Allgemeine Befehle

Projekt anlegen

```
ng new <project-name>
```

Projekt starten (Port 4200)

```
ng serve
```

Component erstellen

```
ng generate component <component-name>
```

Service erstellen

```
ng generate service <service-name>
```

Class erstellen

```
ng generate class <class-name>
```

Material Component erstellen

```
ng generate @angular/material:<material-element> <component-name>
```

Genauerer Beschreibung in den einzelnen Kapiteln

Components Databinding

@Input

1. Values weitergeben beim Component verwenden

```

```

```
<app-greeting
    vorname="Max"
    [nachname]="'Muster'"
>
</app-greeting>
```

2. Im Typescript

```
export class GreetingComponent {
    @Input('vorname') firstname | undefined;
    @Input('nachname') lastname | undefined;
}
```

3. Im HTML von der GreetingComponent

```
<p>Hello {{firstname}} {{lastname}}!</p>
```

@Output

- Übergeordnete Komponenten über Ereignisse zu informieren

1. Greeting Component

```
import {EventEmitter} from '@angular/core';

export class GreetingComponent {
    @Output() loggedOut = new EventEmitter();

    logout() {
        this.loggedOut.emit();
    }
}
```

2. Event im Greeting auslösen

```
<button (click)="logout()">Logout :(<button>
```

3. Event abfangen in Parentcomponent

```
<app-greeting
(loggedOut)="handleLogout()"
```

```
>
</app-greeting>
```

Local References

- Wird über Template an TS-File übergeben
- Syntax: #Bezeichner

```
<input type="text" #nicknameInput>
<button (click)="login(nicknameInput)">Login</button>
Hello {{ nickname }}`!
```

```
export class GreetingComponent {
    nickname;

    login(nicknameInput: HTMLInputElement) {
        this.nickname = nicknameInput.value;
    }
}
```

ViewChild

- Zugriff auf HTML Elemente

```
export class GreetingComponent {
    @ViewChild('nicknameInput', {static: true | false}) nicknameInput: ElementRef;
    nickname;

    login() {
        this.nickname = this.nicknameInput.nativeElement.value;
    }
}
```

```
<input type="text" #nicknameInput>
<button (click)="login()">Logout</button>
```

HTTP Service erstellen

- Dependency Injection!

Setup

```
ng generate service <service-name>
```

Konfiguration für HTTP Service

Imports

Im service-name.ts

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

Im constructor

```
constructor(private http: HttpClient) {

}
```

Im app-module.ts

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

@NgModule({
    declarations: [AppComponent],
    imports: [BrowserModule, FormsModule, HttpClientModule],
    providers: [],
    bootstrap: [AppComponent]
})
```

Test Requests

Optionaler function return type

```
getCustomers(): Observable<Customers[]> {
    return this.http.get<Customers[]>
(`${this.baseUrl}/customers/getAllCustomers/`);
}
```

GET

```
getCustomers() {
    return this.http.get<Customers[]>
```

```
(`${this.baseUrl}/customers/getAllCustomers/`);
}
```

POST

```
createCustomer(customer: Customer) {
    return this.http.post<Customer>(`${this.baseUrl}/customer`, customer);
}
```

UPDATE

```
updateCustomer(customer: Customer) {
    return this.http.put<Customer>(`${this.baseUrl}/customer`, customer);
}
```

DELETE

-> Info laut Julian i habs ned getestet 😉

```
deleteCustomer(customer: Customer) {
    return this.http.delete<Customer>(`${this.baseUrl}/customer`, customer);
}
```

Injecten und Verwendung von HTTP Service

Injecten

```
constructor(private httpService: HttpService) {

}
```

Benutzen von HTTP Service

```
loadCustomers() {
    this.httpService.getCustomers().subscribe((data: Customers[]) => {
      console.log("DO SAN DE CUSTOMER JAWOIIII!")
    }, (error) => {
        console.log("kane customer :(");
        console.error(error);
    });
}
```