Registrieren und Login mit Ionic5 und Datenbankzugriff mittels API

Inhalt:

1. Funktion in registrieren.page.ts erstellen
2. Funktion in apiService erstellen mit HTTP-POST-Request zum API und zur Datenbank
3. Formular in der „registrieren.page.html“ anpassen
4. testen

Ziel: registrieren

- Datenbank mit Tabelle „users“ ist vorhanden
- apiService und app.modules.ts mit http-Modul bereits vorhanden
- apiService – neue Funktion „signupApi“ für einen POST-request erstellen
- registrieren.html anpassen – [(ngModel)] verwenden
- Daten aus dem Formular in die Datenbank schreiben

1)registrieren.ts

Öffne die „registrieren.page.ts“.

Erstelle zuerst im constructor die Methode „apiService“ und nutzte die Möglichkeit, den nötigen IMPORT somit automatisch erstellen zu lassen.

```
14    constructor(
15      private router: Router,
16      public apiService: ApiService) { }
```

```
src > app > pages > registrieren > TS registrieren.page.ts > ...
1   import { Component, OnInit } from '@angular/core';
2   import { ApiService } from 'src/app/api.service';
3   import { Router } from '@angular/router';
```

- export Klasse erstellen
  Erstelle eine Auflistung aller Elemente, genauso wie sie auch als Reihenfolge in der Datenbank bei der Tabelle „users“ vorkommen. Als Werte sollen alle als „any“ vorbereitet werden.

```
10  export class RegistrierenPage implements OnInit {
11    vorname: any;
12    nachname: any;
13    email: any;
14    password: any;
15    telefon: any;
16    strasse: any;
17    plz: any;
18    ort: any;
19
20    constructor(
```

- Funktion „signup“ erstellen

```
24    ngOnInit() {
25    }
26
27  signup() { }
```

Darin die konstante Variable „userData“, welche die eingegeben Daten aus dem Formular (in registrieren.page.html) übernimmt.

```
27  signup() {
28    const userData = {
29      vorname: this.vorname,
30      nachname: this.nachname,
31      email: this.email,
32      password: this.password,
33      telefon: this.telefon,
34      strasse: this.strasse,
35      plz: this.plz,
36      ort: this.ort,
37    };
```

Nun folgt die Verknüpfung zur Funktion im API:

this.apiService.signupApi(userData).subscribe((res: any) => { });

In dieser Funktion

- Erfolgt die Verbindung zum Service mit dem Namen „signupApi“ – der Name ist sehr ähnlich dieser Funktion, die gerade erstellt wird, aber absichtlich mit der Erweiterung Api, damit man sie besser unterscheiden kann.
- Diese hat Zugriff auf die userData
- Dann das Observable mit dem subscribe

```
37    };
38  this.apiService.signupApi(userData).subscribe((res: any) => {
39    console.log('SUCCESS ===',res);
40    this.router.navigate(['/login']);
41  });
42  }}
```

Das Console.log dient nur zur internen Info, wenn man nachschauen will, was in die Console geschrieben wird. Diese ruft man auf mit der Taste „F12“.

Damit nach dem Registrieren automatisch die LOGIN-Seite geöffnet wird – was ja sinnvoll ist, weil man sich gleich anmelden kann – soll diese am Ende der Funktion „signup()“ aufgerufen werden.

Dafür füge diese Router-Navigation ein.

this.router.navigate(['login']);

```
      this.responseData = result;
      localStorage.setItem('userData', JSON.stringify(this.responseData));
      this.router.navigate(['login']);
      });
  }
```

Damit diese Seite aber überhaupt mit dieser Route umgehen kann, muss man diese auch noch

- IMPORTIEREN

```
import { ApiService } from 'src/app/api.service';
import { Router } from '@angular/router';

```

- und im constructor bekannt machen:

```
  constructor(
    private router: Router,
    public apiService: ApiService) { }

```

2)apiService – Funktion zum API erstellen

Da bereits das http-client-modul schon aus der vorher erstellten Abfrage der Berater vorhanden ist, kann man sich hier auf die Funktion konzentrieren.

