

# Mobile Application Development

Prof. Fernando Pinéo

Sejam bem-vindo ao  
2º Semestre 2025

# TanStack Query



É uma biblioteca que ajuda você a buscar, armazenar e manter atualizados os dados da sua aplicação (normalmente vindos de uma API).

# Por que usar o TanQuery?



Quando você busca dados de uma API em um app React ou React Native, normalmente você precisa lidar com:

- Estado de carregamento (loading)
- Estado de erro (error)
- Cache (para não buscar os mesmos dados toda hora)
- Atualizações automáticas (revalidar os dados)
- Requisições em segundo plano
- Armazenamento offline (opcional)

TanStack Query faz tudo isso automaticamente pra você!

Vejamos um exemplo..

# Sem TanQuery VS Com TanQuery

FLANP

```
1  const [data, setData] = useState([]);
2  const [loading, setLoading] = useState(true);
3  const [error, setError] = useState(null);
4
5  useEffect(() => {
6    fetch("https://api.com/dados")
7      .then(res => res.json())
8      .then(setData)
9      .catch(setError)
10     .finally(() => setLoading(false));
11  }, []);
12
13
```

## Sem TanQuery

```
1  const { data, isLoading, isError } = useQuery({
2    queryKey: ['dados'],
3    queryFn: fetchDados
4  });
```

## Com TanQuery

Já lida com cache, erros, loading e revalidação automaticamente.

# Resumindo...

Podemos assimilar o TanQuery como um garçom.

Vc solta os dados, então logo o garçom:

- ❖ Vai buscar o cardápio (API)
- ❖ Te avisa se está demorando (loading)
- ❖ Diz se deu problema (erro)
- ❖ Guarda uma cópia do cardápio (cache)
- ❖ Atualiza o cardápio sozinho quando muda (revalidação)
- ❖ Nunca te entrega o mesmo cardápio de novo se já tem ele na mão (evita requisições repetidas)

# Benefícios



Menos código para buscar dados



Melhor desempenho com cache inteligente



Refetch automático



Funciona com qualquer tipo de API (fetch, axios, GraphQL, etc)



Pode funcionar offline (com configuração extra)



Vamos a exemplo práctico...

# Criando um api exemplo



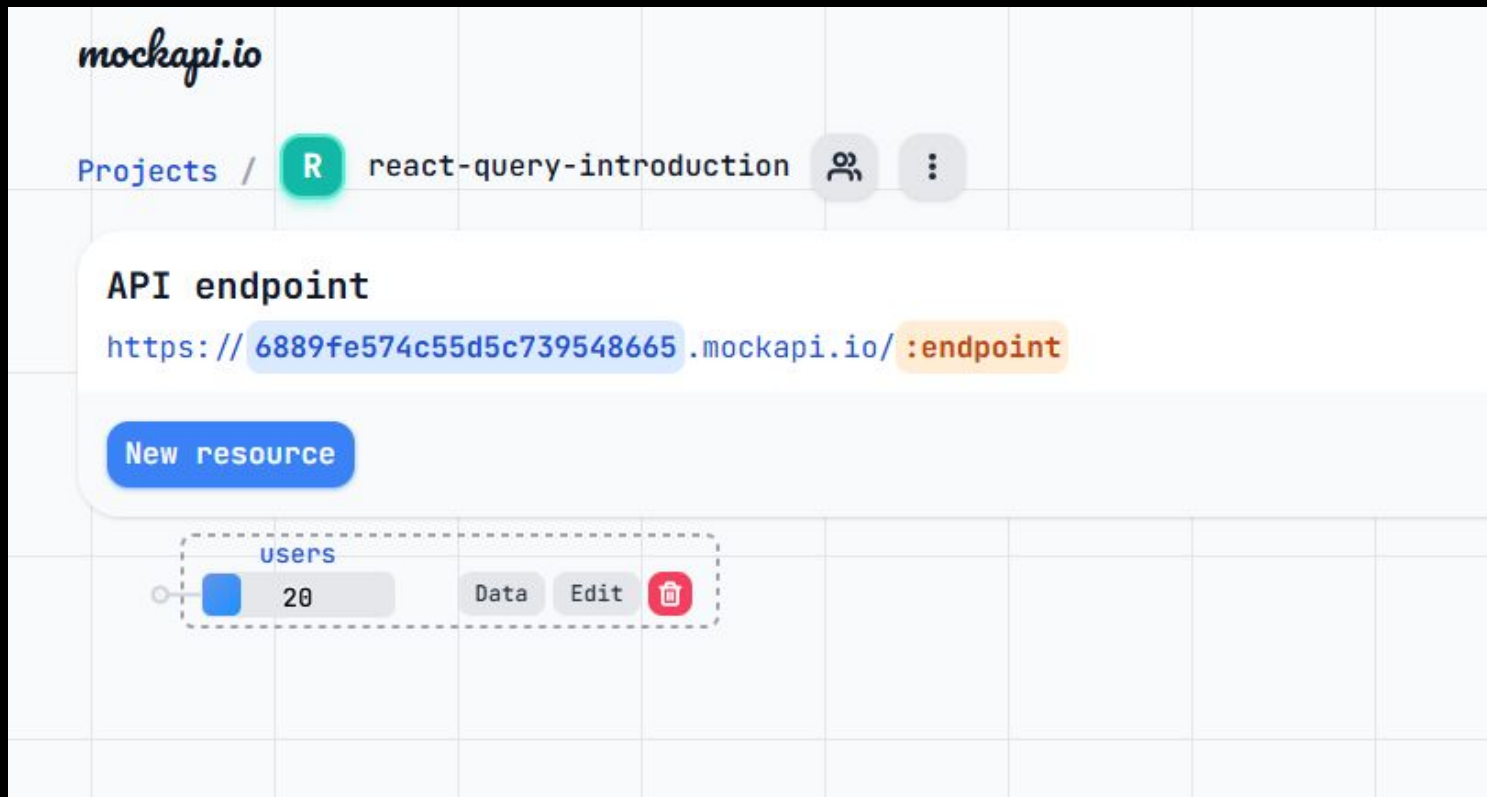
No site <https://mockapi.io/>, vc conseguirá criar uma api para simulação de forma simples e direta.

