`Date(2017,12,24));
  assert(c.endDate = mk_Types`Date(2017,12,30));
 );

 public testCompleteChallenge: () ==> ()
 testCompleteChallenge() == (
  dcl c: RhythmChallenge := createRhythmChallenge(createAdmin());
  dcl w: Workout := new Workout("Morning Run", mk_Types`DateTime(mk_Types`Date(2017,12,21),
      mk_Types`Time(16, 37, 00)), <Running>, mk_Types`Point(0, 0));
  dcl u: User := createUser();
  dcl p: Types`Point := mk_Types`Point(10, 10);
  w.addPoint(p);
  w.endWorkout(u, mk_Types`Duration(0, 20, 0));

  assert(true = c.verifyChallenge(w));
  c.addCompletedUser(u);
  assert(u in set elems c.getCompleted());
 );

 public testGoalDescription: () ==> ()

 testGoalDescription() == (
  dcl c: RhythmChallenge := createRhythmChallenge(createAdmin());
  dcl a: seq of char := c.getGoalDescription();
  assert(a = "Run at a pace of %s min/km");
 );

 public test: () ==> ()
 test() == (
  testChallenge();
  testCompleteChallenge();
  testGoalDescription();
 )
end RhythmChallengeTest
```

4.6 RouteTest

```
class RouteTest is subclass of Test
operations

 public testRouteDistance: () ==> ()
 testRouteDistance() == (
  dcl ps: seq of Types`Point := [mk_Types`Point(38.898556, -77.037852),
      mk_Types`Point(38.897147, -77.043934)];
  dcl r: Route := new Route("Test", ps);

  assert(r.getDistance() = 0.5491293773144347);
  assert(r.getName() = "Test");
  assert(r.getPoints() = ps);
 );


 public test: () ==> ()
```

```
 test() ==
  testRouteDistance();
end RouteTest
```

4.7 UserTest

```
class UserTest is subclass of Test
types
operations

 public createUser: () ==> User
 createUser() ==
  return new User("Bernardo", "Belchior", "up201405381@fe.up.pt", "12345678", 70, 1.70,
      <Masculine>);


 public testUser: () ==> ()
 testUser() == (
  dcl u: User := createUser();
  u.weight := 90;
  u.height := 1.20;
  assert(u.getFirstName() = "Bernardo");
  assert(u.getLastName() = "Belchior");
  assert(u.getEmail() = "up201405381@fe.up.pt");
  assert(u.checkLogin("up201405381@fe.up.pt", "12345678"));
  assert(u.weight = 90);
  assert(u.height = 1.20);
  assert(u.gender = <Masculine>);

 );

 public testAddRoute: () ==> ()
 testAddRoute() == (
  dcl u: User := createUser();
  dcl r: Route := new Route("Sunday Run", [mk_Types`Point(0, 0), mk_Types`Point(0, 1)]);
  u.addRoute(r);

  assert(r in set u.getRoutes());
 );

 public testAddWorkout: () ==> ()
 testAddWorkout() == (
  dcl u: User := createUser();
  dcl w: Workout := new Workout("Morning Run", mk_Types`DateTime(mk_Types`Date(2017,12,21),
      mk_Types`Time(16, 37, 00)), <Running>, mk_Types`Point(0, 0));

  u.addWorkout(w);
  assert(w in set u.getWorkouts());
 );

 public testAddFriend: () ==> ()
 testAddFriend() == (
  dcl u1: User := createUser();

  dcl u2: User := new User("Nuno", "Ramos", "a@b.c", "12345678", 70, 1.70, <Masculine>);
  u1.addFriend(u2);
  assert(u2 in set u1.getFriends());
 );
```

```
 public testLoginSuccess: () ==> ()
 testLoginSuccess() == (
  dcl u: User := createUser();
  assert(u.checkLogin("up201405381@fe.up.pt", "12345678"));
 );


 public testLoginFailure: () ==> ()
 testLoginFailure() == (
  dcl u: User := createUser();
  assert(not u.checkLogin("invalid", "invalid"));
 );

 public test: () ==> ()
 test() == (
  testUser();
  testAddRoute();
  testAddWorkout();
  testAddFriend();
  testLoginSuccess();
  testLoginFailure();
 );
end UserTest
```

4.8 WorkoutTest

