Tecnologias Para Desenvolvimento Web

UNIDADE 06

Firebase

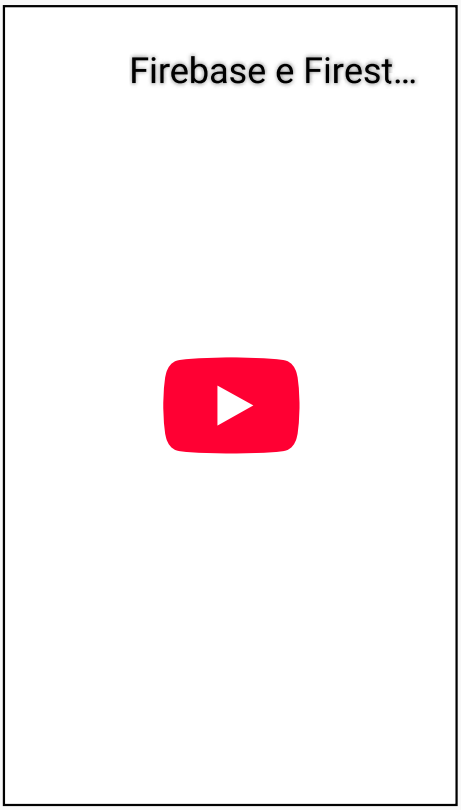
Firebase

Para a persistência dos dados, utilizaremos o banco de dados Firestore, que faz parte da plataforma Firebase. O Firebase refere-se a uma plataforma de back-end como serviço (BaaS) que oferece uma ampla gama de ferramentas e recursos para desenvolvimento móvel e da web [1]. É basicamente um programa de banco de dados NoSQL que segue um protocolo de documento JSON para armazenar dados do usuário.

Primeiramente, precisamos ter uma conta do google. Sendo assim, crie uma conta do google gratuitamente e acesse a página https://firebase.google.com.

Banco de Dados: Firestore

Para a criação e configuração do Firebase, assista ao vídeo da semana.



Após termos o Firebase com o banco de dados criado, agora precisaremos instalar os pacotes do Firebase em nosso projeto React. Sendo assim, execute no terminal o comando indicado na Figura 1, dentro da pasta do projeto React.

Figura 1. Comando de instalação do Firebase.

```
npm install firebase@8.9.1
```

Fonte: o autor, 2021.

Após fazer a instalação, já podemos utilizar o Firebase em nosso projeto. Desta forma vamos criar um novo arquivo de conexão com a plataforma Firebase. Crie um arquivo chamado **Firebase.js** dentro da pasta **src** do seu projeto React. Este arquivo será responsável por fazer a conexão e importação dos pacotes necessários para a utilização do Firebase. Ele será nada mais do que um novo componente React, mas com métodos do Firebase.

Desta forma, precisamos colocar todo o código necessário para a conexão e importação dentro do arquivo, como pode ser visto na Figura 2.

Figura 2. Arquivo do Firebase.

```
1   import firebase from 'firebase/app';
2   import 'firebase/firestore';
3
4   let firebaseConfig = {
5       apiKey: "ASyDKm4h5gAC8uQr5HwPYuCEYNWU-kd8UwF1zw",
6       authDomain: "projetoead-faa62.firebaseapp.com",
7       databaseURL: "https://projetoead-faa62-default-rtdb.firebaseio.com",
8       projectId: "projetoead-faa62",
9       storageBucket: "projetoead-faa62.appspot.com",
10      messagingSenderId: "20124567658",
11      appId: "1:201156964565:web:265595ace1d33e47a465f4",
12      measurementId: "G-KMH4F64TD3"
13  };
14
15  if(!firebase.apps.length){
16      firebase.initializeApp(firebaseConfig);
17  }
18
19  export default firebase;
20
```

Fonte: o autor, 2021.