Coloque o nome do projeto como “react-query-introduction”

Crie um endpoint chamado “users”

# Criando um api exemplo

FIAP



# Criando um app com template blank



```
npx create-expo-app Aula01 --template blank
```


Instalando as dependências necessárias:

```
npx expo install react-native-safe-area-context
```

```
npm install @tanstack/react-query axios
```

# src/QueryClientProvider.js

FIAP



The image shows a code editor interface. On the left is the 'EXPLORADOR' (Explorer) sidebar showing a project structure under 'AULA01'. The 'src' directory is expanded, and 'QueryClientProvider.js' is selected and highlighted with a red box. The main editor area shows the code for 'QueryClientProvider.js'.

```
1 import React from 'react';
2 // Importa os componentes do TanStack Query
3 import { QueryClient, QueryClientProvider as TanstackProvider } from '@tanstack/react-query'
4
5 // Cria uma instância do QueryClient (controla o cache, refetch, etc)
6 const queryClient = new QueryClient();
7
8 // Um componente wrapper que vai envolver nossa aplicação
9 // Isso permite que qualquer componente filho use o TanStack Query
10 export default function QueryClientProvider({ children }) {
11   return (
12     <TanstackProvider client={queryClient}>
13       {children}
14     </TanstackProvider>
15   );
16 }
```

