

Relatório TP | Programação Orientada aos Objetos

Grupo 11 | 2023/2024

João Lobo
(A104356)



Mariana Rocha
(A90817)



Rita Camacho
(A104439)



Índice

1. Introdução	3
1.1. Utilizadores	3
1.2. Atividades	3
1.3. Planos de Treino	3
2. Arquitetura da Aplicação	4
2.1. Visão geral da Arquitetura das Classes	4
2.2. Encapsulamento e abstração da implementação	4
2.3. <i>Models</i>	5
2.3.1. Atividades	5
2.3.2. Utilizadores	7
2.3.3. Plano de Treinos	8
2.4. <i>Controllers</i>	9
2.4.1. Atividades	9
2.4.2. Utilizadores	9
2.4.3. Plano de Treinos	10
2.4.4. Tempo	11
2.4.5. <i>Sportify</i>	11
2.5. <i>Views</i>	12
2.5.1. Componentes	12
2.5.2. <i>Input</i>	12
2.5.3. <i>Main</i>	12
2.5.4. Atividades	12
2.5.5. Planos de treino	13
2.5.6. Utilizador	14
2.5.7. <i>Admin</i>	14
2.5.8. Estatísticas	14
2.6. Diagrama de Classes	15
3. A Aplicação <i>Sportify</i>	16
3.1. Funcionalidades	16
3.1.1. Requisitos base de gestão das entidades	16
3.1.2. Estatísticas	16
3.1.3. Noção de atividade <i>Hard</i>	16
3.1.4. Gerar um plano de treinos	16
3.1.5. Salvaguarda do estado da aplicação	17
3.2. Aspetos Relevantes	18
3.2.1. Documentação	18
3.2.2. Testes unitários	18
4. Conclusão	18

1. Introdução

O presente relatório documenta o trabalho desenvolvido como componente prática, a avaliar na Unidade Curricular de Programação Orientada aos Objetos, pertencente ao 2º Ano da Licenciatura em Engenharia Informática, realizada no ano letivo 2023/2024, na Universidade do Minho.

1.1. Utilizadores

A nossa aplicação permite diferentes perfis de utilizadores: profissionais, amadores e ocasionais. Cada perfil tem diversos atributos, como frequência cardíaca média, que influenciam o cálculo das calorias gastas durante as atividades. O registo das atividades fornece dados importantes sobre o desempenho de cada utilizador.

1.2. Atividades

Os utilizadores podem registar na aplicação atividades que realizaram ou vão realizar. Estas atividades estão pré-definidas na aplicação e englobam diferentes categorias: as que implicam distância, distância e altimetria, repetições e repetições com pesos. As atividades estão associadas a uma data e hora de realização, atributos que variam consoante a sua categoria e uma breve descrição.

1.3. Planos de Treino

Por fim, o programa possui a capacidade de criar/gerar planos de treino associados aos utilizadores, sendo que estes listam atividades a realizar a partir de um data específica, bem como o seu número de repetições.

2. Arquitetura da Aplicação

2.1. Visão geral da Arquitetura das Classes

Nesta aplicação, adotamos o padrão de arquitetura *Model-View-Controller* (MVC) para organizar e estruturar o nosso código de forma eficiente e modular. Esta abordagem permite uma separação clara das preocupações e facilita a manutenção e evolução do sistema.

Os *models* são responsáveis por encapsular os dados da aplicação e definir as operações que podem ser realizadas sobre esses dados. Cada modelo possui métodos para aceder e manipular os seus atributos, garantindo a integridade e consistência dos dados.

Os *controllers* atuam como intermediários entre os modelos e as vistas. Estes coordenam as interações do utilizador com os *models*, processando as solicitações e atualizando o estado da aplicação conforme necessário. Os *controllers* também são responsáveis por gerir a lógica de negócios da aplicação, garantindo que as operações sejam executadas corretamente e de forma eficiente.

As *views* são todas as partes da aplicação que têm interação com o utilizador, estas tratam da apresentação dos dados e da interação com o utilizador. Isso inclui a interface do utilizador (UI) e qualquer entrada ou saída (IO) do sistema. As *views* comunicam com os *controllers* para obter e exibir os dados relevantes para o utilizador.

2.2. Encapsulamento e abstração da implementação

Durante o processo de desenvolvimento da aplicação, priorizamos o uso de técnicas avançadas de encapsulamento para garantir a integridade dos dados e a manutenção da consistência do programa. Isto foi alcançado através da definição cuidadosa de *getters* e *setters* públicos nas classes, permitindo que os detalhes internos de implementação permaneçam ocultos e protegidos de modificações inadvertidas.

Além disso, enfatizamos a importância da abstração na construção do sistema. Utilizámos classes abstratas para definir modelos genéricos e estruturas comuns que representam entidades na aplicação. Esta abstração permitiu uma maior flexibilidade e extensibilidade no código, facilitando a introdução de novas funcionalidades e adaptações futuras com um mínimo de impacto sobre o restante do sistema.

Ao adotar esses princípios de programação orientada a objetos, conseguimos criar uma base sólida e modular para a nossa aplicação, promovendo a reutilização de código, a manutenção e a escalabilidade do projeto a longo prazo.

2.3. Models

Nesta secção, iremos abordar os modelos da nossa aplicação, que abrangem uma variedade de classes relacionadas com as atividades, utilizadores e planos de treino. Cada conjunto de classes desempenha um papel crucial na estrutura da nossa aplicação. Vamos analisá-las destacando as suas principais características e contribuições para a funcionalidade da aplicação.

2.3.1. Atividades

A arquitetura das classes relacionadas com as atividades na nossa aplicação foi projetada com base em princípios de hierarquia e herança, visando oferecer uma estrutura flexível e extensível para representar diferentes tipos de atividades físicas. O objetivo central foi criar uma abordagem modular que permitisse a fácil extensão e personalização das atividades, garantindo uma maior reutilização de código e uma melhor organização do sistema.

