# GET-Request mit Ionic5 und Datenbankzugriff mittels API

Inhalt:

1. Datenbank anlegen
2. http-Modul für Datenbank-Verbindung
3. Service-Provider erstellen
4. NUR zur INFO – zu URL und API
5. Daten in der Seite „team.html“ anzeigen lassen
    a. *ngFor – Schleife
    b. Interpolation mit {{ }}

**Ziel:**

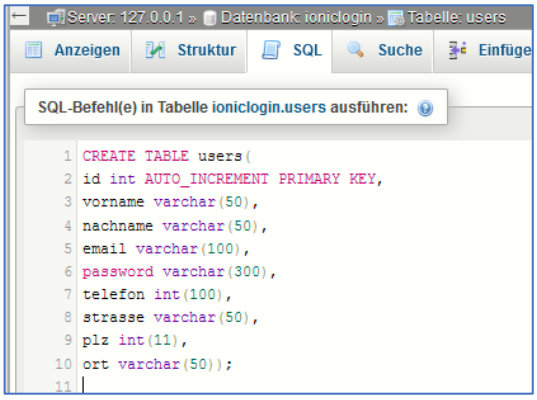
- Zugriff mittels Ionic auf Daten per GET-Request über eine API.
- Die API liegt „irgendwo“ außerhalb der App – hier in Localhost.
- Verwendung eines „services“, welches die Verbindung zur API herstellt
- Daten in der Seite „team.html“ anzeigen lassen

## 1)Datenbank anlegen

Starte Xampp, öffne „phpMyAdmin“ und lege eine neue Datenbank an mit dem Namen „ioniclogin“.


Danach verwende diesen SQL-Code für das Anlegen der Tabelle „users“, damit man es nicht händisch anlegen muss. Klicke auf den Reiter „SQL“ und gib folgenden Code ein, damit es schneller geht:

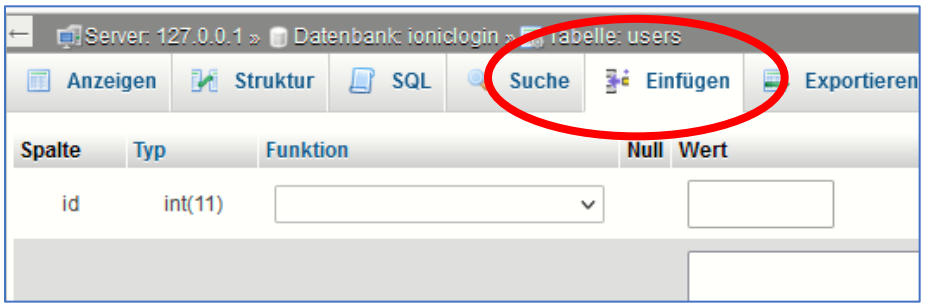
```
CREATE TABLE users(
id int AUTO_INCREMENT PRIMARY KEY,
vorname varchar(50),
nachname varchar(50),
email varchar(100),
password varchar(300),
telefon int(100),
strasse varchar(50),
plz int(11),
ort varchar(50));
```



Ergebnis:

| # | Name | Typ | Kollation | Attribute | Null | Standard | Kommentare | Extra | Aktion |
|---|---|---|---|---|---|---|---|---|---|
| 1 | id | int(11) | | | Nein | *kein(e)* | | AUTO_INCREMENT | Bearb |
| 2 | vorname | text | utf8mb4_general_ci | | Nein | *kein(e)* | | | Bearb |
| 3 | nachname | varchar(50) | utf8mb4_general_ci | | Ja | *NULL* | | | Bearb |
| 4 | email | varchar(100) | utf8mb4_general_ci | | Nein | *kein(e)* | | | Bearb |
| 5 | password | varchar(300) | utf8mb4_general_ci | | Ja | *NULL* | | | Bearb |
| 6 | telefon | int(100) | | | Ja | *NULL* | | | Bearb |
| 7 | strasse | varchar(50) | utf8mb4_general_ci | | Ja | *NULL* | | | Bearb |
| 8 | plz | int(11) | | | Nein | *kein(e)* | | | Bearb |
| 9 | ort | varchar(50) | utf8mb4_general_ci | | Nein | *kein(e)* | | | Bearb |

Erstelle zwei User:
Nutze dazu den Button „Einfügen“:




Ergebnis:

| | id | vorname | nachname | email | password | telefon | strasse | plz | ort |
|---|---|---|---|---|---|---|---|---|---|
| hen | 1 | Kevin | Eckelhart | kevin@gmail.com | 1234 | 664555555 | Bergstrasse 1 | 2130 | Mistelbach |
| hen | 2 | Raphael | Idinger | raphi@gmail.com | 1234 | 6641234 | Hauptstrasse 1 | 2130 | Mistelbach |

# 2)http-Modul für Datenbank-Verbindung

**Theorie:**
Das HttpClient Modul benötigt man, um http **POST, GET, PUT und DELETE requests** machen zu können.

API calls, die das http-Client-Modul nutzen sind asychron. Dabei wird mit modernen JavaScript API-Elementen gearbeitet, nämlich

- Promises
- Observables

Promises
Ein „promise“ kann 3 Zustände haben, nämlich pending (warten), fulfilled und rejected.

Observables
ist der neuere Standard und kann mehr Features als promises.

## Registriere HttpClientModule in AppModule

Öffne den Ordner in VisualStudioCode.

Öffne bzw. schreibe im geöffneten internen Terminal:

**Es sollen 2 Features nachinstalliert werden.**

- Um http requests machen zu können:

```
PS D:\ionic_start\erstesApp> npm install rxjs-compat
```

- Um die Daten zwischen speichern zu können

```
PS D:\ionic_start\erstesApp> npm install --save storage
```

**Öffne die „app.module.ts“**

Um mit diesem http-Modul arbeiten zu können muss man es importieren.

app.module.ts - erstelle den Import von HttpClientModule
NICHT vom Storage!

```
9  import { HttpClientModule } from '@angular/common/http';
```

import { HttpClientModule } from '@angular/common/http';

Füge dazu auch in den „imports“ dieses HttpClientModule nach einem Beistrich ein:

```
15  imports: [BrowserModule, IonicModule.forRoot(), AppRoutingModule,
16    HttpClientModule],
```

Auch nicht vom Storage. Dann würde nämlich die Anzeige des Apps nicht funktionieren.

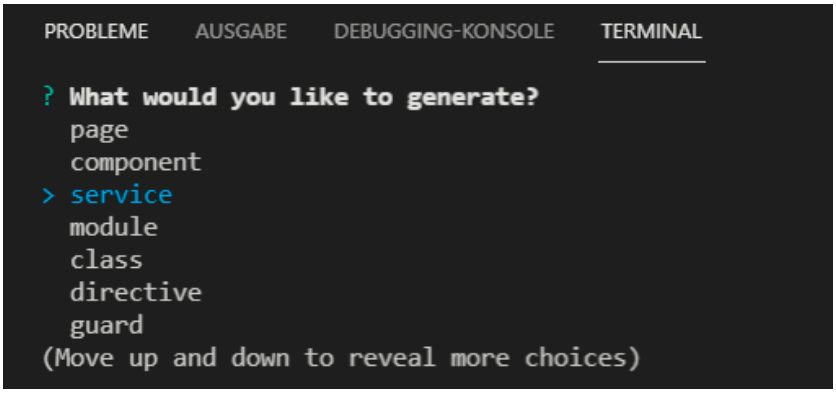
# 3)Service Provider erstellen

Um an einer zentralen guten Stelle die Ausgabe zu organisieren, sollte dies immer in einem vorhandenen Service durchgeführt werden.

Erstelle im integrierten Terminal einen Provider mit dem CMD-Befehl.
Es gibt dafür **2 Varianten:** entweder lässt man IONIC für sich arbeiten – es werden die Auswahlmöglichkeiten angeboten:

1. ionic g

Auswahl mit Pfeiltasten auf „service“ stellen



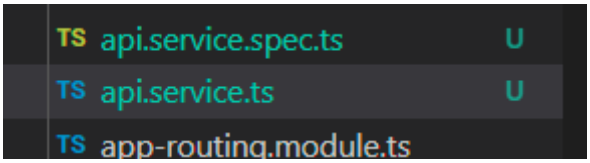

oder selbst eingeben:

2. ionic generate service


Dann wird nach dem Namen gefragt. Gib ein

api

Dieser Name wird automatisch folgendermaßen umgesetzt und in die 2 neue Dateien übernommen:




- app.module.ts

Dieses Service importieren NICHT in app.modules.ts importieren!