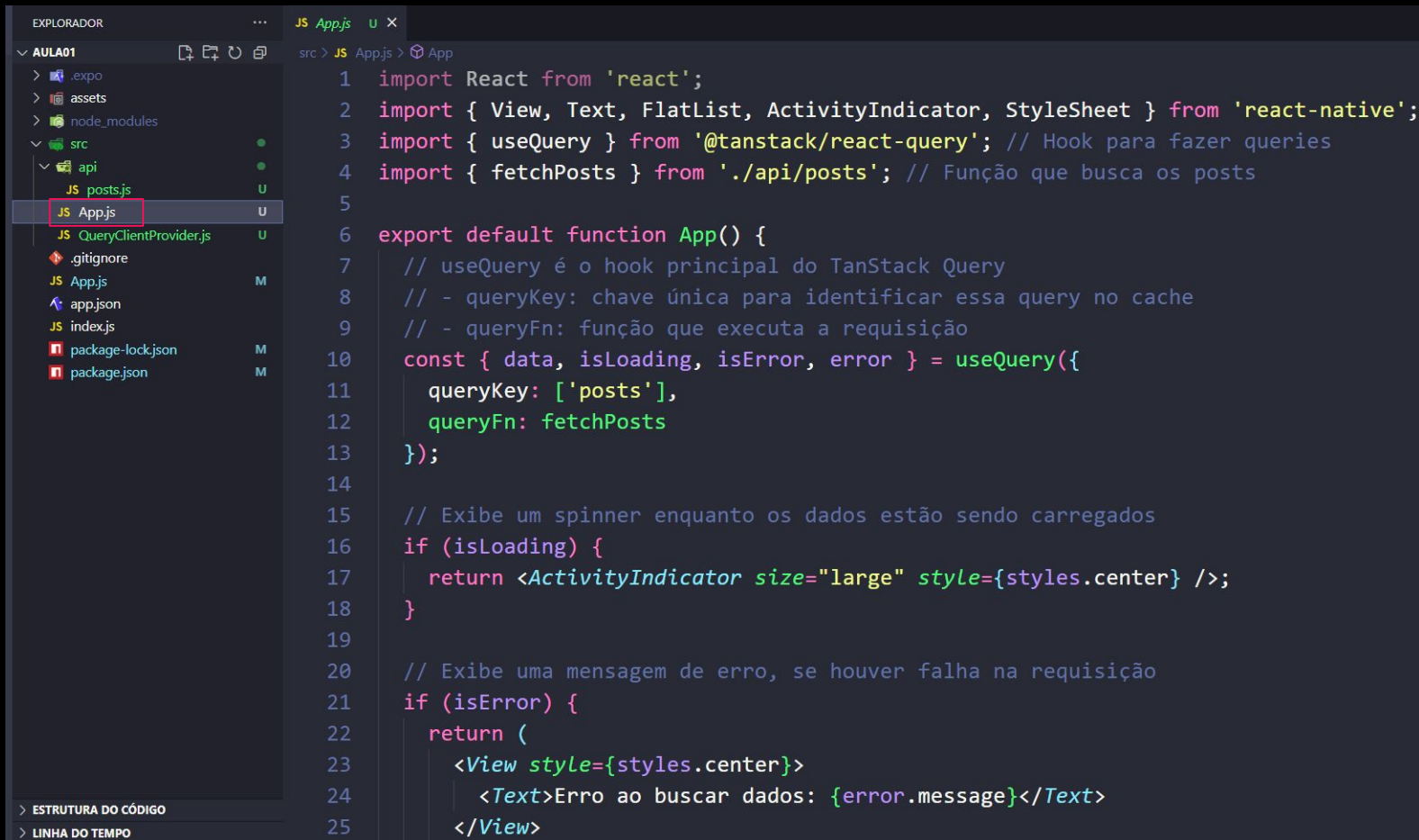
# src/api/posts.js

FIAP



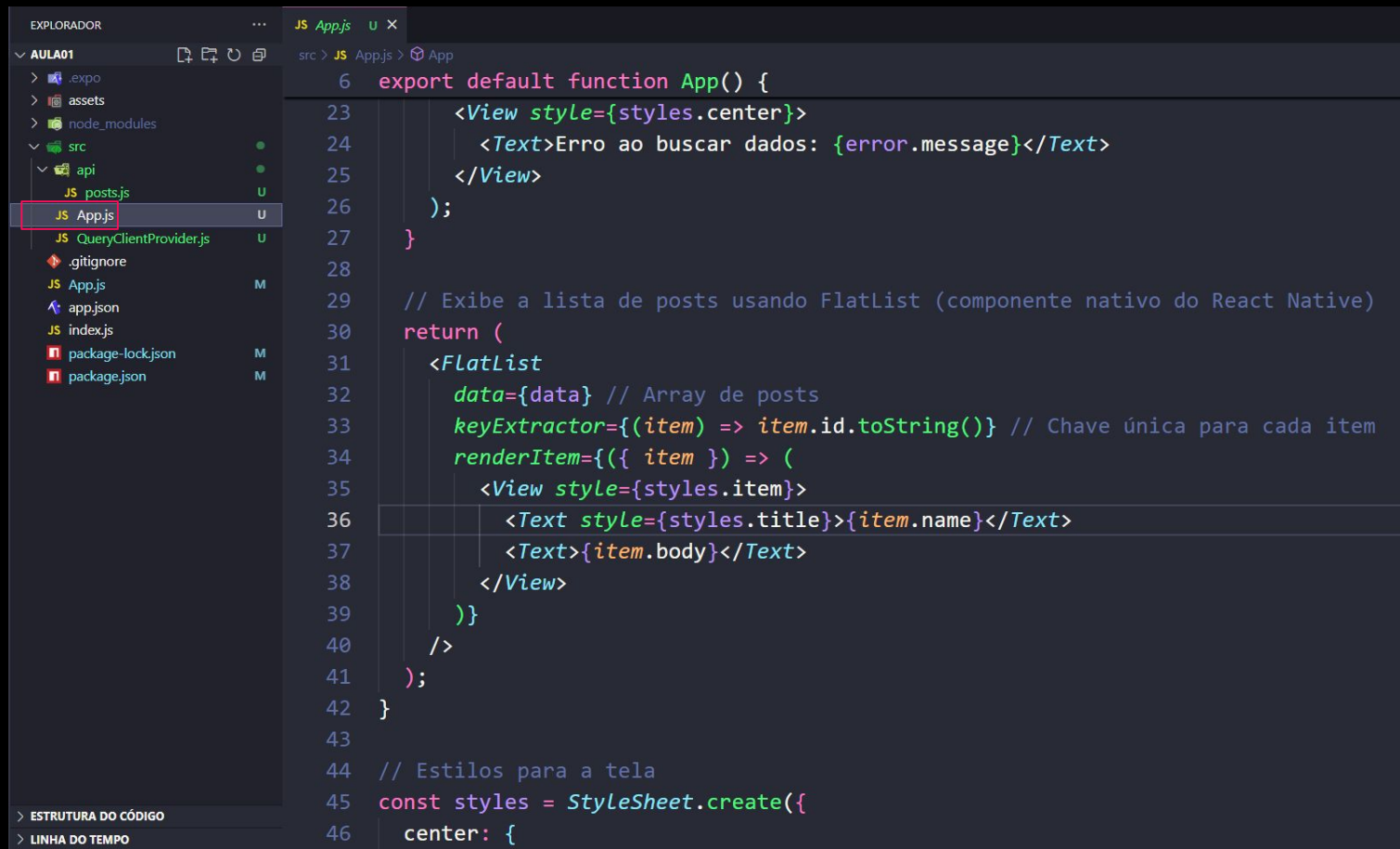
The image shows a code editor interface. On the left is the 'EXPLORADOR' (Explorer) panel showing a file tree for 'AULA01'. The tree includes folders like '.expo', 'assets', 'node\_modules', 'src', and 'api'. Inside 'api', the file 'posts.js' is selected and highlighted with a red box. Other files in 'api' include 'App.js', 'QueryClientProvider.js', '.gitignore', 'app.json', 'index.js', 'package-lock.json', and 'package.json'. On the right is the code editor for 'posts.js'. The code is written in JavaScript and uses the 'import' and 'export' syntax. It imports 'axios' and defines an asynchronous function 'fetchPosts' that uses 'axios.get' to fetch data from a specific URL. The code is as follows:

```
1 import axios from 'axios';
2
3 // Função assíncrona que busca os dados de uma API
4 // Esta função será usada pelo TanStack Query
5 export const fetchPosts = async () => {
6   //const response = await axios.get('https://jsonplaceholder.typicode.com/posts');
7   const response = await axios.get('https://6889fe574c55d5c739548665.mockapi.io/users');
8   return response.data; // Retorna os dados (array de posts)
9 };
10
```



```
EXPLORADOR
... JS App.js U x
AULA01
  > .expo
  > assets
  > node_modules
  > src
    > api
      JS posts.js U
      JS App.js U
    JS QueryClientProvider.js U
  .gitignore
  JS App.js M
  app.json
  JS index.js
  package-lock.json M
  package.json M

src > JS App.js > App
1  import React from 'react';
2  import { View, Text, FlatList, ActivityIndicator, StyleSheet } from 'react-native';
3  import { useQuery } from '@tanstack/react-query'; // Hook para fazer queries
4  import { fetchPosts } from './api/posts'; // Função que busca os posts
5
6  export default function App() {
7    // useQuery é o hook principal do TanStack Query
8    // - queryKey: chave única para identificar essa query no cache
9    // - queryFn: função que executa a requisição
10   const { data, isLoading, isError, error } = useQuery({
11     queryKey: ['posts'],
12     queryFn: fetchPosts
13   });
14
15   // Exibe um spinner enquanto os dados estão sendo carregados
16   if (isLoading) {
17     return <ActivityIndicator size="large" style={styles.center} />;
18   }
19
20   // Exibe uma mensagem de erro, se houver falha na requisição
21   if (isError) {
22     return (
23       <View style={styles.center}>
24         <Text>Erro ao buscar dados: {error.message}</Text>
25       </View>
```

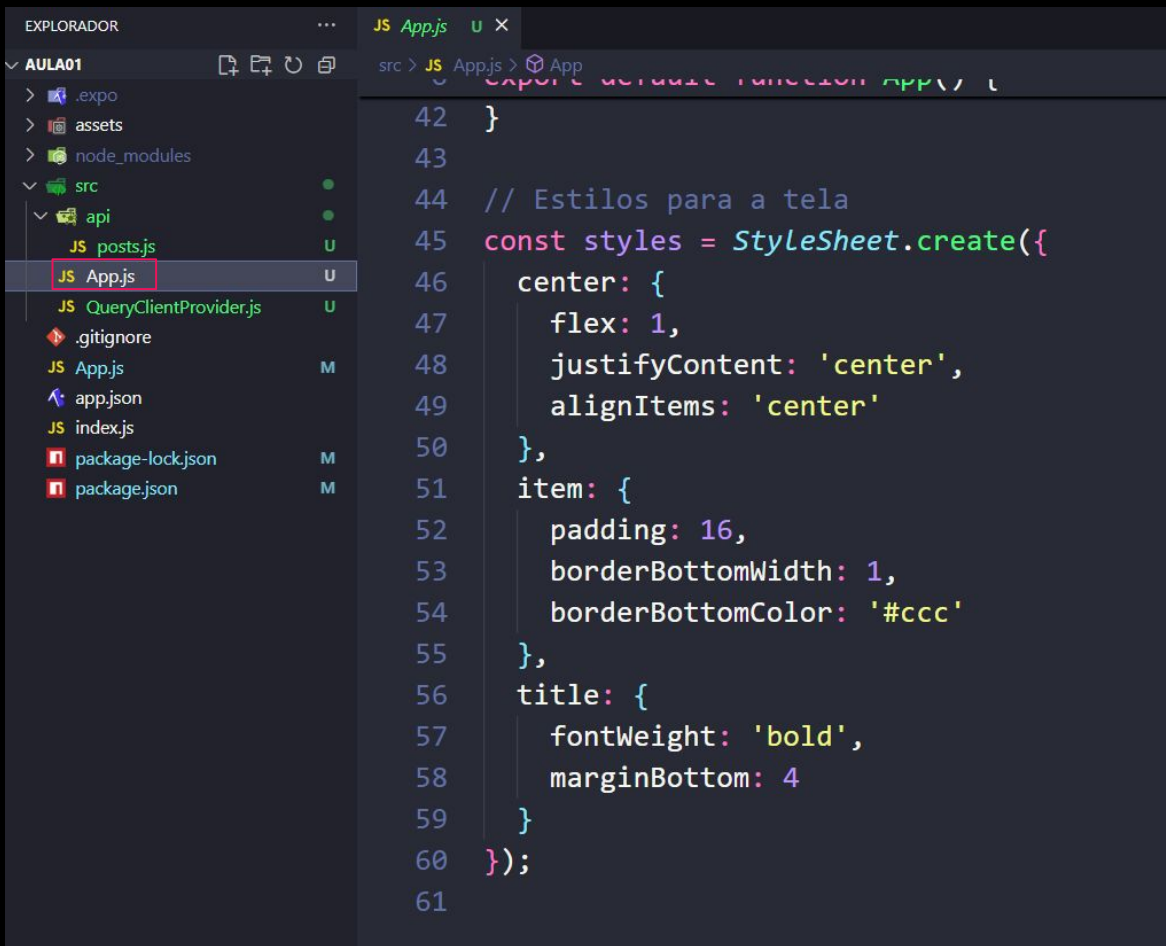


```
6 export default function App() {
23   <View style={styles.center}>
24     <Text>Erro ao buscar dados: {error.message}</Text>
25   </View>
26 );
27 }
28
29 // Exibe a lista de posts usando FlatList (componente nativo do React Native)
30 return (
31   <FlatList
32     data={data} // Array de posts
33     keyExtractor={({item}) => item.id.toString()} // Chave única para cada item
34     renderItem={({ item }) => (
35       <View style={styles.item}>
36         <Text style={styles.title}>{item.name}</Text>
37         <Text>{item.body}</Text>
38       </View>
39     )}
40   />
41 );
42 }
43
44 // Estilos para a tela
45 const styles = StyleSheet.create({
46   center: {
```