Inicialmente, temos a classe abstrata `Activity`, que serve como base para todas as atividades subsequentes. Esta classe encapsula atributos comuns a todas as atividades, como **código, descrição, duração, frequência cardíaca média durante a atividade e data e hora**. Além disso, define um método abstrato `calculateCaloriesConsume`, responsável por calcular o consumo de calorias da atividade para um determinado utilizador.

Para representar as atividades que envolvem repetições, criamos a classe abstrata `ActivityRepetitions`, que estende a classe `Activity`. Esta classe adiciona um atributo para o número de repetições e declara novamente o método `calculateCaloriesConsume`.

Para atividades que envolvem o levantamento de pesos, introduzimos a classe abstrata `ActivityRepetitionsWeight`, que estende a classe `ActivityRepetitions`. Este nível de hierarquia permite a distinção entre atividades com e sem a utilização de pesos, mantendo uma estrutura coesa e modular.

Além disso, para atividades que envolvem distância percorrida, introduzimos a classe abstrata `ActivityDistance`, que estende a classe `Activity`. Esta classe adiciona um atributo para a distância percorrida e declara novamente o método `calculateCaloriesConsume`.

Para atividades que envolvem altitude e distância, estabelecemos a classe abstrata `ActivityDistanceAltitude`, que estende a classe `ActivityDistance`. Esta classe acrescenta um atributo para a altitude e declara novamente o método do cálculo do consumo de calorias da atividade.

Para exemplificar a aplicação dessa arquitetura, implementamos classes concretas como `Burpee`, `Rowing` e `RoadCycling`, que herdam e especializam as funcionalidades das classes abstratas mencionadas e são nestas classes que o método `calculateCaloriesConsume` é definido de acordo com o utilizador e com a atividade que está a ser realizada. Além disso, introduzimos a interface `Hard` (explorada nas funcionalidades da aplicação), que

define quais atividades são consideradas mais difíceis, oferecendo uma maneira de categorizar e distinguir as atividades com base na intensidade.

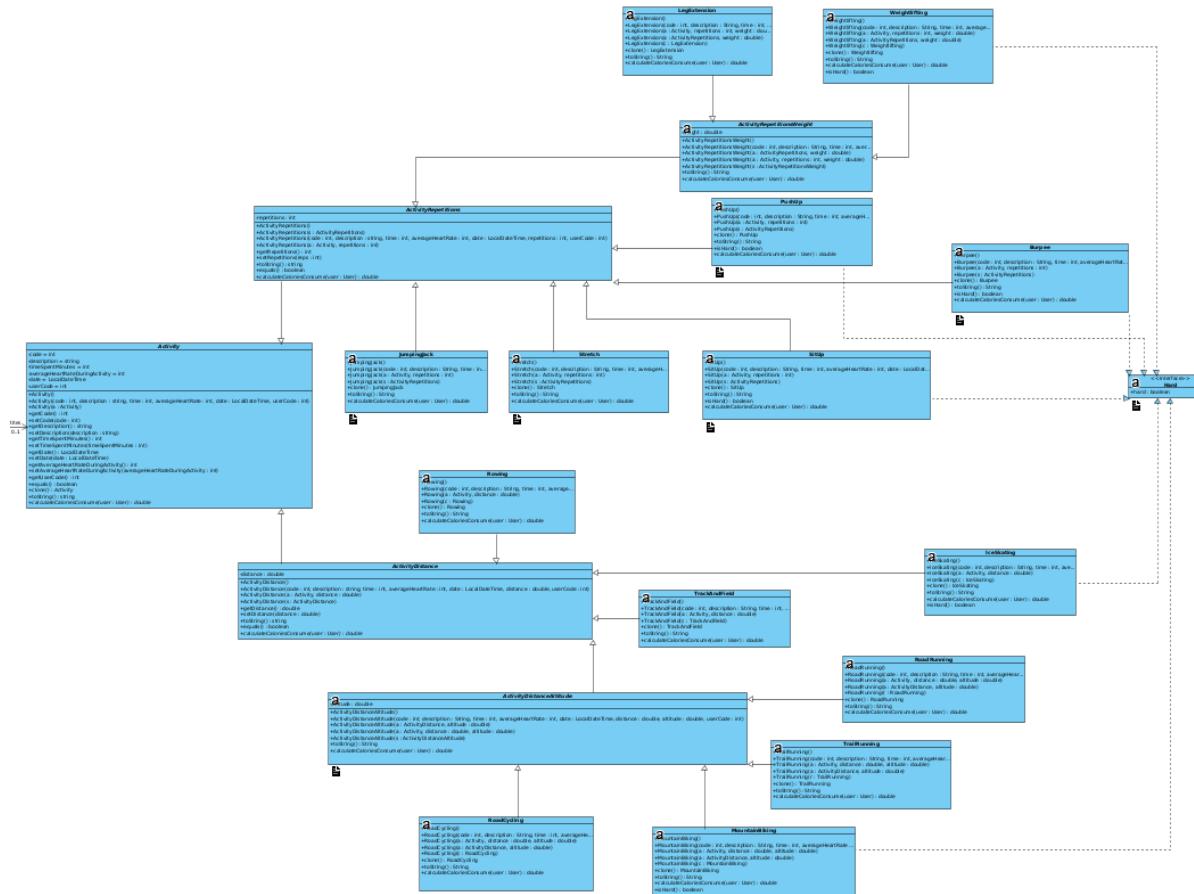


Figura 1: Arquitetura das Atividades

Para facilitar a manipulação e configuração das atividades, foram criadas *enums* **ActivityProperty** e **ActivityType**, que definem as propriedades e os tipos de atividades disponíveis, respectivamente. A *enum* **ActivityProperty** lista os diferentes atributos que podem ser associados a uma atividade, como descrição, tempo gasto, frequência cardíaca média, entre outros. Já a *enum* **ActivityType** enumera os diversos tipos de atividades suportadas pela aplicação, como repetições, repetições com pesos, distância e altitude. Cada tipo de atividade é associado a uma implementação específica e suas propriedades são definidas de acordo com suas características.

