sãojudas universidade	FTCE Faculdade de Tecnologia e Ciências Exatas

USJT - 2019 – Programação para WEB e Dispositivos Móveis

Professores: Anderson e Bossini

Aula: 11
Assunto: Ionic – CRUD no Firebase

Neste material passaremos a realizar as operações de acesso a dados em uma base remota que faz parte do serviço oferecido pelo Google denominado Firebase. A base que utilizaremos se chama **Firebase Realtime Database**. Trata-se de uma base não relacional em que os dados são armazenados como um único objeto JSON.

Passo 1 (Adicionando o Firebase ao projeto) Para usar o Firebase, será necessário criar uma conta do Google (um Gmail comum). Com um Gmail em mãos, acesse o Link 1.1.

Link 1.1
https://firebase.google.com

1.1 Faça login e clique em Ir para o Console, no canto superior direito da tela.

1.2 A seguir, clique em Criar um Projeto e dê um nome apropriado para ele, que te ajude a lembrar a que ele se refere no futuro. Um projeto nada mais é do que um grupo de configurações a serem aplicadas a determinadas aplicações que forem vinculadas a ele.

1.3 Depois de ter criado o projeto, na tela de boas vindas, você verá links que sugerem que você adicione seu aplicativo ao projeto. Clique no ícone referente a aplicações WEB, como mostra a Figura 1.3.1.

Figura 1.3.1

1.4 Na tela a seguir, dê um apelido para o seu projeto. Não é necessário configurar o Firebase Hosting nesse momento. Note que, ao final, você receberá um script Javascript com a definição de um objeto JSON chamado firebaseConfig. Em breve precisaremos desse objeto.

1.5 Agora vamos utilizar o Node Package Manager para instalar o Firebase e a biblioteca do Angular que permite utilizá-lo em Typescript. Para tal, use o Comando 1.5.1.

Comando 1.5.1
npm install firebase@3.9.0 angularfire2@4.0.0-rc.0 –save

Inspecione seu arquivo package.json e verifique que ambas as dependências foram adicionadas.

1.6 Abra o arquivo **app.module.ts** e adicione o conteúdo da Listagem 1.6.1. Note que você deve utilizar seu objeto de configuração do Firebase, descrito no Passo 1.4.

Listagem 1.6.1

```
import { AngularFireModule } from 'angularfire2'

import { AngularFireDatabaseModule } from 'angularfire2/database'



//fora da definição da classe, logo após os imports

export const firebaseConfig = seu objeto JSON aqui;

//adicione as duas linhas em destaque à seção imports

imports: [

BrowserModule,

IonicModule.forRoot(MyApp),

AngularFireModule.initializeApp(firebaseConfig),

AngularFireDatabaseModule

],
```

1.7 A fim de garantir que o Firebase foi adicionado corretamente à aplicação faremos o seguinte teste.

1.7.1 Importar AngularFirebaseDatabase no arquivo **home.ts.**
1.7.2 Injetar uma dependência do tipo AngularFirebaseDatabase no construtor da classe HomePage, chamada db.

1.7.3 Usar o comando console.log (db);, ainda no construtor. Se tudo estiver correto, no log do navegador deveremos ver a representação textual do objeto db exibida, incluindo suas propriedades.

Veja o código na Listagem 1.7.1.

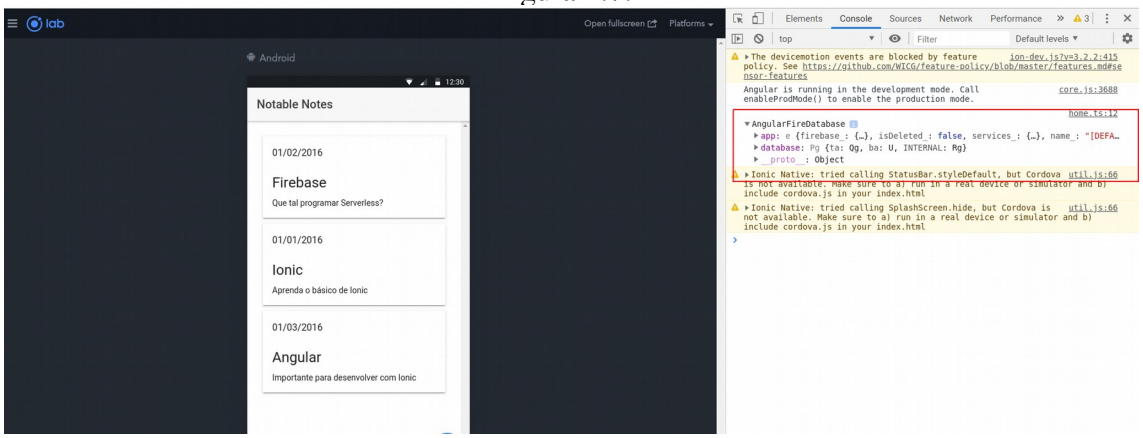
Listagem 1.7.1

```
import {AngularFireDatabase} from 'angularfire2/database'

constructor(public navCtrl: NavController, private noteService: NoteService, db:
AngularFireDatabase) {
   console.log(db);
   this.notes = noteService.notes;
  }
```

