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 Support hinzufügen

```
ng add @angular/material
```

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);
```

```
    });
}
```

Routing

Routing konfgurieren, falls noch nicht vorhanden

```
ng generate module app-routing --flat --module=app
```

Dann sollte app-routing.module.ts vorhanden sein

SETUP

(in app-routing.module.ts oder app.module.ts)

Import

```
import { RouterModule, Routes } from '@angular/router';
```

Routes definieren

```
const routes: Routes = [
  { path: '', component: CustomerComponent },
  { path: 'customer-component', component: CustomerComponent },
  { path: 'reservationlist-component', component: ReservationListComponent },
];
```

```
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
```

Navigieren

Normal über Route

```
<a [routerLink]="['customer-component']">Kundendaten</a>
<a routerLink="customer-component">Kundendaten</a>
```

```
<a  href="reservationlist-component">Reservierungsliste</a>
```

Router Outlet einfügen bitte ahhhh

- In das Router-Outlet wird die aktuelle Component geladen

```
<router-outlet></router-outlet>
```

Nested Routes

```
const routes: Routes = [
  { path: 'reservationlist-component', component: ReservationListComponent,
children: [
      {path: 'r1sub1': component: SubComponent1},
      {path: 'r1sub2': component: SubComponent2},
  ] },
];
```

Wichtig: In den Parent Components gehört immer a router-outlet (also in dem Fall in der AppComponent und ReservationListComponent)

```
<a [routerLink]="['reservationlist-component', 'r1sub1']">Sub</a>
Alternativ:
<a [routerLink]="['reservationlist-component/r1sub1']">Subcomponent</a>
Alternativ:
<a [routerLink]="reservationlist-component/r1sub1">Subcomponent</a>

<router-outlet></router-outlet>
```

Übergabe von Parameter

```
const appRoutes: Routes = [
    {path: 'user/:id/:name', component: UserComponent}
];
```

Auslesen von Params

```
export class UserComponent implements OnInit {
    id: number;
    name: string;
```

```
    constructor(private route: ActivatedRoute) { }

    ngOnInit() {
        // this.id = this.route.snapshot.params['id'];
        // this.name = this.route.snapshot.params['name'];

        this.route.params.subscribe(
            (params: Params) => {
                this.id = params['id'];
                this.name = params['name'];
            }
        )
    }
}
```

Query Parameter

```
<a
    [routerLink]="['/user', 1, 'Max']"
    [queryParams]="{allowEdit: '1' }"
    fragment="editing"
>
    Bearbeite User
</a>

 localhost:4200/user/1/Max?allowEdit=1
```

Auslesen von QueryParams

```
export class UserComponent implements OnInit {
    id: number;
    name: string;

    constructor(private route: ActivatedRoute) { }

    ngOnInit() {
        // this.id = this.route.snapshot.params['id'];
        // this.name = this.route.snapshot.params['name'];

        this.route.queryParams.subscribe(
            (queryParams: Params) => {
                this.allowEdit = queryParams['allowEdit'];
            }
        )
    }
}
```

Navigieren mittels Programmlogik

```
  constructor(private router: Router) {

  }

  navigateTo() {
    // greeting-component/subcomponent/1
    this.router.navigate(['greeting-component', 'subcomponent', '1'],
        {queryParams: {"isEdit": '1'}})
  }
```

==

```
<a
    [routerLink]="['greeting-component', 'subcomponent', '1']"
    [queryParams]="{isEdit: '1' }"
>
```

Forms

Template Driven - TDF

Setup

```
import {FormsModule} from '@angular/forms';
// Auch in imports Array reinballern
```

Example

```
<form (ngSubmit)="onSubmit(f)" #f="ngForm">
    <input type="text " class="form-control" ngModel #username="ngModel"
id="username" name="username" required>

    <input type="email" class="form-control" ngModel name="email" required email
#email="ngModel" min-length="3">
    <span>{{email.value}}</span>

    <button class="btn btn-primary" type="submit" [disabled]="!f.valid ||
!f.dirty">Submit</button>
</form>

Weitere Parameter:
- required
```

```
- min-length="3"
- max-length="3"
- email (email verifier)
- pattern="[0-9]{3}-[0-9]{3}-[0-9]{4}"


