ASIMOV JR

FIREBASE

DESKTOP
07/2024

O seguinte documento apresenta como finalidade facilitar a compreensão de uma das tecnologias mais importantes utilizadas na área de Desktop da Asimov Jr, o banco de dados do Firebase. De forma geral, a plataforma oferece muitos outros recursos além do armazenamento de dados, bem como seu suporte a sistemas de autenticação e de hosting. Nesse documento abordaremos as funcionalidades mais utilizadas nos projetos da empresa de forma aplicada e breve, uma vez que a obtenção de informações mais específicas deve ser feita na documentação oficial do Firebase.

Lembrando que este documento **está sujeito a alterações**, com o intuito de sempre mantê-lo **atualizado** e **capaz de atender as necessidades** da área de Desktop da Asimov Jr.

O guia será dividido em 3 principais sessões, sendo elas:

- Integração Firebase-Angular
  - Criação do projeto no Firebase
  - Angular: versões anteriores
  - Angular: versão atual

- Sistema de Autenticação
  - Registrar
  - Entrar
  - Sair

- E-mail de verificação
- Alterar senha
- Persistência, Estado e Usuário atual

- Firebase Firestore (Banco de dados)
  - O que é um banco de dados?
  - Inserção de dados
  - Consultas no banco
  - Remoção e edição de dados

Asimov Jr

Integração Firebase-Angular

A integração entre o Angular e o Firebase é o passo inicial para a utilização de todos os recursos previstos. Sua realização é simples, entretanto, com a retirada do "**app.module.ts**" das versões atuais do Angular, sua abordagem foi alterada.

Criação do projeto no Firebase

Para a conexão do Firebase com o framework utilizado em Desktop na Asimov Jr., primeiramente é necessário criar o projeto. Após isso, na interface "Comece adicionando o Firebase ao seu aplicativo", selecione a opção "**Web**", dê um nome e copie as configurações do seu App Firebase, que possui essa aparência:

```
const firebaseConfig = {
    apiKey: "SUA_API_KEY",
    authDomain: "SEU_AUTH_DOMAIN",
    projectId: "SEU_PROJECT_ID",
    storageBucket: "SEU_STORAGE_BUCKET",
    messagingSenderId: "SEU_MESSAGING_SENDER_ID",
    appId: "SEU_APP_ID"
  }
```

Feito isso, é possível prosseguir.

Angular: versões anteriores

Esse caso ocorre em certos projetos vigentes da Asimov Jr.. Em versões antigas, na presença do "**app.module.ts**", a configuração acontece da seguinte forma:

- Execute o seguinte comando:

```
ng add @angular/fire
```

- Na pasta **src/environments**, acesse seu environment.ts
- Nesse arquivo, adicione sua **firebaseConfig**, com a seguinte aparência:

```
export const environment = {
  production: false,
  firebaseConfig: {
    apiKey: "API_KEY",
    authDomain: "auth-domain.firebaseapp.com",
    projectId: "project-id",
    storageBucket: "storage-bucket.appspot.com",
    messagingSenderId: "sender-id",
    appId: "app-id"
  }
};
```

- Por fim, em seu **app.module.ts**, faça a importação dos recursos do Firebase, conectando com seu environment.ts, da seguinte forma:

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AngularFireModule } from '@angular/fire';
import { AngularFireAuthModule } from '@angular/fire/auth';
import { AngularFirestoreModule } from '@angular/fire/firestore';
import { environment } from '../environments/environment';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule,
    AngularFireModule.initializeApp(environment.firebaseConfig),
    AngularFireAuthModule, // Módulo de autenticação
    AngularFirestoreModule // Módulo Firestore
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

