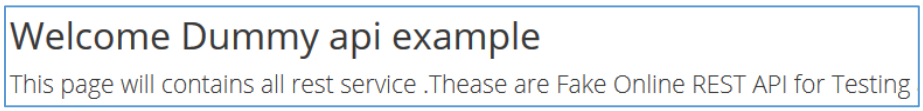
Angular – Formular erstellen und in externer Datenbank speichern

Ein Formular per POST an den Server senden

Externer Server mit bereits freier API für Testzwecke: http://dummy.restapiexample.com



Für GET-Abfragen: http://dummy.restapiexample.com/api/v1/employees

Für POST: http://dummy.restapiexample.com/api/v1/create

There are following public apis

#	Route	Method	Type	Full route
1	/employee	GET	JSON	http://dummy.restapiexample.com/api/v1/employees
2	/employee/{id}	GET	JSON	http://dummy.restapiexample.com/api/v1/employee/1
3	/create	POST	JSON	http://dummy.restapiexample.com/api/v1/create
4	/update/{id}	PUT	JSON	http://dummy.restapiexample.com/api/v1/update/21
5	/delete/{id}	DELETE	JSON	http://dummy.restapiexample.com/api/v1/delete/2

1)Vorbereitungen und HTML-Formular

1a)neue Komponente anlegen, Navigation und Route ändern

- Erstelle eine neue Komponente namens „empoyees“.

```
PS C:\angular_rk\idee> ng g c empoyees
```

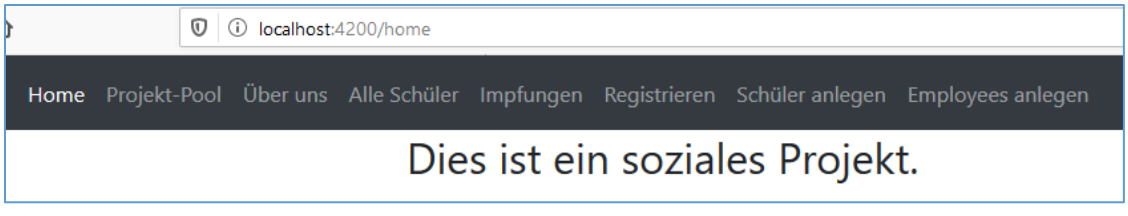
- **Navigation in „app.component.html“ anlegen**

app.component.html

src > app > app.component.html > div.container > nav.navbar.navbar-expand

```
23      </li>
24      <li class="nav-item">
25        <a routerLink='/employees' routerLinkActive="active"
26        class="nav-link">Employees anlegen</a>
27      </li>
```

Ergebnis in der Navigation:



- **Route anlegen**

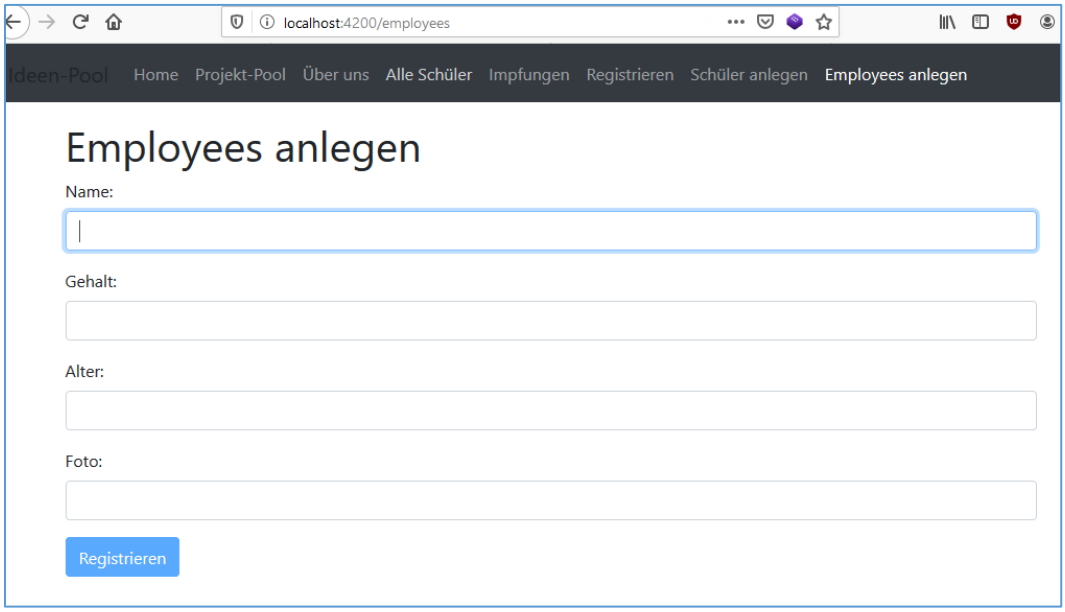
Öffne die „app-routing.modules.ts“

Lege den „path“ und den „import“ dafür an.

```
app.component.html ●   app-routing.module.ts ×
src > app > app-routing.module.ts > AppRoutingModule
13   import { ImpfungComponent } from './impfung/impfung.component';
14   import { EmployeesComponent } from './employees/employees.component';
15
```

```
29    { path: 'form-reg', component: FormRegComponent},
30    { path: 'form-local', component: FormLocalComponent },
31    { path: 'employees', component: EmployeesComponent }
32  ];
```

1b)Formular anlegen in „empoyees.component.html“



Code:

```
<br><br><br>
 <h1>Employees anlegen</h1>
  <form #employeesForm="ngForm" >

   <div class="form-group">
    <label>Name:</label>
    <input type="text" name="name" required #employee_name="ngModel"
     class="form-control" >
   </div>
```

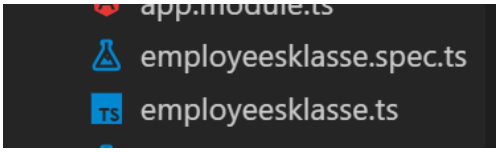
```
  <div class="form-group">
    <label>Gehalt:</label>
    <input type="text" name="salary" class="form-control">
  </div>

  <div class="form-group">
    <label>Alter:</label>
    <input type="text" name="age"  class="form-control">
  </div>

   <button [disabled]="employeesForm.form.invalid" class="btn btn-primary"
   type="submit">Registrieren</button>
</form>
```

2)Klasse erstellen, Instanz erstellen und Data-Binding

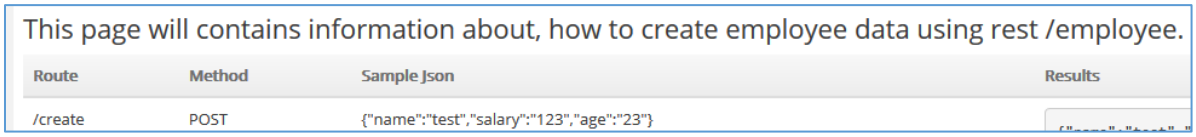
2.1)Klasse erstellen

ng g class employeesklasse

Dadurch entstehen diese beiden Dateien:



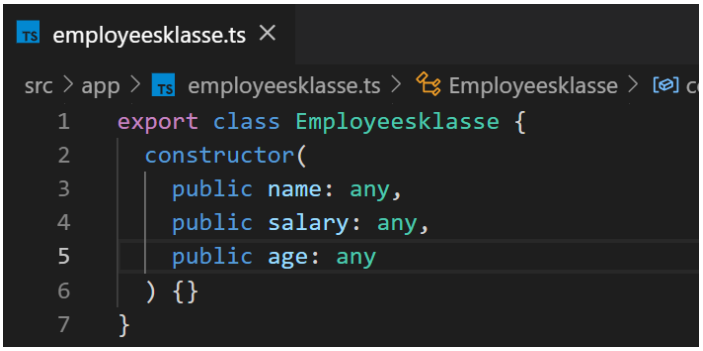
Zuerst muss man die gewünschten Entity-Bezeichnungen in der Ziel-Datenbank ansehen:



Daher:

- name
- salary
- age

Dann kann man auch die Klasse entsprechend gestalten:

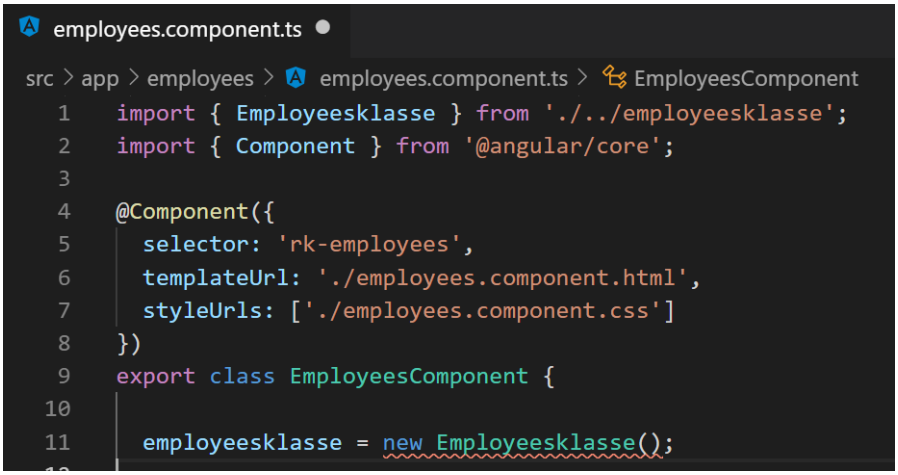


Auf diese „employeesklasse“ wird dann öfters zugegriffen werden.

2.2)Eine Instanz in der zugehörigen ts-Datei anlegen

Öffne die „employees.component.ts“. Vorhanden ist folgendes, wobei man den „constructor“ und „ngOnInit“ löschen kann, ebenfalls das „OnInit“ in Zeile 1 und in Zeile 8:

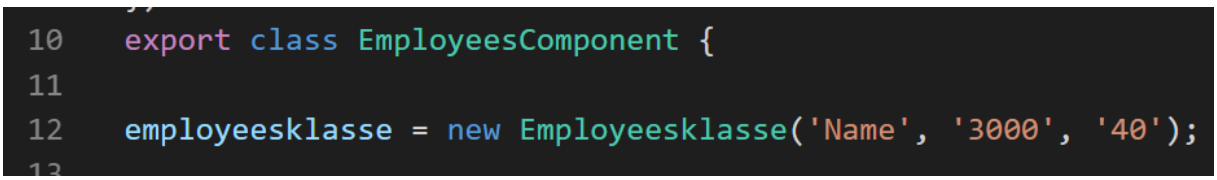
Danach:



Hier folgt nun nach der Zeile 8 im „export“ der neue Code in Zeile 11.

Der „import“ in Zeile 1 erfolgt automatisch bzw. händisch.

Gib danach Elemente ein, die zum Formular (in der richtigen Reihenfolge) passen. Hier sind es eigentlich Platzhalter du keine echten Beispiel-Daten. Dann kennt sich ein Benutzer besser aus und kann sie sofort überschreiben:



Hier wurde somit ein „Model“ erstellt, das nun in das Formular eingebunden wird.

2.3)Data binding im Formular (Html)

Dafür wird nun jedes Formularfeld ein Two-Way-Binding erhalten.

Das „ngModul“ erhält eckige und runde Klammern für das „Two-Way-Data-Binding“. Damit werden neue Werte SOFORT in das „employeesklasse“-Objekt übertragen.