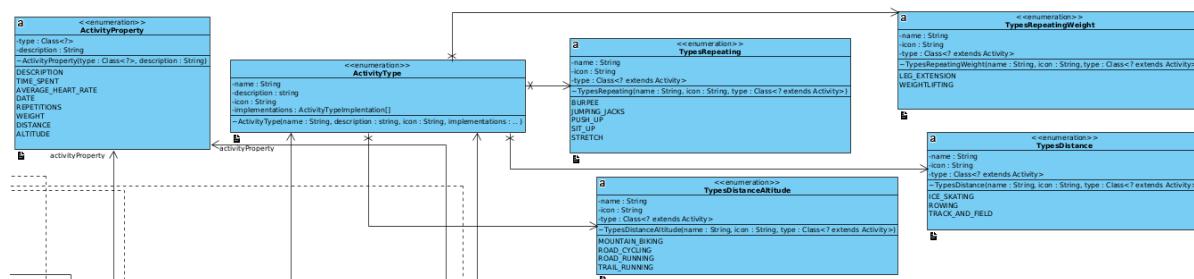


Figura 2: Arquitetura das Atividades (Continuação)

A utilização de múltiplas classes abstratas e uma hierarquia bem definida permite uma melhor organização do código e facilita a extensibilidade e manutenção do sistema, proporcionando uma base sólida para representar uma ampla variedade de atividades físicas com diferentes características e atributos.

2.3.2. Utilizadores

A arquitetura das classes relacionadas aos utilizadores na nossa aplicação foi delineada para acomodar três diferentes tipos de utilizadores: casuais, amadores e profissionais. Cada tipo de utilizador possui características específicas que influenciam o cálculo do fator multiplicativo a ser utilizado no consumo de calorias das suas atividades físicas, tendo em conta o seu Índice de Massa Corporal (IMC). A estrutura das classes foi concebida com o intuito de oferecer flexibilidade para representar diferentes perfis de utilizadores e adaptar-se às suas necessidades individuais.

Em primeiro lugar, temos a classe abstrata `User`, que serve como base para todos os utilizadores. Esta classe encapsula atributos comuns a todos os utilizadores, como **código**, **nome**, **morada**, **email**, **frequência cardíaca média**, **peso** e **altura**. Além disso, esta classe define um método abstrato `calculateCaloriesFactor`, responsável por calcular o fator multiplicativo a ser aplicado nas atividades com base no IMC do utilizador.

Para representar os utilizadores amadores, foi desenvolvida a classe `Amateur`, que estende a classe `User`. Esta classe implementa o método `calculateCaloriesFactor` de acordo com as diretrizes estabelecidas para utilizadores amadores, ajustando o fator multiplicativo com base no IMC do utilizador.

Da mesma forma, para os utilizadores casuais, criamos a classe `Casual`, que também estende a classe `User`. Esta classe define o método `calculateCaloriesFactor` para refletir as especificidades dos utilizadores casuais, adaptando o fator multiplicativo de acordo com o IMC do utilizador.

Por fim, para os utilizadores profissionais, implementamos a classe `Professional`, que herda da classe `User`. Esta classe define o método `calculateCaloriesFactor` para atender às necessidades dos utilizadores profissionais, ajustando o fator multiplicativo com base no IMC do utilizador e nas exigências associadas ao seu perfil.

A utilização de múltiplas classes para representar diferentes tipos de utilizadores permite uma melhor organização do código e uma adaptação mais precisa às características individuais de cada utilizador. Além disso, a estrutura modular facilita a extensibilidade do sistema, possibilitando a inclusão de novos tipos de utilizadores com relativa facilidade.

