

Universidade da Beira Interior

Departamento de Informática



Licenciatura em Engenharia Informática

UC: Programação Orientada a Objetos

*Relatório do Trabalho Prático – Grupo 17*

## A MINHA DIETA

André Schroeder 53949 • Carlos Farinha 53481 • Henrique  
Marques 55129 • João Rodrigues 51779



Covilhã, 29 de dezembro de 2025

## Agradecimentos

A equipa agradece ao corpo docente da Unidade Curricular de *Programação Orientada a Objetos* pela orientação científica e pelo acompanhamento contínuo ao longo do desenvolvimento do projeto. Agradece-se igualmente o contributo de colegas através de *feedback* técnico e validação informal de funcionalidades.

# Resumo

O presente relatório descreve o desenvolvimento do projeto *A Minha Dieta*, uma aplicação *desktop* em Java (JavaFX) destinada ao registo e acompanhamento de alimentação, hidratação e atividade física, integrando cálculo automático de metas e indicadores. A solução foi concebida segundo o padrão **Model–View–Controller (MVC)** (Model–View–Controller), promovendo separação de responsabilidades, manutenibilidade e extensibilidade. Entre as funcionalidades implementadas destacam-se: gestão de perfis, cálculo de **Índice de Massa Corporal (IMC)** e **Taxa Metabólica Basal (TMB)**, diário alimentar com registos de macronutrientes, monitorização de água com progresso face à meta diária, registo de exercício, histórico consultável e exportação de relatórios em **Portable Document Format (PDF)**. O relatório apresenta decisões de desenho e exemplos representativos de implementação (Java, FXML, CSS, persistência e gráficos), bem como uma estratégia de validação por cenários de utilização.

**Palavras-chave:** Programação Orientada a Objetos (POO), Java, JavaFX, MVC, Persistência, Gráficos, PDF.

# Abstract

This report presents *A Minha Dieta*, a Java (JavaFX) desktop application designed to support diet tracking, hydration monitoring, and exercise logging, including automatic computation of personalized goals and indicators. The solution follows the **MVC** architecture, focusing on maintainability and extensibility. Key features include profile management, **IMC** (BMI) and **TMB** (BMR) computation, daily macronutrient logging, hydration progress tracking, exercise monitoring, searchable history, and **PDF** export. We also provide representative code excerpts (Java, FXML, CSS, persistence, and charts) and a validation strategy based on usage scenarios.

**Keywords:** OOP, Java, JavaFX, **MVC**, Persistence, Charts, **PDF**.

## Lista de Acrónimos

<b>POO</b>	Programação Orientada a Objetos
<b>MVC</b>	Model–View–Controller
<b>UI</b>	Interface do Utilizador
<b>FXML</b>	Ficheiro declarativo de interface JavaFX
<b>CSS</b>	Cascading Style Sheets
<b>TMB</b>	Taxa Metabólica Basal
<b>IMC</b>	Índice de Massa Corporal
<b>PDF</b>	Portable Document Format

# Conteúdo

<b>Agradecimentos</b>	<b>i</b>
<b>Resumo</b>	<b>ii</b>
<b>Abstract</b>	<b>iii</b>
<b>Lista de Acrónimos</b>	<b>iv</b>
<b>1 Introdução</b>	<b>1</b>
1.1 Objetivos . . . . .	1
<b>2 Requisitos e Execução</b>	<b>1</b>
2.1 Requisitos . . . . .	1
2.2 Execução (Maven) . . . . .	1
<b>3 Arquitetura do Sistema (MVC)</b>	<b>2</b>
3.1 Visão geral . . . . .	2
<b>4 Modelação do Domínio e Regras de Negócio</b>	<b>2</b>
4.1 Entidades centrais . . . . .	2
4.2 Encapsulamento, invariantes e validação . . . . .	2
<b>5 Dashboard e Visualização de Dados</b>	<b>3</b>
5.1 Indicadores exibidos no Dashboard . . . . .	3
5.2 IMC e classificação . . . . .	3
5.3 FXML do Dashboard (estrutura dos gráficos) . . . . .	4
5.4 Controller: povoamento dos gráficos (lógica central) . . . . .	5
5.5 Agregações no Model (exemplos) . . . . .	6
<b>6 Persistência e Exportação PDF</b>	<b>7</b>
6.1 Persistência por serialização . . . . .	7
6.2 Exportação PDF (OpenPDF/LibrePDF) . . . . .	7

<b>7</b>	<b>Testes e Validação</b>	<b>8</b>
<b>8</b>	<b>Conclusão e Trabalho Futuro</b>	<b>9</b>
8.1	Conclusão . . . . .	9
8.2	Trabalho futuro . . . . .	9
<b>9</b>	<b>Bibliografia e Referências</b>	<b>10</b>

