Flutter Avançado - Aula 3

Exercícios 7 a 9

Fernando Duwe

v 1.0

Dicas de hoje

- 10 melhores extensões para VS Code e Flutter Ver mais

- Tarefas em background são tarefas que são executadas mesmo quando a aplicação estiver fechada
- Com essas tarefas podemos baixar informações e armazená-las em cache, enviar notificações, etc
- Essa rotina funciona para iOS e Android somente
- Para isso, utilizaremos a biblioteca background_fetch

1º Vamos importar a biblioteca necessária no nosso pubspec.yaml e executar o pub get

```
background_fetch
```

2º Vamos configurar o projeto nativo: Como estamos trabalhando diretamente com o SO, é necessário efetuar algumas configurações no nosso projeto.

No site da biblioteca, no pub.dev, existe um passo a passo para cada um dos sistemas operacionais suportados: O iOS e o Android.

Ver mais

3º Vamos agora registrar a *headless task*

A headless task é a tarefa que será executada quando o nosso app for finalizado

```
void main() {
  runApp(const MyApp());

  // Quando a aplicação for finalizada, registramos a tarefa de background
  BackgroundFetch.registerHeadlessTask(backgroundFetchHeadlessTask);
}
```

Tarefas em background - cont

Ver mais

```
// Somente para Android: A Headless Task é executada quando o app é terminado, quando o enableHeadless = True
// É necessário adicionar a anotação no método para evitar problemas em release, em versões Flutter >= 3.3.0
@pragma('vm:entry-point')
void backgroundFetchHeadlessTask(HeadlessTask task) async {
  String vrTaskID = task.taskId;
  bool vrIsTimeOut = task.timeout;

  if (vrIsTimeOut) {
    // O timeout passou, devemos parar o processamento e encerrar a tarefa.
    print("[background_fetch][${vrTaskID}]: Timeout");
    BackgroundFetch.finish(vrTaskID);
    return;
  }

  print("[background_fetch][${vrTaskID}]: Executando");

  // Aqui nós executamos a nossa regra
  LogRepository vrRepository = new LogRepository();

  vrRepository.addLogAsBackgroundTask();

  BackgroundFetch.finish(vrTaskID);
}
```

4º No widget que desejarmos, vamos agora implementar a função que será executada enquanto a aplicação estiver aberta.

Implementaremos também a configuração do background fetch.

Como precisaremos utilizar o initState, vamos criar um widget stateful.

5º No nosso widget, vamos sobrescrever o método initState e dentro dele adicionaremos um novo método.

```
@override
void initState() {
  super.initState();

  initPlatformState();
}
```

Tarefas em background - cont

Ver mais

```
// Mensagens da plataforma são assíncronas
Future<void> initPlatformState() async {
  // Configurando o BackgroundFetch
  int vrStatus = await BackgroundFetch.configure(
      BackgroundFetchConfig(
          minimumFetchInterval: 15, // Deve ser maior ou igual a 15 minutos
          stopOnTerminate: false,
          enableHeadless: true,
          requiresBatteryNotLow: false,
          requiresCharging: false,
          requiresStorageNotLow: false,
          requiresDeviceIdle: false,
          requiredNetworkType: NetworkType.NONE),
      this.Internal_OnFetch,
      this.Internal_OnTimeOut);

  print('[BackgroundFetch] configure success: $vrStatus');

  setState(() {
    _status = vrStatus;
  });

  // Se o widget foi removido da árvore enquanto a mensagem para a plataforma estava
  // sendo enviada, nós deveremos descartar o retorno, para não chamar setState em
  // um widget não existente
  if (!mounted) return;
}
```

Função Timeout

Essa função será chamada sempre que o tempo máximo de execução for atingido. Nesse caso, devemos parar todo o processamento e cancelar o que estamos fazendo.

```
// Evento chamado quando ocorrer um timeout. Devemos parar imediatamente o que estamos fazendo
// e finalizar.
  Future<void> Internal_OnTimeOut(String prTaskID) async {
    print("[BackgroundFetch][${prTaskID}]: Task timeout");

    BackgroundFetch.finish(prTaskID);
  }
```