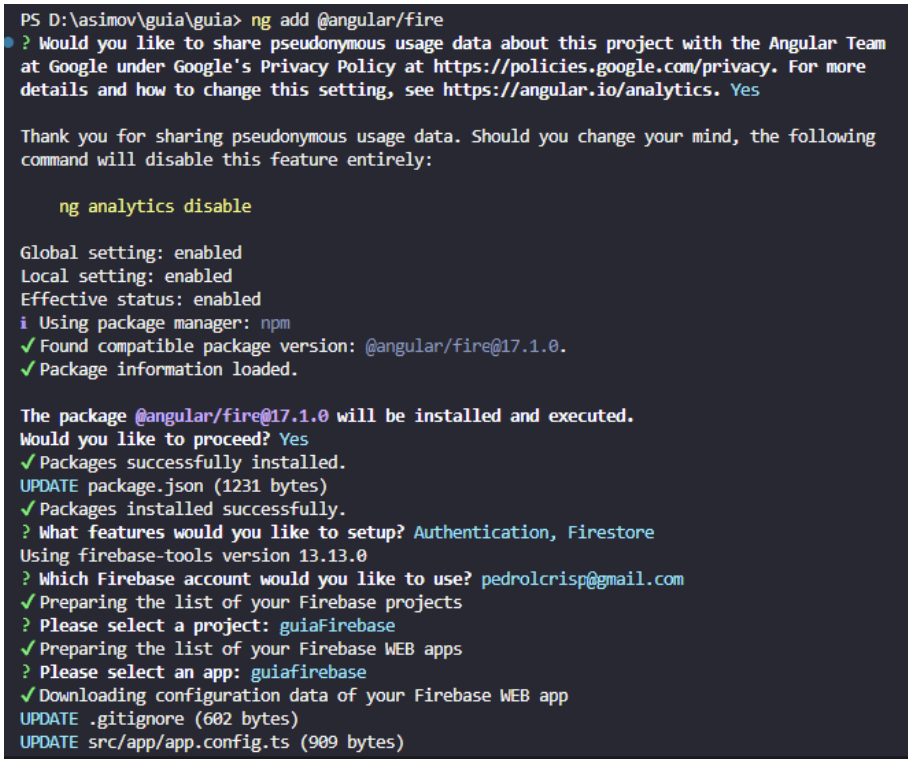
Com isso, seu projeto do Firebase está integrado com o Angular, e você pode desfrutar de seus recursos!

Angular: versão atual

Na versão atual, sem o "**app.module.ts**", a integração com o Angular ocorre de forma muito mais simplificada, sendo feita apenas a partir do terminal. Ao executar o comando:

```
ng add @angular/fire
```

A configuração será feita da seguinte forma:



E por fim, seu "**app.config.ts**" terá a seguinte aparência, de forma automática:

```
import { ApplicationConfig } from '@angular/core';
import { provideRouter } from '@angular/router';

import { routes } from './app.routes';
import { provideClientHydration } from '@angular/platform-browser';
import { initializeApp, provideFirebaseApp } from '@angular/fire/app';
import { getAuth, provideAuth } from '@angular/fire/auth';
import { getFirestore, provideFirestore } from '@angular/fire/firestore';

export const appConfig: ApplicationConfig = {
  providers: [
    provideRouter(routes),
    provideClientHydration(),
    provideFirebaseApp(() => initializeApp(
      {"projectId":"guiafirebase-505f1",
        "appId":"1:799316376261:web:e0bc91ec3c37b4cfad4eac",
        "storageBucket":"guiafirebase-505f1.appspot.com",
        "apiKey":"AIzaSyBgIacU8-D2mNV1aYxOXAvEhBt-o2kIjYc",
        "authDomain":"guiafirebase-505f1.firebaseapp.com",
        "messagingSenderId":"799316376261"})),
    provideAuth(() => getAuth()),
    provideFirestore(() => getFirestore())]
};
```

Pronto! Seu projeto Firebase está integrado com o Angular, e pronto para ser utilizado. Porém uma **OBSERVAÇÃO** é importante: durante a confecção deste documento, o teste foi realizado com um projeto Angular com o nome “guiaFirebase”. Com isso, o seguinte erro foi gerado ao executar “ng add @angular/fire”:

```
✓ Packages installed successfully.
? What features would you like to setup? Authentication, Firestore
Using firebase-tools version 13.13.0
? Which Firebase account would you like to use? pedrolcrisp@gmail.com
Invalid project id: guiaFirebase.
Note: Project id must be all lowercase.
```

Portanto, é importante que para evitar esse obstáculo o projeto utilizado contenha apenas letras minúsculas, tal como “guia”.