```
class WorkoutTest is subclass of Test
operations

 public createWorkout: () ==> Workout
 createWorkout() ==
  return new Workout("Morning Run", mk_Types`DateTime(mk_Types`Date(2017,12,21),
       mk_Types`Time(16, 37, 00)), <Running>, mk_Types`Point(0, 0));


 public testCreateWorkout: () ==> ()
 testCreateWorkout() == (
  dcl w: Workout := createWorkout();
  assert(w.getTitle() = "Morning Run");

 );

 public testSetRoute: () ==> ()
 testSetRoute() == (
  dcl w: Workout := createWorkout();
  dcl r: Route := new Route("Sunday Run", [mk_Types`Point(0, 0), mk_Types`Point(0, 1)]);
  w.route := r;

  assert(w.route = r);
 );

 public testAddPoint: () ==> ()
 testAddPoint() == (
  dcl w: Workout := createWorkout();
  dcl p: Types`Point := mk_Types`Point(10, 10);

  w.addPoint(p);
  assert(p = w.getPoints()(len w.getPoints()))
 );
```

```
 public testEndWorkout: () ==> ()
 testEndWorkout() == (
  dcl w: Workout := createWorkout();
  dcl p: Types`Point := mk_Types`Point(10, 10);
  dcl u: User := new User("John", "Doe", "a@b.c", "12345678", 70, 1.70, <Masculine>);
  w.addPoint(p);
  w.endWorkout(u, mk_Types`Duration(0, 20, 0));
  assert(w.getDistance() = 1568.445093911413);

  assert(w.getCaloriesBurned() = 113743.63821045568);
  assert(w.getAverageRhythm() = 78.42225469557066);
  assert(w.getDuration().minutes = 20);
 );

 public test: () ==> ()
 test() == (
  testCreateWorkout();
  testSetRoute();
  testAddPoint();
  testEndWorkout();
 );
end WorkoutTest
```

5 Verificação do Modelo

5.1 Exemplo de Verificação de um Domínio

Uma proof obligation gerada pelo Overture é a seguinte:

No.	PO Name	Type
7	FitnessApp‘FitnessApp(set of (User))	legal map application

Código em análise:

```
-- Logs the user (or admin) in. Returns true if successful and updates the loggedInUser and
    loggedInAdmin variables.
public login: seq of char * seq of char ==> bool
login(email, password) == (
  dcl admin: Admin;
  dcl user: User;

  if (len email < 5 or len password < 8) then
    return false;

  if (email in set dom admins) then (
    admin := admins(email);
    if admin.checkLogin(email, password) then (
      loggedInUser := admin;
      loggedInAdmin := admin;
      return true;
    );
  );


  if (email in set dom users) then (
    user := users(email);
    if user.checkLogin(email, password) then (
      loggedInUser := user;
      return true;
    );
  );

  return false;
)
post (RESULT = true and loggedInUser <> nil) or RESULT = false;
```

A verificação de um domínio em análise é a seguinte:

```
forall email: seq of char, password: seq of char & email in set dom admins
```

Neste exemplo é possível verificar que a aplicação de um mapa é legal porque englobado por uma estrutura if que garante que o argumento da aplicação pertence ao domínio do mapa.

5.2 Exemplo de Verificação de uma Invariante

Uma proof obligation gerada pelo Overture é a seguinte:

No.	PO Name	Type
24	FitnessApp‘addChallenge(Challenge)	state invariant holds

Código em análise:

```
-- Logs the user out, updating the state variables.
public logout: () ==> ()
logout() == (
  loggedInUser := nil;
  loggedInAdmin := nil;
)
pre loggedInUser <> nil
post loggedInUser = nil and loggedInAdmin = nil;
```

A invariante em análise é a seguinte:

```
inv (loggedInUser = nil and loggedInAdmin = nil) or (loggedInAdmin <> nil and loggedInUser
    = loggedInAdmin) or (loggedInUser <> nil and loggedInAdmin = nil);
```

Depois da execução da operação, a seguinte condição pode ser verificada:

```
loggedInUser = nil and loggedInAdmin = nil
```

É preciso provar que a condição acima implica a invariante, ou seja, é preciso verificar a seguinte condição:

```
(loggedInUser = nil and loggedInAdmin = nil) => (loggedInUser = nil and loggedInAdmin =
    nil) or (loggedInAdmin <> nil and loggedInUser = loggedInAdmin) or (loggedInUser <> nil
    and loggedInAdmin = nil)
```

Através da condição acima é possível verificar que é claramente verdadeira.

6 Geração de Código

6.1 Main

```
import FitnessApp.Admin;
import FitnessApp.FitnessApp;

@SuppressWarnings("all")
public class Main {
    public static void main(String[] args) {
        FitnessApp fitnessApp = new FitnessApp();

        fitnessApp.addAdmin(
                new Admin("Admin", "Admin", "a@d.min",
                    "adminadmin", 70, 1.85, new
                    Object()));

        CommandLineInterface cli = new
            CommandLineInterface(fitnessApp);
        cli.mainMenu();
    }
}
```

6.2 CommandLineInterface

```
import FitnessApp.*;
import FitnessApp.Types.Date;
import FitnessApp.Types.DateTime;
import FitnessApp.Types.Point;
import FitnessApp.Types.Time;
import org.overture.codegen.runtime.VDMSeq;
import org.overture.codegen.runtime.VDMSet;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.Instant;
import java.time.LocalDateTime;
import java.time.temporal.ChronoUnit;
import java.util.AbstractMap.SimpleEntry;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Iterator;
import java.util.Scanner;
import java.util.concurrent.Callable;

public class CommandLineInterface {

    private static final int EMPTY_LINES = 10;

    private Scanner reader = new Scanner(System.in);
    private FitnessApp fitnessApp;
```

```
CommandLineInterface(FitnessApp fitnessApp) {
    this.fitnessApp = fitnessApp;
}

public void mainMenu() {
    printLine();
    System.out.println("Welcome to your favorite
        Fitness app");
    ArrayList<SimpleEntry<String, Callable<Void>>>
        mainMenuEntries = new ArrayList<>();

    while (true) {
        mainMenuEntries.clear();
        addMainMenuEntries(mainMenuEntries);
        printMenuEntries(mainMenuEntries);
        int option = getUserInput(1,
            mainMenuEntries.size() - 1);

        try {
            mainMenuEntries.get(option).getValue().call();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

private void loggedInMenu() {
    printEmptyLines(EMPTY_LINES);
    printLine();
    System.out.println("Logged In Menu");
    ArrayList<SimpleEntry<String, Callable<Void>>>
        loggedInMenuEntries = new ArrayList<>();

    while (true) {
        loggedInMenuEntries.clear();
        addLoggedInMenuEntries(loggedInMenuEntries);
        printMenuEntries(loggedInMenuEntries);
        int option = getUserInput(1,
            loggedInMenuEntries.size() - 1);

        try {
            loggedInMenuEntries.get(option).getValue().call();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

private void userWorkoutsMenu() {
    printEmptyLines(EMPTY_LINES);
    printLine();
    System.out.println("Workouts Menu");
```

```
        ArrayList<SimpleEntry<String, Callable<Void>>>
            workoutsMenuEntries = new ArrayList<>();

        while (true) {
            workoutsMenuEntries.clear();
            addWorkoutsMenuEntries(workoutsMenuEntries);
            printMenuEntries(workoutsMenuEntries);
            int option = getUserInput(1,
                workoutsMenuEntries.size() - 1);

            try {
                workoutsMenuEntries.get(option).getValue().call();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    private void userRoutesMenu() {
        printEmptyLines(EMPTY_LINES);
        printLine();
        System.out.println("Routes Menu");
        ArrayList<SimpleEntry<String, Callable<Void>>>
            routesMenuEntries = new ArrayList<>();

        while (true) {
            routesMenuEntries.clear();
            addRoutesMenuEntries(routesMenuEntries);
            printMenuEntries(routesMenuEntries);
            int option = getUserInput(1,
                routesMenuEntries.size() - 1);

            try {
                routesMenuEntries.get(option).getValue().call();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    private void manageUserFriendsMenu() {
        printEmptyLines(EMPTY_LINES);
        printLine();
        System.out.println("Friends Menu");
        ArrayList<SimpleEntry<String, Callable<Void>>>
            friendsMenuEntries = new ArrayList<>();

        while (true) {
            friendsMenuEntries.clear();
            addFriendsMenuEntries(friendsMenuEntries);
            printMenuEntries(friendsMenuEntries);
            int option = getUserInput(1,
```