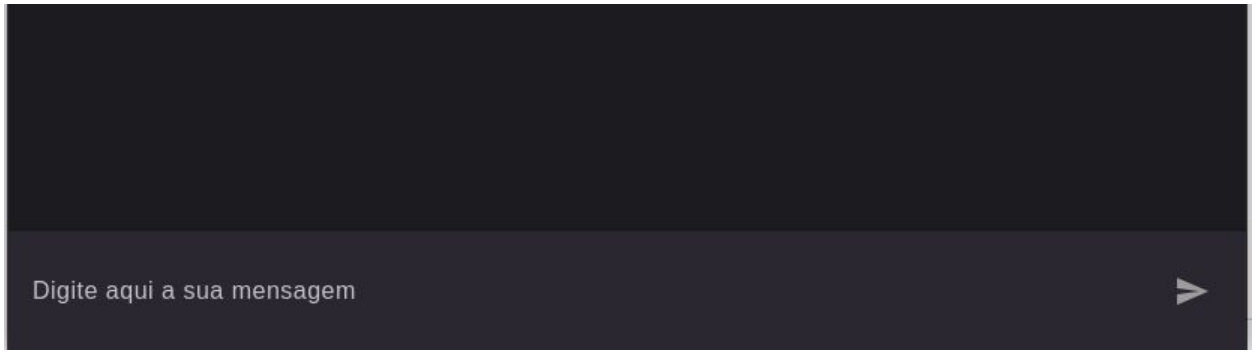
Função Fetch

Essa será a função chamada a cada intervalo, contendo a nossa regra de negócio (enquanto o widget estiver tem tela).

Ver mais

```
// Evento chamado enquanto a aplicação está aberta
Future<void> Internal_OnFetch(String prTaskID) async {
  print("[BackgroundFetch][${prTaskID}]: Event received");

  LogRepository vrRepository = new LogRepository();

  vrRepository.addLogAsAppOpened();

  setState(() {});

  // Importante: Você deve sinalizar que a tarefa foi completada, para o SO
  // não puna sua aplicação por estar demorando demais
  BackgroundFetch.finish(prTaskID);
}
```

- No nosso chat, vamos tratar agora do envio da mensagem
- Na tela de conversa, crie um novo widget que fará o envio da mensagem
- Uma dica, você pode utilizar bottomNavigationBar, do Scaffold para posicionar sua caixa de mensagem no rodapé