Erstelle unterhalb von der letzten anderen Funktion (beratersApi) die neue mit dem passenden Namen „signupApi“, weil

- signup der englische Ausdruck für registrieren ist
- das Api in CamelCase-Schreibweise dazu gehängt, damit es sich von der Funktion „signup“ in der registrieren.page.ts unterscheidet

```
signupApi(userData){
  return this.http.post('http://localhost/apiionic/public/index.php/signup',userData);
}

```

Die Funktion hat die Parameter in der Klammer mit „userData“, welche in der registrieren.page.ts erzeugt wurden bzw. durch das Formular erstellt werden.

Es handelt sich hier um eine POST Abfrage, die Daten in die Datenbank SCHREIBT, zum Unterschied von einer GET Abfrage, die nur lesen kann.

Die URL leitet auf die API hin. In diesem Fall liegt sie am Laufwer C: im Ordner xampp/htdocs.

In der index.php der API liegt die Funktion „signup“ – absichtlich ähnlicher Name, so wie wir es hier auch verwendet haben bei den Funktionen. Diese schickt mit PHP-Code (INSERT INTO) die Daten in die Datenbank.

Nach einem Beistrich muss noch das Array „userData“ übergeben werden, damit es weitergeleitet werden kann.

3)Formular in der „registrieren.page.html“ anpassen

Hier sollen nun die Eingaben des Nutzers weitergereicht werden. Dies passiert mit

[(ngModel)]=“name“ ….mit der jeweiligen Bezeichnung:

```
15      <ion-item>
16        <ion-label position="floating">Vorname</ion-label>
17        <ion-input type="text" [(ngModel)]="vorname"></ion-input>
18      </ion-item>
```

Hiermit wird die Klasse aus „registrieren.page.ts“ ab Zeile 11 verarbeitet:

```
10  export class RegistrierenPage implements OnInit {
11    vorname: any;
12    nachname: any;
13    email: any;
```

Wende die auf alle Input-Felder an.

```
16        <ion-label position="floating">Vorname</ion-label>
17        <ion-input type="text" [(ngModel)]="vorname"></ion-input>
18      </ion-item>
19      <ion-item>
20        <ion-label position="floating">Nachname</ion-label>
21        <ion-input type="text" [(ngModel)]="nachname"></ion-input>
22      </ion-item>
23      <ion-item>
24        <ion-label position="floating">E-Mail</ion-label>
25        <ion-input type="email" [(ngModel)]="email"></ion-input>
26      </ion-item>
27      <ion-item>
28        <ion-label position="floating">Passwort</ion-label>
29        <ion-input type="password" [(ngModel)]="password"></ion-input>
30      </ion-item>
31      <ion-item>
32        <ion-label position="floating">Telefon</ion-label>
33        <ion-input type="tel" [(ngModel)]="telefon"></ion-input>
```

Button „Registrieren“ aktivieren

Da in der „registrieren.page.ts“ bereits die Funktion „signup()“ erstellt wurde, kann diese nun mit einer Click-Funktion aufgerufen werden.

Füge daher in den Button diese ein:

```
70  </ion-card>
71  <ion-button size="full" color="light" (click)="signup()">Registrieren</ion-
72  </ion-content>
```

4)Testen

Öffne die Seite

- im internen Terminal von VisualStudioCode mit „ionic serve“
- xampp muss laufen wegen dem API

Registriere einen Kunden.

Ergebnis in der Datenbank:

id	vorname	nachname	email	password	telefon	strasse	plz	ort
1	Kevin	Eckelhart	kevin@gma[illegible].com	1234	664555555	Bergstrasse 1	2130	Mistelbach
2	Raphael	Idinger	raphi@gmail.[illegible]	1234	6641234	Hauptstrasse 1	2130	Mistelbach
4	Josef2	Berger2	berger@aon.at	81dc9bdb52d04dc20036dbd8313ed055	6666	Brennerweg 8	2130	Mistelbach
5	Franz	Berger	franz@gmx.at	81dc9bdb52d04dc20036dbd8313ed055	664123456	Brennerweg 8	2130	Mistelbach

Da im API das Passwort verschlüsselt wird, kommt es auch so hier an.

5)Login Abfrage erstellen