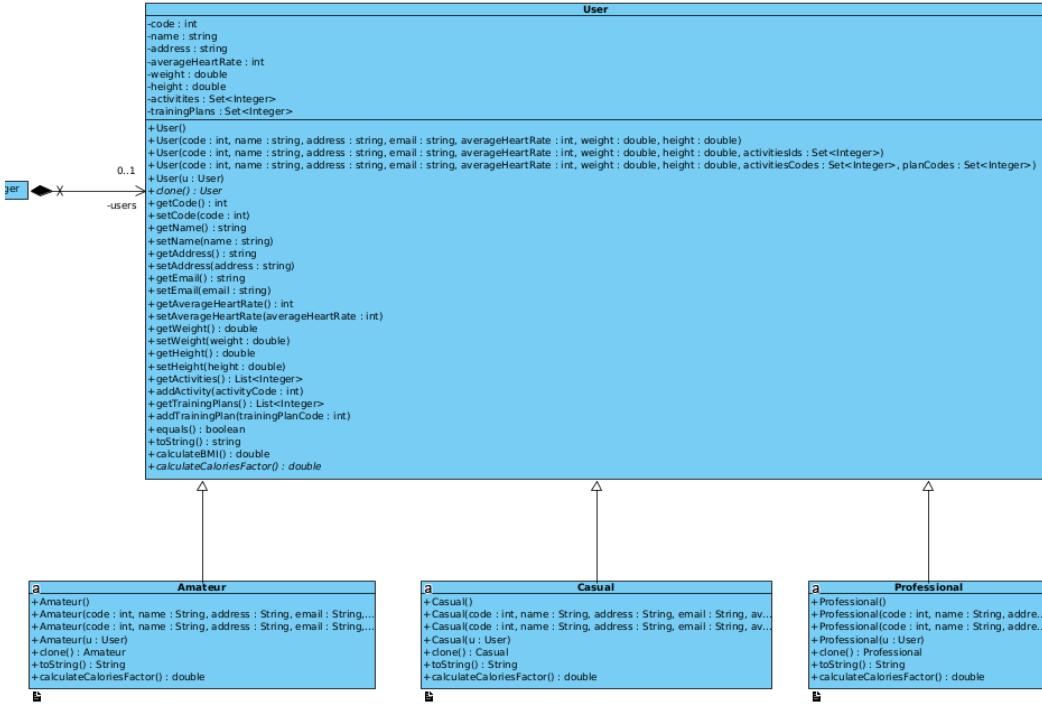


Figura 3: Arquitetura dos Utilizadores

2.3.3. Plano de Treinos

A classe **TrainingPlan** encapsula os dados relacionados com os planos de treino e define as operações que podem ser realizadas sobre esses dados.

Este modelo inclui atributos como a data de início do plano (**startDate**) e os códigos das atividades desse plano de treino (**activities**).

Além disso, a classe **TrainingPlan** oferece métodos para adicionar e remover atividades do plano, o que permite uma manipulação eficiente dos dados. A implementação de métodos como **clone()** e **toString()** facilita a cópia e representação do plano de treino em diferentes contextos.

Em resumo, a classe **TrainingPlan** desempenha um papel fundamental no modelo MVC do projeto *Sportify*, encapsulando os dados dos planos de treino do utilizador e fornecendo métodos para manipular esses dados de forma eficiente e segura.

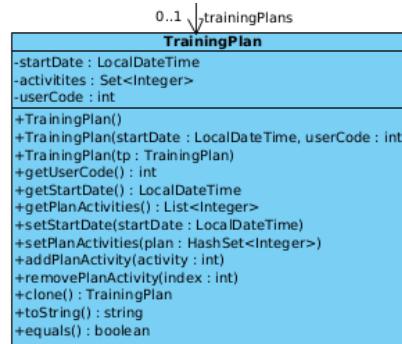


Figura 4: Arquitetura dos Planos de Treino

2.4. Controllers

Os *controllers* desempenham um papel fundamental na coordenação das operações entre os modelos e as vistas. Os *controllers* atuam como intermediários, onde recebem as interações do utilizador e manipulam os dados correspondentes nos modelos subjacentes. Desta forma, cada *controller* é responsável por gerir uma parte específica da lógica de negócios da aplicação.

2.4.1. Atividades

O *controller* das atividades desempenha um papel importante na gestão de todas as operações relacionadas com as atividades físicas registadas no sistema. Este é responsável por coordenar a interação entre as vistas e os modelos de atividades, esta componente é crucial para garantir a consistência e a integridade dos dados.

Este oferece uma variedade de funcionalidades, incluindo a inserção de novas atividades, a visualização de atividades existentes, a atualização de informações e a remoção de atividades conforme necessário. Uma característica importante é a capacidade de realizar operações específicas, como obter todas as atividades entre datas específicas ou visualizar todas as atividades associadas a um utilizador em particular.

Além disso, o *controller* das atividades implementa um mecanismo de validação para garantir que os dados inseridos estejam corretos e consistentes. Por exemplo, ao adicionar uma nova atividade, verifica-se se todos os campos obrigatórios estão preenchidos e se os valores fornecidos são válidos.

Com uma estrutura modular e bem organizada, o *controller* das atividades facilita a extensibilidade e a manutenção do sistema.

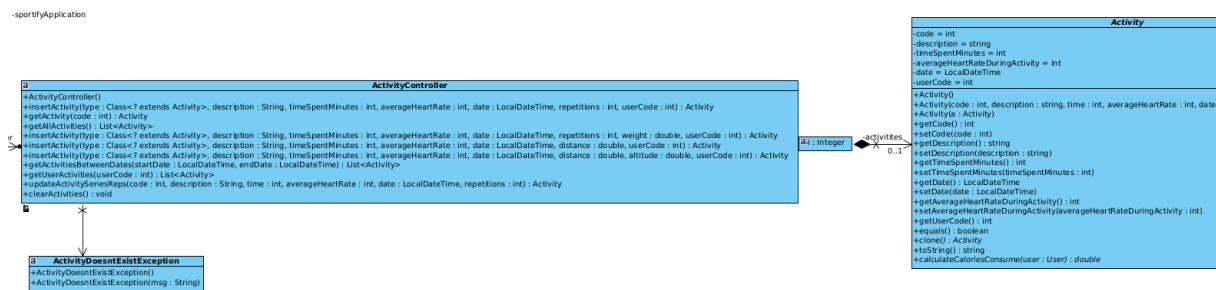


Figura 5: Controller das Atividades

2.4.2. Utilizadores

O *controller* dos utilizadores gera todas as operações relacionadas com os utilizadores registados. Este *controller* desempenha um papel central na criação, atualização e remoção de utilizadores, garantindo a integridade dos dados e facilitando a interação com o sistema.

Uma das funcionalidades principais deste *controller* é a capacidade de inserir novos utilizadores, categorizando-os como amadores, casuais ou profissionais com base nas suas

características e necessidades específicas. Além disso, o *controller* permite a visualização de utilizadores existentes por meio de consultas ao sistema, facilitando o acesso às informações de perfil dos utilizadores quando necessário.

Outro aspecto importante é a gestão das atividades e planos de treino associados a cada utilizador. O *controller* permite a adição de novas atividades e planos de treino aos utilizadores, garantindo que estejam corretamente associados e atualizados conforme necessário.