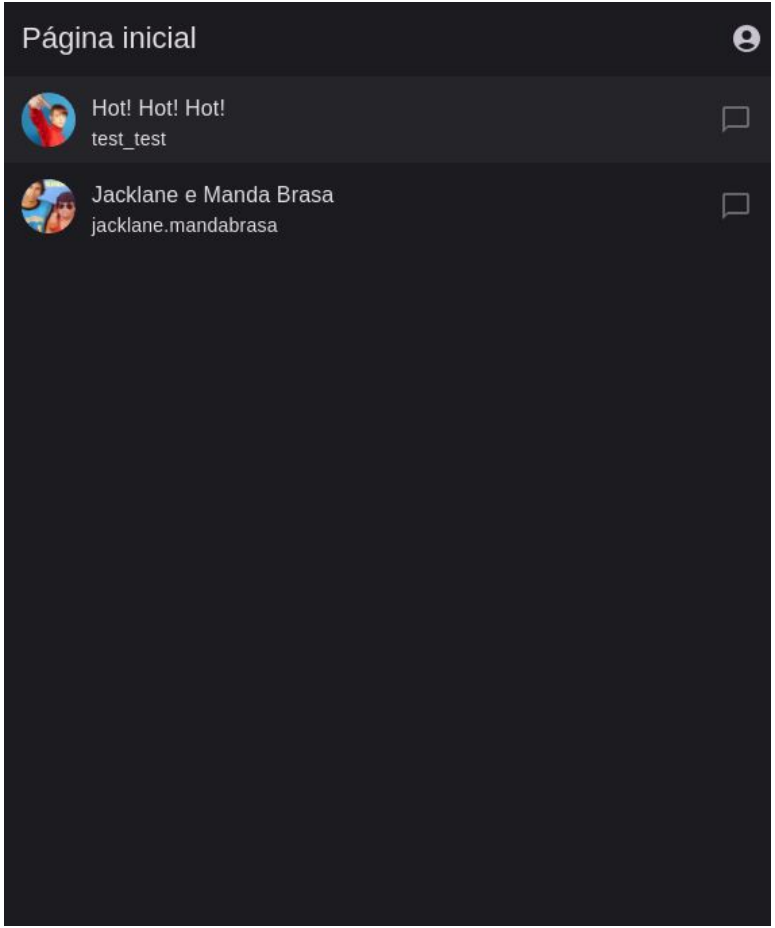


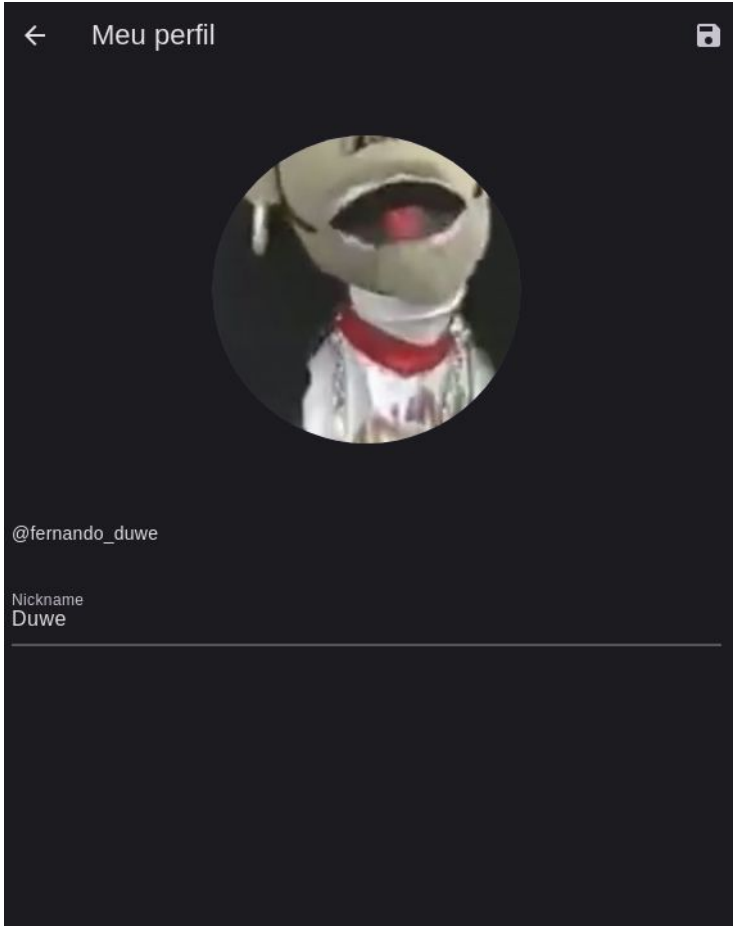
Ver mais

- No nosso chat, vamos criar um mecanismo de verificação do usuário atual
- Ao abrir a aplicação, deveremos buscar os dados do usuário no Firebase
- Na página inicial, não exibir o usuário corrente na lista de usuários cadastrados
- Na página inicial, criar uma tela de perfil, onde o usuário possa alterar seu nickname
- Na tela de chat, não deixar mais fixo qual é o usuário que está enviando a mensagem, buscando o username do usuário atual
- Utilizar uma arquitetura de software para isso, no exemplo foi usado MobX

Exercício 8 - cont

Ver mais





Exercício 9

Ver mais

- Nosso chat não está sendo atualizado conforme as mensagens são enviadas
- Altere a tela de chat, para que a lista seja atualizada conforme as mensagens são enviadas/recebidas