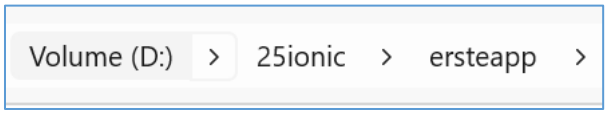
Ionic - Login und Datenbankzugriff mittels API



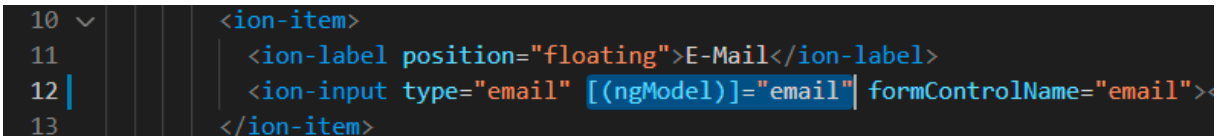
Inhalt:

1. Login.page.html (frontend) anpassen
2. Funktion in login.page.ts erstellen
3. Funktion in apiService erstellen mit HTTP-POST-Request zum API und zur Datenbank

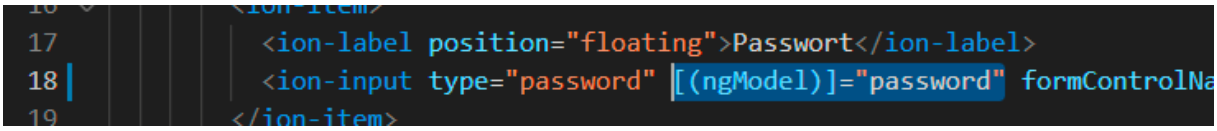
Ziel:
Login mit Überprüfung in die Datenbank, ob der Benutzer (E-Mail und Passwort) vorhanden ist. Dabei wird mit Hilfe eines Services auf den **Endpunkt einer URL** in einer entfernt liegenden API zugegriffen, die wiederum mit der Datenbank verknüpft ist und die „echte“ PHP-Abfrage erstellt. Das API gibt dann die Antwort als JSON zurück, der gelesen wird und in der login.page.ts dann entschieden wird, ob ein „success“ mit „true“ vorliegt. Dann wird auf eine Seite weitergeleitet, ansonsten nicht.

1)login.page.html anpassen

Öffne die Datei und erstelle bei den beiden Inputfeldern ein „ngModel“ mit dem passenden Namen:




```
<ion-item>
 <ion-label position="floating">E-Mail</ion-label>
 <ion-input type="email" [(ngModel)]="email"></ion-input>
</ion-item>
```




```
<ion-item>
  <ion-label position="floating">Passwort</ion-label>
  <ion-input type="password" [(ngModel)]="password"></ion-input>
</ion-item>
```

Nur dadurch kann in die „login.page.ts“ auch diese beiden Inhalte übernommen und verwendet werden.

2)login.page.ts anpassen

Öffne die „login.page.ts“.

Erstelle im „constructor“ zwei neue Methoden, die mit geschicktem Vorgehen auch gleich automatische den Import erzeugen:

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

```
1  import { Component, OnInit } from '@angular/core';
2  import { Router } from '@angular/router';
3  import { ApiService } from 'src/app/api.service';
4
```

In der Klasse werden die beiden Elemente angelegt, die wir in der Datenbank dann vergleichen, an, aber ohne Inhalt, also leer.

```
11  export class LoginPage implements OnInit {
12    email = '';
13    password = '';
14
```

email = '';
password = '';

Unterhalb von der vorhandenen Funktion „registrieren()“ wird nun die bereits vorhandene aber leere Funktion „login()“ befüllt:

- Es wird, anders als die „registrieren“ nicht mit einer Observablen, sondern mit einem „Promise“ erstellt.
- In dessen „body“ werden die beiden Daten der Kundeneingabe geschrieben.
- Darunter wird die richtige Funktion geschrieben
    - Das apiService mit dem Namen „loginApi“ wird dann noch erstellt, aber hier schon miteinbezogen
    - Es nimmt den eben erstellten „body“.
    - Die „if“-Abfrage hat die Bedingung, ob das vom API gelieferte Resultat einen Fehler erzeugt oder nicht. Liegt kein Fehler vor, dann soll auf eine Seite weiter „geroutet“ werden.
    - Hier wird, weil es einfach ist und noch nicht viele Seiten in dem Beispiel vorhanden sind, auf die Seite „Team“ weitergeleitet werden.

```
  registrieren(){
    this.router.navigate(['/registrieren']);
  }

  login() {
    return new Promise((_resolve) => {
      const body = {
        email: this.email,
        password: this.password
      };
```

```

    this.apiService.loginApi(body).subscribe((res: any) => {
     if (res.error !== true) {
       this.router.navigate(['/team']);
       };
     });
    });
  }
  }

```

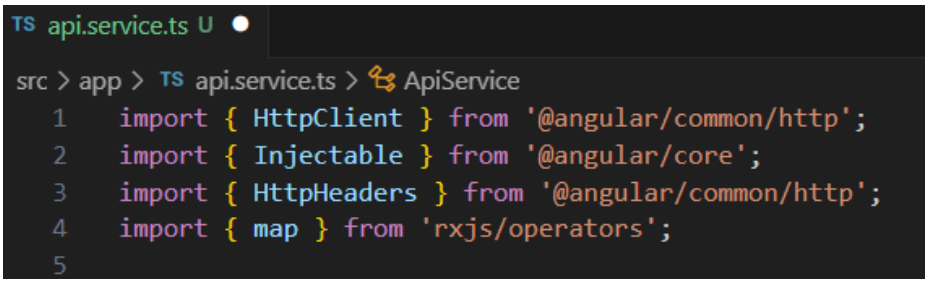
```
login() {
   return new Promise((_resolve) => {
     const body = {
       email: this.email,
       password: this.password
     };

  this.apiService.loginApi(body).subscribe((res: any) => {
   if (res.error !== true) {
     this.router.navigate(['/team']);
     };
   });
  });
}
```

3)das Service anpassen

Öffne die „api.service.ts“.

Da aus früheren Aufgaben der Import von „HttpClient“ schon vorhanden ist, muss man nur 2 neue durchführen:

- HttpHeader
- rxjs – den man dann für die Funktion „map()“ benötigt.



import { HttpHeaders } from '@angular/common/http';
import { map } from 'rxjs/operators';

Unterhalb der letzten vorhandenen Api, oder auch drüber, das ist egal wo man jetzt diese neue „loginApi“ hinsetzt.

Man benötigt, bevor die Funktion mit „return“ erstellt wird zwei Variablen, die in der neuen Ionic5 Version nicht als „let“ sondern als „const“ erstellt werden:

- const headers
- const httpOptions


```
loginApi(userdata: { email: string; password: string; }) {
  const headers = new HttpHeaders({
    'Content-Type': 'application/json; charset=UTF-8'
  });
  const httpOptions = {
    headers: new HttpHeaders({
      'Content-Type': 'application/json',
    })
  };
```

Eventuell nötig:

Wenn man den „Content-Type“ angibt, ist Ionic nicht zufrieden und mit Hilfe der Schnellkorrektur“ – also mit der Maus drüberstehen und dann beim erscheinenden Fenster auf „Schnellkorrektur“ – wird die Zeile darüber automatisch erstellt und die Fehlerwarnung verschwindet.

Darunter kommt nun die große „return“ Funktion. Diese ist prinzipiell so aufgebaut wie die anderen auch (z.B. signupApi).
Es wird ein POST-Request zu einer bestimmten URL erstellt, mit einem JSON der die übergebene E-Mail und das Passwort enthält. Dann wird in der API alles verglichen und das Ergebnis dieser Antwort ist wieder ein JSON. In der „login.page.ts“ wird dann die überprüft, ob die Antwort „res.success“ true oder false ist (If-Abfrage). Je nachdem wird weitergeleitet oder nicht.