Com uma estrutura sólida e organizada, o *controller* dos utilizadores fornece funcionalidades abrangentes e uma gestão eficiente dos utilizadores, este componente contribui significativamente para a experiência global do utilizador e para a eficácia do sistema como um todo.

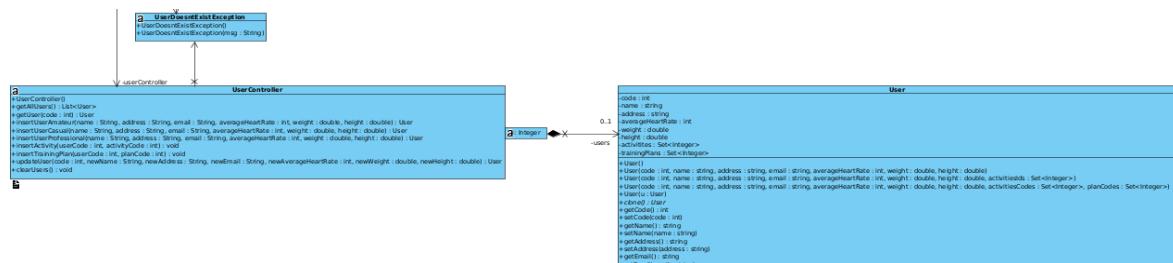


Figura 6: Controller dos Utilizadores

2.4.3. Plano de Treinos

O *controller* dos planos de treino é responsável por gerir todas as operações relacionadas com os planos de treino disponíveis. Esta desempenha um papel crucial na criação, visualização e gestão dos planos de treino, assegurando que estes estejam acessíveis para os utilizadores do sistema.

Uma das funcionalidades principais deste *controller* é a capacidade de obter todos os planos de treino existentes no sistema, fornecendo uma visão abrangente dos planos de treino. Além disso, o *controller* permite a visualização de planos de treino específicos com base nos seus códigos únicos, garantindo uma pesquisa eficiente e precisa.

Com uma estrutura sólida e organizada, o *controller* dos planos de treino disponibiliza de funcionalidades abrangentes e uma gestão eficaz dos planos de treino, este componente contribui significativamente para a eficácia do sistema e para a experiência global do utilizador.

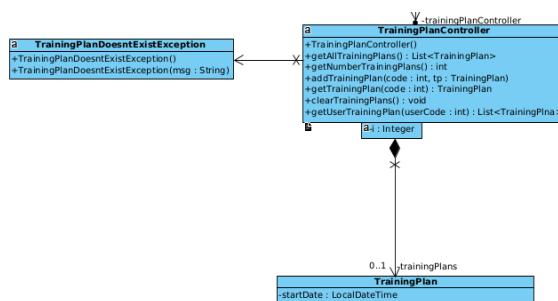


Figura 7: Controller dos Planos de Treino

2.4.4. Tempo

O *controller* do tempo desempenha um papel crucial na gestão do tempo da aplicação, fornecendo funcionalidades para controlar e manipular a data e hora atuais. Este é responsável por garantir que a aplicação mantenha uma noção precisa do tempo, facilitando diversas operações que dependem de cálculos temporais.

Uma das funcionalidades principais deste *controller* é a capacidade de avançar no tempo. Além disso, o *controller* oferece métodos para definir uma data e hora específicas, proporcionando flexibilidade no ajuste do tempo dentro da aplicação.

Ao manter uma estrutura simples e eficiente, o TimeController garante uma gestão confiável do tempo, essencial para a execução precisa de diversas funcionalidades da aplicação.

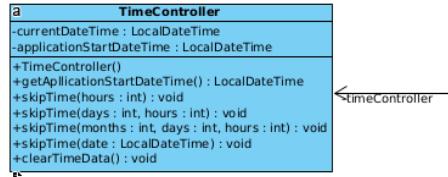


Figura 8: Controller do Tempo

2.4.5. *Sportify*

O *SportifyController* é o núcleo dos *controllers*, onde engloba todos os outros *controllers*. Este *controller* centraliza o acesso aos dados e funcionalidades relacionadas aos utilizadores, atividades, planos de treino e gestão do tempo. A sua estrutura modular e organizada permite a execução eficiente de diversas tarefas, desde a manipulação de dados individuais até a análise de estatísticas avançadas.

Ao integrar os *controllers* de utilizadores, atividades, planos de treino e tempo, o *SportifyController* oferece uma interface unificada para interagir com diferentes partes do sistema. Isso simplifica o desenvolvimento e manutenção da aplicação, promovendo uma arquitetura limpa e modular. Além disso, o *SportifyController* implementa funcionalidades avançadas, como cálculo de estatísticas de atividades e gestão de tempo.

A flexibilidade e extensibilidade deste *controller* tornam-no uma componente fundamental para o funcionamento eficaz do *Sportify*.

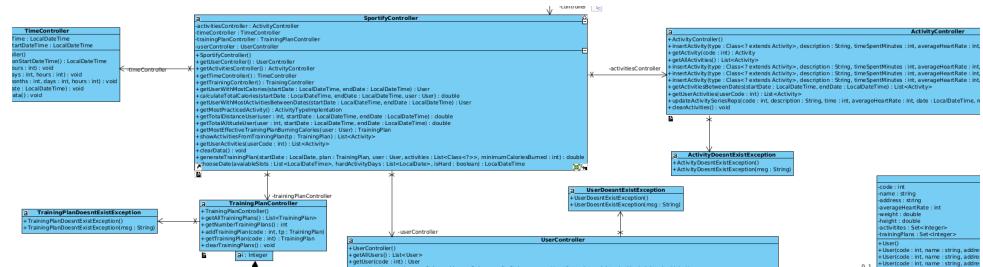


Figura 9: Controller do Sportify

2.5. Views

As *views*, através de comunicação com os *controllers*, responsabilizam-se por apresentar dados e permitir a interação do utilizador com a aplicação via terminal. As vistas estão divididas da mesma forma que os *controllers*, onde cada uma só tem acesso a um escopo específico.

