09

# Conclusão

**Transcrição**

**Parabéns!** Chegamos ao final do curso de **Persistência de Dados Internos em Flutter**.

Ao logo das aulas, aprendemos diversas técnicas comuns com passos a mais em relação aos fundamentos de Flutter para manter informações da usuária ou usuário mesmo fechando e reiniciando o aplicativo e realizar a integração envolvendo **comportamentos assíncronos**, entre outras tarefas.

No início do curso, trabalhamos no aplicativo oficial do **Bytebank** com um novo fluxo baseado tanto no Dashboard quanto na lista de contatos até o formulário para adicionar novos itens.

Vimos algumas peculiaridades ainda desconhecidas, como o caso de **enviar uma imagem** do logotipo para a tela, que possui diversas maneiras de fazer, seja via rede, arquivos do dispositivo e assets.

Em seguida, quando estávamos implementando o fluxo, vimos a necessidade de **modificar o layout** adicionando separação entre os elementos, margens, ajuste de dimensões, cores, fontes e tamanho de texto. O próprio `Column()` dá suporte para estes ajustes através da configuração de alinhamento de seus eixos principais e transversais para atender à proposta.

Aprendemos a implementar a lista e o formulário com uma **nova integração**, complementando o que foi visto em cursos anteriores. Não apenas cria e apresenta conteúdo em memória, mas também armazenamos essas informações por meio do plugin **sqflite**, o qual utiliza a solução do banco de dados **SQLite** por debaixo dos panos.

Em Flutter, vimos que há diversas opções para fazer **persistência de dados**, seja pelo banco, seja por arquivos ou pela chave-valor ou *shared preferences*.

Para dados estruturados que fazem uso de `querys` e conseguem fazer `Futures` e etc., o sqflite é bastante benéfico.

Durante a configuração, tivemos diversos desafios; aprendemos como colocar dependências no projeto em `pubspec.yaml` para trabalharmos com os caminhos de diretórios de cada um dos ambientes, seja **Android** ou **IOs**.

Também vimos como fazer a própria configuração por meio de `app_database.dart` com suas peculiaridades, implementamos `Future` e `.then()` e aprendemos a técnica de `async` `await`.

Por fim, dividimos responsabilidades e o grande arquivo único em `contact_dao.dart` com o padrão comum de projeto **DAO**.

Das novidades de implementação visual, usamos a técnica do `FutureBuilder` e suas particularidades para atualizarmos conteúdo dentro dos `Widgets` de forma a não precisar transformar o principal que representa a tela em um `StatefulWidget`. Desta forma, temos um novo meio de buscar dados e atualizar a tela.

Esperamos que todo esse conteúdo tenha sido de grande utilidade! Estamos sempre prontos para receber dúvidas, críticas e sugestões para melhorar nosso trabalho.

**Até a próxima!**