```
                friendsMenuEntries.size() - 1);

            try {
                friendsMenuEntries.get(option).getValue().call();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    private void challengesMenu() {
        printEmptyLines(EMPTY_LINES);
        printLine();
        System.out.println("Challenges Menu");
        ArrayList<SimpleEntry<String, Callable<Void>>>
            challengesMenuEntries = new ArrayList<>();

        while (true) {
            challengesMenuEntries.clear();
            addChallengesMenuEntries(challengesMenuEntries);
            printMenuEntries(challengesMenuEntries);
            int option = getUserInput(1,
                challengesMenuEntries.size() - 1);

            try {
                challengesMenuEntries.get(option).getValue().call();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    private void
        addMainMenuEntries(ArrayList<SimpleEntry<String,
        Callable<Void>>> mainMenuEntries) {
        if (!fitnessApp.isLoggedIn()) {
            mainMenuEntries.add(new
                SimpleEntry<>("Create Account", () -> {
                createAccountMenu();
                return null;
            }));
            mainMenuEntries.add(new
                SimpleEntry<>("Login", () -> {
                loginMenu();
                return null;
            }));
        } else {
            mainMenuEntries.add(new
                SimpleEntry<>("Logout", () -> {
                fitnessApp.logout();
                loginMenu();
                return null;
```

```
            }));
        }

        mainMenuEntries.add(new SimpleEntry<>("Exit", ()
            -> {
            System.exit(0);
            return null;
        }));
    }

    private void
        addLoggedInMenuEntries(ArrayList<SimpleEntry<String,
        Callable<Void>>> loggedInMenuEntries) {
        if (fitnessApp.isLoggedIn()) {
            loggedInMenuEntries.add(new
                SimpleEntry<>("Challenges", () -> {
                challengesMenu();
                return null;
            }));
            loggedInMenuEntries.add(new
                SimpleEntry<>("My Workouts", () -> {
                userWorkoutsMenu();
                return null;
            }));
            loggedInMenuEntries.add(new
                SimpleEntry<>("My Routes", () -> {
                userRoutesMenu();
                return null;
            }));
            loggedInMenuEntries.add(new
                SimpleEntry<>("Manage Friends", () -> {
                manageUserFriendsMenu();
                return null;
            }));
            loggedInMenuEntries.add(new
                SimpleEntry<>("Logout", () -> {
                fitnessApp.logout();
                printEmptyLines(EMPTY_LINES);
                mainMenu();
                return null;
            }));
        } else {
            mainMenu();
        }
    }

    private void
        addWorkoutsMenuEntries(ArrayList<SimpleEntry<String,
        Callable<Void>>> workoutsMenuEntries) {
        if (fitnessApp.isLoggedIn()) {
            workoutsMenuEntries.add(new
                SimpleEntry<>("View My Workouts", () -> {
```

```
                viewUserWorkoutsMenu();
                return null;
            }));
            workoutsMenuEntries.add(new
                SimpleEntry<>("Start New Workout", () -> {
                startNewWorkoutMenu();
                return null;
            }));
            workoutsMenuEntries.add(new
                SimpleEntry<>("Main Menu", () -> {
                loggedInMenu();
                return null;
            }));
        } else {
            mainMenu();
        }
    }

    private void
        addRoutesMenuEntries(ArrayList<SimpleEntry<String,
        Callable<Void>>> routesMenuEntries) {
        if (fitnessApp.isLoggedIn()) {
            routesMenuEntries.add(new
                SimpleEntry<>("View My Routes", () -> {
                viewUserRoutesMenu();
                return null;
            }));
            routesMenuEntries.add(new
                SimpleEntry<>("Create New Route", () -> {
                createNewRouteMenu();
                return null;
            }));
            routesMenuEntries.add(new
                SimpleEntry<>("Main Menu", () -> {
                loggedInMenu();
                return null;
            }));
        } else {
            mainMenu();
        }
    }

    private void
        addFriendsMenuEntries(ArrayList<SimpleEntry<String,
        Callable<Void>>> friendsMenuEntries) {
        if (fitnessApp.isLoggedIn()) {
            friendsMenuEntries.add(new
                SimpleEntry<>("View My Friends", () -> {
                viewUserFriendsMenu();
                return null;
            }));
            friendsMenuEntries.add(new
```