2.5.1. Componentes

De forma a melhorar a qualidade de código e modularidade do programa, optámos por criar componentes de *interface* visual reutilizáveis que são utilizados diversas vezes pelas vistas da aplicação.

2.5.1.1. Menu

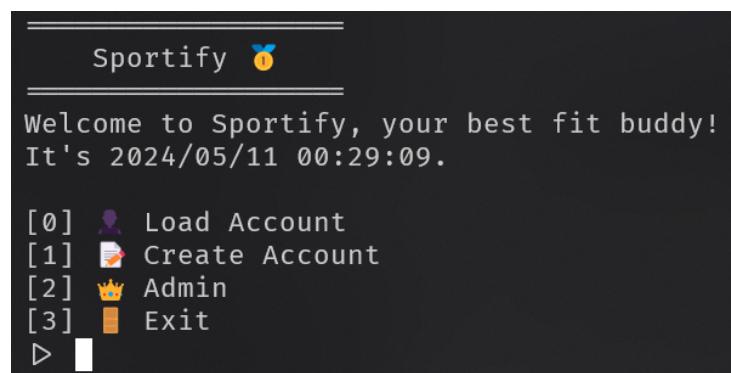
Através de uma lista de opções textuais e ações a tomar, a classe *Menu* apresenta o conjunto e permite que o utilizador selecione uma delas.

2.5.2. Input

A classe *Input* contem uma série de métodos que tratam da obtenção de dados por parte do utilizador e realizam a sua conversão para os tipos de *Java*.

2.5.3. Main

Trata-se da vista inicial do programa, apresenta a data e hora da aplicação e disponibiliza opções ao utilizador.



The screenshot shows a terminal window with a black background and white text. At the top, it displays the application name "Sportify" followed by a yellow trophy icon. Below this, a welcome message reads "Welcome to Sportify, your best fit buddy! It's 2024/05/11 00:29:09.". A menu is presented with four options: "[0] Load Account", "[1] Create Account", "[2] Admin", and "[3] Exit". A small cursor icon is visible at the bottom left of the menu area.

Figura 10: *Main View*

2.5.4. Atividades

Esta vista permite que o utilizador visualize as suas atividades já realizadas e que adicione registos novos. A vista de adição de novo registo de atividade é modular, sendo gerada com base nas classes de atividades presentes no programa.

```

===== Activity Menu =====
[0] NEW Register new Activity
[1] ⏪ View all activities
[2] ⏴ Return
> 

```

Figura 11: Menu da vista de atividades

```

===== Your Activities =====
Ice Skating
Activity:
Code = 6
Description = Ringue de Viseu
Time spent = 45
Average Heart Rate = 87
Date = 2024/05/01 12:40:00
Activity with distance:
Distance = 387.0

```

Figura 12: Listagem das atividades do utilizador

```

===== Register New Activity =====
[0] ⏵ Repeating
[1] 🚶 Repeating with weights
[2] 🏃 Distance
[3] 🚻 Distance and altitude
[4] ⏴ Return
> 

```

Figura 13: Registo de uma nova atividade (categorias)

```

===== Distance =====
This type of activity is distance-based.
[0] 🎤 Ice skating
[1] 🏊 Rowing
[2] 🏃 Track and field
[3] ⏴ Return
> 0

===== 
Creating new activity of type Ice skating.
Activity description:
> 

```

Figura 14: Registo de uma nova atividade

2.5.5. Planos de treino

A vista de planos de treino permite que o utilizador veja os seus planos de treino, crie um novo plano customizado ou que este seja gerado automaticamente pela aplicação com base nas suas opções.

```

===== Training Plan =====
[0] NEW Create new training plan
[1] ⏳ Generate training plan
[2] ⏪ View all training plans
[3] ⏴ Return
> 

```

Figura 15: Menu da vista de planos de treino

```

===== View Training Plans =====
Here are your training plans:
1. 2024-02-01
   6 activities
Ice Skating
Activity:
Code = 0
Description = repetitions
Time spent = 3070
Average Heart Rate = 110
Date = 2024/02/01 09:00:00
Activity with distance

```

Figura 16: Listagem dos planos do utilizador

```

New Training Plan 🍎
Please enter the following details to create a new training plan:
When do you plan on executing this plan?
Please enter the date in the following format: yyyy-MM-dd
▷ 2024-05-11
[0] Repeating
[1] Repeating with weights
[2] Distance
[3] Distance with weights
    
```

Figura 17: Registo de um novo plano

```

Generate Training Plan 🏃
Please enter the following details to generate a training plan:
When do you plan on executing this plan?
Please enter the date in the following format: yyyy-MM-dd
▷ 2024-05-12
How many calories do you want to burn? (minimum)
▷ 3000
    
```

Figura 18: Geração de um plano de treino

2.5.6. Utilizador

As vistas de utilizador estão relacionadas com as contas da aplicação, nomeadamente a criação destas.

```

Create Account 📝
Hello! As you're new around here, we need you to fill out the following
form in order to create your account! 😊
Please enter your name:
▷ João
Please enter your email:
▷ joao@mail.pt
    
```

Figura 19: Criação de conta

```

Load Account 🧑
Please enter your account code:
▷
    
```

Figura 20: Carregar conta

2.5.7. Admin

De forma a facilitar a configuração da aplicação e visualização das suas métricas gerais, a vista de administrador apresenta diversas opções de interação com o programa.

```

Admin 🤴
It's 2024/05/11 00:29:09.

[0] View Statistics
[1] Clear Data
[2] Skip Time
[3] Return
▷
    
```

