

A Minha Dieta

Plano de Requisitos, Estrutura e GUI (Java + JavaFX)

Grupo: _____

7 de dezembro de 2025

Objetivo do Projeto

Desenvolver uma aplicação em **Java** para registo e acompanhamento alimentar, permitindo:

- Registrar **refeições, alimentos, quantidades e valor nutricional** (kcal, proteínas, gorduras, hidratos, etc.).
- Calcular totais de calorias e macros **por refeição, por dia e por semana**.
- Definir **objetivos** (manter/perder/ganhar), compor dietas e acompanhar **consumo diário de água**.
- Registrar **peso** e calcular **IMC** ao longo do tempo.

Requisitos Obrigatórios (Enunciado)

- **Persistência de dados** usando **File** e **ObjectStreams** (**ObjectInputStream/ObjectOutputStream**).
- Representação de datas usando **LocalDate**.
- Desafio: **Interface Gráfica** com **JavaFX + Scene Builder**.

Requisitos Funcionais (Implementação Proposta)

1) Registo de Utilizador (obrigatório antes de tudo)

A aplicação deve iniciar com um ecrã de registo/perfil. Sem perfil criado, o resto do menu fica bloqueado.

- Dados mínimos: nome/username, **idade, peso, altura**.
- Recomendado (melhora cálculos e qualidade do projeto): **sexo e nível de atividade**.
- Ao concluir registo: mostrar automaticamente **IMC, necessidade calórica diária e meta de água**.

2) Gestão de Alimentos

- Criar/editar/remover alimentos.
- Valores nutricionais preferencialmente **por 100g**: kcal, proteína, gordura, hidratos.
- Pesquisa e/ou favoritos (bónus recomendado).

3) Registo de Refeições

- Refeição associada a **LocalDate** e a um tipo (pequeno-almoço/almoço/jantar/lanche).
- Cada refeição contém itens: alimento + quantidade (g/ml/unidades).
- Cálculo automático do total nutricional da refeição (kcal/macros).

4) Totais Diários e Semanais

- Totais do dia: kcal/macros + água ingerida.
- Totais e médias semanais: kcal/macros, consumo de água, variação de peso (se existir).

5) Objetivos / Dieta

- Definir objetivo: **manter**, **perder** ou **ganhar**.
- Definir meta calórica e (opcional) metas de macros.
- Permitir ajustar o défice/superávit (slider simples).

6) Registo de Água

- Registar água ingerida por dia e mostrar progresso em relação à meta.
- Permitir que o utilizador ajuste manualmente a meta (importante, pois é uma estimativa).

7) Registo de Peso e Evolução

- Registo de peso por data.
- Evolução do IMC e do peso ao longo do tempo (gráfico opcional).

Cálculos e Fórmulas (para a aplicação e relatório)

IMC

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

A aplicação deve apresentar a classificação do IMC (ex.: magreza/normal/excesso/obesidade).

Necessidade Calórica Diária (Recomendado: Mifflin–St Jeor)

BMR (metabolismo basal):

- Homem: $BMR = 10w + 6.25h - 5a + 5$
- Mulher: $BMR = 10w + 6.25h - 5a - 161$

onde w = peso (kg), h = altura (cm), a = idade (anos).

TDEE (manutenção):

$$TDEE = BMR \times fatorAtividade$$

Fatores típicos:

- Sedentário 1.2; Ligeiro 1.375; Moderado 1.55; Alto 1.725; Muito alto 1.9

Objetivo:

- Perder: $Meta = TDEE - (300a500)$

- Ganhar: $Meta = TDEE + (200a400)$
- Manter: $Meta = TDEE$

Água Diária (Estimativa simples e editável)

$$\text{Água}(L/dia) \approx 0.035 \times \text{peso}(kg)$$

A meta deve ser ajustável pelo utilizador e identificada como **estimativa**.

GUI (JavaFX) — Proposta Visual e Fluxo

Fluxo obrigatório (para cumprir o enunciado e o nosso design)

1. **Ecrã de Registo/Perfil** (primeiro ecrã)
2. Após perfil criado: acesso ao **Dashboard** e ao **Menu**

Menu (todas as funcionalidades do enunciado organizadas)

Recomendação: Layout em **BorderPane**, com menu lateral (VBox) e área central dinâmica.

- Dashboard
- Perfil
- Alimentos
- Refeições
- Água
- Peso & IMC
- Objetivos / Dieta
- Estatísticas (Dia/Semana)

Estilo (CSS para ficar “premium”)

- Cards com sombra leve, cantos arredondados.
- Cores por estado (ex.: dentro da meta / acima da meta).
- Tipografia consistente e espaçamento generoso.

Estrutura Recomendada do Projeto (Camadas)

Para evitar código confuso, recomenda-se separação em camadas:

Pastas

```
AMinhaDieta/
src/main/java/
  app/Main.java
  app/model/...
  app/service/...
  app/persistence/...
  app/ui/controller/...
```

```
src/main/resources/  
  fxml/  
  css/  
  assets/
```

Modelos (principais classes)

Nota: todos devem implementar `Serializable` para persistência.

- **UserProfile**: nome, idade, peso, altura, sexo, atividade, objetivo, metas.
- **Food**: nome e nutrientes por 100g (kcal, proteína, gordura, hidratos).
- **MealEntry**: data (LocalDate), tipo, lista de itens.
- **MealItem**: alimento + quantidade.
- **WaterEntry**: data + ml ingeridos.
- **WeightEntry**: data + peso.
- **AppState**: agregador com todas as listas (estado global para gravar).

Serviços (lógica e cálculos)

- **HealthService**: IMC, BMR, TDEE, metas.
- **NutritionService**: somatórios por refeição/dia/semana.

Persistência (requisito do enunciado)

- **DataStore**: `load()` e `save(AppState)` com `ObjectInputStream/ObjectOutputStream`.
- Recomendação: guardar um único ficheiro (ex.: `data/appstate.dat`) com todo o estado.

Validações e Boas Práticas (importante para a nota)

- Validar entradas: idade > 0; peso > 0; altura > 0; quantidades > 0.
- Não permitir registos futuros em peso/água/refeições (opcional mas recomendado).
- Mensagens de erro/ajuda na própria janela (sem popups excessivos).
- Tolerância a ficheiro corrompido: se falhar leitura, criar estado novo sem crash.

Extras recomendados (bónus e fácil de justificar)

- Exportação para CSV (alimentos, refeições do dia/semana).
- Pesquisa rápida de alimentos + favoritos.
- Gráficos simples (peso, calorias semanais) caso haja tempo.

Estrutura Sugerida do Relatório (até 15 páginas)

1. Introdução e objetivos
2. Requisitos implementados (tabela requisito → funcionalidade/ecrã)
3. Arquitetura (camadas) e justificação

4. Modelo de dados (classes e responsabilidades)
5. Persistência (AppState + DataStore)
6. Interface gráfica (prints e fluxo)
7. Cálculos (IMC, calorias, água) e validações
8. Exemplos de utilização / testes
9. Conclusão e melhorias futuras

Nota importante: os valores de calorias e água são **estimativas** e devem ser ajustáveis pelo utilizador. Esta decisão é justificada por variações individuais e aumenta a robustez da aplicação.