Sistema de autenticação

O sistema de autenticação do Firebase conta com funções que automatizam o processo, facilitando a abordagem. Para tornar mais claro, é importante frisar que em sua essência as

funções para qualquer caso são as mesmas, entretanto esse guia **propõe** ações conjuntas que configuram boas práticas. **Não utilize a abordagem proposta como regra**, pesquise e adeque seu sistema para as suas necessidades.

Antes de iniciarmos, é crucial ativarmos a opção "**Authentication**" em seu projeto Firebase, e selecionar a opção "**E-mail/senha**". Agora é possível prosseguir.

- Crie um "service" que conterá todas as funções relacionadas a autenticação de usuário.
- No construtor do seu componente, inclua a importação do serviço de autenticação do Firebase da seguinte forma, o qual permitirá a utilização de todas as funções:

```
constructor(
    private auth: AngularFireAuth
  ) { }
```

Registrar

Para registrar um novo usuário, é utilizada a função "**createUserWithEmailAndPassword()**", passando como parâmetros o email e a senha, nessa ordem. Como boas práticas, é interessante registrar as informações do usuário no banco de dados após o sucesso na criação do perfil (representado na imagem por **SetUserData()**, passando "user" que pode conter diversas informações, como user.country e user.phone por exemplo).

Além disso, utilizar uma função que aplica a lógica de verificação de e-mail para novas contas também é importante, garantindo a segurança do seu projeto (função que será tratada posteriormente nesse documento).

Por fim, vale citar que o usuário criado a partir desse método, se não adicionado no banco de dados, apresentará apenas e-mail, senha e uma tag de identificação "**uid**" gerada automaticamente. Por isso, a adição dos usuários no banco de dados é importante para fornecer mais informações.

```
SignUp(user: UserInterface) {
    return this.auth
      .createUserWithEmailAndPassword(user.email, user.password)
      .then(() => {
        this.verifyEmail();
        this.SetUserData(user);
      })
      .catch((error) => {
        window.alert(error.message);
      });
  }
```

Entrar

Para o login de usuários, utiliza-se "**signInWithEmailAndPassword()**", também utilizando e-mail e senha como parâmetros. Nesse sentido, após a utilização dessa função cabe salvar o identificador do usuário "**uid**" no localStorage, pois suas informações serão utilizadas a todo momento para a realização de verificações, como a de se o usuário está logado por exemplo.

Somado a isso, no caso de sucesso o direcionamento à dashboard é importante.

```
SignIn(email: string, password: string) {
    return this.auth.signInWithEmailAndPassword(email, password)
      .then((result) => {
        localStorage.setItem('user', result.user.uid);
        this.router.navigate(['dashboard']);
      })
      .catch((error) => {
        window.alert(error.message);
      });
  }
```

Entretanto, salvar informações importantes no localStorage pode ser uma ação perigosa em contextos de sistemas que requerem segurança extrema. Por isso, a abordagem deve mudar conforme a necessidade, com a utilização de tokens JWT por exemplo. Dessa forma, é possível

salvar todas as informações do usuário de forma segura, ou seja, além de proporcionar segurança ainda facilita acesso aos dados do usuário.

Em sistemas simples, apenas com a finalidade de compreender os conceitos, as informações do usuário podem ser salvas diretamente no localStorage.