Figura 21: Opções de administrador

2.5.8. Estatísticas

A partir da vista de *admin*, acede-se à vista das estatísticas onde é possível obter métricas gerais da aplicação.

```

Statistics 📈
It's 2024/05/11 00:29:09.

[0] User with most calories burned in a period or overall
[1] User with most activities in a period or overall
[2] Most performed activity type
[3] Total distance covered by a user in a period or overall
[4] Total altitude accumulated by a user in a period or overall
[5] Most demanding training plan
[6] List user activities
[7] Return
▷
    
```

Figura 22: Estatísticas da aplicação

2.6. Diagrama de Classes

Como referimos anteriormente, adotámos a arquitetura MVC, além dos componentes tradicionais do MVC, decidimos incluir uma classe extra chamada *Sportify*, que atua como uma fachada para o *SportifyController* e as *views*. Esta classe centraliza o acesso à lógica da aplicação e facilita a integração com as interfaces do utilizador. Ao fornecer um ponto de entrada único para a aplicação, a classe *Sportify* simplifica o fluxo de controle e melhora a coesão do código.

Em suma, a arquitetura MVC adotada promove a modularidade, flexibilidade e reutilização de código, permitindo-nos desenvolver e manter uma aplicação robusta e escalável. Esta abordagem ajuda a garantir uma separação clara das responsabilidades e promove boas práticas de desenvolvimento de *software*.

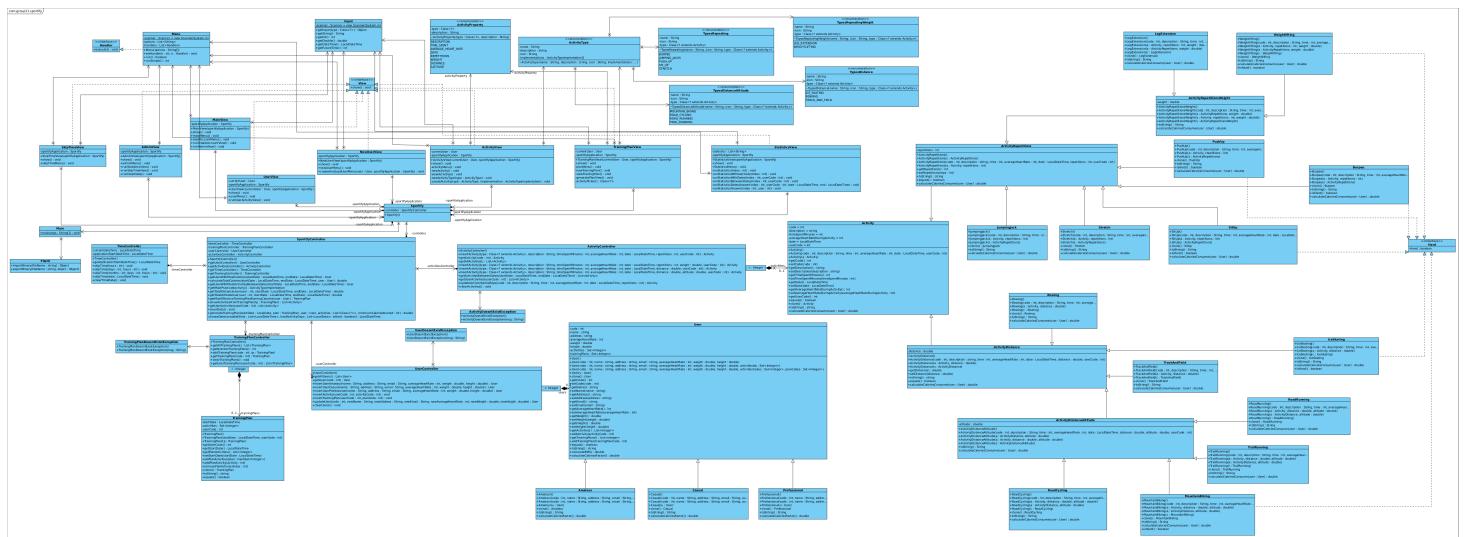


Figura 23: Diagrama de Classes

3. A Aplicação *Sportify*

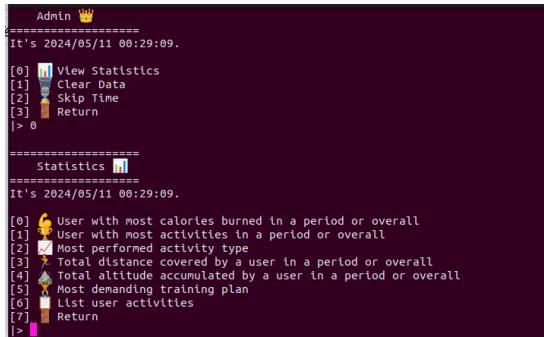
3.1. Funcionalidades

3.1.1. Requisitos base de gestão das entidades

De forma a cumprir os requisitos base para gerir as entidades, implementou-se a criação de utilizadores, atividades e planos de treino, bem como o salto no tempo (havendo realização das atividades ao longo desse período de tempo), algo verificado pelas estatísticas.

3.1.2. Estatísticas

Implementamos 6 estatísticas (sugeridas no enunciado), sendo possível visualizá-las quando selecionado o painel de Admin. Estas encontram-se implementadas no `SportifyController`.



```
Admin 🏃
=====
It's 2024/05/11 00:29:09.

[0] 📈 View Statistics
[1] 🗑 Clear Data
[2] ⏸ Skip Time
[3] 🛡 Return
|> 0

=====
statistics 📈
=====
It's 2024/05/11 00:29:09.

[0] 🏃 User with most calories burned in a period or overall
[1] 🏃 User with most activities in a period or overall
[2] 🚻 Most performed activity type
[3] 🌎 Total distance covered by a user in a period or overall
[4] 🔳 Total altitude accumulated by a user in a period or overall
[5] 🏃 Most demanding training plan
[6] 📊 List user activities
[7] 🛡 Return
|> 1
```

Figura 24: Estatísticas

3.1.3. Noção de atividade *Hard*

Parte das atividades criadas foram definidas como *Hard*, mais concretamente uma de cada tipo (distância, distância e altitude, repetições, repetições com pesos). Foi criada uma interface `Hard` de forma a permitir identificar quais seriam então as atividades deste tipo, implementando-a posteriormente nas classes das mesmas atividades, e acrescentando um método `isHard` que retorna um `bool` (nestes casos, `True`).