## Erstelle http Service mit RxJS Observables

**Öffne „api.service.ts“.**

Schreibe in die Zeile 2 folgenden IMPORT:

import {HttpClient, HttpHeaders} from '@angular/common/http';

```
src > app > TS api.service.ts > ...
  1   import { Injectable } from '@angular/core';
  2   import {HttpClient} from '@angular/common/http';
  3
```

Dann in den Constructor:

```
 9   constructor(
10     public http: HttpClient
11   ) { }
```

Man könnte natürlich auch hier wieder zuerst die Methode im constructor schreiben und sich automatisiert den Import vorschlagen lassen und dann annehmen, um den Import in Zeile 2 nicht selbst schreiben zu müssen.

## Funktion für die Abfrage aus der Datenbank

Diese Funktion soll über den direkten Link zum API auf die Datenbank kommen.

- Dabei ist es EGAL wo die API liegt, weil man mit dem Http-Protokoll ja sowieso überall hinkommt.
- Bei uns liegt die API nicht im der App, sondern extra auf dem Laufwerk C: im Xampp/htdocs-Ordner. Daher muss man für einen erfolgreichen Zugriff dann auch Xampp einschalten.
- Später, wenn die App online sein sollte, wird die App ebenfalls auf dem externen Server (Hoster) liegen, wo die App auch liegen wird. Dann muss man nur den Pfad, den wir hier erstellen werden, ändern.

Erstelle eine Funktion mit einem passenden aussagekräftigen Namen, wie z.B. getBeratersApi. – Beachte die CamelCase-Schreibweise.

- get – weil es eine get-Abfrage sein wird, im Gegensatz zu post oder update oder delete
- Beraters – weil es um die Darstellung der Berater geht, Mehrzahl muss nicht sein, könnte auch Berater lauten. Aber 2. auch deswegen, weil wir in der API, im PHP-Code dort, auch die Funktion „beraters“ genannt haben. Siehe auch später als Ende der URL.
- Api – damit man es von der später zu erstellenden Funktion „getBeraters“ in der „team.page.ts“ besser unterscheiden kann, weil dort beide unmittelbar aufeinander treffen werden.

Die Methode „http.get()“ fragt die Daten per GET vom Server ab.

- Diese Methode „get“ soll ein Array mit verschiedenen Beratern aus der Datenbank zurückgeben
- In der get-Methode wird die URL eingefügt.
- Die „get-Methode“ gibt eine Observable zurück. Dieses Obsevabel besteht aus einem Array mit allen Beratern aus der Datenbank.

```
15
16  getBeratersApi(){
17      return this.http.get('http://localhost/apiionic/public/index.php/beraters');
18    }
19
20  }
```

## 4)NUR zur INFO – zu URL und API

URL: http://localhost/apiionic/public/index.php/beraters

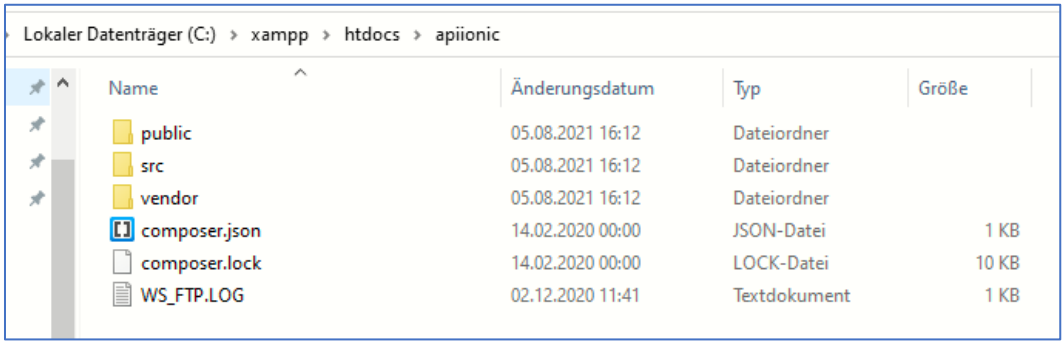
Der Pfad weist ganz hinten in der „index.php“ auf die Funktion „beraters“ hin. Diese ist der ENDPUNKT, welcher die PHP-Abfrage enthält:

```
132 $app->get('/beraters', function(
```

Diese bezieht sich auf die API, die auf C: liegt. Nämlich im Ordner xampp/htdocs. Darin wird in der „index.php“ eine Funktion aufgerufen, die die Beziehung zur Datenbank herstellt. Dies ist eine reine PHP Abfrage, wie wir sie aus dem 3. Jahrgang kennen. Diese API müssen wir nicht selbst erstellen, sondern wird von mir zur Verfügung gestellt.