Sair

Para sair, remover um usuário da sessão, é usada a função "**signOut()**". Para completar a ação, é importante remover o usuário do localStorage e redirecioná-lo para a tela inicial ou de login.

```
SignOut() {
    return this.auth.signOut().then(() => {
      localStorage.removeItem('user');
      this.router.navigate(['sign-in']);
    });
  }
```

E-mail de verificação

A funcionalidade de enviar um e-mail de verificação para o usuário recém registrado é essencial para garantir segurança e integridade dos membros de seu sistema. Nesse sentido, utiliza-se a função "**sendEmailVerification()**".

Com ela, é possível aplicar diversas abordagens. A mais comum delas consiste em ao registar um usuário no banco de dados, conceder um atributo "verified" marcado como **false**. Enquanto esse valor for falso, o usuário não possui acesso a recursos importantes do sistema. Nesse contexto, esse atributo só será modificado na comprovação de que o e-mail foi confirmado.

```
verifyEmail() {
    return this.auth.currentUser
      .then((u: any) => u.sendEmailVerification())
      .then(() => {
        this.router.navigate(['verify-email-address']);
      });
  }
```

Alterar senha

Na situação do usuário esquecer sua senha, utiliza-se "**sendPasswordResetEmail()**". Nesse caso, apenas ações simples são necessárias, como por exemplo verificar se a string passada como parâmetro tem o formato padrão de e-mails, bem como gerar mensagens na tela.

```
forgotPassword(email: string) {
    return this.auth
      .sendPasswordResetEmail(email)
      .then(() => {
        window.alert('Password reset email sent, check your inbox.');
      })
      .catch((error) => {
        window.alert(error);
      });
  }
```

Persistência, Estado e Usuário atual

Por fim, temos funções adicionais que auxiliam na aplicação das boas práticas:

- "**setPersistence()**": configura a persistência do estado de autenticação no Firebase. Ela determina como e onde o estado do usuário autenticado é armazenado entre as sessões e pode ser configurada para três modos diferentes:
  - Local (por padrão): mantêm a sessão mesmo após o fechamento do navegador;
  - Session: mantêm apenas se o usuário recarregar a página;
  - None: ao recarregar ou fechar o navegador, a sessão é perdida.

- “**authState**”: monitora e emite eventos sobre o estado de autenticação do usuário em tempo real.
- “**currentUser**”: retorna uma “Promise” com o usuário autenticado atual ou null, útil para acessos diretos ao estado de autenticação.

Para mais informações referentes ao sistema de autenticação do Firebase, consulte aqui.

Asimov Jr

Firebase Firestore: Banco de dados

A utilização do Firebase Firestore se mostra o ponto chave da criação deste documento. A fluência na utilização do banco de dados em projetos Desktop é primordial para um bom andamento. O guia apresenta a lógica para manipular os dados do banco, entretanto, vale ressaltar que a **organização** é um elemento central.

Modelar bancos de dados se mostra necessário para um bom funcionamento do sistema, fator que demanda tempo e esforço na fase inicial de um projeto, porém a longo prazo facilitará muito seu andamento.

