IMCApp

Arquitetura e Funcionalidades

Objetivo

Aplicativo de saúde para calcular IMC e outros indicadores, com histórico de medições.

Tecnologias

- **UI:** 100% Jetpack Compose para interface moderna e reativa
- **Arquitetura:** MVVM (Model-View-ViewModel)
- **Persistência:** Banco de dados local com Room
- **Assincronicidade:** Coroutines para operações em segundo plano

Arquitetura MVVM

- **View (Composables):** Camada de apresentação, reage a mudanças de estado
- **ViewModel:** Contém lógica de UI e gerencia o estado
- **Model (Repository/Domain):** Abstrai fontes de dados e lógica de negócio

Funcionalidades Principais

- Cálculo de IMC, TMB, Peso Ideal e Necessidade Calórica
- Histórico de medições com detalhes
- Validação de entrada de dados

Destaques da Implementação

Domain Layer

// C:/.../domain/HealthCalculator.kt

```
object HealthCalculator {
  fun calculateBMR(
    weight: Double, height: Double,
    age: Int, isMale: Boolean
  ): Double {
    return if (isMale) {
      (10*weight)+(6.25*height)-(5*age)+5
    } else {
      (10*weight)+(6.25*height)-(5*age)-161
    }
  }
}
```

✓ Lógica isolada e testável

✓ Facilitando testes unitários

Data Layer

// C:/.../data/model/BmiData.kt

```
@Entity(tableName = "bmi_records")
data class BmiData(
  @PrimaryKey(autoGenerate=true) val id: Long,
  val weight: Double, val height: Double,
  val bmi: Double, val classification: String,
  val timestamp: Long = System.currentTimeMillis(),
  val bmr: Double = 0.0, val idealWeight: Double = 0.0
)
```

✓ Persistência segura

✓ Histórico de medições

> **Separação:** Cálculos centralizados no HealthCalculator, garantindo isolamento e testabilidade.

Dificuldades e o Uso de LLMs

Dificuldades Encontradas

1. Gerenciamento de Estado

Complexidade inicial com StateFlow e collectAsState no Compose

2. Operações Assíncronas

Garantir que chamadas ao banco não travassem a UI thread

3. Geração de Código

Implementar fórmulas complexas (BMR, Peso Ideal, Calórica)

Ferramentas

Gemini (Android Studio)

ChatGPT

Prompts Efetivos

STATEFLOW + LAZYCOLUMN

"Como exponho lista do Room como StateFlow no ViewModel e consumo em Composable com LazyColumn?"

ASYNC/ROOM ERROR

"Erro: Cannot access database on main thread. Como usar viewModelScope com coroutines?"

GERAÇÃO DE CÓDIGO

"Crie função em Kotlin que calcula peso ideal pela fórmula de Robinson."

Resultado: Redução de 30-40% no tempo de desenvolvimento

Opinião sobre o Uso de LLMs

> **Opinião do Grupo:** LLMs são assistentes poderosos. Não substituem o conhecimento, mas aumentam drasticamente a produtividade.

Pontos Positivos

- **Velocidade:** Reduz boilerplate repetitivo e acelera criação
- **Aprendizado:** Exemplos práticos rápidos para conceitos novos
- **Documentação:** Gera README.md e outras docs eficazmente

⚠ Pontos de Atenção

- **Supervisão essencial:** Revisar sempre código gerado
- **"Alucinações":** APIs fictícias que não existem
- **Práticas antigas:** Validar padrões e melhores práticas

> ⚡ **Como usar bem:** Revisar sempre. A IA pode usar práticas antigas. É crucial validar e adaptar para seu projeto.

Conclusão Final

LLMs = "Copiloto Experiente"

Você dirige (arquitetura e lógica), ele auxilia com tarefas secundárias e ajuda com bugs. Use LLMs como ferramenta complementar ao seu conhecimento técnico.