## Listings

1	Compilar e executar o projeto (exemplo) . . . . .	1
2	Organização típica por pacotes (representativa) . . . . .	2
3	Construtor com validação (representativo) . . . . .	2
4	Cópias defensivas (representativo) . . . . .	3
5	dashboard.fxml (excerto representativo): PieChart, BarChart e Peso . . . .	4
6	DashboardController: refresh e update dos gráficos (representativo) . . . .	5
7	Agregação de macros por dia (representativo) . . . . .	6
8	Total de calorias queimadas por dia (representativo) . . . . .	6
9	Guardar e carregar estado (representativo) . . . . .	7
10	Exportação PDF (excerto representativo) . . . . .	8

# 1 Introdução

A aplicação *A Minha Dieta* foi desenvolvida no âmbito da Unidade Curricular de [POO](#) com o objetivo de consolidar conceitos fundamentais de modelação orientada a objetos e boas práticas de desenho de software. O domínio do problema envolve registo de hábitos (alimentação, água, exercício e peso), cálculo de metas/indicadores e visualização de progresso através de um *dashboard* interativo.

## 1.1 Objetivos

- Modelar o domínio com classes coesas e responsabilidades bem definidas;
- Implementar uma interface JavaFX consistente e intuitiva;
- Calcular metas e indicadores (IMC, TMB, metas de água e macronutrientes);
- Agregar registos diários e históricos, suportando visualização por gráficos;
- Persistir o estado da aplicação e exportar relatórios em PDF.

# 2 Requisitos e Execução

## 2.1 Requisitos

- Java (JDK 17 ou superior);
- Maven 3.6+;
- Dependências JavaFX geridas via Maven.

## 2.2 Execução (Maven)

```
1 git clone https://github.com/MooniePT/AminhaDieta.git
2 cd AminhaDieta
3 mvn clean compile
4 mvn javafx:run
```

Listing 1: Compilar e executar o projeto (exemplo)

## 3 Arquitetura do Sistema (MVC)

### 3.1 Visão geral

A solução adota o padrão MVC, separando o domínio (*Model*) da interface (*View*) e do controlo de fluxo (*Controller*). Esta decisão reduz acoplamento, melhora testabilidade do domínio e facilita manutenção incremental.

```
1 src/main/java/  
2   app/  
3     model/          (UserProfile, AppState, MealEntry, WaterEntry,  
ExerciseEntry, ...)  
4     controller/     (DashboardController, MealsController,  
HydrationController, ...)  
5     service/        (DataStore, PdfExporter, helpers)  
6 src/main/resources/  
7   fxml/             (dashboard.fxml, meals.fxml, ...)  
8   css/              (theme.css)
```

Listing 2: Organização típica por pacotes (representativa)

## 4 Modelação do Domínio e Regras de Negócio

### 4.1 Entidades centrais

O domínio é centrado no perfil do utilizador, que agrega registos e fornece operações de cálculo e agregação (metas, totais diários e histórico).

### 4.2 Encapsulamento, invariantes e validação

A consistência é assegurada por validações (entradas inválidas são rejeitadas) e por encapsulamento do estado. Sempre que são expostas coleções internas, devolvem-se cópias defensivas, evitando fuga de representação.

```
1 public UserProfile(String name, double weightKg, int heightCm, int age)  
2 {  
3     if (name == null || name.isBlank()) throw new  
IllegalArgumentException("Nome inv lido.");  
4     if (weightKg <= 0) throw new IllegalArgumentException("Peso  
inv lido.");  
5     if (heightCm <= 0) throw new IllegalArgumentException("Altura  
inv lida.");
```

```

5      if (age <= 0) throw new IllegalArgumentException("Idade inv lida.")
        ;
6      this.name = name.trim();
7      this.weightKg = weightKg;
8      this.heightCm = heightCm;
9      this.age = age;
10 }

```

Listing 3: Construtor com validação (representativo)

```

1 public List<MealEntry> getMeals() {
2     return new ArrayList<>(meals);
3 }

```

Listing 4: Cópias defensivas (representativo)

## 5 Dashboard e Visualização de Dados

### 5.1 Indicadores exibidos no Dashboard

O *dashboard* apresenta indicadores de acompanhamento diário:

- **Calorias do dia:** total consumido vs meta diária (barra de progresso);
- **Água do dia:** volume ingerido vs meta diária (barra de progresso);
- **IMC:** valor atual e classificação qualitativa;
- **Macronutrientes:** proteína, hidratos e gordura (consumo vs objetivo);
- **Gráficos:** *PieChart* (consumo diário), *BarChart* (atividade física) e gráfico de **peso** por data.

### 5.2 IMC e classificação

O IMC é calculado por:

$$IMC = \frac{peso(kg)}{altura(m)^2}$$

A Tabela 1 resume uma classificação típica utilizada em contexto académico.