```
                SimpleEntry<>("Add New Friend", () -> {
                    addNewFriendMenu();
                    return null;
                }));
                friendsMenuEntries.add(new
                    SimpleEntry<>("Main Menu", () -> {
                    loggedInMenu();
                    return null;
                }));
            } else {
                mainMenu();
            }
        }

        private void
            addChallengesMenuEntries(ArrayList<SimpleEntry<String,
            Callable<Void>>> challengeMenuEntries) {
            if (fitnessApp.isLoggedIn()) {
                challengeMenuEntries.add(new
                    SimpleEntry<>("View Challenges", () -> {
                    viewChallengesMenu();
                    return null;
                }));
                if (fitnessApp.isAdminLoggedIn())
                    challengeMenuEntries.add(new
                        SimpleEntry<>("Add Challenge", () -> {
                        createNewChallengeMenu();
                        return null;
                    }));
                challengeMenuEntries.add(new
                    SimpleEntry<>("Main Menu", () -> {
                    loggedInMenu();
                    return null;
                }));
            } else {
                mainMenu();
            }
        }

        private void createAccountMenu() {
            printEmptyLines(EMPTY_LINES);
            printLine();
            System.out.println("Create account menu");
            System.out.print("First name: ");
            String firstName = reader.nextLine();
            System.out.print("Last name: ");
            String lastName = reader.nextLine();
            System.out.print("Email: ");
            String email = reader.nextLine();
            System.out.print("Password: ");
            String password = reader.nextLine();
            System.out.print("Weight: ");
```

```
        double weight =
            Double.parseDouble(reader.nextLine());
        System.out.print("Height: ");
        double height =
            Double.parseDouble(reader.nextLine());
        System.out.print("Gender (Masculine, Feminine):
            ");
        String gender = reader.nextLine();
        fitnessApp.addUser(new User(firstName, lastName,
            email, password, weight, height, gender));
        printEmptyLines(EMPTY_LINES);
    }

    private void loginMenu() {
        printEmptyLines(EMPTY_LINES);
        printLine();
        System.out.println("Login menu");
        System.out.print("Email: ");
        String email = reader.nextLine();
        System.out.print("Password: ");
        String password = reader.nextLine();
        if (fitnessApp.login(email, password)) {
            loggedInMenu();
        } else {
            printEmptyLines(EMPTY_LINES);
            System.out.println("Incorrect email,
                password combination. Please try again.");
        }
        printEmptyLines(EMPTY_LINES);
    }

    private void viewUserWorkoutsMenu() {
        printEmptyLines(EMPTY_LINES);

        User loggedInUser = fitnessApp.getLoggedInUser();
        VDMSet userWorkouts = loggedInUser.getWorkouts();

        if (userWorkouts.size() == 0) {
            System.out.println("No Workouts :(");
            System.out.println("Enter to continue");
            reader.nextLine();
            return;
        }

        Iterator<Workout> it = userWorkouts.iterator();
        int i = 1;
        while (it.hasNext()) {
            Workout workout = it.next();
            System.out.println(i + ": " +
                workout.getTitle());
            System.out.println("  Duration: " +
                workout.getDuration() + " min");
```

```
            System.out.println("  Distance: " +
                workout.getDistance() + " km");
            System.out.println("  Average Rhythm: " +
                workout.getAverageRhythm() + " min/km");
            System.out.println("  Calories Burned: " +
                workout.getCaloriesBurned() + " kcal");
            i++;
        }

        printEmptyLines(EMPTY_LINES);
    }

    private void viewUserRoutesMenu() {
        printEmptyLines(EMPTY_LINES);

        User loggedInUser = fitnessApp.getLoggedInUser();
        VDMSet userRoutes = loggedInUser.getRoutes();

        if (userRoutes.size() == 0) {
            System.out.println("No Routes :(");
            System.out.println("Enter to continue");
            reader.nextLine();
            return;
        }