A linha 1 indica que o componente do Firebase para a conexão com a plataforma está sendo importado. Já a linha 2, é a importação do pacote do Firestore, que é o banco de dados que vamos utilizar, que faz parte do Firebase. Na linha 4, estamos criando uma nova variável e colocando dentro dela um objeto com os parâmetros necessários para a configuração do banco de dados Firestore que foi criado. A linha 15, serve para que façamos a inicialização do Firebase apenas uma única vez, sendo assim, caso esta variável tenha o **app.length** diferente de **true**, significa que é falso, ou seja, ainda não foi inicializado. Então neste caso, entraria dentro da condição e inicializaria o Firebase. Na próxima vez que algum componente importar este componente do Firebase, caso já esteja inicializado (igual a **true**), não precisará inicializar tudo novamente. Esta condição não é obrigatória, mas é uma boa prática, por uma questão de melhor desempenho.

O Firestore é um banco de dados NoSQL orientado a documentos. Isso significa que temos dois tipos de dados em nosso banco de dados, **documentos**, que são objetos com os quais podemos trabalhar, e coleções que são **contêineres** que agrupam esses objetos. Com o Firestore podemos armazenar dados como coleções de documentos, que por sua vez, são fáceis de armazenar, pois são muito semelhantes aos da árvore JSON e também.

Para gravar algum dado o Firestore, basta criarmos um novo método para a gravação e executarmos o código da Figura 3.

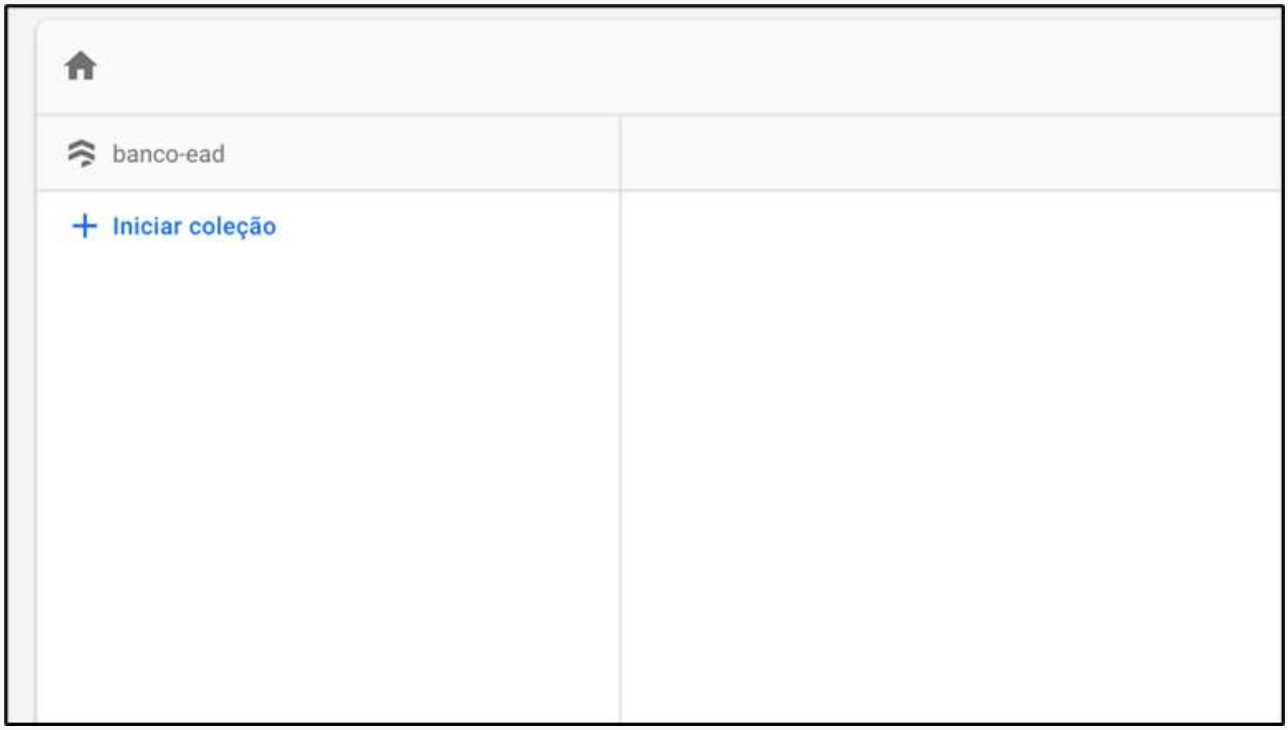
Figura 3. Importação dos componentes para as rotas.

```
firebase.firestore().collection("usuario").add({
  nome: "Eduardo",
  sobrenome: "Lino"
});
```