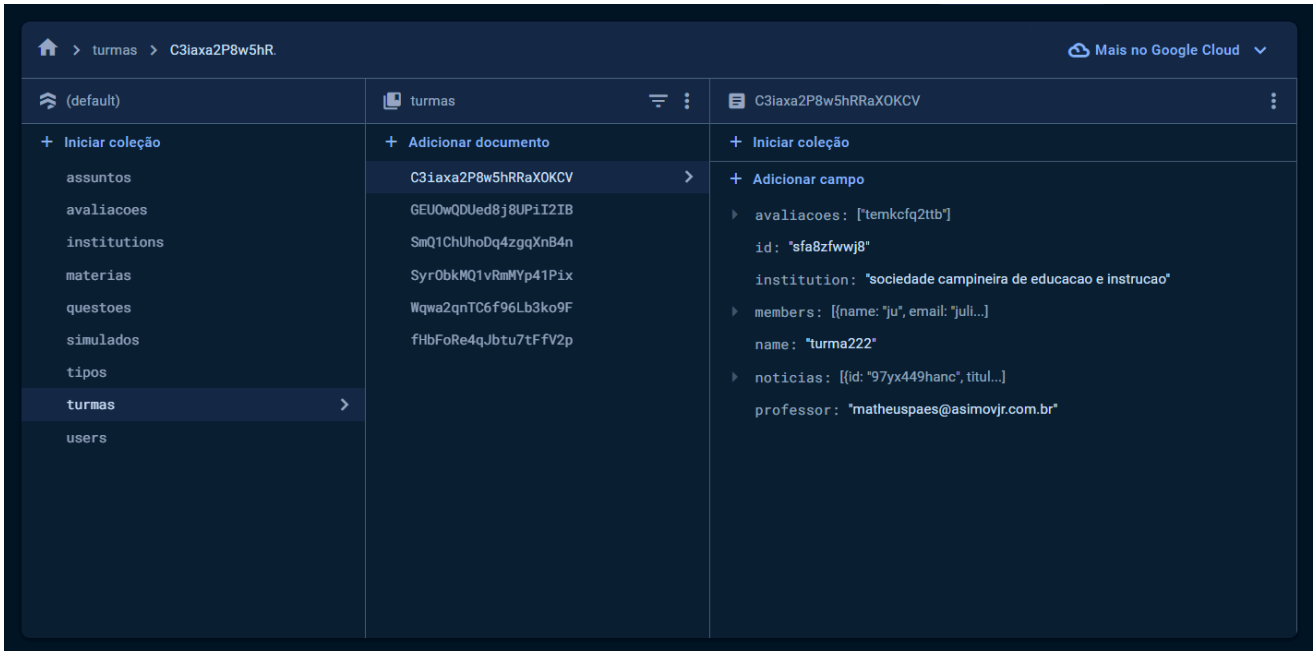
O que é um banco de dados?

O banco de dados basicamente se resume em CRUD:

- Create;
- Read;
- Update;
- Delete;

Com essas quatro ações, é possível estruturar toda a base de dados do seu sistema, implementando lógicas para torná-lo mais robusto. A inserção, consulta, alteração e remoção de dados permite lidar com a maioria das demandas da área de Desktop na Asimov Jr.

A estrutura do Firebase Firestore consiste em coleções, contendo em cada uma delas documentos com “**id**” e seus atributos, sendo ele um banco NoSQL. Aqui temos um exemplo do projeto da Asimov Jr., e-educa:



Nele, é possível observar a magia dos bancos de dados: as relações entre coleções. Nesse contexto, um documento da coleção “turma” possui um vetor de “avaliacoes”, contendo o “**id**” de cada avaliação daquela turma. Com isso, é possível obter todas as informações da avaliação, apenas consultando-a através de seu identificador, tornando a visualização clara e diminuindo a sobrecarga de informações em cada documento.

Antes de qualquer ação referente ao banco, é necessário criar o banco de dados Firestore dentro do Firebase, e selecionar a opção “**Modo de produção**”. Após isso, dirija-se a “Regras” e troque as configurações para o seguinte:

```
rules_version = '2';

service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if true;
    }
  }
}
```

Agora, dentro de seu ambiente Angular, todos os componentes que irão interagir com o banco devem apresentar em seu construtor:

```
constructor(private db: AngularFirestore) { }
```

Pronto! Adiante cada ação será tratada separadamente.

Inserção de dados

A inserção de dados ocorre de forma simples, em que é necessário criar um objeto "**data**", com os atributos desejados, informar a qual coleção ele pertence e adicioná-lo.