        Iterator<Route> it = userRoutes.iterator();
        int i = 1;
        while (it.hasNext()) {
            Route route = it.next();
            System.out.println(i + ": " +
                route.getName());
            System.out.println("   Route Distance: " +
                route.getDistance());

            System.out.println("   Points:");
            VDMSeq points = route.getPoints();
            Iterator<Point> ite = points.iterator();
            int j = 1;
            while (ite.hasNext()) {
                Point point = ite.next();
                System.out.println("      " + j + "º
                    Point (Latitude, Longitude): " +
                    point.lat + "," + point.long_);
                j++;
            }
            i++;
        }

        printEmptyLines(EMPTY_LINES);
    }

    private void viewChallengesMenu() {
```

```
        printEmptyLines(EMPTY_LINES);

        VDMSeq challenges = fitnessApp.getChallenges();

        if (challenges.size() == 0) {
            System.out.println("No Challenges :(");
            System.out.println("Enter to continue");
            reader.nextLine();
            return;
        }

        Iterator<Challenge> it = challenges.iterator();
        int i = 1;
        while (it.hasNext()) {
            Challenge challenge = it.next();
            System.out.println(i + ": " +
                challenge.name);
            System.out.println("  " +
                String.format(challenge.getGoalDescription(),
                challenge.goal));
            System.out.println("  Created by: " +
                challenge.getCreator().getFirstName() + "
                " + challenge.getCreator().getLastName());

            i++;
            VDMSeq usersCompleted =
                challenge.getCompleted();
            Iterator<User> ite =
                usersCompleted.iterator();
            if (usersCompleted.size() > 0) {
                System.out.print("Completed by: ");
            }
            while (ite.hasNext()) {
                System.out.println(ite.next().getFirstName()
                    + " ");
            }
            System.out.println();
        }

        printEmptyLines(EMPTY_LINES);
    }

    private void viewUserFriendsMenu() {
        printEmptyLines(EMPTY_LINES);

        VDMSet userFriends =
            fitnessApp.getLoggedInUser().getFriends();

        if (userFriends.size() == 0) {
            System.out.println("You haven't added any
                friends yet:(");
            System.out.println("Enter to continue");
```

```
            reader.nextLine();
            return;
        }

        Iterator<User> it = userFriends.iterator();
        int i = 1;
        while (it.hasNext()) {
            User friend = it.next();
            System.out.println(i + ": " +
                friend.getFirstName() + " " +
                friend.getLastName() + " with email " +
                friend.getEmail());
        }


        printEmptyLines(EMPTY_LINES);
    }

    private void startNewWorkoutMenu() {
        printEmptyLines(EMPTY_LINES);

        User loggedInUser = fitnessApp.getLoggedInUser();

        System.out.print("Workout Name: ");
        String workoutName = reader.nextLine();
        System.out.print("Activity Type (Bycicle,
            Running, Walking): ");
        String activityType = reader.nextLine();
        System.out.println("First point (latitude,
            longitude)");
        Point p = getPoint();

        LocalDateTime initialDate = LocalDateTime.now();
        DateTime dateTime = new DateTime(new
            Date(initialDate.getYear(),
            initialDate.getMonth().getValue(),
            initialDate.getDayOfMonth()),
                new Time(initialDate.getHour(),
                    initialDate.getMinute(),
                    initialDate.getSecond()));

        Workout newWorkout = new Workout(workoutName,
            dateTime, activityType, p);

        while (true) {
            System.out.println("Next Point (latitude,
                longitude) OR s to stop");
            p = getPointOrStop();
            if (p != null) {
                newWorkout.addPoint(p);
            } else {
                break;
```

```
            }
        }

        LocalDateTime endDate = LocalDateTime.now();
        Types.Duration duration = new Types.Duration(
                ChronoUnit.HOURS.between(initialDate,
                    endDate),
                ChronoUnit.MINUTES.between(initialDate,
                    endDate),
                ChronoUnit.SECONDS.between(initialDate,
                    endDate));

        newWorkout.endWorkout(loggedInUser, duration);
        loggedInUser.addWorkout(newWorkout);

        VDMSeq challenges = fitnessApp.getChallenges();