Esta interface permite distinguir as atividades quanto ao nível de dificuldade, algo essencial para a posterior implementação da geração/criação de planos de treino, por exemplo.

3.1.4. Gerar um plano de treinos

A funcionalidade de geração de planos de treino do *Sportify* é uma implementação adicional mas importante, pois permite aos utilizadores criar planos personalizados com base em diversos parâmetros, como o número mínimo de calorias a serem consumidas, o número de atividades únicas que pretende realizar e quantas vezes deseja realizar

cada atividade. Esta funcionalidade encontra-se na classe `SportifyController` dentro do modelo MVC adotado.

O método `generateTrainingPlan` é responsável por esta funcionalidade. Este método recebe como entrada a data do início do plano de treino, o plano de treino a ser gerado, o utilizador para quem o plano está a ser criado, a lista de atividades a incluir no plano e o número mínimo de calorias a serem consumidas durante o plano.

Internamente, este método calcula as calorias por atividade com base no número mínimo de calorias a serem consumidas e no número total de atividades. Em seguida, seleciona datas e horários disponíveis para a realização das atividades, levando em consideração a natureza das atividades (se são difíceis ou não) e a disponibilidade de tempo. Durante este processo, as datas de atividades difíceis são registadas para garantir uma distribuição equilibrada das atividades ao longo do tempo.

Depois de determinar a data e horário da atividade, o método cria a instância da atividade selecionada e adiciona-a ao plano de treino e assim sucessivamente para todas as atividades. Além disso, calcula o consumo de calorias total previsto para o plano de treino, com base nas atividades selecionadas e nas características do utilizador.

Em resumo, permite aos utilizadores criar planos personalizados de forma eficiente e flexível, com base em diversos parâmetros e restrições.

```
=====
Generate Training Plan ✎
=====
Please enter the following details to generate a training plan:
When do you plan on executing this plan?
Please enter the date in the following format: yyyy-MM-dd
|> 2024-10-23
How many calories do you want to burn? (minimum)
|> 120
How many unique activities should the plan contain?
|> 2
Please select an activity:
[0] 🏃 Repeating
[1] 🏃 Repeating with weights
[2] 🚶 Distance
[3] 🏃 Distance and altitude
|> 0
Please select an activity:
[0] 🌟 Burpee
[1] 🌟 Jumping jacks
[2] 🌟 Push-ups
[3] 🌟 Sit-ups
[4] 🌟 Stretch
|> 4
How many times should this activity be repeated?
|> 9
Please select an activity:
[0] 🏃 Repeating
[1] 🏃 Repeating with weights
[2] 🚶 Distance
[3] 🏃 Distance and altitude
|> 2
Please select an activity:
[0] 🎩 Ice skating
[1] 🏊 Rowing
[2] 🏃 Track and field
|> 0
How many times should this activity be repeated?
|> 8
Added training plan with 1461.7482352941174 calories consume.
```

Figura 25: Geração de plano de treinos

3.1.5. Salvaguarda do estado da aplicação

Para assegurar a salvaguarda dos dados relevantes da aplicação, foi desenvolvida a classe `FileIO`. Esta possibilita a importação e exportação de objetos em formato binário, viajabilizando a gravação e a recuperação do estado da aplicação em arquivos de dados.

Através do método `importBinary`, é possível ler um objeto previamente serializado a partir de um ficheiro especificado pelo seu nome, retornando-o para ser utilizado na aplicação. Por outro lado, o método `exportBinary` permite a gravação de um objeto num ficheiro, garantindo a preservação do estado atual da aplicação.

A utilização da classe `FileIO` proporciona uma maneira eficiente e confiável de guardar dados relevantes da aplicação *Sportify* num ficheiro binário, possibilitando a sua recuperação em momentos posteriores. Isto é fundamental para garantir a integridade e a continuidade das informações, permitindo a realização de simulações e a restauração dos dados caso seja necessário.

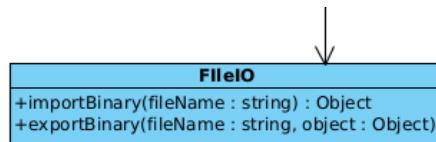


Figura 26: Classe FileIO

3.2. Aspectos Relevantes

3.2.1. Documentação

A documentação do código segue consistentemente o formato de comentários *JavaDoc*, fornecendo informações claras sobre a funcionalidade de cada método criado. Isto permite uma compreensão rápida e fácil do propósito de cada componente do código da aplicação desenvolvida. Esta abordagem facilita a manutenção e colaboração no projeto, permitindo o uso do código de forma eficaz.

3.2.2. Testes unitários

Foram realizados testes unitários para os métodos de todas as classes relacionadas com atividades, planos de treino e utilizadores, excetuando as classes abstratas (ex: `ActivityDistance`).

4. Conclusão

Consideramos que o objetivo principal do trabalho prático foi alcançado com sucesso. O projeto *Sportify* revela uma abordagem bem estruturada e abrangente no desenvolvimento de uma aplicação que envolve gestão de atividades físicas e planos de treinos.

Além disso, a escolha de usar uma arquitetura MVC proporcionou uma organização clara do código, facilitando a evolução do projeto.