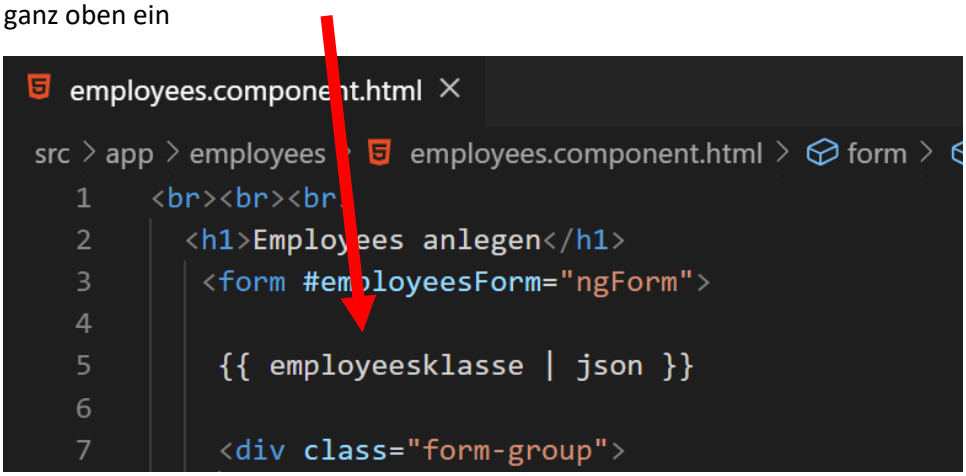
Verwende das bereits vorhandene „ngModel“ und umschließe es mit den beiden Klammern vorne und hinten (sogenannte Banana in the Box). Dann das eben in der „ts-Datei“ angelegte Model und den Namen des Feldes, indem man dieses Binding durchführt:

[(ngModel)]="employeesklasse.name"

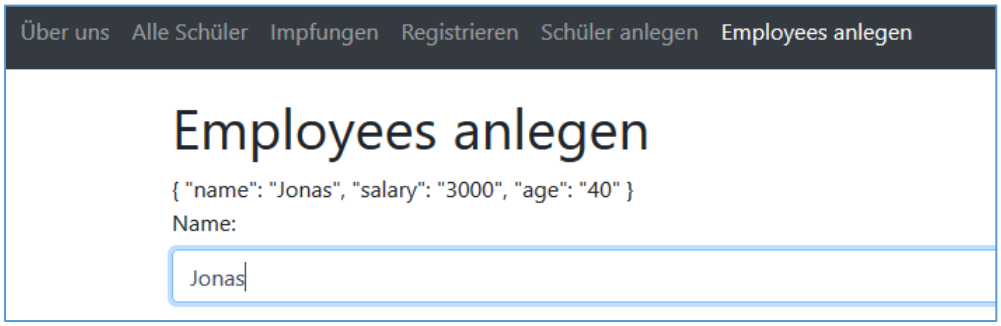
Siehe Zeile 9:

```
5     <div class="form-group">
6       <label>Name:</label>
7       <input type="text" name="name" required #employee_name="ngModel"
8        class="form-control" [(ngModel)]="employeesklasse.name">
9     </div>
```

4)Eine Kontrolle kann erfolgen mit Hilfe der Interpolation im Formular selbst. Füge irgendwo, z.B. ganz oben ein



Nun sieht man im Formular, dass bei einer Änderung **SOFORT** diese Änderung umgesetzt wird. Probiere es aus und überschreibe einige Zellen:



Die nur zur Darstellung angelegte Interpolation mit den doppelten {{ }} kann wieder entfernt werden. Im Programm benötigt man es nicht.

3)Formular umschreiben für das „submitting“ der Form-Daten

- In den <form>-Tag schreibe „novalidate“. Damit wird verhindert, dass eine Validierung seitens eines Browsers vorgenommen wird. Es soll ja die von Angular gelten.

employees.component.html

src > app > employees > employees.component.html > form > div.form-group

```
1   <br><br><br>
2     <h1>Employees anlegen</h1>
3       <form #employeesForm="ngForm" novalidate >
4
```