Für spätere andere Projekte muss man sie entsprechend anpassen.

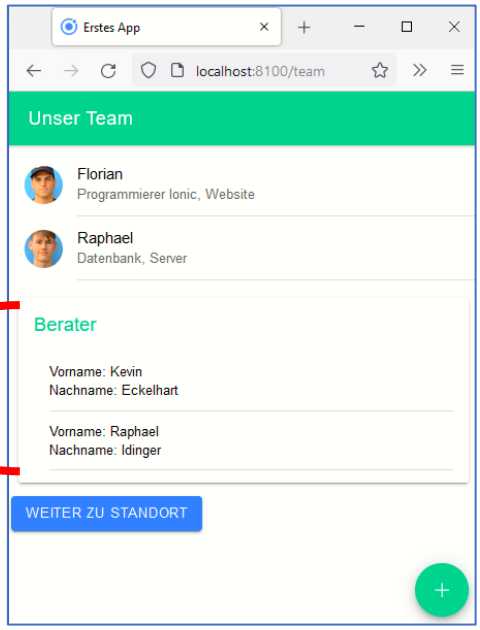
Hier ein Bild davon.




In der „public“ liegt die „index.php“. Diese hat mehrere Funktionen, die alle separat mit Hilfe des von uns erstellten „apiService“ angesprochen werden:

```
131      // create GET HTTP request für alle Benutzer
132  $app->get('/beraters', function( Request $request, Response $response){
133      // echo 'Benutzer';
134
135      $sql = "SELECT vorname, nachname FROM users";
136
137      try {
138        // Get DB Object
139        $db = new db();
140
141        // connect to DB
142        $db = $db->connect();
143
144        // query
145        $stmt = $db->query( $sql );
146        $customers = $stmt->fetchAll( PDO::FETCH_OBJ );
147        $db = null; // clear db object
148
149        // print out the result as json format
150        echo json_encode( $customers );
151
152
153      } catch( PDOException $e ) {
154
155        // show error message as Json format
156        echo '{"error": {"msg": ' . $e->getMessage() . '}';
157      }
158
159    });
160
```

# 5)Daten in der Seite „team.html“ anzeigen lassen




Ziel:
unter den beiden Teammitgliedern soll eine Liste von Beratern angezeigt werden. Diese soll aus der Datenbank kommen und alle aus der betreffenden Tabelle darstellen.

**Öffne „team.page.html“.**

Erstelle nach der ion-list eine

- Ion-card
- mit Titel und
- Inhalt

Bediene dich und kopiere dabei aus der Datei „registrieren.page.html“ was nötig ist oder schreibe den Code selbst:

```
26      </ion-item>
27    </ion-list>
28
29    <ion-card>
30      <ion-card-header>
31        <ion-card-title color="success">Berater</ion-card-title>
32      </ion-card-header>
33      <ion-card-content>
34        <ion-item>
35
36      </ion-item>
37      </ion-card-content>
38    </ion-card>
39
```

In den content soll nun mit einer **For-Schleife der Vorname und Nachname** angeführt werden.

Es soll hier nicht die Listenansicht erstellt werden, sondern eine „card“, damit es sich von der oberen Aufzählung unterscheidet.

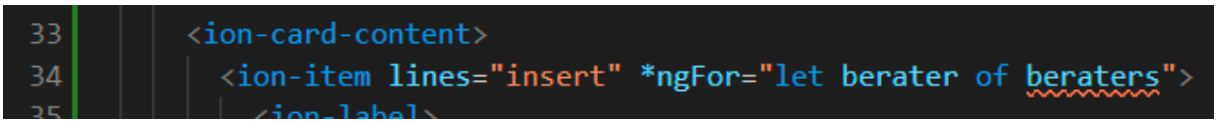
Unterschied zur Aufzählung darüber, mit den beiden Bildern:

- Die Daten sollen aus der Datenbank kommen
- Dafür ist eine FOR-Schleife nötig

## *ngFor:

- Das ist eine Direktive, um durch die Liste der Berater zu iterieren (durchlaufen)
- Der **vorangestellte Stern „*“** gibt Auskunft darüber, dass es sich beim Inhalt des aktuellen Elements um ein sogenanntes Template handelt.
- ngFor wiederholt das Element und dessen Inhalte für jedes Element.