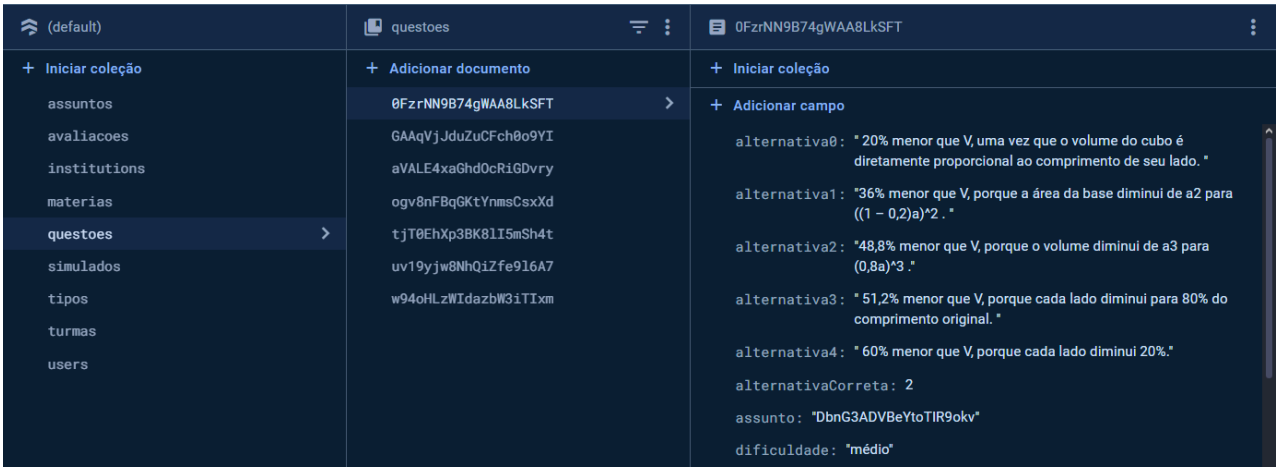
Entretanto, é importante se atentar a realizar verificações. Por exemplo, a criação de duas turmas com o mesmo nome pode gerar conflitos, portanto cabe realizar uma consulta para verificar se aquele nome já está em uso.

Além disso, o Firestore gera um "**id**" automático para cada documento, porém sua utilização atrapalha na praticidade do código (fator que será explicado na sessão "Consultas no banco"), portanto a criação de um "**id**" nos atributos é adequada.

```
addQuestao(){
    const data = {
      enunciado: this.questaoForm.value['enunciado'],
      alternativa0: this.questaoForm.value['alternativa0'],
      alternativa1: this.questaoForm.value['alternativa1'],
      alternativa2: this.questaoForm.value['alternativa2'],
      alternativa3: this.questaoForm.value['alternativa3'],
      alternativa4: this.questaoForm.value['alternativa4'],
      alternativaCorreta: this.respostaCorreta,
      assunto: this.selectedAssunto,
      dificuldade: this.selectedDificuldade,
      profEmail: "",
      id:Math.random().toString(36).substring(2),
    }

    window.alert("Questão cadastrada com sucesso!");
    return this.db.collection('questoes').add(data);
  }
```

Resultado no banco (parte dele):



Consultas no banco

Se tratando da fração mais complexa dos recursos do banco de dados, a consulta pode ser feita de várias maneiras. De maneira geral, sua realização demanda conhecimento das relações entre coleções, ou seja, um estudo da estrutura do banco. Todavia, se essa estrutura se mostra clara e organizada, consultar passa a não ser uma tarefa complexa.

Uma consulta no Firebase Firestore gera um array chamado "**docs**", um vetor com cada documento encontrado. Dentro de cada elemento desse vetor, é possível acessar o "**docs[0].id**" ou "**docs[0].data()**", sendo essa última a responsável por guardar todos os atributos do objeto (**docs[0].data().alternativaCorreta**, por exemplo, sendo essa a alternativa correta da primeira questão encontrada).