- Danach binde zum „submit“ Event, wenn der Submit-Button geklickt wird. Danach folgt ein Event-Handler „onSubmit()“

```
2     <h1>Employees anlegen</h1>
3       <form #employeesForm="ngForm" (ngSubmit)="onSubmit()" novalidate >
4
```

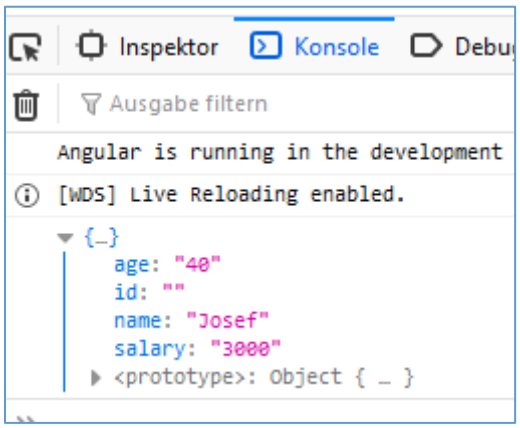
- Öffne die „employees.component.ts“ und ganz unten, vor der letzten geschwungenen Klammer füge ein:

```
onSubmit() {
  console.log(this.employeesklasse);
}
```

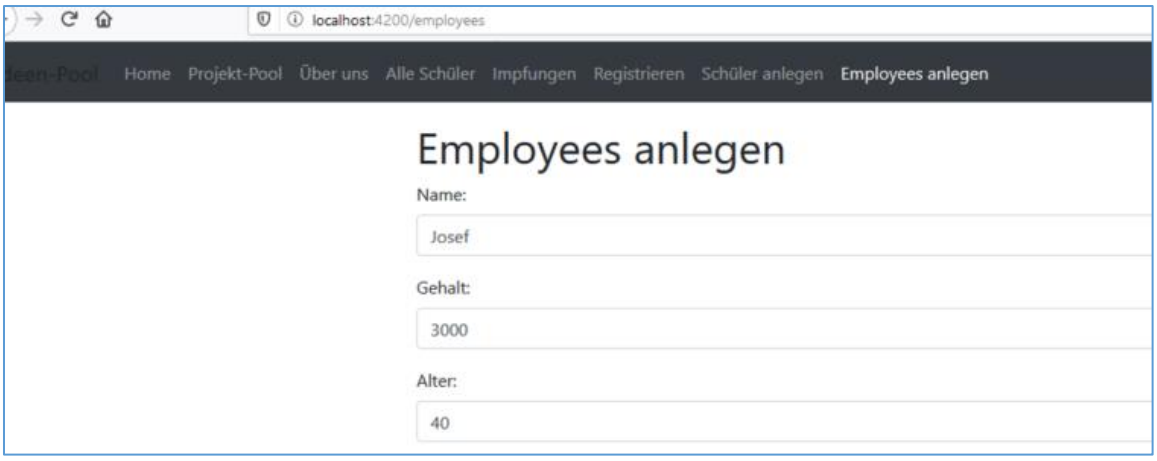
```
11  employeesklasse = new Employeesklasse('Name', '3000', '40');
12
13  onSubmit() {
14    console.log(this.employeesklasse);
15  }
16  }
```

Betrachte das Formular im Browser uns sende es ab durch Klick auf „Registrieren“.

In die Konsole (F12) wird nun folgendes geschrieben:



Ergebnis:

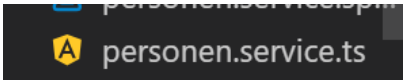


4)Service erstellen

Um die Daten an einen Server senden zu können, benötigt man ein Service.

Entweder erstellt man nun ein neues Service (im Terminal: ng g s name) oder man greift auf ein bereits vorhandenes Zurück und erstellt dort neuen Code.

In einem neuen Service ist es wichtig das http-Modul einzubinden. In unsrem Fall ist das schon erledigt.