# src/App.js

FLANP

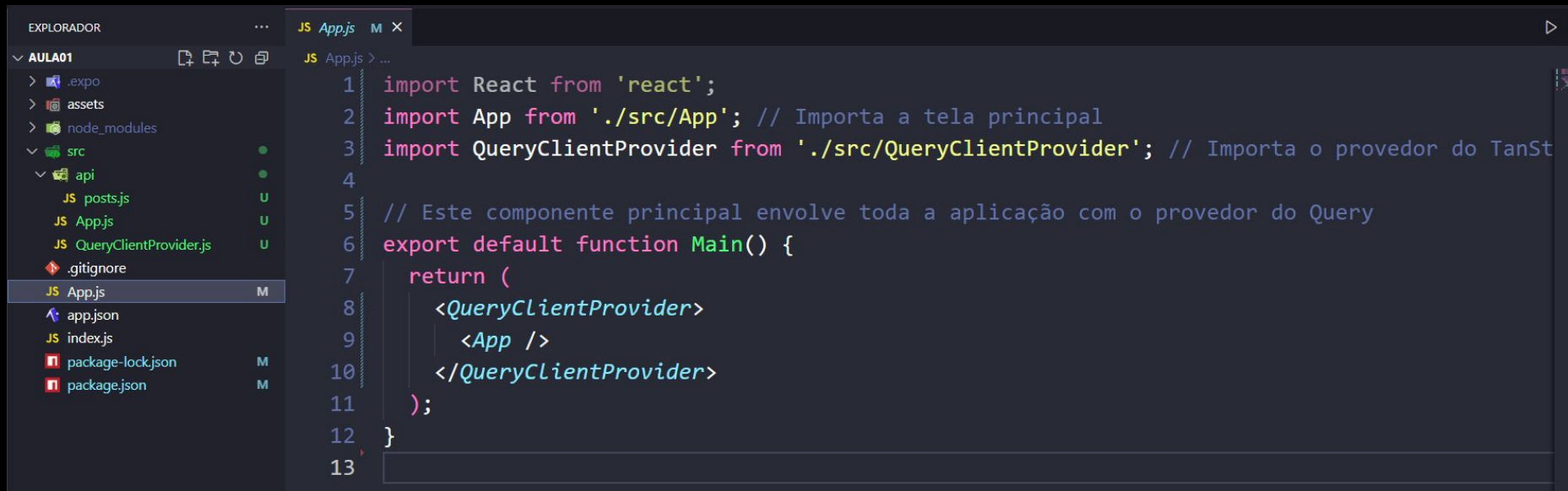


The image shows a code editor interface. On the left is the 'EXPLORADOR' (File Explorer) panel showing a project structure. The 'src' directory is expanded, and 'App.js' is selected. The main editor area shows the code for 'App.js'.

```
42 }
43
44 // Estilos para a tela
45 const styles = StyleSheet.create({
46   center: {
47     flex: 1,
48     justifyContent: 'center',
49     alignItems: 'center'
50   },
51   item: {
52     padding: 16,
53     borderBottomWidth: 1,
54     borderBottomColor: '#ccc'
55   },
56   title: {
57     fontWeight: 'bold',
58     marginBottom: 4
59   }
60 });
61
```

# App.js (Arq principal)

FIAP



The screenshot shows a code editor with a dark theme. On the left is the 'EXPLORADOR' (File Explorer) panel showing a project structure. The main editor area displays the code for 'App.js'.

**EXPLORADOR**

- AULA01
  - .expo
  - assets
  - node\_modules
  - src
    - api
      - posts.js
      - App.js
      - QueryClientProvider.js
    - .gitignore

**JS App.js**

```
1 import React from 'react';
2 import App from './src/App'; // Importa a tela principal
3 import QueryClientProvider from './src/QueryClientProvider'; // Importa o provedor do TanSt
4
5 // Este componente principal envolve toda a aplicação com o provedor do Query
6 export default function Main() {
7   return (
8     <QueryClientProvider>
9       <App />
10    </QueryClientProvider>
11  );
12 }
13
```

- `QueryClientProvider` cria e fornece o contexto do `TanStack Query` para toda a aplicação.
- Dentro do `App.js`, usamos `useQuery()` para buscar dados da API.
- O hook retorna estado (`isLoading`, `isError`, `data`) e lida com cache, revalidação, etc.
- A UI responde automaticamente ao estado da query — mostrando loading, erro ou os dados.