- post – request: eine LOGIN ist immer eine POST Anfrage und keine GET, da ja auch Daten hingesendet werden und nicht nur geholt.
- Der Endpunkt wird in der URL wieder auf die API gelegt, wobei in der dortigen „index.php“ der genaue Endpunkt und der gesamte Code dafür im Bereich „login“ liegt.
- JSON.stringify wird immer genutzt für Daten-Austausch mit einem Webserver. Unsere API gibt nämlich auch JSON-Daten aus. Nähere Info dazu gibt die Website w3schools: https://www.w3schools.com/js/js_json_stringify.asp
  Diese hat das gleiche Argument wie oben der Start der „loginApi“ nämlich „userData“, was aber namentlich frei ist, nur diese beiden müssen das selbe enthalten. Ich habe halt den gleichen Begriff genommen, wie oben bei „signupApi“.
- httpOptions
- map(): Die map-Funktion kommt von RxJS und updated den Wert vom erhaltenen (return) Wert.

```
29      };
30
31      return this.http.post(('https://localhost/22apiionic/public/index.php/login'),
32      JSON.stringify(userdata), httpOptions).pipe(map(res=>res));
33
34  }
```

Code:

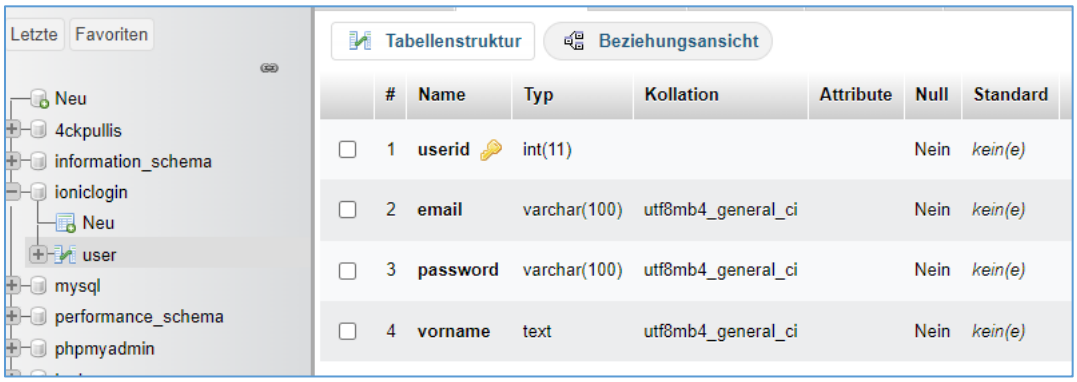
```
return this.http.post(('https://localhost/22apiionic/public/index.php/login'),
  JSON.stringify(userdata), httpOptions).pipe(map(res=>res));
```

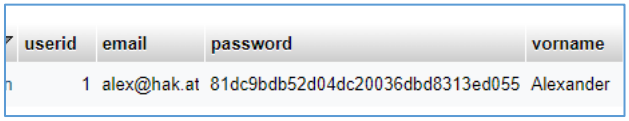
Kontrolle:

Datenbankverbindung in der API: src/config/db.php:

```
C: > xampp > htdocs > 22apiionic > src > config > db.php
1   <?php
2   /**
3    * Connect MySQL with PDO class
4    */
5   class db {
6
7     private $dbhost = 'localhost';
8     private $dbuser = 'root';
9     private $dbpass = '';
10    private $dbname = 'ioniclogin';
11
12    public function connect() {
```

Datenbank:





PSW: 1234

Quelle:

Abgeändert, aber prinzipiell von: https://www.youtube.com/watch?v=Yd9gEtTybxs

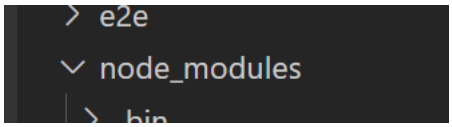
Jetzt nicht mehr: PROBLEM mit „.map“ (bei der Version 6.6.)