Fonte: o autor, 2021.

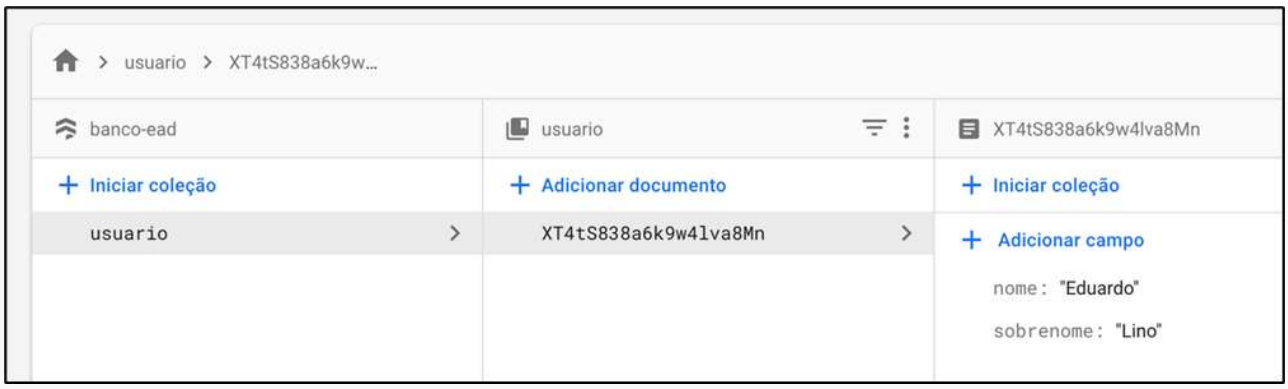
Da forma como foi feito na Figura 3, estamos executando o Firestore, que está dentro do Firebase e acionando a coleção chamada “usuário”. Esta coleção ainda não existia, antes do código ser executado, mas não há problema! Com o Firebase, se estamos adicionando um novo documento a uma coleção inexistente, ela cria a coleção e adiciona os dados do documento nesta coleção. Após acessar então a coleção, é executado o método **add()**, que passa como parâmetro o objeto Javascript, com chave e valor dos dados que desejamos. Antes de executar este código, nosso banco de dados Firestore estava exatamente como é ilustrado na Figura 4, agora que o código foi executado, temos o que é ilustrado na Figura 5.

Figura 4. Firestore vazio.



Fonte: o autor, 2021.

Figura 5. Firestore preenchido.



Fonte: o autor, 2021.

Perceba que temos a coleção **usuario** e dentro dela, o documento **XT4tS838a6k9w4lva8Mn**. Este valor é um índice que foi criado para este documento. No Firestore, podemos adicionar um índice manualmente (Figura 6) ou simplesmente não passar nada e fazer com que crie automaticamente um índice aleatório.