Öffne „personen.service.ts“ 

4.1)URL erstellen

Erstelle den passenden Pfad zur REST-Url: Hier ist das der letzte, der api5:

```
16    private api4 = 'http://localhost/2019_angular_db/angular2.php';
17    private api5 = 'http://dummy.restapiexample.com/api/v1/create';
18
19      constructor(private http: HttpClient) { }
20
```

4.2)POST-Request erstellen

Erstelle danach ganz unten, vor der letzten geschwungenen Klammer die Methode „make“:
Der Name „make“ ist frei wählbar und darf hier nicht nochmals vorkommen.

```
36
37    make(employees: Employeesklasse) {
38      return this.http.post<any>(this.api5, employees);
39    }
40
```

Die Employeesklasse ist die Klasse, die das Model des Schülers enthält.

INFO:
In der REST-Architektur dient die http-POST-Methode dazu, neue Entitäten anzulegen.

Bei dieser Implementierung der Methode profitiert man von einer sehr bequemen Annehmlichkeit: erkennt das http-Framework, dass man als Body ein JavaScript-Objekt übermittelt, so wird dieses Objekt automatisch in einen JSON-String umgewandelt. Des Weiteren wird automatisch der „Content-Type-Header“ des Requests auf den Wert „application/json“ gesetzt.

Dieser Code ist somit eine spezielle Kurzform.

4.3)subscribe vom Observable
Diese obige Kurzform ist ein Observable, auch wenn es nicht direkt darin steht. Daher muss man es in der passenden ts-Datei „subscriben“.

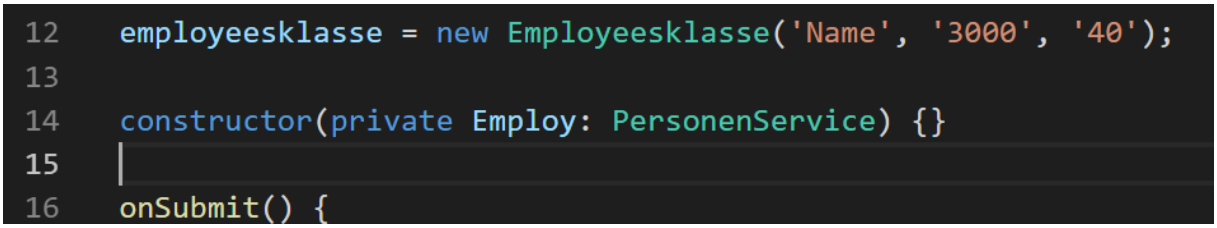
Öffne die „employees.component.ts“.

- Zuerst muss man das benutzte Service importieren:

import { PersonenService } from './../personen.service';

- Vor der „onSubmit“ Methode muss man den „constructor“ erstellen und das Service verwenden

constructor(private Employ: PersonenService) {}



Der Name „Employ“ ist frei wählbar und muss weiterunten im subscribe wiederholt werden.

- Lösche den Inhalt von „onSubmit“ und schreibe

```
14    constructor(private Employ: PersonenService) {}
15
16  onSubmit() {
17    this.Employ.make(this.employeesklasse)
18    .subscribe();
19    }
20    }
```

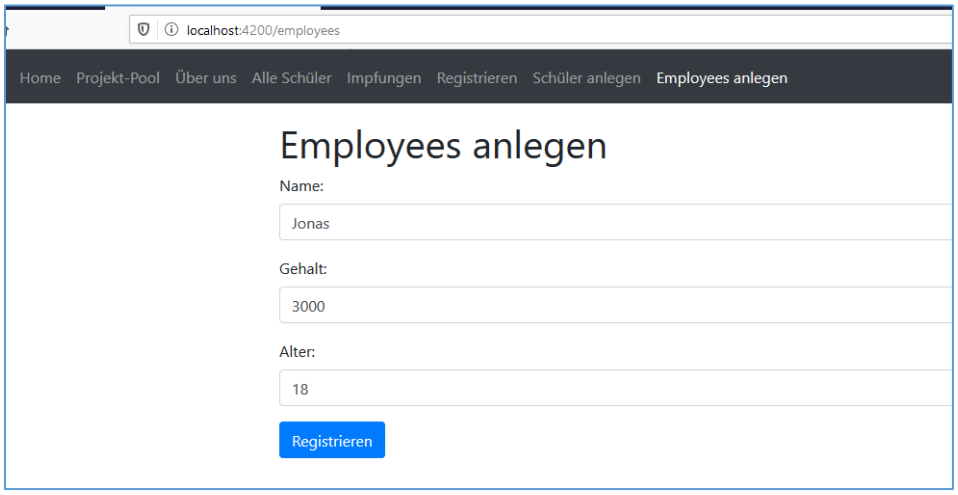
„Employ“ ist der Begriff aus dem „constructor“. Kann prinzipiell frei gewählt werden, Hauptsache er passt im „constructor“ und hier zusammen.
employeesklasse ist das Model aus der Klasse.