Por fim, é essencial salientar que consultas envolvem impor condições ao resultado gerado, assim conseguindo obter exatamente o objeto desejado. As consultas, no geral, envolvem dois cenários, sendo o primeiro deles quando o desenvolvedor tem a ciência de que será retornado apenas um valor:

```
this.db.collection('users/').ref.where('email','==',emailDesejado).get().then((resultado: any)=>{
        this.usuarioDesejado = resultado.docs[0].data();
})
```

Nesse caso, acessamos a coleção usuários e informamos que é desejado apenas o usuário que apresenta o e-mail igual ao "**emailDesejado**". Como sabemos que um e-mail não pode ser utilizado por mais de uma conta, acessamos o resultado da consulta com resultado.docs[0], ou seja, a primeira posição do array. Dessa forma, atribuímos apenas a "**data()**" à variável, justificando a criação de um "**id**" na inserção para tornar o código mais limpo.

No cenário de uma consulta que gera vários resultados, utiliza-se a seguinte abordagem:

```
this.db.collection('users/').ref.where('instituicao','==','unifei').get().then((resultado: any)=>{
        resultado.docs.forEach((aluno: any) => {
                this.alunosUnifei.push(aluno.data());
        })
})
```

Com isso, filtramos apenas os usuários que possuem o atributo instituição como "**unifei**", e adicionamos um a um no vetor de **"alunosUnifei**", através de um **"forEach**".

É importante frisar que consultas compostas acontecerão com frequência, em que é necessário buscar em uma coleção, para obter uma informação e a partir dela realizar outra busca, situação que pode ser complexa. Nesses casos, consulte a documentação ou procure membros da área de Desktop para facilitar o processo.

Remoção e edição de dados

Por fim, temos as operações de remoção e edição dos dados de um documento no Firestore. Nesse sentido, suas abordagens são similares, necessitando de uma busca do elemento desejado e sua posterior remoção ou edição. No cenário da edição temos:

```
this.db.collection('users/').ref.where('institution','==',user.name).get().then((res:any)=>{
        res.docs.forEach(async (u:any)=>{
          let dados = u.data();

          if(dados.name !== dados.institution){
            dados.institution = '';
            await this.db.collection('users/').doc(u.id).update(dados);
          }
        })
      })
```

Na imagem, observamos o seguinte funcionamento:

1. **Consulta de Usuários**: O código começa consultando a coleção users no Firestore, especificamente buscando documentos onde o campo institution é igual a "**unifei**".
2. **Extração e Modificação dos Dados**: Para cada documento (u), o código extrai os dados usando let dados = u.data(). Esses dados são armazenados na variável dados. Em seguida, o código remove o valor do atributo institution dos dados com dados.institution = ''.
3. **Atualização no Firestore**: Após modificar os dados, o código utiliza "**update(dados)**"; para atualizar o documento no Firestore. Note que a utilização de u.id representa o ID único gerado automaticamente pelo Firestore para cada documento, não o "**id**" gerado manualmente, uma vez que dessa forma é possível fazer o acesso direto.

Finalmente, o cenário da remoção é bem similar:

```
this.db.collection('users/').ref.where('institution','==','unifei').get().then((res:any)=>{
    res.docs.forEach(async (u:any)=>{
          await this.db.collection('users/').doc(u.id).delete().
    })
})
```

Observação: a utilização do async/await representa a obrigação de um sincronismo nas ações realizadas, ou seja, a remoção deve ser obrigatoriamente concluída antes de seguir ao próximo documento. Para mais informações consulte a documentação.

Asimov Jr

Conclusão

O guia abordou de maneira aplicada e breve as funcionalidades essenciais do Firebase Firestore e do sistema de autenticação do Firebase, dois pilares fundamentais para o desenvolvimento de projetos Desktop na Asimov Jr. Ao longo do documento, destacamos a integração inicial com o Angular, desde configurações básicas até as práticas recomendadas para autenticação de usuários e manipulação de dados no Firestore.

Em suma, este guia não apenas oferece uma introdução prática ao Firebase para Desktop, mas também encoraja os desenvolvedores a explorar mais a fundo as capacidades da plataforma, adaptando-as às necessidades específicas de cada projeto. Para informações detalhadas e atualizações, recomenda-se consultar a documentação oficial do Firebase e estar atento às melhores práticas de desenvolvimento de software.

Asimov Jr