- Die Definition der Schleife bearbeitet alle „beraters“ (Bezeichnung hinter dem „of“ – das ist meistens der gleiche Begriff wie vor dem „of“ aber mit einem Mehrzahl „s“. Es darf nicht der gleiche Begriff sein. Es könnte auch ein beliebig anderer Begriff sein wie z.B. „engelbert“ ☺
- Die Bezeichnung „let“ ist das gleiche wie „var“ für Variable – aber in TypeScript heißt es eben zum Unterschied zu JavaScript eben „let“.
- Der Namen der Variable ist hier „berater“ und kann beliebig gewählt werden, solange er unterhalb genauso wieder verwendet wird.




## Interpolation

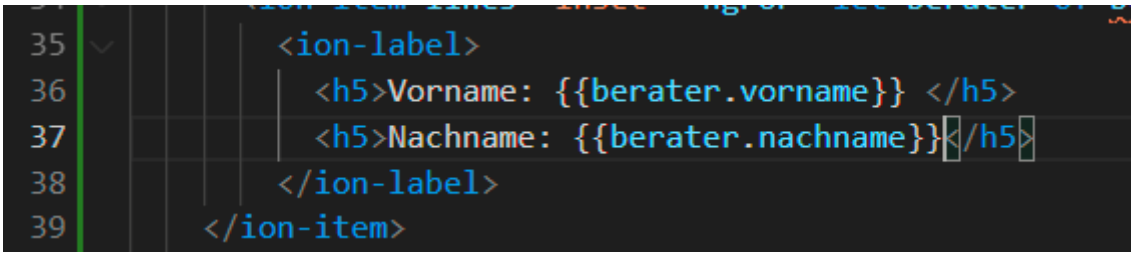
Kommt aus der Open-Source-Software ANGULAR, welche hier mit Ionic die Grundlage bildet.

Dabei werden eingesetzt:

- doppelte geschwungene Klammern vorne und hinten
- Name der Variable aus der For-Schleife
- Nach dem Punkt der Name des Elements aus der Datenbank – die in der .ts – Datei mithilfe des Services aus dem API geholt wird.

Es wird die „**Interpolation**“ **(„{{}}“)** verwendet, um Propertys der Komponentenklasse im Template auszugeben. Die Komponenten-Klasse wird später in der passenden „ts“ in „@Component“ erstellt werden.

<h5>Vorname: {{berater.vorname}} </h5>
<h5>Nachname: {{berater.nachname}}</h5>




Info:
Das Element aus der Datenbank (nach dem Punkt) wird in der „team.page.ts“ über die Funktion „getBeraters()“ und einem „Observable“ (mit „subscribe“) über das „apiService“ aus der Datenbank geholt. Dazwischen steht hier, zur Absicherung, das API, welches per http-Request im Service angesteuert wird.

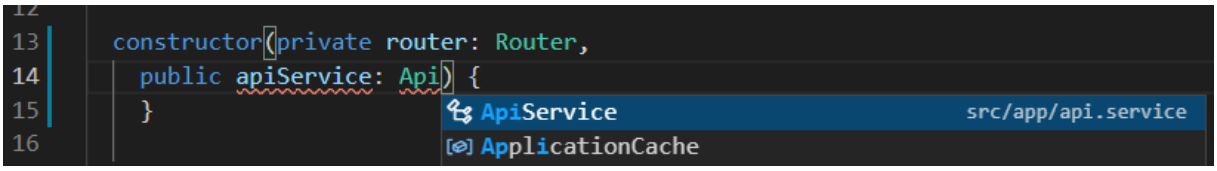
Die HTML ist somit fertig.