- In das „subscribe“ wird dann das bisherige „employees“-Objekt der Klasse durch das vom Service zurückgelieferte ersetzt.

```
16    onSubmit() {
17      this.Employ.make(this.employeesklasse)
18      .subscribe(employees => console.log('Erfolgreich gespeichert', employees));
19    }
20    }
```

5.)Testen

Öffne das eben angelegte Formular und gib einen neuen Namen und Gehalt und Alter ein.



Klicke auf Registrieren.

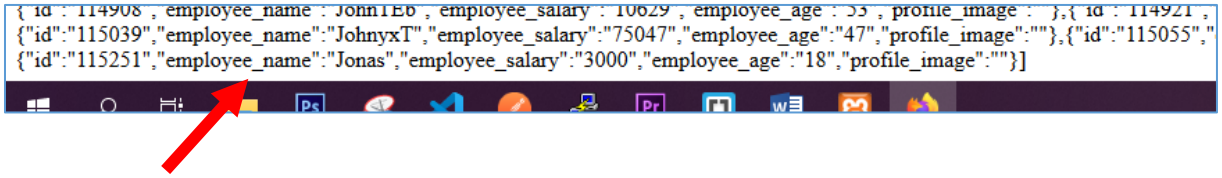
Öffne den Link.

http://dummy.restapiexample.com/api/v1/employees

Unsere Konsole zeigt schon mal keinen Fehler an.

Dieser zeigt in JSON Darstellung den Inhalt der entfernten Datenbank an. Da schon sehr viele dieses Service genutzt haben, sind schon viele Daten enthalten.

Unsere sollte die letzten ganz unten sein:



Erfolgreich angelegt!!!

Das fertige Formular als Code: employees.component.html

```
<br><br><br>
 <h1>Employees anlegen</h1>
  <form #employeesForm="ngForm" (ngSubmit)="onSubmit()" novalidate >

   <div class="form-group">
    <label>Name:</label>
    <input type="text" name="name" required #employee_name="ngModel"
     class="form-control" [(ngModel)]="employeesklasse.name">
   </div>

   <div class="form-group">
    <label>Gehalt:</label>
    <input type="text" name="salary" class="form-control"
    [(ngModel)]="employeesklasse.salary">
   </div>

   <div class="form-group">
    <label>Alter:</label>
    <input type="text" name="age"  class="form-control"
    [(ngModel)]="employeesklasse.age">
   </div>


    <button [disabled]="employeesForm.form.invalid" class="btn btn-primary"
    type="submit">Registrieren</button>
 </form>
```

Quellen:

Woiwode, Malcher, u.a. in: Angular, dpunkt.verlag, 2018, S.209-225

Christoph Höller, in: Angular, Das umfassende Handbuch, Rheinwerk Verlag, 2019, S.308-332

https://www.youtube.com/watch?v=2cWPk2X79is&list=PLC3y8-rFHvwhwL-XH04cHOpJnkgRKykFi&index=6

https://www.youtube.com/watch?v=lmteIO2FQYA&list=PLC3y8-rFHvwhwL-XH04cHOpJnkgRKykFi&index=12