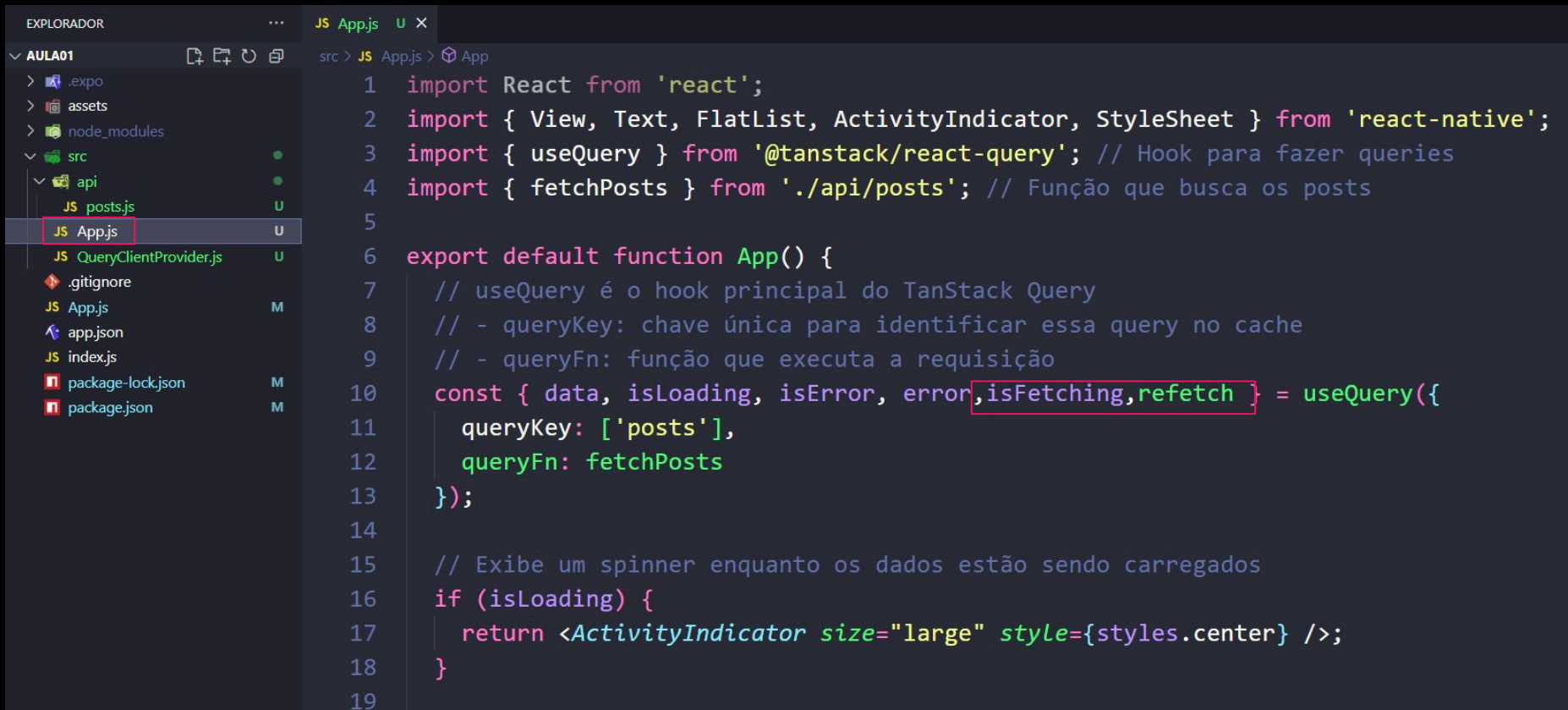
# Implementando o refetch com pull-to-refresh



- Quando o usuário puxar a lista pra baixo, a aplicação deve:
- Mostrar o carregamento
- Fazer uma nova requisição (refetch)
- Atualizar os dados da lista

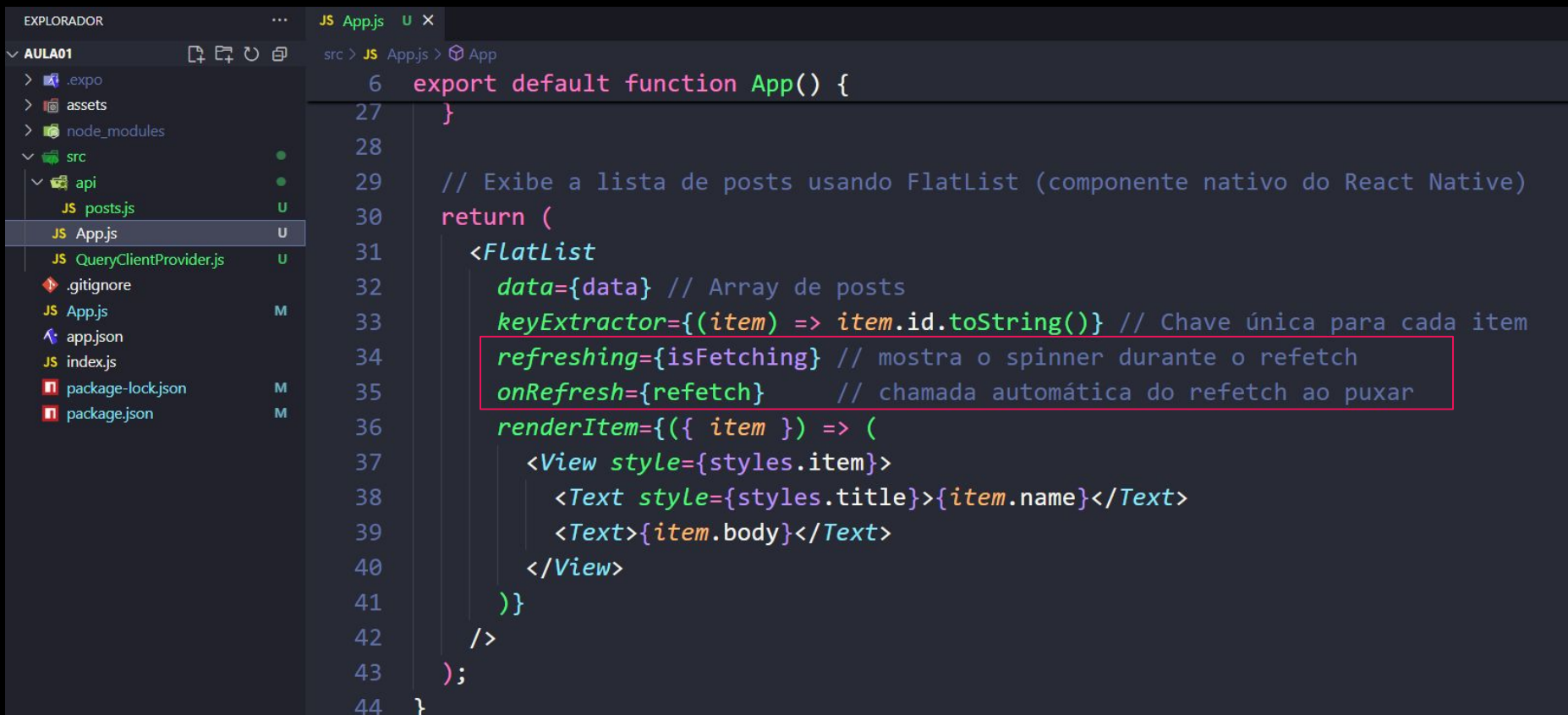
# Usar refetch e isFetching do useQuery

FILAP



```
1 import React from 'react';
2 import { View, Text, FlatList, ActivityIndicator, StyleSheet } from 'react-native';
3 import { useQuery } from '@tanstack/react-query'; // Hook para fazer queries
4 import { fetchPosts } from './api/posts'; // Função que busca os posts
5
6 export default function App() {
7   // useQuery é o hook principal do TanStack Query
8   // - queryKey: chave única para identificar essa query no cache
9   // - queryFn: função que executa a requisição
10  const { data, isLoading, isError, error, isFetching, refetch } = useQuery({
11    queryKey: ['posts'],
12    queryFn: fetchPosts
13  });
14
15  // Exibe um spinner enquanto os dados estão sendo carregados
16  if (isLoading) {
17    return <ActivityIndicator size="large" style={styles.center} />;
18  }
19
```