Input Types:
- number
- tel
```

Auslesen der Daten bei Submit

```
onSubmit(f: NgForm) {
    console.log(f.value['username'])
    console.log(f.value)
    // deep copy erstellen
    var obj = Object.assign({}, f.value);
}
```

Reactive

Setup

```
import {ReactiveFormsModule} from '@angular/forms';
// Auch in imports Array reinballern
```

Erstellen von am Form

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

Example

```
<form [formGroup]="addressForm" novalidate (ngSubmit)="onSubmit()">
  <mat-card class="shipping-card">
    <mat-card-content>

      <div class="row">
        <div class="col">
          <mat-form-field class="full-width">
            <input matInput placeholder="First name" formControlName="firstName">
            <mat-error
*ngIf="addressForm.controls['firstName'].hasError('required')">
              First name is <strong>required</strong>
            </mat-error>
```

```
          </mat-form-field>
        </div>

      </div>
      <div class="row">
        <div class="col">
          <mat-form-field class="full-width">
            <textarea matInput placeholder="Address" formControlName="address">
</textarea>
            <mat-error
*ngIf="addressForm.controls['address'].hasError('required')">
              Address is <strong>required</strong>
            </mat-error>
          </mat-form-field>
        </div>
      </div>


      <div class="row">
        <div class="col">
          <mat-form-field class="full-width">
            <input matInput #postalCode maxlength="5" placeholder="Postal Code"
type="number"
              formControlName="postalCode">
            <mat-hint align="end">{{postalCode.value.length}} / 5</mat-hint>
          </mat-form-field>
        </div>
      </div>

    </mat-card-content>
    <mat-card-actions>
      <button  [disabled]="!this.addressForm.valid"  mat-raised-button
color="primary" type="submit">Edit</button>
    </mat-card-actions>
  </mat-card>
</form>
```

```
export class ReactiveFormComponent implements OnInit{
  addressForm = this.fb.group({
    firstName: [null, Validators.required],
    address: [null, Validators.required],
    postalCode: [null, Validators.compose([
      Validators.required, Validators.minLength(5), Validators.maxLength(5)])
    ],
  });

  // values reinpatchen!
  ngOnInit(): void {
    this.addressForm.patchValue(
      data
```

```
    );

    // oder für einzelnes value
    this.addressForm.setValue(
      {
        firstName: "Gerry",
        address: "Gerry"
      }
    )

    // oder
    this.addressForm.controls['firstName'].setValue("Gerry");
  }

  constructor(private fb: FormBuilder, private httpService: HttpServiceService) {}


  // on submit
  onSubmit(): void {
    if (this.addressForm.valid) {
      console.log("Voi valid!");
      var res = Object.assign({}, this.addressForm.value);
      var username = this.addressForm.get('username')
    }
  }
}
```

Direktive

```
<p [style.backgroundColor]="testvar == 5 ? 'red' : 'blue'">Test</p>

<li  *ngFor="let u of users; let i=index">{{i}} - {{u}</li>}

<p [ngStyle]="[backgroundColor: getColor()]">Text</p>
```

Classes

Setup

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

Example Class

```
export class Customer {
    id: number = 0;
    firstname: string = "";
    lastname: string = "";
    street: string = "";
    houseno: string = "";
    zip: string = "";
    city: string = "";
    password: string = "";
}
```

Subject

```
export class SearchService {
    searchSource = new Subject<string>();
    constructor() {}

    setSearchString(message: string) {
        this.searchSource.next(message);
    }
}
```

```
export class Comp1 implements OnInit {
    constructor(private searchService: SearchService) {}

    ngOnInit() {

    }

    search(word: HTMLInputElement) {
        this.searchService
            .setSearchString(word.value);
    }
}
```

```
export class Comp2Component implements OnInit, OnDestroy{

    searchSubscription: Subscription;

    constructor(private searchService: SearchService){ }

    ngOnInit() {
        this.searchSubscription = this.searchService.searchSource.subscribe(
            (data:string)=>{
                console.log(data);
```

```
            }
        )
    )

    ngOnDestroy(){
        this.searchSubscription.unsubscribe();
    }

}
```

Menu

Setup

```
ng generate @angular/material:navigation Menu
```