Certifique-se de que o servidor (ionic serve --lab) está em execução, abra o console do seu navegador e veja o resultado. Se necessário, reinicie o servidor. O resultado deverá ser parecido com o que mostra a Figura 1.7.1.

Figura 1.7.1



Passo 2 (Explorando a base de dados remota) Visite o Link 2.1 para acessar a base de dados remota do Firebase.

Link 2.1
https://console.firebase.google.com

2.1 No canto superior esquerdo, clique em Desenvolver e então escolha Database. Desça a página até encontrar a opção Criar banco de dados do **Realtime Database** (não use o Cloud Firestore).

2.2 Escolha iniciar em modo de teste, o que permitirá que sua base seja acessada por qualquer um que possa seu link. É claro que em produção isso não poderá ser mantido assim. Porém, para os testes iniciais, é suficiente.

2.3 Observe que temos um único objeto JSON que representa a base inteira. É possível adicionar novos elementos clicando no botão +. Interaja com o ambiente para se familiarizar com ele.

Passo 3 (Implementando a funcionalidade de adição de nova nota) A classe responsável por interagir com o Firebase será a NoteService.

3.1 O primeiro passo é importar e injetar o Firebase na classe NoteService. Veja a Listagem 3.1.1.

Listagem 3.1.1

```
import {AngularFireDatabase} from 'angularfire2/database'
export class NoteService{
   constructor (private db: AngularFireDatabase){

   }
```

3.2 A nova implementação do método addNote é exibida na Listagem 3.2.1. Note que cada nota ficará vinculada a uma única chave da base chamada notes. O Firebase se encarrega de gerar um identificador único para cada nota.

Listagem 3.2.1

```
addNote (note){
      //this.notes.push(note);
      this.db.list("/notes/").push({
         title: note.title,
         content: note.content,
         date: note.date
      });
   }
```

3.3 Teste a aplicação, adicionando uma nova nota. Abra o Console do Firebase para checar se ela foi, de fato, criada por lá.

Passo 4 (Buscando novas notas) A fim de listar as notas existentes no Firebase, iremos implementar um método que devolve a lista de objetos associadas à chave notes na base remota.

4.1 Na classe NoteService, implemente o método da Listagem 4.1.1. Ele devolve o vetor de notas existentes na base remota.

Listagem 4.1.1

```
fetchNotes (){
      return this.db.list ("/notes/");
}
```

4.2 A classe HomePage é responsável pela exibição das notas. Assim, vamos definir o método **ngOnInit** e chamar o método fetchNotes, configurando sua propriedade notes com o valor devolvido por ele. O método ngOnInit é chamado automaticamente quando uma página é carregada. Note que também é preciso deixar de utilizar a propriedade notes diretamente, como fazíamos no construtor de HomePage. Veja a Listagem 4.2.1.

Listagem 4.2.1

```
constructor(public navCtrl: NavController, private noteService: NoteService, db:
AngularFireDatabase) {
   //console.log(db);
   //this.notes = noteService.notes;
  }
ngOnInit (){
   this.notes = this.noteService.fetchNotes();
}
```

4.3 Além disso, é preciso usar o pipe "**async**" já que os dados podem ser baixados em ordem arbitrária e também por não querermos bloquear a interface com o usuário. Veja a Listagem 4.3.1. A alteração deve ser feita no arquivo home.html.