1.Lösung:

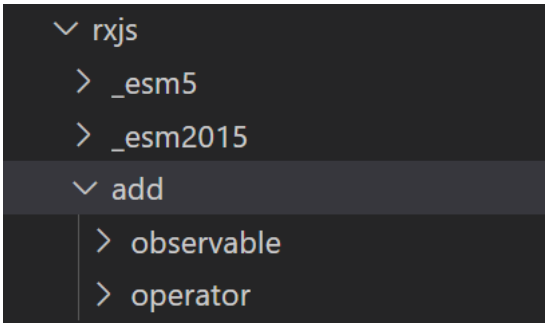
Problem – in neuen Installation wird im Service für das Login die „.map()“ unterstrichen:

```
26   return this.http.post(('https://votech.dig
27     JSON.stringify(userData), httpOptions)
28     .map(res => res);
```

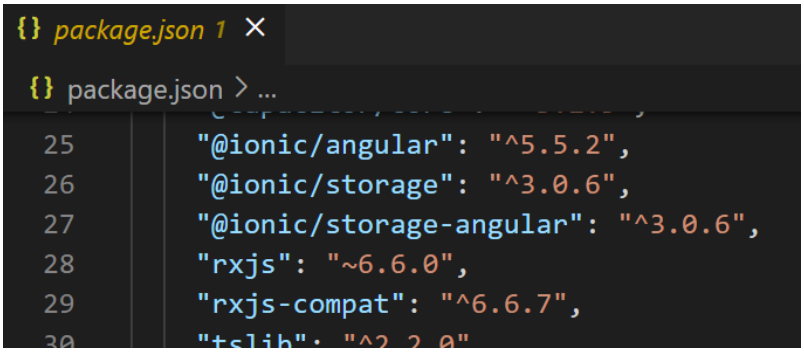
RXJS in der Version von 7.x.x funktioniert nicht.

Im Ordner 

node_modules gibt es den Ordner „rxjs“ wo der Ordner „add“ fehlt und somit der Ordner „observable“

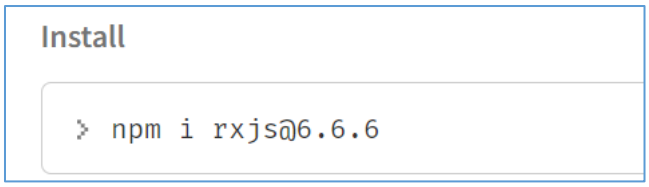


Kontrolle in der „package.json“: Es passt die Version „6.6.0“ - ab der Version 7 gibt es nämlich die Probleme



Lösung:

https://www.npmjs.com/package/rxjs/v/6.6.6

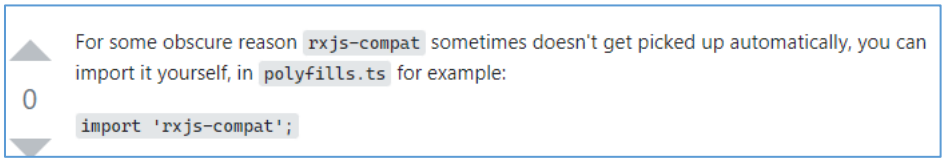


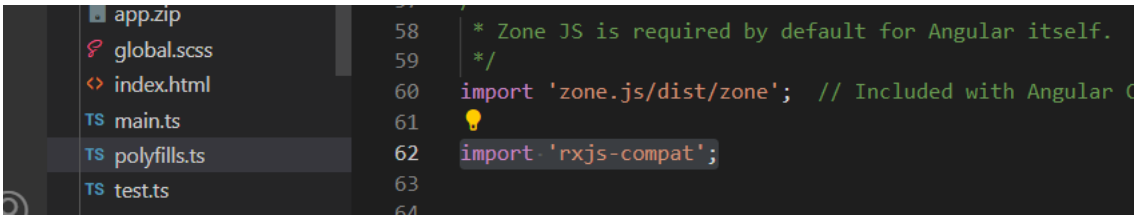
Im Terminal diese Installation durchführen, welche die falsche Version überschreibt.

npm i rxjs@6.6.0

2.Lösung

Wenn ein Problem auftritt, dass das „.map“ hier rot unterwellt wird, dann heißt das, dass es nicht akzeptiert wird. Das kann man lösen. Es ist nämlich manchmal möglich, dass das oben mittels „npm“ eingebaute „rxjs“-Element nicht anläuft, dann muss man folgendes in der Datei „polyfills.ts“ eingreifen:




app.zip
global.scss
index.html
main.ts
polyfills.ts
test.ts
58 * Zone JS is required by default for Angular itself.
59 */
60 import 'zone.js/dist/zone'; // Included with Angular
61
62 import 'rxjs-compat';
63