# Adicionar refreshing e onRefresh no FlatList



The image shows a VS Code editor interface. On the left, the 'EXPLORADOR' (Explorer) sidebar displays a project structure for 'AULA01'. The files listed include '.expo', 'assets', 'node\_modules', 'src' (containing 'api' with 'posts.js'), 'App.js', 'QueryClientProvider.js', '.gitignore', 'app.json', 'index.js', 'package-lock.json', and 'package.json'. The 'App.js' file is selected and highlighted in blue. The main editor area shows the code for 'App.js'. The code is as follows:

```
6  export default function App() {
27  }
28
29  // Exibe a lista de posts usando FlatList (componente nativo do React Native)
30  return (
31    <FlatList
32      data={data} // Array de posts
33      keyExtractor={({item}) => item.id.toString()} // Chave única para cada item
34      refreshing={isFetching} // mostra o spinner durante o refetch
35      onRefresh={refetch}      // chamada automática do refetch ao puxar
36      renderItem={({ item }) => (
37        <View style={styles.item}>
38          <Text style={styles.title}>{item.name}</Text>
39          <Text>{item.body}</Text>
40        </View>
41      )}
42    />
43  );
44 }
```

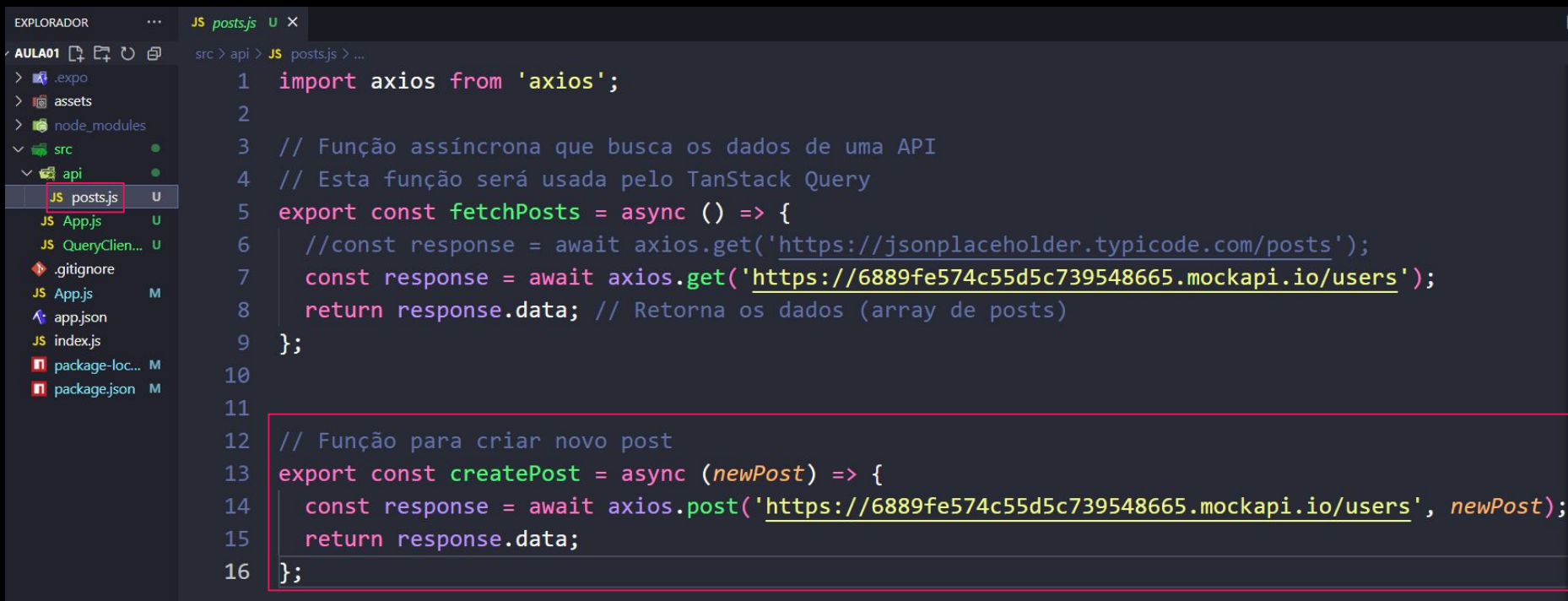
A red rectangular box highlights the following props in the `<FlatList>` component:

- `refreshing={isFetching}` // mostra o spinner durante o refetch
- `onRefresh={refetch}` // chamada automática do refetch ao puxar

# Implementando o refetch com pull-to-refresh

- refetch Reexecuta a requisição quando o usuário puxa a lista
- isFetching Indica se está buscando dados (mesmo após o carregamento inicial)
- refreshing={isFetching} Mostra o "spinner" de atualização no FlatList
- onRefresh={refetch} Associa o gesto de "puxar para atualizar" ao refetch