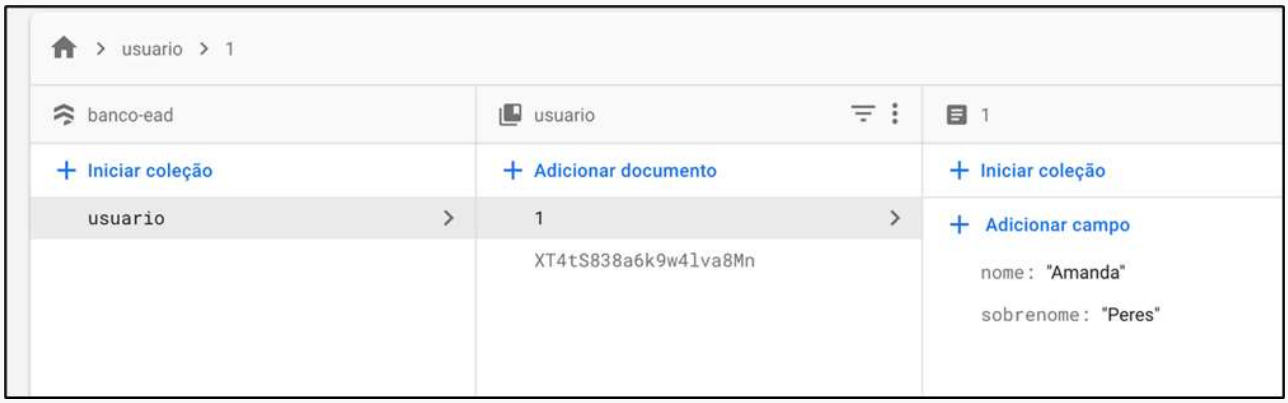
Figura 6. Adicionando um índice manual.

```
firebase.firestore().collection("usuario").doc("1").set({
  nome: "Amanda",
  sobrenome: "Peres"
});
```

Fonte: o autor, 2021.

Desta forma, temos a nossa coleção **usuario** com dois documentos adicionados, conforme é ilustrado na Figura 7.

Figura 7. Firestore com documentos.



Fonte: o autor, 2021.

Para listar os dados de uma coleção na página de seu projeto, devemos executar o método **get()**. Para isso, basta que execute o código ilustrado na Figura 8 e perceba que irá ocorrer um problema no console da página ao imprimir o retorno do método **get()**,

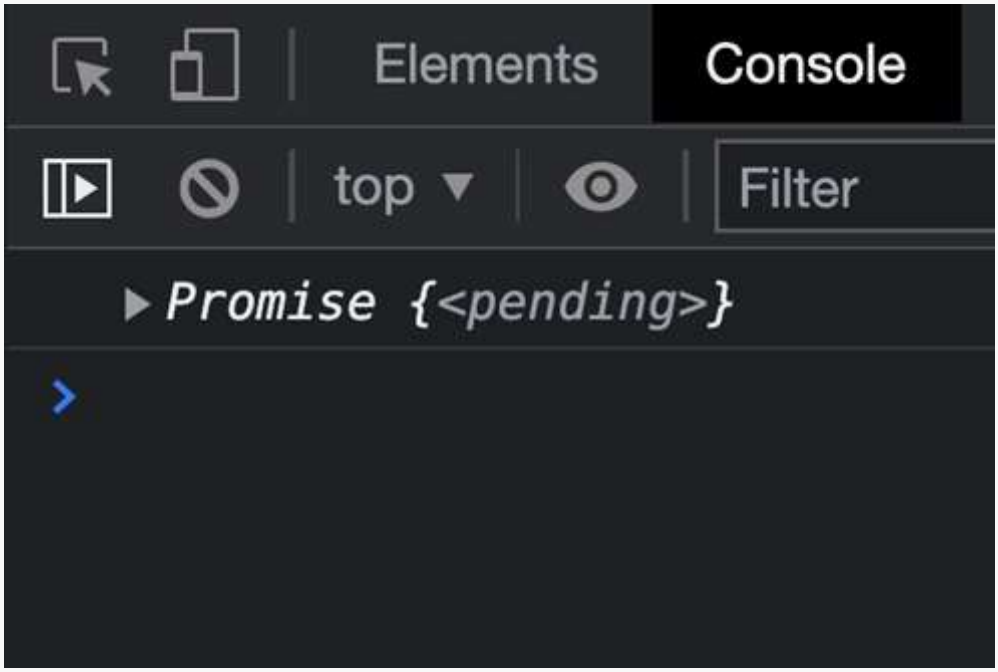
como é demonstrado na Figura 9.