Listagem 4.3.1

```
<ion-card *ngFor="let note of notes | async" (click)="onItemClick(note)">
```

4.4 Execute a aplicação e verifique se as notas existentes na base do Firebase são exibidas na lista da Home Page. Faça inserções por meio do Console do Firebase e verifique o resultado na aplicação. Ela é notificada imediamente. Faça inserções por meio da aplicação e verifique no Console. O mesmo é verdade por lá.

Passo 5 (Editando uma nota) A edição de uma nota se dá por meio de sua chave, que é gerada automaticamente pelo Firebase. A sintaxe **$key** nos permite acessar a chave de um objeto salvo no Firebase.

5.1 Implemente o método da Listagem 5.1.1 na classe **NoteService**. Note que usamos o método **object** do objeto **db** para fazer referência a um objeto específico. O objeto referenciado é aquele cuja chave é especificada no parâmetro do método object. O método **update** recebe um JSON com valores que desejamos utilizar para editar os valores existentes no objeto sendo editado.

Listagem 5.1.1

```
editNote (note){
      this.db.object("/notes/"+note.$key).update({
         title: note.title,
         content: note.content,
         date: note.date
      });
}
```

5.2 Vamos usar o método editNote no método ionViewWillLeave de DetailPage. Veja a Listagem 5.2.1.

Listagem 5.2.1

```
ionViewWillLeave (){
   if (this.newNoteFlag){
     if (this.note.title != "" && this.note.content != "" && this.note.date != "")
       this.noteService.addNote(this.note);
```

```
  }
  else{
    this.noteService.editNote(this.note);
  }
 }
```

5.3 Teste novamente a aplicação, dessa vez fazendo atualizações em notas por meio da aplicação e veja se o resultado aparece no Console do Firebase.

Passo 6 (Apagando uma nota) Para remover uma nota da base também precisamos utilizar sua chave.

6.1 Implemente o método da Listagem 6.1.1 na classe NoteService. Seu funcionamento é análogo ao método de edição de notas. Note que registramos duas arrow functions usando as funções then e catch. A primeira será executada se a nota for apagada com sucesso. A segunda será executada somente se houver uma exceção durante a operação.

Listagem 6.1.1

```
removeNote (note){
      this.db.object("/notes/"+note.$key).remove()
      .then(
        x => console.log ("Note deleted successfully")
      ).
      catch( error => {
        console.log ("Could not delete note");
        alert ("Could not delete note")
      });
  }
```

6.2 Note que ainda há um problema com a aplicação. Quando clicamos no ícone para apagar uma nota, ela é, de fato, removida da base remota. Porém, logo a seguir o método ionViewWillLeave é chamado o que implica na chamada do método editNote também. O método editNote chama o método update, o que faz com que a nota seja inserida novamente. Esse é seu comportamento padrão. Assim, precisamos de algum mecanismo para diferenciar uma operação de remoção de uma operação de edição. Iremos utilizar uma flag para isso. Ela indicará se uma nota está sendo removida ou não. Veja a Listagem 6.2.1.

Listagem 6.2.1

```
//na classe DetailPage
 deleteNoteFlag = false;
//handler do botão que confirma a remoção da nota
       handler: () =>{
         this.deleteNoteFlag = true;
         this.noteService.removeNote(this.note);
         this.navCtrl.pop();
       }
ionViewWillLeave (){
```

```
  if (this.newNoteFlag){
    if (this.note.title != "" && this.note.content != "" && this.note.date != "")
      this.noteService.addNote(this.note);
  }
  else if (!this.deleteNoteFlag){
    //somente editamos se não for uma remoção
    this.noteService.editNote(this.note);
  }
}
```

6.3 Teste novamente a aplicação e certifique-se de que as quatro operações de manipulação de dados estão funcionando.

Exercício para avaliação

Ajuste o chat para usar o Firebase. Sua aplicação deve ter as seguintes características.

- Cada sala é representada por um objeto JSON com seu nome. Ou seja, há as chaves cinema, esportes e diversos na raiz do JSON principal do Firebase.

- Não são permitidos usuários com nome repetido em uma determinada sala.

- Cada mensagem enviada é armazenada como um objeto JSON, filho do JSON referente à sala em que ela foi enviada.

- Cada mensagem tem uma data de envio. Cada sala mostra as mensagens como uma pilha: a última enviada é a primeira a ser exibida.

Suba a solução no Github.