        Iterator<Challenge> it = challenges.iterator();
        while (it.hasNext()) {
            Challenge challenge = it.next();
            if (challenge.verifyChallenge(newWorkout)) {
                challenge.addCompletedUser(loggedInUser);
                System.out.println("You completed
                    challenge with name " +
                    challenge.name);
            }
        }

        printEmptyLines(EMPTY_LINES);
    }

    /**
     * Gets a Point from the command line.
     *
     * @return {Point}
     */
    private Point getPoint() {
        while (true) {
            try {
                System.out.print("Latitude: ");
                double latitude =
                    Double.parseDouble(reader.nextLine());
                System.out.print("Longitude: ");
                double longitude =
                    Double.parseDouble(reader.nextLine());
                return new Point(latitude, longitude);
            } catch (NumberFormatException e) {
                System.out.println("Invalid number
                    format. Please try again.");
            }
        }
    }
```

```
/**
 * Gets a Point from the command line or null if the
    user stops the input.
 *
 * @return {Point}
 */
private Point getPointOrStop() {
    while (true) {
        try {
            System.out.print("Latitude: ");
            String nextLine = reader.nextLine();

            if (nextLine.equals("s")) {
                return null;
            }

            double latitude =
                Double.parseDouble(nextLine);

            System.out.print("Longitude: ");
            nextLine = reader.nextLine();
            double longitude =
                Double.parseDouble(reader.nextLine());

            if (nextLine.equals("s")) {
                return null;
            }

            return new Point(latitude, longitude);
        } catch (NumberFormatException e) {
            System.out.println("Invalid number
                format. Please try again.");
        }
    }
}

private void createNewRouteMenu() {
    printEmptyLines(EMPTY_LINES);

    User loggedInUser = fitnessApp.getLoggedInUser();

    System.out.print("Route Name: ");
    String routeName = reader.nextLine();

    VDMSeq points = new VDMSeq();

    System.out.println("First point (latitude,
        longitude)");
    points.add(getPoint());

    while (true) {
```

```
            System.out.println("Next Point (latitude,
                longitude) OR s to stop");

            Point p = getPointOrStop();
            if (p != null) {
                points.add(p);
            } else {
                break;
            }
        }

        Route newRoute = new Route(routeName, points);
        loggedInUser.addRoute(newRoute);

        printEmptyLines(EMPTY_LINES);
    }

    private void addNewFriendMenu() {
        printEmptyLines(EMPTY_LINES);

        User loggedInUser = fitnessApp.getLoggedInUser();

        System.out.print("New Friend Email: ");
        String friendName = reader.nextLine();

        VDMSet users = fitnessApp.getUsers();

        Iterator<User> it = users.iterator();
        boolean found = false;
        while (it.hasNext()) {
            User user = it.next();
            if (user.getEmail().equals(friendName) &&
                !loggedInUser.equals(user)) {
                loggedInUser.addFriend(user);
                System.out.println("Friend Added");
                found = true;
                break;
            }
        }

        if (!found) {
            System.out.println("User Not Found!");
        }

        printEmptyLines(EMPTY_LINES);
    }

    private void createNewChallengeMenu() {
        printEmptyLines(EMPTY_LINES);

        Admin admin = fitnessApp.getLoggedInAdmin();
```

```
System.out.print("Challenge Name: ");
String challengeName = reader.nextLine();
System.out.print("Challenge description: ");
String challengeDescription = reader.nextLine();
SimpleDateFormat simpleDateFormat = new
    SimpleDateFormat("dd-MM-yyyy");

System.out.print("End date (dd-mm-yyy): ");
java.util.Date endDate = null;

try {
    endDate =
        simpleDateFormat.parse(reader.nextLine());

    if
        (endDate.compareTo(java.util.Date.from(Instant.now()))
        <= 0) {
        throw new IllegalArgumentException();
    }
} catch (Exception e) {
    boolean validDate = false;
    while (!validDate) {
        System.out.print("Invalid date. Please
            enter end date (dd-mm-yyyy): ");
        try {
            endDate =
                simpleDateFormat.parse(reader.nextLine());
            validDate = true;
        } catch (ParseException e1) {
            e1.printStackTrace();
        }
    }
}

