🚀 Flutter Offline-First na Prática: O Guia Completo com POO, Patterns e Arquitetura

🩺 O Problema Real

Imagine o Dr. Carlos, médico de plantão em um hospital no interior.
Ele precisa emitir uma **nota fiscal** no app da clínica, mas a internet está oscilando.
Se o app falhar, ele perde tempo precioso — e pode até atrasar o repasse de seus honorários.

👉 É aqui que entra a **arquitetura offline-first com Flutter**, garantindo que o app funcione mesmo em condições ruins de conectividade.

1. Programação Orientada a Objetos (POO) aplicada no Flutter

Conceito

POO ajuda a organizar o código em **objetos com responsabilidades claras**.

Exemplo

```dart
class User {
  final String id;
  final String name;

  User(this.id, this.name);
}
```

Analogia

📌 Pense em uma **carteira de identidade**: ela encapsula (esconde) toda a burocracia por trás, mas te entrega apenas o que importa — seu nome, sua foto e seu número.

2. Modelagem de Entidades

Exemplo

```
class DraftInvoice {
  final String id;
  final double amount;
  bool synced;

  DraftInvoice(this.id, this.amount, {this.synced = false});
}
```

Analogia

📌 Um **rascunho de cheque**: você escreve o valor, mas só quando for validado no banco (API) ele se torna oficial.

3. Encapsulamento

Exemplo

```
class LocalQueueService {
  final List<DraftInvoice> _queue = [];

  void add(DraftInvoice invoice) => _queue.add(invoice);

  List<DraftInvoice> get pending => List.unmodifiable(_queue);
}
```

Analogia

📌 Uma **caixa fechada**: você pode colocar papéis (notas), mas só acessa de forma controlada, nunca direto no fundo da caixa.

4. Polimorfismo

Exemplo

```
abstract class SyncStrategy {
  Future<void> sync(DraftInvoice invoice);
}

class AutoSync implements SyncStrategy {
  @override
  Future<void> sync(DraftInvoice invoice) async {
    // sincroniza automaticamente
  }
}

class ManualSync implements SyncStrategy {
  @override
  Future<void> sync(DraftInvoice invoice) async {
    // depende de ação do usuário
  }
}
```

5. Design Patterns

Repository Pattern

```
class InvoiceRepository {
  final LocalQueueService local;
  final ApiService remote;

  InvoiceRepository(this.local, this.remote);
}
```

📌 **Biblioteca**: você pede um livro (nota), e o bibliotecário decide se pega da prateleira local ou encomenda de fora.

Strategy Pattern

Mostrado acima com `SyncStrategy`.

📌 **Motor de carro intercambiável**: o mesmo chassi aceita motor elétrico ou a combustão.

Observer (State Management)

```
class InvoiceNotifier extends ChangeNotifier {
  List<DraftInvoice> invoices = [];

  void add(DraftInvoice invoice) {
    invoices.add(invoice);
    notifyListeners();
  }
}
```

📌 **Grupo do WhatsApp**: quando alguém manda mensagem (nota nova), todos recebem automaticamente.

6. Arquitetura Offline-First

- **Salvar primeiro localmente**
- **Sincronizar com backoff exponencial**
- **Marcar stale (desatualizado)** até confirmação

📌 **Exemplo real**: Como mandar mensagem no WhatsApp sem internet → aparece o relógio ⏳ e só vira check ✓ quando sincroniza.

7. Comparação Antes vs Depois

Antes (sem patterns)

```
// Lógica da API dentro da UI
onPressed: () async {
  final response = await ApiService().sendInvoice(invoice);
  if (response.success) {
    // atualizar UI
  }
}
```

Depois (com patterns)

```
onPressed: () async {
  await context.read<InvoiceRepository>().sync(invoice);
}
```

👉 Resultado: UI mais limpa, testável e resiliente.

8. Erros Comuns

- ❌ Salvar só local e esquecer de sincronizar
- ❌ Não marcar dados como stale → UI engana o usuário
- ❌ Retry sem backoff → flooda a API

9. Cheat Sheet (Resumo Final)

Conceito	Exemplo Flutter	Analogia
POO	`class User`	Carteira de identidade
Encapsulamento	`LocalQueueService`	Caixa fechada
Polimorfismo	`SyncStrategy`	Câmbio manual vs automático
Repository	`InvoiceRepository`	Bibliotecário
Strategy	`AutoSync/ManualSync`	Motor intercambiável
Observer	`ChangeNotifier`	Grupo do WhatsApp
Offline-First	Fila local + backoff	WhatsApp offline

10. Perguntas de Entrevista (com respostas curtas)

- **Q:** Como trataria falta de internet?
  A: Offline-first → salva local, retry com backoff, feedback na UI.
- **Q:** Como separar lógica de persistência da UI?
  A: Repository Pattern.
- **Q:** Como evitar flooding de API em conexões ruins?
  A: Retry com backoff exponencial.
- **Q:** Como notificar a UI sobre mudanças automáticas?
  A: Observer Pattern (ChangeNotifier, Bloc, Riverpod).

💡 Sugestão de Pitch na entrevista:

> “Se o médico perder a internet, meu app continua funcionando.
> Ele salva a nota localmente, sincroniza depois e avisa na UI o que está atualizado ou não.
> Isso reduz fricção e aumenta confiança no produto.”