Figura 8. Listando dados do Firestore.

```
var dados = firebase.firestore().collection("usuario").get();
console.log(dados);
```

Fonte: o autor, 2021.

Figura 9. Promisse.



Fonte: o autor, 2021.

Perceba que não está sendo mostrado no console da página os dados retornados da coleção **usuario**, conforme executamos em nosso código, isso se dá pelo fato de que o método **get()** é uma **Promisse** (Promessa).

Uma **Promisse** é uma promessa de execução e não a execução em si! Ou seja, quando executamos uma **Promisse** o método está nos dizendo que promete executar esse código, ele apenas não sabe quando irá nos retornar o resultado. Também podemos dizer que uma **Promisse** é uma execução Assíncrona! Sendo assim, o código está sendo executado, mas foi feita execução e o processamento está lá dentro da plataforma do Firebase e ainda não deu tempo de terminar. Quando executamos o código para mostrar o retorno na tela, ela ainda era uma promessa e finalizada por completo. Sendo assim, precisamos de um método de retorno, também chamado de call-back. Esse método, garante que quando terminar a execução, teremos o retorno dentro dele. Esse método se chama **then()**. Ao colocar o método **then()** logo em seguida da execução do método **get()**, teremos o call-back da chamada e quando terminar, o retorno estará dentro da variável passada como parâmetro do método **then()**.

A Figura 10 ilustra o código da execução do método **get()** e com o retorno dentro do método **then()**.

Figura 10. Call-back da execução do método get().

```
firebase.firestore().collection("usuario").get().then((retorno) => {

  console.log(retorno);

});
```

Fonte: o autor, 2021.

Ainda assim, precisamos percorrer a variável **retorno** para que possamos obter os itens gravados na coleção **usuario**. Desta forma, podemos utilizar o laço de repetição **forEach** e percorrer todos os itens desta coleção, conforme é ilustrado na Figura 11.

Figura 11. ForEach dos itens do Firestore.

```
firebase.firestore().collection("usuario").get().then((retorno) => {
  retorno.forEach((item) => {
    console.log(item.data().nome);
  });
});
```

Fonte: o autor, 2021.

Perceba que o objeto **item** possui um método chamado **data()**, necessário para obter os dados deste item. Sendo assim, precisamos executar este método e acessar o atributo publico **nome**, pois **nome** é o descritivo da chave que foi gravada no Firestore anteriormente (Figura 3 e 6).