# Incluindo a função para criar o novo post FLANP

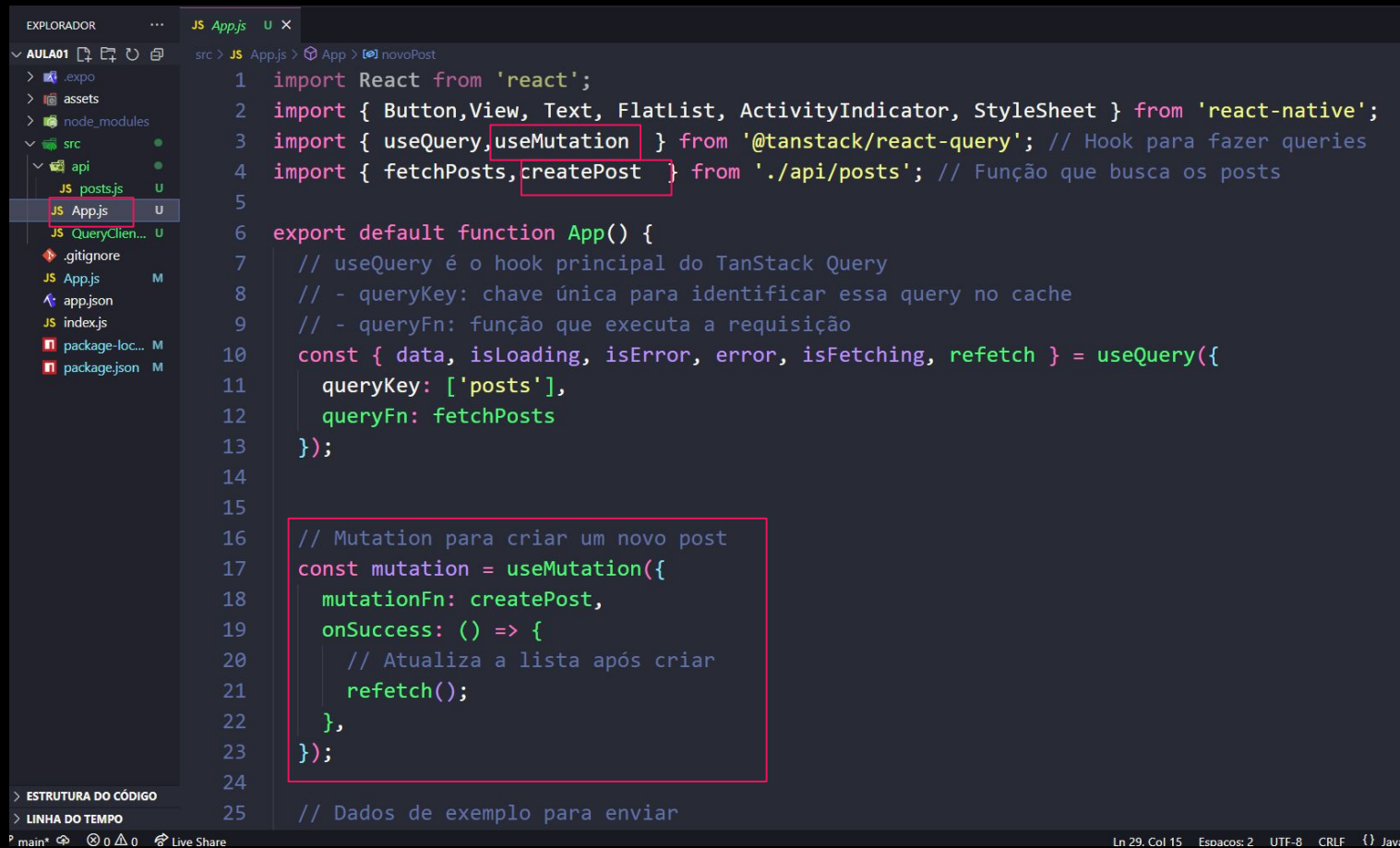


The screenshot shows a code editor with a sidebar on the left and a main editor area. The sidebar, titled 'EXPLORADOR', shows a file tree with 'AULA01' as the root. Under 'AULA01', there are folders for '.expo', 'assets', and 'node\_modules'. The 'src' folder is expanded, showing 'api' and 'posts.js' (which is highlighted with a red box). Other files in 'src' include 'App.js', 'QueryClient.js', '.gitignore', 'app.json', 'index.js', 'package-lock.json', and 'package.json'. The main editor area shows the code for 'posts.js'. The code includes an import for 'axios', a comment about an asynchronous function, and a 'fetchPosts' function. A new function 'createPost' is being added, highlighted with a red box, which uses 'axios.post' to create a new post at a specific URL.

```
1 import axios from 'axios';
2
3 // Função assíncrona que busca os dados de uma API
4 // Esta função será usada pelo TanStack Query
5 export const fetchPosts = async () => {
6   //const response = await axios.get('https://jsonplaceholder.typicode.com/posts');
7   const response = await axios.get('https://6889fe574c55d5c739548665.mockapi.io/users');
8   return response.data; // Retorna os dados (array de posts)
9 };
10
11
12 // Função para criar novo post
13 export const createPost = async (newPost) => {
14   const response = await axios.post('https://6889fe574c55d5c739548665.mockapi.io/users', newPost);
15   return response.data;
16 };
```



# Incluindo a função para criar o novo post FLANP



The screenshot shows a VS Code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with a 'src' directory containing 'api' and 'posts.js'. The code editor shows the 'novosPost' file in the 'src' directory. The code implements a React Native app using TanStack Query for data fetching and mutations. The 'App' function is the main component, which uses the 'useQuery' hook to fetch posts and the 'useMutation' hook to create a new post. The 'createPost' function is defined as a mutation that calls the 'createPost' API endpoint and updates the state after a successful request. The 'App' function also includes a 'refetch' function to update the list of posts after a new post is created.

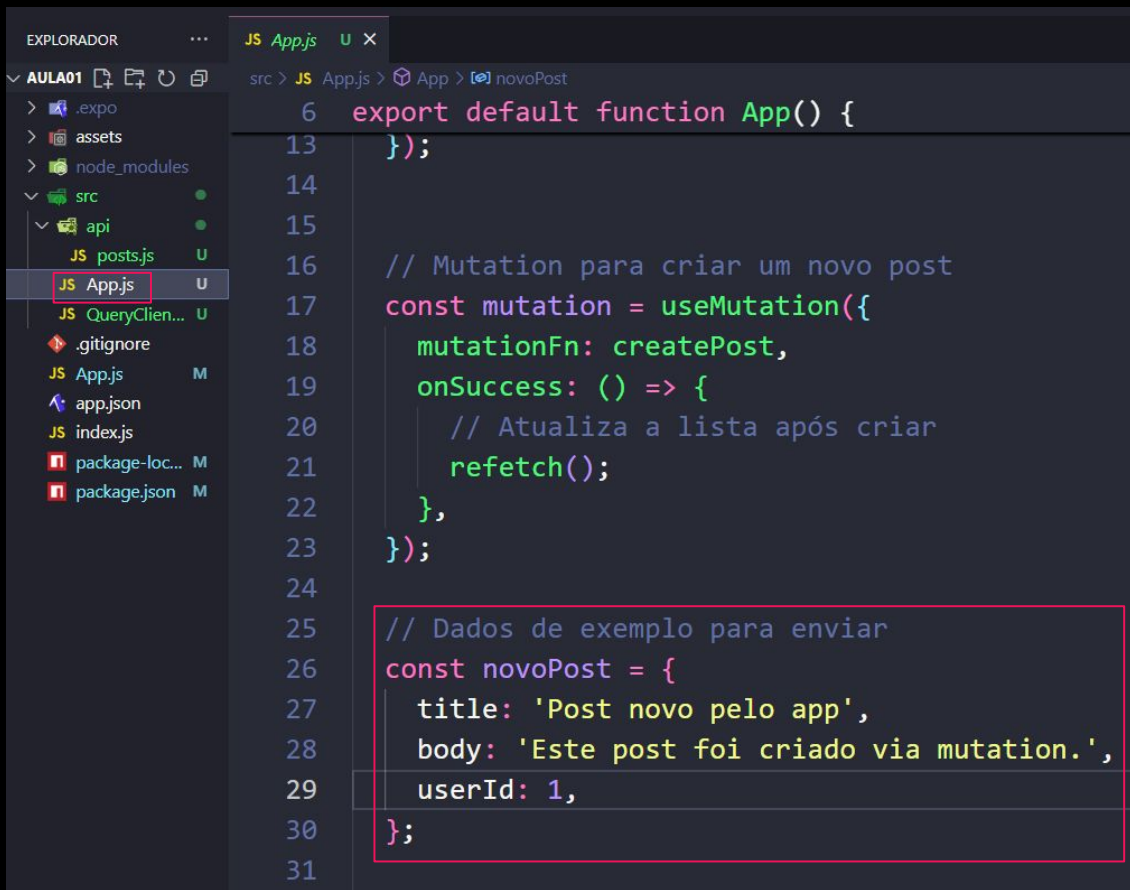
```
1 import React from 'react';
2 import { Button, View, Text, FlatList, ActivityIndicator, StyleSheet } from 'react-native';
3 import { useQuery, useMutation } from '@tanstack/react-query'; // Hook para fazer queries
4 import { fetchPosts, createPost } from './api/posts'; // Função que busca os posts
5
6 export default function App() {
7   // useQuery é o hook principal do TanStack Query
8   // - queryKey: chave única para identificar essa query no cache
9   // - queryFn: função que executa a requisição
10  const { data, isLoading, isError, error, isFetching, refetch } = useQuery({
11    queryKey: ['posts'],
12    queryFn: fetchPosts
13  });
14
15  // Mutation para criar um novo post
16  const mutation = useMutation({
17    mutationFn: createPost,
18    onSuccess: () => {
19      // Atualiza a lista após criar
20      refetch();
21    },
22  });
23
24  // Dados de exemplo para enviar
25
```

