Ionic 8 standalone mit Angular 17

Shop: Datenbank und Service-Provider

Inhalt:

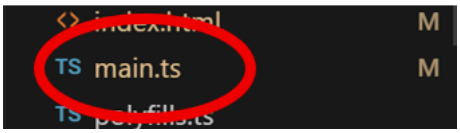
1. http-Modul für Datenbank-Verbindung in der „main.ts“ addieren
2. Service-Provider erstellen

1)http-Modul für Datenbank-Verbindung

Theorie:
Das HttpClient Modul benötigt man, um http **POST, GET, PUT und DELETE requests** machen zu können. Dadurch wird es möglich Daten zum API (Verbindung zur Datenbank) zu senden bzw. zu erhalten.

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

Öffne die „main.ts“ und addiere eine Zeile in den „providers“. Am besten ganz unten, nach dem Beistrich.




```
bootstrapApplication(AppComponent, {
  providers: [
    { provide: RouteReuseStrategy, useClass: IonicRouteStrategy },
    provideIonicAngular(),
    provideRouter(routes, withPreloading(PreloadAllModules)),
    provideHttpClient(),
  ],
});
```

Bei guter Auswahl wird der IMPORT gleich automatisch durchgeführt:

```
import { AppComponent } from './app/app.component';
import { provideHttpClient } from '@angular/common/http';

bootstrapApplication(AppComponent, {
```

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

2)Service Provider erstellen

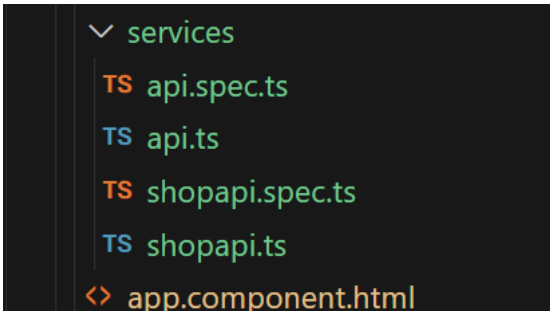
Um an einer zentralen guten Stelle die Ausgabe zu organisieren, sollte dies immer in einem Service durchgeführt werden.

Erstelle im integrierten Terminal einen Provider ein Service.

ionic g service services/shopapi

```
ionic g service services/shopapi
```

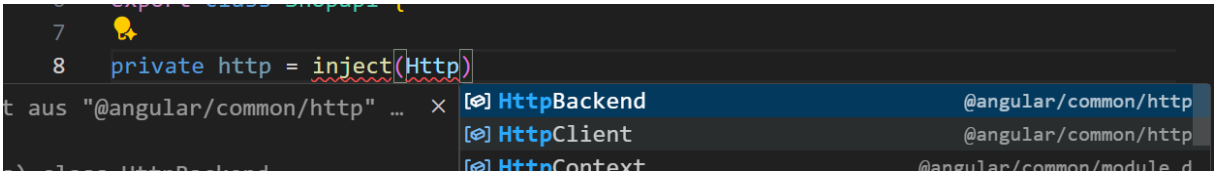
Ergebnis:



Nutze die neue direkt **INJECT-Funktion**, um den http-Client aufzurufen:

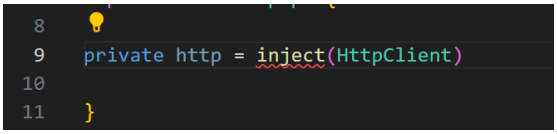
Öffne die „shopapi.ts“:

Nutze dabei den auto-Import:



Damit wird der Import ganz oben durchgeführt und man muss in nicht händisch schreiben.

Dann ist zwar noch der Fehler, aber der benötigt entweder die Schnellfehlerlösung oder man schreibt selbst in den Import das „inject“.



Ergebnis:

```ts
import { HttpClient } from '@angular/common/http';
import { inject, Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class Shopapi {

private http = inject(HttpClient);

}
```

Info: https://www.youtube.com/watch?v=tbrJJkSYQ04 Zeit:7:10

Nur zur Info für die spätere Verwendung: wird später noch genau erklärt

Darunter folgt dann immer eine oder mehrere Funktionen, die wie hier z.B. mit einer GET-Abfrage auf die API verweisen

```ts

  private http = inject(HttpClient);

  getSchuheApi() {
    return this.http.get('https://');
  }
  }

```

Zum Beispiel in der „shop.page.ts“

Wie bekannt lautet die „export class Shopapi“, also mit Großbuchstaben „S“ und dann Shopapi.

```ts
})
export class Shopapi {

private http = inject(HttpClient);
```

Diese genaue Bezeichnung muss man dann in der dazu passenden .ts, hier die „shop.page.ts“ übernehmen – hinten in der „inject“ – wenn man es wieder sinnvoll eingibt, kann man den Import automatisieren - ansonsten muss man das händisch importieren.

```
export class ShopPage implements OnInit {
  items: any[] = [];

  private schuheapi = inject(Shopapi);

  constructor() {}
```

Die Bezeichnung nach dem „private“ muss sich aber unterscheiden, daher ist hier „schuheapi“ verwendet.

```
  items: any[] = [];

  private schuheapi = inject(Shopapi);
```

Import aus "src/app/services/shopa… ×

Shopapi src/app/services/shop…

```
import { IonContent, IonHeader, IonTitle, IonToolbar, IonRow,
import { Shopapi } from 'src/app/services/shopapi';
```

Die private Methode „schuheapi“ wird dann verwendet, wenn eine neue Funktion die Verbindung zur API herstellt.

```
  private schuheapi = inject(Shopapi);

  constructor() {
    this.getSchuhe();
  }

  ngOnInit() { }

getSchuhe() {
  this.schuheapi.getSchuheApi().subscribe((response: any) => {
    this.items = response;
    console.log(this.items);
  });
}
}
```