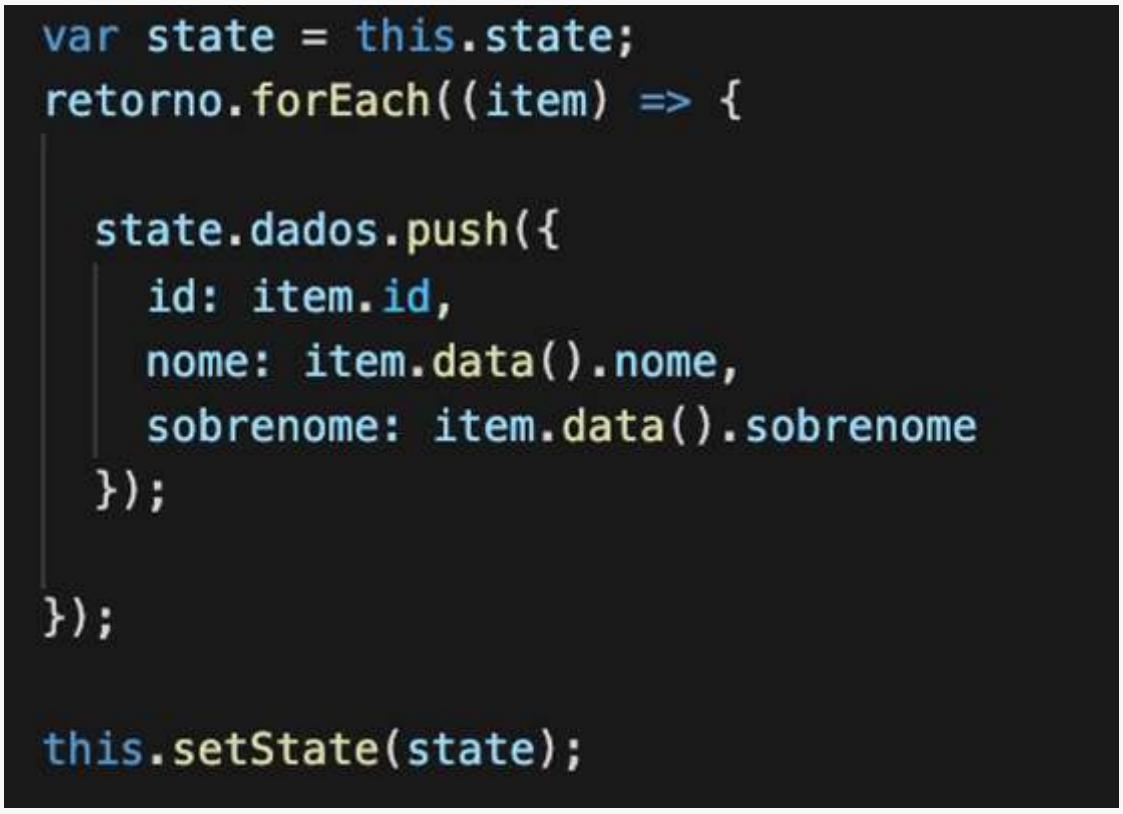
Da forma como estamos realizando o passo a passo para mostrar os dados, não é a forma como trabalhar nesta disciplina, pois os dados estão sendo mostrados no console e apesar do console ser muito útil e importante, vamos mostrá-los também, na página HTML de nossa aplicação React. Para isso, ao invés de mostrar os dados no console, precisamos colocar estes dados em um novo array, que deve estar na **state** do componente, conforme é ilustrado na Figura 12.

Figura 12. Array no console do componente.

Fonte: o autor, 2021.

Após criar o array dados, já podemos colocar o retorno dos dados do Firestore, dentro deste array, conforme podemos ver na Figura 13.

Figura 13. Lógica de inserção dos dados Firestore no array.



Fonte: o autor, 2021.

Agora já temos os dados que foram retornados do Firestore, dentro da variável **dados** da state do componente. Como já sabemos, ao executar o método **setState**, o estado do componente irá ser atualizado, desta forma, precisamos fazer com que os dados deste array dados esteja no HTML do componente, conforme é ilustrado na Figura 14.

Figura 14. Lógica de visualização dos dados do array no HTML.

Fonte: o autor, 2021.

Neste exemplo, o laço de repetição **map** possui uma variável como parâmetro, que é o **item**. Esta variável **item**, representa cada objeto que está inserido dentro do nosso array e para mostrá-los na página, precisamos também dentro do **map**, o método **return**, com o retorno do nosso HTML.

Desta forma, conseguimos concluir a etapa de persistência em banco de dados NoSQL. Perceba que não foi necessário nenhum conhecimento em SQL ou MySQL para que conseguisse gravar ou obter os dados em banco de dados e mostrá-los em sua aplicação. Com a plataforma Firebase, podemos fazer isso de uma forma muito simples e rápida.

CONCLUSÃO

Nesta unidade, foi apresentado o conceito de componentes em classe e estado do componente. Estes dois conceitos são essenciais para o andamento do nosso conteúdo e entendimento da construção de componentes mais avançados.

REFERÊNCIAS

[1] React. ***React – Uma biblioteca JavaScript para criar interfaces de usuário****. Disponível em: https://pt-br.reactjs.org/. Acessado em: 12 de outubro de 2021.*