EXPLORADOR

src > JS App.js > App > novosPost

main\* 0 0 Live Share

Ln 29, Col 15 Espaços: 2 UTF-8 CRLF {} Jav

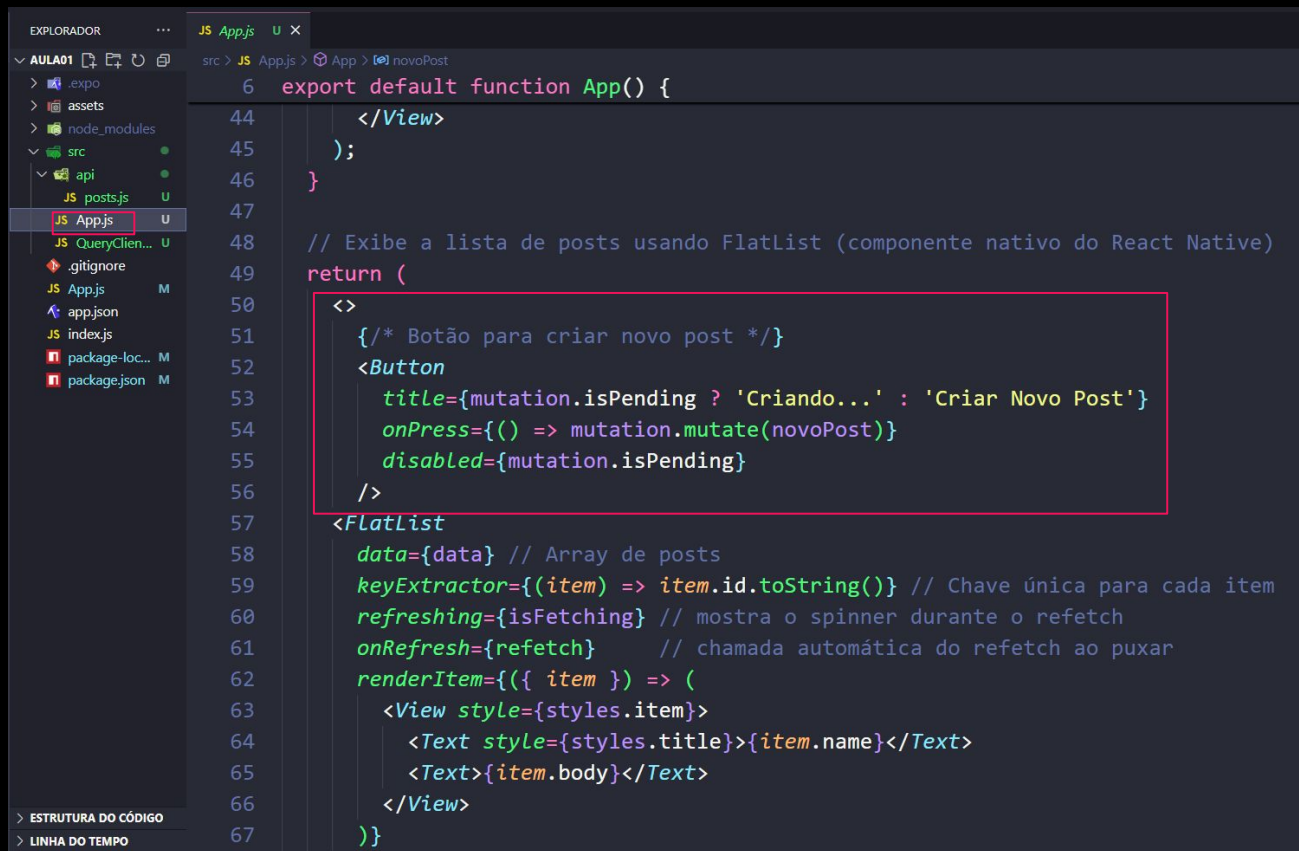


```
EXPLORADOR  ...  JS App.js  U X
AULA01  [+] [x] [u] [c]
  > .expo
  > assets
  > node_modules
  > src
    > api
      JS posts.js  U
      JS App.js    U
      JS QueryClien...  U
    .gitignore
    JS App.js      M
    app.json
    JS index.js
    package-loc... M
    package.json  M

src > JS App.js > App > [x] novoPost
6  export default function App() {
13  };
14
15
16  // Mutation para criar um novo post
17  const mutation = useMutation({
18    mutationFn: createPost,
19    onSuccess: () => {
20      // Atualiza a lista após criar
21      refetch();
22    },
23  });
24
25  // Dados de exemplo para enviar
26  const novoPost = {
27    title: 'Post novo pelo app',
28    body: 'Este post foi criado via mutation.',
29    userId: 1,
30  };
31
```

# Criando button para criar novo post

FILAP



```
6 export default function App() {
44   </View>
45 };
46 }
47
48 // Exibe a lista de posts usando FlatList (componente nativo do React Native)
49 return (
50   <>
51     { /* Botão para criar novo post */ }
52     <Button
53       title={mutation.isPending ? 'Criando...' : 'Criar Novo Post'}
54       onPress={() => mutation.mutate(novoPost)}
55       disabled={mutation.isPending}
56     />
57     <FlatList
58       data={data} // Array de posts
59       keyExtractor={({item}) => item.id.toString()} // Chave única para cada item
60       refreshing={isFetching} // mostra o spinner durante o refetch
61       onRefresh={refetch} // chamada automática do refetch ao puxar
62       renderItem={({ item }) => (
63         <View style={styles.item}>
64           <Text style={styles.title}>{item.name}</Text>
65           <Text>{item.body}</Text>
66         </View>
67       )}
```

Dúvidas?