### Öffne die „team.page.ts“

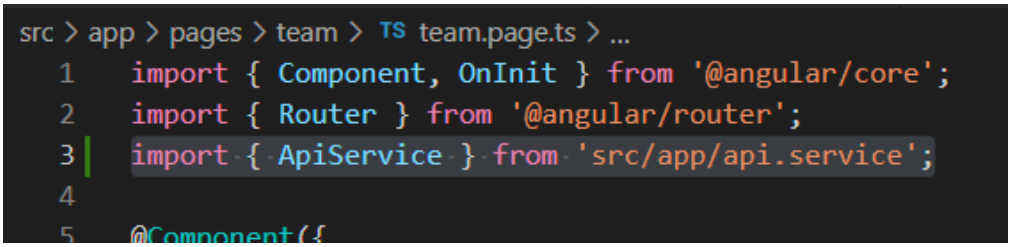
Der dazu passende TypeScript-Code wird hier nun erstellt, damit die HTML funktioniert.

In der ts-Datei wird das Service zur URL (Datenbank-Verbindung) hergestellt, damit die Daten in der HTML angezeigt werden können.

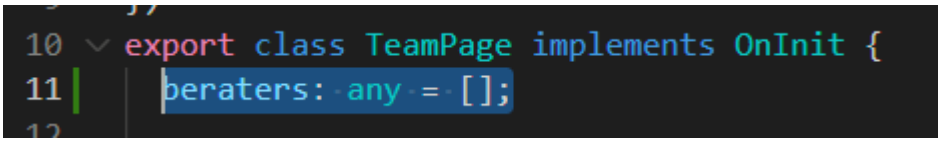
1. Zuerst muss man das **api.service in den constructor importieren**, damit es hier verwendet werden kann:
   Um den Import aber nicht selbst schreiben zu müssen ist es leichter, ZUERST im „constructor“ die Methode anzuschreiben und dabei die automatische IMPORT-Funktion zu nutzen.
   Schreibe die ersten Buchstaben und lasse dann den Vorschlag erscheinen, wobei der oberste, wenn „service“ dabei steht angeklickt werden soll:




   Daraufhin schreibt sich der IMPORT in den obersten (Import-) Zeilen von selbst.




2. In der export-Klasse dieser Seite muss man ein Array erstellen, welches jeden Wert annehmen kann (any) und den Namen hat, der als 2. Name in der FOR-Schleife aus der HTML-Datei vorkommt, hier, wie immer, die Mehrzahl „beraters“. Diese beiden müssen übereinstimmen.




3. Funktion erstellen:
   ein guter Name für das Holen aus der Datenbank könnte „getBeraters“ heißen.
   Diese repliziert auf sich (this) und nutzt die Methode „apiService“ welches „getBeratersApi“ heißt.
   Mit „subscribe“ wird die Möglichkeit von „Observables“ genutzt.

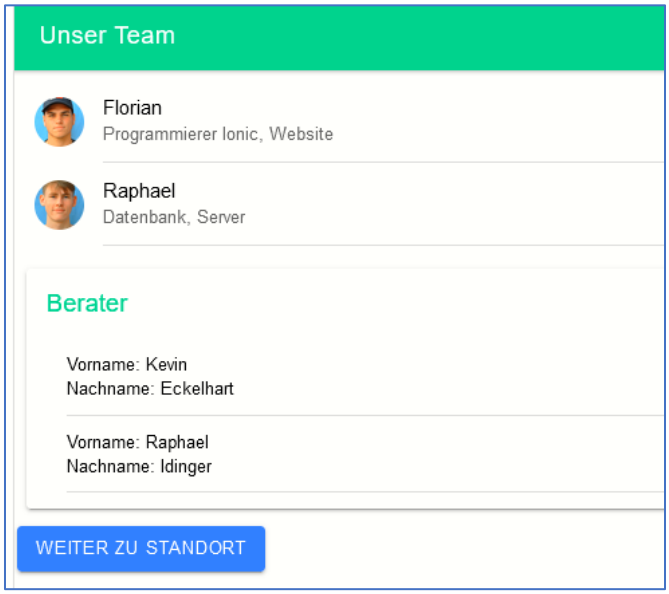
```
getBeraters(){
 this.apiService.getBeratersApi().subscribe((res: any) => {
 this.beraters = res;
 });
}
```

```
24  getBeraters(){
25    this.apiService.getBeratersApi().subscribe((res: any) => {
26      this.beraters = res;
27    });
28  }
```

4. Diese Funktion im constructor aufrufen

```
13    constructor(private router: Router,
14      public apiService: ApiService) {
15      this.getBeraters();
16      }
```

Ergebnis:




Quelle:

https://www.youtube.com/watch?v=bqiHfIBh8Xk