Intervalo (IMC)	Classificação
$< 18.5$	Baixo peso
$[18.5, 24.9]$	Saudável
$[25.0, 29.9]$	Excesso de peso
$\geq 30.0$	Obesidade

Tabela 1: Classificação típica do IMC.

### 5.3 FXML do Dashboard (estrutura dos gráficos)

O excerto seguinte ilustra a declaração dos três gráficos principais e alguns nós associados (IDs usados pelo *Controller*).

```

1 <AnchorPane xmlns="http://javafx.com/javafx"
2           xmlns:fx="http://javafx.com/fxml"
3           fx:controller="app.controller.DashboardController">
4
5   <PieChart fx:id="dailyPie" layoutX="40" layoutY="430" prefWidth="480"
6     prefHeight="240"/>
7
8   <BarChart fx:id="activityBar" layoutX="560" layoutY="430" prefWidth="
9     520" prefHeight="240">
10     <xAxis>
11       <CategoryAxis label="Dia"/>
12     </xAxis>
13     <yAxis>
14       <NumberAxis label="Kcal"/>
15     </yAxis>
16   </BarChart>
17
18   <ScatterChart fx:id="weightChart" layoutX="1100" layoutY="430"
19     prefWidth="360" prefHeight="240">
20     <xAxis>
21       <CategoryAxis label="Data"/>
22     </xAxis>
23     <yAxis>
24       <NumberAxis label="Kg"/>
25     </yAxis>
26   </ScatterChart>
27 </AnchorPane>

```

Listing 5: dashboard.fxml (excerto representativo): PieChart, BarChart e Peso

## 5.4 Controller: povoamento dos gráficos (lógica central)

O *Controller* agrega dados do *Model* e atualiza os gráficos. O ciclo típico é: evento do utilizador → atualização do modelo → `refresh()` do dashboard.

```
1 public class DashboardController {
2
3     @FXML private PieChart dailyPie;
4     @FXML private BarChart<String, Number> activityBar;
5     @FXML private ScatterChart<String, Number> weightChart;
6
7     private AppState state;
8
9     public void setState(AppState state) {
10         this.state = state;
11         refresh();
12     }
13
14     private void refresh() {
15         UserProfile p = state.getActiveProfile();
16         LocalDate today = LocalDate.now();
17
18         updateDailyPie(p, today);
19         updateActivityBar(p, today);
20         updateWeightChart(p);
21     }
22
23     private void updateDailyPie(UserProfile p, LocalDate day) {
24         MacroTotals t = p.totalsFor(day);
25
26         dailyPie.getData().setAll(
27             new PieChart.Data(" gua (" + p.totalWaterTodayMl() + "ml)",
28 p.totalWaterTodayMl()),
29             new PieChart.Data("Prote na (" + (int)t.getProteinG() + "g)",
30 t.getProteinG()),
31             new PieChart.Data("Hidratos (" + (int)t.getCarbsG() + "g)",
32 t.getCarbsG()),
33             new PieChart.Data("Gordura (" + (int)t.getFatG() + "g)", t.
34 getFatG())
35         );
36     }
37
38     private void updateActivityBar(UserProfile p, LocalDate today) {
39         XYChart.Series<String, Number> s = new XYChart.Series<>();
40         s.setName("Kcal queimadas");
41
42         for (int i = 6; i >= 0; i--) {
```

```

39         LocalDate d = today.minusDays(i);
40         int burned = p.totalBurnedCaloriesOn(d);
41         s.getData().add(new XYChart.Data<>(d.getDayOfMonth() + "/" +
d.getMonthValue(), burned));
42     }
43
44     activityBar.getData().setAll(s);
45 }
46
47 private void updateWeightChart(UserProfile p) {
48     XYChart.Series<String, Number> s = new XYChart.Series<>();
49     s.setName("Peso");
50
51     for (WeightEntry w : p.getWeightHistory()) {
52         String label = w.getDate().getDayOfMonth() + "/" + w.getDate
().getMonthValue();
53         s.getData().add(new XYChart.Data<>(label, w.getWeightKg()));
54     }
55
56     weightChart.getData().setAll(s);
57 }
58 }

```

Listing 6: DashboardController: refresh e update dos gráficos (representativo)

## 5.5 Agregações no Model (exemplos)

Para suportar os gráficos, o *Model* disponibiliza métodos de agregação diária.

```

1 public MacroTotals totalsFor(LocalDate day) {
2     int kcal = 0;
3     double p = 0, c = 0, f = 0;
4
5     for (MealEntry m : meals) {
6         if (m.getWhen().toLocalDate().equals(day)) {
7             kcal += m.getCalories();
8             p += m.getProteinG();
9             c += m.getCarbsG();
10            f += m.getFatG();
11        }
12    }
13    return new MacroTotals(kcal, p, c, f);
14 }