Challenge newChallenge = null;
boolean validChallenge = false;
while (!validChallenge) {
    System.out.print("Type of Activity (0 ->
        distance(km) | 1 -> number of
        calories(kcal) | 2 -> rhythm(min/km)): ");
    int typeOfActivity =
        Integer.parseInt(reader.nextLine());

    System.out.print("Challenge Goal: ");
    double challengeGoal =
        Double.parseDouble(reader.nextLine());

    LocalDateTime initialDate =
        LocalDateTime.now();

    Calendar calendar = Calendar.getInstance();
    calendar.setTime(endDate);
```

```
switch (typeOfActivity) {
    case 0:
        newChallenge = new
            DistanceChallenge(admin,
            challengeName,
            challengeDescription,
                new
                    Date(initialDate.getYear(),
                    initialDate.getMonth().getValue(),
                    initialDate.getDayOfMonth()),
                new
                    Date(calendar.get(Calendar.YEAR),
                    calendar.get(Calendar.MONTH),
                    calendar.get(Calendar.DAY_OF_MONTH)),
                    challengeGoal);
        validChallenge = true;
        break;
    case 1:
        newChallenge = new
            CalorieChallenge(admin,
            challengeName,
            challengeDescription,
                new
                    Date(initialDate.getYear(),
                    initialDate.getMonth().getValue(),
                    initialDate.getDayOfMonth()),
                new
                    Date(calendar.get(Calendar.YEAR),
                    calendar.get(Calendar.MONTH),
                    calendar.get(Calendar.DAY_OF_MONTH)),
                    challengeGoal);
        validChallenge = true;
        break;
    case 2:
        newChallenge = new
            RhythmChallenge(admin,
            challengeName,
            challengeDescription,
                new
                    Date(initialDate.getYear(),
                    initialDate.getMonth().getValue(),
                    initialDate.getDayOfMonth()),
                new
                    Date(calendar.get(Calendar.YEAR),
                    calendar.get(Calendar.MONTH),
                    calendar.get(Calendar.DAY_OF_MONTH)),
                    challengeGoal);
        validChallenge = true;
        break;
    default:
        System.out.println("Invalid type of
```

```
                        activity. Try again: ");
                    break;
                }
            }

            fitnessApp.addChallenge(newChallenge);

            printEmptyLines(EMPTY_LINES);
        }

        private void printLine() {
            System.out.println("========================================");
        }

        private void
            printMenuEntries(ArrayList<SimpleEntry<String,
            Callable<Void>>> menuEntries) {
            for (int i = 0; i < menuEntries.size(); i++) {
                System.out.println((i + 1) + ": " +
                    menuEntries.get(i).getKey());
            }
        }

        private int getUserInput(int bottomBound, int
            upperBound) {
            System.out.print("Choose an option: ");
            int option = Integer.parseInt(reader.nextLine());

            if (option < bottomBound && option > upperBound)
                {
                System.out.println("Invalid option");
                option = getUserInput(bottomBound,
                    upperBound);
            }

            return option - 1;
        }

        private void printEmptyLines(int linesToPrint) {
            for (int i = 0; i < linesToPrint; i++) {
                System.out.println();
            }
        }
    }
```

7 Conclusões

7.1 Resultados Obtidos

O grupo conseguiu obter os resultados pretendidos, tendo modelado corretamente a especificação pedida. O programa final permite um utilizador criar uma conta, iniciar a sua sessão, criar um novo treino, ver os seus antigos treinos, assim como participar em desafios disponíveis a todos os utilizadores do sistema. Também é possível administrar o sistema devido ao sistema de privilégios que foi criado para esse propósito. Todas estas funcionalidades foram implementadas em VDM++ com as condições necessárias para modelar corretamente o comportamento desejado.

7.2 Possíveis Melhoramentos

A equipa cumpriu os objetivos especificados da modelação em VDM++, mas a interface com o utilizador é apenas feita através de uma linha de comandos e poderia ser bastante melhorada ao criar uma interface para uma plataforma móvel, que seria onde esta aplicação seria mais útil.

7.3 Contribuição

O trabalho exercido pelos membros do grupo foi de valor equivalente.

8 Referências

1. Informações disponibilizadas pelos docentes da unidade curricular de Métodos Formais em Engenharia de Software.
2. Overture tool website, http://overturetool.org/
3. Map My Fitness Application, www.mapmyfitness.com