```

Listing 7: Agregação de macros por dia (representativo)

```

1 public int totalBurnedCaloriesOn(LocalDate day) {

```

```

2      int total = 0;
3      for (ExerciseEntry e : exercises) {
4          if (e.getWhen().toLocalDate().equals(day)) {
5              total += e.getCaloriesBurned();
6          }
7      }
8      return total;
9  }

```

Listing 8: Total de calorias queimadas por dia (representativo)

## 6 Persistência e Exportação PDF

### 6.1 Persistência por serialização

A persistência é assegurada por serialização do estado global da aplicação, permitindo restaurar perfis e histórico entre execuções.

```

1  public final class DataStore {
2      private static final String FILE = "data/appstate.dat";
3
4      public static void save(AppState state) throws IOException {
5          Files.createDirectories(Paths.get("data"));
6          try (ObjectOutputStream out =
7              new ObjectOutputStream(new FileOutputStream(FILE))) {
8              out.writeObject(state);
9          }
10     }
11
12     public static AppState load() throws IOException,
13     ClassNotFoundException {
14         File f = new File(FILE);
15         if (!f.exists()) return new AppState();
16         try (ObjectInputStream in =
17             new ObjectInputStream(new FileInputStream(FILE))) {
18             return (AppState) in.readObject();
19         }
20     }

```

Listing 9: Guardar e carregar estado (representativo)

### 6.2 Exportação PDF (OpenPDF/LibrePDF)

```

1 public final class PdfExporter {
2     public static void exportDaily(UserProfile p, LocalDate day, String
    outFile) throws Exception {
3         Document doc = new Document();
4         PdfWriter.getInstance(doc, new FileOutputStream(outFile));
5         doc.open();
6
7         Font title = new Font(Font.HELVETICA, 16, Font.BOLD);
8         doc.add(new Paragraph("Relat rio Di rio      A Minha Dieta",
    title));
9         doc.add(new Paragraph("Utilizador: " + p.getName()));
10        doc.add(new Paragraph("Data: " + day));
11        doc.add(new Paragraph(" "));
12
13        PdfPTable table = new PdfPTable(4);
14        table.addCell("Descr i o");
15        table.addCell("Kcal");
16        table.addCell("Prot (g)");
17        table.addCell("HC (g)");
18
19        for (MealEntry m : p.mealsOn(day)) {
20            table.addCell(m.getDescription());
21            table.addCell(String.valueOf(m.getCalories()));
22            table.addCell(String.valueOf(m.getProteinG()));
23            table.addCell(String.valueOf(m.getCarbsG()));
24        }
25        doc.add(table);
26        doc.close();
27    }
28 }

```

Listing 10: Exportação PDF (excerto representativo)

## 7 Testes e Validação

A validação foi conduzida por cenários de uso (testes manuais) e verificação de invariantes no modelo. A independência do *Model* relativamente à UI sugere, como extensão natural, testes unitários ao nível de cálculos e agregações.

CT	Descrição e resultado esperado
CT01	Criar perfil válido → metas calculadas e persistência após reinício.
CT02	Registrar refeição → totais e macros diários atualizados + PieChart coerente.
CT03	Registrar água → barra de progresso e pie slice de água atualizados.
CT04	Registrar exercício → BarChart semanal atualizado.
CT05	Registrar peso → gráfico de peso atualizado e histórico consistente.
CT06	Exportar relatório → PDF gerado com dados do dia selecionado.

Tabela 2: Casos de teste representativos (testes manuais).

## 8 Conclusão e Trabalho Futuro

### 8.1 Conclusão

O projeto *A Minha Dieta* permitiu consolidar conceitos fundamentais de [POO](#) num cenário realista, integrando arquitetura [MVC](#), modelação de domínio, validação, persistência e visualização de dados através de gráficos no *dashboard*. A separação entre domínio e interface promove manutenibilidade e suporta evolução incremental do sistema.

### 8.2 Trabalho futuro

- Persistência baseada em SQLite para maior robustez e consultas avançadas;
- Integração com APIs externas para enriquecer a base de alimentos;
- Testes unitários automatizados no *Model* e testes de UI;
- Relatórios analíticos (tendências semanais/mensais) com indicadores comparativos.

## 9 Bibliografia e Referências

### Referências

- [1] Java Documentation. <https://docs.oracle.com/en/java/>
- [2] OpenJFX (JavaFX). <https://openjfx.io/>
- [3] OpenPDF (LibrePDF). <https://github.com/LibrePDF/OpenPDF>
- [4] Basal Metabolic Rate (contexto TMB/BMR). [https://en.wikipedia.org/wiki/Basal\\_metabolic\\_rate](https://en.wikipedia.org/wiki/Basal_metabolic_rate)